ТИМУР МАШНИН

bhv®

Web-сервисы Java

ОСНОВЫ ТЕХНОЛОГИИ WEB-СЕРВИСОВ В СПЕЦИФИКАЦИЯХ ПЕРВОГО И ВТОРОГО УРОВНЯ

СТАНДАРТЫ ТЕХНОЛОГИИ WEB-СЕРВИСОВ ПЛАТФОРМЫ JAVA

JAVA-СТЕКИ WEB-СЕРВИСОВ: Metro, CXF И Axis2

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Материалы на www.bhv.ru

Тимур Машнин

Web-сервисы Java

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.06
ББК 32.973.26-018.2
М38

Машнин Т. С.

М38 Web-сервисы Java. — СПб.: БХВ-Петербург, 2012. — 560 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0778-3

Рассмотрены основы технологии Web-сервисов в спецификациях первого и второго уровня, реализация технологии Web-сервисов в виде стандартов платформы Java и в таких распространенных Java-стеках Web-сервисов, как Metro, CXF и Axis2. Материал книги сопровождается более 70 примерами с подробным анализом исходных кодов. На сайте издательства находятся примеры проектов из книги, а также дополнительные материалы.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 31.10.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 45,15.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0778-3

© Машнин Т. С., 2011
© Оформление, издательство "БХВ-Петербург", 2011

Оглавление

Введение **7**

Глава 1. Архитектура XML Web-сервисов **13**
Модель Message Oriented 14
Модель Service Oriented 15
Модель Resource Oriented 16
Модель Policy 17
Архитектура Service Oriented Architecture (SOA) 17
Основные технологии архитектуры Web-сервисов 19
XML 19
XML Namespaces 21
XML Infoset 22
XML Schema 27
SOAP 1.2 41
WSDL 2.0 50
Практическое применение Web-сервисов 59
UDDI 62
ebXML 64
DISCO 67
JAXR 68
Языки WS-BPEL и WS-CDL 71
WS-BPEL 2.0 71
WS-CDL 1.0 86

Глава 2. Расширения технологии Web-сервисов **97**
WS-Policy, WS-PolicyAttachment и WS-PolicyAssertions 98
WS-Addressing 103
WS-Security 108
WS-Trust 117
WS-SecureConversation 130
WS-SecurityPolicy 136
WS-Federation 160
WS-Transfer 171
WS-ResourceTransfer и WS-Fragment 174

WS-MetadataExchange ... 176
WS-Enumeration ... 179
WS-Eventing ... 184
WS-Management ... 188
WS-Discovery ... 193
WS-ReliableMessaging ... 197
WS-ReliableMessaging Policy ... 202
WS-MakeConnection ... 204
WS-Coordination ... 204
WS-AtomicTransaction ... 206
WS-BusinessActivity ... 208

Глава 3. Java Web-сервисы ... 210
JAXM и SAAJ ... 211
Пример Web-сервиса и клиента на основе JAXM и SAAJ ... 211
JAXP ... 222
Пример использования JAXP ... 223
JAXB ... 229
Инструменты xjc и schemagen ... 230
Binding Declaration ... 232
JAXB API ... 240
Пример использования JAXB ... 241
JAX-RPC ... 244
Инструменты wscompile и wsdeploy ... 249
JAX-RPC API ... 259
Пример использования JAX-RPC ... 259
JAX-WS ... 262
JAX-WS API ... 264
Модель программирования JAX-WS ... 264
Модель программирования на стороне сервера ... 264
Модель программирования на стороне клиента ... 266
Развертывание JAX-WS Web-сервисов и JAX-WS-клиентов ... 267
Пример создания JAX-WS Web-сервиса и JAX-WS-клиента ... 270
JAX-RS ... 294
JAX-RS API ... 295
Модель программирования и развертывания JAX-RS Web-сервисов ... 295
Формат JSON ... 297
WADL ... 299
Применение технологии JAX-RS ... 303

Глава 4. Проект Metro ... 312
Тестирование стека Metro ... 313
Оптимизация передачи двоичных данных (MTOM) ... 315
Адресация ... 319
Надежная доставка сообщений ... 321
Система безопасности ... 325
Создание клиента Web-сервиса ... 331
Опция *Проверка подлинности имени пользователя с помощью симметричного ключа* ... 333

Опция *Username Authentication with Password Derived Key* ... 341
Опция *Безопасность совместных сертификатов* ... 344
Опция *Симметричная привязка к маркеру Kerberos* ... 347
Опция *Безопасность транспорта (SSL)* ... 351
Опция *Проверка подлинности сообщения по SSL* ... 356
Опция *Проверка подлинности SAML по SSL* ... 361
Опция *Одобрение сертификата* ... 364
Опция *Подтверждение подлинности отправителя SAML сертификатом* ... 366
Опция *Держатель ключа SAML* ... 369
Опция *Выпущенный STS маркер* ... 372
Опция *Выпущенный STS маркер с сертификатом службы* ... 379
Опция *Выпущенный STS маркер одобрения* ... 380
Опция *Выпущенный STS маркер поддержки* ... 382
Поддержка протокола SOAP/TCP ... 383
Поддержка кодировки Fast Infoset ... 384
Поддержка WS-MakeConnection ... 386

Глава 5. Проект Apache CXF ... 388
Архитектура платформы CXF ... 389
Создание SOAP Web-сервисов с использованием CXF API ... 393
Связывание данных Aegis ... 400
Связывание данных XMLBeans ... 403
Опции *features* и обработчики Interceptors ... 404
Протоколы передачи сообщений ... 413
Поддержка протокола SOAP/HTTP ... 413
Поддержка протокола XML/HTTP ... 415
Поддержка протокола HTTPS ... 419
Apache Camel, JMS и Apache ActiveMQ ... 422
Проект Apache Camel ... 422
Проект Apache ActiveMQ ... 430
Локальный транспорт ... 439
Поддержка MTOM ... 440
Поддержка спецификаций WS-* ... 442
WS-Addressing ... 442
WS-ReliableMessaging ... 444
WS-Security ... 447
WS-SecurityPolicy ... 451
WS-Trust ... 453
WS-SecureConversation ... 454
JAX-RS ... 455
JavaScript ... 461

Глава 6. Проект Axis2 ... 464
Конфигурационный файл axis2.xml ... 467
Архив AAR и развертывание Web-сервиса ... 469
Модули Axis2 ... 473
Модель программирования Axis2 Web-сервисов ... 476
Axis2 XML-модель AXIOM ... 478
Client API ... 484

Поддержка архитектуры REST ... 493
Связывание данных ... 500
ADB (Axis2 Databinding) ... 503
XMLBeans ... 504
JiBX ... 504
JAXB ... 519
Поддержка MTOM ... 519
Поддержка протокола HTTPS ... 524
HttpClient и аутентификация ... 527
Транспортные протоколы проекта Axis2 ... 530
TCP ... 531
JMS ... 532
WS-ReliableMessaging ... 537
WS-Security ... 541

Приложение. Описание электронного архива ... 549

Список литературы ... 558

Предметный указатель ... 559

Введение

Технология Web-сервисов — это технология создания распределенных систем, составленных из взаимодействующих между собой программных продуктов, созданных и работающих на основе различных платформ.

Web-сервисы призваны согласовывать работу больших, состоящих из множества частей приложений, предоставляя для приложений бизнес-функции обмена данными.

Помимо функции обмена данными между различными приложениями и платформами, Web-сервисы могут выступать как повторно-используемые компоненты приложения, предоставляющие разнообразные сервисы — от прогноза погоды до перевода с одного языка на другой.

Web-сервисы представляют собой программные компоненты, имеющие идентификатор URI, и взаимодействие с которыми осуществляется по Интернету с помощью открытых протоколов.

Коммуникация с Web-сервисами может выполняться с помощью различных транспортных протоколов, таких как HTTP, HTTPS, FTP, SMTP, BEEP, при этом Web-сервисы можно подразделить на три вида: SOAP Web-сервисы, ориентированные на модель RPC — вызов удаленных процедур, XML Web-сервисы, ориентированные на сообщения, и RESTful Web-сервисы.

Первая группа Web-сервисов — это Web-сервисы, взаимодействие с которыми производится с использованием XML-сообщений по SOAP-протоколу (Simple Object Access Protocol), и имеющие интерфейсы, описанные в формате WSDL (Web Services Description Language). Такое описание интерфейса сервиса обеспечивает автоматическую генерацию кода на клиентской стороне, необходимого для связи с сервисом. Описание WSDL Web-сервиса может быть доступно клиенту с помощью реестра UDDI (Universal Description, Discovery, and Integration), в котором Web-сервис предварительно зарегистрирован. SOAP-протокол может использовать различные транспортные протоколы — HTTP, FTP SMTP и др., однако чаще всего SOAP используется поверх HTTP. SOAP-сообщения, участвующие в обмене между клиентом и SOAP RPC Web-сервисом, имеют строго определенную структуру для передачи имени вызываемой удаленной процедуры и ее параметров, а также результата ее вызова.

Вторая группа Web-сервисов — это XML Web-сервисы, ориентированные на сообщения. Эти XML Web-сервисы обеспечивают низкоуровневую обработку XML-сообщений, при этом Web-сервис обрабатывает полученные XML-данные целиком, как они есть, и полностью формирует ответное XML-сообщение. XML Web-сервисы могут передавать и получать сообщения как в формате SOAP, так и в чистом XML-формате.

Третья группа Web-сервисов — это RESTful Web-сервисы, представляющие удаленные ресурсы, доступные с помощью HTTP-запросов. RESTful Web-сервисы обеспечивают взаимодействие с удаленными ресурсами, передавая клиенту их представление. RESTful Web-сервисы идентифицируются URL-адресом и обрабатывают HTTP-методы GET, PUT, POST и DELETE в ответ на запрос клиента. Технология REST Web-сервисов также может использовать WSDL-описание и SOAP-протокол для передачи сообщений, но может обходиться и без них.

Альтернативой использования технологии Web-сервисов для создания распределенных систем является применение технологий CORBA (Common Object Request Broker Architecture), Java RMI (Remote Method Implementation) и DCOM (Distributed Component Object Model).

Технология Web-сервисов развивается под эгидой организации W3C.

> ***Примечание***
>
> World Wide Web Consortium (W3C) — международное сообщество, состоящее из постоянных членов (в 2009 году 338 организаций), штатных сотрудников и общественности, цель которого — разработка стандартов Web.

Стандарты сообщества W3C объединены в следующие группы:

- Web-дизайн и приложения — стандарты для создания и отображения Web-страниц, включая обеспечение их доступности и интернационализации. Данная группа описывает такие технологии, как HTML, CSS, SVG, Ajax и др., а также создание Web-приложений для мобильных устройств;
- Web-архитектура — описывает базисные технологии и принципы, включая URI и HTTP;
- семантическая Web-информация — блок посвящен технологиям Web-данных, позволяющим создавать хранилища Web-данных, словари, а также определять правила для управления данными;
- XML-стандарты — представляет такие стандарты, как XML, XQuery, XML Schema, XSLT, XSL-FO, Efficient XML Interchange (EXI) и др.;
- Web-сервисы — описывает технологии обмена сообщениями;
- Web-устройства — технологии Web-доступа везде, в любое время и с использованием любого устройства;
- браузеры и инструменты — технологии для разработки программного обеспечения Web.

W3C-группа Web-сервисы содержит следующие подгруппы спецификаций (**http://www.w3.org/standards/webofservices/**).

- Данные.
  - XML Schema — спецификации посвящены языку описания структуры XML-документа. XML-схема используется для проверки XML-документа на соответствие XML-схеме, после чего могут быть созданы объекты, соответствующие структуре XML-документа.
  - XML-binary Optimized Packaging — спецификации определяют способ включения бинарных данных в XML-документ. Стандарт включения бинарных данных оптимизирует передачу вместе с XML-документом двоичных данных большого объема, таких как изображения и различного рода мультимедийных данных.
  - XML — спецификации описывают язык Extensible Markup Language (XML).
  - RDF — спецификации описывают платформу Resource Description Framework (RDF) для представления данных в Web-сети.
  - GRDDL — спецификации описывают механизм Gleaning Resource Descriptions from Dialects of Languages извлечения RDF-контента из XML-документов.
- Протоколы.
  - HTTP — спецификации описывают механизм расширения HTTP-протокола, включая расширение HTTP-протокола для электронной торговли, альтернативный HTTP-протоколу — протокол HTTP-NG и его части — протокол SMUX, протокол Binary Wire Protocol и Web Interface, представление HTTP-протокола с использованием RDF, библиотеку API для получения событий сервера с помощью HTTP-протокола.
  - SOAP — спецификации описывают SOAP-протокол обмена XML-сообщениями.
  - Web Services Addressing — спецификации определяют механизм обращения к Web-сервисам и обмена с ними сообщениями.
  - Web Services Architecture — спецификации описывают архитектуру и механизм работы технологии Web-сервисов.
- Описание сервиса.
  - WSDL — спецификации определяют язык описания Web-сервисов Web Services Description Language.
  - Web Services Choreography — спецификации определяют язык Web Services Choreography Description Language (WS-CDL) описания последовательности и условий обмена сообщениями и взаимодействия Web-сервисов.
  - Web Services Policy — спецификации описывают механизм обеспечения требований и условий взаимодействия с Web-сервисами, позволяя Web-сервису ассоциироваться с набором правил — политикой, которой должен придерживаться потребитель Web-сервиса.

- Semantic Annotation for WSDL and XML Schema — спецификации определяют расширение WSDL для классификации, регистрации/обнаружения, сопоставления, композиции и вызова Web-сервисов.
- Service Modeling Language (SML) — спецификации описывают язык моделирования сложных систем взаимосвязанных сервисов и ресурсов.
- Web Services Resource Access — спецификации определяют основанные на SOAP протоколы для передачи больших данных, получения уведомлений о событиях Web-сервисов, управления основанными на Web-сервисах ресурсами.

❒ Безопасность.

- XML Encryption — спецификации описывают механизм шифрования данных и представления результатов шифрования в XML-документе.
- XML Signature — спецификации описывают механизм цифровой подписи XML-документов.
- XML Key Management Specification (XKMS) — спецификации описывают протокол распространения и регистрации публичных ключей цифровой подписи XML-документов.

❒ Интернационализация — спецификации описывают интернационализацию Web-сервисов, HTML- и XML-документов.

Технология Web-сервисов впервые официально была включена в качестве стандарта в спецификацию J2EE 1.4 в 2003 году. При этом основой Java-технологии Web-сервисов стала спецификация Java API for XML-Based RPC (JAX-RPC), которая вместе с технологией Java APIs for XML Messaging (или SOAP with Attachments API for Java (SAAJ)) обеспечивала поддержку функциональной совместимости Web-сервисов. С тех пор технология Web-сервисов платформы Java прошла большой эволюционный путь до спецификации Java API for XML-Based Web Services (JAX-WS), ставшей стандартом платформы Java EE 5.

Знание технологии Web-сервисов является необходимым условием профессиональной квалификации программиста. Это видно из анализа требований к Java-программисту в вакансиях на рынке труда (табл. B1).

Таблица B1. *Требования к знаниям и опыту Java-программиста на рынке труда*

Технология	Вакансии с требованием знания технологии в % соотношении от общего числа вакансий Java-программиста
JDBC	57
Servlets	61
JSF	30
Struts	19
XML/XSLT	53
EJB	38

Таблица В1 *(окончание)*

Технология	Вакансии с требованием знания технологии в % соотношении от общего числа вакансий Java-программиста
Hibernate	38
Spring	31
JSP	65
Web Services	35
JNDI	15
JMS	12
JPA	12

В данной книге предлагаются к рассмотрению основы самой технологии Web-сервисов, реализации технологии Web-сервисов в виде стандартов платформы Java и в таких распространенных Java-стеках Web-сервисов, как Metro, Apache CXF и Axis2.

Стек Metro представляет стек Web-сервисов, состоящий из реализаций технологий JAX-WS, JAXB и WSIT. Технология Metro является частью платформы Java EE и интегрирована с сервером GlassFish, позволяя создавать и развертывать безопасные и надежные Web-сервисы с поддержкой транзакций. При этом технология Metro гарантирует совместимость между Web-сервисами платформ Java EE и Microsoft .NET в приложениях, основанных на архитектуре Service Oriented Architecture (SOA).

Стеки Apache CXF и Axis2 представляют собой открытые платформы со средой выполнения для разработки и развертывания Web-сервисов, обеспечивающие поддержку спецификаций как самой технологии Web-сервисов, так и стандартов ее реализации платформой Java.

ГЛАВА 1

Архитектура XML Web-сервисов

Технология Web-сервисов обеспечивает взаимодействие между приложениями, работающими на различных платформах, с помощью программных компонентов — Web-сервисов.

Web-сервисы — это программные компоненты, которые имеют интерфейс, описанный в формате, пригодном для компьютерной обработки, как правило — это WSDL-формат, и взаимодействие с которыми осуществляется согласно их WSDL-описанию с помощью сообщений, передаваемых обычно по HTTP-протоколу.

Web-сервис является ресурсом, который характеризуется URI-адресом. При этом интерфейс Web-сервиса представляет его абстрактную функциональность, а конкретная реализация интерфейса Web-сервиса обеспечивает отправку и получение сообщений при взаимодействии клиента с Web-сервисом.

Различные реализации интерфейса Web-сервиса по-разному реализуют его общую функциональность, предоставляя многообразие услуг в рамках одного сервисного набора.

Конкретную реализацию интерфейса Web-сервиса обеспечивает *агент* — программный компонент, осуществляющий отправку и получение сообщений и принадлежащий *поставщику сервиса*. Клиент, осуществляющий запрос к Web-сервису с помощью клиентского приложения, также именуемого агентом, называется *потребителем сервиса*.

Как было сказано, взаимодействие с Web-сервисом осуществляется с помощью обмена сообщениями. Для того чтобы такой обмен был успешным, его механизм должен быть определен для потребителя сервиса его поставщиком. Такое определение механизма обмена сообщениями обеспечивается WSDL-описанием Web-сервиса.

WSDL-описание Web-сервиса определяет формат сообщений, участвующих в обмене, тип передаваемых данных, протокол передачи сообщений и адреса доставки сообщений Web-сервису.

Общепризнанным кроссплатформенным форматом обмена сообщений является XML-формат.

Архитектура XML Web-сервисов описывается набором моделей:

- Message Oriented Model — модель описывает сообщения Web-сервиса, включая структуру сообщений и механизм их доставки;
- Service Oriented Model — модель характеризует сервис, выполняемые действия при взаимодействии с сервисом, агентов поставщиков и потребителей сервиса, а также определяет метаданные, используемые для описания сервиса;
- Resource Oriented Model — модель описывает Web-сервисы как ресурсы, идентифицируемые URI-адресом, принадлежащие поставщикам сервиса и предоставляемые потребителям сервиса;
- Policy Model — модель описывает условия доступа к ресурсам.

Модель Message Oriented

Модель Message Oriented Model (MOM) описывает сообщения и их обработку, включая структуру сообщений, отношения между отправителем и получателем сообщений, а также механизм их передачи.

MOM определяет понятия адреса, политики доставки, сообщения, тела сообщения, корреляции сообщения, оболочки сообщения, модели обмена сообщениями Message Exchanage Pattern (MEP), заголовка сообщения, получателя сообщения, надежности сообщения, отправителя сообщения, последовательности сообщений, транспортировки сообщений.

Для того чтобы сообщение было доставлено получателю, необходим адрес доставки сообщения.

Формат адреса доставки сообщения зависит от механизма транспортировки сообщения. В случае HTTP-протокола передачи сообщений — это URL-адрес получателя.

Адрес доставки сообщения может содержаться в самом сообщении, как правило, в его оболочке.

Политика доставки — это условия доставки сообщения, которые могут заключаться в гарантиях доставки сообщения, его кодировке при передаче, формировании отчета о доставке и т. д.

Модель MOM определяет сообщение как единицу данных, передаваемую от одного агента другому. Структура сообщений определяется в WSDL-описании Web-сервиса.

Сообщение состоит из конверта (оболочки), одного или нескольких заголовков и тела сообщения. Конверт сообщения служит контейнером для его компонентов и может содержать информацию об адресе доставки сообщения. Заголовок сообщения содержит информацию о сообщении, относящуюся к системе безопасности, механизму транзакций, маршрутизации сообщения и т. д. Заголовок сообщения может обрабатываться независимо от самого сообщения, поэтому каждое сообщение может иметь различные заголовки, определяющие способы обработки тела со-

общения. Тело сообщения содержит его контент или адрес соответствующего ресурса данных.

Клиент может вызывать Web-сервис с помощью простых HTTP-запросов GET и POST. В случае GET-запроса заголовок сообщения — это HTTP-заголовок, а параметры URL — это тело сообщения. В случае POST-запроса заголовок сообщения — это HTTP-заголовок, а тело сообщения — это передаваемые данные.

В ответ на HTTP GET- и POST-запросы XML Web-сервис передает клиенту сообщения в XML-формате.

Если клиент взаимодействует с Web-сервисом по SOAP-протоколу, тогда сообщения, участвующие в обмене, имеют SOAP-структуру, состоящую из конверта, заголовков и тела.

При обмене сообщениями с Web-сервисом устанавливается соответствующий контекст, в котором сообщение запроса ассоциируется с сообщением ответа, что важно в случае асинхронного обмена.

Обмен сообщениями с Web-сервисом регламентируется шаблоном Message Exchange Pattern (MEP), который определяет поток сообщений (последовательность) между агентами, включая обработку ошибок и связи между входящими и выходящими сообщениями.

Шаблоны MEP могут описывать обмен сообщениями не только с одним Web-сервисом, но и последовательности, в которых Web-сервисы взаимодействуют друг с другом.

Модель MOM определяет надежность и гарантированность сообщения как степень уверенности, что сообщение будет доставлено в необходимом порядке. Механизм надежности и гарантированности уменьшает ошибки и обеспечивает информацию о статусе доставки, делая возможным, в случае возникновения ошибки доставки, повторную отправку сообщения и восстановления его порядка доставки.

Модель Service Oriented

Модель Service Oriented Model (SOM) описывает сервис и действия.

Модель SOM определяет сервис как ресурс, представляющий возможность выполнения задач, реализующих функциональность, которая определена поставщиком сервиса. Чтобы сервис мог использоваться, он должен быть реализован хотя бы одним агентом.

Web-сервис, так же как и Web-ресурс, идентифицируется URI-адресом, однако главное отличие Web-сервиса от Web-ресурса в том, что Web-сервис необязательно должен иметь представление — данные состояния ресурса в общедоступных форматах XML, HTML, CSS, JPEG, PNG, получаемые с помощью метода HTTP GET.

Web-сервис имеет свое описание, которое представляет собой набор документов, подлежащий компьютерной обработке и содержащий описание интерфейса Web-сервиса и его поведения. Описание Web-сервиса может быть реализовано с помощью языка Service Description Language (SDL) в виде набора XML-документов.

Описание Web-сервиса может использоваться для его конструирования, развертывания, определения местонахождения, установления связи между агентами потребителя и поставщика сервиса.

Web-сервис ассоциируется с действиями, решающими определенные задачи, которые представляют функциональность сервиса. Модель SOM определяет действие как операцию, которая может быть выполнена агентом в результате получения или отправки сообщения или другого изменения состояния. Обычно действия реализуются путем выполнения фрагмента программы, представляющей агента получателя или отправителя сообщения.

Агент — это программа, исполняющая действия и реализующая потребителя или поставщика Web-сервиса.

Последовательность и условия, при которых сложная система взаимодействующих между собой агентов обменивается сообщениями, описываются *хореографией Web-сервисов*. Хореография Web-сервисов — это модель, описывающая последовательность операций, состояний и условий, контролирующих взаимодействие сервисов. Результатом такого взаимодействия является выполнение определенной задачи, которую решает поставщик сервиса перед его потребителем.

Хореография Web-сервисов может описываться языком Choreography Description Language (CDL), который определяет композицию сервисов, их роли и связи, а также управление состоянием системы связанных сервисов.

Последовательность и условия, при которых один Web-сервис вызывает другие Web-сервисы для решения конкретной задачи, определяются оркестровкой Web-сервисов.

Модель Resource Oriented

Модель Resource Oriented Model (ROM) описывает ресурсы.

Модель ROM определяет ресурс как объект, имеющий имя-идентификатор, свое представление и собственника, обладающего правом назначать политику ресурса, которая определяет правила взаимодействия с ресурсом.

Web-сервисы являются разновидностью ресурсов, которые имеют описание, пригодное для компьютерной обработки и содержащее идентификатор ресурса в виде URI-адреса. С помощью описания потребитель узнает о полезности ресурса и устанавливает связь с ним. Описание облегчает поиск и доступ к ресурсу, включая в себя информацию о местонахождении ресурса, способах доступа к ресурсу и политиках ресурса.

Как было сказано ранее, Web-сервисы — это разновидность ресурсов, отличающаяся от обычных Web-ресурсов тем, что Web-сервисы необязательно должны иметь свое представление — данные состояния ресурса, получаемые с помощью запроса HTTP GET.

Поиск описаний Web-сервисов может осуществляться с использованием специальных сервисов, отвечающих за публикацию и поиск описаний согласно определен-

ным критериям. При статическом поиске описания Web-сервиса его потенциальный потребитель связывается с таким сервисом с помощью соответствующего программного обеспечения, например Web-браузера, и получает необходимое описание Web-сервиса. При динамическом поиске описания Web-сервиса агент потребителя сервиса напрямую обращается к специальному сервису для установки связи с агентом поставщика сервиса.

Передача описания Web-сервиса между поставщиком и потребителем сервиса может осуществляться и без специального сервиса — напрямую или с помощью других источников, таких как файловые системы, Web-сайты и др.

Модель Policy

Модель Policy Model описывает политики сервиса, представляющие собой ограничения на допустимые действия или состояния агентов или их собственников.

Политики могут определять ограничения относительно доступа к ресурсам/состояний ресурсов или возможных действий/состояний агентов поставщиков или потребителей сервисов.

Политики могут быть двух типов:

- политики разрешений;
- политики обязательств.

Политика разрешений позволяет агентам выполнять определенные действия, иметь доступ к определенным ресурсам, достигать определенного состояния.

За выполнением политики разрешений следит механизм защиты разрешений, гарантирующий соответствие использования сервиса — политики, установленной поставщиком сервиса.

Политика обязательств выдвигает требования для агентов исполнять определенные действия или достигать определенных состояний.

За выполнением политики обязательств следит механизм аудита, проверяющий выполнение обязательств, установленных поставщиком сервиса.

Политики могут иметь описание в формате, пригодном для компьютерной обработки. Описание политики определяет ее и используется для решения применения политики к конкретной ситуации.

Архитектура Service Oriented Architecture (SOA)

Web-сервисы являются программными компонентами распределенных приложений, имеющих сервис-ориентированную архитектуру SOA.

Распределенные системы могут создаваться с помощью технологии Web-сервисов, реализующей архитектуру SOA, или технологий COM/CORBA. Однако технологии COM/CORBA не обеспечивают в полной мере кроссплатформенность.

Реализация архитектуры SOA с помощью технологии Web-сервисов более приемлема для создания распределенных приложений, компоненты которых взаимодействуют через Интернет, разворачиваются и работают на различных платформах.

В архитектуре SOA распределенные системы состоят из взаимодействующих агентов поставщиков и потребителей сервисов, выполняющих определенные задачи. Агенты распределенной системы не работают в одной и той же среде выполнения и связываются друг с другом с помощью протоколов через сеть.

В распределенных системах скорость выполнения задач зависит от скорости удаленного доступа к агентам, а надежность — от ошибок коммуникации между агентами.

Архитектура SOA характеризуется следующими свойствами.

- *Логическое представление.* В логическом представлении SOA сервис — это абстрактное, логическое представление реальной программы, базы данных, бизнес-процесса и т. д., определенное в терминах функциональности.
- *Сообщения.* Сервис определяется в терминах сообщений, участвующих в обмене между агентами поставщиков и потребителей сервиса. Архитектура SOA не определяет внутреннюю структуру агентов, их реализацию, тем самым обеспечивая совместимость между любыми программными компонентами, удовлетворяющими требованиям обмена сообщениями в определении сервиса.
- *Описание.* Сервисы описываются метаданными, подлежащими компьютерной обработке. Описания сервисов обеспечивают публичность архитектуры SOA.
- *Степень детализации.* Сервисы используют небольшой набор операций с большим набором сообщений.
- *Передача по сети.* Взаимодействие с сервисами может осуществляться с помощью сетевого соединения.
- *Кроссплатформенность.* Сообщения, участвующие в обмене с сервисами, имеют стандартный кроссплатформенный XML-формат.

При реализации архитектуры SOA необходимо решать проблемы, связанные с ненадежностью транспортных протоколов, параллельным удаленным доступом, совместимостью программных компонентов, возможными ошибками одного или нескольких компонентов системы, обеспечением достаточной памяти для вызова объектов.

Архитектура SOA может быть реализована с помощью трех типов XML Web-сервисов:

- *RESTful Web-сервисы* обеспечивают обмен и управление представлениями ресурсов с помощью HTTP-методов GET, PUT, POST и DELETE;
- *RPC SOAP Web-сервисы* обеспечивают определенный набор операций;
- *XML Web-сервисы* обеспечивают получение, обработку и отправку сообщений.

Все три типа Web-сервисов имеют URI-идентификаторы и XML-формат сообщений. При этом RESTful и XML Web-сервисы могут использовать HTTP- или

SOAP/HTTP-протоколы для обмена сообщениями, а SOAP Web-сервисы могут использовать SOAP-протокол вместе с транспортными протоколами HTTP, SMTP, FTP и др.

Основные технологии архитектуры Web-сервисов

XML

XML-технология обеспечивает кроссплатформенный и расширяемый стандарт для формата данных.

При обсуждении XML-технологии мы будем опираться на спецификации XML 1.0, XML Information Set, Namespaces in XML 1.0 и XML Schema.

В архитектуре Web-сервисов XML-формат используется как для описания сервисов, так и для обмена сообщениями с Web-сервисами.

eXtensible Markup Language (XML) — это язык разметки документов, обеспечивающий текстовый формат хранения данных. Язык XML является подмножеством языка Standard Generalized Markup Language (SGML) и описывает определенный класс объектов, называемых *XML-документами*.

XML-формат представляет собой платформонезависимый способ структурирования информации путем представления документа в виде дерева элементов, каждый из которых может иметь набор атрибутов, представляющих пару "имя/значение", и содержать другие элементы и/или текст. При этом каждый элемент дерева может ссылаться на другие элементы с помощью своих атрибутов.

Приложения, работающие с XML-документами, для их обработки используют программный компонент — XML-процессор, обеспечивающий доступ к содержимому и структуре XML-документа. В технологии Java XML-процессоры представлены семейством Java for XML Processing (JAXP), включающим в себя процессоры DOM, SAX, StAX и TrAX.

XML-процессоры подразделяются на две категории — потоковые и объектные обработчики. При *потоковой обработке* XML-документа XML-процессор анализирует документ последовательно, в реальном времени, что экономит ресурсы памяти. SAX — это потоковый XML-процессор. При *объектной обработке* XML-документа XML-процессор создает в памяти объекты, представляющие состояние документа, что дает возможность в произвольном порядке анализировать его части. DOM — это процессор, поддерживающий объектную модель документа.

Процессор StAX является промежуточным между SAX и DOM, обеспечивая управление анализом XML-документа в приложении.

Процессор TrAX дает возможность трансформации XML-документа в другие форматы данных с использованием таблицы стилей XSLT. Существуют также XML-процессоры, трансформирующие XML-документы в формат HTML и XHTML с использованием каскадных таблиц стилей CSS.

Модель XML-документа описывает его в терминах логической и физической структуры. *Логическая структура* состоит из объявления, определения типа документа, элементов, комментариев, ссылок и инструкций по обработке документа.

Каждый XML-документ может начинаться с объявления, ограниченного тегами `<?xml` и `?>` и включающего в себя атрибуты `version` (версия XML-спецификации — на сегодняшний день только `1.0`), `encoding` (символьная кодировка документа, по умолчанию `Unicode`) и `standalone` (по умолчанию `no`, означает, что документ может иметь свое внешнее описание).

Далее в XML-документе может находиться описание структуры документа — определение типа документа DTD (Document Type Definition) или ссылка на внешний DTD-файл.

После объявления и определения типа документа его логическая структура представлена набором элементов, комментариев, ссылок и инструкций по обработке документа.

Элементы XML-документа, ограниченные тегами, которые называются *разметкой*, формируют его иерархическую структуру и могут иметь атрибуты, которые так же, как и элементы, способны хранить данные. XML-документ должен содержать как минимум один корневой элемент.

XML-теги определяют значение, типы данных, а не способ их отображения, как в формате HTML.

Атрибуты элементов представляют собой пары "имя/значение" и могут быть трех типов — строковый тип, лексемы и перечисления.

Дочерний элемент корневого элемента — это такое же его свойство, как и атрибут, за исключением того, что дочерний элемент может иметь свою сложную структуру.

Комментарии ограничиваются тегами `<!--` и `-->` и предназначены для документирования XML-документа.

Ссылки ограничиваются амперсандом (`&`) и точкой с запятой (`;`) и используются в XML-документах для подстановки при обработке документа вместо них символов или различного рода данных, описанных в определении DTD. Подставляемые данные могут быть обрабатываемыми XML-процессором, т. е. представлены в XML-формате, или необрабатываемыми — текстовые данные не формата XML, изображения и другие двоичные данные. Кроме того, для включения в XML-документ символьных данных, которые не следует обрабатывать XML-процессором, используется секция `CDATA`, ограниченная тегами `<![CDATA[` и `]]>`.

XML-ссылка — это ссылка на внешний объект, сущность, содержимое которого размещается в текущем месте, т. е. ссылка на сущность работает как подстановка и обеспечивает модульность XML-документа, в отличие от HTML-ссылки, являющейся инструкцией приложению для перехода на другой HTML-документ или фрагмент HTML-документа.

Инструкции по обработке ограничиваются тегами `<?` и `?>` и предназначены для передачи информации приложению, работающему с XML-документом.

Физическая структура XML-документа описывает его как набор сущностей, которые могут быть обрабатываемыми и необрабатываемыми, внешними и внутренними. XML-документ должен содержать как минимум одну сущность — корневую сущность документа. Сущности могут включаться в XML-документ с помощью ссылок.

XML-документы характеризуются условиями корректности (правильности) и действительности. *Правильный* XML-документ — это документ, имеющий один корневой элемент, элементы которого не перекрываются, удовлетворяющий другим требованиям спецификации XML. *Действительный* XML-документ — это документ, имеющий описание своей структуры и соответствующий этому описанию.

Помимо базового XML-синтаксиса, основные понятия XML-технологии, используемые в технологии Web-сервисов, — это XML Infoset, XML Schema и XML Namespaces.

XML Namespaces

Для предотвращения конфликта имен в XML-документах используются пространства имен, представляющие собой коллекции имен, в каждой из которых все имена уникальны, при этом каждая такая коллекция имеет свой уникальный идентификатор. Следовательно, каждое XML-имя может характеризоваться идентификатором *пространства имен* и *локальным именем* в пределах данного пространства имен.

Идентификатор XML-пространства имен является URI-адресом и используется в определении типа элемента XML-документа (имени начального и конечного тега элемента) и имени атрибута элемента.

Пространство имен объявляется с помощью зарезервированного имени `xmlns`, после которого может следовать префикс пространства имен, являющийся, по сути, сокращением идентификатора, и далее идет URI-идентификатор пространства имен.

Префикс пространства имен и локальное имя вместе составляют уточненное имя QName элементов и атрибутов XML-документа. Если пространство имен объявлено без префикса, тогда оно становится *пространством имен по умолчанию* для всех элементов XML-документа, не уточненных префиксом какого-либо пространства имен. Надо заметить, что это не относится к атрибутам XML-документа, которые для связывания с пространством имен всегда необходимо уточнять соответствующим префиксом. Поэтому корректный XML-документ не должен содержать разные по значению атрибуты с одинаковыми, не уточненными префиксом, локальными именами в пределах одного элемента.

Таким образом, механизм XML Namespaces позволяет присваивать уникальные имена элементам и атрибутам XML-документа, предоставляя каждому элементу и атрибуту свое уточненное QName-имя в пределах глобального XML-пространства имен `http://www.w3.org/XML/1998/namespace`.

XML Infoset

XML-формат делает возможным представление документа в виде иерархической структуры, определяющей его информационное пространство XML Infoset в виде множества информационных единиц, имеющих свойства.

Спецификация XML Information Set (XML Infoset) описывает абстрактную модель данных, представленных XML-документом, в виде набора информационных единиц. Каждая информационная единица является абстрактным описанием определенной части XML-документа и имеет набор связанных с ней свойств.

Необязательно каждый XML-документ должен иметь информационное пространство XML Infoset. Информационное пространство XML Infoset имеют лишь XML-документы, отвечающие правилам XML-синтаксиса согласно спецификации XML и соответствующие пространству имен согласно спецификации XML Namespaces.

Таким образом, правильно оформленный XML-документ — это сериализованная форма определенного информационного пространства.

Модель XML Infoset реализуется различными языками программирования в виде объектов, обеспечивающих доступ к частям XML-документа.

Спецификация XML Infoset определяет для XML-документа 11 типов информационных единиц — каждая с соответствующим набором свойств.

Информационное пространство XML-документа содержит как минимум корневую информационную единицу `document`. Все остальные информационные единицы информационного пространства XML-документа доступны с помощью свойств корневой информационной единицы `document` или свойств дочерних для `document` информационных единиц.

Информационная единица `document` имеет следующие свойства:

- `[children]` — список дочерних для `document` информационных единиц верхнего уровня, при этом список содержит как минимум одну информационную единицу — `document element`. Список также содержит информационные единицы `processing instruction` (инструкция по обработке информационной единицы) и `comment` (комментарий). Список может содержать информационный элемент `document type declaration` (DTD-описание или ссылка на DTD-описание структуры XML-документа);
- `[document element]` — идентификатор информационной единицы `document element`;
- `[notations]` — набор информационных единиц `notation` (нотации, которые дают возможность передавать приложениям различную информацию, например, способ обработки сущности, не подлежащей разбору);
- `[unparsed entities]` — набор информационных единиц `unparsed entity` (сущности, не подлежащие разбору);
- `[base URI]` — URI-идентификатор XML-документа;
- `[character encoding scheme]` — символьная кодировка XML-документа;

- `[standalone]` — если `YES`, тогда указывает автономность документа, т. е. возможность обработки XML-документа без привлечения других документов;
- `[version]` — версия XML-документа;
- `[all declarations processed]` — если `true`, тогда XML-процессор должен полностью обработать DTD-описание структуры XML-документа.

Каждый XML-документ состоит из элементов, ограниченных тегами и представляющих соответствующие информационные единицы `element` информационного пространства XML Infoset.

Информационные единицы `element` являются дочерними по отношению к информационной единице `document element` и имеют следующие свойства:

- `[namespace name]` — URI-идентификатор пространства имен, к которому принадлежит элемент;
- `[local name]` — локальная часть имени элемента, которая вместе со свойством `[namespace name]` образует имя элемента;
- `[prefix]` — префикс пространства имен элемента;
- `[children]` — список дочерних информационных единиц, содержащий информационные единицы `element`, а также информационные единицы `processing instruction` (инструкции для обработки), `unexpanded entity reference` (ссылка на необработанную внешнюю сущность, подлежащую разбору), `character` (единица текста — последовательности символов) и `comment` (комментарии) для данного элемента;
- `[attributes]` — набор информационных единиц `attribute`, представленный атрибутами элемента XML-документа;
- `[namespace attributes]` — набор информационных единиц `attribute`, представленный атрибутами элемента, объявляющими используемые элементом пространства имен;
- `[in-scope namespaces]` — набор информационных единиц `namespace` (пространства имен, используемые элементом);
- `[base URI]` — URI-идентификатор элемента;
- `[parent]` — информационная единица, являющаяся родительской.

Атрибуты элементов в виде пар "имя/значение" XML-документа представляют информационные единицы `attribute` информационного пространства XML Infoset. Атрибуты могут быть определены в XML-документе или установлены по умолчанию.

Атрибуты подразделяются на три типа — строковые, лексемы (знаковый тип) и перечисления.

Информационная единица `attribute` имеет следующие свойства:

- `[namespace name]` — URI-идентификатор пространства имен, к которому принадлежит атрибут;

- [local name] — локальная часть имени атрибута, которая вместе со свойством [namespace name] образует имя атрибута;
- [prefix] — префикс пространства имен атрибута;
- [normalized value] — нормализованное значение атрибута с преобразованием пробелов;
- [specified] — флаг, указывающий, что атрибут определен в начальном теге элемента или установлен по умолчанию в DTD-описании;
- [attribute type] — тип атрибута, объявленный в описании структуры XML-документа. Возможные значения — ID, IDREF, IDREFS (идентификаторы и ссылки на идентификаторы — знаковые типы), ENTITY, ENTITIES (имена сущностей, не подлежащих разбору — знаковые типы), NMTOKEN, NMTOKENS (лексемы имен — знаковые типы), NOTATION (информация, используемая приложением — строковый тип), CDATA (символьная строка — строковый тип) и ENUMERATION (список допустимых значений атрибута — перечисление);
- [references] — если тип атрибута равен IDREF, IDREFS, ENTITY, ENTITIES или NOTATION, тогда список информационных единиц element, unparsed entity или notation; если тип атрибута — ID, NMTOKEN, NMTOKENS, CDATA или ENUMERATION, тогда свойство не имеет значения;
- [owner element] — информационная единица element, к которой относится данная единица attribute.

Информационная единица processing instruction, представленная инструкцией обработки элемента в XML-документе, имеет следующие свойства:

- [target] — имя приложения, исполняющего инструкцию;
- [content] — данные для приложения в виде пар "имя/значение";
- [base URI] — URI-идентификатор инструкции;
- [notation] — информационная единица notation, определяющая имя приложения, которое исполняет инструкцию;
- [parent] — информационная единица, к которой относится данная инструкция.

Информационная единица unexpanded entity reference представлена ссылкой на необработанную XML-процессором внешнюю сущность, подлежащую разбору. Эта информационная единица имеет следующие свойства:

- [name] — имя сущности;
- [system identifier] — системный идентификатор сущности в виде URI-адреса сущности;
- [public identifier] — публичный идентификатор общедоступной сущности в виде URI-адреса сущности;
- [declaration base URI] — базовый URI-адрес, относительно которого разрешается системный идентификатор;
- [parent] — информационная единица element, к которой относится данная сущность.

Информационная единица `character` представлена символьными данными XML-документа и имеет следующие свойства:

- `[character code]` — символьная кодировка ISO 10646;
- `[element content whitespace]` — если `true`, тогда символ является пробелом;
- `[parent]` — информационная единица `element`, к которой относится данная информационная единица `character`.

Информационная единица `comment` представлена комментариями XML-документа и имеет следующие свойства:

- `[content]` — строка комментария;
- `[parent]` — информационная единица `document` или `element`, к которой относится данный комментарий.

Информационная единица `document type declaration` представлена ссылкой или самим DTD-описанием структуры XML-документа и имеет следующие свойства:

- `[system identifier]` — системный идентификатор внешнего DTD-описания в виде URI-адреса;
- `[public identifier]` — публичный идентификатор общедоступного внешнего DTD-описания в виде URI-адреса;
- `[children]` — список информационных единиц `processing instruction`, представленных инструкциями обработки DTD-описания;
- `[parent]` — информационная единица `document`, к которой относится данное DTD-описание.

Информационная единица `unparsed entity` представлена сущностью, не подлежащей обработке XML-процессором и объявленной в описании структуры XML-документа. Обычно это ресурс в двоичном формате или другом не XML-формате, например GIF, BMP и др. Информационная единица `unparsed entity` имеет свойства:

- `[name]` — имя сущности;
- `[system identifier]` — системный идентификатор сущности в виде URI-адреса сущности;
- `[public identifier]` — публичный идентификатор общедоступной сущности в виде URI-адреса сущности;
- `[declaration base URI]` — базовый URI-адрес, относительно которого разрешается системный идентификатор;
- `[notation name]` — имя нотации, определяющей формат сущности, которая не подлежит обработке;
- `[notation]` — информационная единица `notation` с именем, указанным в свойстве `[notation name]`.

Информационная единица `notation` представлена нотацией, объявленной в описании структуры XML-документа. Нотации используются для определения формата

сущностей (ресурсов), не подлежащих обработке XML-процессором, а также для указания приложений, получающих инструкции по обработке элементов XML-документа. Эта информационная единица имеет следующие свойства:

- `[name]` — имя нотации;
- `[system identifier]` — системный идентификатор нотации в виде URI-адреса;
- `[public identifier]` — публичный идентификатор общедоступной нотации в виде URI-адреса;
- `[declaration base URI]` — базовый URI-адрес, относительно которого разрешается системный идентификатор.

Информационная единица `namespace` представлена пространством имен элемента XML-документа и имеет следующие свойства:

- `[prefix]` — часть имени, следующая после `xmlns:`, если префикс отсутствует, тогда пространство имен является установленным по умолчанию;
- `[namespace name]` — пространство имен, следующее после префикса.

Как уже было сказано, XML-документы предназначены для хранения данных. Но приложение, использующее XML-документ, работает не непосредственно с документом, а с его информационным пространством XML Infoset, получаемым как результат разбора XML-документа XML-процессором. Однако информационное пространство XML Infoset может быть получено и другими способами, например, с использованием библиотек API DOM. В этом случае такое информационное пространство называется *синтетическим*.

XML-документы могут быть двух типов — документы, созданные с учетом правил, которым они должны подчиняться, и документы, не имеющие никаких правил.

При использовании XML-документа, не имеющего правил, вся ответственность за корректность написания документа лежит на его авторе.

Однако чаще всего XML-документы имеют описание своей структуры — XML-схему, представляющую собой набор логических и структурных правил. В этом случае проверку документа на соответствие правилам, определенным в описании структуры документа, производит XML-процессор.

XML-процессоры также могут быть двух типов — обработчики, которые обязаны осуществлять проверку XML-документа на соответствие описания его структуры, и обработчики, не осуществляющие такую проверку.

Описание структуры XML-документа, накладывающее ограничения на структуру и содержание документов данного типа, может быть создано с помощью различных языков, таких как Document Type Definition (DTD), XML Schema, RELAX NG и др.

Первым языком для создания XML-схем был язык Document Type Definition (DTD). Язык DTD является компактным и позволяет включать описание структуры XML-документа непосредственно в сам XML-документ. Но синтаксис языка DTD отличается от XML-синтаксиса, поэтому он прямо не поддерживается платформами и инструментами технологии XML. Кроме того, язык DTD не поддерживает пространства имен, а поддержка типов данных ограничена. Язык DTD не позволяет

определять элементы, содержащие целые и вещественные числа, даты и времена и др., и не может указывать сложные связи между элементами.

Язык XML Schema дает более богатые возможности по сравнению с DTD и устраняет его недостатки. Синтаксис XML Schema является XML-синтаксисом, кроме того, в языке XML Schema обеспечена поддержка пространств имен и всех необходимых типов данных. Также язык XML Schema позволяет создавать собственные типы данных и расширять существующие. Использование языка XML Schema обеспечивает трансформацию XML-документа в иерархию объектов определенных типов, доступных программным способом с помощью интерфейса (функциональность Post-Schema-Validation Infoset (PSVI)).

Язык RELAX NG, как и XML Schema, использует XML-синтаксис и по сравнению с XML Schema является более простым и легким в изучении. Язык RELAX NG сочетает в себе простоту DTD и богатые возможности XML Schema. Однако так как язык RELAX NG является в определенном смысле упрощением XML Schema, то он не поддерживает функциональность PSVI и имеет упрощенную типовую модель, что делает необходимым использование сторонних библиотек типов данных.

Здесь мы более подробно остановимся на языке XML Schema.

XML Schema

Язык XML Schema (XML Schema Definition (XSD)) позволяет очертить определенный круг XML-документов путем создания описания их структуры, накладывающим ограничения на данный тип документов и обеспечивающим их правильную интерпретацию. Описание XML-документов содержит определения элементов и атрибутов, которые могут появляться в данном типе документов, а также наследование элементов, включая порядок и количество потомков. Кроме того, язык XML Schema определяет тип содержимого элементов, типы данных элементов и атрибутов и, наконец, значения элементов и атрибутов по умолчанию и их фиксированные значения.

XML-документ ссылается на свое описание — XML-схему, созданную с помощью языка XML Schema, используя атрибуты `xsi:schemaLocation` и `xsi:noNamespaceSchemaLocation`, где *xsi* — префикс пространства имен, указанного в XML-документе с помощью атрибута `xmlns`:

```
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
```

Атрибут `schemaLocation` ссылается на XML-схему, имеющую определение пространства имен, а атрибут `noNamespaceSchemaLocation` — на XML-схему, не указывающую пространство имен для элементов и атрибутов. Атрибуты `xsi:schemaLocation` и `xsi:noNamespaceSchemaLocation` ссылаются на XML-схему своими значениями, представленными в виде URI-адреса схемы.

Корневым компонентом XML Schema является элемент `<schema>`, имеющий следующие атрибуты.

- Необязательный атрибут `attributeFormDefault`. Значение атрибута `"qualified"` указывает, что атрибуты XML-документа должны уточняться (квалифициро-

ваться) префиксом их пространства имен. По умолчанию значение атрибута — "unqualified".

- Необязательный атрибут blockDefault устанавливает по умолчанию значение атрибута block для всех элементов схемы. Возможные значения атрибута:
  - #all — в XML-документах запрещена замена базового элемента его производными, созданными любым способом;
  - extension — запрещена замена в XML-документах базового элемента производными элементами, полученными путем расширения базового типа элемента;
  - restriction — запрещена замена в XML-документах базового элемента производными элементами, полученными путем ограничения базового типа элемента;
  - substitution — в XML-документах запрещена любая замена базового элемента.
- Необязательный атрибут elementFormDefault. Значение атрибута "qualified" указывает, что элементы XML-документа должны уточняться (квалифицироваться) префиксом их пространства имен. По умолчанию значение атрибута — "unqualified".
- Необязательный атрибут finalDefault устанавливает по умолчанию значение атрибута final для всех типов данных схемы. Возможные значения атрибута:
  - #all — в XML-схеме запрещено создание производных типов любым способом;
  - extension — в XML-схеме запрещено создание производных типов путем расширения базового типа;
  - restriction — в XML-схеме запрещено создание производных типов путем ограничения базового типа;
  - list — в XML-схеме для простых типов запрещено создание производных типов путем создания списков;
  - union — в XML-схеме для простых типов запрещено создание производных типов путем объединения.
- Необязательный атрибут id — идентификатор схемы.
- Необязательный атрибут targetNamespace указывает пространство имен для элементов XML-документа, определяемых данной схемой.
- Необязательный атрибут version — версия схемы.
- Необязательный атрибут xml:lang определяет по умолчанию язык для всех комментариев схемы.

Элемент <schema> имеет также атрибут xmlns:xsd="http://www.w3.org/2001/XMLSchema", указывающий пространство имен для элементов и типов данных самой

схемы. При наличии префикса для данного пространства имен все элементы и типы данных схемы, включая элемент `<schema>`, должны уточняться этим префиксом (`xsd:`*`schema`*). Если префикс отсутствует, тогда атрибут `xmlns` указывает для схемы пространство имен по умолчанию. Данное пространство имен используется, чтобы очертить круг элементов и простых типов, относящихся к словарю языка XML Schema, а не к словарю автора схемы.

Элемент `<schema>` (рис. 1.1) может содержать вложенные элементы `<include>`, `<import>`, `<redefine>`, `<annotation>`, `<type>` (`<simpleType>`, `<complexType>`), `<group>`, `<attributeGroup>`, `<element>`, `<attribute>`, `<notation>`, `<identityConstraint>` (`<key>`, `<unique>`, `<keyref>`).

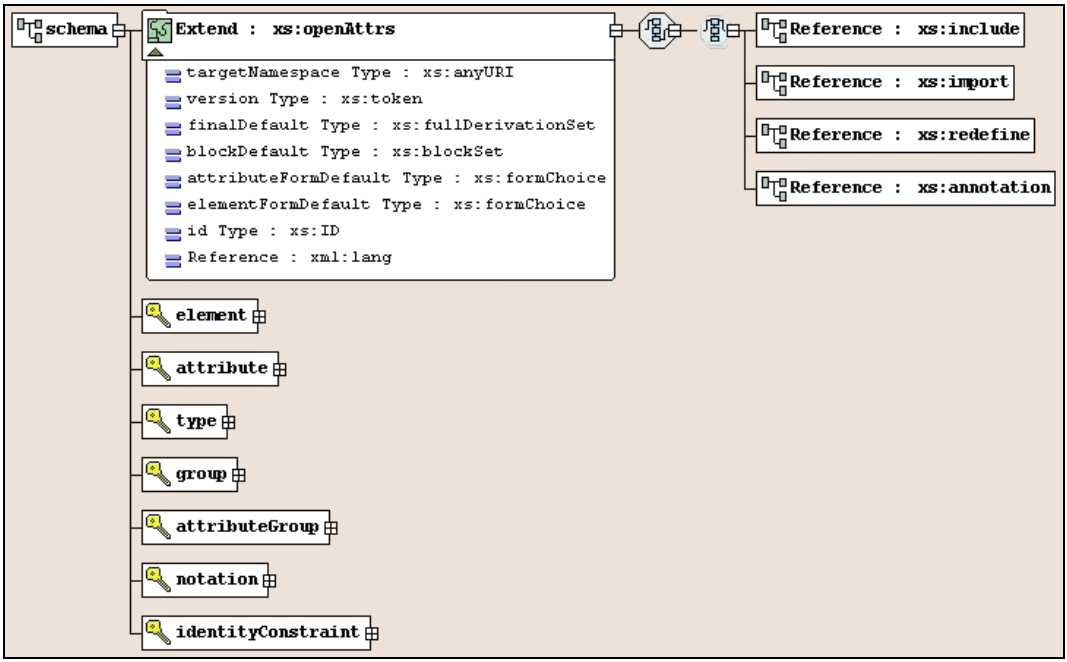


Рис. 1.1. Общая схема элемента `<schema>`

Язык XML Schema позволяет создавать описание структуры XML-документов, состоящее из нескольких схем, с помощью элементов `<include>` и `<import>`.

Элемент `<include>` добавляет все компоненты указанной схемы в основную схему, а элемент `<import>` дает возможность использовать компоненты указанной схемы в основной схеме.

Разница между элементами `<include>` и `<import>` состоит в том, что при использовании элемента `<include>` целевое пространство имен включаемой схемы (атрибут `targetNamespace`) должно быть таким же, что и целевое пространство имен основной схемы, а при использовании элемента `<import>` можно ссылаться на компоненты схемы с произвольным целевым пространством имен.

Элемент `<include>` имеет следующие атрибуты:

- необязательный атрибут `id` — идентификатор элемента;
- обязательный атрибут `schemaLocation` — URI-адрес включаемой схемы.

Элемент `<import>` имеет следующие атрибуты:

- необязательный атрибут `id` — идентификатор элемента;
- необязательный атрибут `namespace` — целевое пространство имен компонентов схемы, на которые можно ссылаться в основной схеме;
- необязательный атрибут `schemaLocation` — URI-адрес схемы, на компоненты которой можно ссылаться в основной схеме.

С помощью элемента `<redefine>` можно переопределять компоненты внешней схемы, которая имеет такое же целевое пространство имен, что и основная схема.

Элемент `<redefine>` имеет атрибуты: необязательный `id` (идентификатор элемента) и обязательный `schemaLocation` (URI-адрес схемы, компоненты которой переопределяются).

Во все компоненты XML-схемы можно включать документацию с помощью элемента `<annotation>`, содержащего в свою очередь элементы `<appinfo>` и `<documentation>`.

Элемент `<appinfo>` XML-схемы позволяет давать информацию приложениям, использующим схему, а элемент `<documentation>` содержит текстовые комментарии. Информация элемента `<appinfo>` используется приложением в качестве инструкции по обработке. Элемент `<appinfo>` имеет необязательный атрибут `source`, указывающий URI-адрес приложения.

Элемент `<documentation>` имеет необязательные атрибуты `source` (URI-адрес приложения, использующего комментарии) и `xml:lang` (язык комментариев).

XML-схема определяет элементы и атрибуты для XML-документов, используя элементы `<element>` и `<attribute>`.

Элементы `<element>` и `<attribute>` называются глобальными, если они являются дочерними для элемента `<schema>`.

Элемент `<element>` (рис. 1.2) имеет следующие атрибуты.

- Необязательный атрибут `abstract`. Если `true`, тогда этот элемент является абстрактным и сам не может использоваться в XML-документе, а должен замещаться элементами, имеющими в качестве значения атрибута `substitutionGroup` — имя данного абстрактного элемента. По умолчанию значение атрибута `abstract="false"`.
- Необязательный атрибут `block` ограничивает использование вместо данного элемента элементов с определенным типом наследования. Возможные значения:
  - `#all` — в XML-документах запрещена замена данного элемента его производными, созданными любым способом;

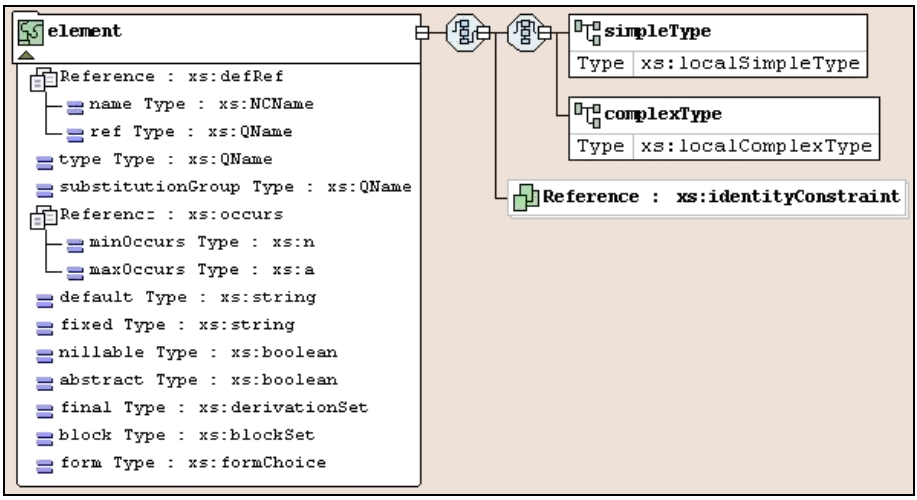


Рис. 1.2. Общая схема элемента `<element>`

- `extension` — запрещена замена в XML-документах данного элемента его производными элементами, полученными путем расширения типа данного элемента;
- `restriction` — запрещена замена в XML-документах данного элемента его производными элементами, полученными путем ограничения типа данного элемента;
- `substitution` — в XML-документах запрещена любая замена данного элемента.

❒ Необязательный атрибут `default` устанавливает для элементов с простым типом данных значение по умолчанию.

❒ Необязательный атрибут `final` ограничивает тип наследования данного элемента. Возможные значения:

- `#all` — для данного элемента в XML-схеме запрещено создание производных любым способом;
- `extension` — для данного элемента в XML-схеме запрещено создание производных путем расширения типа данного элемента;
- `restriction` — для данного элемента в XML-схеме запрещено создание производных путем ограничения типа данного элемента.

❒ Необязательный атрибут `fixed` для элемента с простым типом данных задает неизменяемое значение. Атрибут является взаимоисключающим с атрибутом `default`.

❒ Необязательный атрибут `form` задает форму элемента. Возможные значения:

- `qualified` — элемент необходимо уточнять префиксом пространства имен;
- `unqualified` — элемент не обязательно уточнять префиксом пространства имен.

- Необязательный атрибут `id` — идентификатор элемента.
- Необязательный атрибут `maxOccurs` задает максимальное число появлений данного элемента в содержащем его элементе XML-документа. Возможные значения — целое число, большее или равное 0, или `unbounded` (без ограничений). Данный атрибут запрещен для глобального элемента.
- Необязательный атрибут `minOccurs` задает минимальное число появлений данного элемента в содержащем его элементе XML-документа. Возможные значения — целое число до 1 или 0 (элемент необязателен). Данный атрибут запрещен для глобального элемента.
- Необязательный атрибут `name` задает имя элемента в XML-документе. Атрибут является обязательным для глобального элемента.
- Необязательный атрибут `nillable`. Если значение атрибута `true`, тогда в XML-документе для элемента можно задать его содержимое пустым с помощью атрибута `xsi:nil="true"`. По умолчанию значение атрибута — `false`.
- Необязательный атрибут `ref` — ссылка на элемент этой или внешней схемы. Значение атрибута должно быть QName-имя элемента (префикс пространства имен и локальное имя элемента). Ссылаться можно только на глобальные элементы, т. е. элементы, являющиеся дочерними для корневого элемента `schema`. Данный атрибут запрещен для глобального элемента.
- Необязательный атрибут `substitutionGroup` задает имя элемента, который можно заменить данным элементом, при этом элемент должен иметь тот же тип данных или тип данных, унаследованный от типа данных указанного элемента. Значение атрибута должно быть QName-имя элемента.
- Необязательный атрибут `type` указывает QName-имя элементов `simpleType` или `complexType`, задающих тип данных элемента. Атрибут является взаимоисключающим с атрибутом `ref`. С помощью атрибута `type` можно прямо указать существующий тип данных элемента, например, `type="xs:string"`.

Элемент `<element>` как контейнер может содержать элементы `<annotation>`, `<simpleType>` или `<complexType>`, `<unique>` либо `key` или `<keyref>`.

Элемент `<key>` указывает, что значение элемента-поля, определенного с помощью вложенного элемента `<field>`, должно быть ключом в пределах его элемента-контейнера, указанного с помощью вложенного элемента `<selector>`. Элемент `<key>` имеет атрибуты: необязательный `id` (идентификатор ключа) и обязательный `name` (имя ключа). Ключ должен быть ненулевым и постоянно доступным.

Элемент `<unique>` указывает, что значение элемента-поля, определенного с помощью вложенного элемента `<field>`, должно быть уникальным в пределах его элемента-контейнера, указанного с помощью вложенного элемента `<selector>`. Элемент `<unique>` имеет атрибуты: необязательный `id` (идентификатор) и обязательный `name` (имя). Значение элемента-поля должно быть уникальным или пустым.

Элемент `<keyref>` является ссылкой на элемент `<unique>` или `<key>`. Элемент `<keyref>` имеет атрибуты: необязательный `id` (идентификатор элемента `<keyref>`),

обязательный `name` (имя элемента `<keyref>`) и обязательный `refer` (QName-имя элемента `<unique>` или `<key>`, определенного в этой или внешней схеме).

Элементы `<unique>`, `<key>` и `<keyref>` в свою очередь являются контейнерами для элементов `<annotation>`, `<selector>` и `<field>`.

Элемент `<selector>` содержит XPath-выражение, определяющее множество элементов-контейнеров, которые имеют значения элементов-полей. Элемент `<selector>` имеет атрибуты: необязательный `id` (идентификатор) и обязательный `xpath` (XPath-выражение). Элемент `selector` должен быть одним для своего элемента-контейнера.

Элемент `<field>` содержит XPath-выражение, определяющее элемент-поле, и имеет атрибуты: необязательный `id` (идентификатор) и обязательный `xpath` (XPath-выражение).

> ***Примечание***
>
> XPath — это язык запросов к элементам XML-документа. Подробно язык XPath рассмотрен в приложении "Язык XPath" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Элемент `<simpleType>` объявляет простой тип содержимого элемента XML-документа и имеет следующие атрибуты:

- необязательный атрибут `final` накладывает ограничения на наследование данного простого типа с помощью возможных значений:
  - `#all` — запрещает любое наследование;
  - `list` — запрещает наследование списком;
  - `union` — запрещает наследование объединением;
  - `restriction` — запрещает наследование ограничением;
- необязательный атрибут `id` — идентификатор простого типа;
- атрибут `name` — имя типа, присутствует только в элементах `<simpleType>`, дочерних для элемента `schema`.

Элемент `<simpleType>` является контейнером для элемента `annotation` и одного из элементов `<restriction>`, `<list>` или `<union>`.

Простые типы, объявляемые элементом `<simpleType>`, представляют собой производные от встроенных типов данных XML Schema или производные от уже объявленных простых типов.

Создание простого типа, как производного, осуществляется с помощью вложенных элементов `<restriction>`, `<list>` или `<union>`.

Элемент `<restriction>` определяет простой тип, как производный от существующего, путем ограничения возможных значений, используя следующие атрибуты и вложенные элементы:

- обязательный атрибут `base` указывает базовый тип, из которого создается данный производный простой тип. Базовый тип может быть встроенным типом XML Schema или уже объявленным простым типом;

- необязательный атрибут `id` — идентификатор в пределах схемы;
- вложенный элемент `<annotation>` — документация;
- вложенный элемент `<simpleType>` — дочерний простой тип;
- вложенный элемент `<minExclusive>` указывает исключающую нижнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` (если `true`, тогда значение `value` является неизменяемым для производных, по умолчанию — `false`) и `id` (идентификатор);
- вложенный элемент `<minInclusive>` указывает включающую нижнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<maxExclusive>` указывает исключающую верхнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<maxInclusive>` указывает включающую верхнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<totalDigits>` — максимальное количество десятичных знаков перед запятой для типов, унаследованных от `decimal` с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<fractionDigits>` — максимальное количество десятичных знаков после запятой для типов, унаследованных от `decimal` с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<length>` — количество единиц длины, для строк — это количество символов, для двоичных данных `hexBinary` и `base64Binary` — это 8 бит, для списков — это количество элементов списка. Элемент имеет атрибуты `value`, `fixed` и `id`;
- вложенный элемент `<minLength>` — минимальное количество единиц длины, имеет атрибуты `value`, `fixed` и `id`;
- вложенный элемент `<maxLength>` — максимальное количество единиц длины, имеет атрибуты `value`, `fixed` и `id`;
- вложенный элемент `<enumeration>` ограничивает область возможных значений фиксированным набором значений с помощью атрибута `value`;
- вложенный элемент `<whiteSpace>` указывает для типов, производных от `string`, правила нормализации с помощью трех возможных значений атрибута `value` — `preserve` (нормализация не производится), `replace` (символы `#x9`, `#xA` и `#xD` заменяются символом `#x20`), `collapse` (пробелы в начале и в конце удаляются, дублирующие пробелы объединяются в один). Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<pattern>` ограничивает область возможных значений шаблоном, основанным на регулярном выражении с помощью атрибута `value`.

Элемент `<list>` определяет простой тип, как производный от существующего, путем создания списка, используя следующие атрибуты и вложенные элементы:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- атрибут `itemType` — имя встроенного типа данных или уже объявленного простого типа. Атрибут обязательный, если нет дочернего элемента `<simpleType>`, с которым он является взаимоисключающим;
- вложенный элемент `<simpleType>` — производный тип;
- вложенный элемент `<annotation>` — документация.

Элемент `<union>` определяет простой тип, как производный от существующих типов, путем объединения, используя следующие атрибуты и вложенные элементы:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- необязательный атрибут `memberTypes` — список имен встроенных типов данных или уже объявленных простых типов;
- вложенный элемент `<simpleType>` определяет участника объединения;
- вложенный элемент `<annotation>` — документация.

XML Schema поддерживает следующие встроенные типы данных.

- `string` — символьная строка. Встроенный производный тип данных — `normalizedString` (символьная строка с нормализацией пробелов). Встроенный производный тип данных от `normalizedString` — `token` (лексема), который также имеет встроенные производные типы данных — `language` (идентификатор языка), `NMTOKEN` (лексемы, соответствующие XML-типу атрибута `NMTOKEN`, имеет производный тип — `NMTOKENS`), `Name` (XML-имена, имеет производный тип — `NCName`). Встроенный тип `NCName`, часть XML-имени без двоеточия, имеет производные типы `ID` (идентификатор), `IDREF` (ссылка на идентификатор, имеет производный тип `IDREFS`) и `ENTITY` (XML-сущность, имеет производный тип `ENTITIES`).
- `boolean` — логическое значение `true`, `false`.
- `decimal` — десятичные числа. Встроенный производный тип данных — `integer` (целые числа). Производный тип данных `integer` имеет, в свою очередь, производные типы `nonPositiveInteger` (верхняя граница значений — 0, имеет производный тип `negativeInteger` — верхняя граница –1), `nonNegativeInteger` (нижняя граница значений — 0, имеет производные типы `unsignedLong` с верхней границей 18 446 744 073 709 551 615 и `positiveInteger` с нижней границей 1), `long` (значения от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, имеет производный тип `int` со значениями от –2 147 483 648 до 2 147 483 647). Тип `unsignedLong` имеет производный тип `unsignedInt` с верхней границей значений 4 294 967 295. Тип `unsignedInt` имеет производный тип `unsignedShort` с верхней границей 65 535. Тип `unsignedShort` имеет производный тип `unsignedByte` с верхней границей 255. Тип `int` имеет производный тип `short` со значениями от –32 768 до 32 767. Тип `short` имеет производный тип `byte` со значениями от –128 до 127.
- `float` — числа с плавающей запятой.
- `double` — числа с плавающей запятой двойной точности.

- duration — промежуток времени (PnYn MnDTnH nMnS).
- dateTime — последовательность даты и времени.
- time — время (hh:mm:ss.sss).
- date — дата.
- gYearMonth — год и месяц.
- gYear — год.
- gMonthDay — месяц и день.
- gDay — день.
- gMonth — месяц.
- hexBinary — представление двоичных данных в виде последовательностей символов.
- base64Binary — Base64-кодировка двоичных данных.
- anyURI — URI-адрес.
- QName — префикс, локальная часть имени.
- NOTATION — значение одноименного XML-атрибута.

Элемент <complexType> определяет сложный тип содержимого элемента XML-документа и имеет следующие атрибуты:

- необязательный атрибут abstract. Если true, тогда в XML-документе используется сложный тип, производный от данного типа. По умолчанию значение атрибута — false;
- необязательный атрибут block ограничивает использование в XML-документе сложного типа с указанным типом наследования вместо данного сложного типа с помощью следующих возможных значений: #all (запрет использования вместо этого сложного типа всех производных сложных типов), extension (запрет использования вместо этого сложного типа всех производных сложных типов, созданных расширением), restriction (запрет использования вместо этого сложного типа всех производных сложных типов, созданных ограничением);
- необязательный атрибут final ограничивает тип наследования с помощью следующих возможных значений: #all (запрет наследования), extension (запрет наследования расширением), restriction (запрет наследования ограничением);
- необязательный атрибут id — идентификатор элемента;
- необязательный атрибут mixed. Если true, тогда в XML-документе возможно наличие символьных данных между дочерними элементами этого типа. По умолчанию значение атрибута — false. Если есть дочерний элемент <simpleContent>, тогда использовать атрибут нельзя;
- атрибут name — имя типа. Атрибут обязательный, если элемент является дочерним для элемента <schema>, в других случаях использовать атрибут нельзя.

Элементы с простым типом данных могут содержать только символьные данные и не могут включать атрибуты и другие элементы.

Сложный тип определяет содержимое элементов XML-документа, которые могут включать в себя атрибуты и другие элементы.

Элемент `<complexType>` является контейнером для элементов `<annotation>`, `<simpleContent>`, `<complexContent>`, `<group>`, `<all>`, `<choice>`, `<sequence>`, `<anyAttribute>`, `<attribute>` и `<attributeGroup>`.

Элемент `<simpleContent>` определяет содержимое элемента, как символьные данные и атрибуты, имеет необязательный атрибут `id` (идентификатор элемента в пределах схемы) и должен содержать один из элементов `<restriction>` или `<extension>`.

Элемент `<restriction>`, дочерний для элемента `<simpleContent>`, накладывает ограничения на базовый тип данных, определяя тип содержимого элемента XML-документа, и описывает его атрибуты с помощью следующих атрибутов и вложенных элементов:

- обязательный атрибут `base` указывает базовый тип, из которого создается данный производный сложный тип. Базовый тип может быть встроенным типом XML Schema или уже объявленным простым типом или сложным типом;
- необязательный атрибут `id` — идентификатор в пределах схемы;
- вложенный элемент `<annotation>` — документация;
- вложенный элемент `<simpleType>` — дочерний простой тип;
- вложенный элемент `<minExclusive>` указывает исключающую нижнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` (если `true`, тогда значение `value` является неизменяемым для производных, по умолчанию — `false`) и `id` (идентификатор);
- вложенный элемент `<minInclusive>` указывает включающую нижнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<maxExclusive>` указывает исключающую верхнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<maxInclusive>` указывает включающую верхнюю границу области значений с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<totalDigits>` — максимальное количество десятичных знаков перед запятой для типов, унаследованных от `decimal` с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<fractionDigits>` — максимальное количество десятичных знаков после запятой для типов, унаследованных от `decimal` с помощью атрибута `value`. Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<length>` — количество единиц длины, для строк — это количество символов, для двоичных данных `hexBinary` и `base64Binary` — это 8 бит, для списков — это количество элементов списка. Элемент имеет атрибуты `value`, `fixed` и `id`;
- вложенный элемент `<minLength>` — минимальное количество единиц длины. Элемент имеет атрибуты `value`, `fixed` и `id`;

- вложенный элемент `<maxLength>` — максимальное количество единиц длины и имеет атрибуты `value`, `fixed` и `id`;
- вложенный элемент `<enumeration>` ограничивает область возможных значений фиксированным набором значений с помощью атрибута `value`;
- вложенный элемент `<whiteSpace>` указывает для типов, производных от `string`, правила нормализации с помощью трех возможных значений атрибута `value` — `preserve` (нормализация не производится), `replace` (символы `#x9`, `#xA` и `#xD` заменяются символом `#x20`), `collapse` (пробелы в начале и в конце удаляются, дублирующие пробелы объединяются в один). Имеет также атрибуты `fixed` и `id`;
- вложенный элемент `<pattern>` ограничивает область возможных значений шаблоном, основанным на регулярном выражении с помощью атрибута `value`;
- вложенный элемент `<attribute>` объявляет атрибут;
- вложенный элемент `<attributeGroup>` группирует атрибуты. Элемент `<attributeGroup>` имеет следующие атрибуты и вложенные элементы:
  - необязательный атрибут `id` — идентификатор;
  - необязательный атрибут `name` — имя группы атрибутов, может использоваться, если родительский элемент — `<schema>`;
  - необязательный атрибут `ref` — ссылка на группу атрибутов, взаимоисключающий атрибут с атрибутом `name`, может использоваться, если родительский элемент не является элементом `<schema>`;
  - вложенные элементы `<annotation>`, `<attribute>`, `<attributeGroup>`, `<anyAttribute>`;
- вложенный элемент `<anyAttribute>` разрешает любому атрибуту из указанного пространства имен появляться в содержащем их элементе. Элемент `<anyAttribute>` имеет следующие атрибуты:
  - необязательный атрибут `id` — идентификатор;
  - необязательный атрибут `namespace` принимает одно из значений: `##any` (по умолчанию могут включаться атрибуты из любого пространства имен), `##other` (могут включаться атрибуты из любого пространства имен, которое не является целевым пространством имен родительского элемента), URI-идентификаторы пространств имен, `##targetNamespace` (могут включаться атрибуты из целевого пространства имен родительского элемента), `##local` (могут включаться локальные атрибуты, не принадлежащие пространству имен);
  - необязательный атрибут `processContents` указывает способ проверки XML-документов, принимает одно из значений `strict` (по умолчанию, обязательная проверка атрибутов на соответствие указанному пространству имен), `lax` (необязательная проверка) и `skip` (проверка не производится).

Элемент `<extension>`, дочерний для элемента `<simpleContent>`, расширяет базовый тип данных, определяя тип содержимого элемента XML-документа, и описывает его атрибуты с помощью следующих атрибутов и вложенных элементов:

- обязательный атрибут `base` — базовый тип, из которого создается данный производный сложный тип. Базовый тип может быть встроенным типом XML Schema или уже объявленным простым типом или сложным типом;
- необязательный атрибут `id` — идентификатор в пределах схемы;
- вложенные элементы `<annotation>`, `<attribute>`, `<attributeGroup>`, `<anyAttribute>`.

Элемент `<complexContent>` допускает наличие дочерних элементов в содержимом элемента XML-документа. Элемент `<complexContent>` имеет необязательные атрибуты `id` (идентификатор) и `mixed` (если `true`, тогда между дочерними элементами могут находиться символьные данные, по умолчанию `false`), а также вложенные элементы `<annotation>`, `<restriction>` или `<extension>`.

Элементы `<restriction>` и `<extension>`, дочерние для `<complexContent>`, имеют следующие атрибуты и вложенные элементы:

- обязательный атрибут `base` — базовый тип, из которого создается данный производный сложный тип. Базовый тип должен быть уже объявленным сложным типом;
- необязательный атрибут `id` — идентификатор в пределах схемы;
- вложенные элементы — `<annotation>`, `<group>`, `<all>`, `<choice>`, `<sequence>`, `<attribute>`, `<attributeGroup>`, `<anyAttribute>`.

Элемент `<group>` позволяет создать группу элементов с помощью следующих атрибутов и вложенных элементов:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- необязательный атрибут `name` — имя группы элементов, используется, если только родительский элемент — `<schema>`;
- необязательный атрибут `maxOccurs` — максимальное число появлений группы элементов в родительском элементе XML-документа. Значение атрибута `unbounded` снимет ограничение на максимальное число;
- необязательный атрибут `minOccurs` — минимальное число появлений группы элементов в родительском элементе XML-документа. Значение атрибута `0` указывает, что появление группы необязательно;
- необязательный атрибут `ref` — ссылка на имя уже определенной группы элементов. Атрибут является взаимоисключающим с атрибутом `name`;
- вложенные элементы `<annotation>`, `<all>`, `<choice>`, `<sequence>`.

Элемент `<all>` накладывает ограничения на появление элементов в родительском элементе XML-документа с помощью следующих атрибутов и вложенных элементов:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- необязательный атрибут `maxOccurs` — максимальное число вхождений элемента. Значение атрибута должно быть равно `1`;

- необязательный атрибут `minOccurs` — минимальное число вхождений данного элемента. Значение атрибута может быть `1` (по умолчанию) или `0` (элемент необязателен);
- вложенные элементы `<annotation>`, `<element>`.

Элемент `<choice>` позволяет присутствовать в родительском элементе XML-документа одному-единственному элементу из указанных с помощью следующих атрибутов и вложенных элементов:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- необязательный атрибут `maxOccurs` — максимальное число появлений данного выбора элемента в родительском элементе XML-документа. Значение атрибута `unbounded` снимет ограничение на максимальное число. По умолчанию — `1`;
- необязательный атрибут `minOccurs` — минимальное число появлений данного выбора элемента в родительском элементе XML-документа. Значение атрибута `0` указывает, что появление элемента необязательно. По умолчанию — `1`;
- вложенные элементы `<annotation>`, `<element>`, `<group>`, `<choice>`, `<sequence>`, `<any>`.

Элемент `<sequence>` указывает наличие упорядоченной последовательности элементов в родительском элементе XML-документа с помощью следующих атрибутов и вложенных элементов:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- необязательный атрибут `maxOccurs` — максимальное число появлений данной последовательности элементов в родительском элементе XML-документа. Значение атрибута `unbounded` снимет ограничение на максимальное число. По умолчанию — `1`;
- необязательный атрибут `minOccurs` — минимальное число появлений данной последовательности элементов в родительском элементе XML-документа. Значение атрибута `0` указывает, что появление последовательности необязательно. По умолчанию — `1`;
- вложенные элементы `<annotation>`, `<element>`, `<group>`, `<choice>`, `<sequence>`, `<any>`.

Элемент `<attribute>` (рис. 1.3) описывает атрибуты XML-документа, используя следующие атрибуты и вложенные элементы:

- необязательный атрибут `default` задает значение атрибута XML-документа по умолчанию;
- необязательный атрибут `fixed` задает неизменяемое значение атрибута XML-документа, является взаимоисключающим с атрибутом `default`;
- необязательный атрибут `form` указывает, как атрибут должен представляться в XML-документе. Возможные значения атрибута — `qualified` (атрибут XML-документа уточняется значением атрибута `targetNamespace` элемента `<schema>`) и `unqualified` (атрибут представляется только локальным именем);

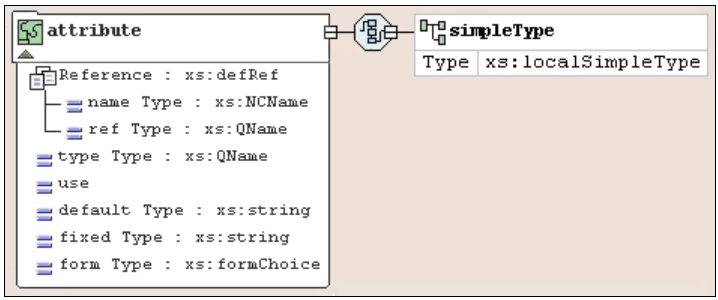


Рис. 1.3. Общая схема элемента `<attribute>`

- необязательный атрибут `id` — идентификатор в пределах схемы;
- необязательный атрибут `name` — имя атрибута, является обязательным, если родительский элемент — `<schema>`;
- необязательный атрибут `ref` — ссылка на уже объявленный атрибут, является взаимоисключающим с атрибутами `name`, `form`, `type` и `simpleType`;
- необязательный атрибут `type` — указывает встроенный тип данных XML Schema или простой тип данных для значений атрибута XML-документа;
- необязательный атрибут `use` — определяет использование атрибута XML-документа. Возможные значения — `optional` (по умолчанию атрибут является необязательным), `prohibited` (использование атрибута запрещено), required (наличие атрибута обязательно);
- вложенные элементы `<annotation>`, `<simpleType>`.

Элемент `<notation>`, являющийся дочерним элементом элемента `<schema>`, объявляет нотацию, описывающую формат данных, отличный от XML-формата.

Элемент `<notation>` имеет следующие атрибуты:

- необязательный атрибут `id` — идентификатор в пределах схемы;
- обязательный атрибут `name` — имя нотации;
- необязательный атрибут `system` — системный URI-идентификатор нотации;
- обязательный атрибут `public` — публичный URI-идентификатор нотации.

SOAP 1.2

SOAP — это протокол обмена сообщениями XML-формата в распределенной среде. Протокол SOAP является соглашением о структуре XML-документов, представляющих участвующие в обмене сообщения, которое позволяет передавать как символьную, так и двоичную информацию, используя различные протоколы SMTP, FTP, HTTP, HTTPS и др. Символьная информация может передаваться в теле SOAP-сообщения, а двоичная информация может быть прикреплена к SOAP-сообщению.

Использование XML-формата делает SOAP-протокол платформонезависимым, а гибкая структура SOAP-протокола позволяет использовать его поверх разнообраз-

ных транспортных протоколов и адаптировать SOAP-сообщения к различным требованиям распределенной среды, таким как безопасность, надежность и т. д.

В своем обсуждении SOAP-технологии будем опираться на спецификацию SOAP Version 1.2. Более ранняя спецификация SOAP 1.1 рассмотрена в приложении "Протокол SOAP 1.1" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Структура SOAP-сообщения представлена четырьмя основными элементами: `<Envelope>`, `<Header>`, `<Body>` и `<Fault>`, с локальными именами элементов и атрибутов, принадлежащих пространству имен `http://schemas.xmlsoap.org/soap-envelope`.

`<Envelope>` является корневым элементом, указывающим, что данный XML-документ является SOAP-сообщением.

Помимо объявления пространства имен `http://schemas.xmlsoap.org/soap-envelope` элемент `<Envelope>` может содержать объявления других пространств имен, используемых в SOAP-сообщении.

Необязательный элемент `<Header>` содержит информацию, которая, при получении SOAP-сообщения приложением, влияет на обработку тела сообщения. Элемент `<Header>` может включать в себя несколько дочерних элементов, называемых *блоками заголовка*, которые могут нести метаданные, связанные с безопасностью, надежностью, маршрутизацией и т. д.

Блоки заголовка могут иметь различные атрибуты, среди которых могут быть четыре важных атрибута, описываемых спецификацией, — это атрибуты `encodingStyle`, `role`, `mustUnderstand` и `relay`.

Атрибут `encodingStyle` должен иметь значение в виде URI-адреса правил или XML-схемы, описывающей, как были преобразованы (сериализованы) данные в формат SOAP-сообщения, и необходимой для десериализации передаваемых сообщением данных. Например, значением атрибута может быть URI-адрес стандартной SOAP-кодировки — `http://www.w3.org/2003/05/soap-encoding`.

> ***Примечание***
> Сериализация — это процесс преобразования объекта в XML-элемент.

При обмене сообщениями в распределенной среде, SOAP-сообщение может проходить путь от первоначального SOAP-отправителя к конечному SOAP-получателю через промежуточные SOAP-обработчики. При этом каждый блок заголовка предназначается для обработки определенным промежуточным SOAP-обработчиком или конечным SOAP-получателем. С помощью вышеперечисленных атрибутов в блоке заголовка указывается его обработчик и задается обязательность обработки данного блока заголовка.

Атрибут `role` указывает URI-идентификатор обработчика, для которого предназначается данный блок заголовка, определяя для SOAP-обработчика SOAP-роли. Спецификация определяет три стандартных значения для атрибута `role`:

- `http://www.w3.org/2003/05/soap-envelope/none` — данный блок заголовка непосредственно не обрабатывается ни одним приложением, в этом случае он несет информацию, необходимую при обработке других блоков заголовков;

- `http://www.w3.org/2003/05/soap-envelope/next` — каждый обработчик на всем пути SOAP-сообщения должен обрабатывать данный блок заголовка;
- `http://www.w3.org/2003/05/soap-envelope/ultimateReceiver` — только конечный SOAP-получатель обрабатывает данный блок заголовка. Это значение установлено по умолчанию.

Атрибут `mustUnderstand`, если он имеет значение `true`, обязывает обработчика обрабатывать данный блок заголовка. Если обработчик не понял данный блок заголовка, тогда, в случае значения атрибута `true`, он генерирует сообщение об ошибке и обработка SOAP-сообщения прерывается. Если значение атрибута `false`, тогда при возникновении ошибки обработка SOAP-сообщения не прерывается и блок заголовка игнорируется.

Атрибут `relay`, имея значение `true`, указывает, что данный блок заголовка, если он не был обработан промежуточным SOAP-обработчиком, т. к. возникла ошибка, остается в SOAP-сообщении при передаче его следующему обработчику вплоть до конечного SOAP-получателя. Промежуточные SOAP-обработчики, в случае успешной обработки блоков заголовков, удаляют их из SOAP-сообщения, при этом в результате обработки в SOAP-сообщение могут быть помещены заголовки с тем же либо измененным значением или новые заголовки. Удаляются также те блоки заголовков, которые были проигнорированы, если значение атрибутов `relay` и `mustUnderstand` установлено `false`. Если же значение атрибута `mustUnderstand` установлено `true`, тогда обработка SOAP-сообщения прерывается. Значение атрибута `false` является значением по умолчанию.

Блоки заголовка могут быть объединены в SOAP-модули логически по их функциональности, при этом каждый SOAP-модуль должен иметь свой URI-идентификатор.

Обязательный элемент `<Body>` предназначен для передачи основных данных конечному SOAP-получателю. Дочерние элементы `<Body>` содержат передаваемую информацию и могут иметь различные атрибуты, среди которых может быть атрибут `encodingStyle` (см. элемент `<Header>`). Спецификация определяет один дочерний элемент для элемента `<Body>` — элемент `<Fault>`, содержащийся в сообщении об ошибке, которое генерируется SOAP-обработчиком, и используемый для хранения информации об ошибках, возникающих при обработке SOAP-сообщения.

Элемент `<Fault>` содержит следующие дочерние элементы.

- Обязательный элемент `<Code>` — содержит обязательный элемент `<Value>` и дополнительный элемент `<Subcode>`. Элемент `<Value>` указывает код ошибки и имеет следующие возможные значения, представленные локальными именами, принадлежащими пространству имен `http://www.w3.org/2003/05/soap-envelope`:
  - `VersionMismatch` — в SOAP-сообщении неправильно указано локальное имя или пространство имен корневого элемента `<Envelope>`. В этом случае в генерируемое SOAP-сообщение об ошибке включается блок заголовка с локальным именем `Upgrade` (пространство имен `http://www.w3.org/2003/05/soap-envelope`), содержащий дочерний элемент `<SupportedEnvelope>`, который ука-

зывает поддерживаемые SOAP-обработчиком локальное имя и пространство имен элемента `<Envelope>`, используя атрибуты `qname` и `xmlns` соответственно;

- `mustUnderstand` — SOAP-обработчик не понимает блок заголовка со значением атрибута `mustUnderstand="true"`. В этом случае в генерируемое SOAP-сообщение об ошибке включается блок заголовка с локальным именем `NotUnderstood` (пространство имен `http://www.w3.org/2003/05/soap-envelope`), который указывает локальное имя и пространство имен вызывающего ошибку блока заголовка, используя атрибуты `qname` и `xmlns` соответственно;
- `DataEncodingUnknown` — SOAP-обработчик не поддерживает кодировку, указанную с помощью атрибута `encodingStyle`;
- `Sender` — SOAP-сообщение содержит синтаксические ошибки или некорректную информацию;
- `Receiver` — ошибка относится не к содержимому SOAP-сообщения, а к сбою при его обработке.

Элемент `<Subcode>` является контейнером для обязательного элемента `<Value>`, уточняющего код ошибки, указанный элементом `<Value>` верхнего уровня, и дополнительного элемента `<Subcode>`. Такая иерархическая структура элемента `<Subcode>` позволяет наиболее полно описать ошибку, возникающую при обработке SOAP-сообщения:

- обязательный элемент `<Reason>` — описывает возникшую ошибку с помощью дочерних элементов `<Text>`, содержащих текстовую информацию об ошибке. Каждый элемент `<Text>` должен иметь свое значение обязательного атрибута `lang` (пространство имен `http://www.w3.org/XML/1998/namespace`);
- необязательный элемент `<Node>` — указывает URI-идентификатор SOAP-обработчика, сгенерировавшего сообщение об ошибке. Элемент является обязательным для промежуточных SOAP-обработчиков;
- необязательный элемент `<Role>` — указывает SOAP-роль обработчика в момент возникновения ошибки;
- необязательный элемент `<Detail>` — содержит дополнительную текстовую информацию об ошибке. Дочерние элементы могут иметь атрибут `encodingStyle` (см. элемент `<Header>`).

Из обсуждения структуры SOAP-сообщения видно, что оболочка сообщения не зависит от протокола, по которому SOAP-сообщение передается. Использование различных протоколов для передачи SOAP-сообщения регулируется стандартами взаимодействия протоколов. Спецификация описывает стандарты взаимодействия протоколов для передачи SOAP-сообщения поверх протоколов HTTP и SMTP.

При использовании метода POST HTTP-протокола SOAP-сообщение оформляется следующим образом:

```
POST / [имя ресурса] HTTP/1.1
Host: [имя сервера]
```

```
Content-Type: application/soap+xml; charset="utf-8"; action="[URI адресата]"
Content-Length: bytes
?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
. . .
</env:Envelope>
```

При этом SOAP-сообщение передается в теле HTTP-запроса и HTTP-отклика.

Использование метода GET HTTP-протокола подразумевает, что клиент посылает GET-запрос SOAP-серверу, который в ответ выдает клиенту SOAP-сообщение. Такая модель делает взаимодействие с ресурсом безопасным, т. е. не изменяющим ресурс, т. к. при GET-запросе производится только извлечение информации. При использовании в SOAP HTTP-метода GET, GET-запрос должен иметь следующую структуру:

```
GET [URI ресурса + параметры]  HTTP/1.1
Host: [имя сервера]
Accept: text/html; q=[коэффициент предпочтения формата text/html, как правило,
0.5], application/soap+xml
```

HTTP-отклик на GET-запрос состоит из стандартных HTTP-заголовков и SOAP-сообщения:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
. . .
</env:Envelope>
```

При передаче SOAP-сообщения с помощью протокола SMTP, SOAP-сообщение включается в тело e-mail-сообщения:

```
From: [отправитель]
To: [получатель]
Subject: [тема]
Date: [дата]
Message-Id: [идентификатор]
Content-Type: application/soap+xml

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
. . .
</env:Envelope>
```

Достаточно часто задачей SOAP-сообщения является передача не только символьных данных, но различного рода двоичных данных — изображений, аудио- и ви-

деофайлов и т. д. Двоичные данные могут передаваться по SOAP-протоколу различными способами:

- в форматах `hexBinary` и `base64Binary` в теле SOAP-сообщения;
- SOAP Attachment — механизм прикрепления двоичных данных к SOAP-сообщению;
- SOAP MTOM — оптимизированный механизм передачи двоичных данных.

Если двоичные данные передаются непосредственно в теле SOAP-сообщения, они должны быть закодированы в виде символьных последовательностей, например в виде `hexBinary`- или `base64Binary`-данных. Такое кодирование, а затем раскодирование, в случае пересылки двоичных данных большого размера занимает большие ресурсы и достаточно неэффективно. Проблема пересылки данных в теле SOAP-сообщения также возникает, если двоичные данные имеют цифровую подпись или необходимо в SOAP-сообщении переслать XML-фрагменты, имеющие различную символьную кодировку.

Спецификация SOAP 1.2 Attachment Feature описывает механизм передачи двоичных данных в виде прикрепленных к основному SOAP-сообщению частей, каждая из которых характеризуется своим URI-идентификатором. Реализация такого механизма может быть разной:

- основное SOAP-сообщение и вложения упаковываются в сообщение формата DIME, которое передается по базовому протоколу;
- основное SOAP-сообщение и вложения упаковываются в сообщение формата MIME Multipart/Related, которое передается по базовому протоколу;
- основное SOAP-сообщение передается по HTTP-протоколу без вложений, которые запрашиваются клиентом отдельно с помощью HTTP GET-запроса.

Direct Internet Message Encapsulation (DIME) — формат сообщений, позволяющий упаковывать данные различного типа и размера в одно сообщение. Каждая часть упакованных данных (DIME-запись) характеризуется типом данных, размером и необязательным идентификатором. DIME-записи большого размера или DIME-записи для динамически генерируемых данных, размер которых изначально неизвестен, формат DIME разбивает на "фрагменты записи" (DIME-фрагменты). Сообщение в формате DIME создается DIME-генератором, который, упаковывая данные в DIME-записи, размечает их DIME-заголовками, содержащими двоичные данные, используемые затем DIME-анализатором для интерпретации сообщения. Каждый DIME-заголовок содержит следующие поля:

- `VERSION` (5 бит) — версия формата. Все DIME-записи одного сообщения должны иметь одно и то же значение этого поля;
- `MB` (1 бит) — если значение поля равно `1`, тогда DIME-запись является первой в сообщении;
- `ME` (1 битт) — если значение поля равно `1`, тогда DIME-запись является последней в сообщении;

- `CF` (1 бит) — если значение поля равно `1`, тогда это DIME-фрагмент. Все DIME-фрагменты одной DIME-записи должны иметь один и тот же тип данных и идентификатор;
- `TYPE_T` (4 бита) — структура и формат поля `TYPE`;
- `RESERVED` (4 бита) — зарезервированное поле для расширений формата;
- `OPTIONS_LENGTH` (16 бит) — длина поля `OPTIONS`;
- `ID_LENGTH` (16 бит) — длина поля `ID`;
- `TYPE_LENGTH` (16 бит) — длина поля `TYPE`;
- `DATA_LENGTH` (32 бита) — длина поля `DATA`;
- `OPTIONS` — дополнительная информация для DIME-анализатора;
- `ID` — идентификатор;
- `TYPE` — тип данных;
- `DATA` — данные.

При упаковке SOAP-сообщения с вложениями в DIME-сообщение основное SOAP-сообщение является первой DIME-записью, а вложения упаковываются как последующие DIME-записи. Основное SOAP-сообщение ссылается на вложения, используя идентификаторы соответствующих DIME-записей как значения атрибута `href`. При пересылке DIME-сообщения по HTTP-протоколу HTTP-заголовок `Content-type` должен иметь значение `application/dime` вместо `application/soap+xml`.

Multipurpose Internet Mail Extensions (MIME) — стандарт, описывающий передачу данных различного типа через Интернет. MIME-тип Multipart/Related определяет механизм передачи сообщения, состоящего из множества взаимосвязанных частей. При этом части такого сообщения отделяются друг от друга разделительными строками. Таким образом, основное отличие формата MIME Multipart/Related от DIME состоит в том, что при передаче сообщения MIME Multipart/Related его части не характеризуются длиной, при отправке каждой части такого сообщения в конце просто добавляется разделительная строка.

При пересылке по HTTP-протоколу SOAP-сообщения с вложениями, как MIME Multipart/Related сообщения, в начале сообщения в стандартном наборе HTTP-заголовков должен быть HTTP-заголовок `Content-Type` со значением `multipart/related` и следующими параметрами:

- `type` — значение параметра должно быть `application/soap+xml`;
- `start` — идентификатор первой части сообщения, которая должна быть основным SOAP-сообщением. Поэтому значение параметра `type` устанавливается равной `application/soap+xml`, т. к. данный параметр указывает тип данных корневой части составного сообщения;
- `start-info` — содержит информацию для обработчика сообщения;
- `boundary` — содержит строку-разделитель частей сообщения.

Далее следует строка-разделитель, после которой идут заголовки первой части сообщения и основное SOAP-сообщение:

```
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: [идентификатор]
Content-Location: [URI-адрес основного SOAP-сообщения]

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
. . .
</SOAP-ENV:Envelope>
```

Заголовок `Content-Type` указывает тип данных части сообщения, его параметр `charset` — символьную кодировку.

Заголовок `Content-Transfer-Encoding` указывает представление данных при пересылке. Возможные значения этого заголовка:

- `7bit` — короткие строки с кодировкой US-ASCII;
- `8bit` — короткие строки с символами не ASCII;
- `binary` — длинные строки с символами не ASCII.

Далее в сообщении следуют строки-разделители, ограничивающие вложения, каждое из которых предваряется заголовками:

```
Content-Type: [тип данных, например image/tiff]
Content-Transfer-Encoding: [представление данных при пересылке, например,
binary]
Content-ID: [идентификатор вложения]
Content-Location: [URI-адрес вложения]
```

Основное SOAP-сообщение ссылается на вложения, используя значения заголовка `Content-ID` или заголовка `Content-Location` соответствующих вложений как значения атрибута `href`.

Спецификация SOAP Message Transmission Optimization Mechanism (MTOM) описывает механизм оптимизации передачи SOAP-сообщений, включающих в себя двоичные данные, закодированные в формате `base64Binary`. При реализации механизма MTOM, приложение, отправляющее такое SOAP-сообщение, создает пустой XOP-пакет (XML-binary Optimized Packaging), затем определяет в SOAP-сообщении данные формата `base64Binary`, которые нуждаются в оптимизации, основываясь на размере этих данных. После определения данных для оптимизации производится их декодирование из формата `base64Binary` обратно в двоичный формат. Затем формируется для пересылки окончательный XOP-пакет, состоящий из основного SOAP-сообщения и вложений, представляющих собой декодированные при оптимизации данные. В случае если оптимизации подвергается SOAP-сообщение с вложениями, первоначальные вложения просто копируются в XOP-пакет.

Как правило, для создания XOP-пакета используется стандарт MIME Multipart/Related. При этом упаковка MIME Multipart/Related XOP имеет свои особенности:

- параметр `type` заголовка `Content-Type` должен иметь значение `application/xop+xml`;
- параметр `start-info` заголовка `Content-Type` должен иметь значение, такое же, как и у параметра `type` заголовка `Content-Type` корневой части пакета. В случае упаковки SOAP-сообщения корневая часть — это основное SOAP-сообщение, а значение параметра `type` и, соответственно, параметра `start-info` — `application/soap+xml`;
- в основном SOAP-сообщении первоначальные данные формата `base64Binary`, подвергшиеся оптимизации, заменяются элементами:

```
<xop:Include xmlns:xop='http://www.w3.org/2004/08/xop/include'
             href='cid:[идентификатор вложения]'/>
```

Для удобства представления вызова удаленных процедур (Remote Procedure Call, RPC) в SOAP-сообщениях спецификация вводит понятие стиля SOAP-сообщений RPC.

Сообщения SOAP RPC являются подмножеством SOAP-сообщений в стиле "документ" и отличаются от них тем, что имеют следующие ограничения:

- сообщение SOAP RPC должно передавать URI-адрес конечной точки RPC. URI-адрес может передаваться в блоке заголовка SOAP-сообщения, или, в случае передачи SOAP-сообщения по HTTP-протоколу, URI-адрес содержится в HTTP-заголовке;
- сообщение SOAP RPC должно передавать имя процедуры, а также имена и значения параметров процедуры.

Наиболее удобным способом передачи имени процедуры, имен и значений параметров процедуры в сообщении SOAP RPC является применение стандартных правил написания кода SOAP. Данные правила требуют, чтобы как вызов процедуры RPC, так и ответ клиенту были описаны в теле SOAP-сообщения (элемент `<Body>`). При этом:

- вызов процедуры описывается структурой, представленной одним элементом с именем, идентичным имени процедуры, и его вложенными элементами с именами, идентичными именам параметров процедуры;
- RPC-ответ описывается структурой, представленной одним элементом с именем, состоящим из имени процедуры и добавляемого, как правило, слова "Response", а также его вложенными элементами с именами, идентичными именам параметров процедуры на выходе. Если процедура имеет возвращаемое значение, тогда оно представлено элементом с локальным именем `result` пространства имен `http://www.w3.org/2003/05/soap-rpc`;
- если используется стандартное SOAP-кодирование (`encodingStyle="http://www.w3.org/2003/05/soap-encoding"`), тогда элемент `<Body>` SOAP-сообщения мо-

жет иметь только один дочерний элемент, описывающий RPC-запрос или RPC-ответ;

- при возникновении ошибок обработки RPC-запроса дополнительно устанавливаются следующие значения элемента `<Value>` (см. элемент `<Fault>`), представляющие код ошибки:
  - `Receiver` — недостаток памяти при выполнении процедуры;
  - `DataEncodingUnknown` — кодировка аргументов процедуры неизвестна;
  - `ProcedureNotPresent` — данная процедура не поддерживается;
  - `BadArguments` — ошибка аргументов процедуры.

WSDL 2.0

Спецификация WSDL определяет язык Web Services Description Language — XML-язык описания Web-сервисов. WSDL-описание Web-сервиса содержит имя и адрес сервиса, указывает способ и протокол взаимодействия с сервисом, а также информацию о политиках сервиса, определяющих условия взаимодействия с ним.

Язык WSDL описывает Web-сервисы распределенной среды как набор конечных точек, обменивающихся сообщениями. WSDL-описание Web-сервисов документирует такую распределенную среду и автоматизирует процессы коммуникации в ней.

WSDL-описание является платформонезависимым и может применяться к Web-сервисам, созданным с использованием различных языков программирования, платформ, моделей и систем сообщений.

В обсуждении WSDL будем основываться на спецификации Web Services Description Language (WSDL) Version 2.0. Более ранняя версия WSDL 1.1 рассмотрена в приложении "WSDL 1.1" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Язык WSDL описывает Web-сервис на двух уровнях:

- первый уровень — абстрактный. На этом уровне Web-сервис описывается в терминах сообщений, которые он получает и посылает. Абстрактный уровень содержит WSDL-элемент `<interface>`, определяющий операции и сообщения Web-сервиса;
- второй уровень — детальный. На этом уровне определяются детали протокола обмена сообщениями и детали формата сообщений, адреса сервиса и его функциональность. Детальный уровень содержит WSDL-элементы `<binding>`, `<endpoint>` и `<service>`.

Корневым элементом WSDL-документа является элемент `<description>` (пространство имен `http://www.w3.org/ns/wsdl`), представляющий собой контейнер для всех остальных WSDL-элементов верхнего уровня.

Элемент `<description>` (рис. 1.4) имеет обязательный атрибут `targetNamespace`, который определяет пространство имен для элементов WSDL-документа, а также не-

обязательные атрибуты, определяющие другие пространства имен, используемые в документе и не относящиеся к пространству имен `http://www.w3.org/ns/wsdl`.

Как корневой элемент `<description>`, так и все его элементы могут иметь вложенный элемент `<documentation>`, содержащий документацию к соответствующим элементам.

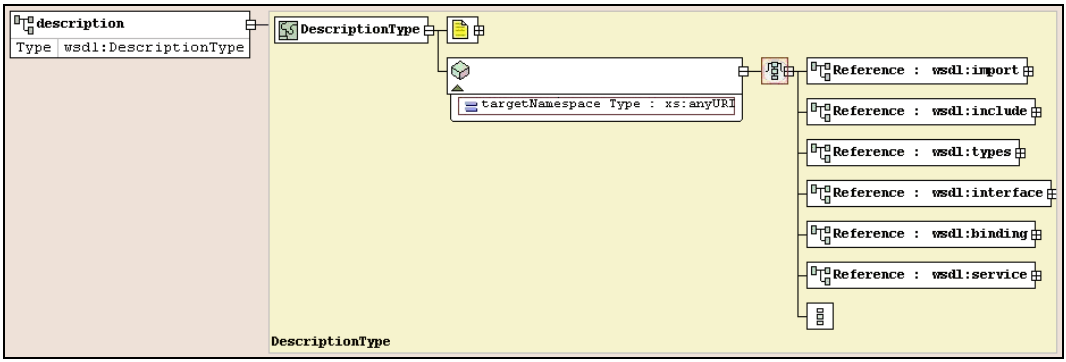


Рис. 1.4. Общая схема элемента `<description>`

Элемент верхнего уровня `<interface>` описывает интерфейс Web-сервиса как набор операций и сообщений об ошибках, при этом он может расширять другие интерфейсы.

Элемент `<interface>` описывает последовательности сообщений, которые сервис получает и отправляет. Последовательности входящих и выходящих сообщений группируются в операции сервиса, представляющие интерфейс сервиса.

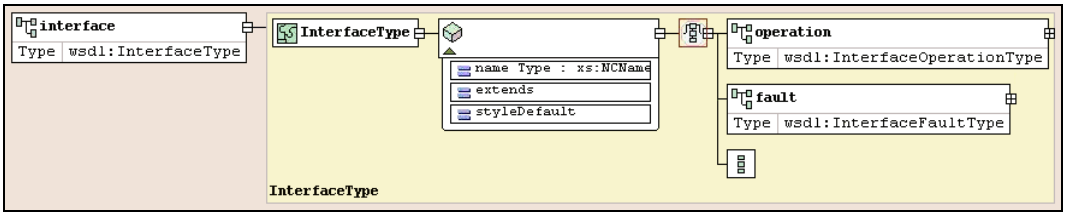


Рис. 1.5. Общая схема элемента `<interface>`

Элемент `<interface>` (рис. 1.5) имеет следующие атрибуты:

- `name` — обязательный атрибут, значение которого составляет вместе со значением атрибута `targetNamespace` элемента `<description>` QName-имя интерфейса;
- `extends` — дополнительный атрибут, перечисляет имена (атрибут `name`) интерфейсов, которые данный интерфейс расширяет;
- `styleDefault` — дополнительный атрибут, определяет стиль сообщений по умолчанию для всех операций интерфейса. Возможные значения атрибута:
  - стиль RPC (значение атрибута `http://www.w3.org/ns/wsdl/style/rpc`) подразумевает, что получение сообщения Web-сервисом вызывает выполнение за-

ранее известного метода интерфейса с параметрами. В случае формата сообщений SOAP, оформление SOAP-сообщений производится согласно правилам стиля RPC с указанием имени вызываемого метода, а также имен и значений параметров метода;

- стиль IRI (значение атрибута `http://www.w3.org/ns/wsdl/style/iri`) — при получении сообщения Web-сервис выполняет действия, основываясь на содержании сообщения. В случае формата сообщений SOAP, SOAP-сообщения оформляются в стиле документ;
- стиль Multipart (значение атрибута `http://www.w3.org/ns/wsdl/style/multipart`) — для XForm-клиента сообщение преобразуется в формат Multipart/form-data.

Примечание

Стиль сообщений никак не определяет модель программирования Web-сервиса. Web-сервис, реализующий вызов удаленной процедуры, может отправлять и получать сообщения в стиле "документ".

Элемент `<interface>` имеет вложенные элементы `<fault>` и `<operation>`.

Элемент `<fault>` описывает ошибку, которая может произойти при вызове операции интерфейса сервиса. При этом указывается имя (идентификатор) ошибки с помощью обязательного атрибута `name` и описывается содержимое сообщения об ошибке с помощью необязательного атрибута `element`.

Атрибут `element` элемента `<fault>` описывает содержимое сообщения об ошибке, используя объединение ссылки на QName соответствующего элемента `<element>`, объявленного в элементе `<types>`, и одно из следующих значений:

- `#any` — содержимое представлено любым XML-элементом;
- `#none` — содержимое пусто;
- `#other` (по умолчанию) — содержимое описывается языком, отличным от XML Schema.

Элемент `<operation>` (рис. 1.6) описывает операции Web-сервиса. В данном контексте операция — это обмен сообщениями с сервисом при его взаимодействии с клиентом. Последовательность и количество сообщений, участвующих в обмене, определяют обязательный атрибут `pattern` элемента `<operation>`.

Атрибут `pattern` задает для операции шаблон обмена сообщениями Message Exchange Patterns (MEP).

Спецификация описывает следующие шаблоны MEP и соответственно возможные значения атрибута `pattern`:

- шаблон In-Only (значение атрибута `http://www.w3.org/ns/wsdl/in-only`) обозначает, что Web-сервис получает единственное сообщение, при этом в случае возникновения ошибки сообщение о ней не генерируется;
- шаблон Robust In-Only (значение атрибута `http://www.w3.org/ns/wsdl/robust-in-only`) обозначает, что Web-сервис получает единственное сообщение, при

этом в случае возникновения ошибки может генерироваться сообщение о ней, которое доставляется отправителю сообщения;

- шаблон In-Out (значение атрибута `http://www.w3.org/ns/wsdl/in-out`) обозначает, что в обмене участвуют два сообщения — Web-сервис получает и отправляет сообщение. При этом в случае возникновения ошибки может генерироваться сообщение о ней;

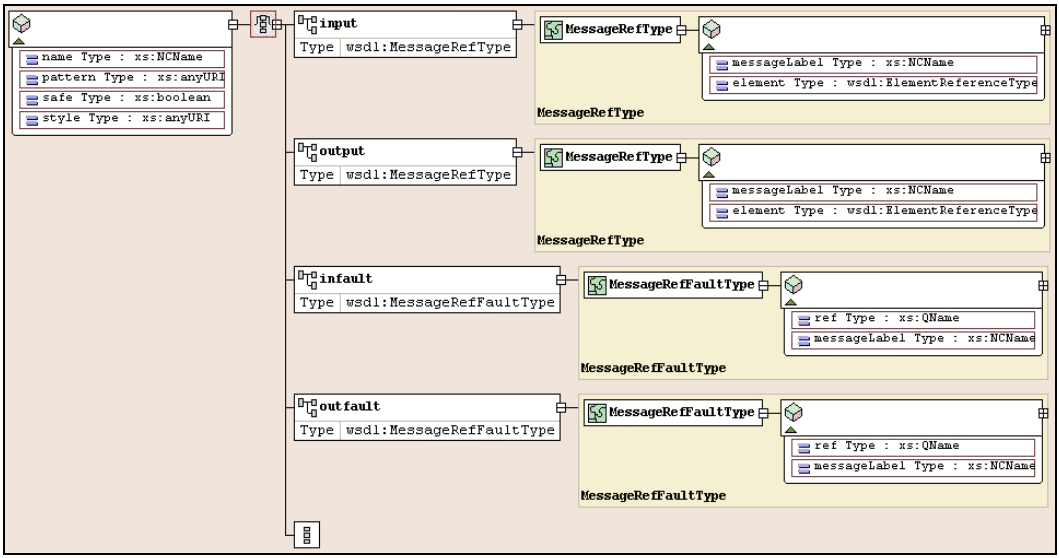


Рис. 1.6. Общая схема элемента `<operation>`

- шаблон In-Optional-Out (значение атрибута `http://www.w3.org/ns/wsdl/in-opt-out`) обозначает, что в обмене участвуют одно или два сообщения — Web-сервис получает сообщение и может отправить сообщение. При этом в случае возникновения ошибки может генерироваться сообщение о ней, которое доставляется отправителю сообщения;
- шаблон Out-Only (значение атрибута `http://www.w3.org/ns/wsdl/out-only`) обозначает, что Web-сервис отправляет единственное сообщение, при этом в случае возникновения ошибки сообщение о ней не генерируется;
- шаблон Robust Out-Only (значение атрибута `http://www.w3.org/ns/wsdl/robust-out-only`) обозначает, что Web-сервис отправляет единственное сообщение, при этом в случае возникновения ошибки может генерироваться сообщение о ней, которое доставляется Web-сервису;
- шаблон Out-In (значение атрибута `http://www.w3.org/ns/wsdl/out-in`) обозначает, что в обмене участвуют два сообщения — Web-сервис отправляет и получает сообщение. При этом в случае возникновения ошибки может генерироваться сообщение о ней;
- шаблон Out-Optional-In (значение атрибута `http://www.w3.org/ns/wsdl/out-opt-in`) обозначает, что в обмене участвуют одно или два сообщения — Web-сервис

отправляет сообщение и может получить сообщение. При этом в случае возникновения ошибки может генерироваться сообщение о ней, которое доставляется отправителю сообщения.

Элемент `<operation>` также имеет атрибуты: обязательный `name` (идентификатор операции) и дополнительные `style` (стиль операции, см. атрибут `styleDefault` элемента `<interface>`) и `safe` (декларирует безопасность вызова операции, по умолчанию значение атрибута равно `false`).

Атрибут `wrpc:signature` (пространство имен `http://www.w3.org/ns/wsdl/rpc`) вместе со стилем RPC позволяет описать параметры процедуры. Значение атрибута `wrpc:signature` — это последовательность имен параметров с указанием их типов (`#in`, `#out`, `#inout`, `#return`).

Безопасность операции, определяемая значением `true` атрибута `wsdlx:safe` (пространство имен `http://www.w3.org/ns/wsdl-extensions`), означает, что вызов операции не модифицирует ресурсы, и операция может быть вызвана много раз с одинаковым результатом.

Элемент `<operation>` является контейнером для элементов `<input>`, `<output>`, `<infault>` и `<outfault>`.

Элементы `<input>` и `<output>` описывают сообщения, участвующие в обмене с Web-сервисом. Данные элементы имеют необязательные атрибуты `messageLabel` и `element`.

Атрибут `messageLabel` элементов `<input>` и `<output>` указывает роль, которую сообщение играет в шаблоне MEP операции сервиса. Для шаблонов, описанных спецификацией, это значения атрибута `In` или `Out`.

Атрибут `element` элементов `<input>` и `<output>` описывает содержимое сообщения с помощью объединения ссылки на QName соответствующего элемента `<element>`, определенного в элементе `<types>`, и одного из следующих значений:

- `#any` — содержимое является любым XML-сообщением;
- `#none` — содержимое пусто;
- `#other` (по умолчанию) — содержимое описывается языком, отличным от XML Schema.

Элементы `<infault>` и `<outfault>` описывают входящие и исходящие сообщения об ошибках с помощью обязательного атрибута `ref`, который ссылается на вложенный элемент `<fault>` элемента `<interface>`, и необязательного атрибута `messageLabel`, указывающего роль сообщения в шаблоне MEP. Такое применение элемента `<fault>` и ссылок на него обеспечивает многократное использование описания ошибки для различных операций.

Элемент верхнего уровня `<service>` (рис. 1.7) описывает конкретную реализацию интерфейса Web-сервиса как набор конечных точек (endpoints), представляющих адреса для доставки сообщений Web-сервиса.

Элемент `<service>` имеет обязательные атрибуты `name` (имя сервиса) и `interface` (имя элемента `<interface>`). Элемент `<service>` служит контейнером для элементов `<endpoint>`, описывающих конечные точки сервиса.

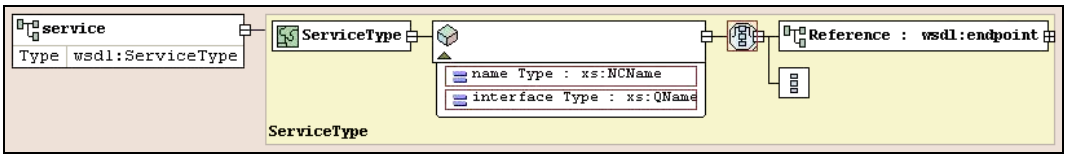


Рис. 1.7. Общая схема элемента `<service>`

Элемент `<endpoint>` имеет обязательные атрибуты `name` (имя конечной точки сервиса) и `binding` (ссылка на соответствующий элемент `<binding>`), а также дополнительный атрибут `address` (URI-адрес конечной точки).

Элемент верхнего уровня `<binding>` (рис. 1.8) конкретизирует формат сообщений (SOAP), протокол передачи сообщений (HTTP) и MEP-модель, что обеспечивает доступ к сервису.

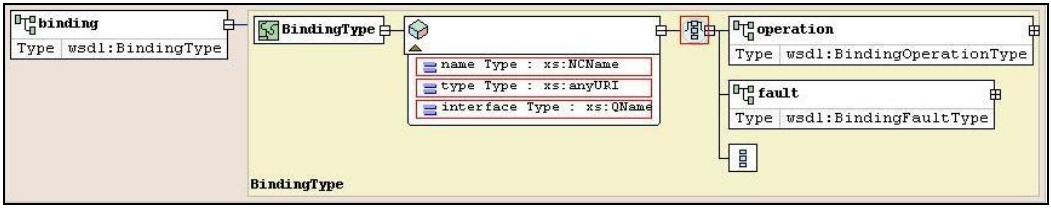


Рис. 1.8. Общая схема элемента `<binding>`

Такая конкретизация необходима для каждой операции и ошибки интерфейса Web-сервиса. Однако если расширение WSDL для конкретного формата сообщений и протокола передачи сообщений обеспечивает правила по умолчанию, тогда элемент `<binding>` может быть не связан напрямую с конкретным интерфейсом и может не конкретизировать каждую его операцию. В этом случае связь элемента `<binding>` с интерфейсом Web-сервиса осуществляется косвенно через атрибут `binding` элемента `<endpoint>`, и описание `binding` автоматически применимо ко всем операциям интерфейса.

Элемент `<binding>` имеет обязательный атрибут `name`, значение которого составляет вместе со значением атрибута `targetNamespace` элемента `<description>` QName-имя элемента `<binding>`, и обязательный атрибут `type`, указывающий формат сообщений (для SOAP значение атрибута — `http://www.w3.org/ns/wsdl/soap`). Также элемент `<binding>` имеет необязательный атрибут `interface`, своим значением ссылающийся на соответствующее QName-имя элемента `<interface>`. Элемент `<binding>` служит контейнером для элементов `<operation>` и `<fault>`. Кроме того, вложенные элементы расширения WSDL обеспечивают информацию для конкретного формата сообщений и протокола передачи сообщений.

Вложенный элемент `<fault>` имеет обязательный атрибут `ref` (ссылка на QName-имя вложенного элемента `<fault>` элемента `<interface>`) и вложенные элементы расширения WSDL.

Вложенный элемент `<operation>` элемента `<binding>` имеет обязательный атрибут `ref` (ссылка на QName-имя вложенного элемента `<operation>` элемента `<interface>`)

и вложенные элементы <input>, <output>, <infault> и <outfault>, а также вложенные элементы расширения WSDL.

Элементы <input> и <output> имеют необязательный атрибут messageLabel (роль сообщения в шаблоне MEP) и вложенные элементы расширения WSDL.

Элементы <infault> и <outfault> имеют обязательный атрибут ref (ссылка на QName-имя вложенного элемента <fault> элемента <interface>), необязательный атрибут messageLabel (роль сообщения в шаблоне MEP) и вложенные элементы расширения WSDL.

Элементы расширения WSDL для протокола SOAP (пространство имен xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"):

- атрибут wsoap:version элемента <binding> — версия протокола SOAP, по умолчанию значение атрибута равно 1.2, что соответствует протоколу SOAP 1.2;
- атрибут wsoap:protocol элемента <binding> — протокол передачи сообщений. Например, для SOAP 1.2 — wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/", а для SOAP 1.1 — wsoap:protocol="http://www.w3.org/2006/01/soap11/bindings/HTTP/";
- атрибут wsoap:mepDefault элемента <binding> — MEP-шаблон обмена сообщениями протокола SOAP 1.2 (не применимо к SOAP 1.1) по умолчанию для всех операций. Возможные значения — http://www.w3.org/2003/05/soap/mep/request-response/ и http://www.w3.org/2003/05/soap/mep/soap-response/;
- элемент <wsoap:module> объявляет использование SOAP-модуля с помощью обязательного атрибута ref (ссылка на URI-идентификатор SOAP-модуля) и необязательного атрибута required (обязательность использования SOAP-модуля, значения true/false);
- элемент <wsoap:header> объявляет использование блока заголовка SOAP-сообщения с помощью обязательного атрибута element (ссылка на описание блока заголовка в элементе <type>), а также необязательных атрибутов mustUnderstand (если true, тогда блок заголовка SOAP-сообщения снабжается атрибутом mustUnderstand="true") и required (обязательность использования блока заголовка, значения true/false);
- атрибуты wsoap:code и wsoap:subcodes элемента <fault> — код ошибки и уточнение кода ошибки;
- атрибут wsoap:mep элемента <operation> — MEP-шаблон, см. wsoap:mepDefault;
- атрибут wsoap:action элемента <operation> — для протокола SOAP 1.2 значение атрибута http://www.w3.org/2003/05/soap/features/action/ указывает поддержку параметра action MIME-типа application/soap+xml.

Правила по умолчанию для протокола SOAP 1.2:

- по умолчанию атрибут wsoap:action не имеет значения;
- по умолчанию атрибут wsoap:mep имеет значение http://www.w3.org/2003/05/soap/mep/request-response/;

- если атрибут `wsoap:protocol` имеет значение `http://www.w3.org/2003/05/soap/bindings/HTTP/`, тогда при значении атрибута `wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response/"` HTTP-метод — `POST`, а при значении атрибута `wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response/"` HTTP-метод — `GET`.

Элементы расширения WSDL для протокола HTTP (пространство имен `xmlns:whttp="http://www.w3.org/ns/wsdl/http"`, значение атрибута `type="http://www.w3.org/ns/wsdl/http"` элемента `<binding>`):

- атрибут `whttp:methodDefault` элемента `<binding>` — HTTP-метод по умолчанию для операций интерфейса Web-сервиса, по умолчанию значение атрибута равно `POST`, если атрибут `wsdlx:safe="true"`, тогда значение атрибута — `GET`;
- атрибут `whttp:queryParameterSeparatorDefault` элемента `<binding>` указывает символ разделения параметров HTTP-запроса по умолчанию для операций интерфейса Web-сервиса;
- атрибут `whttp:cookies` элемента `<binding>` — поддержка cookies (`false` — по умолчанию);
- атрибут `whttp:contentEncodingDefault` элемента `<binding>` устанавливает значение заголовка `Content-Encoding` по умолчанию;
- атрибут `whttp:code` элемента `<fault>` указывает код ошибки;
- атрибут `whttp:contentEncoding` устанавливает значение заголовка `Content-Encoding`;
- элемент `<whttp:header>` объявляет HTTP-заголовок с помощью обязательных атрибутов `name` (имя заголовка) и `type` (тип значения заголовка), а также необязательного атрибута `required` (обязательность включения в сообщение данного заголовка, `true`/`false`);
- атрибут `whttp:location` элемента `<operation>` определяет шаблон сериализации данных в URL-запросе;
- атрибут `whttp:method` элемента `<operation>` — HTTP-метод операции;
- атрибут `whttp:inputSerialization` элемента `<operation>` указывает формат сериализации входящих сообщений. По умолчанию формат входящих сообщений для метода GET — `application/x-www-form-urlencoded`, для метода POST — `application/xml`, для метода PUT — `application/xml`, для метода DELETE — `application/x-www-form-urlencoded`;
- атрибут `whttp:outputSerialization` элемента `<operation>` указывает формат сериализации исходящих сообщений. По умолчанию для HTTP-методов GET, POST, PUT и DELETE формат исходящих сообщений — `application/xml`;
- атрибут `whttp:faultSerialization` элемента `<operation>` указывает формат сериализации сообщений об ошибках (`application/x-www-form-urlencoded`, `application/xml`, `multipart/form-data`);

Примечание

Формат сериализации application/x-www-form-urlencoded используется только для сообщений HTTP-запроса, при этом значением атрибута style элемента <operation> должно быть http://www.w3.org/ns/wsdl/style/iri. Формат сериализации multipart/form-data также используется только для сообщений HTTP-запроса, при этом значением атрибута style элемента <operation> должно быть http://www.w3.org/ns/wsdl/style/multipart.

- атрибут whttp:queryParameterSeparator элемента <operation> указывает символ разделения параметров HTTP-запроса для операций интерфейса Web-сервиса;
- атрибут whttp:contentEncodingDefault элемента <operation> устанавливает значение заголовка Content-Encoding по умолчанию;
- атрибут whttp:ignoreUncited элемента <operation> определяет, должны ли добавляться в строку HTTP-запроса элементы, не указанные в шаблоне whttp:location, по умолчанию значение атрибута — false;
- атрибут whttp:contentEncoding элементов <input> и <output> устанавливает значение заголовка Content-Encoding;
- атрибут whttp:authenticationScheme элемента <endpoint> указывает HTTP-схему аутентификации при доступе к Web-сервису. Возможные значения — basic (базовая схема аутентификации с помощью логина и пароля) или digest (схема аутентификации с помощью закодированной строки, содержащей логин и пароль);
- атрибут whttp:authenticationRealm элемента <endpoint> указывает группу, к которой принадлежит клиент, проходящий аутентификацию.

Элемент верхнего уровня <types> описывает тип содержимого сообщений, участвующих в обмене с Web-сервисом. Спецификация позволяет включать описания типов данных сообщений, основанные на языке XML Schema, непосредственно в WSDL-документ.

Описание содержимого сообщений с помощью языка XML Schema может быть включено в WSDL-документ двумя способами.

Первый способ — это импорт схемы в WSDL-документ. В этом случае используется вложенный элемент <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"?/> элемента <types>, где атрибут namespace указывает пространство имен импортируемой схемы, а атрибут schemaLocation — ее URI-адрес.

Второй способ — это применение элемента <types> в качестве контейнера для схемы. В этом случае используется вложенный элемент <xs:schema targetNamespace="xs:anyURI"?/> элемента <types>, содержащий схему.

Кроме того, спецификация позволяет использовать другие языки схем, помимо языка XML Schema, для описания содержимого сообщений Web-сервиса. Для этого должны быть реализованы элементы расширения WSDL, являющиеся вложенными элементами элемента <types>.

Два атрибута расширения WSDL — wsdlx:interface и wsdlx:binding (пространство имен xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions") — могут использовать-

ся для описания в элементе `<types>` сообщений, которые ссылаются на другие Web-сервисы. Значения атрибутов `wsdlx:interface` и `wsdlx:binding` — QName-имена соответственно элементов `<interface>` и `<binding>` WSDL-описания Web-сервиса, на который ссылается сообщение. В частности, данные атрибуты могут быть применены для описания XML-элемента сообщения, значением которого является URL-адрес конечной точки Web-сервиса.

Два элемента верхнего уровня — `<include>` и `<import>` — обеспечивают модульность WSDL-описания Web-сервиса.

Элемент `<include>` позволяет включить элементы `<interface>`, `<binding>` и `<service>` из другого WSDL-документа, имеющего то же пространство имен, что и основной WSDL-документ. Элемент `<include>` имеет обязательный атрибут `location`, значением которого является URL-адрес включаемого WSDL-документа.

Элемент `<import>` дает возможность ссылаться в основном WSDL-документе на элементы импортируемого WSDL-документа, принадлежащие другому пространству имен. Элемент `<import>` имеет обязательный атрибут `namespace`, указывающий пространство имен импортируемого WSDL-документа, и необязательный атрибут `location` — URL-адрес импортируемого WSDL-документа.

Практическое применение Web-сервисов

Web-сервисы реализуются в виде программных компонентов на основе какой-либо из платформ, например Microsoft .NET или Java, и разворачиваются на сервере приложений, например IIS или GlassFish. При этом кроссплатформенность взаимодействия с Web-сервисами обеспечивается XML-форматом их описания и XML-форматом сообщений, участвующих в обмене с Web-сервисами. Однако такая архитектура налагает и ограничения на применение технологии Web-сервисов.

Взаимодействие с Web-сервисом подразумевает разбор XML-сообщения при его получении, затем, после выполнения необходимых операций, упаковку результатов снова в XML-сообщение перед его отправкой. Такая обработка увеличивает накладные расходы. Поэтому при создании программного комплекса на одной платформе, работающего на одном компьютере или в локальной сети, для увеличения скорости работы лучше обойтись без включения в программный комплекс Web-сервисов. Если же Web-сервис создается с целью предоставления услуги для широкого круга потребителей, заранее не известных и клиентские приложения которых могут работать на различных платформах, или для включения в программный комплекс, компоненты которого работают в распределенной среде на различных платформах, тогда применение технологии Web-сервисов будет хорошим решением.

Проектирование распределенной SOA-системы, реализующей технологию Web-сервисов, основывается на классификации Web-сервисов по их типу взаимодействия с клиентом, по типу клиента для Web-сервиса, по модели обработки запроса Web-сервисом.

Исходя из модели обработки клиентского запроса, Web-сервисы подразделяются следующим образом:

- метод-ориентированные Web-сервисы;
- документ-ориентированные Web-сервисы;
- ресурс-ориентированные Web-сервисы.

В модели обработки *метод-ориентированного*, или RPC, Web-сервиса содержимое сообщения тесно связано с вызываемыми процедурами интерфейса, явно указывая имя процедуры, а также имена и значения параметров процедуры. Таким образом, главная задача такого сообщения — это передача параметров процедуры интерфейса RPC Web-сервиса. При получении сообщения RPC Web-сервисом данные сообщения преобразуются в значения параметров вызываемой процедуры, после выполнения которой формируется ответ клиенту, также тесно связанный с процедурой.

Такой подход имеет свои недостатки. Каждый клиент RPC Web-сервиса должен заранее точно знать все детали интерфейса Web-сервиса, что вызывает тесную связь между клиентом и Web-сервисом — при изменении имени или параметров процедуры необходимо делать обновление всех клиентов такого Web-сервиса.

В модели обработки *документ-ориентированного* Web-сервиса содержание сообщений явно не связано с процедурами интерфейса, и их структура полностью определяется XML-схемой, что дает возможность проверки сообщений относительно XML-схемы в реальном времени. Это создает более слабую связь между клиентом и Web-сервисом, т. к. все изменения производятся в XML-схеме, а не в WSDL-описании и самих сообщениях.

Документ-ориентированный Web-сервис оперирует сообщениями целиком, а не параметрами процедуры, как RPC Web-сервис. Такой подход имеет и свои недостатки — увеличивается сложность интерпретации функциональности Web-сервиса, ведь для этого дополнительно требуется анализ структуры сообщений, обрабатываемых Web-сервисом. Для строго документ-ориентированных Web-сервисов в WSDL-описании вообще не требуется определение операций — центральную роль играет XML-схема. Для строго метод-ориентированных Web-сервисов основную нагрузку несет WSDL-описание, содержащее определения процедур интерфейса и их параметров, а XML-схема не требуется.

Ресурс-ориентированные Web-сервисы основываются на архитектуре передачи состояния представления REST (Representational State Transfer) и оперируют представлениями ресурсов. RESTful Web-сервисы поддерживают четыре операции — GET, PUT, POST и DELETE, — аналогичные соответствующим HTTP-методам, с помощью которых производятся запрашиваемые клиентом действия с ресурсами, представленными Web-сервисами, каждый из которых идентифицируется своим URL-адресом.

Архитектура REST упрощает анализ клиентского запроса, т. к. HTTP-методы запроса напрямую связываются с операциями REST Web-сервиса. Однако при этом каждый клиентский запрос, в случае строго ресурс-ориентированных Web-

сервисов, должен содержать всю необходимую информацию для его обработки, т. е. состояние клиентского запроса не сохраняется сервером. Кроме того, отличие REST Web-сервисов состоит в ограничении "один ресурс — один URL-адрес" и в названии и количестве операций, ограниченных GET, PUT, POST и DELETE.

Web-сервисы могут иметь клиентов двух типов. Первый тип клиента — это клиентское приложение, взаимодействие которого с Web-сервисом инициирует реальный пользователь. Такого рода приложения могут быть либо "тонким" клиентом (Web-браузер), либо "толстым" клиентом (самостоятельные настольные приложения с графическим интерфейсом пользователя). Второй тип клиента — это программный компонент, автоматически совершающий запрос Web-сервису без участия человека.

Взаимодействие клиента с Web-сервисом может быть синхронным или асинхронным. В случае *синхронного взаимодействия*, после отправки клиентом запроса Web-сервису, все его действия блокируются до тех пор, пока не будет получен ответ. При *асинхронном взаимодействии* действия клиента не блокируются, а обработка ответа производится после его получения от Web-сервиса.

Таким образом, существуют следующие основные сценарии взаимодействия с Web-сервисом:

- однонаправленный запрос — клиент посылает сообщение Web-сервису или Web-сервисам, которые его обрабатывают, но не генерируют ответ;
- синхронный "запрос — ответ" — синхронный обмен сообщениями между клиентом и Web-сервисом;
- асинхронный "запрос — ответ" — асинхронный обмен сообщениями между клиентом и Web-сервисом;
- RPC-вызов — клиент вызывает процедуру интерфейса Web-сервиса путем сериализации параметров процедуры в сообщение и отправки его Web-сервису;
- ошибки — выполнение процедуры интерфейса Web-сервиса завершается ошибкой, и клиент получает сообщение об ошибке;
- запрос с подтверждением — клиент получает от Web-сервиса сообщение об успешной или неудачной доставке клиентского запроса;
- передача через посредников — клиентское сообщение передается конечному Web-сервису через промежуточные Web-сервисы, при этом возможна маршрутизация и трекинг;
- кэширование — в случае схожего клиентского запроса ответ ему посылается из кэша;
- диалог — сообщения, участвующие в обмене клиента с Web-сервисом, группируются в логические наборы с сохранением состояния;
- дополнительная функциональность — обмен сообщениями производится с дополнительными требованиями, такими как кодирование (при сериализации пересылаемые данные дополнительно шифруются), аутентификация (клиент должен послать в запросе логин и пароль или сертификат для доступа к Web-

сервису), целостность (сообщения снабжаются электронной подписью для гарантии их целостности при пересылке), транзакции (при обмене сообщений устанавливается контекст транзакции) и др.

При создании Web-сервиса с целью предоставления услуги для широкого круга потребителей, заранее не известных, необходимо позаботиться о том, чтобы потенциальные пользователи Web-сервиса смогли его найти.

Для обычных пользователей Интернета Web-сервисы становятся доступными с помощью элементов управления, размещенных на Web-страничках сайтов. Заполняя форму и нажимая кнопку **Submit** или активируя гиперссылку, посетитель сайта, сам того не осознавая, может осуществлять запрос к Web-сервису.

Для Web-разработчиков публичный Web-сервис необходимо зарегистрировать в каком-либо из открытых реестров, чтобы его могли включить в SOA-систему. Такого рода реестры доступны в Интернете на Web-страницах сайтов, публикующих списки Web-сервисов, которые предназначены для широкой аудитории. Примеры публичных Web-сервисов можно посмотреть по адресу **http://www.xmethods.net** или **http://www.service-repository.com**.

При использовании Web-сервисов в корпоративных средах создаются специальные закрытые реестры, позволяющие осуществлять как статический, так и динамический поиск нужного Web-сервиса. Существуют различные стандарты для создания корпоративных реестров. Наиболее распространенные среди них — это UDDI, ebXML и DISCO.

UDDI

UDDI (Universal Description, Discovery and Integration, **http://uddi.xml.org**) — стандарт для создания платформонезависимого, основанного на XML, реестра, позволяющего публиковать и находить WSDL-описания Web-сервисов.

UDDI-реестр дает возможность доступа потребителям Web-сервисов к информации о поставщиках, функциональности и деталях реализации Web-сервисов. Для администрирования UDDI-реестров создаются или используются готовые UDDI-серверы, представляющие собой приложения, которые обеспечивают хранение и управление информацией о Web-сервисах в различного рода базах данных, включая классификацию, каталогизацию, обновление версий и тестирование Web-сервисов. Кроме того, подписка на UDDI-реестр гарантирует получение уведомлений об изменениях в реестре. По своей сути UDDI-сервер представляет собой набор Web-сервисов, осуществляющих управление UDDI-реестром. Пример UDDI-сервера с открытым исходным кодом можно посмотреть по адресу **http://openuddi.sourceforge.net** или **http://ws.apache.org/juddi**.

UDDI-реестр логически разделен на три типа каталогов, образующих иерархию, подобную телефонному справочнику:

- белые страницы — предоставляют информацию о поставщиках Web-сервисов, включая имя поставщика, его описание, контактную информацию;
- желтые страницы — классифицируют Web-сервисы по сферам применения;

- зеленые страницы — предоставляют информацию о деталях реализации Web-сервисов, необходимую для доступа к ним.

Информация, относящаяся к вышеперечисленным каталогам, описывается моделью данных UDDI, представляющей данные в виде наборов сущностей, каждая из которых хранится в виде XML-элемента соответствующего XML-файла:

- `<businessEntity>` описывает поставщика Web-сервиса;
- `<businessService>` описывает набор Web-сервисов, предоставляемых определенным поставщиком;
- `<bindingTemplate>` содержит техническую информацию о реализации Web-сервиса, ссылается на `tModel`;
- `<tModel>` содержит информацию о типе Web-сервиса, протоколах, используемых Web-сервисом, дополнительных требованиях к использованию Web-сервиса, характеристики Web-сервиса и т. д.;
- `<publisherAssertion>` описывает отношения между элементами `businessEntity`;
- `<subscription>` устанавливает требование отслеживать изменения в наборе сущностей.

WSDL-описание Web-сервиса и UDDI-документ соотносятся между собой следующим образом:

- WSDL-элемент `<interface>` описывается UDDI-элементом `<tModel>`;
- WSDL-элемент `<service>` описывается UDDI-элементами `<businessService>` и `<bindingTemplate>`.

Каждый хранящийся в UDDI-реестре XML-файл с данными имеет свой уникальный UDDI-идентификатор и структуру, которая определяется XML-схемами.

Управление информацией UDDI-реестра осуществляют UDDI-узлы, представляющие собой наборы Web-сервисов, поддерживающих специальные узловые API:

- UDDI-запрос;
- UDDI-публикация;
- UDDI-безопасность;
- UDDI-хранение;
- UDDI-подписка;
- UDDI-репликация.

Клиент UDDI-реестра, в свою очередь, должен поддерживать следующие клиентские интерфейсы API:

- UDDI-прослушивание подписки;
- UDDI-установка пользовательского набора значений, характеризующих Web-сервис для более эффективного поиска.

Все UDDI API представлены XML-элементами, которые упаковываются в SOAP-сообщения, участвующие в обмене между клиентом и UDDI-узлом. При этом кон-

кретная UDDI-реализация в виде UDDI-сервера предоставляет клиентам свою библиотеку API, позволяющую программным образом взаимодействовать с UDDI-реестром и динамически осуществлять публикацию и поиск Web-сервисов.

ebXML

ebXML (Electronic Business using eXtensible Markup Language, **http://ebxml.xml.org**) — стандарт, основанный на XML и обеспечивающий инфраструктуру обмена бизнес-информацией и создание общей XML бизнес-среды для различных компаний. Стандарт ebXML позволяет создавать с помощью XML бизнес-документы и описания бизнес-процессов и хранить их в реестре, давая возможность статически и динамически осуществлять поиск информации о бизнес-партнерах. Таким образом, предназначение ebXML гораздо шире, чем хранение и управление информацией о Web-сервисах в корпоративной среде. Основной целью стандарта ebXML является создание единого глобального электронного рынка. Поэтому организация ebXML-реестра возможна как для отдельной корпоративной среды, так и для консорциума, объединяющего компании определенной индустрии.

Стандартный сценарий использования реализации ebXML состоит из следующих шагов:

1. Компания *A* решает присоединиться к использованию ebXML-реестра, после этого она запрашивает у ebXML-реестра требования к локальной ebXML-реализации. Далее компания *A* устанавливает свою локальную ebXML-систему, позволяющую взаимодействовать с ebXML-реестром и локальными ebXML-системами бизнес-партнеров.
2. Компания *A* создает и регистрирует для хранения профиль Collaboration Protocol Profile (CPP) в ebXML-реестре, используя свою локальную ebXML-систему. Компания может зарегистрировать несколько CPP-профилей, описывающих различные возможности сотрудничества компании для различных регионов или подразделений компании. Каждому CPP-профилю в ebXML-реестре присваивается свой уникальный глобальный идентификатор GUID. CPP-профиль — это XML-документ, структура которого определяется следующими элементами:
   - `<CollaborationProtocolProfile>` — корневой элемент CPP-профиля. Элемент имеет атрибуты, определяющие пространства имен для элементов документа (`xmlns:tp="http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd"`), для XML Digital Signature (`xmlns:ds="http://www.w3.org/2000/09/xmldsig#"`) и для XLink (`xmlns:xlink="http://www.w3.org/1999/xlink"`), а также атрибуты `cppid` (идентификатор документа) и `version` (версия XML-схемы документа);
   - `<PartyInfo>` — информация о компании в целом или о подразделениях компании. В последнем случае присутствует несколько элементов `<PartyInfo>`. Элемент имеет атрибуты `partyName` (имя организации или подразделения), `defaultMshChannelId` (указывает элемент `<DeliveryChannel>`, используемый по умолчанию для асинхронной доставки сообщений, переопределяется элемен-

том <OverrideMshActionBinding>) и defaultMshPackageId (указывает элемент <Packaging>, используемый по умолчанию для отправки сообщений). Элемент <PartyInfo> содержит следующие дочерние элементы:

- <PartyId> — идентификатор;
- <PartyRef> — с помощью атрибута xlink:href указывает URI-ссылку на ресурс, содержащий дополнительную информацию о компании или подразделении;
- <CollaborationRole> — роль, которую компания или подразделение выполняет в контексте Business Process Specification, например покупатель или продавец. Элемент <CollaborationRole> содержит вложенные элементы: <ProcessSpecification> (ссылка на документ Business Process Specification, определяющий взаимодействие с бизнес-партнерами, т. е. бизнес-процессы), <Role> (указывает, какая роль в документе Business Process Specification поддерживается), <ApplicationCertificateRef> (указывает сертификат, используемый на уровне приложения), <ApplicationSecurityDetailsRef> (указывает политику безопасности, используемую на уровне приложения) и <ServiceBinding> (указывает элемент <DeliveryChannel>, используемый для обмена сообщениями). Элемент <ServiceBinding> содержит вложенные элементы <Service> (указывает Web-сервис, оперирующий сообщениями), <CanSend> (описывает детали исходящих сообщений), <CanReceive> (описывает детали входящих сообщений);
- <Certificate> — сертификат, используемый в целях безопасности;
- <SecurityDetails> описывает политику безопасности и доверенные сертификаты с помощью вложенных элементов <TrustAnchors> и <SecurityPolicy>;
- <DeliveryChannel> описывает детали обмена сообщениями, включая протоколы с помощью вложенного элемента <MessagingCharacteristics> и ссылок на элементы <Transport> и <DocExchange>;
- <Transport> описывает детали транспортного протокола обмена сообщениями с помощью вложенных элементов <TransportSender> и <TransportReceiver>;
- <DocExchange> описывает детали обмена сообщениями, такие как электронная подпись и шифрование, используя вложенные элементы <ebXMLSenderBinding> и <ebXMLReceiverBinding>;
- <OverrideMshActionBinding> определяет элемент <DeliveryChannel>, используемый для асинхронной доставки сообщений;

- <SimplePart> — компоненты, используемые для создания составных сообщений; имеет атрибуты id (идентификатор компонента) и mimetype (тип содержимого части сообщения);
- <Packaging> описывает упаковку сообщений перед отправкой с помощью вложенных элементов <ProcessingCapabilities> и <CompositeList>;

- `<Signature>` содержит электронные подписи CPP-профиля;
- `<Comment>` — комментарии.

XML-документы Business Process Specification, на которые ссылается CPP-профиль, описывают взаимоотношения с бизнес-партнерами, роли и ответственность компании, хореографию бизнес-документов компании. Документы Business Process Specification также хранятся в ebXML-реестре.

3. Компания *B* находит с помощью своей локальной ebXML-системы в ebXML-реестре CPP-профиль компании *A*. Если компания *B*, после анализа CPP-профиля компании *A*, заинтересуется участием в бизнес-процессах компании *A*, тогда из двух CPP-профилей компаний *B* и *A* генерируется документ Collaboration Protocol Agreement (CPA), который затем посылается компании *A* для утверждения. После утверждения CPA-документа компания *A* пересылает его обратно компании *B*, и CPA-документ используется для конфигурирования обеих локальных ebXML-систем, что делает возможным обмен сообщениями между двумя компаниями. Далее две компании обмениваются бизнес-документами и следуют бизнес-процессам в соответствии с CPA-документом. CPA-документ также является XML-документом и имеет следующую структуру:

   - `<CollaborationProtocolAgreement>` — корневой элемент документа, имеет атрибуты `cpaid` (идентификатор документа) и `version` (версия XML-схемы);
   - `<Status>` — текущий статус документа (`proposed`, `agreed` и `signed`);
   - `<Start>` — дата и время начала создания документа;
   - `<End>` — дата и время окончания создания документа;
   - `<ConversationConstraints>` — ограничение количества взаимодействий между двумя компаниями с помощью атрибутов `invocationLimit` и `concurrentConversations`;
   - `<PartyInfo>`, структура элемента схожа со структурой элемента `PartyInfo` CPP-профиля за исключением того, что должны быть как минимум два элемента для каждой из компаний, участвующих в соглашении;
   - `<SimplePart>`, структура элемента схожа со структурой аналогичного элемента CPP-профиля;
   - `<Packaging>`, структура элемента схожа со структурой аналогичного элемента CPP-профиля;
   - `<Signature>` — цифровые подписи одной или обеих компаний;
   - `<Comment>`, структура элемента схожа со структурой аналогичного элемента CPP-профиля.

Таким образом, ebXML-реестр может служить хранилищем для CPP-документов, характеризующих компании и их возможности, документов Business Process Specification, описывающих бизнес-процессы, в которых компании могут участвовать, CPA-соглашений и других сопутствующих документов.

Локальные ebXML-системы компаний обмениваются между собой ebXML-сообщениями по протоколу ebXML Message Service Protocol с помощью Web-сервисов сообщений Messaging Service, детали которых описаны в документах CPP, CPA и Business Process Specification. В свою очередь ebXML-реестр имеет интерфейс, реализованный набором Web-сервисов, обеспечивающих взаимодействие с ebXML-реестром путем обмена ebXML-сообщениями между ebXML-реестром и локальными ebXML-системами компаний. Взаимодействие между ebXML-реестром и локальной ebXML-системой также может определяться отдельным CPA-соглашением.

Интерфейс ebXML-реестра состоит из двух частей:

- Life Cycle Management отвечает за управление объектами в ebXML-реестре;
- Query Management обеспечивает поиск информации в ebXML-реестре.

Web-сервисы ebXML-реестра могут поддерживать обмен сообщениями с клиентом как по SOAP-протоколу, так и по ebXML-протоколу. В случае обмена по SOAP-протоколу клиент ebXML-реестра должен получить WSDL-описание Web-сервисов реестра для того, чтобы начать с ним взаимодействие. В случае же обмена по ebXML-протоколу клиент должен заключить CPA-соглашение с ebXML-реестром.

При обмене сообщениями между двумя компаниями происходит обмен бизнес-документами. При этом бизнес-документы могут использовать компоненты библиотек Core Library и Business Library, которые локальные ebXML-системы компаний могут загрузить из ebXML-реестра. Библиотеки Core Library и Business Library содержат общую бизнес-информацию и описания общих бизнес-процессов для данного ebXML-реестра.

ebXML-описания Web-сервисов гораздо шире, чем WSDL- и UDDI-описания, т. к. включают в себя более детальную характеристику компаний и определения бизнес-процессов, а формат ebXML-сообщений является расширением SOAP-формата. Спецификация ebXML Message Specification определяет ряд элементов расширения для блоков заголовков и тела SOAP-конверта, относящихся к маршрутизации, деталям обмена, безопасности, надежности и т. д.

Познакомиться с конкретными реализациями технологии ebXML можно на официальном сайте проекта (**http://www.ebxml.org/tools**).

DISCO

DISCO — это технология компании Microsoft, предназначенная для публикации и поиска Web-сервисов.

Технология DISCO определяет формат документа, описывающий функциональность и способ взаимодействия с Web-сервисом. Такой DISCO-документ размещается в определенном каталоге сервера, предоставляющего данный Web-сервис. Технология DISCO поддерживается платформой Microsoft .NET, соответственно используемый сервер — это IIS (Internet Information Services).

DISCO-документ имеет следующую структуру:

- `<disco:discovery>` — корневой элемент, имеет обязательный атрибут `xmlns:disco="http://schemas.xmlsoap.org/disco/"`, указывающий пространство имен документа;
- `<contractRef>` — ссылка на WSDL-описание Web-сервиса с помощью атрибута `ref`, атрибут элемента `<docRef>` ссылается на дополнительную документацию Web-сервиса. Элемент `<contractRef>` принадлежит к пространству имен `xmlns="http://schemas.xmlsoap.org/disco/scl/"`;
- `discoveryRef` и `schemaRef` указывают ссылки на другие DISCO-документы и XML-схемы, используя атрибуты `ref`. Элемент `<schemaRef>` принадлежит к пространству имен `xmlns="http://schemas.xmlsoap.org/disco/schema/"`.

Поиск DISCO-документов осуществляется с помощью утилиты командной строки disco.exe, которая генерирует файл results.discomap, содержащий информацию о результатах поиска, и загружает все найденные файлы. Утилита disco.exe принимает в качестве аргумента URL-адрес DISCO-документа. Далее утилита командной строки wsdl.exe генерирует клиентские заглушки Web-сервиса, используя сгенерированный файл results.discomap.

DISCO-документ также может иметь следующий вид:

```
<dynamicDiscovery
  xmlns="urn:schemas-dynamicdiscovery:disco.2000-03-17">
</dynamicDiscovery>
```

В этом случае при запросе к данному DISCO-документу утилитой disco.exe происходит автоматическая генерация DISCO-документа, описывающего Web-сервисы, развернутые в каталоге vroot IIS-сервера. При этом с помощью элемента `<exclude>` первоначального DISCO-документа (атрибут `path`) можно исключить подкаталог vroot из поиска. Такой поиск называется *динамическим DISCO-поиском*.

JAXR

Библиотека Java API for XML Registries (JAXR) обеспечивает стандартный механизм для доступа к различным XML-реестрам, таким как ebXML или UDDI.

Клиентские приложения, использующие JAXR, могут быть Web-браузерами XML-реестров, компонентами Java EE или настольными приложениями. При этом реализация библиотеки JAXR должна обеспечивать доступ к Web-сервисам интерфейса конкретного XML-реестра.

Библиотека JAXR API состоит из двух пакетов:

- `javax.xml.registry.infomodel` (табл. 1.1) содержит интерфейсы, описывающие информационную модель JAXR, которая определяет типы объектов и отношения между объектами, хранящимися в XML-реестре;
- `javax.xml.registry` (табл. 1.2—1.4) содержит интерфейсы и классы, обеспечивающие доступ к XML-реестрам.

Таблица 1.1. *Интерфейсы пакета* `javax.xml.registry.infomodel`

Интерфейс	Описание
`Association`	Связь между двумя объектами `RegistryObject`
`AuditableEvent`	События, связанные с изменением состояния объекта `RegistryObject`
`Classification`	Характеристики объекта `RegistryObject`
`ClassificationScheme`	Схема классификации объекта `RegistryObject`
`Concept`	Элементы схемы классификации объекта `RegistryObject`
`EmailAddress`	E-mail-адрес
`ExtensibleObject`	Расширяемый динамически объект путем добавления атрибутов объекта
`ExternalIdentifier`	Дополнительные идентификаторы объекта `RegistryObject`
`ExternalLink`	Внешняя URI-ссылка
`ExtrinsicObject`	MIME-тип, связанный с объектом
`InternationalString`	Строка, которая должна быть локализована
`Key`	Уникальный идентификатор объекта `RegistryObject`
`LocalizedString`	Связывает строку с ее локализацией
`Organization`	Информация о компании
`PersonName`	Имя человека
`PostalAddress`	Почтовый адрес
`RegistryEntry`	Расширение объекта `RegistryObject` дополнительными метаданными
`RegistryPackage`	Набор логически связанных объектов `RegistryEntries`
`Service`	Объекты `RegistryObject`, обеспечивающие информацию о Web-сервисе, предоставляемом компанией
`ServiceBinding`	Объекты `RegistryObject`, обеспечивающие информацию о деталях реализации Web-сервиса, предоставляемого компанией
`Slot`	Обеспечивает динамическое добавление атрибутов объекту `RegistryObject`
`SpecificationLink`	Обеспечивает связь между `ServiceBinding` и технической документацией
`TelephoneNumber`	Телефонный номер
`URIValidator`	Обеспечивает проверку URI-адреса
`User`	Информация о зарегистрированных пользователях XML-реестра
`Versionable`	Обеспечивает управление версиями объектов

Таблица 1.2. *Интерфейсы пакета* `javax.xml.registry`

Интерфейс	Описание
`BulkResponse`	Обеспечивает доступ к набору объектов, возвращаемых в результате запроса к XML-реестру
`BusinessLifeCycleManager`	Обеспечивает функциональность интерфейса Life Cycle Management XML-реестра

Таблица 1.2 *(окончание)*

Интерфейс	Описание
`BusinessQueryManager`	Обеспечивает функциональность интерфейса Query Management XML-реестра
`CapabilityProfile`	Обеспечивает информацию о возможностях реализации библиотеки JAXR
`Connection`	Обеспечивает соединение между клиентом и XML-реестром
`DeclarativeQueryManager`	Обеспечивает выполнение SQL-зопросов или ebXML Filter Query запросов к XML-реестру
`FederatedConnection`	Обеспечивает соединение с распределенным XML-реестром
`FindQualifier`	Обеспечивает опции, повышающие эффективность поиска в XML-реестре
`JAXRResponse`	Обеспечивает информацию о запросе к XML-реестру
`LifeCycleManager`	Обеспечивает управление объектами XML-реестра
`Query`	Представляет SQL или ebXML Filter Query запрос
`QueryManager`	Обеспечивает доступ к ответу от XML-реестра
`RegistryService`	Обеспечивает доступ к возможностям реализации библиотеки JAXR

Таблица 1.3. *Класс пакета* `javax.xml.registry`

Класс	Описание
`ConnectionFactory`	Абстрактный базовый класс-фабрика для создания JAXR-соединений

Таблица 1.4. *Исключения пакета* `javax.xml.registry`

Исключения	Описание
`DeleteException`	Ошибка при выполнении операции удаления объекта из XML-реестра
`FindException`	Ошибка при выполнении поиска в XML-реестре
`InvalidRequestException`	Ошибка при выполнении запроса к XML-реестру
`JAXRException`	Общая JAXR-ошибка выполнения
`RegistryException`	Общая ошибка XML-реестра
`SaveException`	Ошибка при выполнении операции сохранения объекта в XML-реестре
`UnexpectedObjectException`	Ошибка при выполнении поиска в XML-реестре, связанная с ошибкой искомого объекта
`UnsupportedCapabilityException`	Ошибка при вызове метода, не поддерживаемого данной реализацией библиотеки JAXR

Языки WS-BPEL и WS-CDL

Основой корпоративной деятельности является бизнес-процесс. Бизнес-процесс — это последовательность действий, приводящая к заданному бизнес-результату. Компания может состоять из множества подразделений, которые вовлекаются в ее различные бизнес-процессы. Кроме того, компания может участвовать в бизнес-процессах с другими компаниями.

Для автоматизации учета и управления корпоративной деятельностью могут создаваться и внедряться распределенные SOA-системы, состоящие из Web-сервисов. Такие SOA-системы также могут автоматизировать деятельность как внутри отдельной компании, так и взаимодействуя между несколькими компаниями.

Работа корпоративной SOA-системы тоже основывается на выполнении бизнес-процессов, состоящих из взаимодействий входящих в SOA-систему Web-сервисов. Соответственно для проектирования корпоративной SOA-системы необходима формализация бизнес-процессов и описание их связи с Web-сервисами. Поэтому для моделирования бизнес-процессов в контексте технологии Web-сервисов были разработаны языки WS-BPEL (Web Services Business Process Execution Language) и WS-CDL (Web Services Choreography Description Language).

Язык WS-BPEL описывает оркестровку Web-сервисов, а язык WS-CDL — их хореографию. В чем же различие между оркестровкой и хореографией?

Оркестровка Web-сервисов — это описание локального бизнес-процесса компании в виде набора взаимодействий внутренних и внешних для данной компании Web-сервисов. При оркестровке контроль над бизнес-процессом осуществляется данной компанией с помощью центрального координатора, который также может быть представлен Web-сервисом. Центральный координатор управляет взаимодействием Web-сервисов, вовлеченных в бизнес-процесс, и определяет порядок выполнения ими заданных действий.

Хореография Web-сервисов — это описание глобального бизнес-процесса или, иначе, публичного протокола, с помощью которого несколько участников, которые могут быть и различными компаниями, обмениваются сообщениями с целью достижения общего бизнес-результата. При таком подходе не требуется центральный координатор, т. к. в данном случае каждый Web-сервис сам владеет информацией о том, когда выполнять свои операции и с каким другим Web-сервисом ему необходимо взаимодействовать.

Таким образом, оркестровка и хореография — это два разных подхода в описании бизнес-процессов и интеграции Web-сервисов. Оркестровка — описание бизнес-процесса с точки зрения центрального координатора, а хореография — описание бизнес-процесса как совместного действия различных участников, каждый из которых знает в нем свою роль. Можно сказать, что хореография Web-сервисов способна реализоваться набором оркестровок Web-сервисов.

WS-BPEL 2.0

Язык WS-BPEL предназначен для описания оркестровки бизнес-процесса на основе Web-сервисов. При этом BPEL-процесс представляется в виде XML-документа,

являющегося контейнером для декларации взаимоотношений между участниками процесса, обработки данных и выполняемых действий.

Спецификация WS-BPEL 2.0 поддерживает такие спецификации, как XML Schema 1.0, XPath 1.0, SOAP 1.1, WSDL 1.1, UDDI 2.0 и WS-Addressing.

XML-документ BPEL-процесса, как правило, создается с помощью BPEL-визуального редактора и затем исполняется BPEL-сервером управления бизнес-процессами. Таким образом, BPEL-процесс объединяет Web-сервисы и выступает как центральный координатор — к BPEL-серверу, исполняющему BPEL-процесс, поступают клиентские запросы, инициирующие запуск BPEL-процесса, и от BPEL-сервера исходят ответы клиенту. BPEL-процесс устанавливает порядок, в котором должны быть вызваны, последовательно или параллельно, объединенные BPEL-процессом Web-сервисы. При этом набор Web-сервисов, вовлеченных в конкретный процесс, может меняться в зависимости от того, какой клиент вызывает BPEL-процесс. Реализуется же BPEL-процесс в виде Web-сервиса, фактически являющегося составным из объединяемых Web-сервисов, т. е. такой новый составной Web-сервис и разворачивается в BPEL-сервере. Поэтому BPEL-проект состоит из BPEL-описания (*.bpel), WSDL-описания составного BPEL Web-сервиса (*.wsdl) и XML-схем, описывающих структуры данных проекта (*.xsd).

Технология BPEL поддерживается различными серверами приложений и средами разработок, в том числе Oracle BPEL Process Manager, Oracle Application Server, NetBeans, GlassFish, Eclipse, JBoss, WebLogic, WebSphere и др.

BPEL-описание может быть двух типов — абстрактное и исполняемое.

Абстрактное BPEL-описание дает общее описание бизнес-процесса, скрывая его конкретные детали исполнения и тем самым давая возможность по-разному реализовывать BPEL-процесс. Абстрактное BPEL-описание определяет, *что* должен BPEL-процесс делать, но не описывает, *как* он должен это делать. Абстрактное BPEL-описание может использоваться в качестве формальной документации или как шаблон для реализации в виде исполняемого BPEL-описания.

Исполняемое BPEL-описание полностью определяет и детализирует выполнение бизнес-процесса и поэтому сразу может выполняться конкретным BPEL-сервером.

Исполняемое BPEL-описание

Корневым элементом BPEL-документа является элемент `<process>` (рис. 1.9), который имеет следующие атрибуты:

- обязательный атрибут `name` — имя документа;
- обязательный атрибут `targetNamespace` — целевое пространство имен BPEL-описания;
- необязательный атрибут `queryLanguage` — используемый язык запросов, по умолчанию значение атрибута `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0` представляет использование языка XPath 1.0 для создания запросов;
- необязательный атрибут `expressionLanguage` — используемый язык выражений, по умолчанию значение атрибута `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0` представляет использование языка выражений XPath 1.0;

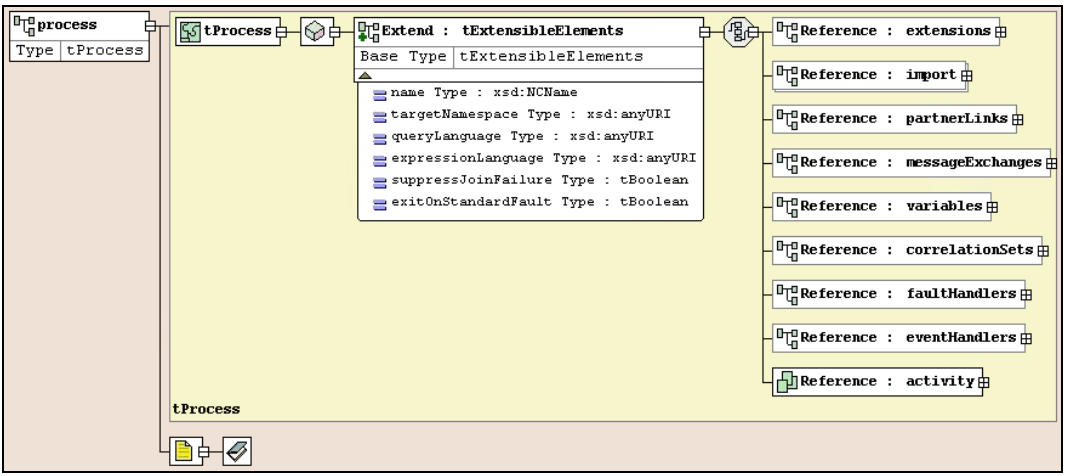


Рис. 1.9. Общая схема элемента `<process>` исполняемого BPEL-документа

- необязательный атрибут `suppressJoinFailure` определяет обработку ошибок условий, определенных в элементе `<joinCondition>`. Если значение атрибута `no` (по умолчанию), тогда исключение перехватывается соответствующим обработчиком ошибок, определенным элементом `<faultHandlers>`. Если значение атрибута `yes`, тогда действие, связанное с ошибочным условием, пропускается и производится проверка других условий, если они присутствуют;
- необязательный атрибут `exitOnStandardFault`. Если значение атрибута `yes`, тогда BPEL-процесс завершается при возникновении стандартной BPEL-ошибки (за исключением `joinFailure`). В противном случае (по умолчанию) стандартная BPEL-ошибка обрабатывается соответствующим обработчиком ошибок;
- обязательный атрибут `xmlns` определяет, является ли BPEL-документ абстрактным или исполняемым. Для абстрактного BPEL-документа пространство имен — `http://docs.oasis-open.org/wsbpel/2.0/process/abstract`, для исполняемого — `http://docs.oasis-open.org/wsbpel/2.0/process/executable`.

BPEL-описание может расширяться с помощью элементов `<extensions>`, которые могут содержать новые атрибуты и элементы, расширяющие BPEL-спецификацию. В качестве обязательных элемент `<extensions>` содержит атрибуты `namespace` (пространство имен расширений) и `mustUnderstand` (`yes` или `no`; указывает, обязательна ли для BPEL-процессора обработка расширений).

С помощью элемента `<scope>` можно разделить BPEL-процесс на части, каждая из которых будет реализовывать свою бизнес-логику. Элемент `<scope>` имеет обязательный атрибут `name` и определяет, что все его дочерние элементы имеют область видимости только в пределах данного элемента `<scope>`.

Элемент `<import>` позволяет объявить зависимость BPEL-описания от внешних XML-схем или WSDL-описаний. Элемент `<import>` имеет следующие атрибуты:

- необязательный атрибут `namespace` — пространство имен импортируемого документа;

- необязательный атрибут `location` — URI-адрес импортируемого документа;
- обязательный атрибут `importType` — URI-идентификатор кодировки импортируемого документа (`http://www.w3.org/2001/XMLSchema` — для документов XML Schema 1.0 и `http://schemas.xmlsoap.org/wsdl/` — для документов WSDL 1.1).

Элемент `<partnerLinks>` описывает связи BPEL-процесса с партнерами, которые могут быть Web-сервисами, вызываемыми BPEL-процессом, Web-сервисами, вызывающими BPEL-процесс и Web-сервисами, вызываемыми и вызывающими BPEL-процесс. Каждый BPEL-процесс имеет, по крайней мере, одну связь с вызывающим его клиентом и как минимум одну связь с вызываемым им Web-сервисом, исполняющим функцию, требуемую клиентом.

Элемент `<partnerLinks>` является контейнером для элементов `<partnerLink>`, описывающих конкретные связи с партнерами. Элемент `<partnerLink>` имеет следующие атрибуты:

- обязательный атрибут `name` — имя связи;
- обязательный атрибут `partnerLinkType` — ссылка на QName-имя элемента `<partnerLinkType>` WSDL-описания BPEL-процесса;
- необязательный атрибут `myRole` — роль BPEL-процесса в связи;
- необязательный атрибут `partnerRole` — роль партнера в связи;

> ***Примечание***
>
> Существует два типа связей BPEL-процесса с партнерами. Первый тип связи описывает асинхронное взаимодействие BPEL-процесса с партнером. В этом случае необходимо определить две роли. Например, BPEL-процесс асинхронно вызывает Web-сервис, тогда `partnerRole` указывает, что партнер предоставляет свой Web-сервис для BPEL-процесса, а `myRole` указывает, что производится обратный вызов BPEL-процесса для возврата ответа. Или клиент асинхронно вызывает BPEL-процесс, тогда `myRole` указывает, что BPEL-процесс предоставляет сервис клиенту, а `partnerRole` указывает, что производится обратный вызов клиента для возврата ответа. Второй тип связи описывает синхронное взаимодействие BPEL-процесса с партнером. В этом случае нужно определить только одну роль. Например, BPEL-процесс синхронно вызывает Web-сервис, тогда `partnerRole` указывает, что партнер предоставляет свой Web-сервис для BPEL-процесса. Или клиент синхронно вызывает BPEL-процесс, тогда `myRole` указывает, что BPEL-процесс предоставляет сервис клиенту.

- необязательный атрибут `initializePartnerRole` указывает необходимость инициализации связей при развертывании BPEL-процесса (возможные значения атрибута — `yes` или `no`). Если процесс сам инициализирует свои связи, например при использовании действия `assign`, тогда значение атрибута устанавливается `no`; если же связи необходимо инициализировать заранее при развертывании, тогда значение атрибута устанавливается равным `yes`.

Каждая связь характеризуется взаимоотношениями между двумя Web-сервисами с указанием ролей, выполняемых Web-сервисами, и определением интерфейсов `portType`, обеспеченных Web-сервисами для обмена сообщениями. Каждая роль связывается с `portType` с помощью элемента `<partnerLinkType>`, определенного в WSDL-описании BPEL-процесса, на который ссылается обязательный атрибут `partnerLinkType` элемента `<partnerLinks>`.

Элемент `<partnerLinkType>` включается в WSDL-описание BPEL-процесса как элемент расширения WSDL-спецификации с пространством имен `http://docs.oasis-open.org/wsbpel/2.0/plnktype`. Элемент `<partnerLinkType>` имеет следующий синтаксис:

```
<wsdl:definitions name="NCName" targetNamespace="anyURI" ...>
   ...
   <plnk:partnerLinkType name="NCName">
      <plnk:role name="NCName" portType="QName"/>
      <plnk:role name="NCName" portType="QName"/>?
   </plnk:partnerLinkType>
   ...
</wsdl:definitions>
```

Соответственно синхронным или асинхронным взаимодействиям в элементе `<partnerLinkType>` могут быть определены одна или две роли.

Таким образом, элемент `<partnerLinks>` определяет взаимоотношения с партнерами, обеспечивая при этом связь с Web-сервисами. Однако связь с Web-сервисом партнера может быть установлена и динамически, например, действием `assign` с использованием элемента `<sref:service-ref>` (пространство имен `http://docs.oasis-open.org/wsbpel/2.0/serviceref`).

Элемент `<sref:service-ref>` содержит информацию о конкретной реализации интерфейса Web-сервиса партнера и используется для динамической связи с ним. Элемент `<sref:service-ref>` имеет атрибут `reference-scheme` (XML-схема конечной точки Web-сервиса) и дочерние элементы, которые могут быть элементами WS-Addressing, описывающие реализацию Web-сервиса.

Элемент `<messageExchanges>` BPEL-описания определяет сообщения, участвующие в обмене BPEL-процесса с партнерами. Элемент `<messageExchanges>` служит контейнером для элементов `<messageExchange>`, имеющих обязательный атрибут `name` — имя сообщения.

Элемент `<variables>` представляет BPEL-переменные, предназначенные для хранения BPEL-данных. BPEL-данные могут двух типов — данные, содержащиеся в сообщениях, и данные самого BPEL-процесса. Поэтому состояние всего BPEL-процесса сохраняется с помощью BPEL-переменных.

Элемент `<variables>` имеет обязательный атрибут `name` — имя переменной. Существуют три типа BPEL-переменных — переменные WSDL-сообщений, переменные простых типов или имеющие тип, описанный XML-схемой, и переменные, определенные элементами XML-схемы. Соответственно тип BPEL-переменной указывается атрибутами `messageType`, `type` и `element` элемента `<variables>`. BPEL-переменные, объявленные как дочерние элементы `<variables>` элемента `<process>`, называются *глобальными*, а BPEL-переменные, объявленные как дочерние элементы `<variables>` элемента `<scope>`, называются *локальными*.

BPEL-переменные могут иметь свойства. Свойства BPEL-переменных определяются в WSDL-описании BPEL-процесса с помощью элемента расширения

<vprop:property> (пространство имен http://docs.oasis-open.org/wsbpel/2.0/varprop). Синтаксис элемента <vprop:property> следующий:

```
<wsdl:definitions name="NCName">
 <vprop:property name="NCName" type="QName"? element="QName"? />
   ...
</wsdl:definitions>
```

BPEL-свойство характеризуется именем и типом данных.

Для указания соответствия BPEL-свойства определенному полю сообщения или значению BPEL-переменной существует элемент WSDL-расширения <vprop:propertyAlias>. Синтаксис элемента <vprop:propertyAlias> таков:

```
<wsdl:definitions name="NCName" ...>
  <vprop:propertyAlias propertyName="QName" messageType="QName"?
    part="NCName"? type="QName"? element="QName"?>
    <vprop:query queryLanguage="anyURI"?>?
      queryContent
    </vprop:query>
  </vprop:propertyAlias>
   ...
</wsdl:definitions>
```

Для указания соответствия BPEL-свойства полю сообщения используются атрибуты messageType и part (соответствие части сообщения), а для указания соответствия BPEL-свойства значению BPEL-переменной — атрибуты type или element. Вложенный элемент <query> устанавливает соответствие значения BPEL-свойства значению определенного поля сообщения или значению определенной BPEL-переменной.

Предназначением BPEL-процесса является интеграция Web-сервисов в одном бизнес-процессе с определением порядка их вызова. Поэтому в качестве главных элементов BPEL-описание содержит объявления действий (рис. 1.10), определяющих поведение BPEL-процесса. BPEL-действия могут быть двух типов — *структурные действия* (табл. 1.5), содержащие другие действия и определяющие связи между ними, и *базовые действия* (табл. 1.6) для выполнения простых операций BPEL-процесса.

Таблица 1.5. *Структурные BPEL-действия*

BPEL-действие	Обязательные атрибуты или дочерние элементы	Описание
sequence		Содержит действия, выполняемые последовательно
if	Вложенный элемент <condition> (дополнительный атрибут expressionLanguage) — условие "если...тогда"	Определяет условия, при которых выполняются вложенные действия. Дополнительные вложенные элементы: • вложенный элемент <elseif> (вложенный элемент <condition>) — условие "иначе...если...тогда"; • вложенный элемент <else> — условие "иначе...тогда"

Таблица 1.5 *(продолжение)*

BPEL-действие	Обязательные атрибуты или дочерние элементы	Описание
while	Вложенный элемент <condition> — условие повторения действия	Определяет условие повторения выполнения вложенного действия
repeatUntil	Вложенный элемент <condition> — условие повторения действия	Определяет условие повторения выполнения вложенного действия с выполнением действия хотя бы один раз
pick	Вложенный элемент <onMessage> — ожидание получения сообщения, аналогичен действию <receive>. При получении сообщения выполняет вложенное действие	Выполняет действие при получении события, связанного с данным действием. Дополнительный атрибут createInstance — создание нового экземпляра BPEL-процесса при получении сообщения, yes или no. Дополнительный вложенный элемент <onAlarm> определяет временной сигнал, при котором выполняется вложенное действие; имеет вложенные элементы <for> (дополнительный атрибут expressionLanguage, содержит выражение, возвращающее временной интервал сигнала) или until (дополнительный атрибут expressionLanguage, содержит выражение, возвращающее временной интервал, по истечении которого генерируется сигнал)
flow		Обеспечивает параллельное и синхронизированное выполнение действий. Содержит набор параллельно выполняемых действий. Дополнительный вложенный элемент <links> обеспечивает синхронизацию действий и служит контейнером для элементов <link> (обязательный атрибут name — имя связи). Синхронизация действий обеспечивается с помощью дополнительных вложенных элементов, которые может иметь каждое действие. Дополнительные вложенные элементы действий — <targets> и <sources>. Элемент <targets> является контейнером для обязательного элемента <target> (обязательный атрибут linkName — ссылка на имя связи) и дополнительного элемента <joinCondition> (дополнительный атрибут expressionLanguage). Элемент <sources> служит контейнером для обязательного элемента <source> (обязательный атрибут linkName — ссылка на имя связи), который в свою очередь может содержать дополнительный элемент <transitionCondition> (дополнительный атрибут expressionLanguage). Действие, содержащее элемент <targets>, будет исполнено, если будет завершено действие, содержащее элемент <sources>. Выражения, содержащиеся в элементах <joinCondition> и <transitionCondition>, устанавливают статус связи — true или false. Каждое действие может иметь также дополнительные атрибуты name (имя действия) и suppressJoinFailure (см. элемент <process>)

***Таблица 1.5** (окончание)*

BPEL-действие	Обязательные атрибуты или дочерние элементы	Описание
forEach	• Атрибут counterName — имя переменной цикла. • Атрибут parallel (yes или no) — параллельное выполнение элемента scope. • Вложенный элемент <startCounterValue> (дополнительный атрибут expressionLanguage) — начальное значение переменной цикла. • Вложенный элемент <finalCounterValue> (дополнительный атрибут expressionLanguage) — конечное значение переменной цикла. • Вложенный элемент <scope> содержит выполняемые в цикле действия	Выполняет вложенный элемент <scope> *N*+1 раз, где *N* = finalCounterValue – startCounterValue. Дополнительный вложенный элемент <completionCondition> содержит условие, находящееся в его вложенном элементе <branches>, которое ограничивает число выполняемых элементов <scope>. Дополнительный атрибут successfulBranchesOnly (yes или no) элемента <branches> указывает, что подсчет ведется только в отношении успешно завершенных элементов <scope>

***Таблица 1.6.** Базовые BPEL-действия*

BPEL-действие	Обязательные атрибуты или дочерние элементы	Описание
invoke	• Атрибут partnerLink — имя партнерской связи. • Атрибут operation — вызываемая операция Web-сервиса партнера	Используется для вызова Web-сервиса партнера. Дополнительные атрибуты: portType (ссылка на элемент <portType> WSDL-описания Web-сервиса партнера); inputVariable (имя BPEL-переменной исходящего сообщения); outputVariable (имя BPEL-переменной входящего сообщения). В случае однонаправленного взаимодействия с Web-сервисом используется только переменная inputVariable, в случае двунаправленного взаимодействия "запрос — ответ" используются обе переменные. Атрибуты inputVariable и outputVariable могут быть заменены вложенными элементами <toParts> и <fromParts>
receive	• Атрибут partnerLink — имя партнерской связи. • Атрибут operation — операция Web-сервиса партнера, отвечающая за отправку сообщения BPEL-процессу	Получение сообщения от партнера. Дополнительные атрибуты: portType (ссылка на элемент <portType> WSDL-описания Web-сервиса партнера); variable (имя BPEL-переменной получаемого сообщения); createInstance (создание нового экземпляра BPEL-процесса при получении сообщения, yes или no); messageExchange (имя сообщения, объявленного в соответствующем элементе <messageExchange>). Атрибут variable может быть заменен вложенным элементом <fromParts>

Таблица 1.6 *(продолжение)*

BPEL-действие	**Обязательные атрибуты или дочерние элементы**	**Описание**
reply	• Атрибут partnerLink — имя партнерской связи. • Атрибут operation — операция Web-сервиса партнера, отвечающая за получение сообщения от BPEL-процесса	Посылает ответ после получения сообщения от партнера. Дополнительные атрибуты: portType (ссылка на элемент <portType> WSDL-описания Web-сервиса партнера); variable (имя BPEL-переменной отправляемого сообщения); messageExchange (имя сообщения, объявленного в соответствующем элементе <messageExchange>); faultName (имя ошибки, определенной в операции Web-сервиса). Атрибут variable может быть заменен вложенным элементом <toParts>
assign	Возможны варианты: • вложенный элемент <copy> — копирование значений; • вложенный элемент extensionAssignOperation содержит элементы BPEL-расширения, отвечающие за обновление данных	Обновляет значения BPEL-переменных путем копирования значения из одной переменной в другую или занесением нового значения с помощью выражений. Обновляет партнерские связи путем копирования ссылок на конечные точки Web-сервисов в partnerLinks. Дополнительный атрибут validate (yes или no) устанавливает проверку всех BPEL-переменных, измененных в результате действия, относительно их XML-определений
copy	• Вложенный элемент <from> указывает исходный элемент копирования. • Вложенный элемент <to> указывает конечный элемент копирования	Копирует значение из исходного элемента в конечный элемент. Дополнительный атрибут keepSrcElementName (yes или no) устанавливает замену имени конечного элемента на имя исходного элемента. Дополнительный атрибут ignoreMissingFromData (yes или no) устанавливает игнорирование стандартной ошибки bpel:selectionFailure
from	Возможны варианты: • атрибут variable — имя переменной сообщения (имеет дополнительный атрибут part — часть сообщения, дополнительный вложенный элемент <query> — выборка значения); • атрибуты partnerLink и endpointReference (myRole или partnerRole) — копируемая партнерская связь; • атрибуты variable и property — копирование свойства переменной;	Указывает источник копирования, который может быть BPEL-переменной сообщения, партнерской связью, BPEL-переменной с BPEL-свойством, выражением и значением

Таблица 1.6 *(продолжение)*

BPEL-действие	Обязательные атрибуты или дочерние элементы	Описание
	• вложенное выражение, возвращающее значение (имеет дополнительный атрибут `expressionLanguage`); • вложенный элемент `<literal>`, содержащий копируемое значение	
`to`	Возможны варианты: • атрибут `variable` (имеет дополнительный атрибут `part` — имя части сообщения, дополнительный вложенный элемент `<query>`) — имя BPEL-переменной сообщения, в которую производится копирование; • атрибут `partnerLink` — имя партнерской связи, в которую производится копирование; • атрибуты `variable` и `property` — BPEL-переменная с BPEL-свойством; • вложенное выражение (имеет дополнительный атрибут `expressionLanguage`)	Указывает конечный элемент копирования, который может быть BPEL-переменной сообщения, партнерской связью, BPEL-переменной с BPEL-свойством или выражением
`throw`	Атрибут `faultName` — имя ошибки	Генерирует ошибку BPEL-процесса. Дополнительный атрибут `faultVariable` указывает имя BPEL-переменной, содержащей информацию об ошибке
`wait`	Возможны варианты: • вложенный элемент `<for>` (имеет дополнительный атрибут `expressionLanguage`) содержит выражение, возвращающее временной интервал задержки; • вложенный элемент `<until>` (имеет дополнительный атрибут `expressionLanguage`) содержит выражение, возвращающее временной интервал, до истечения которого производится задержка	Определяет задержку BPEL-процесса
`empty`		Пустое действие, используется в целях обработки ошибок или синхронизации
`extensionActivity`		Действие BPEL-расширения
`exit`		Завершение экземпляра BPEL-процесса
`rethrow`		Генерация ошибки, которая была перехвачена обработчиком ошибок. Используется как вложенный элемент элементов `<catch>` и `<catchAll>`
`validate`	Атрибут `variables` — ссылка на имя переменной	Осуществляет проверку BPEL-переменной относительно XML- и WSDL-определений

***Таблица 1.6** (окончание)*

BPEL-действие	Обязательные атрибуты или дочерние элементы	Описание
`compensateScope`	Атрибут `target` — ссылка на элемент `<scope>` или `<invoke>`, содержащий компенсационный обработчик	Вызывает альтернативные действия компенсационного обработчика, содержащегося в именованном элементе `<scope>` или `<invoke>`
`compensate`		Вызывает компенсационный обработчик по умолчанию

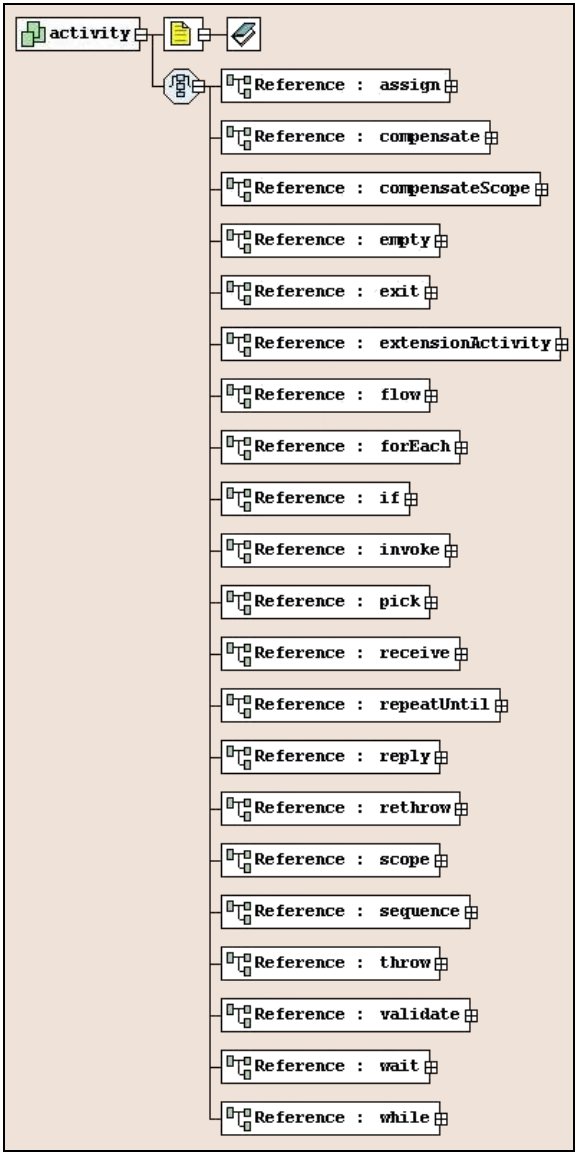


Рис. 1.10. BPEL-действия

Элементы <toParts> и <fromParts> служат контейнерами для соответствующих элементов <toPart> и <fromPart>. Элемент <toPart> отвечает за копирование данных из BPEL-переменной (обязательный атрибут fromVariable — имя переменной) в часть исходящего сообщения (обязательный атрибут part — имя части сообщения). Элемент <fromPart> отвечает за копирование данных из части сообщения (обязательный атрибут part — имя части сообщения) в BPEL-переменную (обязательный атрибут toVariable — имя переменной).

При взаимодействии BPEL-сервера с клиентами и Web-сервисами партнеров может быть создано несколько одновременно работающих экземпляров BPEL-процесса. Поэтому возникает необходимость доставки сообщений нужному экземпляру BPEL-процесса. Элемент <correlationSets> позволяет объединять взаимодействия с партнерами в диалоги. Элемент <correlationSets> может быть дочерним элементом как элемента <process>, так и элемента <scope>. Элемент <correlationSets> имеет обязательные атрибуты name (имя корреляции) и properties (набор свойств BPEL-переменных сообщений, которые идентифицируют диалог). Свойства корреляции, содержащиеся в сообщениях, должны иметь простые типы, и как раз они и обеспечивают объединение взаимодействий с партнерами в диалоги. Ссылка на корреляцию осуществляется в действиях <invoke>, <receive> и <reply>, связанных с обменом сообщениями, с помощью вложенного элемента <correlations>. Элемент <correlations> служит контейнером для элемента <correlation>, имеющего обязательный атрибут set (ссылка на имя элемента <correlationSets>) и дополнительные атрибуты initiate (yes — действие должно инициализировать корреляцию, join — действие должно инициализировать корреляцию, если она еще не инициализирована, и no — действие не инициализирует корреляцию) и pattern (request, или response, или request-response, указывает, к какому сообщению в обмене применяется корреляция).

При возникновении ошибок BPEL-процесса они могут быть перехвачены и обработаны обработчиком ошибок. Обработчик ошибок BPEL-процесса описывается элементом <faultHandlers>, который служит контейнером для обязательных элементов <catch> и дополнительного элемента <catchAll>.

Элемент <catch> содержит действия, обрабатывающие ошибку, и имеет дополнительные атрибуты faultName (имя обрабатываемой ошибки), faultVariable (имя переменной, содержащей информацию об ошибке) и faultMessageType (WSDL-тип данных ошибки) или faultElement (XML-тип данных ошибки). Элемент <catchAll> содержит действия, которые могут обрабатывать все остальные ошибки.

Обработчик ошибок может быть встроен в действие <invoke> с помощью элементов <catch> и <catchAll>. Существуют стандартные ошибки BPEL-процесса, которые описаны в табл. 1.7.

Таблица 1.7. *Стандартные BPEL-ошибки*

Ошибка	Описание
ambiguousReceive	Попытка одновременно совершить несколько действий получения сообщения с разными корреляциями для одного набора partnerLink, portType, operation

Таблица 1.7 *(окончание)*

Ошибка	Описание
completionConditionFailure	Ошибка условия завершения цикла <forEach>
conflictingReceive	Попытка одновременно совершить несколько действий получения сообщения с одной корреляцией для одного набора partnerLink, portType, operation
conflictingRequest	Попытка нескольких одновременных запросов для одного набора partnerLink, operation, messageExchange
correlationViolation	Ошибка корреляции сообщения
invalidBranchCondition	Значение условия завершения цикла <forEach> больше, чем значение для начала цикла <forEach>
invalidExpressionValue	Значение, возвращаемое выражением, не соответствует XML-типу, за исключением действия assign
invalidVariables	Значение переменной не соответствует XML-типу
joinFailure	При значении yes атрибута suppressJoinFailure элемент <joinCondition> равен false
mismatchedAssignmentFailure	Ошибка XML-типа действия assign
missingReply	Потеря выполнения действия <reply>
missingRequest	Действие <reply> не может быть связано с набором partnerLink, operation, messageExchange
scopeInitializationFailure	Ошибка инициализации объектов элемента <scope>
selectionFailure	Ошибка выражения выбора значения
subLanguageExecutionFault	Ошибка языка выражений или запросов
uninitializedPartnerRole	Отсутствие необходимой инициализации связи с партнером
uninitializedVariable	Отсутствие необходимой инициализации переменной
unsupportedReference	Ошибка интерпретации схемы для конечной точки Web-сервиса партнера
xsltInvalidSource	Ошибка источника трансформации функции bpel:doXslTransform
xsltStylesheetNotFound	Отсутствие таблицы стилей для выполнения функции bpel:doXslTransform

При возникновении каких-либо ошибок BPEL-процесса может возникнуть необходимость отменить уже выполненные действия. Для совершения альтернативных действий служит компенсационный обработчик. Компенсационный обработчик описывается элементом <compensationHandler>, содержащим альтернативные действия. Элемент <compensationHandler> может объявляться в элементе <scope> или быть прямо встроенным в действие <invoke> и вызываться действиями <compensateScope> или <compensate>, которые используются в элементах <catch>, <catchAll>, <compensationHandler> и <terminationHandler>.

В случае возникновения ошибки BPEL-процесса необходимо корректным образом завершить работающие действия перед запуском обработчика ошибок. Для этого

существует обработчик завершения, который описывается элементом `<terminationHandler>`, содержащим завершающие действия. Обработчик завершения может быть объявлен по умолчанию, как и компенсационный обработчик и обработчик ошибок, а может быть добавлен в отдельный элемент `<scope>`.

BPEL-процесс также может иметь набор обработчиков событий, вызываемых в случае возникновения соответствующего события. Обрабатываемые события могут быть двух типов: входящие WSDL-сообщения или временные сигналы. Обработчик событий описывается элементом `<eventHandlers>`.

Элемент `<eventHandlers>` служит контейнером для элементов `<onEvent>` и `<onAlarm>`. Элемент `<onEvent>` аналогичен действию `<receive>`, дополнительно включая элемент `<scope>` с обрабатывающими действиями. Элемент `<onAlarm>` может содержать элементы `<for>`, `<until>` и `<repeatEvery>`, определяющие временные интервалы, и содержит элемент `<scope>` с обрабатывающими действиями.

Как уже было сказано, целью применения элемента `<scope>` является разделение BPEL-процесса на части, каждая из которых реализует свою бизнес-логику. Элемент `<scope>` сам является действием и обеспечивает контекст для своих вложенных действий путем определения переменных, партнерских связей, обмена сообщений, корреляций и различного рода обработчиков. Элемент `<scope>` имеет следующие дополнительные атрибуты:

- `name` — идентификатор элемента;
- `suppressJoinFailure` определяет обработку ошибок условий, определенных в элементе `<joinCondition>`;
- `isolated`. Если значение атрибута `yes`, тогда действие `<scope>` обеспечивает контроль над доступом к совместно используемым с другими действиями `<scope>` ресурсами — переменным, партнерским связям и т. д., т. е. изоляция гарантирует завершение всех операций изолированного элемента `<scope>`, изменяющих ресурсы, перед доступом к ним другого элемента `<scope>`. По умолчанию значение атрибута — `no`;
- `exitOnStandardFault`. Если значение атрибута `yes`, тогда BPEL-процесс завершается при возникновении стандартной BPEL-ошибки (за исключением `joinFailure`). Если значение атрибута `no` (по умолчанию) — стандартная BPEL-ошибка обрабатывается соответствующим обработчиком ошибок.

Язык BPEL обеспечивает несколько стандартных функций для работы с данными. Это BPEL-функции `bpel:getVariableProperty` и `bpel:doXslTransform`.

Функция `object bpel:getVariableProperty(`*`string`*`, `*`string`*`)` извлекает свойства из BPEL-переменных. Первый аргумент функции — имя переменной, второй аргумент — имя свойства.

Функция `object bpel:doXslTransform(`*`string`*`, `*`node-set`*`, (`*`string`*`, `*`object`*`)*)` используется в действии `<assign>` для XSLT-преобразования содержимого BPEL-переменной из одного XML-типа в другой XML-тип. Первый аргумент функции — URI-адрес таблицы XSLT-стилей преобразования одного XML-документа в другой

XML-документ, второй аргумент — BPEL-переменная, содержимое которой трансформируется, представленная XPath-узлом элемента, и дополнительные параметры функции являются XSLT-параметрами "имя — значение".

Для работы с данными язык BPEL использует запросы и выражения, языки которых определяются атрибутами `queryLanguage` и `expressionLanguage`. По умолчанию для запросов и выражений используется язык XPath 1.0 (значение атрибутов `queryLanguage` и `expressionLanguage` — `urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`). Кроме того, т. к. язык XPath 1.0 не обеспечивает трансформацию XML-документов, язык BPEL поддерживает спецификацию XSLT 1.0 (функция `bpel:doXslTransform`). Спецификация XSLT 1.0 рассмотрена в приложении "Язык XSLT" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Абстрактное BPEL-описание

Абстрактное BPEL-описание регламентирует BPEL-процесс частично, опуская конкретные детали его выполнения или заменяя их специальной конструкцией `opaque`.

Абстрактное BPEL-описание может использоваться для описания BPEL-процесса с разных точек зрения, как первоначальный этап в разработке BPEL-процесса, для определения самой возможности взаимодействия между бизнес-партнерами и т. д. Поэтому каждое абстрактное BPEL-описание должно указывать свой профиль использования, определяющий предназначение абстрактного BPEL-описания.

Синтаксически абстрактное BPEL-описание отличается от исполняемого BPEL-описания:

- значением `http://docs.oasis-open.org/wsbpel/2.0/process/abstract` атрибута `xmlns` элемента `<process>`, указывающим пространство имен BPEL-документа;
- дополнительным атрибутом `abstractProcessProfile` элемента `<process>`, указывающим URI-идентификатор профиля использования абстрактного BPEL-описания;
- наличием конструкций `opaque` в описании BPEL-процесса.

Конструкция `opaque` может скрывать конкретные детали BPEL-процесса, замещая следующие элементы BPEL-описания:

- выражения — выражения, содержащиеся в элементах `<transitionCondition>`, `<joinCondition>`, `<condition>`, `<until>`, `<for>`, `<repeatEvery>`, `<startCounterValue>`, `<finalCounterValue>`, `<branches>`, `<from>`, `<to>` заменяются атрибутом соответствующего элемента `opaque="yes"`;
- действия — элемент действия замещается элементом `<opaqueActivity>`;
- значения атрибутов заменяются `##opaque`;
- элемент `<from>` заменяется `<opaqueFrom/>`.

При этом конкретный профиль использования абстрактного BPEL-описания определяет, какие именно элементы BPEL-описания замещаются конструкцией `opaque`.

WS-CDL 1.0

Web Services Choreography Description Language (WS-CDL) — основанный на XML язык описания взаимодействий бизнес-партнеров, определяющий порядок обмена между ними сообщений, результатом которого является решение общей для бизнес-партнеров задачи.

Язык CDL не является исполняемым языком описания бизнес-процесса, он позволяет лишь создать общую модель взаимодействия бизнес-партнеров, которая должна затем быть реализована с помощью исполняемого языка, например BPEL, или непосредственно с использованием какого-либо из языков программирования.

Язык CDL поддерживается различными инструментами разработки — WS-CDL Eclipse, Pi4SOA CDL Editor, LTSA WS-Engineer, позволяющими моделировать и анализировать CDL-документы.

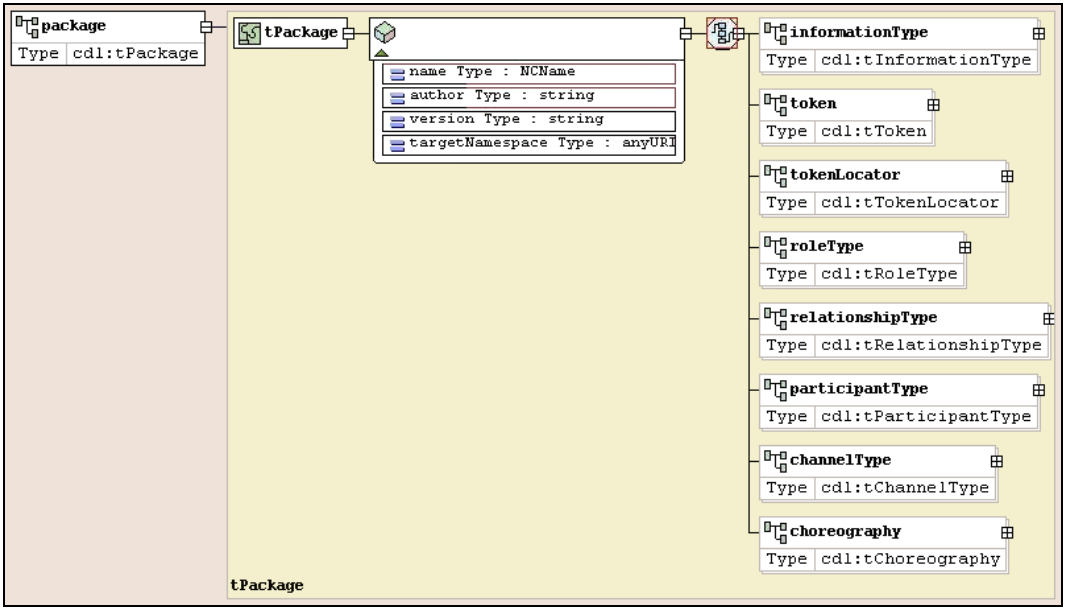


Рис. 1.11. Общая схема элемента `<package>`

Корневым элементом CDL-документа служит элемент `<package>` (рис. 1.11), имеющий следующие атрибуты:

- `name` — имя документа;
- `author` — автор документа;
- `version` — версия документа;
- `targetNamespace` — пространство имен CDL-документа;
- `xmlns` — `http://www.w3.org/2005/10/cdl` — пространство имен языка CDL.

Элемент `<roleType>` описывает роли участников бизнес-процесса. Элемент `<roleType>` имеет обязательный атрибут `name` (имя роли) и вложенные элементы

<behavior>, указывающие поведение (операцию) бизнес-партнера с помощью обязательного атрибута name (имя операции) и дополнительного атрибута interface (ссылка на WSDL-интерфейс бизнес-партнера). Если атрибут interface отсутствует, тогда бизнес-партнер не поддерживает определенный интерфейс Web-сервиса.

Элемент <relationshipType> описывает взаимоотношения между двумя бизнес-партнерами согласно их ролям и поведениям. Элемент <relationshipType> имеет обязательный атрибут name (имя связи) и два вложенных элемента <roleType>, указывающие роль партнера в связи. Элемент <roleType> имеет обязательный атрибут typeRef (ссылка на имя элемента <roleType>) и дополнительный атрибут behavior, указывающий перечень операций роли бизнес-партнера. Если атрибут behavior отсутствует, тогда в связи участвуют все операции бизнес-партнера.

Элемент <participantType> группирует вместе роли одного бизнес-партнера, определяя его общее участие в бизнес-процессе как сервис. Участие бизнес-партнера в бизнес-процессе определяет его как участника бизнес-процесса. Элемент <participantType> имеет обязательный атрибут name (имя участника бизнес-процесса (сервиса)) и вложенные элементы <roleType> с обязательным атрибутом typeRef (ссылка на имя роли).

Элемент <channelType> дает реализацию взаимодействия между участниками бизнес-процесса, описывая канал обмена сообщениями, связанный с определенной ролью. Дополнительно информация о канале может передаваться в виде сообщения от одного Web-сервиса другому, обеспечивая динамический поиск и коммуникацию между многочисленными участниками бизнес-процесса, вовлеченными в сложный обмен сообщениями.

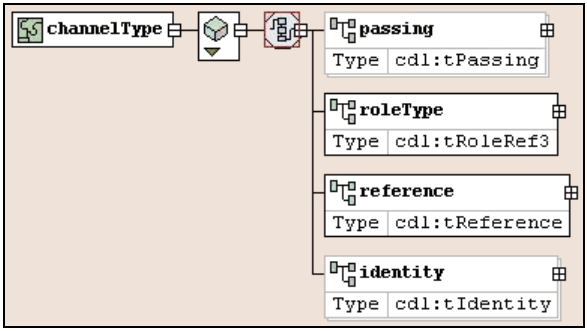


Рис. 1.12. Общая схема элемента <channelType>

Элемент <channelType> (рис. 1.12) имеет следующие атрибуты:

- обязательный атрибут name — имя канала взаимодействия;
- дополнительный атрибут usage — использование канала. Возможные значения:
  - once — данный канал используется для разового взаимодействия, или информация о канале пересылается другой роли;
  - distinct — по умолчанию, многократное использование канала одним участником для многочисленных взаимодействий. При этом отсутствует возмож-

ность параллельных взаимодействий для одного и того же экземпляра канала. Кроме того, роль, с которой связан данный канал, может послать информацию о канале только одной другой роли;

- shared — многократное использование канала различными участниками для многочисленных взаимодействий; при этом нет ограничений на рассылку информации о канале для роли, с которой данный канал связан. Для данного режима также отсутствует возможность параллельных взаимодействий для одного и того же экземпляра канала;

- дополнительный атрибут action — тип обмена сообщениями для данного канала, возможные значения: request-respond, request (по умолчанию) и respond.

Дополнительный вложенный элемент <passing> элемента <channelType> описывает рассылку информации о канале другим участникам бизнес-процесса. Если элемент <passing> отсутствует, тогда данный канал не может использоваться для рассылки информации о нем. Элемент <passing> имеет следующие атрибуты:

- обязательный атрибут channel — имя канала, для которого осуществляется рассылка информации;
- дополнительный атрибут action указывает, когда информация о канале должна быть послана. Возможные значения атрибута — request-respond, request (по умолчанию) и respond;
- дополнительный атрибут new. Если значение атрибута true, тогда всегда посылается информация только о новом экземпляре канала.

Обязательный вложенный элемент <roleType> элемента <channelType> связывает канал с определенной ролью с помощью обязательного атрибута typeRef (имя роли) и дополнительного атрибута behavior (специфическое поведение роли для канала).

Обязательный вложенный элемент <reference> элемента <channelType> указывает ссылку на Web-сервис, представленный участником бизнес-процесса, с помощью вложенного элемента <token>, обязательный атрибут name которого содержит имя соответствующего элемента <token>, определяющего ссылку на Web-сервис.

Дополнительные вложенные элементы <identity> элемента <channelType> используются для идентификации экземпляров канала и соответственно корреляции сообщений, связанных с экземплярами канала. Элемент <identity> содержит обязательный вложенный элемент <token>, его обязательный атрибут name содержит имя соответствующего элемента <token>, определяющего данный идентификатор. Элемент <identity> имеет обязательный атрибут usage, содержащий следующие возможные значения:

- primary — первичный ключ для экземпляра канала создается первым сообщением, связанным с данным экземпляром канала. Далее все сообщения данного экземпляра канала содержат поле с его первичным ключом;
- alternate — в процессе обмена сообщениями, какое-либо из сообщений может инициировать создание альтернативного идентификатора для данного экземпляра канала. Далее все сообщения данного экземпляра канала содержат поле с его

первичным ключом и альтернативным идентификатором, что дает возможность сообщениям коррелировать на основе первичного ключа или альтернативного идентификатора;

- `derived` — в процессе обмена сообщениями, какое-либо из сообщений может содержать поле с данным дополнительным идентификатором. Впоследствии другой экземпляр этого или другого канала может ссылаться на этот дополнительный идентификатор как на свой первичный ключ, что дает корреляцию между двумя экземплярами;
- `association` — этот идентификатор обеспечивает корреляцию данного экземпляра канала с предшествующим экземпляром этого или другого канала. Этот идентификатор связан с каким-либо идентификатором предшествующего экземпляра и содержится в соответствующем поле всех сообщений текущего экземпляра, включая первое сообщение.

Элемент `<informationType>` определяет тип информации, используемый в хореографии бизнес-процесса. Элемент `<informationType>` содержит обязательный атрибут `name` (идентификатор типа) и дополнительный атрибут `type` или `element` (описывает тип информации, соответствующий типу или элементу WSDL-описания или XML-схемы).

Элемент `<token>` содержит часть используемых данных определенного типа. Элемент `<token>` имеет обязательные атрибуты `name` (имя элемента) и `informationType` (тип данных элемента).

Элемент `<tokenLocator>` определяет расположение части данных, содержащихся в элементе `<token>`, в сообщениях бизнес-процесса. Элемент `<tokenLocator>` имеет обязательные атрибуты `tokenName` (имя элемента `<token>`), `informationType` (тип документа, в котором локализуется `<token>`) и `query` (XPath-выражение, локализующее `<token>`), а также дополнительные атрибуты `name` (имя элемента) и `part` (имя части сообщения для локализации `<token>`).

Элемент `<choreography>` (рис. 1.13) определяет порядок обмена сообщениями и условия взаимодействия участников бизнес-процесса. Элементы `<choreography>` могут иметь вложенные элементы `<choreography>`, разделяющие корневой элемент на выполняемые фрагменты. Элемент `<choreography>` имеет обязательный атрибут `name` — имя хореографии и дополнительные атрибуты:

- `complete` — XPath-выражение условия завершения хореографии;
- `isolation` — для вложенной хореографии определяет видимость ее переменных для родительской хореографии, по умолчанию `false` — все переменные глобальные;
- `root` указывает, что хореография является корневой, по умолчанию `false`;
- `coordination`. Если `true`, тогда протокол координации хореографии гарантирует, что все роли согласованы в отношении способа завершения хореографии.

Элемент `<choreography>` содержит вложенные элементы:

- `<relationship>`, которые перечисляют связи, участвующие в хореографии, используя обязательный атрибут `type` — имя связи;

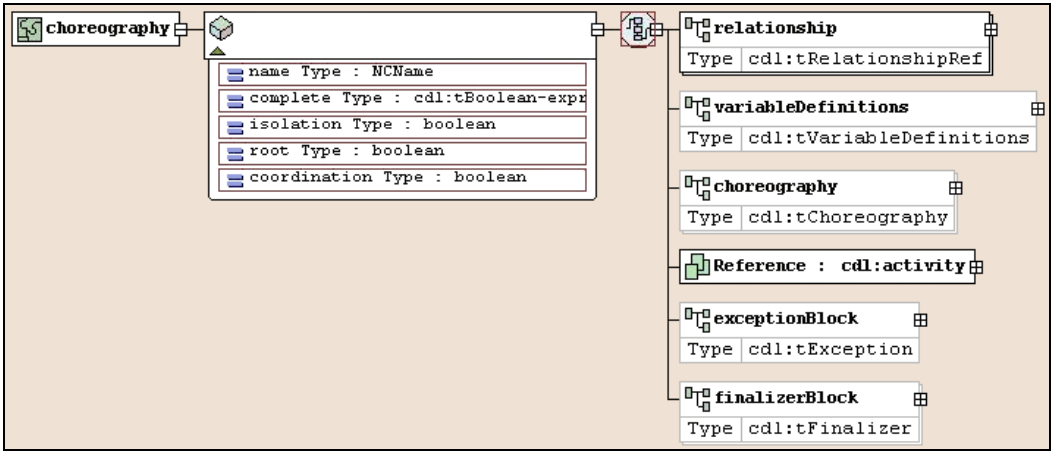


Рис. 1.13. Общая схема элемента `<choreography>`

- `<variableDefinitions>` определяет переменные хореографии;
- элементы действий — `<sequence>`, `<parallel>`, `<choice>`, `<workunit>`, `<interaction>`, `<perform>`, `<assign>`, `<silentAction>`, `<noAction>` и `<finalize>` (рис. 1.14);
- `<exceptionBlock>` управляет исключительными ситуациями;
- `<finalizerBlock>` управляет завершением хореографии после ее успешного выполнения.

Элемент `<variableDefinitions>` содержит вложенные элементы `<variable>`, определяющие переменные хореографии бизнес-процесса. Переменная хореографии может быть следующего типа:

- переменная, представляющая содержимое сообщения, участвующего в обмене между участниками бизнес-процесса;
- переменная состояния взаимодействия;
- переменная, содержащая информацию о канале взаимодействия;
- переменная исключения (ошибки).

Элемент `<variable>` содержит следующие атрибуты:

- обязательный атрибут `name` — имя переменной;
- дополнительный атрибут `informationType` — ссылка на имя элемента `<informationType>` или дополнительный атрибут `channelType` — ссылка на имя элемента `<channelType>`;
- дополнительный атрибут `mutable` — возможное значение `true` (по умолчанию значение переменной после инициализации может изменяться) или `false` (значение переменной после инициализации не может изменяться);
- дополнительный атрибут `free` — возможное значение `true` (область видимости переменной распространяется на вложенные элементы `<choreography>`) или `false` (по умолчанию переменная определена только для данного элемента `<choreography>`);

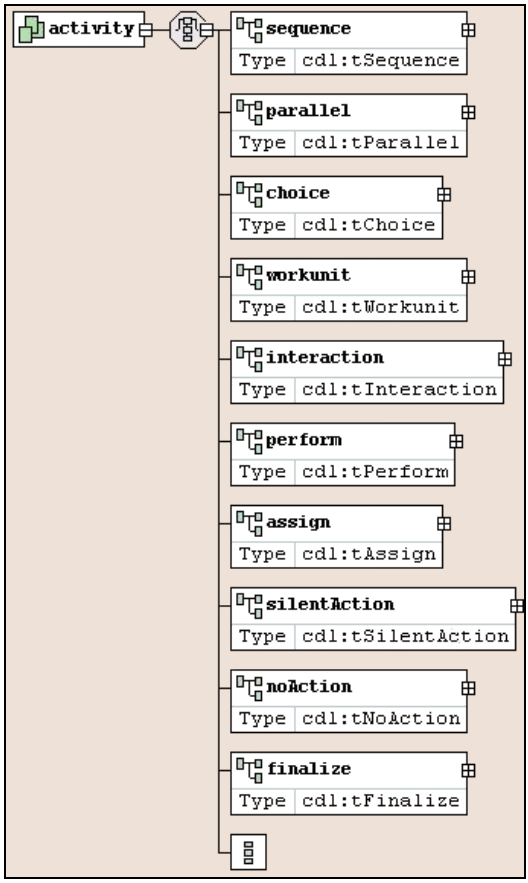


Рис. 1.14. CDL-действия

- дополнительный атрибут `silent` — возможное значение `true` (переменная является внутренней для участника) или `false` (по умолчанию);
- дополнительный атрибут `roleTypes` — перечень ролей, с которыми данная переменная связана.

Для определения различного рода условий и работы с переменными язык CDL использует язык XPath 1.0. В дополнение к функциям языка XPath 1.0, язык CDL определяет свои функции (пространство имен `http://www.w3.org/2005/10/cdl`).

- `xsd:time getCurrentTime(xsd:QName roleTypeName?)`

  Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает текущее время для роли. Если параметр отсутствует, роль берется из контекста.

- `xsd:date getCurrentDate(xsd:QName roleTypeName?)`

  Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает текущую дату для роли. Если параметр отсутствует, роль берется из контекста.

- `xsd:dateTime getCurrentDateTime(xsd:QName roleTypeName?)`

  Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает текущую дату и время для роли. Если параметр отсутствует, роль берется из контекста.

- `xsd:Boolean hasDurationPassed(xsd:duration elapsedTime, xsd:QName roleTypeName?)`

  Обязательный параметр `elapsedTime` — период времени. Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает `true`, когда для роли истекает временной промежуток с начала установки условия. Если параметр `roleTypeName` отсутствует, роль берется из контекста.

- `xsd:Boolean hasDeadlinePassed(xsd:dateTime deadlineTime, xsd:QName roleTypeName?)`

  Обязательный параметр `deadlineTime` — дата и время. Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает `true`, когда для роли наступает дата и время, давая старт условию. Если параметр `roleTypeName` отсутствует, роль берется из контекста.

- `xsd:any getVariable(xsd:string varName, xsd:string part, xsd:string documentPath, xsd:QName roleTypeName?)`

  Обязательные параметры: `varName` — имя переменной; `part` — имя части сообщения; `documentPath` — фрагмент документа.

  Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает информацию переменной. Если параметр `roleTypeName` отсутствует, роль берется из контекста.

- `xsd:Boolean isVariableAvailable(xsd:string varName, xsd:QName roleTypeName?)`

  Обязательный параметр `varName` — имя переменной. Дополнительный параметр `roleTypeName` — имя роли.

  Возвращает `true`, если информация переменной существует. Если параметр `roleTypeName` отсутствует, роль берется из контекста.

- `xsd:boolean variablesAligned(xsd:string varName, xsd:string withVarName, xsd:QName relationshipTypeName)`

  Обязательные параметры: `varName` — имя переменной первой роли; `withvarName` — имя переменной второй роли; `relationshipTypeName` — имя связи двух ролей.

  Возвращает `true`, если переменная первой роли в связи содержит ту же информацию, что и переменная второй роли связи, т. е. участники взаимоотношения имеют общую информацию.

- `xsd:any getChannelReference(xsd:string varName)`

  Обязательный параметр `varName` — имя переменной.

Возвращает информацию вложенного элемента `<reference>` элемента `<channelType>`.

- `xsd:any getChannelIdentity(xsd:string varName)`

  Обязательный параметр `varName` — имя переменной.

  Возвращает информацию об идентификации экземпляров канала

- `xsd:Boolean globalizedTrigger(xsd:string expression1, xsd:string roleTypeName1, xsd:string expression2, xsd:string roleTypeName2, ...)`

  Обязательные параметры: `expression` — выражение; `roleTypeName` — имя роли.

  Возвращает `true`, если каждое выражение, определенное для своей роли, возвращает `true`.

- `xsd:boolean hasExceptionOccurred(xsd:QName exceptionType)`

  Обязательный параметр `exceptionType` — тип ошибки.

  Возвращает `true`, если возникает указанная исключительная ситуация.

- `xsd:boolean hasChoreographyCompleted(xsd:string choreoName, xsd:string choreoInstanceId?)`

  Обязательный параметр `choreoName` — имя хореографии. Дополнительный параметр `choreoInstanceId` — идентификатор экземпляра хореографии.

  Возвращает `true`, если выполнение указанной хореографии завершается.

- `xsd:string getChoreographyStatus(xsd:string choreoName, xsd:string choreoInstanceId?`

  Обязательный параметр `choreoName` — имя хореографии. Дополнительный параметр `choreoInstanceId` — идентификатор экземпляра хореографии.

  Возвращает текущий статус выполнения указанной хореографии.

Элемент `<interaction>` (рис. 1.15) представляет базовое CDL-действие, обеспечивающее информационный обмен между участниками бизнес-процесса в виде обмена сообщениями между ролями через определенный канал. Элемент `<interaction>` имеет следующие атрибуты:

- обязательный атрибут `name` — имя взаимодействия;
- обязательный атрибут `channelVariable` — имя переменной, содержащей информацию о канале обмена сообщениями;
- обязательный атрибут `operation` — имя операции канала;
- дополнительный атрибут `align`. Если `true`, тогда взаимодействие завершается успешно только в том случае, если все обмены информацией завершены успешно. По умолчанию — `false`.

Вложенный элемент `<participate>` элемента `<interaction>` определяет участников взаимодействия с помощью атрибутов `relationshipType` (имя связи ролей), `fromRoleTypeRef` (имя запрашивающей роли), `toRoleTypeRef` (роль канала взаимодействия — имя элемента `<roleType>`, определенного в элементе `<channelType>`).

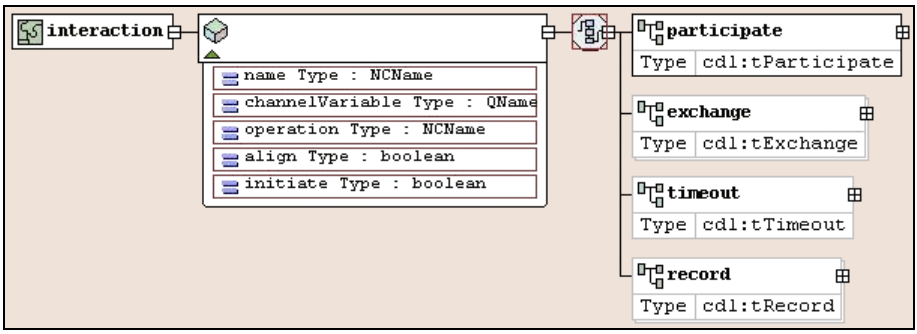


Рис. 1.15. Общая схема элемента `<interaction>`

Вложенный элемент `<exchange>` элемента `<interaction>` описывает информацию обмена, используя обязательные атрибуты `name` (имя элемента) и `action` (`request` или `respond`, указывает направление информационного обмена), а также дополнительных атрибутов `faultName` (определяет элемент как информацию об ошибке, указывая имя ошибки) и `informationType` или `channelType` — тип информации.

Вложенные элементы `<send>` и `<receive>` элемента `<exchange>` описывают информацию, отправляемую и получаемую ролями. Элементы `<send>` и `<receive>` имеют необязательные атрибуты `variable` (информация переменной), `recordReference` (перечень ссылок на элементы `<record>`) и `causeException` (указывает исключение, которое должно быть вызвано).

Вложенный элемент `<timeout>` элемента `<interaction>` указывает время завершения взаимодействия, используя обязательный атрибут `time-to-complete` (время завершения или период времени, в течение которого взаимодействие должно завершиться), а также дополнительные атрибуты `fromRoleTypeRecordRef` (перечень элементов `<record>`, которые должны быть завершены при запросе) и `toRoleTypeRecordRef` (перечень элементов `<record>`, которые должны быть завершены при ответе).

Вложенный элемент `<record>` элемента `<interaction>` используется для создания или изменения информации переменных. Элемент `<record>` имеет обязательные атрибуты `name` (имя записи) и `when` (`before` — запись перед отправлением-получением, `after` — запись после отправления-получения, `timeout` — запись по истечении времени ожидания), а также дополнительный атрибут `causeException` (указывает исключение, которое может быть вызвано).

Элемент `<record>` содержит вложенные элементы `<source>` (запись при отправке информации, атрибут `variable` (информация переменной) или `expression` (XPath-выражение информации переменной)) и `<target>` (запись при получении информации, атрибут `variable` — информация переменной).

Элемент `<perform>` представляет базовое CDL-действие, позволяющее выполнять другую хореографию в текущей. Элемент `<perform>` имеет обязательный атрибут `choreographyName` (имя выполняемой хореографии), а также дополнительные атрибуты `choreographyInstanceId` (XPath-выражение, дающее идентификатор экземпляра хореографии) и `block` (если `true` — по умолчанию, тогда вызываемая хореогра-

фия блокирует выполнение текущей до тех пор, пока та не будет выполнена). Элемент <perform> содержит вложенные элементы <bind> (обязательный атрибут name — имя связи), дающие возможность сделать общей информацию для вызываемой и текущей хореографий с помощью вложенных элементов <this> и <free> (обязательные атрибуты variable и roleType).

Элемент <assign> представляет базовое CDL-действие, обеспечивающее копирование информации из указанного источника в переменную. Элемент <assign> имеет обязательный атрибут roleType (имя роли, с которой связана изменяемая переменная) и вложенные элементы <copy> — действие копирования. Элемент <copy> содержит, в свою очередь, вложенные элементы <source> (источник копирования, обязательный атрибут variable или expression) и <target> (цель копирования, обязательный атрибут variable).

Элемент <silentAction> представляет базовое CDL-действие, указывающее выполнение участником своих внутренних действий, не видимых другим участникам. Элемент <silentAction> имеет необязательные атрибуты name (имя действия) и roleType (роль участника).

Элемент <noAction> представляет базовое CDL-действие, указывающее, что участник не должен выполнять никаких действий. Элемент <noAction> имеет необязательный атрибут roleType (роль участника).

Элемент <finalize> представляет базовое CDL-действие, обеспечивающее завершающие действия для вложенных хореографий. Элемент <finalize> имеет обязательный атрибут choreographyName (имя хореографии, для которой определяется завершение) и дополнительные атрибуты name (имя элемента), choreographyInstanceId (идентификатор экземпляра хореографии) и finalizerName (имя элемента <finalizerBlock>, содержащего завершающие действия).

Элемент <workunit> определяет условия выполнения для содержащихся в нем CDL-действий. Элемент <workunit> имеет обязательный атрибут name (имя элемента) и дополнительные атрибуты guard (условие выполнения элемента — XPath-выражение, возвращающее true или false), repeat (условие повторного выполнения элемента — XPath-выражение, возвращающее true или false) и block (по умолчанию false; если true, тогда выполняемый элемент <workunit> должен ожидать доступности переменных в условии guard и его значения true).

Элементы <sequence>, <parallel> и <choice> представляют структурированные действия, которые объединяют базовые CDL-действия, включая <workunit>, для их выполнения последовательно, параллельно и с выбором. Элемент <choice> определяет, что только одно из всех, включенных в него, действий должно быть выполнено. Если успешно выполняется одно из всех действий, другие действия элемента <choice> не выполняются.

Элемент <exceptionBlock> имеет обязательный атрибут name (имя элемента) и содержит вложенные элементы <workunit>, управляющие исключительными ситуациями.

Элемент `<finalizerBlock>` имеет обязательный атрибут `name` (имя элемента) и содержит вложенные элементы CDL-действий, завершающих хореографии.

В CDL-описание можно включать повторно используемые фрагменты CDL-описания с помощью элемента `<include>` (пространство имен `http://www.w3.org/2001/XMLSchema-instance`) посредством обязательного атрибута `href` — URI-адрес документа. Также возможно включать в CDL-описание элементы CDL-расширения с обязательным указанием их пространства имен.

ГЛАВА 2

Расширения технологии Web-сервисов

Основой технологии Web-сервисов является набор спецификаций XML, SOAP и WSDL. Расширения технологии Web-сервисов представлены спецификациями следующего уровня, обеспечивающими безопасность, надежность, а также поддержку транзакций Web-сервисов.

Спецификации, расширяющие основной набор, описывают такие технологии, как WS-Policy и WS-PolicyAttachment, WS-Security и WS-SecurityPolicy, WS-MetadataExchange, WS-ReliableMessaging и WS-ReliableMessaging Policy, WS-MakeConnection, WS-AtomicTransaction, WS-Coordination, WS-Trust и др. (табл. 2.1).

Таблица 2.1. *Спецификации второго уровня технологии Web-сервисов*

Спецификация	**Описание**
WS-Policy	Описывает синтаксис выражения политик Web-сервисов
WS-PolicyAttachment	Определяет механизм связывания политик с элементами описания Web-сервиса
WS-PolicyAssertions	Определяет стандартные утверждения политики Web-сервиса
WS-Addressing	Определяет механизм адресации Web-сервисов и сообщений Web-сервисов
WS-Security	Обеспечивает целостность и конфиденциальность сообщений Web-сервисов с помощью стандартного набора SOAP-расширений
WS-Trust	Определяет механизм получения маркеров защиты для их дальнейшего использования
WS-SecureConversation	Обеспечивает создание безопасной сессии обмена сообщениями
WS-SecurityPolicy	Определяет набор утверждений политики безопасности Web-сервиса
WS-Federation	Обеспечивает объединение различных защищенных доменов
WS-Transfer	Определяет механизм получения, обновления, удаления и создания ресурсов
WS-ResourceTransfer	Обеспечивает частичный доступ к ресурсам
WS-Fragment	Заменяет спецификацию WS-ResourceTransfer

Таблица 2.1 *(окончание)*

Спецификация	Описание
WS-MetadataExchange	Определяет механизм получения метаданных Web-сервиса
WS-Enumeration	Определяет механизм получения данных большого размера
WS-Eventing	Обеспечивает получение уведомлений о событиях Web-сервисов
WS-Management	Определяет общий SOAP-протокол для управления различными системами
WS-Discovery	Определяет механизм публикации и поиска Web-сервисов
WS-ReliableMessaging	Обеспечивает надежность передачи сообщений между Web-сервисами
WS-ReliableMessaging Policy	Определяет утверждения политики надежной передачи сообщений Web-сервиса
WS-MakeConnection	Обеспечивает обмен сообщениями между двумя Web-сервисами, один из которых не имеет доступного адреса
WS-Coordination	Определяет общий механизм координации взаимодействий между Web-сервисами
WS-AtomicTransaction	Обеспечивает поддержку отдельных транзакций Web-сервисов
WS-BusinessActivity	Определяет механизм координации бизнес-действий

Все перечисленные технологии второго уровня и будут обсуждаться в этой главе.

WS-Policy, WS-PolicyAttachment и WS-PolicyAssertions

Взаимодействие с Web-сервисом может включать в себя дополнительные возможности, требования и характеристики, относящиеся к безопасности, надежности и поддержке транзакций. Такие дополнительные свойства Web-сервиса называются его *политиками*.

Политика Web-сервиса представляется в виде XML-документа, который может быть и отдельным документом, и встроенным в такие документы описания Web-сервиса, как WSDL- и UDDI-документы. Описания WSDL и UDDI могут непосредственно включать в себя политики Web-сервиса или ссылаться на отдельные документы политик.

Чтобы клиент Web-сервиса смог получить информацию о его политиках, необходим стандартный механизм объявления политик и связывания их с XML-, WSDL- и UDDI-элементами описаний Web-сервисов. Спецификация WS-Policy определяет расширяемый синтаксис выражения политик, а спецификация WS-PolicyAttachment описывает механизм связывания политик с элементами описания Web-сервиса.

Спецификация Web Services Policy 1.2 Framework (WS-Policy) 2006 г. (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/policy`) определяет следующие XML-элементы:

- `<Policy>` (рис. 2.1) — корневой элемент политики. Этот элемент может иметь следующие дополнительные атрибуты:
  - `Name` — URI-идентификатор политики, используется механизмом WS-PolicyAttachment;
  - `wsu:Id` — идентификатор политики, при ее включении в другой XML-документ (пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`);
- `<ExactlyOne>` указывает, что должен быть выполнен один из дочерних элементов элемента `<ExactlyOne>`;
- `<All>` указывает, что должны быть выполнены все дочерние элементы;
- `<PolicyReference>` позволяет включить содержимое другой политики в текущую политику с помощью атрибута `URI`, который указывает идентификатор включаемой политики. Элемент `<PolicyReference>` может иметь также дополнительные атрибуты `Digest` (`base64Binary`-значение цифровой подписи политики) и `DigestAlgorithm` (URI-идентификатор алгоритма цифровой подписи, по умолчанию — `http://schemas.xmlsoap.org/ws/2004/09/policy/Sha1Exc`).

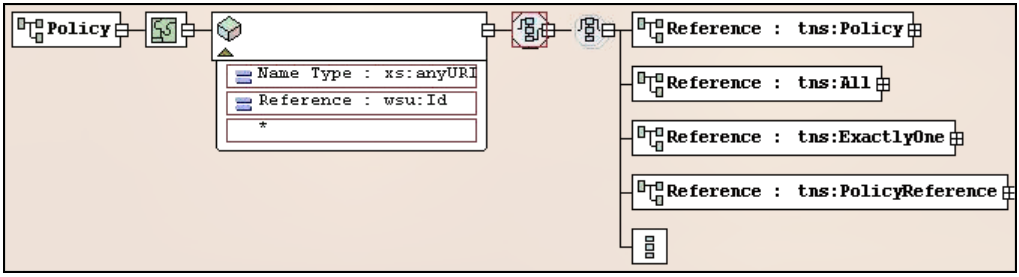


Рис. 2.1. Общая схема элемента `<Policy>` WS-Policy 1.2 2006 г.

Дочерние элементы элементов `<ExactlyOne>` и `<All>` являются XML-элементами, имеющими свое пространство имен и представляющими утверждения, которые описывают дополнительные свойства Web-сервиса. Утверждения могут содержать дополнительный атрибут `Optional` (значение `true` или `false`), указывающий, что данное утверждение является необязательным. Кроме того, любое утверждение также может содержать вложенную политику.

Спецификация Web Services Policy Framework (WS-Policy) 2004 г. для элемента `<Policy>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/policy`) определяет атрибуты `xml:base` (базовый URI-идентификатор политики), `wsu:Id` (относительный URI-идентификатор политики, который вместе с `xml:base` формирует полный URI-идентификатор политики) и `TargetNamespace` (пространство имен утверждений политики).

Спецификация Web Services Policy 1.5 (пространство имен `http://www.w3.org/ns/ws-policy`) определяет следующие XML-элементы:

- `<Policy>` — корневой элемент политики, который может иметь дополнительные атрибуты:
  - `Name` — IRI-идентификатор политики, используется механизмом WS-Policy-Attachment;
  - идентификатор политики при ее включении в другой XML-документ. Идентификатор может быть представлен атрибутом `wsu:Id` (пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`) или атрибутом `xml:id` (пространство имен `http://www.w3.org/XML/1998/namespace`);
- `<ExactlyOne>` указывает, что должен быть выполнен один из дочерних элементов элемента `<ExactlyOne>`;
- `<All>` указывает, что должны быть выполнены все дочерние элементы;
- `<PolicyReference>` позволяет включить содержимое другой политики в текущую политику с помощью атрибута `URI`, представляющего идентификатор включаемой политики. Элемент `<PolicyReference>` может иметь также дополнительные атрибуты `Digest` (`base64Binary`-значение цифровой подписи политики) и `DigestAlgorithm` (URI-идентификатор алгоритма цифровой подписи, по умолчанию — `http://www.w3.org/ns/ws-policy/Sha1Exc`).

Дочерние элементы элементов `<ExactlyOne>` и `<wsp:All>` являются XML-элементами, имеющими свое пространство имен и представляющими утверждения, которые описывают дополнительные свойства Web-сервиса. Утверждения могут содержать дополнительные атрибуты `Optional` (значение `true` или `false`; указывает, что данное утверждение является необязательным) и `Ignorable` (значение `true` или `false`; указывает, что утверждение может игнорироваться при конфликте совместимости с другими утверждениями). Также любое утверждение может содержать вложенные политики.

Документ, содержащий политику Web-сервиса, может иметь MIME-тип `application/wspolicy+xml` с файловым расширением wspolicy.

Спецификация WS-PolicyAttachment версий 1.2 и 1.5 определяет механизм связывания политик соответственно спецификаций WS-Policy версий 1.2 и 1.5 с элементами XML-, WSDL- и UDDI-описаний Web-сервисов. Спецификация WS-PolicyAttachment версии 1.2 поддерживает спецификацию WSDL 1.1, а спецификация WS-PolicyAttachment версии 1.5 — спецификации WSDL 1.1 и WSDL 2.0.

Механизм связывания политик с элементами описаний Web-сервисов может быть двух типов. Первый тип связывания — это включение политик непосредственно в описание Web-сервиса. Второй тип связывания — это использование ссылок на политики, которые определены независимо в виде отдельных XML-документов.

Включение непосредственно политик в описания Web-сервиса может производиться с помощью элементов `<Policy>` и `<PolicyReference>`. Ссылаться на политики позволяет атрибут `PolicyURIs` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/policy` или `http://www.w3.org/ns/ws-policy`), содержащий URI- или IRI-

адреса политик. Атрибут `PolicyURIs` присутствует непосредственно в элементах описания Web-сервиса.

Кроме того, для связывания политики с описаниями Web-сервиса независимо от их определений существует элемент `<PolicyAttachment>`, включаемый в конфигурационный файл Web-сервиса. Элемент `<PolicyAttachment>` имеет вложенный элемент `<AppliesTo>`, указывающий область применения политики (конечная точка, операция, сообщение или ресурс). Также элемент `<PolicyAttachment>` содержит вложенный элемент `<Policy>` или `<PolicyReference>`, определяющий утверждения политики, применяемые к указанной области.

Спецификация WS-PolicyAttachment 1.5 дополнительно определяет элемент `<URI>`, как дочерний элемент элемента `<AppliesTo>`, для указания IRI-идентификатора ресурса, к которому применяется политика.

При непосредственном включении политик в WSDL 1.1 описание Web-сервиса элемент `<Policy>` рекомендовано использовать как дочерний элемент элемента `<definition>`, а для WSDL 2.0 описания — как дочерний элемент элемента `<description>`. Элемент `<PolicyReference>` используется в качестве дочернего элемента WSDL-элементов для их связывания с определенными политиками, при этом элемент `<PolicyReference>` имеет WSDL-атрибут `wsdl11:required` или `wsdl20:required` со значением `true`, гарантирующим обязательность обработки связанной политики.

Для WSDL 1.1 описания Web-сервиса элемент `<PolicyReference>` не используется как дочерний элемент элемента `<wsdl:portType>` и его дочерних элементов `<wsdl:operation>`, `<wsdl:input>`, `<wsdl:output>` и `<wsdl:fault>`, т. к. для них не допускаются элементы расширения. Поэтому для данных элементов применяется атрибут `PolicyURIs`. Для WSDL 2.0 описания Web-сервиса таких ограничений нет.

Для UDDI-описания политики связываются с UDDI-элементами `<businessEntity>`, `<businessService>`, `<bindingTemplate>` и `<tModel>`. Для связывания удаленной политики с UDDI-описанием используется атрибут `tModel` с предопределенным значением.

Спецификация WS-PolicyAssertions 1.1 основывается на спецификациях WS-Policy 1.1 и WS-PolicyAttachment 1.1 и определяет ряд стандартных утверждений политики Web-сервиса.

В спецификации WS-Policy 1.1 элемент `<Policy>` (пространство имен `http://schemas.xmlsoap.org/ws/2002/12/policy`) имеет дополнительный атрибут `TargetNamespace` — URI-идентификатор пространства имен, который в паре с атрибутом `Name` образует QName-имя политики. Утверждения элемента `<Policy>` спецификации WS-Policy 1.1 могут иметь атрибуты `Usage` и `Preference`, а не `Optional` и `Ignorable`, как в спецификации WS-Policy 1.5. Атрибут `Usage` определяет обработку утверждения с помощью следующих возможных значений:

- `Required` — утверждение должно быть обработано;
- `Rejected` — утверждение не подлежит обработке;
- `Optional` — обработка утверждения не обязательна;

- `Observed` — утверждение применяется ко всем элементам;
- `Ignored` — утверждение обрабатывается, но игнорируется.

Атрибут `Preference` определяет уровень предпочтения утверждения в виде целого числа.

Спецификация WS-Policy 1.1 определяет также дочерний элемент `<OneOrMore>` элемента `<Policy>`. Элемент `<OneOrMore>` указывает требование обработки, по крайней мере, одного вложенного утверждения.

Элемент `<PolicyReference>` спецификации WS-Policy 1.1 может иметь дополнительный атрибут `Ref`, указывающий QName-имя политики, на которую ссылаются.

Спецификация WS-PolicyAttachment 1.1 дополнительно к атрибуту `PolicyURIs` элементов описания Web-сервиса определяет атрибут `PolicyRefs`, указывающий QName-имена политик, на которые ссылаются. Для гарантии обработки утверждений политики Web-сервиса в его WSDL-документе спецификация WS-PolicyAttachment 1.1 определяет дочерний элемент `<wsp:UsingPolicy wsdl:Required="true">` элемента `<wsdl:definitions>`.

Спецификация WS-PolicyAttachment 2004 г. не определяет атрибут `PolicyRefs`, но определяет элемент `<wsp:UsingPolicy wsdl:Required="true">`, а атрибут `PolicyURIs` данной спецификации использует значение полного URI-идентификатора политики, сформированного из значений атрибутов `xml:base` и `wsu:Id` элемента `<Policy>`.

Стандартные утверждения политики Web-сервиса согласно спецификации WS-PolicyAssertions 1.1 (пространство имен `http://schemas.xmlsoap.org/ws/2002/12/policy`) составляют следующий набор:

- `<TextEncoding>` определяет кодировку символов сообщения с помощью атрибута `Encoding`;
- `<Language>` указывает поддерживаемый язык сообщения с помощью атрибута `Language`;
- `<SpecVersion>` указывает совместимость с версией спецификации с помощью атрибута `URI` (URI-идентификатор версии спецификации);
- `<MessagePredicate>` определяет предварительное условие для сообщения. Элемент `<MessagePredicate>` содержит выражение условия, значением которого должно быть `true`. Атрибут `Dialect` указывает тип выражения, по умолчанию значение атрибута — `http://www.w3.org/TR/1999/REC-xpath-19991116` (XPath 1.0 выражение). Для XPath 1.0 выражений спецификация определяет следующие функции:
  - `GetBody(node)` возвращает элемент `<Body>` сообщения;
  - `IsInBody(node)` возвращает `true`, если элемент содержится в теле сообщения;
  - `GetHeader(node)` возвращает элемент `<Header>` сообщения;
  - `IsInHeader(node)` возвращает `true`, если элемент содержится в заголовке сообщения;

- `RoleURIForHeaderBlock(node)` возвращает роль для блока заголовка сообщения;
- `IsMandatoryHeaderBlock(node)` возвращает `true`, если блок заголовка имеет атрибут `mustUnderstand=true`;
- `IsRoleURIForNext(node, string)` возвращает `true`, если роль блока заголовка `http://www.w3.org/2003/05/soap-envelope/next`;
- `IsRoleURIForUltimateReceiver(node,string)` возвращает `true`, если роль блока заголовка `http://www.w3.org/2003/05/soap-envelope/ultimateReceiver`;
- `GetNodesetForNode(node)` возвращает набор элементов с атрибутами.

Если атрибут `Dialect` имеет значение `http://schemas.xmlsoap.org/2002/12/wsse#part`, тогда элемент `<MessagePredicate>` содержит перечень обязательных частей сообщения, указываемых с помощью следующих функций:

- `Body()` идентифицирует тело сообщения;
- `Header(x)` идентифицирует заголовок с QName-именем.

WS-Addressing

Спецификация WS-Addressing описывает независимый от транспортного протокола механизм адресации Web-сервисов и сообщений, участвующих в обмене с Web-сервисами.

Спецификация WS-Addressing 1.0 — Core определяет XML-элементы (пространство имен `xmlns:wsa="http://www.w3.org/2005/08/addressing"`), содержащие информацию о конечных точках Web-сервисов и адресную информацию сообщений.

Информацию о конечной точке Web-сервиса представляет XML-элемент `<wsa:EndpointReference>` (рис. 2.2), имеющий следующие дочерние элементы.

❒ Обязательный элемент `<wsa:Address>` указывает адрес конечной точки Web-сервиса. Спецификация предопределяет два значения этого элемента:

- `http://www.w3.org/2005/08/addressing/anonymous` — конечная точка не определяется точным адресом;

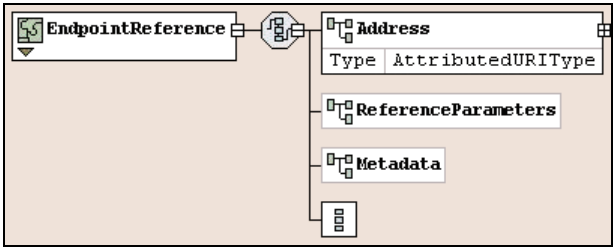


Рис. 2.2. Общая схема элемента `<EndpointReference>` пространства имен `http://www.w3.org/2005/08/addressing`

- http://www.w3.org/2005/08/addressing/none указывает, что сообщение не может быть послано.

- Необязательный элемент <wsa:ReferenceParameters> содержит элементы, принадлежащие своему пространству имен, которые представляют дополнительные параметры, связанные с конечной точкой, например, указывают идентификаторы ресурсов, предоставляемые Web-сервисом.
- Необязательный элемент <wsa:Metadata> содержит метаданные, описывающие поведение, политику и возможности конечной точки.

Спецификация Web Services Addressing 1.0 — Metadata определяет элементы, включаемые в элемент <wsa:Metadata>, обеспечивающие ссылки на элементы WSDL-описания или представляющие встроенное WSDL-описание.

При этом элемент <wsa:Metadata> может иметь атрибут wsdlLocation (пространство имен http://www.w3.org/ns/wsdl-instance), указывающий адрес WSDL-описания. В случае встроенного WSDL-описания элемент <wsa:Metadata> содержит сам WSDL-документ. Элемент <wsa:Metadata> также может содержать ссылки на WSDL-описание с помощью вложенных элементов <wsam:InterfaceName> (пространство имен http://www.w3.org/2007/05/addressing/metadata (рис. 2.3), содержит QName-имя элемента <interface> или <portType>) и <wsam:ServiceName> (содержит QName-имя элемента <service>, атрибут EndpointName может указывать имя элемента <endpoint> или <port>).

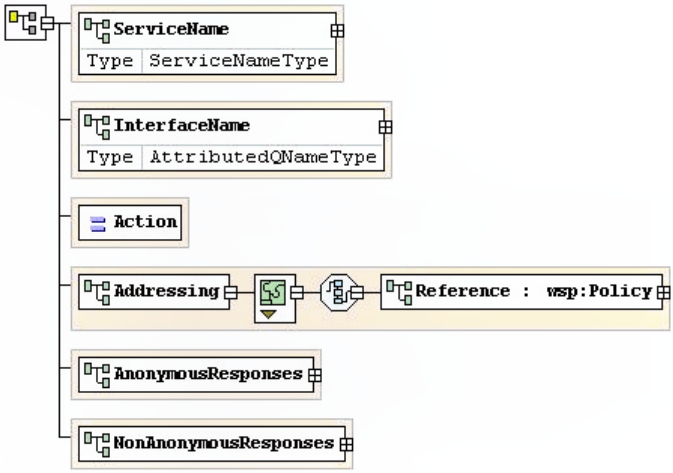


Рис. 2.3. Элементы и атрибуты пространства имен http://www.w3.org/2007/05/addressing/metadata

В WSDL-описании для указания того, что конечная точка Web-сервиса поддерживает спецификацию WS-Addressing, к элементам <wsdl20:endpoint>, <wsdl20:binding> или <wsdl11:port>, <wsdl11:binding> прикрепляется политика Web-сервиса, представленная элементом <wsam:Addressing> элемента <wsp:Policy>. Атрибут wsp:Optional (true/false) элемента <wsam:Addressing> указывает необязательность

поддержки WS-Addressing. Если в элементе `<wsam:Addressing>` присутствует элемент `<wsam:NonAnonymousResponses/>`, то это указывает на то, что конечная точка Web-сервиса требует ответных сообщений, которые используют адрес конечной точки ответа, отличный от `http://www.w3.org/2005/08/addressing/anonymous`. Если же в элементе `<wsam:Addressing>` присутствует элемент `<wsam:AnonymousResponses/>`, то это указывает на то, что конечная точка Web-сервиса требует ответных сообщений, использующих адрес конечной точки ответа как `http://www.w3.org/2005/08/addressing/anonymous`, адрес `http://www.w3.org/2005/08/addressing/none` тоже может при этом использоваться.

Также определен атрибут `wsam:Action` WSDL-элементов `<input>` и `<output>`, определяющий действие для сообщений операции.

Спецификация Web Services Addressing 1.0 — WSDL Binding определяет те же элементы `<wsaw:InterfaceName>` и `<wsaw:ServiceName>` (пространство имен `http://www.w3.org/2006/05/addressing/wsdl`, рис. 2.4) и атрибут `wsdlLocation` (пространство имен `http://www.w3.org/2006/01/wsdl-instance`). Для указания поддержки спецификации WS-Addressing данная спецификация определяет элемент `<wsaw:UsingAddressing wsdl:required="true"/>` или элемент `<wsoap:module uri="http://www.w3.org/2005/08/addressing/module" required="true"/>` WSDL-элемента `<binding>`. Также определен элемент `<wsaw:Anonymous>` — аналог `<wsam:AnonymousResponses/>` и `<wsam:NonAnonymousResponses/>`. Элемент `<wsaw:Anonymous>` используется в элементе `<binding>` и имеет возможные значения `optional`, `required` или `prohibited`. Аналогом атрибута `wsam:Action` является атрибут `wsaw:Action`.

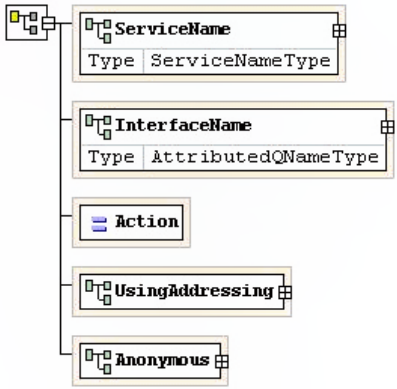


Рис. 2.4. Элементы и атрибуты пространства имен `http://www.w3.org/2006/05/addressing/wsdl`

Элемент `<wsa:EndpointReference>` используется совместно со спецификациями: WS-Trust — в элементе `<wsp:AppliesTo>`; WS-Discovery — как вложенный элемент элементов WS-Discovery тела SOAP-сообщения; BPEL — как переменная `partnerReference`; WSDL — в качестве элемента расширения элемента `<port>` или `<endpoint>`.

Адресную информацию сообщений представляют перечисленные далее элементы, включаемые в заголовок SOAP-сообщения.

- Необязательный элемент `<wsa:To>` указывает URI-адрес получателя данного сообщения, по умолчанию — `http://www.w3.org/2005/08/addressing/anonymous`.
- Необязательный элемент `<wsa:From>` указывает URI-адрес конечной точки отправителя данного сообщения. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы.
- Необязательный элемент `<wsa:ReplyTo>` указывает URI-адрес конечной точки получателя ответа на это сообщение, по умолчанию — `http://www.w3.org/2005/08/addressing/anonymous`. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы.
- Необязательный элемент `<wsa:FaultTo>` указывает URI-адрес конечной точки получателя сообщения об ошибке для данного сообщения. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы.
- Обязательный элемент `<wsa:Action>` — значение соответствующего атрибута `wsaw:Action` или `wsam:Action` WSDL-описания, которое содержит URI-идентификатор предназначения сообщений, представленных элементами `<input>`, `<output>` и `<fault>` элемента `<operation>` WSDL-описания. Данный идентификатор используется для фильтрации сообщений по их цели. По умолчанию значение `Action` формируется на основе значения шаблона обмена сообщениями MEP или имени ошибки.
- Необязательный элемент `<wsa:MessageID>` содержит IRI-идентификатор сообщения.
- Необязательный элемент `<wsa:RelatesTo>` указывает связь данного сообщения с другим сообщением с помощью пары IRI-идентификаторов. IRI-идентификатор связанного сообщения может принимать значение `http://www.w3.org/2005/08/addressing/unspecified`. Элемент имеет дополнительный атрибут `RelationshipType`, указывающий тип связи. Спецификация предопределяет значение атрибута `RelationshipType` как `http://www.w3.org/2005/08/addressing/reply`.
- Необязательный элемент `<wsa:ReferenceParameters>` содержит дополнительные параметры конечной точки.

Использование элементов, содержащих адресную информацию, в SOAP-сообщении снимает зависимость SOAP-протокола от транспортного протокола, т. к. представление адресной информации в заголовках транспортного протокола необходимо адаптировать в каждом конкретном случае.

Спецификация Web Services Addressing 1.0 — SOAP Binding определяет атрибут `wsa:IsReferenceParameter` для блоков заголовков SOAP-сообщения, принимающий значение `true`, если данный элемент определен в элементе `<wsa:ReferenceParameters>` элемента `<wsa:EndpointReference>`, на основе которого формируются сообщения.

Спецификация WS-Addressing 2004 г. определяет элемент `<wsa:EndpointReference>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/08/addressing`, рис. 2.5) как элемент со следующими дочерними элементами:

- обязательный элемент `<wsa:Address>` — URI-адрес конечной точки Web-сервиса;
- дополнительный элемент `<wsa:ReferenceProperties>` — свойства конечной точки Web-сервиса;
- дополнительный элемент `<wsa:ReferenceParameters>` — параметры конечной точки Web-сервиса;
- дополнительный элемент `<wsa:PortType>` — QName-имя элемента `<wsdl:portType>` WSDL 1.1 описания Web-сервиса;
- дополнительный элемент `<wsa:ServiceName>` — QName-имя элемента `<wsdl:service>` WSDL 1.1 описания Web-сервиса. Элемент `<wsa:ServiceName>` может иметь атрибут `PortName`, указывающий QName-имя элемента `<wsdl:port>` WSDL 1.1 описания Web-сервиса;
- дополнительный элемент `<wsp:Policy>` — политика Web-сервиса.

Спецификация WS-Addressing 2004 г. не определяет элемент `<wsa:ReferenceParameters>` в заголовке SOAP-сообщения.

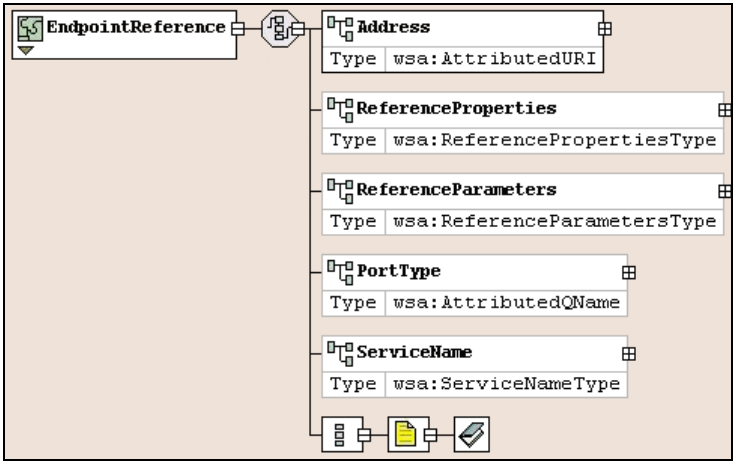


Рис. 2.5. Общая схема элемента `<EndpointReference>` пространства имен `http://schemas.xmlsoap.org/ws/2004/08/addressing`

Спецификация WS-Addressing 2004 г. предопределяет URI-идентификатор неопределенного адреса конечной точки Web-сервиса как `http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`, значение атрибута `RelationshipType` элемента `<wsa:RelatesTo>` по умолчанию — `wsa:Reply`, а также значение элемента `<wsa:RelatesTo>` для неопределенной связи с другим сообщением как `http://schemas.xmlsoap.org/ws/2004/08/addressing/id/unspecified`.

WS-Security

Спецификация WS-Security определяет стандартный набор SOAP-расширений для обеспечения целостности и конфиденциальности сообщений, участвующих в обмене с Web-сервисом, и тем самым предотвращения чтения и модификации SOAP-сообщений третьими лицами в процессе передачи. Спецификация WS-Security поддерживает обе версии протокола — SOAP 1.1 и SOAP 1.2.

Защита SOAP-сообщений обеспечивается путем передачи в SOAP-сообщении маркеров защиты (security token), таких как логин/пароль или сертификат безопасности, шифрованием маркеров защиты и данных сообщения, а также использованием цифровой подписи для обеспечения целостности SOAP-сообщения в процессе передачи.

Шифрование данных обеспечивается поддержкой спецификации XML Encryption (см. приложение "Спецификация XML Encryption" справки "Приложения" и PDF-файл в каталоге Приложения компакт-диска), а создание цифровой подписи предусмотрено спецификацией XML Signature (см. приложение "Спецификация XML Signature" справки "Приложения" и PDF-файл в каталоге Приложения компакт-диска).

Спецификация WS-Security определяет заголовок SOAP-сообщения, представленный дочерним элементом `<wsse:Security>` (пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`) элемента `<Header>`. Заголовок `<wsse:Security>` может быть не одним, при этом каждый заголовок `<wsse:Security>` предназначается для своего адресата, определенного с помощью атрибута `actor` или `role`.

Элемент `<wsse:Security>` предназначен для передачи информации, относящейся к обеспечению безопасности SOAP-сообщения, путем включения в него элементов SOAP-расширения. Спецификация WS-Security определяет вложенные элементы `<wsse:UsernameToken>` и `<wsse:BinarySecurityToken>` для передачи в SOAP-сообщении маркеров защиты логина/пароля и сертификата безопасности соответственно.

Элемент `<wsse:UsernameToken>` имеет атрибут `wsu:Id` (идентификатор элемента, пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`), вложенный элемент `<wsse:Username>` (содержит логин) и вложенные элементы, определяемые спецификацией UsernameToken Profile.

Спецификация UsernameToken Profile для элемента `<wsse:UsernameToken>` определяет вложенный элемент `<wsse:Password>`, который вместе с элементом `<wsse:Username>` образует пару "логин — пароль". При этом элемент `<wsse:Password>` имеет атрибут `Type`, указывающий тип пароля. Возможные значения атрибута `Type`:

- `PasswordText` — по умолчанию указывает, что пароль передается как простой текст;
- `PasswordDigest` указывает, что пароль зашифрован, используя формулу `Base64 (SHA-1(nonce + created + password))`.

Значения `nonce` и `created` определяются дополнительными вложенными элементами `<Nonce>` и `<Created>` (пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`). Элемент `<Nonce>` имеет атрибут `EncodingType` (URI-идентификатор кодировки, по умолчанию `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary`) и содержит случайное число, которое меняется в каждом сообщении, а элемент `<Created>` — дату создания.

Пароль типа `PasswordDigest` применяется только в случае, если первоначальный пароль заранее известен как отправителю, так и получателю сообщения.

Спецификация UsernameToken Profile версии 1.1, дополнительно к версии 1.0, определяет вложенные элементы `<Salt>` и `<Iteration>` (пространство имен `http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd`), которые используются для обеспечения целостности или конфиденциальности сообщения, при этом сам элемент `<Password>` отсутствует, т. е. пароль заранее известен отправителю и получателю сообщения. Элемент `<Salt>` содержит 128-битовое число в формате Base64, а элемент `<Iteration>` — количество итераций хэш-операции при кодировании пароля (по умолчанию 1000). Значение элемента `<Salt>` используется для получения ключа согласно следующему алгоритму:

```
K1 = SHA1(password + Salt)
. . .
Kn = SHA1(Kn-1)
```

Далее полученный ключ используется для получения значения кода проверки подлинности сообщения (Message Authentication Code, MAC), определяющего целостность сообщения при передаче, или как ключ для кодирования сообщения. Значение элемента `<Salt>`, имеющее высший битовый порядок 01, применяется для получения MAC-значения, а с высшим битовым порядком 02 используется для получения ключа шифрования сообщения.

Элемент `<wsse:BinarySecurityToken>` содержит X.509-сертификат или Kerberos-билет. Элемент `<wsse:BinarySecurityToken>` имеет следующие атрибуты:

- `wsu:Id` — идентификатор элемента;
- `EncodingType` — URI-идентификатор кодировки данных, по умолчанию `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary`;
- `ValueType` — URI-идентификатор типа данных элемента `<BinarySecurityToken>`.

В случае если элемент `<wsse:BinarySecurityToken>` содержит X.509-сертификат, применяемый для передачи публичного ключа, который может использоваться для аутентификации SOAP-сообщения или его шифрования, спецификация X.509 Certificate Token Profile предопределяет следующие значения атрибута `ValueType`:

- `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` — данные представляют сертификат X.509 v3;
- `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1` — данные представляют PKI-путь сертификата;

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7 — формат PKCS #7, содержащий сертификаты и дополнительно список CRL отозванных сертификатов;
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#x509v1 — данные представляют сертификат X.509 v1.

Если элемент <wsse:BinarySecurityToken> содержит Kerberos-билет, используемый для передачи сеансовых ключей, с помощью которых осуществляется аутентификация, цифровая подпись и шифрование SOAP-сообщения, спецификация Kerberos Token Profile определяет следующие значения атрибута ValueType:

- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ (или http://www.docs.oasis-open.org/wss/2004/07/oasis000000-wss-kerberos-token-profile-1.0#Kerberosv5_AP_REQ) — Kerberos-билет в формате Kerberos v5 AP-REQ (билет спецификации RFC 1510 и RFC 4120);
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ — Kerberos-билет в формате GSS Kerberos v5 AP-REQ (билет спецификации RFC 1510 и RFC 4120);
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510 — Kerberos-билет в формате Kerberos v5 AP-REQ (билет спецификации RFC 1510);
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510 — Kerberos-билет в формате GSS Kerberos v5 AP-REQ (билет спецификации RFC 1510);
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120 — Kerberos-билет в формате Kerberos v5 AP-REQ (билет спецификации RFC 4120);
- http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120 — Kerberos-билет в формате GSS Kerberos v5 AP-REQ (билет спецификации RFC 4120).

Элемент <wsse:Security> также может содержать в качестве маркеров защиты элементы <saml:Assertion> (пространство имен xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" или xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"), обеспечивающие поддержку стандарта Security Assertion Markup Language (SAML) (см. приложение "SAML" справки "Приложения" и PDF-файл в каталоге Приложения компакт-диска) и определенные спецификацией SAML Token Profile.

Спецификация SAML Token Profile 1.0 поддерживает стандарт SAML 1.1, а спецификация SAML Token Profile 1.1 — стандарты SAML 1.1 и SAML 2.0.

Элемент <wsse:Security> может содержать REL-маркеры защиты согласно спецификациям ISO/IEC 21000-5 Rights Expressions и Rights Expression Language (REL) Token Profile.

Rights Expression Language (REL) — это язык прав и разрешений использования цифровых ресурсов, таких как электронные книги, видео- и аудиоресурсы, инте-

рактивные игры и др. REL основан на языке XrML (eXtensible Rights Markup Language). REL-маркер защиты представлен XML-элементом `<license>`, имеющим дочерние элементы `<grant>` (описывает разрешения для пользователя, включая информацию о ключе, представленную элементом `<keyHolder>`) и `<issuer>` (содержит информацию о стороне, выдавшей разрешения).

Спецификация Rights Expression Language (REL) Token Profile определяет пространство имен для элементов REL-маркера как `urn:mpeg:mpeg21:2003:01-REL-R-NS` и URI-идентификатор типа маркера защиты как `http://docs.oasis-open.org/wss/oasis-wss-rel-token-profile-1.0.pdf#license`.

Маркеры защиты, включаемые в заголовок `wsse:Security`, могут быть зашифрованы согласно спецификации XML Encryption. В этом случае элемент `<xenc:EncryptedData>` (пространство имен `http://www.w3.org/2001/04/xmlenc#`) может использоваться в качестве вложенного элемента для передачи зашифрованных маркеров защиты.

Для обеспечения целостности SOAP-сообщения в процессе его передачи, все тело сообщения или его части могут быть подписаны цифровой подписью. При использовании цифровой подписи в заголовок `wsse:Security` добавляются элементы `<ds:Signature>` (пространство имен `http://www.w3.org/2000/09/xmldsig#`), описывающие цифровые подписи. Каждый элемент `<ds:Signature>` при этом содержит элементы `<ds:Reference>`, которые указывают подписанные элементы SOAP-сообщения с помощью атрибута `URI`, значением которого служит идентификатор элемента (`"#[`*идентификатор*`]"`). Элементы SOAP-сообщения могут быть идентифицированы локальными атрибутами `ID`, глобальным атрибутом `wsu:Id` (пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`), глобальным атрибутом `xml:id` или другими специфическими атрибутами. Дочерний элемент `<ds:KeyInfo>` элемента `<ds:Signature>` содержит информацию о публичном ключе цифровой подписи.

При создании цифровой подписи SOAP-сообщения или его шифровании используются ключи, информация о которых может передаваться в элементах `<wsse:UsernameToken>`, `<wsse:BinarySecurityToken>` и `<saml:Assertion>`. На основе значения дочернего элемента `<wsse:Password>` элемента `<wsse:UsernameToken>` может быть создан производный ключ подписи или шифрования, а элемент `<wsse:BinarySecurityToken>` может передавать сертификаты публичных ключей или Kerberos-билеты сеансовых ключей. Элемент `<saml:Assertion>` также может содержать ключи, используя элемент `<saml:Subject>`. Для того чтобы сослаться на элементы `<wsse:UsernameToken>`, `<wsse:BinarySecurityToken>` и `<saml:Assertion>`, несущие информацию о ключе, в элементе `<ds:KeyInfo>` используется вложенный элемент `<wsse:SecurityTokenReference>`, имеющий следующие атрибуты:

- `wsu:Id` — идентификатор элемента;
- `wsse11:TokenType` (пространство имен `http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd`) — тип маркера защиты, на который производится ссылка. При ссылке на зашифрованный ключ, содержащийся в элементе

<xenc:EncryptedKey>, значение атрибута — http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKey;

- wsse:Usage — идентификатор типа использования элемента <wsse:SecurityTokenReference>.

Спецификация определяет три типа ссылок, осуществляемых с помощью элемента <wsse:SecurityTokenReference>:

- прямые ссылки;
- ссылки, использующие идентификаторы ключей;
- встроенные маркеры защиты.

Прямая ссылка производится с помощью элемента <wsse:Reference>, являющегося дочерним элементом элемента <wsse:SecurityTokenReference>. Элемент <wsse:Reference> имеет атрибуты URI (идентификатор маркера защиты) и ValueType (идентификатор типа маркера защиты). При использовании зашифрованного ключа, информация о котором содержится в элементе <xenc:EncryptedKey>, атрибут URI имеет значение идентификатора элемента <xenc:EncryptedKey>.

Элемент <wsse:SecurityTokenReference> может ссылаться на идентификаторы ключей, используя вложенный элемент <wsse:KeyIdentifier>, содержащий закодированный идентификатор ключа и имеющий следующие атрибуты:

- wsu:Id — идентификатор элемента;
- ValueType — тип значения элемента <wsse:KeyIdentifier>. При ссылке на зашифрованный ключ значение атрибута — http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKeySHA1. Спецификация также определяет значение атрибута ValueType для ссылки на любой маркер защиты как http://docs.oasis-open.org/wss/oasiswss-soap-message-security-1.1#ThumbPrintSHA1;
- EncodingType — формат значения идентификатора (#Base64Binary).

Элемент <wsse:SecurityTokenReference> может сам содержать маркеры защиты, используя вложенный элемент <wsse:Embedded> (дополнительный атрибут wsu:Id).

Вместо элемента <wsse:SecurityTokenReference> элемент <ds:KeyInfo> может содержать элемент <ds:KeyName>, указывающий имя ключа.

При ссылке на элемент <wsse:UsernameToken> используется только прямая ссылка, при этом атрибут ValueType принимает значение http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken.

При ссылке на элемент <wsse:BinarySecurityToken> используется прямая ссылка с помощью атрибута URI элемента <wsse:Reference> и ссылка по идентификатору ключа посредством элемента <wsse:KeyIdentifier>, содержащего значение SubjectKeyIdentifier сертификата, при этом атрибут ValueType принимает значение http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier. Кроме того, элемент <wsse:SecurityTokenReference> может ссылаться на организацию, выпустившую сертификат, и серийный номер

сертификата, используя вложенный элемент `<ds:X509Data>` и его элементы `<ds:X509IssuerSerial>`, `<ds:X509IssuerName>` и `<ds:X509SerialNumber>`.

Если элемент `<wsse:BinarySecurityToken>` содержит Kerberos-билет, тогда атрибут `ValueType` элемента `<wsse:Reference>` или атрибут `wsse11:TokenType` элемента `<wsse:SecurityTokenReference>` принимает значение атрибута `ValueType` элемента `<wsse:BinarySecurityToken>`, а значение атрибута `ValueType` элемента `<wsse:KeyIdentifier>` — `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5APREQSHA1`.

При ссылке на элемент `<saml:Assertion>` используется прямая ссылка, ссылка по идентификатору ключа и встроенное утверждение. Прямые ссылки применяются только для утверждений SAML V2.0, при этом атрибут `URI` элемента `<wsse:Reference>` принимает значение HTTP-запроса, состоящего из адреса конечной точки центра аутентификации и значения атрибута `ID` элемента `<saml:Assertion>`. Атрибут `wsse11:TokenType` элемента `<wsse:SecurityTokenReference>` в случае прямой ссылки принимает значение `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0`, а атрибут `ValueType` элемента `<wsse:Reference>` отсутствует. Прямые ссылки обязательны для утверждений SAML V2.0, отсутствующих в том сообщении, в котором располагаются ссылки на них. Если же ссылка и утверждение SAML V2.0 находятся в одном сообщении, помимо прямой ссылки может использоваться *ссылка по идентификатору ключа*. Ссылка по идентификатору ключа также используется для ссылок на утверждения SAML V1.1, как находящихся в том же сообщении, что и ссылка, так и отсутствующих. Для утверждений, находящихся в том же сообщении, что и ссылка, атрибут `ValueType` элемента `<wsse:KeyIdentifier>` должен принимать значение `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0#SAMLAssertionID` (SAML V1.1) или `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID` (SAML V2.0), атрибут `wsse11:TokenType` элемента `<wsse:SecurityTokenReference>` должен принимать значение `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1` или `http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0`, а элемент `<wsse:KeyIdentifier>` при этом содержит значение идентификатора элемента `<saml:Assertion>`. Для утверждений SAML V1.1, не находящихся в том же сообщении, что и ссылка, элемент `<wsse:SecurityTokenReference>` должен содержать элемент `<saml:AuthorityBinding>` со значением атрибута `AuthorityKind= "samlp:AssertionIdReference"`. В случае встроенного утверждения элемент `<wsse:SecurityTokenReference>` содержит элемент `<wsse:Embedded>`, включающий в себя элемент `<saml:Assertion>`.

При ссылке на REL-маркер может использоваться значение как атрибута `wsu:Id`, так и значение атрибута `licenseId` элемента `<license>`. Если используется значение атрибута `licenseId`, то значение атрибута `ValueType` элемента `<wsse:Reference>` устанавливается как `http://docs.oasis-open.org/wss/oasis-wss-rel-token-profile-1.0.pdf#license`.

Для обеспечения целостности SOAP-сообщения *маркеры защиты* могут быть также снабжены цифровой подписью. Спецификация определяет общий механизм включения маркеров защиты в цифровую подпись. При этом элемент

<ds:Reference> ссылается своим атрибутом URI не на маркер защиты, а на идентификатор элемента <wsse:SecurityTokenReference>, в свою очередь ссылающийся на маркер защиты. Кроме того, атрибут Algorithm элемента <ds:Transform> (корневой элемент <ds:SignedInfo>) имеет значение http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STRTransform, определяя алгоритм трансформации STR Dereference Transform. Алгоритм STR Dereference Transform означает, что цифровая подпись применяется не к самому элементу <wsse:SecurityTokenReference>, а к маркеру защиты, на который он ссылается.

При получении подписанного цифровой подписью SOAP-сообщения адресат должен отправить в ответ сообщение, содержащее в заголовке <wsse:Security> полученную им цифровую подпись. Это дополнительно гарантирует защиту SOAP-сообщения от несанкционированных действий третьих лиц. Цифровая подпись будет находиться в элементе <wsse11:SignatureConfirmation>, как значение его атрибута Value. Если полученное адресатом сообщение не содержит ни одного элемента <ds:Signature>, тогда атрибут Value элемента <wsse11:SignatureConfirmation> будет отсутствовать.

Для обеспечения конфиденциальности SOAP-сообщения все тело сообщения или его части, а также заголовки SOAP-сообщения могут быть зашифрованы согласно спецификации XML Encryption. При шифровании тела SOAP-сообщения элемент <SOAP-ENV:Body> содержит элементы <xenc:EncryptedData>, описывающие зашифрованные данные. Если же зашифрован заголовок SOAP-сообщения, то в элемент <SOAP-ENV:Header> добавляется элемент <wsse11:EncryptedHeader>, включающий в себя элемент <xenc:EncryptedData> (атрибут wsu:Id — идентификатор элемента), который описывает зашифрованный заголовок.

Для отображения перечня зашифрованных элементов SOAP-сообщения в заголовке <wsse:Security> спецификация определяет вложенный элемент <xenc:ReferenceList>, который содержит элемент <xenc:DataReference>. Элемент <xenc:DataReference> имеет атрибут URI, принимающий значение идентификатора элемента, который описывает зашифрованные данные. Таким элементом может быть элемент <xenc:EncryptedData> тела SOAP-сообщения или элемент <wsse11:EncryptedHeader> заголовка SOAP-сообщения.

Для описания ключей шифрования спецификация определяет вложенный элемент <xenc:EncryptedKey> заголовка <wsse:Security>. При этом вложенный элемент <ds:KeyInfo> элемента <xenc:EncryptedKey> будет нести информацию о ключе шифрования. Элемент <ds:KeyInfo> может содержать элемент <wsse:SecurityTokenReference>, ссылающийся на соответствующий маркер защиты. Элемент <xenc:EncryptedKey> может также включать в себя элементы <xenc:ReferenceList>, указывающие зашифрованные этим ключом данные.

При обмене защищенными SOAP-сообщениями информация, содержащаяся в заголовке <wsse:Security>, может со временем устареть. Для определения времени создания заголовка <wsse:Security> и его времени окончания действия спецификация определяет вложенный элемент <wsu:Timestamp> (пространство имен http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd) элемен-

та `<wsse:Security>`. Элемент `<wsu:Timestamp>` может иметь следующие атрибуты и вложенные элементы:

- атрибут `wsu:Id` — идентификатор элемента;
- элемент `<wsu:Created>` — время создания, имеет необязательный атрибут `ValueType` — формат значения элемента;
- элемент `<wsu:Expires>` — время окончания действия, имеет необязательный атрибут `ValueType` — формат значения элемента.

Если SOAP-сообщение снабжается цифровой подписью перед шифрованием, тогда элементы, относящиеся к цифровой подписи, должны располагаться в заголовке `<wsse:Security>` после элементов, относящихся к шифрованию. Если же SOAP-сообщение снабжается цифровой подписью после его шифрования, тогда элементы, относящиеся к цифровой подписи, должны располагаться в заголовке `<wsse:Security>` перед элементами, которые касаются шифрования.

Основные отличия спецификации WS-Security версии 1.1 от WS-Security версии 1.0:

- для спецификации WS-Security 1.0 созданы профили UsernameToken Profile 1.0, X.509 Certificate Token Profile 1.0, SAML Token Profile 1.0, Kerberos Token Profile 1.0, а для спецификации WS-Security 1.1 — профили UsernameToken Profile 1.1, X.509 Certificate Token Profile 1.1, SAML Token Profile 1.1, Kerberos Token Profile 1.1;
- в WS-Security 1.1 маркер защиты может быть зашифрован и помещен в элементе `<xenc:EncryptedData>` в заголовок `<wsse:Security>`;
- в WS-Security 1.1 элемент `<wsse:SecurityTokenReference>` имеет атрибут `wsse11:TokenType`;
- в WS-Security 1.1 элемент `<wsse11:SignatureConfirmation>` посылается в ответ на полученный элемент `<ds:SignatureValue>`;
- в WS-Security 1.1 определен элемент `<wsse11:EncryptedHeader>` для зашифрованного заголовка.

Спецификация UsernameToken Profile 1.0 отличается от спецификации UsernameToken Profile 1.1 тем, что в ней определены элементы `<wsse11:Salt>` и `<wsse11:Iteration>` для получения производного ключа.

Спецификация X.509 Certificate Token Profile 1.1, в отличие от спецификации X.509 Certificate Token Profile 1.0, поддерживает пространство имен `http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd`.

Спецификация Kerberos Token Profile 1.1 определяет значение атрибута `ValueType` элемента `<wsse:KeyIdentifier>` равным `http://docs.oasis-pen.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5APREQSHA1`, а также другие значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>`.

Профиль SAML Token Profile 1.1 поддерживает спецификацию SAML 1.0 и SAML 2.0, а профиль SAML Token Profile 1.0 — только спецификацию SAML 1.0. Специ-

фикация SAML Token Profile 1.1 определяет значение атрибута wsse11:TokenType элемента <wsse:SecurityTokenReference> как http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1 или http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0, а значение атрибута ValueType элемента <wsse:KeyIdentifier> как http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0#SAMLAssertionID или http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID.

Спецификация Rights Expression Language (REL) Token Profile 1.1, в отличие от спецификации Rights Expression Language (REL) Token Profile 1.0, поддерживает пространство имен http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd.

Для передачи данных не XML-формата или XML-данных большого размера используются SOAP-сообщения с вложениями. В случае если SOAP-сообщение передается с вложениями, спецификация Web Services Security SOAP Messages with Attachments (SwA) Profile описывает применение стандарта WS-Security для такого рода сообщений.

В случае если вложение SOAP-сообщения снабжается цифровой подписью, элемент <ds:Reference>, содержащийся в элементе <ds:Signature> заголовка <wsse:Security>, ссылается на вложение с помощью своего атрибута URI. Атрибут URI элемента <ds:Reference>, при ссылке на вложение SOAP-сообщения, принимает значение заголовка Content-ID с префиксом cid: (схема URL CID), т. е. при Content-ID имеем ID — <ds:Reference URI="cid:ID">. Для атрибута Algorithm элемента <ds:Transform>, являющегося дочерним элементом элемента <ds:Reference>, спецификация определяет следующие возможные значения:

- http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform или http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform указывает, что только содержимое вложения снабжено цифровой подписью;
- http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Complete-Signature-Transform или http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete-Transform указывает, что вместе с содержимым вложения подписаны и его заголовки Content-Description, Content-Disposition, Content-ID, Content-Location и Content-Type;
- http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Ciphertext-Transform указывает, что цифровой подписью снабжено зашифрованное содержимое вложения SOAP-сообщения.

В случае если вложение SOAP-сообщения шифруется, дочерний элемент <xenc:CipherData> элемента <xenc:EncryptedData>, находящегося в заголовке <wsse:Security>, должен содержать элемент <xenc:CipherReference URI=cid:ID"> (Content-ID: <ID>), ссылающийся на вложение. Элемент <xenc:CipherReference> должен иметь дочерний элемент <xenc:Transforms>, вложенный элемент <ds:Transform> которого имеет атрибут Algorithm со значением http://docs.oasis-

open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Ciphertext-Transform или http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform. Для атрибута Type элемента <xenc:EncryptedData> спецификация определяет следующие возможные значения:

- http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Only или http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only указывает, что шифруется только содержимое вложения;
- http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Complete или http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete указывает, что шифруется содержимое вложения и его заголовки.

WS-Trust

Спецификация WS-Security определяет, каким образом обеспечивается защита SOAP-сообщений с помощью передачи маркеров защиты, которые могут использоваться для аутентификации, подписи и шифрования SOAP-сообщений. Спецификации WS-Policy и WS-PolicyAttachment дают возможность Web-сервисам, участвующим в обмене SOAP-сообщениями, объявлять свои политики, которые могут накладывать определенные ограничения на входящие SOAP-сообщения, такие как требования к указанным частям входящих SOAP-сообщений по шифрованию и снабжению цифровой подписью с использованием определенных маркеров защиты. Однако данные спецификации не определяют механизма получения маркеров защиты для их дальнейшего использования. X.509-сертификаты получают у Центра сертификации (Certification Authority, CA), Kerberos-билеты — у Центра распространения ключей (Key Distribution Center, KDC), SAML-утверждения — у Центра аутентификации. Спецификация Web Services Trust Language (WS-Trust) как раз и описывает стандартный механизм обмена маркерами защиты и установления доверительных взаимоотношений между сторонами. Спецификация WS-Trust позволяет Web-сервисам, участвующим во взаимодействии, согласовывать свои политики и характеристики маркеров защиты для успешной обработки SOAP-сообщений.

> ***Примечание***
>
> Спецификация SAML также определяет SAML-протоколы, позволяющие обмениваться SAML-утверждениями, однако спецификация SAML фокусируется на системе, состоящей из HTTP-агента клиента (как правило, Web-браузера) и Web-сайтов, одни из которых аутентифицируют клиента и выдают ему SAML-утверждения, а другие на основе SAML-утверждений разрешают клиенту доступ к локальным ресурсам. Спецификация же WS-Trust определяет общий для всех маркеров защиты механизм обмена, предназначенный для распределенной системы Web-сервисов.

Спецификация WS-Trust описывает систему, состоящую из взаимодействующих между собой сторон.

- *Web-сервис, запрашивающий доступ к другому Web-сервису.* Запрашивающий Web-сервис может иметь свою политику, предъявляющую требования к входя-

щим SOAP-сообщениям, которые должны включать в себя определенные маркеры защиты (security token). Маркеры защиты состоят из набора утверждений (claim), содержащих информацию о ключах, имени, идентификаторах, правах и т. д. другого Web-сервиса. Запрашивающий Web-сервис имеет свой сервис установления доверия (trust engine). Сервис установления доверия отвечает за проверку соответствия полученных маркеров защиты политике Web-сервиса, проверку цифровых подписей, проверку того, что маркеры защиты выпущены доверенным центром.

- *Web-сервис, который представляет центр, отвечающий за выдачу маркеров защиты (Security Token Service, STS).* STS-сервис может сам генерировать маркеры защиты или обращаться к другому STS-сервису за выдачей маркеров защиты. STS-сервис также может иметь свою политику, предъявляющую требования к входящим SOAP-сообщениям, которые должны включать в себя определенные маркеры защиты, а также сервис установления доверия. С STS-сервисом взаимодействует как Web-сервис, запрашивающий доступ к другому Web-сервису для получения необходимых маркеров защиты, так и запрашиваемый Web-сервис для проверки получаемых маркеров защиты. Таким образом, STS-сервис отвечает за выпуск, обновление и подтверждение маркеров защиты.
- *Web-сервис, который запрашивают*, может иметь свою политику и сервис установления доверия.

Спецификация WS-Trust определяет элемент `<wst:RequestSecurityToken>` тела SOAP-сообщения (элемент `<SOAP-ENV:Body>`) для запроса маркера защиты (RST-запрос). Префикс `wst` является префиксом пространства имен:

- `http://docs.oasis-open.org/ws-sx/ws-trust/200512` — спецификации WS-Trust 1.4 и WS-Trust 1.3;
- `http://schemas.xmlsoap.org/ws/2005/02/trust` — спецификация WS-Trust 1.2.

При запросе маркера защиты элемент `<wst:RequestSecurityToken>` должен быть снабжен цифровой подписью отправителя запроса.

Элемент `<wst:RequestSecurityToken>` (рис. 2.6) имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `Context` — URI-идентификатор запроса, позволяющий устанавливать соответствие между запросом и ответом;
- необязательный элемент `<wst:TokenType>` — содержит URI-идентификатор типа запрашиваемого маркера защиты;
- обязательный элемент `<wst:RequestType>` — содержит URI-идентификатор типа запроса;
- необязательный элемент `<wst:SecondaryParameters>` — содержит параметры запроса.

Элемент `<wst:RequestSecurityTokenResponse>`, который может находиться в теле SOAP-сообщения или в его заголовке, содержит ответ на запрос маркера защиты (RSTR-ответ). Элемент `<wst:RequestSecurityTokenResponse>`, как правило, снабжен

Рис. 2.6. Общая схема элемента `<wst:RequestSecurityToken>`

цифровой подписью отправителя и имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `Context` — URI-идентификатор ответа, позволяющий устанавливать соответствие между ответом и запросом, или, если данные посылаются без запроса, атрибут содержит URI-идентификатор использования маркера защиты;
- необязательный элемент `<wst:TokenType>` — URI-идентификатор типа возвращаемого маркера защиты;
- необязательный элемент `<wst:RequestedSecurityToken>` — маркер защиты или ссылка на него, определенная с помощью вложенного элемента `<wsse:SecurityTokenReference>`. Ссылка на маркер защиты используется, если маркер защиты обеспечивает защиту сообщения, содержащего элемент `<wst:RequestSecurityTokenResponse>`, и находится в заголовке `<wsse:Security>` сообщения или расположен вне сообщения.

В случае RST-запроса, посылаемого STS-сервису для выдачи маркера защиты, элемент `<wst:RequestType>` содержит URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue`, а элемент `<wst:RequestSecurityToken>` содержит следующие дополнительные элементы.

- `<wsp:AppliesTo>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/policy` или `http://www.w3.org/ns/ws-policy`) указывает элементы описания запрашиваемого Web-сервиса, к которым применяется данный маркер защиты согласно элементу `<AppliesTo>` элемента `<PolicyAttachment>` политики Web-сервиса. Как правило, элемент `<wsp:AppliesTo>` указывает URL-адрес конечной точки Web-сервиса, которому полученный маркер защиты будет послан, с помощью вложенных элементов `<wsa:EndpointReference>` и `<wsa:Address>` (пространство имен `http://www.w3.org/2005/08/addressing`, спецификация WS-Addressing).
- `<wst:Claims>` содержит информацию о ключах, имени, идентификаторах, правах и т. д., которую запрашивающий Web-сервис просит включить в генерируемый STS-сервисом маркер защиты. Этот набор утверждений формируется на основе политики Web-сервиса, для доступа к которому у STS-сервиса запрашивается маркер доступа. Элемент `<wst:Claims>` имеет обязательный атрибут `Dialect`, указывающий URI-идентификатор синтаксиса, используемого для определения набора запрашиваемых утверждений.
- `<wst:Entropy>` содержит значение, используемое для создания ключа. Данное значение указывается с помощью элемента `<xenc:EncryptedKey>` или `<wst:BinarySecret>`. Элемент `<wst:BinarySecret>` содержит значение первичного ключа в формате Base64 и имеет дополнительный атрибут `Type`, указывающий тип ключа. Спецификация предопределяет следующие значения для атрибута `Type`:
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/AsymmetricKey` — персональный ключ для публичного ключа;

- http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey — симметричный ключ, по умолчанию;
- http://docs.oasis-open.org/ws-sx/ws-trust/200512/Nonce — случайное значение.

❒ <wst:Lifetime> определяет рекомендуемое время действия генерируемого маркера защиты с помощью дополнительных вложенных элементов <wsu:Created> (время создания) и <wsu:Expires> (время окончания действия) (пространство имен http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd).

Элемент <wst:RequestSecurityToken> позволяет запросить выдачу STS-сервисом только одного маркера защиты. Если же возникает необходимость генерации нескольких различных маркеров защиты, то элемент <wst:RequestSecurityToken-Collection> позволяет запросить несколько маркеров защиты в одном запросе (RSTC-запрос).

Элемент <wst:RequestSecurityTokenCollection> содержит два или более элементов <wst:RequestSecurityToken>, элементы <wst:RequestType> которых содержат URI-идентификатор http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchIssue.

Элемент <wst:RequestSecurityTokenResponseCollection> (RSTRC-ответ) содержит ответ на RSTC-запрос. Элемент <wst:RequestSecurityTokenResponseCollection> содержит два или более элементов <wst:RequestSecurityTokenResponse>.

Элемент <wst:RequestSecurityTokenResponse> (рис. 2.7), возвращающий сгенерированный STS-сервисом маркер защиты, может содержать, дополнительно к элементам <wst:TokenType> и <wst:RequestedSecurityToken>, перечисленные далее элементы.

Примечание

Элемент wst:RequestedSecurityToken, как правило, содержит маркер защиты, снабженный цифровой подписью доверенного центра, который его выпустил (Signed Security Token). Это может быть X.509-сертификат, Kerberos-билет или SAML-утверждение.

❒ <wsp:AppliesTo> (пространство имен http://schemas.xmlsoap.org/ws/2004/09/policy или http://www.w3.org/ns/ws-policy) указывает элементы описания запрашиваемого Web-сервиса, к которым применяется данный маркер защиты.

❒ <wst:RequestedAttachedReference> и <wst:RequestedUnattachedReference> позволяют использовать маркер защиты без знания его типа и без его анализа. Элемент <wst:RequestedAttachedReference> указывает полную ссылку на маркер защиты, расположенный в сообщении, с помощью элемента <wsse:SecurityTokenReference>. Элемент <wst:RequestedUnattachedReference> указывает полную ссылку на маркер защиты, расположенный вне сообщения, с помощью элемента <wsse:SecurityTokenReference>. При наличии элементов <wst:RequestedAttachedReference> и <wst:RequestedUnattachedReference> ссылка <wsse:SecurityTokenReference> не создается посредством идентификатора маркера защиты запрашивающим Web-сервисом при отправке маркера защиты

Рис. 2.7. Общая схема элемента `<wst:RequestSecurityTokenResponse>`

запрашиваемому Web-сервису, а применяется полученная ссылка, содержащаяся в элементах `<wst:RequestedAttachedReference>` и `<wst:RequestedUnattachedReference>`.

- `<wst:RequestedProofToken>` содержит маркер защиты (Proof-of-Possession Token, POP). POP-маркер имеет ключ, дающий право его получателю на использование связанного с POP-маркером возвращаемого маркера защиты `<wst:RequestedSecurityToken>`, например, для цифровой подписи маркера защиты `<wst:RequestedSecurityToken>`. POP-маркер может представлять логин и пароль или ключ, которые могут быть зашифрованы общим ключом, например публичным ключом запрашивающего Web-сервиса. Если POP-маркер обеспечивает защиту сообщения, содержащего элемент `<wst:RequestSecurityTokenResponse>`, и находится в заголовке `<wsse:Security>`, тогда элемент `<wst:RequestedProofToken>` ссылается на маркер защиты, используя элемент `<wsse:SecurityTokenReference>`. Элемент `<wst:RequestedProofToken>` определяет сгенерированный STS-сервисом ключ с помощью вложенного элемента `<xenc:EncryptedKey>` или `<wst:BinarySecret>`.
- `<wst:Entropy>` содержит значение, используемое для создания ключа. Данное значение указывается с помощью элемента `<xenc:EncryptedKey>` или `<wst:BinarySecret>`. Если элемент `<wst:Entropy>` запроса не используется, тогда сгенерированный STS-сервисом ключ возвращается в элементе `<wst:RequestedProofToken>`. Если же в запросе и ответе присутствует элемент `<wst:Entropy>`, тогда элемент `<wst:RequestedProofToken>` не содержит ключ, а включает в себя элемент `<wst:ComputedKey>`, указывающий способ генерации производного ключа. Спецификация предопределяет значение элемента `<wst:ComputedKey>` как `http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1` (алгоритм `key = P_SHA1(EntREQ, EntRES)`).
- `<wst:Lifetime>` определяет время действия созданного маркера защиты с помощью дополнительных вложенных элементов `<wsu:Created>` (время создания) и `<wsu:Expires>` (время окончания действия) (пространство имен `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`).

Сгенерированные STS-сервисом маркеры защиты могут возвращаться не в элементах `<wst:RequestSecurityTokenResponse>` и `<wst:RequestSecurityTokenResponseCollection>`, а в элементе `<wst:IssuedTokens>`, расположенном в заголовке SOAP-сообщения. В этом случае заголовок `<wst:IssuedTokens>` содержит элементы `<wst:RequestSecurityTokenResponse>` с созданными маркерами защиты.

В случае RST-запроса, посылаемого STS-сервису для обновления уже созданного маркера защиты, время действия которого истекло, элемент `<wst:RequestType>` содержит URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew`, а элемент `<wst:RequestSecurityToken>`, помимо элементов `<wst:TokenType>` и `<wst:RequestType>`, содержит следующие дополнительные элементы:

- обязательный элемент `<wst:RenewTarget>` содержит маркер защиты, требующий обновления или ссылку на него `<wsse:SecurityTokenReference>`;

- необязательный элемент `<wst:AllowPostdating/>` указывает, что возвращаемый маркер защиты может использоваться позднее, а не сразу;
- необязательный элемент `<wst:Renewing/>` определяет условия обновления с помощью атрибутов `Allow` (если `true`, по умолчанию, тогда время действия маркера защиты может быть продлено путем обновления) и `OK` (если `true`, тогда время действия маркера защиты может быть заново установлено после его окончания путем обновления, по умолчанию `false`);
- необязательный элемент `<wst:Lifetime>` указывает запрашиваемое время действия маркера защиты.

При RSTC-запросе обновления (`<wst:RequestSecurityTokenCollection>`) элемент `<wst:RequestType>` содержит URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchRenew`.

RSTR-ответ `<wst:RequestSecurityTokenResponse>` содержит элемент `<wst:RequestedSecurityToken>` с обновленным маркером защиты и элемент `<wst:Lifetime>`, определяющий новое время действия маркера защиты.

В случае RST-запроса, посылаемого STS-сервису для прекращения действия уже созданного маркера защиты, элемент `<wst:RequestType>` указывает URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel`, а элемент `<wst:RequestSecurityToken>`, помимо элемента `<wst:RequestType>`, включает в себя обязательный элемент `<wst:CancelTarget>`, который содержит маркер защиты, требующий прекращения действия или ссылку `<wsse:SecurityTokenReference>` на него. При RSTC-запросе прекращения действия (`<wst:RequestSecurityTokenCollection>`) элемент `<wst:RequestType>` содержит URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchCancel`. В ответ STS-сервис посылает сообщение с вложенным элементом `<wst:RequestedTokenCancelled/>` элемента `<wst:RequestSecurityTokenResponse>`, указывающим, что операция прекращения прошла успешно.

Если действие маркера защиты прекращает сам STS-сервис, тогда он посылает сообщение, содержащее элемент `<wst:RequestSecurityToken>` с вложенными элементами:

- `<wst:RequestType>` указывает `http://docs.oasis-open.org/ws-sx/ws-trust/200512/STSCancel`;
- `<wst:CancelTarget>` содержит маркер защиты, действие которого прекращено, или ссылку `<wsse:SecurityTokenReference>` на него.

В случае RST-запроса, посылаемого STS-сервису для подтверждения маркера защиты, элемент `<wst:RequestType>` указывает URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate`, а элемент `<wst:RequestSecurityToken>`, помимо элементов `<wst:TokenType>` и `<wst:RequestType>`, включает в себя обязательный элемент `<wst:ValidateTarget>`, содержащий маркер защиты, требующий подтверждения, или ссылку `<wsse:SecurityTokenReference>` на него.

В результате подтверждения STS-сервис выдает статус подтверждения и/или новый маркер защиты. Если от STS-сервиса требуется только статус, тогда элемент

`<wst:TokenType>` запроса указывает `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status`. Ответ STS-сервиса (`<wst:RequestSecurityTokenResponse>`) содержит элементы `<wst:TokenType>`, `<wst:RequestedSecurityToken>` (новый маркер защиты, созданный на основе полученного в запросе) и `<wst:Status>`. Элемент `<wst:Status>` содержит элементы `<wst:Code>` (`http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid` или `http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/invalid`) и `<wst:Reason>` (дополнительная информация статуса).

При RSTC-запросе подтверждения (`<wst:RequestSecurityTokenCollection>`) элемент `<wst:RequestType>` содержит URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/BatchValidate`.

В случае RST-запроса, посылаемого STS-сервису для получения маркера защиты, используемого запрашивающим Web-сервисом для шифрования, элемент `<wst:RequestType>` указывает URI-идентификатор `http://docs.oasis-open.org/ws-sx/ws-trust/200512/KET`.

В некоторых случаях STS-сервис, после получения первоначального запроса, может потребовать от запрашивающего Web-сервиса дополнительную информацию, например, в случае просроченного элемента `<wsu:Timestamp>` заголовка `<wsse:Security>` запроса. При этом STS-сервис отправляет Web-сервису сообщение, включающее в себя элемент `<wst:RequestSecurityTokenResponse>` с вложенными элементами, представляющими дополнительные требования STS-сервиса. В ответ запрашивающий Web-сервис посылает свой ответ `<wst:RequestSecurityTokenResponse>` с элементами ответа на дополнительные требования. Далее STS-сервис, после обработки ответа, посылает окончательное сообщение `<wst:RequestSecurityTokenResponseCollection>` запрашивающему Web-сервису, содержащее маркеры защиты. Спецификация определяет несколько перечисленных далее элементов, представляющих дополнительные требования.

- `<wst:SignChallenge>` — требование снабжать цифровой подписью указанную информацию. Эта информация содержится в дочернем элементе `<wst:Challenge>`. В ответе на данное требование находится элемент `<wst:SignChallengeResponse>` с элементом `<wst:Challenge>` в том же виде, как он был получен, т. е. элемент `<wst:Challenge>` ответа должен содержать информацию, требующую цифровой подписи. STS-сервис обрабатывает созданную цифровую подпись, содержащуюся в сообщении ответа, и в случае ее подтверждения посылает Web-сервису маркеры защиты.
- `<wst14:InteractiveChallenge>` (пространство имен `http://docs.oasis-open.org/ws-sx/ws-trust/200802`) — требование пользователю запрашивающего Web-сервиса предоставить дополнительную информацию. Этот элемент имеет следующие дочерние элементы:
  - `<wst14:Title>` — заголовок требования, отображаемый пользователю;
  - `<wst14:TextChallenge>` — требование пользователю ввести текстовую информацию. Этот элемент имеет обязательный атрибут `RefId` (идентификатор требования) и дополнительные атрибуты `Label` (метка текстового поля ввода),

MaxLen (максимальная длина строки ввода), HideText (true/false; указывает, что вводимый текст должен быть скрытым, например, символами *), а также дочерний элемент <wst14:Image> (содержит изображение в формате Base64, обязательный атрибут MimeType — формат изображения, например image/gif или image/jpg). Если STS-сервис требует от пользователя ввода PIN-кода или одноразового пароля (OTP), тогда атрибут RefId элемента <wst14:TextChallenge> имеет значение http://docs.oasis-open.org/ws-sx/ws-trust/200802/challenge/PIN или http://docs.oasis-open.org/ws-sx/ws-trust/200802/challenge/OTP соответственно;

- <wst14:ChoiceChallenge> — требование пользователю выбрать элементы. Этот элемент имеет обязательный атрибут RefId (идентификатор требования) и дополнительные атрибуты Label (метка списка выбора) и ExactlyOne (true/false; указывает необходимость выбора только одного элемента в списке, по умолчанию — false), а также обязательные элементы <wst14:Choice> (содержат элементы списка выбора, обязательный атрибут RefId, дополнительный атрибут Label и вложенный элемент <wst14:Image>);
- <wst14:ContextData> — значение, которое возвращается обратно STS-сервису в ответе, используется как cookie STS-сервисом. Имеет обязательный атрибут RefId.

Ответ на требование <wst14:InteractiveChallenge> содержит элемент <wst14:InteractiveChallengeResponse> с дочерними элементами <wst14:TextChallengeResponse> (введенный пользователем текст), <wst14:ChoiceChallengeResponse> (выбранный элемент, указанный вложенным элементом <wst14:ChoiceSelected>) и <wst14:ContextData> (cookie STS-сервиса), имеющими обязательный атрибут RefId (идентификатор соответствующего элемента требования).

В некоторых случаях в запросах и ответах <wst:RequestSecurityToken> и <wst:RequestSecurityTokenResponse> могут пересылаться бинарные данные, используемые для подписи и шифрования. Для этих целей спецификация определяет элемент <wst:BinaryExchange>, имеющий обязательные атрибуты ValueType (URI-идентификатор типа данных) и EncodingType (URI-идентификатор кодировки данных).

В запросах и ответах <wst:RequestSecurityToken> и <wst:RequestSecurityTokenResponse> также могут пересылаться первоначальные ключи, используемые для создания производных ключей. Для такого обмена спецификация определяет два элемента: <wst:RequestKET> (указывает, что требуется послать в ответ ключ) и <wst:KeyExchangeToken> (содержит ключ).

После обмена первоначальными ключами STS-сервис может послать запрашивающему Web-сервису информацию, подтверждающую правильность создания производного ключа. Для этого используется элемент <wst:RequestSecurityTokenResponseCollection>, который включает в себя следующие элементы:

- RSTR-ответ <wst:RequestSecurityTokenResponse> содержит маркер защиты, созданный в результате обмена;

- RSTR-ответ `<wst:RequestSecurityTokenResponse>` содержит элемент `<wst:Authenticator>`, передающий информацию о производном ключе. Информация о производном ключе может передаваться с помощью элемента `<wst:CombinedHash>`, значением которого является величина в формате Base64 выражения `P_SHA1(computed-key, H + "AUTH-HASH")`. Первый и второй RSTR-ответы `<wst:RequestSecurityTokenResponse>` имеют атрибут `Context` (URI-идентификатор), позволяющий установить соответствие между информацией о производном ключе и созданным маркером защиты.

Web-сервис может запрашивать маркеры защиты у STS-сервиса для другого Web-сервиса. Для указания информации о Web-сервисе, для которого запрашивается маркер защиты, в RST-запрос `<wst:RequestSecurityToken>`, помимо элементов `<wst:TokenType>` и `<wst:RequestType>`, с помощью элемента `<wsse:SecurityTokenReference>` или `<wsa:EndpointReference>` включается элемент `<wst:OnBehalfOf>`, определяющий Web-сервис, который нуждается в маркере защиты. При этом, в RST-запрос `<wst:RequestSecurityToken>` также может включаться элемент `<wst:Issuer>`, указывающий STS-сервис, выпустивший маркеры защиты для данного сообщения. Элемент `<wst:Issuer>` определяет STS-сервис, используя элемент `<wsa:EndpointReference>`.

Для детализации типа запрашиваемых ключей и алгоритмов в RST-запросе спецификация определяет следующие дочерние элементы элемента `<wst:RequestSecurityToken>`:

- `<wst:AuthenticationType>` — URI-идентификатор желаемого типа аутентификации (например, `http://schemas.xmlsoap.org/ws/2006/12/authorization/authntypes/Ssl`, спецификация WS-Federation);
- `<wst:KeyType>` — URI-идентификатор желаемого типа ключа в маркере защиты. Спецификация предопределяет следующие URI-идентификаторы:
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey` — запрашивается маркер защиты публичного ключа;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey` — запрашивается маркер защиты симметричного ключа, по умолчанию;
  - `http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer` — запрашивается маркер защиты без `<wst:RequestedProofToken>`;
- `<wst:KeySize>` — целое число, указывающее желаемый битовый размер ключа;
- `<wst:SignatureAlgorithm>` — URI-идентификатор алгоритма цифровой подписи (спецификация XML-Signature), который желательно использовать в запрашиваемом маркере защиты для его подписи;
- `<wst:EncryptionAlgorithm>` — URI-идентификатор алгоритма шифрования (спецификация XML-Encrypt), который желательно использовать в запрашиваемом маркере защиты для его шифрования;
- `<wst:CanonicalizationAlgorithm>` — URI-идентификатор алгоритма канонизации XML-документов (спецификация XML-Signature), который желательно использовать в запрашиваемом маркере защиты для его подписи;

- `<wst:ComputedKeyAlgorithm>` — URI-идентификатор желаемого алгоритма получения производного ключа для элемента `<wst:RequestedProofToken>`;
- `<wst:Encryption>` — маркер защиты (или ссылка на него), который используется для шифрования возвращаемого маркера защиты;
- `<wst:ProofEncryption>` — маркер защиты (или ссылка на него), который используется для шифрования POP-маркера;
- `<wst:KeyWrapAlgorithm>` — URI-идентификатор алгоритма, в котором симметричный ключ используется для шифрования асимметричного ключа, с помощью которого STS-сервис зашифровывает маркер защиты для запрашиваемого Web-сервиса;
- `<wst:UseKey>` — маркер защиты (или ссылка на него), описывающий POP-ключ, который должен использоваться в возвращаемом маркере защиты, вместо того, чтобы создавать новый ключ. Атрибут `Sig` может указывать идентификатор цифровой подписи маркера защиты;
- `<wst:SignWith>` — URI-идентификатор желаемого алгоритма цифровой подписи с использованием POP-ключа, который запрашивающий Web-сервис намерен применять для возвращаемого маркера защиты. Как правило, данный алгоритм требуется политикой запрашиваемого Web-сервиса;
- `<wst:EncryptWith>` — URI-идентификатор желаемого алгоритма шифрования с использованием POP-ключа, который запрашивающий Web-сервис намерен применять для возвращаемого маркера защиты. Как правило, данный алгоритм требуется политикой запрашиваемого Web-сервиса.

Спецификация также определяет ряд дочерних элементов элемента `<wst:RequestSecurityToken>` для делегирования и перенаправления запроса маркера защиты:

- `<wst:DelegateTo>` — сторонний Web-сервис, который будет использовать созданный маркер защиты, как третья сторона, с помощью маркера защиты или ссылки на него. Созданный маркер защиты будет делегирован другому Web-сервису;
- `<wst:Forwardable>` — `true` (по умолчанию) или `false`; если `true`, тогда созданный маркер защиты может быть перенаправлен;
- `<wst:Delegatable>` — `true` или `false` (по умолчанию); если `true`, тогда созданный маркер защиты может быть делегирован;
- `<wst:ActAs>` — указание на то, что желательно, чтобы созданный маркер защиты содержал информацию о Web-сервисе, посылающем запрос маркера защиты, и о Web-сервисе, для которого данный маркер защиты создается. Элемент `<wst:ActAs>` содержит информацию о Web-сервисе, для которого данный маркер защиты создается посредством маркера защиты или ссылки на него.

Для передачи политики Web-сервиса спецификация определяет дочерние элементы `<wsp:Policy>` и `<wsp:PolicyReference>` элемента `<wst:RequestSecurityToken>`, которые определяют политику или ссылку на нее согласно спецификации WS-Policy.

В элементе `<wst:RequestSecurityToken>` может передаваться информация о сторонах, которые имеют право использовать запрашиваемый маркер защиты. Это делается с помощью элемента `<wst:Participants>`, имеющего дочерние элементы `<wst:Primary>` (указывает основного пользователя маркера доступа, используя маркер доступа или ссылку на конечную точку) и `<wst:Participant>` (указывает дополнительного пользователя маркера доступа, используя маркер доступа или ссылку на конечную точку).

Основные отличия спецификаций WS-Trust версий 1.4, 1.3 и 1.2:

- префикс `wst` является префиксом пространства имен:
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512` — спецификации WS-Trust 1.4 и WS-Trust 1.3;
  - `http://schemas.xmlsoap.org/ws/2005/02/trust` — спецификация WS-Trust 1.2;
- спецификация WS-Trust 1.4 определяет новые элементы `<wst14:InteractiveChallenge>` и `<wst:ActAs>`;
- спецификации WS-Trust 1.4 и WS-Trust 1.3 определяют следующие значения действия `Action` спецификации WS-Addressing:
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Cancel`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Cancel`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/CancelFinal`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/STSCancel`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Issue`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Renew`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Renew`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/RenewFinal`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Validate`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Validate`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/ValidateFinal`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/KET`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/KET`;
  - `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/KETFinal`;
- спецификация WS-Trust 1.2 определяет следующие значения действия `Action` спецификации WS-Addressing:
  - `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue`;
  - `http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue`;

- http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Renew;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Renew;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Cancel;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Cancel;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Validate;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Validate;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RST/KET;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/KET;

❒ спецификация WS-Trust 1.2 не определяет следующие элементы:

- в RST-запросе элемент <wst:SecondaryParameters>;
- в RST-запросе подтверждения элемент <wst:ValidateTarget>;
- в RST-запросе ключей и алгоритмов определенного типа элемент <wst:KeyWrapAlgorithm>.

WS-SecureConversation

Спецификация WS-Security обеспечивает защиту отдельных SOAP-сообщений с помощью передачи в них маркеров защиты, используемых для цифровой подписи и шифрования SOAP-сообщений. Спецификация WS-Trust описывает стандартный механизм получения такого рода маркеров защиты у доверенных центров. Спецификация WS-SecureConversation расширяет спецификации WS-Security и WS-Trust, определяя получение и использование нового маркера защиты контекста (Security Context Token (SCT)) для создания безопасной сессии обмена многочисленными SOAP-сообщениями между несколькими сторонами с применением сессионных ключей, общими для всех участников сессии.

Маркер защиты контекста SCT представлен элементом <wsc:SecurityContextToken>, принадлежащим пространству имен:

❒ http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512 — спецификации WS-SecureConversation версий 1.4 и 1.3;

❒ http://schemas.xmlsoap.org/ws/2005/02/sc — спецификация WS-SecureConversation версии 1.2.

При использовании маркера защиты <wsc:SecurityContextToken> URI-идентификатор типа маркера защиты, указываемый в элементах спецификаций WS-Security и WS-Trust, принимает значение http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct (WS-SecureConversation 1.4 и 1.3) или http://schemas.xmlsoap.org/ws/2005/02/sc/sct (WS-SecureConversation 1.2).

Примечание

URI-идентификатор типа маркера защиты используется в спецификации WS-Security как значение атрибута wsse11:TokenType элемента <wsse:SecurityTokenReference>, зна-

чение атрибута `ValueType` дочернего элемента `<wsse:Reference>` элемента `<wsse:SecurityTokenReference>`, значение атрибута `ValueType` дочернего элемента `<wsse:KeyIdentifier>` элемента `<wsse:SecurityTokenReference>`. В спецификации WS-Trust URI-идентификатор типа маркера защиты используется как значение элемента `<wst:TokenType>`.

Маркер защиты `<wsc:SecurityContextToken>` не допускает ссылки на себя с помощью идентификатора ключа (элемент `<wsse:KeyIdentifier>`) или имени ключа (элемент `<ds:KeyName>`).

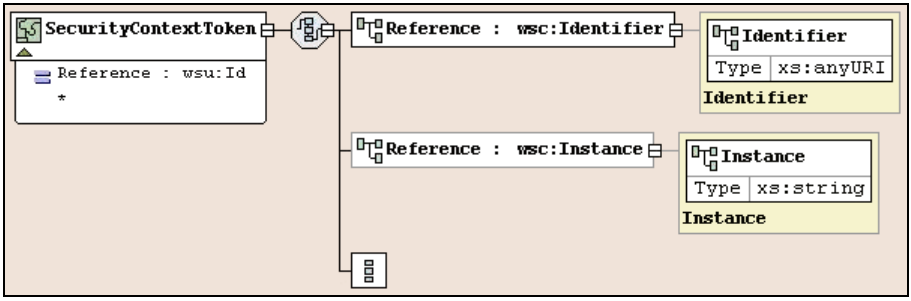


Рис. 2.8. Общая схема элемента `<wsc:SecurityContextToken>`

Элемент `<wsc:SecurityContextToken>` (рис. 2.8) содержит следующие атрибуты и дочерние элементы:

- необязательный атрибут `wsu:Id` — идентификатор маркера защиты контекста в пределах XML-документа, используемый для ссылок на маркер защиты контекста с помощью дочернего элемента `<wsse:Reference>` элемента `<wsse:SecurityTokenReference>`;
- обязательный элемент `<wsc:Identifier>` — URI-идентификатор контекста защиты, уникальный во времени и пространстве, используемый для ссылок на маркер защиты контекста с помощью дочернего элемента `<wsse:Reference>` элемента `<wsse:SecurityTokenReference>`;
- необязательный элемент `<wsc:Instance>` — идентификатор экземпляра контекста защиты при его возобновлении.

Спецификация WS-SecureConversation описывает три различных механизма определения контекста защиты для сессии взаимодействия различных участников.

Первый механизм определяет сценарий, в котором одна из сторон запрашивает STS-сервис для выдачи SCT-маркера защиты контекста `<wsc:SecurityContextToken>`, используя RST-запрос `<wst:RequestSecurityToken>`, в котором элемент `<wst:TokenType>` имеет значение `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct`, а также маркеры защиты, например X.509-сертификат, позволяющие произвести STS-сервисом аутентификацию/авторизацию запрашивающей стороны. При этом RST-запрос `<wst:RequestSecurityToken>` может содержать элемент `<wsp:AppliesTo>`, указывающий адреса конечных точек Web-сервисов других сторон, или STS-сервис предварительно уже обладает информацией об адресах конечных точек всех участников. В ответ STS-сервис рассылает всем

участникам сессии RSRC-ответ `<wst:RequestSecurityTokenResponseCollection>`, содержащий элемент `<wst:RequestSecurityTokenResponse>` с SCT-маркером защиты контекста `<wsc:SecurityContextToken>` и ключ `<wst:RequestedProofToken>` для цифровой подписи и шифрования SOAP-сообщений, участвующих в обмене между участниками сессии. Далее начинается защищенное взаимодействие.

Второй механизм определяет сценарий, в котором SCT-маркер создается одной из сторон, которой доверено создание SCT-маркера. Далее сторона, создавшая SCT-маркер, рассылает его и ключ, информация о котором содержится в элементе `<wst:RequestedProofToken>`, с помощью незапрашиваемого RSR-ответа `<wst:RequestSecurityTokenResponse>`, снабженного цифровой подписью рассылающей стороны. Атрибут `Context` указывает предназначение такого RSR-ответа `<wst:RequestSecurityTokenResponse>`. После этого начинается защищенная сессия обмена SOAP-сообщениями, содержащими заголовок `<wsse:Security>` с SCT-маркером, с которым связан ключ, используемый для защиты SOAP-сообщений сессии.

Третий механизм определяет сценарий, в котором одна из сторон посылает RST-запрос `<wst:RequestSecurityToken>` другой стороне. В результате обмена RSR-ответами `<wst:RequestSecurityTokenResponse>`, содержащими элементы `<wst:SignChallenge>`, `<wst14:InteractiveChallenge>`, `<wst:BinaryExchange>`, `<wst:RequestKET>` и `<wst:KeyExchangeToken>`, запрашивающей стороне посылается RSR-ответ, передающий элемент `<wst:RequestedSecurityToken>` с SCT-маркером и ключом `<wst:RequestedProofToken>`. После этого начинается защищенная сессия.

При совместном использовании спецификаций WS-Trust и WS-SecureConversation для действия `Action` спецификации WS-Addressing применяются значения `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT` и `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT`.

После создания защищенного контекста сессии может возникнуть необходимость изменения его параметров, например, дополнение его другими маркерами, используемыми для защиты SOAP-сообщений сессии. В этом случае для действия `Action` спецификации WS-Addressing применяются значения `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Amend` и `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Amend` в RST-запросах и RSTR-ответах исправления контекста защиты. В RST-запросе исправления контекста защиты в заголовке `<wsse:Security>` содержится маркер защиты, несущий информацию о дополнительном ключе, а также SCT-маркер `<wsc:SecurityContextToken>`. Ключом `<wst:RequestedProofToken>`, связанным с SCT-маркером, подписываются тело и заголовки SOAP-сообщения RST-запроса, а дополнительный ключ используется для цифровой подписи поверх подписи ключом `<wst:RequestedProofToken>`. Элемент `<wst:RequestType>` принимает при этом значение `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue`. RSTR-ответ исправления контекста защиты передает элемент `<wst:RequestedSecurityToken>` с SCT-маркером.

Если требуется увеличение времени действия SCT-маркера, тогда для действия `Action` спецификации WS-Addressing применяются значения `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew` и `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew` в RST-запросах и RSTR-ответах возобновления

контекста защиты. При RST-запросе на возобновление SCT-маркера запрашивающая сторона проходит повторную аутентификацию, а RST-запрос `<wst:RequestSecurityToken>` содержит элемент `<wst:RequestType>` со значением `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew`, элемент `<wst:RenewTarget>`, указывающий SCT-маркер `<wsse:SecurityTokenReference>`, и элемент `<wst:Lifetime>` с запрашиваемым временем действия SCT-маркера. RSTR-ответ `<wst:RequestSecurityTokenResponse>` содержит элемент `<wst:RequestedSecurityToken>` с SCT-маркером `<wsc:SecurityContextToken>` и элемент `<wst:Lifetime>`, определяющий новое время жизни SCT-маркера.

В случае если требуется прекращение действия SCT-маркера, для действия `Action` спецификации WS-Addressing применяются значения `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Cancel` и `http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Cancel` в RST-запросах и RSTR-ответах прекращения контекста защиты. В RST-запросе `<wst:RequestSecurityToken>` прекращения действия SCT-маркера элемент `<wst:RequestType>` имеет значение `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel`, а элемент `<wst:CancelTarget>` указывает SCT-маркер. RSTR-ответ прекращения действия SCT-маркера содержит элемент `<wst:RequestedTokenCancelled/>`, указывающий успешное прекращение контекста защиты.

Ключ, связанный с SCT-маркером, может напрямую использоваться для цифровой подписи и шифрования SOAP-сообщений или быть первичным ключом для создания производных ключей защищенной сессии. Производные ключи защищенной сессии могут также создаваться с помощью других маркеров защиты. При использовании производных ключей спецификация определяет элемент `<wsc:DerivedKeyToken>` (рис. 2.9), имеющий следующие атрибуты и дочерние элементы:

- необязательный атрибут `wsu:Id` — идентификатор элемента;
- необязательный атрибут `Algorithm` — URI-идентификатор алгоритма создания производных ключей. По умолчанию атрибут принимает значение `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk/p_sha1`, определяющее алгоритм `P_SHA1(secret, label + seed)`, где `secret` — первичный ключ, `label` — метка запрашивающего Web-сервиса плюс метка STS-сервиса, `seed` — случайное значение;
- необязательный элемент `<wsse:SecurityTokenReference>` — маркер защиты, с помощью которого создается производный ключ;
- необязательный элемент `<wsc:Properties>` — метаданные, связанные с производным ключом и используемые при создании других производных ключей из указанного производного ключа. Элемент `<wsc:Properties>` имеет следующие дочерние элементы:
  - необязательный элемент `<wsc:Name>` — URI-имя производного ключа;
  - необязательный элемент `<wsc:Nonce>` — случайное значение;
  - необязательный элемент `<wsc:Label>` — метка;

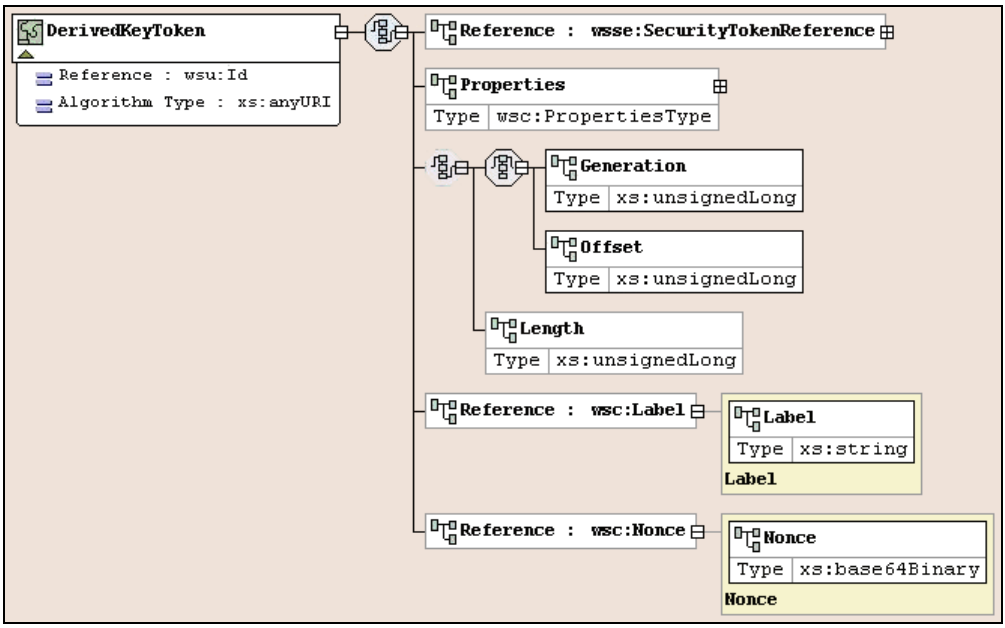


Рис. 2.9. Общая схема элемента `<wsc:DerivedKeyToken>`

- необязательный элемент `<wsc:Offset>` — если производный ключ имеет фиксированную длину, то указывает, с какого байта в сгенерированном значении нужно считывать производный ключ, по умолчанию значение элемента равно `0`. Например, при значении элемента, равном `0`, производный ключ берется с 0-го по 32-й байты, если не указан размер ключа;
- необязательный элемент `<wsc:Length>` — длина ключа;
- необязательный элемент `<wsc:Generation>` используется вместо элемента `<wsc:Offset>` и указывает, с какого цикла по размеру ключа нужно считывать производный ключ в сгенерированном значении, т. е. `offset = (generation) * fixed_size`;
- необязательный элемент `<wsc:Label>` — метка для создания производного ключа, по умолчанию значение элемента — `WS-SecureConversationWS-Secure Conversation`;
- необязательный элемент `<wsc:Nonce>` — случайное значение в формате Base64 для создания производного ключа.

Спецификация определяет URI-идентификатор типа маркера `<wsc:DerivedKeyToken>` как `http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk` (WS-SecureConversation версий 1.4 и 1.3) или `http://schemas.xmlsoap.org/ws/2005/02/sc/dk` (WS-SecureConversation версии 1.2).

В SOAP-сообщении элемент `<wsc:DerivedKeyToken>` передается в заголовке `<wsse:Security>` вместе с маркером защиты, на основе которого был создан данный производный ключ. Элемент `<ds:KeyInfo>` при использовании цифровой подписи или шифрования ссылается на производный ключ с помощью элемента `<wsse:`

SecurityTokenReference>, указывающего идентификатор элемента <wsc:DerivedKeyToken>. Такой механизм называется Explicit Derived Keys (явные производные ключи).

Спецификация также определяет механизм Implied Derived Keys (подразумеваемые производные ключи) неявной ссылки элемента <ds:KeyInfo> на производный ключ. При этом вложенный элемент <wsse:SecurityTokenReference> элемента <ds:KeyInfo> содержит атрибуты wsc:Nonce (аналог элемента <wsc:Nonce>) и wsc:Length (аналог элемента <wsc:Length>) и указывает не идентификатор элемента <wsc:DerivedKeyToken>, которого нет в заголовке <wsse:Security>, а идентификатор маркера защиты, на основе которого создается производный ключ.

Основные отличия спецификаций WS-SecureConversation версий 1.4, 1.3 и 1.2:

- пространство имен спецификации WS-SecureConversation версий 1.4 и 1.3 — http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512;
- пространство имен спецификации WS-SecureConversation версии 1.2 — http://schemas.xmlsoap.org/ws/2005/02/sc;
- URI-идентификатор типа маркера защиты контекста спецификации WS-SecureConversation версий 1.4 и 1.3 — http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct;
- URI-идентификатор типа маркера защиты контекста спецификации WS-SecureConversation версии 1.2 — http://schemas.xmlsoap.org/ws/2005/02/sc/sct;
- спецификация WS-SecureConversation версий 1.4 и 1.3 определяет следующие действия Action спецификации WS-Addressing:
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Amend;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Amend;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Cancel;
  - http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Cancel;
- спецификация WS-SecureConversation версии 1.2 определяет следующие действия Action спецификации WS-Addressing:
  - http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT;
  - http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT;
  - http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Amend;
  - http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Amend;
  - http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Renew;
  - http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Renew;

- http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel;
- http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/SCT/Cancel;

- URI-идентификатор типа маркера производного ключа спецификации WS-SecureConversation версий 1.4 и 1.3 — http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/dk;
- URI-идентификатор типа маркера производного ключа спецификации WS-SecureConversation версии 1.2 — http://schemas.xmlsoap.org/ws/2005/02/sc/dk;
- спецификация WS-SecureConversation версий 1.4, 1.3 и 1.2 поддерживает спецификацию WS-Trust версий 1.4, 1.3 и 1.2 соответственно.

WS-SecurityPolicy

Спецификация WS-SecurityPolicy определяет, каким образом спецификации WS-Security, WS-Trust и WS-SecureConversation используются совместно со спецификацией WS-Policy для выражения политик Web-сервисов, утверждающих определенные требования и характеристики взаимодействия с Web-сервисами.

Спецификация WS-SecurityPolicy определяет утверждения политики, в том числе относящиеся к системе безопасности технологии Web-сервисов.

Требование утверждения политики Web-сервиса, относящееся к защите сообщений, может удовлетворяться различными способами: включением в SOAP-сообщение заголовка <wsse:Security>, содержащего информацию о маркерах защиты, цифровой подписи и шифровании SOAP-сообщения; передачей сообщения по защищенному протоколу и др.

Утверждения спецификации WS-SecurityPolicy принадлежат к пространству имен:

- http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802 — спецификация WS-SecurityPolicy версии 1.3 (префикс sp13);
- http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 — спецификация WS-SecurityPolicy версий 1.3 и 1.2;
- http://schemas.xmlsoap.org/ws/2005/07/securitypolicy — спецификация WS-SecurityPolicy версии 1.1.

Утверждение <sp:SignedParts> (пространство имен http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702) определяет части SOAP-сообщения вне заголовка <wsse:Security>, требующие защиты целостности с помощью цифровой подписи. Утверждение не обязывает указывать части сообщения, оно требует, в случае присутствия, защиты их целостности.

Элемент <sp:SignedParts> может иметь следующие дочерние элементы:

- <sp:Body> определяет, что тело сообщения должно быть защищено;
- <sp:Header> указывает заголовок сообщения, требующий защиты, с помощью обязательного атрибута Namespace (пространство имен заголовка) и дополнительного атрибута Name (локальное имя заголовка);

- `<sp:Attachments>` предполагает наличие элемента и указывает, что все вложения SOAP-сообщения должны быть защищены. Элемент `<sp:Attachments>` может содержать элементы `<sp13:ContentSignatureTransform/>` и `<sp13:AttachmentCompleteSignatureTransform/>` (пространство имен `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802`). Элемент `<sp13:ContentSignatureTransform/>` указывает, что только содержимое вложения снабжается цифровой подписью, а элемент `<sp13:AttachmentCompleteSignatureTransform/>` — что вместе с содержимым вложения подписываются и его заголовки `Content-Description`, `Content-Disposition`, `Content-ID`, `Content-Location` и `Content-Type`.

Утверждение `<sp:SignedElements>` определяет элементы SOAP-сообщения, требующие защиты целостности. Элемент `<sp:SignedElements>` может иметь атрибут `XPathVersion` (URI-идентификатор используемой версии XPath, по умолчанию применяется XPath 1.0) и содержит следующие элементы:

- `<sp:XPath>` — XPath-выражение, которое определяет элементы, требующие защиты;
- `<sp13:Xpath2>` — выражение XPath 2.0, которое определяет элементы, требующие защиты. Обязательный атрибут `Filter` задает выражение фильтрации.

Утверждение `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования в случае их присутствия. Элемент `<sp:EncryptedParts>` может иметь следующие дочерние элементы:

- `<sp:Body>` указывает, что для тела сообщения должна быть обеспечена конфиденциальность. При этом тип зашифрованных данных задается как `http://www.w3.org/2001/04/xmlenc#Content`;
- `<sp:Header>` указывает заголовок сообщения, требующий конфиденциальности, с помощью обязательного атрибута `Namespace` (пространство имен заголовка) и дополнительного атрибута `Name` (локальное имя заголовка);
- `<sp:Attachments>` задает наличие элемента и указывает, что все вложения SOAP-сообщения должны быть защищены.

Утверждение `<sp:EncryptedElements>` определяет элементы сообщения, требующие конфиденциальности. При этом тип зашифрованных данных указывается как `http://www.w3.org/2001/04/xmlenc#Element`. Элемент `<sp:EncryptedElements>` может иметь атрибут `XPathVersion` (URI-идентификатор используемой версии XPath, по умолчанию используется XPath 1.0) и содержит элемент `<sp:XPath>`, который включает в себя XPath-выражение, определяющее элементы, требующие защиты.

Утверждение `<sp:ContentEncryptedElements>` определяет элементы сообщения, содержимое которых требует конфиденциальности. При этом тип зашифрованных данных указывается как `http://www.w3.org/2001/04/xmlenc#Content`. Элемент `<sp:ContentEncryptedElements>` может иметь атрибут `XPathVersion` (URI-идентификатор используемой версии XPath, по умолчанию применяется XPath 1.0) и содержит элемент `<sp:XPath>`, который включает в себя XPath-выражение, определяющее элементы, содержимое которых требует защиты.

Утверждение `<sp:RequiredElements>` определяет блоки заголовка, которые должно содержать сообщение. Элемент `<sp:RequiredElements>` может иметь атрибут `XPathVersion` (URI-идентификатор используемой версии XPath, по умолчанию применяется XPath 1.0) и содержит элемент `<sp:XPath>`, который включает в себя XPath-выражение, определяющее требуемые заголовки.

Утверждение `<sp:RequiredParts>` является альтернативой утверждению `<sp:RequiredElements>` и также определяет блоки заголовка, которые должно содержать сообщение, используя вложенный элемент `<sp:Header>`, имеющий обязательные атрибуты `Name` (указывает локальное имя заголовка) и `Namespace` (указывает пространство имен заголовка).

Утверждение `<sp:UsernameToken>` требует включения в сообщение маркера защиты `<wsse:UsernameToken>`. Элемент `<sp:UsernameToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение. Спецификация предопределяет следующие возможные значения атрибута `sp:IncludeToken`:
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never` — маркер защиты не включается в сообщение, а используется внешняя ссылка на него;
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Once` — маркер защиты должен быть включен только в одно сообщение, посылаемое запрашиваемой стороне, при этом в данном сообщении используются только внутренние ссылки на маркер защиты. Все остальные сообщения используют внешние ссылки на маркер защиты;
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient` — маркер защиты должен быть включен во все сообщения, посылаемые запрашиваемой стороне. Маркер защиты не включается в ответные сообщения;
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToInitiator` — маркер защиты должен быть включен во все сообщения, посылаемые запрашивающей стороне. Маркер защиты не включается в сообщения, посылаемые запрашиваемой стороне;
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Always` — маркер защиты должен быть включен во все сообщения, участвующие в обмене между двумя сторонами. Данное значение используется по умолчанию;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;

- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:NoPassword/>` указывает, что элемент `<wsse:Password>` не должен присутствовать в элементе `<wsse:UsernameToken>`;
  - `<sp:HashPassword/>` — альтернатива элементу `<sp:NoPassword/>`, указывает, что элемент `<wsse:Password>` должен присутствовать в элементе `<wsse:UsernameToken>`. При этом атрибут `Type` элемента `<wsse:Password>` должен иметь значение `PasswordDigest`, которое указывает, что пароль зашифрован с помощью формулы `Base64(SHA-1(nonce + created + password))`;
  - `<sp13:Created/>` — альтернатива элементам `<sp:NoPassword/>` и `<sp:HashPassword/>`, указывает, что в случае значения `PasswordText` (по умолчанию; определяет, что пароль передается как простой текст) атрибута `Type` элемента `<wsse:Password>` необходимо наличие элемента `<wsse:Created>`;
  - `<sp13:Nonce/>` в паре с элементом `<sp13:Created/>` указывает необходимость присутствия элемента `<wsse:Nonce>` в элементе `<wsse:UsernameToken>`;
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - `<sp:RequireImpliedDerivedKeys/>` — альтернатива элементу `<sp:RequireDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
  - `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:RequireDerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
  - `<sp:WssUsernameToken10/>` — для маркера защиты `<wsse:UsernameToken>` указывает использование спецификации UsernameTokenProfile версии 1.0;
  - `<sp:WssUsernameToken11/>` — альтернатива элементу `<sp:WssUsernameToken10/>`, для маркера защиты `<wsse:UsernameToken>` указывает использование спецификации UsernameTokenProfile версии 1.1.

Утверждение `<sp:IssuedToken>` требует генерации маркера защиты согласно спецификации WS-Trust. Элемент `<sp:IssuedToken>` содержится в политике запрашиваемого Web-сервиса, которая передается запрашивающему Web-сервису для создания RST-запроса маркера защиты STS-сервису.

Элемент `<sp:IssuedToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- обязательный элемент `<sp:RequestSecurityTokenTemplate>` содержит элементы, которые должны быть скопированы в элемент `<wst:SecondaryParameters>` RST-запроса маркера защиты STS-сервису. Элемент `<sp:RequestSecurityToken Template>` может иметь атрибут `TrustVersion`, указывающий URI-идентификатор пространства имен используемой версии спецификации WS-Trust;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:Derived KeyToken>` и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - `<sp:RequireImpliedDerivedKeys/>` — альтернатива элементу `<sp:Require DerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
  - `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:Require DerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
  - `<sp:RequireExternalReference/>` указывает, что STS-сервис при генерации маркера защиты обязан дать внешние ссылки на него, которые и должны в дальнейшем использоваться;
  - `<sp:RequireInternalReference/>` указывает, что STS-сервис при генерации маркера защиты должен дать внутренние ссылки на него, которые нужно в дальнейшем использовать.

Утверждение `<sp:X509Token>` требует включения в сообщение маркера защиты `<wsse:BinarySecurityToken>`, содержащего X509-сертификат. Элемент `<sp:X509Token>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - `<sp:RequireImpliedDerivedKeys/>` — альтернатива элементу `<sp:RequireDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
  - `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:RequireDerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
  - `<sp:RequireKeyIdentifierReference/>` указывает, что при ссылке на маркер защиты должен использоваться элемент `<wsse:KeyIdentifier>`;
  - `<sp:RequireIssuerSerialReference/>` указывает, что при ссылке на маркер защиты должен использоваться элемент `<ds:X509IssuerSerial>`;
  - `<sp:RequireEmbeddedTokenReference/>` указывает, что при ссылке на маркер защиты должен использоваться элемент `<wsse:Embedded>`;
  - `<sp:RequireThumbprintReference/>` указывает, что при ссылке на маркер защиты должен использоваться элемент `<wsse:KeyIdentifier>` со значением атрибута `ValueType` равным `http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#ThumbPrintSHA1`;
  - `<sp:WssX509V3Token10/>` указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` спецификации X.509 Certificate Token Profile версии 1.0;

- `<sp:WssX509Pkcs7Token10/>` указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7` спецификации X.509 Certificate Token Profile версии 1.0;
- `<sp:WssX509PkiPathV1Token10/>` указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1` спецификации X.509 Certificate Token Profile версии 1.0;
- `<sp:WssX509V1Token11/>` указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#x509v1` спецификации X.509 Certificate Token Profile версии 1.1;
- `<sp:WssX509V3Token11/>` указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` спецификации X.509 Certificate Token Profile версии 1.1;
- `<sp:WssX509Pkcs7Token11/>` — указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7` спецификации X.509 Certificate Token Profile версии 1.1;
- `<sp:WssX509PkiPathV1Token11/>` — указывает требование значения атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` как `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1` спецификации X.509 Certificate Token Profile версии 1.1.

Утверждение `<sp:KerberosToken>` требует включения в сообщение маркера защиты `<wsse:BinarySecurityToken>`, содержащего Kerberos-билет. Элемент `<sp:KerberosToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:Derived`

KeyToken> и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;

- <sp:RequireImpliedDerivedKeys/> — альтернатива элементу <sp:Require DerivedKeys/>, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии <wsc:DerivedKeyToken> и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
- <sp:RequireExplicitDerivedKeys/> — альтернатива элементам <sp:Require DerivedKeys/> и <sp:RequireImpliedDerivedKeys/>, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии <wsc:DerivedKeyToken> и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
- <sp:RequireKeyIdentifierReference/> указывает, что при ссылке на маркер защиты должен использоваться элемент <wsse:KeyIdentifier>;
- <sp:WssKerberosV5ApReqToken11/> указывает требование равенства значения атрибута ValueType элемента <wsse:BinarySecurityToken> соответствующему билету Kerberos Version 5 AP-REQ;
- <sp:WssGssKerberosV5ApReqToken11/> указывает требование равенства значения атрибута ValueType элемента <wsse:BinarySecurityToken> соответствующему билету GSS Kerberos Version 5 AP-REQ.

Утверждение <sp:SpnegoContextToken> требует обмена сообщениями с использованием элемента <wst:BinaryExchange> спецификации WS-Trust для реализации протокола SPNEGO с получением маркера защиты <wsc:SecurityContextToken>.

> ***Примечание***
>
> Протокол SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism) — механизм аутентификации, используемый в том случае, когда ни одна из сторон не знает, какой протокол аутентификации поддерживает другая сторона.

Элемент <sp:SpnegoContextToken> имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут sp:IncludeToken указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент <sp:Issuer> указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента <wsa:EndpointReference> и содержит все его дочерние элементы;
- необязательный элемент <sp:IssuerName> — альтернатива элементу <sp:Issuer>, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент <wst:Claims> — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент <wsp:Policy> указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:

- <sp:RequireDerivedKeys/> — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии <wsc:DerivedKeyToken> и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
- <sp:RequireImpliedDerivedKeys/> — альтернатива элементу <sp:RequireDerivedKeys/>, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии <wsc:DerivedKeyToken> и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
- <sp:RequireExplicitDerivedKeys/> — альтернатива элементам <sp:RequireDerivedKeys/> и <sp:RequireImpliedDerivedKeys/>, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии <wsc:DerivedKeyToken> и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
- <sp:MustNotSendCancel/> указывает, что запросы спецификации WS-SecureConversation SCT/Cancel не поддерживаются;
- <sp:MustNotSendAmend/> указывает, что запросы спецификации WS-SecureConversation SCT/Amend не поддерживаются;
- <sp:MustNotSendRenew/> — указывает, что запросы спецификации WS-SecureConversation SCT/Renew не поддерживаются.

Утверждение <sp:SecurityContextToken> требует включения в сообщение маркера защиты <wsc:SecurityContextToken>. Элемент <sp:SecurityContextToken> имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут sp:IncludeToken указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент <sp:Issuer> указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента <wsa:EndpointReference> и содержит все его дочерние элементы;
- необязательный элемент <sp:IssuerName> — альтернатива элементу <sp:Issuer>, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент <wst:Claims> — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент <wsp:Policy> указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - <sp:RequireDerivedKeys/> — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии <wsc:DerivedKeyToken> и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - <sp:RequireImpliedDerivedKeys/> — альтернатива элементу <sp:RequireDerivedKeys/>, наличие элемента устанавливает обязательность использова-

ния производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;

- `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:RequireDerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
- `<sp:RequireExternalUriReference/>` указывает требование внешних ссылок на маркер защиты;
- `<sp:SC13SecurityContextToken/>` указывает, что использование маркера защиты должно соответствовать спецификации WS-SecureConversation.

Утверждение `<sp:SecureConversationToken>` требует получение маркера защиты `<wsc:SecurityContextToken>`. Элемент `<sp:SecureConversationToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - `<sp:RequireImpliedDerivedKeys/>` — альтернатива элементу `<sp:RequireDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
  - `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:RequireDerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;

- `<sp:RequireExternalUriReference/>` указывает требование внешних ссылок на маркер защиты;
- `<sp:SC13SecurityContextToken/>` указывает, что использование маркера защиты должно соответствовать спецификации WS-SecureConversation;
- `<sp:MustNotSendCancel/>` указывает, что запросы спецификации WS-SecureConversation SCT/Cancel не поддерживаются;
- `<sp:MustNotSendAmend/>` указывает, что запросы спецификации WS-SecureConversation SCT/Amend не поддерживаются;
- `<sp:MustNotSendRenew/>` указывает, что запросы спецификации WS-SecureConversation SCT/Renew не поддерживаются;
- `<sp:BootstrapPolicy>` описывает политику получения маркера защиты с помощью дочернего элемента `<wsp:Policy>`.

Утверждение `<sp:SamlToken>` требует SAML-маркера. Элемент `<sp:SamlToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - `<sp:RequireImpliedDerivedKeys/>` — альтернатива элементу `<sp:RequireDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
  - `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:RequireDerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;

- `<sp:RequireKeyIdentifierReference/>` указывает, что при ссылке на маркер защиты должен использоваться элемент `<wsse:KeyIdentifier>`;
- `<sp:WssSamlV11Token10/>` указывает, что маркер защиты должен соответствовать спецификациям SAML версии 1.1 и SAMLTokenProfile версии 1.0;
- `<sp:WssSamlV11Token11/>` указывает, что маркер защиты должен соответствовать спецификациям SAML версии 1.1 и SAMLTokenProfile версии 1.1;
- `<sp:WssSamlV20Token11/>` — указывает, что маркер защиты должен соответствовать спецификациям SAML версии 2.0 и SAMLTokenProfile версии 1.1.

Утверждение `<sp:RelToken>` требует REL-маркера. Элемент `<sp:RelToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:RequireDerivedKeys/>` — наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения обоих механизмов использования производного ключа — Explicit Derived Keys и Implied Derived Keys;
  - `<sp:RequireImpliedDerivedKeys/>` — альтернатива элементу `<sp:RequireDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Implied Derived Keys;
  - `<sp:RequireExplicitDerivedKeys/>` — альтернатива элементам `<sp:RequireDerivedKeys/>` и `<sp:RequireImpliedDerivedKeys/>`, наличие элемента устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>` и возможность применения только одного механизма использования производного ключа Explicit Derived Keys;
  - `<sp:RequireKeyIdentifierReference/>` указывает, что при ссылке на маркер защиты должен использоваться элемент `<wsse:KeyIdentifier>`;
  - `<sp:WssRelV10Token10/>` указывает, что должен использоваться REL-маркер 1.0 согласно спецификации RELTokenProfile 1.0;

- `<sp:WssRelV20Token10/>` указывает, что должен использоваться REL-маркер 2.0 согласно спецификации RELTokenProfile 1.0;
- `<sp:WssRelV10Token11/>` указывает, что должен использоваться REL-маркер 1.0 согласно спецификации RELTokenProfile 1.1;
- `<sp:WssRelV20Token11/>` указывает, что должен использоваться REL-маркер 2.0 согласно спецификации RELTokenProfile 1.1.

Утверждение `<sp:HttpsToken>` требует использование HTTPS-протокола. Элемент `<sp:HttpsToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- необязательный элемент `<sp:Issuer>` указывает адрес конечной точки доверенного центра, создающего маркер защиты. Элемент имеет тип элемента `<wsa:EndpointReference>` и содержит все его дочерние элементы;
- необязательный элемент `<sp:IssuerName>` — альтернатива элементу `<sp:Issuer>`, содержит URI-идентификатор доверенного центра, создающего маркер защиты;
- необязательный элемент `<wst:Claims>` — элемент спецификации WS-Trust, определяет набор утверждений, которые необходимо включить в маркер защиты;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать следующие элементы:
  - `<sp:HttpBasicAuthentication/>` — аутентификация запрашивающей стороны должна производиться с помощью логина и пароля;
  - `<sp:HttpDigestAuthentication/>` — аутентификация запрашивающей стороны должна производиться с помощью хэш-значения логина и пароля;
  - `<sp:RequireClientCertificate/>` — аутентификация запрашивающей стороны должна производиться с помощью сертификата.

Утверждение `<sp:KeyValueToken>` требует наличие маркера защиты, содержащего элемент `<ds:KeyValue>` с ключом. Согласно спецификации XML Signature элемент `<ds:KeyValue>` является дочерним элементом элемента `<ds:KeyInfo>`. Элемент `<sp:KeyValueToken>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `sp:IncludeToken` указывает, каким образом маркер защиты включается в сообщение;
- обязательный элемент `<wsp:Policy>` указывает дополнительные требования использования маркера защиты и может содержать элемент `<sp:RsaKeyValue/>`, указывающий, что дочерним элементом элемента `<ds:KeyValue>` должен быть элемент `<ds:RSAKeyValue>` с публичным RSA-ключом.

Утверждение `<sp:AlgorithmSuite>` требует определенного набора алгоритмов для выполнения операций шифрования с симметричным или асимметричным ключом. Этот элемент имеет обязательный элемент `<wsp:Policy>`, который определяет использование набора алгоритмов с помощью дочерних элементов:

- `<sp:Basic256/>` — требование набора алгоритмов: Sha1, Aes256, KwAes256, KwRsaOaep, PSha1L256, PSha1L192, минимальная длина симметричного ключа равна 256;
- `<sp:Basic192/>` — требование набора алгоритмов: Sha1, Aes192, KwAes192, KwRsaOaep, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic128/>` — требование набора алгоритмов: Sha1, Aes128, KwAes128, KwRsaOaep, PSha1L128, минимальная длина симметричного ключа равна 128;
- `<sp:TripleDes/>` — требование набора алгоритмов: Sha1, TripleDes, KwTripleDes, KwRsaOaep, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic256Rsa15/>` — требование набора алгоритмов: Sha1, Aes256, KwAes256, KwRsa15, PSha1L256, PSha1L192, минимальная длина симметричного ключа равна 256;
- `<sp:Basic192Rsa15/>` — требование набора алгоритмов: Sha1, Aes192, KwAes192, KwRsa15, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic128Rsa15/>` — требование набора алгоритмов: Sha1, Aes128, KwAes128, KwRsa15, PSha1L128, минимальная длина симметричного ключа равна 128;
- `<sp:TripleDesRsa15/>` — требование набора алгоритмов: Sha1, TripleDes, KwTripleDes, KwRsa15, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic256Sha256/>` — требование набора алгоритмов: Sha256, Aes256, KwAes256, KwRsaOaep, PSha1L256, PSha1L192, минимальная длина симметричного ключа равна 256;
- `<sp:Basic192Sha256/>` — требование набора алгоритмов: Sha256, Aes192, KwAes192, KwRsaOaep, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic128Sha256/>` — требование набора алгоритмов: Sha256, Aes128, KwAes128, KwRsaOaep, PSha1L128, минимальная длина симметричного ключа равна 128;
- `<sp:TripleDesSha256/>` — требование набора алгоритмов: Sha256, TripleDes, KwTripleDes, KwRsaOaep, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic256Sha256Rsa15/>` — требование набора алгоритмов: Sha256, Aes256, KwAes256, KwRsa15, PSha1L256, PSha1L192, минимальная длина симметричного ключа равна 256;
- `<sp:Basic192Sha256Rsa15/>` — требование набора алгоритмов: Sha256, Aes192, KwAes192, KwRsa15, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:Basic128Sha256Rsa15/>` — требование набора алгоритмов: Sha256, Aes128, KwAes128, KwRsa15, PSha1L128, минимальная длина симметричного ключа равна 128;

- `<sp:TripleDesSha256Rsa15/>` — требование набора алгоритмов: Sha256, TripleDes, KwTripleDes, KwRsa15, PSha1L192, минимальная длина симметричного ключа равна 192;
- `<sp:InclusiveC14N/>` — канонизация XML-документа согласно спецификации Canonical XML Version 1.0, по умолчанию Exclusive XML Canonicalization Version 1.0;
- `<sp:InclusiveC14N11/>` — канонизация XML-документа согласно спецификации Canonical XML 1.1, по умолчанию Exclusive XML Canonicalization Version 1.0;
- `<sp:SOAPNormalization10/>` — SOAP-нормализация согласно спецификации SOAP Version 1.2 Message Normalization;
- `<sp:STRTransform10/>` — механизм STR-преобразования согласно спецификации WS-Security 1.0;
- `<sp:XPath10/>` — использование языка XPath 1.0;
- `<sp:XPathFilter20/>` — использование языка XML-Signature XPath Filter 2.0;
- `<sp:AbsXPath/>` — использование абсолютных путей в XPath-выражениях.

Утверждение `<sp:Layout>` требует определенного порядка элементов в заголовке `<wsse:Security>`. Элемент `<sp:Layout>` имеет обязательный элемент `<wsp:Policy>`, который определяет порядок элементов в заголовке `<wsse:Security>` с помощью следующих дочерних элементов:

- `<sp:Strict/>` указывает, что элементы заголовка `<wsse:Security>` должны быть продекларированы перед их использованием, т. е., например, маркер защиты, используемый для цифровой подписи, должен быть перед элементом `<ds:Signature>`;
- `<sp:Lax/>` указывает, что элементы заголовка `<wsse:Security>` располагаются в произвольном порядке согласно спецификации WS-Security;
- `<sp:LaxTsFirst/>` указывает, что элементы заголовка `<wsse:Security>` располагаются в произвольном порядке согласно спецификации WS-Security, но первым элементом является элемент `<wsu:Timestamp>`;
- `<sp:LaxTsLast/>` указывает, что элементы заголовка `<wsse:Security>` располагаются в произвольном порядке согласно спецификации WS-Security, но последним элементом является элемент `<wsu:Timestamp>`.

Утверждение `<sp:TransportBinding>` требует защиты сообщений с помощью транспортного протокола. Элемент `<sp:TransportBinding>` имеет обязательный элемент `<wsp:Policy>`, определяющий требования защиты сообщений, используя следующие дочерние элементы:

- обязательный элемент `<sp:TransportToken>` содержит элемент `<wsp:Policy>`, определяющий требования к транспортному протоколу, например, с помощью элемента `<sp:HttpsToken>`;
- обязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов защиты сообщения;

- необязательный элемент `<sp:Layout>` определяет порядок расположения элементов в заголовке `<wsse:Security>`;
- необязательный элемент `<sp:IncludeTimestamp/>` требует наличие элемента `<wsu:Timestamp>`.

Утверждение `<sp:SymmetricBinding>` содержит требования защиты сообщений как для запрашивающей стороны, так и для запрашиваемой стороны, с помощью цифровой подписи и шифрования согласно спецификации WS-Security. Элемент `<sp:SymmetricBinding>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- необязательный элемент `<sp:EncryptionToken>` содержит элемент `<wsp:Policy>` и определяет требования к шифрованию сообщений с помощью таких дочерних элементов элемента `<wsp:Policy>`, как `<sp:KerberosToken>`, `<sp:IssuedToken>`, `<sp:SecurityContextToken>` и др.;
- необязательный элемент `<sp:SignatureToken>` содержит элемент `<wsp:Policy>` и определяет требования к цифровой подписи сообщений с помощью таких дочерних элементов элемента `<wsp:Policy>`, как `<sp:X509Token>`, `<sp:IssuedToken>`, `<sp:SecurityContextToken>` и др.;
- необязательный элемент `<sp:ProtectionToken>` содержит элемент `<wsp:Policy>` и определяет требования к шифрованию и цифровой подписи сообщений с помощью таких дочерних элементов элемента `<wsp:Policy>`, как `<sp:KerberosToken>`, `<sp:IssuedToken>`, `<sp:SecurityContextToken>` и др.;

Примечание

Элемент `<sp:EncryptionToken>` используется в паре с элементом `<sp:SignatureToken>`, а элемент `<sp:ProtectionToken>` является альтернативой элементам `<sp:EncryptionToken>` и `<sp:SignatureToken>`. Элемент `<sp:SymmetricBinding>` должен содержать элементы `<sp:EncryptionToken>` и `<sp:SignatureToken>` или элемент `<sp:ProtectionToken>`.

- обязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:Layout>` определяет порядок элементов в заголовке `<wsse:Security>`;
- необязательный элемент `<sp:IncludeTimestamp/>` требует наличие элемента `<wsu:Timestamp>`;
- необязательный элемент `<sp:EncryptBeforeSigning/>` указывает, что цифровая подпись сообщения создается после его шифрования, при этом используются симметричные ключи;
- необязательный элемент `<sp:EncryptSignature/>` указывает, что цифровая подпись должна быть зашифрована;
- необязательный элемент `<sp:ProtectTokens/>` указывает, что маркеры защиты, используемые для цифровой подписи, сами должны быть снабжены этой цифровой подписью;

- необязательный элемент `<sp:OnlySignEntireHeadersAndBody/>` указывает, что цифровой подписью должно снабжаться все тело SOAP-сообщения, а применительно к заголовку SOAP-сообщения — полностью весь заголовок, дочерние элементы отдельно не снабжаются цифровой подписью. Исключение составляет заголовок `<wsse:Security>`.

Утверждение `<sp:AsymmetricBinding>` содержит требования защиты сообщений с помощью цифровой подписи и шифрования с применением асимметричных ключей. Элемент `<sp:AsymmetricBinding>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- необязательный элемент `<sp:InitiatorToken>` содержит элемент `<wsp:Policy>` и с помощью его дочерних элементов определяет требования цифровой подписи и шифрования к запрашивающей стороне;
- необязательный элемент `<sp:InitiatorSignatureToken>` содержит элемент `<wsp:Policy>` и с помощью его дочерних элементов определяет требования цифровой подписи к запрашивающей стороне;
- необязательный элемент `<sp:InitiatorEncryptionToken>` содержит элемент `<wsp:Policy>` и с помощью его дочерних элементов определяет требования шифрования к запрашивающей стороне;
- необязательный элемент `<sp:RecipientToken>` содержит элемент `<wsp:Policy>` и с помощью его дочерних элементов определяет требования цифровой подписи и шифрования к запрашиваемой стороне;
- необязательный элемент `<sp:RecipientSignatureToken>` содержит элемент `<wsp:Policy>` и с помощью его дочерних элементов определяет требования цифровой подписи к запрашиваемой стороне;
- необязательный элемент `<sp:RecipientEncryptionToken>` содержит элемент `<wsp:Policy>` и с помощью его дочерних элементов определяет требования шифрования к запрашиваемой стороне;

> ***Примечание***
>
> Элемент `<sp:AsymmetricBinding>` должен содержать элементы `<sp:InitiatorToken>` и `<sp:RecipientToken>` или пары элементов `<sp:InitiatorSignatureToken>` и `<sp:InitiatorEncryptionToken>`, `<sp:RecipientSignatureToken>` и `<sp:RecipientEncryptionToken>`.

- обязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:Layout>` определяет порядок элементов в заголовке `<wsse:Security>`;
- необязательный элемент `<sp:IncludeTimestamp/>` требует наличие элемента `<wsu:Timestamp>`;
- необязательный элемент `<sp:EncryptBeforeSigning/>` указывает, что цифровая подпись сообщения создается после его шифрования, при этом используются асимметричные ключи;

- необязательный элемент `<sp:EncryptSignature/>` указывает, что цифровая подпись должна быть зашифрована;
- необязательный элемент `<sp:ProtectTokens/>` указывает, что маркеры защиты, используемые для цифровой подписи, сами должны быть подписаны этой цифровой подписью;
- необязательный элемент `<sp:OnlySignEntireHeadersAndBody/>` указывает, что цифровой подписью должно снабжаться все тело SOAP-сообщения, а применительно к заголовку SOAP-сообщения — полностью весь заголовок, дочерние элементы отдельно не сопровождаются цифровой подписью. Исключение составляет заголовок `<wsse:Security>`.

Утверждение `<sp:SupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, а также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:SupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:SignedSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, которые для защиты должны быть снабжены цифровой подписью, а также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:SignedSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;

- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:EndorsingSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, имеющих ключи, которые используются для цифровой подписи первоначальной цифровой подписи, а также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:EndorsingSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:SignedEndorsingSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, имеющих ключи, которые используются для цифровой подписи первоначальной цифровой подписи, при этом данные маркеры защиты сами снабжены первоначальной цифровой подписью. Таким образом, маркеры защиты для первоначальной цифровой подписи и маркеры защиты для вторичной цифровой подписи подписывают друг друга. Элемент `<sp:SignedEndorsingSupportingTokens>` также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:SignedEndorsingSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;

- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:SignedEncryptedSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, которые для защиты должны быть зашифрованы и снабжены цифровой подписью, а также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:SignedEncryptedSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:EncryptedSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, которые для защиты должны быть зашифрованы, а также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:EncryptedSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;

- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:EndorsingEncryptedSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, имеющих ключи, которые используются для цифровой подписи первоначальной цифровой подписи, при этом данные маркеры защиты должны быть зашифрованы, а также дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:EndorsingEncryptedSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;
- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:SignedEndorsingEncryptedSupportingTokens>` требует включения в сообщение одного или нескольких маркеров защиты, имеющих ключи, которые используются для цифровой подписи первоначальной цифровой подписи, при этом данные маркеры защиты сами снабжены первоначальной цифровой подписью, а также зашифрованы. Элемент `<sp:SignedEndorsingEncryptedSupportingTokens>` дополнительно определяет цифровую подпись и шифрование для частей сообщения. Элемент `<sp:SignedEndorsingSupportingTokens>` имеет обязательный элемент `<wsp:Policy>`, включающий в себя следующие элементы:

- утверждения для маркеров защиты `<sp:UsernameToken>`, `<sp:IssuedToken>`, `<sp:X509Token>`, `<sp:KerberosToken>`, `<sp:SpnegoContextToken>`, `<sp:SecurityContextToken>`, `<sp:SecureConversationToken>`, `<sp:SamlToken>`, `<sp:RelToken>`, `<sp:HttpsToken>`, `<sp:KeyValueToken>`;
- необязательный элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для защиты сообщений;

- необязательный элемент `<sp:SignedParts>` определяет части сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:SignedElements>` определяет элементы сообщения, требующие цифровой подписи;
- необязательный элемент `<sp:EncryptedParts>` определяет части сообщения, требующие шифрования;
- необязательный элемент `<sp:EncryptedElements>` определяет элементы сообщения, требующие шифрования.

Утверждение `<sp:Wss10>` определяет поддержку свойств спецификации WS-Security версии 1.0. Элемент `<sp:Wss10>` имеет обязательный элемент `<wsp:Policy>`, который может включать в себя следующие элементы:

- `<sp:MustSupportRefKeyIdentifier/>` требует поддержки ссылок на маркеры защиты с помощью элемента `<wsse:KeyIdentifier>`;
- `<sp:MustSupportRefIssuerSerial/>` требует поддержки ссылок на маркеры защиты с помощью элемента `<ds:X509IssuerSerial>`;
- `<sp:MustSupportRefExternalURI/>` требует поддержки внешних ссылок на маркеры защиты;
- `<sp:MustSupportRefEmbeddedToken/>` требует поддержки встроенных ссылок `<wsse:Embedded>`.

Утверждение `<sp:Wss11>` определяет поддержку свойств спецификации WS-Security версии 1.1. Элемент `<sp:Wss11>` имеет обязательный элемент `<wsp:Policy>`, который может включать в себя следующие элементы:

- `<sp:MustSupportRefKeyIdentifier/>` требует поддержки ссылок на маркеры защиты с помощью элемента `<wsse:KeyIdentifier>`;
- `<sp:MustSupportRefIssuerSerial/>` требует поддержки ссылок на маркеры защиты с помощью элемента `<ds:X509IssuerSerial>`;
- `<sp:MustSupportRefExternalURI/>` требует поддержки внешних ссылок на маркеры защиты;
- `<sp:MustSupportRefEmbeddedToken/>` требует поддержки встроенных ссылок `<wsse:Embedded>`;
- `<sp:MustSupportRefThumbprint/>` требует при ссылках поддержки URI-идентификатора `http://docs.oasis-open.org/wss/oasiswss-soap-message-security-1.1#ThumbPrintSHA1`;
- `<sp:MustSupportRefEncryptedKey/>` требует при ссылках поддержки URI-идентификатора `http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKeySHA`;
- `<sp:RequireSignatureConfirmation/>` требует поддержки элемента `<wsse11:SignatureConfirmation>`.

Утверждение `<sp:Trust13>` определяет поддержку свойств спецификации WS-Trust версии 1.3. Элемент `<sp:Trust13>` имеет обязательный элемент `<wsp:Policy>`, который может включать в себя следующие элементы:

- `<sp:MustSupportClientChallenge/>` требует поддержки элемента `<wst:SignChallenge>` для RST-запроса;
- `<sp:MustSupportServerChallenge/>` требует поддержки элемента `<wst:SignChallenge>` для RSTR-ответа;
- `<sp:RequireClientEntropy/>` требует наличия элемента `<wst:Entropy>` в RST-запросе;
- `<sp:RequireServerEntropy/>` требует наличия элемента `<wst:Entropy>` в RSTR-ответе;
- `<sp:MustSupportIssuedTokens/>` требует поддержки заголовка `<wst:IssuedTokens>`;
- `<sp:RequireRequestSecurityTokenCollection/>` требует наличия элемента `<wst:RequestSecurityTokenCollection>`;
- `<sp:RequireAppliesTo/>` требует наличия элемента `<wsp:AppliesTo>` в RST-запросе;
- `<sp13:ScopePolicy15/>` требует для элемента `<wsp:AppliesTo>` пространство имен `http://www.w3.org/ns/ws-policy`;
- `<sp13:MustSupportInteractiveChallenge/>` требует поддержки элемента `<wst14:InteractiveChallenge>`.

Основные отличия версий 1.1, 1.2 и 1.3 спецификации WS-SecurityPolicy:

- пространство имен:
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200802` — спецификация WS-SecurityPolicy версии 1.3;
  - `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702` — спецификация WS-SecurityPolicy версий 1.3 и 1.2;
  - `http://schemas.xmlsoap.org/ws/2005/07/securitypolicy` — спецификация WS-SecurityPolicy версии 1.1;
- спецификация WS-SecurityPolicy версии 1.3 определяет новые элементы:
  - `<sp13:ContentSignatureTransform/>`, `<sp13:AttachmentCompleteSignatureTransform/>` — утверждение `<sp:SignedParts>`;
  - `<sp13:Xpath2>` — утверждение `<sp:SignedElements>`;
  - `<sp13:Created/>`, `<sp13:Nonce/>` — утверждение `<sp:UsernameToken>`;
  - `<sp13:ScopePolicy15/>`, `<sp13:MustSupportInteractiveChallenge/>` — утверждение `<sp:Trust13>`;
- спецификация WS-SecurityPolicy версии 1.1 поддерживает начальные версии спецификаций WS-Policy, WS-Trust, WS-SecureConversation, WS-Addressing;

- спецификация WS-SecurityPolicy версии 1.1 не определяет следующее:
  - элемент `<sp:Attachments/>` — утверждения `<sp:SignedParts>` и `<sp:EncryptedParts>`;
  - утверждение `<sp:ContentEncryptedElements>`;
  - утверждение `<sp:RequiredParts>`;
  - значение `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToInitiator` атрибута `IncludeToken`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:NoPassword/>`, `<sp:HashPassword/>`, `<sp:RequireDerivedKeys/>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>` утверждения `<sp:UsernameToken>`;
  - элементы `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>` утверждения `<sp:IssuedToken>`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireDerivedKeys/>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>` утверждения `<sp:X509Token>`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>` утверждения `<sp:KerberosToken>`;
  - элементы `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>`, `<sp:MustNotSendCancel/>`, `<sp:MustNotSendAmend/>`, `<sp:MustNotSendRenew/>` утверждения `<sp:SpnegoContextToken>`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>`, вместо элемента `<sp:SC13SecurityContextToken/>` — элемент `<sp:SC10SecurityContextToken/>` утверждения `<sp:SecurityContextToken>`;
  - элементы `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>`, вместо элемента `<sp:SC13SecurityContextToken/>` — элемент `<sp:SC10SecurityContextToken/>`, `<sp:MustNotSendCancel/>`, `<sp:MustNotSendAmend/>`, `<sp:MustNotSendRenew/>` утверждения `<sp:SecureConversationToken>`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>` утверждения `<sp:SamlToken>`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:RequireImpliedDerivedKeys/>`, `<sp:RequireExplicitDerivedKeys/>` утверждения `<sp:RelToken>`;
  - элементы `<sp:Issuer>`, `<sp:IssuerName>`, `<wst:Claims>`, `<sp:HttpBasicAuthentication/>`, `<sp:HttpDigestAuthentication/>`, вместо эле-

мента `<sp:RequireClientCertificate/>` — атрибут `RequireClientCertificate` утверждения `<sp:HttpsToken>`;

- утверждение `<sp:KeyValueToken>`;
- элемент `<sp:AbsXPath/>` утверждения `<sp:AlgorithmSuite>`;
- элементы `<sp:InitiatorSignatureToken>`, `<sp:InitiatorEncryptionToken>`, `<sp:RecipientSignatureToken>`, `<sp:RecipientEncryptionToken>` утверждения `<sp:AsymmetricBinding>`;
- утверждения `<sp:SignedEncryptedSupportingTokens>`, `<sp:EncryptedSupportingTokens>`, `<sp:EndorsingEncryptedSupportingTokens>`, `<sp:SignedEndorsingEncryptedSupportingTokens>`;
- вместо утверждения `<sp:Trust13>` — утверждение `<sp:Trust10>`.

WS-Federation

Спецификация WS-Federation 1.1 определяет механизм объединения различных защищенных зон. При таком объединении защищенных зон или доменов сторона, прошедшая аутентификацию в одном домене, получает доступ к ресурсам в другом домене.

> ***Примечание***
> Защищенный домен — это изолированная программная среда, защита которой гарантируется определенным набором политик.

Объединение защищенных доменов позволяет запрашивающей стороне не создавать свои учетные записи в каждом из доменов, а использовать идентификаторы и атрибуты, управляемые своим доменом, для авторизации в другом защищенном домене. Объединение доменов производится в результате установления доверительных отношений между доменами без изменения их политик.

Таким образом, *федерация доменов*, в основе которой лежит установление взаимных доверительных отношений между доменами, — это совместное использование идентификационных данных и политик без их дублирования.

Существуют различные сценарии использования федерации доменов. Один защищенный домен может включать в себя запрашивающую сторону, которая хочет получить доступ к определенному ресурсу, находящемуся в другом защищенном домене, и IP/STS-сервис (Identity Provider/Security Token Service), производящий аутентификацию и выдающий маркеры защиты, которые содержат идентификационные данные. Запрашивающая сторона может быть представлена Web-сервисом или Web-браузером. Другой защищенный домен может состоять из сервиса ресурса и своего IP/STS-сервиса. Запрашивающая сторона может обратиться к сервису ресурса. Сервис ресурса перенаправит данный запрос своему IP/STS-сервису, с которым у сервиса ресурсов установлены доверительные отношения. Далее IP/STS-сервис ресурса затребует у запрашивающей стороны информацию о ее IP/STS-сервисе. После получения ответа IP/STS-сервис ресурса перенаправляет первона-

чальный запрос запрашивающей стороны ее IP/STS-сервису, который производит аутентификацию запрашивающей стороны. После успешной аутентификации запрашивающей стороны ее IP/STS-сервис посылает маркер защиты IP/STS-сервису ресурса. В результате IP/STS-сервис ресурса посылает свой маркер защиты сервису ресурса и сервис ресурса разрешает доступ запрашивающей стороне к ресурсу.

В другом сценарии запрашивающая сторона проходит аутентификацию у своего IP/STS-сервиса и получает от него маркер защиты, посылает полученный маркер защиты IP/STS-сервису ресурса и получает от него другой маркер защиты, позволяющий получить доступ к необходимому ресурсу. Или же запрашивающая сторона посылает маркер защиты, полученный от своего IP/STS-сервиса, сервису ресурса, а тот, в свою очередь, обменивает полученный маркер защиты на требуемый маркер защиты у IP/STS-сервиса ресурса. В этот процесс могут быть вовлечены третьи IP/STS-сервисы, выдающие маркеры защиты, при этом IP/STS-сервис запрашивающей стороны и IP/STS-сервис ресурса выступают в роли посредников.

В федерацию доменов могут быть вовлечены и другие типы STS-сервисов, помимо IP/STS-сервиса, выдающего маркеры идентификации. Спецификация WS-Federation определяет три дополнительных типа STS-сервисов.

Первый дополнительный тип STS-сервиса — это *сервис атрибутов*. Конечное решение о разрешении доступа запрашивающей стороны к ресурсу принимает сервис ресурса. Для принятия такого решения сервису ресурса может понадобиться дополнительная информация о запрашивающей стороне, которой нет в маркере защиты. В этом случае сервис ресурса запрашивает сервис атрибутов, находящийся в домене запрашивающей стороны, и получает от него дополнительные атрибуты запрашивающей стороны, после чего разрешает доступ к ресурсу.

Другой дополнительный тип STS-сервиса — это *сервис псевдонимов*. Запрашивающая сторона получает маркер защиты от своего IP/STS-сервиса и посылает его IP/STS-сервису ресурса. IP/STS-сервис ресурса с помощью сервиса псевдонимов, находящегося в домене запрашивающей стороны, регистрирует псевдоним и посылает маркер защиты, содержащий псевдоним, запрашивающей стороне, которая и получает доступ к ресурсу с помощью псевдонима.

Третий дополнительный тип STS-сервиса — это *сервис авторизации*. Этот сервис в ответ на запрос маркера защиты выдает маркер, содержащий права доступа к запрашиваемому Web-сервису. Сервис авторизации может быть соединен с запрашиваемым Web-сервисом, а может быть отдельным STS-сервисом. Сервис авторизации создает две логические таблицы, на основе которых он принимает решение о наделении запрашивающей стороны правами доступа. Одна таблица — это таблица требований, создаваемая исходя из адреса конечной точки запрашиваемого Web-сервиса, его свойств, требуемых параметров RST-запроса и внешних политик контроля доступа. Другая таблица — это таблица утверждений, создаваемая исходя из подтвержденных утверждений RST-запроса запрашивающей стороны, дополнительной информации запроса и внешних политик авторизации.

Спецификация WS-Federation определяет два типа запрашивающей стороны.

Первый тип запрашивающей стороны — это сторона активного запроса (Web-сервис или Web-браузер), которая способна создавать SOAP-сообщения. Сторона

активного запроса обменивается политиками и маркерами защиты с помощью SOAP-сообщений.

Второй тип запрашивающей стороны — это сторона пассивного запроса (Web-браузер), оперирующая HTTP-запросами. В этом случае RST-запросы и RSTR-ответы кодируются в HTTP-сообщения.

Для автоматизации процесса объединения защищенных доменов, в результате которого между IP/STS-сервисом запрашивающей стороны и IP/STS-сервисом ресурса устанавливаются доверительные отношения, спецификация WS-Federation определяет XML-документ метаданных федерации. После решения о создании федерации ее участники должны опубликовать и обменяться документами метаданных федерации, позволяющими сконфигурировать IP/STS-сервисы для выдачи маркеров защиты, обеспечивающих доступ к запрашиваемым ресурсам.

Документ метаданных федерации может быть отдельным документом или может быть включен в WSDL-описание с помощью элемента `<fed:FederationMetadata>`.

Корневым элементом XML-документа метаданных федерации является элемент `<fed:FederationMetadata>` (пространство имен `http://schemas.xmlsoap.org/ws/2006/12/federation`), который служит контейнером для секций с метаданными федераций, представленных элементами `<fed:Federation>`. Элемент `<fed:Federation>` имеет атрибут `FederationID`, содержащий URI-идентификатор федерации. Элементов `<fed:Federation>` может быть несколько, т. к. сервис может участвовать в нескольких федерациях. Только у одного из элементов `<fed:Federation>` может отсутствовать атрибут `FederationID`, при этом данная федерация будет федерацией по умолчанию. Кроме того, элемент `<fed:FederationMetadata>` содержит информацию о своей цифровой подписи, обеспечивающей защиту документа метаданных федераций.

Элемент `<fed:Federation>` может содержать элементы `<mex:MetadataReference>`, `<fed:FederationInclude>`, `<fed:AttributeServiceEndpoint>` (определяются в документе любого участника), `<fed:TokenSigningKeyInfo>`, `<fed:PseudonymServiceEndpoint>`, `<fed:SingleSignOutSubscriptionEndpoint>`, `<fed:TokenTypesOffered>`, `<fed:UriNamedClaimTypesOffered>`, `<fed:AutomaticPseudonyms>`, `<fed:IssuerNamesOffered>` (определяются в документе STS-сервиса), `<fed:TokenIssuerName>`, `<fed:TokenIssuerEndpoint>`, `<fed:TokenKeyTransferKeyInfo>`, `<fed:SingleSignoutNotificationEndpoint>` (определяются в документе запрашиваемой стороны).

Элемент `<fed:Federation>` может сам не содержать метаданные федерации, а ссылаться на них с помощью элемента `<mex:MetadataReference>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/mex` спецификации WS-MetadataExchange), который указывает адрес конечной точки для получения документа метаданных федерации, или элемента `<fed:FederationInclude>`, указывающего URI-идентификатор элемента `<fed:Federation>`.

Элемент `<fed:TokenSigningKeyInfo>` содержит информацию о ключе, с помощью которого подписываются выпускаемые STS-сервисом маркеры защиты. Информация о ключе может указываться с помощью вложенного элемента `<wsse:SecurityTokenReference>` спецификации WS-Security.

Элемент `<fed:TokenKeyTransferKeyInfo>` определяет ключ для шифрования ключей. Информация о ключе шифрования также может указываться с помощью вложенного элемента `<wsse:SecurityTokenReference>`.

Элемент `<fed:IssuerNamesOffered>` указывает логическое имя, ассоциированное с STS-сервисом, выпускающим маркеры защиты, используя атрибут `Uri` вложенного элемента `<fed:IssuerName>`.

Элемент `<fed:TokenIssuerName>` указывает логическое имя STS-сервиса запрашиваемой стороны (URI-идентификатор).

Элемент `<fed:TokenIssuerEndpoint>` определяет адрес конечной точки STS-сервиса запрашиваемой стороны и имеет тип элемента `<wsa:EndpointReferenceType>` спецификации WS-Addressing.

Элемент `<fed:PseudonymServiceEndpoint>` определяет адрес конечной точки сервиса псевдонимов и имеет тип элемента `<wsa:EndpointReferenceType>`.

Элемент `<fed:AttributeServiceEndpoint>` определяет адрес конечной точки сервиса атрибутов и имеет тип элемента `<wsa:EndpointReferenceType>`.

Элемент `<fed:SingleSignOutSubscriptionEndpoint>` определяет адрес конечной точки сервиса, подписанного на получение сообщений об окончании федерации. Элемент `<fed:SingleSignOutSubscriptionEndpoint>` также имеет тип элемента `<wsa:EndpointReferenceType>`. Механизм окончания федерации (sign-out) позволяет очистить все кэшированные состояния и маркеры защиты внутри федерации. Запрашивающая сторона может послать сообщение sign-out своему IP/STS-сервису, который в свою очередь разошлет сообщения sign-out подписчикам. В результате вся кэшированная информация, относящаяся к данному объединению, будет очищена.

Элемент `<fed:SingleSignOutNotificationEndpoint>` определяет адрес конечной точки сервиса, который будет получать сообщения sign-out непосредственно от запрашивающей стороны. Элемент `<fed:SingleSignOutNotificationEndpoint>` имеет тип элемента `<wsa:EndpointReferenceType>`.

Элемент `<fed:TokenTypesOffered>` определяет список типов маркеров защиты, которые STS-сервис способен выпустить. Перечень возможных маркеров защиты формируется с помощью вложенных элементов `<fed:TokenType>`, имеющих атрибут `Uri` — URI-идентификатор типа маркера защиты.

Элемент `<fed:UriNamedClaimTypesOffered>` определяет список типов утверждений, которые могут быть в маркерах защиты, выпускаемых STS-сервисом. Перечень возможных утверждений формируется с помощью вложенных элементов `<fed:ClaimType>`, имеющих атрибут `Uri` (URI-идентификатор типа утверждения) и дочерние элементы `<fed:DisplayName>` (отображаемое имя типа утверждения) и `<fed:Description>` (описание утверждения).

Элемент `<fed:AutomaticPseudonyms>` указывает с помощью значения `true`/`false`, применяет ли STS-сервис автоматически псевдонимы.

Метаданные федерации могут быть включены в WSDL-описание Web-сервиса с помощью элемента `<fed:FederationMetadata>` таким же способом, как и политика Web-сервиса, или же могут быть представлены отдельным XML-документом.

Документ метаданных федерации может быть получен с помощью HTTP GET-запроса:

`http://`*`server-name`*`/FederationMetadata/`*`spec-version`*`/`*`fed-id`*`/FederationMetadata.xml`

или HTTPS GET-запроса:

`https://`*`server-name`*`/FederationMetadata/`*`spec-version`*`/`*`fed-id`*`/FederationMetadata.xml`

где *`server-name`* — DNS-имя сервера, обеспечивающего метаданные, *`spec-version`* — версия спецификации (`"2006-12"`), *`fed-id`* — идентификатор FederationID.

Кроме того, документ метаданных федерации может быть получен с помощью GET-запроса протокола WS-Transfer или WS-ResourceTransfer.

Для GET-запроса WS-Transfer или WS-ResourceTransfer получения документа федерации спецификация WS-Federation 1.1 определяет заголовок SOAP-сообщения `<fed:FederationMetadataHandler/>`, присутствие которого сразу указывает, что запрос требует документ метаданных федерации.

Если же метаданные федерации включаются в элемент `<mex:Metadata>` согласно спецификации WS-MetadataExchange, тогда для атрибута `Dialect` соответствующей секции `<mex:MetadataSection>` применяется значение `http://schemas.xmlsoap.org/ws/2006/12/federation`.

Для SOAP-сообщения с sign-out-запросом прекращения федерации спецификация WS-Federation 1.1 определяет следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/2006/12/Federation/SignOut`;
- заголовок `<wsu:Timestamp>` определяет время действия запроса;
- заголовок `<wsse:Security>` содержит информацию о цифровой подписи запроса;
- элемент `<SOAP:Body>` с дочерним элементом `<fed:SignOut>`, содержащим следующие атрибуты и элементы:
  - дополнительный атрибут `wsu:Id` — идентификатор элемента;
  - обязательный элемент `<fed:SignOutBasis>` указывает информацию о запрашивающей стороне с помощью включения в элемент маркеров защиты;
  - дополнительный элемент `<fed:Realm>` указывает URI-идентификатор защищенного домена, к которому обращаются с запросом.

Конечная точка сервиса может подписаться на получение сообщений об окончании федерации согласно спецификации WS-Eventing. В этом случае SOAP-сообщение с Subscribe-запросом в теле сообщения содержит элемент `<wse:Subscribe>`, который

может включать в себя элемент `<wse:Filter>`, фильтрующий подписку по защищенным доменам и маркерам защиты с помощью значения атрибута `Dialect="http://schemas.xmlsoap.org/ws/2006/12/federation/ssoevt"`, а также дочерних элементов `<fed:Realm>` и маркеров защиты.

IP/STS-сервис осуществляет взаимодействие с сервисом псевдонимов с помощью механизма, определенного в спецификациях WS-Transfer и WS-ResourceTransfer.

Для GET-запроса WS-ResourceTransfer спецификация WS-Federation 1.1 определяет элемент `<fed:FilterPseudonyms>`, позволяющий осуществлять фильтрацию псевдонимов по маркерам защиты и области их действия. Каждый псевдоним может иметь область действия, применимую к одному или нескольким сервисам ресурсов. При этом атрибут `Dialect` элемента `<wsrt:Get>` имеет значение `http://schemas.xmlsoap.org/ws/2006/12/federation/pseudonymdialect`, а элемент `<fed:FilterPseudonyms>`, находящийся в элементе `<wsrt:Expression>`, может содержать следующие элементы:

- `<fed:PseudonymBasis>` — маркеры защиты;
- `<fed:RelativeTo>` — область действия псевдонима аналогично элементу `<wsp:AppliesTo>`.

В ответном SOAP-сообщении тело сообщения в элементе `<wsrt:Result>` содержит элемент `<fed:Pseudonym>` со следующими дочерними элементами:

- `<fed:PseudonymBasis>` — маркеры защиты, связанные с псевдонимом;
- `<fed:RelativeTo>` — область действия псевдонима;
- `<wsu:Expires>` — время действия псевдонима;
- `<fed:SecurityToken>` — маркер защиты псевдонима;
- `<fed:ProofToken>` — ключ к маркеру защиты псевдонима.

Также сервис псевдонимов поддерживает PUT- и CREATE-запросы для добавления, удаления и создания псевдонимов, содержащие элементы `<fed:FilterPseudonyms>` (находится в элементе `<wsrt:Expression>`) и `<fed:Pseudonym>` (находится в элементе `<wsrt:Value>`).

Для RST-запроса запрашивающей стороны IP/STS-сервису, интегрированному с сервисом псевдонимов, определен дочерний элемент `<fed:RequestPseudonym>` элемента `<wst:RequestSecurityToken>`, имеющий следующие атрибуты:

- `SingleUse` — если `true`, тогда запрашивается псевдоним для одноразового использования, по умолчанию `false`;
- `Lookup` — если `true` (по умолчанию), тогда используется псевдоним с определенной областью действия для конкретных сервисов ресурсов, в противном случае используется только первичный идентификатор запрашивающей стороны.

Сервис псевдонимов хранит маркеры защиты (логин и пароль, публичный и симметричный ключи), связанные с псевдонимами и созданные IP/STS-сервисом запрашивающей стороны.

Запрашивающая сторона может послать сервису ресурса не сам маркер защиты, полученный ею от IP/STS-сервиса ресурса, а ссылку на него. Далее сервис ресурса запрашивает свой IP/STS-сервис на получение самого маркера защиты. Для этого сценария определяется элемент `<fed:ReferenceToken>`, возвращаемый запрашивающей стороне IP/STS-сервисом ресурса в RSTR-ответе. Элемент `<fed:ReferenceToken>` имеет URI-идентификатор `http://schemas.xmlsoap.org/ws/2006/12/federation/reftoken` и содержит следующие элементы:

- обязательный элемент `<fed:ReferenceEPR>` — адрес конечной точки для получения самого маркера защиты;
- дополнительный элемент `<fed:ReferenceDigest>` — хэш-код маркера защиты;
- дополнительный элемент `<fed:ReferenceType>` — URI-идентификатор маркера защиты, на который ссылаются;
- дополнительный элемент `<fed:SerialNo>` — URI-идентификатор маркера защиты.

В случае если сторона, отправляющая RST-запрос, участвует в нескольких федерациях доменов, для RST-запроса и RSTR-ответа определяется элемент `<fed:FederationID>`, указывающий URI-идентификатор контекста федерации, для которого запрашивается маркер защиты.

Если IP/STS-сервисы запрашивающего сервиса и сервиса ресурса отвечают за выдачу им сессионных ключей, для RST-запроса определяется элемент `<fed:RequestProofToken>`, в ответ на который IP/STS-сервис посылает элемент `<wst:RequestedProofToken>` с сессионным ключом.

Для RST-запроса получения маркера защиты псевдонима спецификация дополнительно определяет элемент `<fed:ClientPseudonym>`, указывающий дополнительную информацию для выдачи псевдонима. Элемент `<fed:ClientPseudonym>` может содержать следующие элементы:

- `<fed:PPID>` — персональный идентификатор;
- `<fed:DisplayName>` — отображаемое имя псевдонима;
- `<fed:Email>` — адрес электронной почты.

В RST-запросе может быть указано время действия данных, на основе которых выдается маркер защиты. Для этого определен элемент `<fed:Freshness>`, указывающий количество минут, в течение которых данные актуальны. Элемент `<fed:Freshness>` также имеет атрибут `AllowCache`, позволяющий при значении `true` (по умолчанию) использовать кэшированные данные для обработки запроса.

Сервис авторизации реализует этап принятия решения о доступе к сервису ресурса и выдает запрашивающей стороне маркеры защиты, дающие права доступа к сервису ресурсов. Делается это в ответ на RST-запрос, указывающий свойства требуемых маркеров защиты. Необходимые свойства маркеров защиты описываются в политике сервиса ресурсов. Для передачи в RST-запросе дополнительной информации, влияющей на выдачу маркеров защиты, спецификация WS-Federation определяет элемент `<auth:AdditionalContext>` (`http://schemas.xmlsoap.org/ws/2006/12/authorization`), являющийся дочерним элементом элемента `<wst:RequestSecurity`

Token> и содержащий элементы <auth:ContextItem>, которые передают дополнительные свойства в виде пар "имя/значение". Имя свойства указывается с помощью атрибута Name элемента <auth:ContextItem>, а его значение — с помощью дочернего элемента <auth:Value> элемента <auth:ContextItem>. Элемент <auth:ContextItem> также имеет атрибут Scope, указывающий URI-идентификатор области действия свойства. Возможные значения атрибута Scope:

- http://schemas.xmlsoap.org/ws/2006/12/authorization/ctx/requestor — свойство относится к запрашивающей стороне;
- http://schemas.xmlsoap.org/ws/2006/12/authorization/ctx/target — свойство относится к запрашиваемой стороне;
- http://schemas.xmlsoap.org/ws/2006/12/authorization/ctx/action — свойство относится к действию запрашиваемой стороны.

Для передачи в RST-запросе сервису авторизации информации о ключах, идентификаторах, правах и т. д., которую запрашивающий Web-сервис просит включить в генерируемый сервисом авторизации маркер защиты, спецификация определяет значение атрибута Dialect элемента <wst:Claims> как http://schemas.xmlsoap.org/ws/2006/12/authorization/authclaims, а также определяет дочерний элемент <auth:ClaimType> элемента <wst:Claims>.

Элемент <auth:ClaimType> может передавать данные с помощью дочернего элемента <auth:Value> и иметь атрибут Optional (если true, тогда утверждение необязательно для включения в маркер защиты). Обязательный атрибут Uri элемента <auth:ClaimType> указывает тип утверждения. Спецификация предопределяет значение атрибута Uri как http://schemas.xmlsoap.org/ws/2006/12/authorization/claims/action — определяется значение элемента <wsa:Action>.

Сервис ресурса при получении запроса от запрашивающей стороны может потребовать от нее дополнительную информацию, требование которой не было первоначально определено в политике сервиса ресурса. Для этого сервис ресурса посылает запрашивающей стороне SOAP-сообщение, содержащее в теле сообщения элемент <SOAP:Fault> со следующими дочерними элементами:

- <SOAP:Code> содержит элемент <SOAP:Value> со значением fed:SpecificMetadata;
- <SOAP:Detail> содержит элемент <mex:Metadata> с дополнительной политикой.

Запрашивающая сторона, после получения такого сообщения, создает новый запрос сервису ресурсов с учетом дополнительных требований.

В RST-запросе получения маркера защиты запрашивающая сторона может указать информацию, включаемую в маркер защиты, которая должна быть зашифрована для гарантии того, что только нужный сервис ресурса сможет обработать ее при получении маркера защиты. Для этого определен элемент <priv:ProtectData> (пространство имен http://schemas.xmlsoap.org/ws/2006/12/privacy), включаемый в элемент <wst:RequestSecurityToken> и содержащий элементы <wst:Claims> для описания утверждений запрашиваемого маркера защиты, требующих защиты конфиденциальности.

RST-запрос может содержать параметры для обеспечения конфиденциальности запрашиваемого маркера защиты. Для того чтобы запрашивающая сторона могла узнать, были ли приняты STS-сервисом параметры и какие значения параметров им использовались, определены следующие элементы, включаемые в элемент `<wst:RequestSecurityToken>`:

- `<priv:EnumerateParameters>` содержит список QName-имен параметров, которые должны быть возвращены в RSTR-ответе со значениями, используемыми STS-сервисом;
- `<priv:FaultOnUnacceptedRstParameters>` указывает, что если какие-либо из параметров не были приняты STS-сервисом, тогда он должен послать ответное сообщение об ошибке;
- `<priv:EnumerateAllClaims>` указывает, что все утверждения, возвращаемые в маркере защиты, должны быть идентифицированы так, что запрашивающая сторона могла бы их проинспектировать, даже если они зашифрованы и нечитаемые для запрашивающей стороны. Утверждения возвращаются в элементе `<wst:Claims>` RSTR-ответа.

Web-сервис может содержать в своей политике требования защиты конфиденциальности. Для указания адреса конечной точки, по которому можно получить политику конфиденциальности, спецификация определяет элемент `<priv:PrivacyPolicyEndpoint>`, имеющий тип элемента `<wsa:EndpointReference>`. Политика конфиденциальности может быть получена с помощью GET-запроса спецификации WS-Transfer/WS-ResourceTransfer. Атрибут `SupportsMex` (`true`/`false`) элемента `<priv:PrivacyPolicyEndpoint>` указывает, может ли политика конфиденциальности быть получена с помощью GET-запроса спецификации WS-MetadataExchange. Для атрибута `Dialect` элемента `<mex:MetadataSection>` (WS-MetadataExchange) спецификация определяет значение `http://schemas.xmlsoap.org/ws/2006/12/privacy/privacypolicy` политики конфиденциальности.

В случае если запрашивающая сторона представлена Web-браузером, неспособным формировать SOAP-сообщения, все запросы должны кодироваться в виде HTTP GET- и POST-запросов, содержащих необходимые параметры участия в федерации доменов.

Спецификация WS-Federation 1.1 определяет ряд параметров для HTTP GET- и POST-запросов:

- обязательный параметр `wa` — действие сообщения (аналог заголовку `<wsa:Action>`). Для участия в федерации значение параметра — `wsignin1.0`;
- дополнительный параметр `wreply` — URL-адрес ответа на данное сообщение (аналог заголовка `<wsa:ReplyTo>`);
- дополнительный параметр `wres` указывает URL-адрес запрашиваемого ресурса;
- дополнительный параметр `wctx` — аналог атрибута `Context` элемента `<wst:RequestSecurityToken>`;
- дополнительный параметр `wp` — URL-адрес политики;

- дополнительный параметр `wct` — время запроса, аналог элемента `<wsu:Timestamp>`;
- дополнительный параметр `wfed` — аналог элемента `<fed:FederationID>` RST-запроса.

Для HTTP GET- и POST-запросов IP/STS-сервису используются следующие параметры, дополнительно к вышеперечисленным:

- обязательный параметр `wtrealm` — URI-идентификатор домена запрашиваемого ресурса;
- дополнительный параметр `wfresh` — аналог элемента `<fed:Freshness>`;
- дополнительный параметр `wauth` — аналог элемента `<wst:AuthenticationType>`;
- дополнительный параметр `wreq` — XML-элементы самого RST-запроса, представленного элементом `<wst:RequestSecurityToken>` или элементами полного сообщения;
- дополнительный параметр `whr` — идентификатор, используемый для указания адреса IP/STS-сервиса Web-браузера;
- дополнительный параметр `wreqptr` — указывает URL-адрес, по которому можно запросить XML-элементы самого RST-запроса (параметр `wreq`) в случае, если данные слишком большого размера.

В ответ IP/STS-сервис возвращает маркер защиты HTTP POST-методом. Для ответа определены параметры:

- обязательный параметр `wresult` — XML-элементы RSTR-ответа;
- дополнительный параметр `wctx` — аналог атрибута `Context` элемента `<wst:RequestSecurityTokenResponse>`;
- дополнительный параметр `wresultptr` — URL-адрес, по которому можно запросить методом HTTP GET XML-элементы RSTR-ответа (параметр `wresult`) в случае, если данные слишком большого размера.

Для HTTP-запроса прекращения федерации sign-out спецификация определяет следующие параметры:

- обязательный параметр `wa` — действие сообщения (аналог заголовку `<wsa:Action>`). Для действия sign-out значение параметра равно `wsignout1.0`;
- дополнительный параметр `wreply` — URL-адрес для ответного сообщения об успешном завершении федерации.

При первоначальном обращении клиентского Web-браузера к ресурсу Web-сервис ресурса может запросить у клиента дополнительные атрибуты. В этом случае клиент должен запросить атрибуты у сервиса атрибутов и передать их Web-сервису ресурса. Для HTTP-запроса и ответа атрибутов спецификация определяет следующие параметры:

- обязательный параметр `wa` — действие со значением `wattr1.0`;
- дополнительный параметр `wreply` — URL-адрес для возврата атрибутов;

- дополнительный параметр `wattr` — XML-элементы запроса атрибутов;
- дополнительный параметр `wattrptr` — URL-адрес, по которому можно получить XML-элементы запроса атрибутов;
- дополнительный параметр `wresult` — XML-элементы ответа на запрос атрибутов;
- дополнительный параметр `wresultptr` — URL-адрес, по которому можно получить XML-элементы ответа на запрос атрибутов.

При обращении Web-браузера к ресурсу Web-сервис ресурса может запросить у клиента псевдоним. При этом клиент должен запросить псевдоним у сервиса псевдонимов и передать его Web-сервису ресурса. Для HTTP-запроса и ответа псевдонима спецификация определяет следующие параметры:

- обязательный параметр `wa` — действие со значением `wpseudo1.0`;
- дополнительный параметр `wreply` — URL-адрес для возврата псевдонима;
- дополнительный параметр `wpseudo` — XML-элементы запроса псевдонима;
- дополнительный параметр `wpseudoptr` — URL-адрес, по которому можно получить XML-элементы запроса псевдонима;
- дополнительный параметр `wresult` — XML-элементы ответа на запрос псевдонима;
- дополнительный параметр `wresultptr` — URL-адрес, по которому можно получить XML-элементы ответа на запрос псевдонима.

Спецификация определяет ряд утверждений политики для поддержки WS-Federation 1.1.

Утверждение `<fed:RequireReferenceToken>` требует поддержки элемента `<fed:ReferenceToken>`. Элемент `<fed:RequireReferenceToken>` может иметь атрибут `sp:IncludeToken` (пространство имен `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512` спецификации WS-SecurityPolicy) и элемент `<wsp:Policy>` с дочерним элементом `<fed:RequireReferenceToken11>`, указывающим требование поддержки элемента `<fed:ReferenceToken>` спецификации WS-Federation 1.1.

Утверждение `<fed:WebBinding>` требует поддержки Web-клиента с HTTP GET- и POST-запросами. Элемент `<fed:WebBinding>` может иметь дочерний элемент `<wsp:Policy>` со следующими элементами:

- утверждение `<sp:TransportToken>` спецификации WS-SecurityPolicy;
- `<fed:AuthenticationToken>` указывает требуемый тип маркера защиты для аутентификации. Элемент `<fed:AuthenticationToken>` содержит элемент `<wsp:Policy>`, который включает требования к маркеру аутентификации и может содержать элемент `<fed:ReferenceToken>`, указывающий поддерживаемую ссылку;
- `<fed:RequireBearerTokens/>` указывает поддержку только маркеров защиты, несущих информацию о клиенте, за исключением ключей. Маркеры защиты Bearer Tokens не содержат ключи Proof Key, подтверждающие право на использование данных маркеров защиты клиентом;

- `<fed:RequireSignedTokens/>` требует цифровой подписи маркера защиты;
- `<fed:RequireSharedCookies/>` требует использование cookies для обращения к IP/STS-сервису Web-браузера.

Утверждение `<fed:RequiresGenericClaimDialect/>` требует поддержки элемента `<auth:ClaimType>`.

Утверждение `<fed:IssuesSpecificMetadataFault/>` требует поддержки значения `fed:SpecificPolicy` элемента `<SOAP:Value>`, содержащегося в элементе `<SOAP:Code>` элемента `<SOAP:Fault>`. Сервис ресурса выдает такое сообщение об ошибке запрашивающей стороне, чтобы та скорректировала свой запрос с учетом дополнительных требований, не отраженных в основной политике.

Утверждение `<fed:AdditionalContextProcessed/>` требует поддержки элемента `<auth:AdditionalContext>`.

WS-Transfer

Спецификация WS-Transfer 2006 г. определяет механизм получения, обновления, удаления и создания ресурса, представленного XML-документом и находящегося по адресу конечной точки Web-сервиса, которая указывается согласно спецификации WS-Addressing (пространство имен `xmlns:wsa="http://www.w3.org/2005/08/addressing"`).

XML-документ, представляющий ресурс, может быть получен с помощью операции `Get`, которая подразумевает отправку SOAP-сообщения, содержащего следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/Get`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения.

В ответ Web-сервис посылает сообщение, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse`;
- заголовок `<wsa:RelatesTo>` указывает связь данного сообщения с другим сообщением с помощью идентификатора;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- тело сообщения с XML-документом.

Операция `Put` обновления ресурса подразумевает отправку SOAP-сообщения Web-сервису со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/Put`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;

- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- тело сообщения с XML-документом, представляющим ресурс для обновления.

В ответ Web-сервис посылает сообщение, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/PutResponse`;
- заголовок `<wsa:RelatesTo>` указывает связь данного сообщения с другим сообщением с помощью идентификатора;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- тело сообщения с XML-документом, представляющим обновленный ресурс.

Операция `Delete` удаления ресурса подразумевает отправку SOAP-сообщения Web-сервису со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/Delete`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения.

В ответ Web-сервис посылает сообщение, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/DeleteResponse`;
- заголовок `<wsa:RelatesTo>` указывает связь данного сообщения с другим сообщением с помощью идентификатора;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения.

Операция `Create` создания ресурса на основе посылаемого XML-документа подразумевает отправку SOAP-сообщения Web-сервису со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/Create`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- тело сообщения с XML-документом, представляющим ресурс для его создания.

В ответ Web-сервис посылает сообщение, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/transfer/CreateResponse`;
- заголовок `<wsa:RelatesTo>` указывает связь данного сообщения с другим сообщением с помощью идентификатора;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- тело сообщения с элементом `<wxf:ResourceCreated>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/transfer`), указывающим адрес конечной точки созданного ресурса, и XML-документом, представляющим созданный ресурс.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с Get-запросом представления ресурса как сообщение, содержащее:

- заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-tra/Get`;
- элемент `<SOAP:Body>` с дочерним элементом `<wst:Get>` (пространство имен `http://www.w3.org/2010/03/ws-tra`), имеющим необязательный атрибут `Dialect` — IRI-идентификатор способа обработки элемента `<wst:Get>`. Значение атрибута `Dialect="http://www.w3.org/2010/03/ws-fra"` подразумевает обработку элемента `<wst:Get>` согласно спецификации WS-Fragment.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с ответом на Get-запрос как сообщение, содержащее:

- заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-tra/GetResponse`;
- элемент `<SOAP:Body>` с дочерним элементом `<wst:GetResponse>`, который содержит элемент `<wst:Representation>` с полным представлением ресурса, не являющимся фрагментом ресурса.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с Put-запросом представления ресурса как сообщение, содержащее:

- заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-tra/Put`;
- элемент `<SOAP:Body>` с дочерним элементом `<wst:Put>`, имеющим необязательный атрибут `Dialect` — IRI-идентификатор способа обработки элемента `<wst:Put>`. Значение атрибута `Dialect="http://www.w3.org/2010/03/ws-fra"` подразумевает обработку элемента `<wst:Put>` согласно спецификации WS-Fragment. Элемент `<wst:Put>` содержит элемент `<wst:Representation>` с полным представлением ресурса, не являющимся фрагментом ресурса, для обновления ресурса.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с ответом на Put-запрос как сообщение, содержащее:

- заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-tra/PutResponse`;
- элемент `<SOAP:Body>` с дочерним элементом `<wst:PutResponse>`, содержащим элемент `<wst:Representation>` с полным обновленным представлением ресурса, не являющимся фрагментом ресурса.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с Delete-запросом представления ресурса как сообщение, содержащее:

- заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-tra/Delete`;
- элемент `<SOAP:Body>` с дочерним элементом `<wst:Delete>`, имеющим необязательный атрибут `Dialect` — IRI-идентификатор способа обработки элемента `<wst:Delete>`.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с ответом на Delete-запрос как сообщение, содержащее:

- заголовок <wsa:Action> со значением http://www.w3.org/2010/03/ws-tra/ DeleteResponse;
- элемент <SOAP:Body> с дочерним элементом <wst:DeleteResponse>.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с Create-запросом создания ресурса как сообщение, содержащее:

- заголовок <wsa:Action> со значением http://www.w3.org/2010/03/ws-tra/Create;
- элемент <SOAP:Body> с дочерним элементом <wst:Create>, имеющим необязательный атрибут Dialect — IRI-идентификатор способа обработки элемента <wst:Create>. Элемент <wst:Create> содержит элемент <wst:Representation> с полным представлением ресурса для создания ресурса.

Спецификация WS-Transfer 2010 г. определяет SOAP-сообщение с ответом на Create-запрос как сообщение, содержащее:

- заголовок <wsa:Action> со значением http://www.w3.org/2010/03/ws-tra/CreateResponse;
- элемент <SOAP:Body> с дочерним элементом <wst:CreateResponse>, содержащим элемент <wst:Representation> с полным представлением созданного ресурса, а также элемент <wst:ResourceCreated>, указывающий адрес конечной точки созданного ресурса.

WS-ResourceTransfer и WS-Fragment

Спецификация WS-ResourceTransfer (WS-RT) определяет расширения спецификации WS-Transfer 2006 г., позволяющие получить частичный доступ к XML-документам, представляющим ресурсы.

Операция Get расширяется включением заголовка <wsrt:ResourceTransfer SOAP:mustUnderstand="true"?/> (пространство имен http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer), указывающим, что Web-сервис должен обработать тело SOAP-сообщения, содержащее WS-RT-расширения, а также включением в тело SOAP-сообщения элемента <wsrt:Get>, определяющим извлечение фрагмента ресурса.

Элемент <wsrt:Get> имеет атрибут Dialect (URI-идентификатор типа выражений) и дочерние элементы <wsrt:Expression> (выражение, определяющее фрагмент ресурса, который должен быть возвращен в ответном сообщении). Спецификация предопределяет возможные значения атрибута Dialect:

- http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/Dialect/QName — элемент <wsrt:Expression> содержит QName-имя элемента XML-документа ресурса. В ответном сообщении возвращается содержимое элемента, включая все его дочерние элементы;
- http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/Dialect/XPath-Level-1 — элемент <wsrt:Expression> содержит XPath-выражение с ограниченными возможностями;

- http://www.w3.org/TR/1999/REC-xpath-19991116 — элемент <wsrt:Expression> содержит XPath-выражение с полноценной поддержкой библиотеки функций.

В ответном SOAP-сообщении присутствует заголовок <wsrt:ResourceTransfer/> и тело SOAP-сообщения содержит элемент <wsrt:Result> с фрагментом ресурса.

Операция Put расширяется включением в SOAP-сообщение заголовка <wsrt:ResourceTransfer SOAP:mustUnderstand="true"?/>, а также включением в тело SOAP-сообщения элемента <wsrt:Put>, определяющего фрагмент ресурса для обновления.

Элемент <wsrt:Put> имеет атрибут Dialect (URI-идентификатор типа выражений) и вложенные элементы <wsrt:Fragment> с дочерними элементами <wsrt:Expression> и <wsrt:Value>.

Элемент <wsrt:Fragment> имеет атрибут Mode, указывающий режим обновления фрагмента ресурса. Возможные значения атрибута Mode: Remove (удаление фрагмента), Modify (замена существующего фрагмента), Insert (добавление фрагмента). Элемент <wsrt:Expression> определяет фрагмент ресурса, который должен быть удален, заменен или добавлен. Элемент <wsrt:Value> содержит данные, которые используются в случае операции Modify или Insert.

В ответном SOAP-сообщении присутствует заголовок <wsrt:ResourceTransfer/>.

Операция Create расширяется включением в SOAP-сообщение заголовка <wsrt:ResourceTransfer SOAP:mustUnderstand="true"?/>, а также включением в тело SOAP-сообщения элемента <wsrt:Create>, определяющим фрагмент ресурса для его инициализации при создании ресурса и создаваемые метаданные ресурса.

Элемент <wsrt:Create> имеет атрибут Dialect (URI-идентификатор типа выражений) и вложенные элементы <wsmex:Metadata> и <wsrt:Fragment> (дочерние элементы <wsrt:Expression> и <wsrt:Value>). Элемент <wsmex:Metadata> (пространство имен http://schemas.xmlsoap.org/ws/2004/09/mex спецификации WS-Metadata-Exchange) определяет создаваемые метаданные ресурса, например WSDL-описание Web-сервиса ресурса. Элемент <wsrt:Fragment> определяет фрагмент ресурса для его инициализации при создании ресурса с помощью элементов <wsrt:Expression> — выражение для идентификации фрагмента ресурса и <wsrt:Value> — данные фрагмента ресурса.

Элемент <wsmex:Metadata> может содержать элемент <wsmex:MetadataSection> (атрибут Dialect со значением http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer), имеющим дочерний элемент <wsrt:Metadata> с вложенным элементом <wsrt:Lifetime>, определяющим жизненный цикл ресурса.

Элемент <wsrt:Lifetime> содержит элемент <wsrt:TerminateAt> с вложенным элементом <wsrt:TerminationTime>, указывающим время ликвидации ресурса, и вложенным элементом <wsrt:CurrentTime>, указывающим время инициации создания ресурса.

В ответном SOAP-сообщении присутствует заголовок <wsrt:ResourceTransfer/>.

Спецификация WS-Fragment 2010 г. определяет расширения спецификации WS-Transfer 2010 г., позволяющие получить частичный доступ к XML-документам, представляющим ресурсы.

Операция `Get` расширяется включением в элемент `<SOAP:Body>` дочернего элемента `<wst:Get>` (пространство имен `http://www.w3.org/2010/03/ws-tra`), имеющим атрибут `Dialect` со значением `http://www.w3.org/2010/03/ws-fra`. При этом элемент `<wst:Get>` содержит элементы `<wsf:Expression>` (пространство имен `http://www.w3.org/2010/03/ws-fra`), идентифицирующие фрагменты ресурса. Элемент `<wsf:Expression>` имеет атрибут `Language`, указывающий IRI-идентификатор языка выражений элемента `<wsf:Expression>`.

В ответном SOAP-сообщении элемент `<SOAP:Body>` имеет дочерний элемент `<wst:GetResponse>`, содержащий элемент `<wsf:Value>` с фрагментом ресурса.

Операция `Put` расширяется включением в элемент `<SOAP:Body>` дочернего элемента `<wst:Put>`, имеющим атрибут `Dialect` со значением `http://www.w3.org/2010/03/ws-fra`. При этом элемент `<wst:Put>` содержит элементы `<wsf:Fragment>` с дочерним элементом `<wsf:Expression>`, идентифицирующим фрагмент ресурса для обновления, а также элемент `<wsf:Value>`, содержащий данные обновления. Элемент `<wsf:Expression>` имеет атрибут `Language`, указывающий IRI-идентификатор языка выражений элемента `<wsf:Expression>`, и атрибут `Mode` — IRI-идентификатор режима обновления (по умолчанию `http://www.w3.org/2010/03/ws-fra/Modes/Replace`). Возможные значения атрибута `Mode`:

- `http://www.w3.org/2010/03/ws-fra/Modes/Replace` — замена указанного выражением фрагмента ресурса;
- `http://www.w3.org/2010/03/ws-fra/Modes/Add` — добавление относительно указанного выражением фрагмента ресурса;
- `http://www.w3.org/2010/03/ws-fra/Modes/InsertBefore` — добавление перед указанным выражением фрагментом ресурса;
- `http://www.w3.org/2010/03/ws-fra/Modes/InsertAfter` — добавление после указанного выражением фрагмента ресурса;
- `http://www.w3.org/2010/03/ws-fra/Modes/Remove` — удаление фрагмента ресурса.

В ответном SOAP-сообщении элемент `<SOAP:Body>` имеет дочерний элемент `<wst:PutResponse>`, содержащий элемент `<wst:Representation>` с обновленным представлением ресурса.

Спецификация WS-Fragment не расширяет операции `Delete` и `Create`.

WS-MetadataExchange

Спецификация WS-MetadataExchange версии 1.1 определяет представление метаданных Web-сервиса (его политик, WSDL-описания и др.) как ресурсов согласно спецификации WS-Transfer, включение метаданных Web-сервиса в ссылки на конечные точки Web-сервиса и механизм получения метаданных из конечных точек Web-сервиса.

Для представления метаданных Web-сервиса как ресурса, связанного с конечной точкой Web-сервиса, спецификация определяет элемент `<mex:Metadata>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/mex`). Элемент `<mex:Metadata>` позволяет собрать вместе все метаданные Web-сервиса — его WSDL-описание, политики, XML-схемы и т. д. — с помощью дочерних элементов `<mex:MetadataSection>`.

Элемент `<mex:MetadataSection>` представляет определенную единицу метаданных Web-сервиса — его WSDL-описание, политику, XML-схему или др. Элемент `<mex:MetadataSection>` имеет следующие атрибуты и дочерние элементы:

- атрибут `Dialect` — формат и версия метаданных. Спецификация предопределяет следующие значения атрибута `Dialect`:
  - `http://www.w3.org/2001/XMLSchema` — в элементе `<mex:MetadataSection>` содержится XML-схема `<xs:schema>`;
  - `http://schemas.xmlsoap.org/wsdl/` — формат метаданных `<wsdl:definitions>` WSDL 1.1;
  - `http://schemas.xmlsoap.org/ws/2004/09/policy` — формат метаданных `<wsp:Policy>`;
  - `http://schemas.xmlsoap.org/ws/2004/09/policy/attachment` — формат метаданных `<wsp:PolicyAttachment>`;
  - `http://schemas.xmlsoap.org/ws/2004/09/mex` — элемент `<mex:MetadataSection>` содержит ссылку на метаданные;
- атрибут `Identifier` — идентификатор секции `<mex:MetadataSection>` (для XML-схемы и WSDL 1.1 описания — это значение атрибута `targetNamespace`, для политики — это значение атрибута `Name`);
- элемент `<mex:MetadataReference>` — ссылка на метаданные согласно спецификации WS-Addressing;
- элемент `<mex:Location>` — альтернатива элементу `<mex:MetadataReference>`, содержит URL-адрес для получения метаданных с помощью HTTP GET-запроса;
- при отсутствии ссылок на метаданные элемент `<mex:MetadataSection>` содержит сами метаданные.

Для получения метаданных Web-сервиса в SOAP-сообщении спецификация определяет два вида SOAP-сообщений с Get-запросом. Первый тип сообщений — это сообщения с Get-запросом согласно спецификации WS-Transfer. Второй тип сообщений — это сообщения с GetMetadata-запросом согласно спецификации WS-MetadataExchange.

Сообщение с GetMetadata-запросом содержит заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Request` и элемент `<SOAP:Body>` с дочерним элементом `<mex:GetMetadata>`, который может уточнять запрашиваемые метаданные с помощью элементов `<mex:Dialect>` и `<mex:Identifier>` — формат и идентификатор запрашиваемой секции `<mex:MetadataSection>`.

Ответное сообщение содержит заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Response` и элемент `<SOAP:Body>` с дочерним элементом `<mex:Metadata>` с метаданными Web-сервиса.

Спецификация WS-MetadataExchange определяет возможность включения метаданных в элемент `<wsa:EndpointReference>` спецификации WS-Addressing для альтернативного получения, помимо Get-запросов, информации о Web-сервисе.

Для спецификации WS-Addressing 2004 элемент `<mex:Metadata>` включается непосредственно в элемент `<wsa:EndpointReference>`, а для спецификации WS-Addressing 1.0 Core элемент `<mex:Metadata>` включается в дочерний элемент `<wsa:Metadata>` элемента `<wsa:EndpointReference>`.

Спецификация WS-MetadataExchange 2010 г. (пространство имен `http://www.w3.org/2010/03/ws-mex`) определяет специальный GetWSDL-запрос для получения WSDL-описания Web-сервиса. SOAP-сообщение с GetWSDL-запросом содержит заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-mex/GetWSDL` и элемент `<SOAP:Body>` с дочерним элементом `<mex:GetWSDL>`. Ответное сообщение содержит заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-mex/GetWSDLResponse` и элемент `<SOAP:Body>` с дочерним элементом `<mex:GetWSDLResponse>`, содержащим WSDL-описание Web-сервиса `<wsdl:definitions>`.

SOAP-сообщение с GetMetadata-запросом содержит заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-mex/GetMetadata` и элемент `<SOAP:Body>` с дочерним элементом `<mex:GetMetadata>`, который может включать в себя элементы `<mex:Dialect>`, уточняющие запрашиваемые метаданные с помощью атрибутов:

- `URI` — идентификатор формата метаданных;
- `Identifier` — идентификатор секции метаданных;
- `Content` — идентификатор содержимого метаданных. Возможные значения атрибута `Content`:
  - `http://www.w3.org/2010/03/ws-mex/Content/EPR` — возвращается элемент `<mex:MetadataReference>`;
  - `http://www.w3.org/2010/03/ws-mex/Content/URI` — возвращается элемент `<mex:Location>`;
  - `http://www.w3.org/2010/03/ws-mex/Content/Metadata` — возвращаются сами метаданные;
  - `http://www.w3.org/2010/03/ws-mex/Content/Any` — Web-сервис сам решает, что возвращать;
  - `http://www.w3.org/2010/03/ws-mex/Content/All` — возвращаются все возможные данные.

Ответное сообщение содержит заголовок `<wsa:Action>` со значением `http://www.w3.org/2010/03/ws-mex/GetMetadataResponse` и элемент `<SOAP:Body>` с дочерним элементом `<mex:GetMetadataResponse>`, включающим в себя элемент `<mex:Metadata>` с возвращаемыми метаданными.

Спецификация WS-MetadataExchange 2010 г. определяет атрибуты элемента `<mex:Location>`:

- обязательный атрибут `URL` — адрес метаданных;
- обязательный атрибут `Type` — QName-имя метаданных;
- дополнительный атрибут `Identifier` — идентификатор секции метаданных.

В дочерний элемент `<wsa:Metadata>` элемента `<wsa:EndpointReference>` метаданные включаются с помощью элементов `<mex:Location>`, `<mex:Reference>` (ссылка для WS-Transfer Get-запроса, атрибуты `Type` и `Identifier`) и `<mex:Metadata>`.

Для поддержки спецификации WS-MetadataExchange, спецификация WS-MetadataExchange 2010 г. определяет утверждение политики Web-сервиса, представленное элементом `<mex:MetadataExchange>` с дочерним элементом `<mex:GetMetadataSupported>` (требование поддержки GetMetadata-запроса), который может содержать элемент `<mex:MetadataExchangeDialect>` (поддержка формата метаданных, указываемого с помощью атрибута `URI`) с дочерним элементом `<mex:MetadataContent>` — поддержка содержимого метаданных, указываемого с помощью атрибута `URI`.

WS-Enumeration

Спецификация WS-Enumeration определяет механизм, позволяющий запрашивающему Web-сервису получать от Web-сервиса, представляющего источник данных, XML-данные большого размера, используя несколько SOAP-сообщений. При этом между двумя Web-сервисами создается сессия, характеризующаяся контекстом перечисления и обеспечивающая получение последовательности XML-элементов.

Спецификация WS-Enumeration 2006 г. (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/enumeration`) определяет SOAP-сообщение с Enumerate-запросом, которое запрашивающий Web-сервис должен послать Web-сервису источника данных для того, чтобы Web-сервис источника данных создал контекст перечисления, обеспечивающий определенную последовательность XML-элементов. Далее запрашивающий Web-сервис может частями получать XML-данные из общей последовательности. SOAP-сообщение с Enumerate-запросом содержит следующие элементы:

- заголовок `<wsa:Action>` (WS-Addressing) со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:Enumerate>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/09/enumeration`), который может содержать следующие элементы:
  - `<wsen:EndTo>` — адрес конечной точки, куда Web-сервис источника данных должен послать сообщение в случае, если Web-сервис источника данных по каким-то причинам неожиданно прервал сессию перечисления;

- `<wsen:Expires>` — запрашиваемое время окончания сессии перечисления. Реальное время окончания сессии перечисления устанавливает Web-сервис источника данных;
- `<wsen:Filter>` — фильтрующее выражение для формируемой Web-сервисом источника данных общей последовательности XML-элементов. Элемент `<wsen:Filter>` содержит атрибут `Dialect`, указывающий URI-идентификатор языка выражений, по умолчанию значение атрибута `http://www.w3.org/TR/1999/REC-xpath-19991116` — XPath 1.0-выражения.

В ответ Web-сервис источника данных посылает SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerateResponse`;
- заголовок `<wsa:ReplyTo>` — URI-адрес конечной точки получателя ответа на это сообщение;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:EnumerateResponse>`, содержащим следующие элементы:
  - дополнительный элемент `<wsen:Expires>` — время окончания сессии перечисления;
  - обязательный элемент `<wsen:EnumerationContext>` — строка, идентифицирующая контекст перечисления.

После установления контекста перечисления запрашивающий Web-сервис может послать SOAP-сообщение с Pull-запросом для получения XML-элементов из общей последовательности. SOAP-сообщение с Pull-запросом содержит следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/Pull`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:ReplyTo>` — URI-адрес конечной точки получателя ответа на это сообщение;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:Pull>`, содержащим следующие элементы:
  - обязательный элемент `<wsen:EnumerationContext>` — идентификатор контекста перечисления;
  - дополнительный элемент `<wsen:MaxTime>` — максимальное время для формирования ответа Web-сервисом источника данных;
  - дополнительный элемент `<wsen:MaxElements>` — максимальное количество возвращаемых элементов;

- дополнительный элемент `<wsen:MaxCharacters>` — максимальный размер возвращаемых элементов.

В ответ Web-сервис источника данных посылает SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/PullResponse`;
- заголовок `<wsa:RelatesTo>` — идентификатор связанного сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:PullResponse>`, который может содержать следующие элементы:
  - `<wsen:EnumerationContext>` — идентификатор контекста перечисления;
  - `<wsen:Items>` — возвращаемые XML-элементы;
  - `<wsen:EndOfSequence/>` — указывает, что все элементы общей последовательности возвращены и дальнейшие запросы невозможны.

Запрашивающий Web-сервис может послать SOAP-сообщение с Renew-запросом Web-сервису источника данных для продления сессии перечисления. SOAP-сообщение с Renew-запросом содержит следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/Renew`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:ReplyTo>` — URI-адрес конечной точки получателя ответа на это сообщение;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- заголовок `<wsa:FaultTo>` — URI-адрес конечной точки получателя сообщения об ошибке для данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:Renew>`, содержащим следующие элементы:
  - обязательный элемент `<wsen:EnumerationContext>` — идентификатор контекста перечисления;
  - дополнительный элемент `<wsen:Expires>` — запрашиваемое время окончания сессии перечисления.

В ответ Web-сервис источника данных посылает SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/RenewResponse`;
- заголовок `<wsa:RelatesTo>` — идентификатор связанного сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;

- элемент `<SOAP:Body>` с дочерним элементом `<wsen:RenewResponse>`, который может содержать следующие элементы:
  - `<wsen:Expires>` — время окончания сессии перечисления;
  - `<wsen:EnumerationContext>` — идентификатор контекста перечисления.

Запрашивающий Web-сервис может послать SOAP-сообщение с GetStatus-запросом Web-сервису источника данных для получения информации о состоянии сессии перечисления. SOAP-сообщение с GetStatus-запросом содержит следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatus`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:ReplyTo>` — URI-адрес конечной точки получателя ответа на это сообщение;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- заголовок `<wsa:FaultTo>` — URI-адрес конечной точки получателя сообщения об ошибке для данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:GetStatus>`, который может содержать элемент `<wsen:EnumerationContext>` — идентификатор контекста перечисления.

В ответ Web-сервис источника данных посылает SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/GetStatusResponse`;
- заголовок `<wsa:RelatesTo>` — идентификатор связанного сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:GetStatusResponse>`, который может содержать элемент `<wsen:Expires>` — время окончания сессии перечисления.

Запрашивающий Web-сервис может послать SOAP-сообщение с Release-запросом Web-сервису источника данных для окончания сессии перечисления. SOAP-сообщение с Release-запросом содержит следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/Release`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:ReplyTo>` — URI-адрес конечной точки получателя ответа на это сообщение;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;

- элемент `<SOAP:Body>` с дочерним элементом `<wsen:Release>`, который должен содержать элемент `<wsen:EnumerationContext>` — идентификатор контекста перечисления.

В ответ Web-сервис источника данных посылает SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/ReleaseResponse`;
- заголовок `<wsa:RelatesTo>` — идентификатор связанного сообщения;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения.

В случае если Web-сервис источника данных по каким-то причинам неожиданно прервал сессию перечисления, он посылает SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/09/enumeration/EnumerationEnd`;
- заголовок `<wsa:To>` — URI-адрес получателя данного сообщения;
- элемент `<SOAP:Body>` с дочерним элементом `<wsen:EnumerationEnd>`, содержащим следующие элементы:
  - обязательный элемент `<wsen:EnumerationContext>` — идентификатор контекста перечисления;
  - обязательный элемент `<wsen:Code>` — код завершения сессии перечисления. Спецификация предопределяет два значения элемента `<wsen:Code>` — `http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceShuttingDown` (управляемая ошибка источника данных) и `http://schemas.xmlsoap.org/ws/2004/09/enumeration/SourceCancelling` (преждевременное окончание сессии перечисления по другим причинам);
  - дополнительный элемент `<wsen:Reason>` — описание причин преждевременного окончания сессии перечисления. Этот элемент имеет атрибут `xml:lang`, который указывает код языка текста.

Спецификация WS-Enumeration 2010 г. (пространство имен `http://www.w3.org/2010/03/ws-enu`) в SOAP-сообщении с Enumerate- и Renew-запросом для элемента `<wsen:Expires>` определяет атрибуты `min` (минимальное запрашиваемое время окончания сессии перечисления), `max` (максимальное время окончания сессии перечисления) и `exact` (если `true`, тогда атрибуты `min` и `max` игнорируются; по умолчанию — `false`).

Для атрибута `Dialect` элемента `<wsen:Filter>` SOAP-сообщения с Enumerate-запросом определяется значение по умолчанию `http://www.w3.org/2010/03/ws-enu/Dialects/XPath10`.

В ответном SOAP-сообщении на Enumerate-, Renew- и GetStatus-запрос в теле сообщения содержится элемент `<wsen:GrantedExpires>` вместо элемента `<wsen:Expires>`.

В ответном SOAP-сообщении на Release-запрос в теле сообщения содержится элемент `<wsen:ReleaseResponse>`.

Спецификация WS-Enumeration 2010 г. определяет утверждение политики Web-сервиса для поддержки спецификации WS-Enumeration, представленное элементом `<wsenu:DataSource>`, который может содержать следующие элементы:

- `<wsenu:FilterDialect>` — требование поддержки языка выражений, указанного с помощью атрибута `URI`;
- `<wsenu:MaxExpires>` — требование, указывающее значение максимального времени жизни сессии перечисления;
- `<wsenu:MaxTime>` — требование, указывающее значение максимального количества времени для формирования ответа Web-сервисом источника данных;
- `<wsenu:MaxElements>` — требование, указывающее максимальное количество возвращаемых элементов;
- `<wsenu:MaxCharacters>` — требование, указывающее максимальный размер возвращаемых элементов;
- `<wsenu:EndToSupported/>` — требование поддержки элемента `<wsen:EndTo>`.

WS-Eventing

Спецификация WS-Eventing определяет механизм, позволяющий одному Web-сервису зарегистрироваться на получение уведомлений о событиях от другого Web-сервиса.

Для того чтобы Web-сервису зарегистрироваться на подписку событий, он должен послать SOAP-сообщение с Subscribe-запросом Web-сервису, который является источником событий.

Спецификация WS-Eventing 2006 г. (пространство имен `http://schemas.xmlsoap.org/ws/2004/08/eventing`) определяет следующие элементы SOAP-сообщения с Subscribe-запросом:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:Subscribe>` (пространство имен `http://schemas.xmlsoap.org/ws/2004/08/eventing`), содержащим следующие элементы:
  - `<wse:EndTo>` — адрес конечной точки, куда Web-сервис событий отправит сообщение в случае, если подписка неожиданно прерывается;
  - `<wse:Delivery>` — адрес доставки сообщений о событиях. Этот элемент имеет атрибут `Mode`, указывающий режим доставки сообщений о событиях. Спецификация предопределяет значение атрибута `Mode` как `Mode="http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push"`, означающее, что сообщение о событии является простым асинхронным сообщением. При

этом элемент `<wse:Delivery>` содержит элемент `<wse:NotifyTo>` с адресом конечной точки, куда должны отправляться сообщения о событиях;

- `<wse:Expires>` — запрашиваемое время окончания подписки;
- `<wse:Filter>` — выражение, фильтрующее отправку сообщений о событиях. Атрибут `Dialect` указывает язык выражений, по умолчанию значение атрибута равно `http://www.w3.org/TR/1999/REC-xpath-19991116` — XPath 1.0 язык выражений.

В ответ Web-сервис событий отправляет SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:SubscribeResponse>`, содержащим следующие элементы:
  - `<wse:SubscriptionManager>` — адрес конечной точки Web-сервиса — менеджера подписки, которому нужно отправлять запросы получения статуса подписки, продления подписки и удаления подписки;
  - `<wse:Expires>` — время окончания подписки.

Для продления подписки Web-сервис может отправить SOAP-сообщение с Renew-запросом, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/Renew`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:Renew>`, который может включать в себя элемент `<wse:Expires>` — запрашиваемое время окончания подписки.

В ответ Web-сервис, являющийся менеджером подписки, отправляет SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/RenewResponse`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:RenewResponse>`, имеющим дочерний элемент `<wse:Expires>`, который указывает время окончания подписки.

Для получения статуса подписки Web-сервис может отправить SOAP-сообщение с GetStatus-запросом, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatus`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:GetStatus>`.

В ответ Web-сервис, являющийся менеджером подписки, отправляет SOAP-сообщение со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/GetStatusResponse`;

- элемент `<SOAP:Body>` с дочерним элементом `<wse:GetStatusResponse>`, имеющим дочерний элемент `<wse:Expires>`, который указывает время окончания подписки.

Для удаления подписки Web-сервис может отправить SOAP-сообщение с Unsubscribe-запросом, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/Unsubscribe`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:Unsubscribe>`.

В ответ Web-сервис, являющийся менеджером подписки, отправляет SOAP-сообщение с заголовком `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/UnsubscribeResponse`.

В случае если подписка неожиданно прерывается, Web-сервис событий отправляет SOAP-сообщение, содержащее следующие элементы:

- заголовок `<wsa:Action>` со значением `http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscriptionEnd`;
- элемент `<SOAP:Body>` с дочерним элементом `<wse:SubscriptionEnd>`, содержащим следующие элементы:
- `<wse:SubscriptionManager>` — адрес конечной точки Web-сервиса — менеджера подписки;
- `<wse:Status>` — код завершения подписки. Спецификация предопределяет следующие значения элемента `<wse:Status>`:
  - `http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryFailure` — проблемы с доставкой сообщений о событиях;
  - `http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceShuttingDown` — ошибка источника событий;
  - `http://schemas.xmlsoap.org/ws/2004/08/eventing/SourceCancelling` — завершение подписки по каким-либо другим причинам;
- `<wse:Reason>` — текстовое описание причин завершения подписки. Атрибут `xml:lang` указывает язык текста.

Спецификация WS-Eventing 2010 г. (пространство имен `http://www.w3.org/2010/03/ws-evt`) в SOAP-сообщении с Subscribe-запросом не определяет для элемента `<wse:Delivery>` атрибут `Mode`, а для элемента `<wse:Expires>` определяет атрибуты `min` (минимальное запрашиваемое время окончания подписки), `max` (максимальное время окончания подписки) и `exact` (если `true`, тогда атрибуты `min` и `max` игнорируются; по умолчанию — `false`), а также дополнительно определяет элемент `<wse:Format>`, который указывает формат сообщений о событиях с помощью атрибута `Name`, имеющего следующие возможные значения:

- `http://www.w3.org/2010/03/ws-evt/DeliveryFormats/Unwrap` — сообщения о событиях не содержат SOAP-элементы, определенные спецификацией;
- `http://www.w3.org/2010/03/ws-evt/DeliveryFormats/Wrap` — сообщения о событиях содержат SOAP-элементы, определенные спецификацией.

Для атрибута `Dialect` элемента `<wse:Filter>` спецификация WS-Eventing 2010 г. определяет значение по умолчанию `http://www.w3.org/2010/03/ws-evt/Dialects/XPath10`.

В ответ на Subscribe-, Renew- или GetStatus-запрос SOAP-сообщение содержит вместо элемента `<wse:Expires>` элемент `<wse:GrantedExpires>`, указывающий время окончания подписки.

Для Renew-запроса элемент `<wse:Expires>` также имеет атрибуты `min` (минимальное запрашиваемое время окончания подписки), `max` (максимальное время окончания подписки) и `exact` (если `true`, тогда атрибуты `min` и `max` игнорируются; по умолчанию — `false`).

В ответ на Unsubscribe-запрос тело SOAP-сообщения содержит элемент `<wse:UnsubscribeResponse>`.

Спецификация WS-Eventing 2010 г. определяет утверждение политики Web-сервиса событий для поддержки спецификации WS-Eventing, представленное элементом `<wse:EventSource>`, который может содержать следующие элементы:

- `<wse:DateTimeSupported/>` — время окончания подписки указывается не как период, а как дата/время;
- `<wse:FilterDialect>` — требование поддержки языка выражений, указываемого с помощью атрибута `URI`;
- `<wse:MaxExpires>` — максимальное время подписки;
- `<wse:FormatName>` — требование поддержки определенного формата сообщений о событиях, указываемого с помощью атрибута `URI`;
- `<wse:EndToSupported/>` — требование поддержки элемента `<wse:EndTo>`;
- `<wse:NotificationPolicy>` — определяет политику конечной точки Web-сервиса событий с помощью дочернего элемента `<wsp:Policy>` или `<wsp:PolicyReference>`.

Спецификация WS-Eventing 2010 г. определяет утверждение политики Web-сервиса — менеджера подписки для поддержки спецификации WS-Eventing, представленное элементом `<wse:SubscriptionManager>`, который может содержать следующие элементы:

- `<wse:DateTimeSupported/>` — время окончания подписки указывается не как период, а как дата/время;
- `<wse:GetStatusOperationSupported/>` — требование поддержки запроса статуса подписки;
- `<wse:UnsubscribeOperationSupported/>` — требование поддержки запроса на прекращение подписки;
- `<wse:MaxExpires>` — максимальное время подписки.

Спецификация WS-EventDescriptions определяет для Web-сервиса событий XML-документ, с помощью которого он может описать свои события.

Корневым элементом такого документа является элемент `<wsevd:EventDescriptions>` (пространство имен `http://www.w3.org/2010/03/ws-evd`), имеющий следующие атрибуты и дочерние элементы:

- атрибут `targetNamespace` — пространство имен типа событий;
- элемент `<wsevd:types>` — определение типа событий с помощью вложенного элемента `<xs:schema>` (пространство имен `http://www.w3.org/2001/XMLSchema`);
- элемент `<wsevd:eventType>` — указание типа события с помощью атрибутов `name` (имя типа события), `element` (ссылка на XML-схему типа события, определенную в элементе `<wsevd:types>`) и `actionURI` (URI-идентификатор содержимого события).

Документ описания событий Web-сервиса может быть включен непосредственно в утверждение политики `<wse:EventSource>` или его представление соответствует спецификации WS-MetadataExchange.

WS-Management

Спецификация WS-Management определяет общий SOAP-протокол для управления такими системами, как персональные компьютеры, серверы, различные устройства, Web-сервисы и другие приложения.

Спецификация WS-Management описывает механизм, основанный на технологии Web-сервисов, с помощью которого компоненты распределенной системы получают доступ и обмениваются управляющей информацией, что дает возможность администраторам осуществлять удаленное управление различными устройствами.

Управление компонентами распределенной системы осуществляется с применением Web-сервисов, реализующих такие функции, как операции `Get`, `Put`, `Create` и `Delete` для системных значений компонентов, последовательный доступ к большим таблицам и журналам регистраций, подписка на получение событий и исполнение управляющих операций с входящими и выходящими параметрами.

Таким образом, спецификация WS-Management опирается на спецификации WS-Addressing, WS-Eventing, WS-Enumeration, WS-Transfer, WS-Policy, WS-Trust, WS-Security, определяя для SOAP-протокола расширения спецификации WS-Management.

Для спецификации WS-Addressing спецификация WS-Management определяет элементы `<wsman:ResourceURI>` и `<wsman:SelectorSet>` (пространство имен `xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"`), являющиеся дочерними элементами элемента `<wsa:ReferenceParameters>`.

Элемент `<wsman:ResourceURI>` содержит URI-идентификатор класса ресурса или экземпляра класса ресурса. Данный URI вместе с URI элемента `<wsa:To>` формируют полный адрес класса ресурса или экземпляра класса ресурса конечной точки Web-сервиса. Элемент `<wsman:SelectorSet>` содержит один или несколько элементов `<wsman:Selector>`, указывающих идентификаторы экземпляров класса ресурса.

Элемент `<wsman:Selector>` имеет атрибут `Name` — имя селектора. Селектор может иметь имя EPR — адрес конечной точки, и включать в себя элемент `<wsa:EndpointReference>` с элементами `<wsman:ResourceURI>` и `<wsman:Selector>`.

Спецификация WS-Management также определяет ряд общих заголовков для SOAP-сообщения:

- заголовок `<wsman:OperationTimeout>` указывает время, в течение которого запрашивающая сторона ожидает ответа;
- заголовок `<wsman:MaxEnvelopeSize>` указывает максимальный размер ответного SOAP-сообщения;
- заголовок `<wsman:Locale>` указывает локализацию для ответного SOAP-сообщения с помощью атрибута `xml:lang`;
- заголовок `<wsman:OptionSet>` используется для передачи Web-сервису набора опций, применяемых при обработке запроса. Дочерние элементы `<wsman:Option>`, содержащие значения свойств, имеют атрибуты `Name` (идентификатор опции), `MustComply` (если `true`, тогда опция должна быть применена) и `Type` (тип значения опции).

В случае если запрос ресурса способен изменить его адрес конечной точки, запрашивающая сторона может включить в запрос заголовок `<wsman:RequestEPR/>`, указывающий, что в ответном сообщении требуется наличие заголовка `<wsman:RequestedEPR>` со следующими дочерними элементами:

- `<wsa:EndpointReference>` содержит новый адрес конечной точки;
- `<wsman:EPRInvalid/>` указывает, что Web-сервис не смог сформировать новый адрес конечной точки для его возврата, хотя адрес конечной точки был изменен в результате запроса;
- `<wsman:EPRUnknown/>` указывает, что Web-сервис не определяет, какой из двух адресов конечной точки действителен — старый или новый адрес.

Для частичного доступа к ресурсу используется заголовок `<wsman:Fragment Transfer>`, содержащий выражение, идентифицирующее требуемый фрагмент ресурса. Атрибут заголовка `Dialect` указывает язык выражения, который по умолчанию является языком XPath 1.0. В ответном SOAP-сообщении в его теле содержится элемент `<wsman:XmlFragment>` с фрагментом ресурса.

Для Enumerate- и Pull-запросов спецификации WS-Enumeration определен заголовок `<wsman:RequestTotalItemsCountEstimate/>`, указывающий, что в ответном сообщении требуется наличие заголовка `<wsman:TotalItemsCountEstimate>`, содержащего общее количество элементов в последовательности.

Для элемента `<wsen:Enumerate>` спецификация WS-Management определяет два дополнительных дочерних элемента: `<wsman:OptimizeEnumeration/>` (указывает, что в ответном сообщении должна содержаться вся последовательность) и `<wsman:MaxElements>` (указывает максимальное количество возвращаемых элементов последовательности). При наличии данных элементов в Enumerate-запросе ответное

сообщение включает в себя элемент `<wsen:EnumerateResponse>` с дочерними элементами `<wsman:Items>` (содержит элементы последовательности) и `<wsman:EndOfSequence/>` (указывает конец последовательности).

Для элемента `<wsen:Enumerate>` также определен дочерний элемент `<wsman:Filter>` с атрибутом `Dialect` (указывает язык выражения), позволяющий фильтровать создаваемый контекст перечисления.

Иногда требуется создавать контекст перечисления, включающий в себя как сами объекты, так и их адреса конечных точек. Для этого определен дочерний элемент `<wsman:EnumerationMode>` элемента `<wsen:Enumerate>`, имеющий следующие возможные значения:

- `EnumerateEPR` — создается последовательность только из адресов конечных точек объектов;
- `EnumerateObjectAndEPR` — создается последовательность из объектов и их адресов конечных точек.

Для спецификации WS-Eventing спецификация WS-Management определяет дочерний элемент `<wsman:Filter>` элемента `<wse:Subscribe>` с атрибутом `Dialect` (указывает язык выражения), позволяющий фильтровать создаваемую подписку на события.

Кроме того, определен дочерний элемент `<wsman:ConnectionRetry>` элемента `<wse:Subscribe>`, указывающий, что если подписанная на получение событий сторона недоступна для доставки сообщения о событии, тогда Web-сервис событий должен повторить попытку соединения. Элемент `<wsman:ConnectionRetry>` имеет атрибут `Total` с количеством попыток установить соединение и содержит время ожидания между попытками установить соединение.

Для того чтобы подписанная сторона знала о том, что подписка действует, для элемента `<wse:Delivery>` определен дочерний элемент `<wsman:Heartbeats>`, указывающий, что Web-сервис событий должен после создания подписки периодически посылать псевдособытия подписанной стороне. Элемент `<wsman:Heartbeats>` содержит временной интервал между отправкой псевдособытий.

Сообщение с псевдособытием содержит заголовок `<wsa:Action>` со значением `http://schemas.dmtf.org/wbem/wsman/1/wsman/Heartbeat` и пустое тело сообщения.

Чтобы обеспечить надежность доставки сообщений о событиях, спецификация WS-Management определяет механизм закладок. Данный механизм предполагает следующие шаги:

1. Включение в элемент `<wse:Subscribe>` тела Subscribe-запроса элемента `<wsman:SendBookmarks/>`, указывающего, что Web-сервис событий должен в сообщении о событии посылать закладку. Закладка связана с событием с помощью журнала регистрации, который должен вести Web-сервис событий.
2. В сообщении о событии присутствует заголовок `<wsman:Bookmark>`, содержащий маркер — указатель на событие.

3. Если по каким-то причинам подписка на события прерывается, и подписанная сторона желает получать события, начиная с последнего полученного, тогда она делает Subscribe-запрос с дочерними элементами `<wsman:Bookmark>` (последняя полученная закладка) и `<wsman:SendBookmarks/>` (Web-сервис событий должен в дальнейшем посылать закладки) элемента `<wse:Subscribe>`.
4. Web-сервис событий, ориентируясь по последней закладке и журналу регистрации, восстанавливает подписку.

Для атрибута `Mode` элемента `<wse:Delivery>` спецификация WS-Management определяет, дополнительно к значению `http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push`, следующие значения режима доставки сообщений о событиях:

- `http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck` — каждое SOAP-сообщение содержит информацию только об одном событии. При этом Web-сервис событий формирует очередь из событий и отправляет их последовательно одно за другим. Получатель сообщения должен отправить подтверждение о получении сообщения о событии, после получения подтверждения Web-сервис событий отправит следующее событие;
- `http://schemas.dmtf.org/wbem/wsman/1/wsman/Events` — каждое SOAP-сообщение может содержать несколько событий, которые организуются в пакет. При этом создается очередь из пакетов. От получателя требуется отправка подтверждения для каждого пакета;
- `http://schemas.dmtf.org/wbem/wsman/1/wsman/Pull` — синхронное получение событий.

Для режима доставки `http://schemas.xmlsoap.org/ws/2004/08/eventing/DeliveryModes/Push` и `http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck` дополнительно определен дочерний элемент `<wsman:MaxEnvelopeSize>` элемента `<wse:Delivery>`, содержащий максимальный размер сообщения о событии. Элемент `<wsman:MaxEnvelopeSize>` имеет атрибут `Policy` со значениями `CancelSubscription` (прекратить подписку при первом сообщении о событии, превышающем указанный размер), `Skip` (исключать сообщения, превышающие указанный размер) и `Notify` (уведомлять о потере событий). Уведомление о потере событий подразумевает, что Web-сервис событий отправляет сообщение с заголовком `<wsa:Action>` — `http://schemas.dmtf.org/wbem/wsman/1/wsman/DroppedEvents` и с элементом `<wsman:DroppedEvents>` тела сообщения. Элемент `<wsman:DroppedEvents>` содержит общее число потерянных событий и имеет атрибут `Action` — URI-действия событий.

Для режима доставки `http://schemas.dmtf.org/wbem/wsman/1/wsman/PushWithAck` и `http://schemas.dmtf.org/wbem/wsman/1/wsman/Events` требуется отправка подтверждения о получении события или пакета событий. Подтверждение — это ответное сообщение на сообщение события. При этом сообщение события включает в себя заголовок `<wsa:ReplyTo>` — адрес для подтверждения и заголовок `<wsman:AckRequested/>`, требующий подтверждения. Подтверждение содержит заголовки `<wsa:To>` (адрес доставки подтверждения), `<wsa:Action>` (значение `http://schemas.dmtf.org/wbem/wsman/1/wsman/Ack`) и `<wsa:RelatesTo>` (идентификатор сообщения события).

При пакетировании событий элемент `<wsme:Delivery>` (значение атрибута `Mode="http://schemas.dmtf.org/wbem/wsman/1/wsman/Events"`) Subscribe-запроса дополнительно содержит элементы:

- `<wsman:MaxElements>` — максимальное количество событий в пакете;
- `<wsman:MaxTime>` — максимальное время формирования пакета;
- `<wsman:MaxEnvelopeSize>` — максимальный размер SOAP-сообщения с пакетом. Атрибут `Policy` имеет значения `CancelSubscription`, `Skip` и `Notify`.

SOAP-сообщение с пакетом содержит заголовок `<wsa:Action>` со значением `http://schemas.dmtf.org/wbem/wsman/1/wsman/Events` и тело сообщения с элементом `<wsman:Events>`, который имеет дочерние элементы `<wsman:Event>` (атрибут `<wsman:Event>` — значение заголовка `<wsa:Action>` для события). Каждый элемент `<wsman:Event>` несет тело сообщения отдельного события.

Спецификация WS-Management также определяет дочерний элемент `<wsman:Auth>` элемента `<wse:Delivery>` Subscribe-запроса, содержащий информацию аутентификации, используемую Web-сервисом событий. Элемент `<wsman:Auth>` имеет атрибут `Profile` — URI-идентификатор профиля безопасности для доставки событий. Спецификация предопределяет значение атрибута `Profile` как `http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/https/mutual` (стандартная взаимная аутентификация с использованием сертификата) и `http://schemas.dmtf.org/wbem/wsman/1/wsman/secprofile/http/digest` (HTTP Basic- или Digest-аутентификация).

Спецификация WS-Management определяет расширения для запроса/ответа, которые упрощают установление связи между Web-сервисами путем передачи информации о запрашиваемом Web-сервисе. Запрос специфической информации о Web-сервисе содержит в теле сообщения элемент `<wsmid:Identify>` (пространство имен `xmlns:wsmid="http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd"`). Ответное сообщение содержит в теле сообщения элемент `<wsmid:IdentifyResponse>` с дочерними элементами:

- обязательный элемент `<wsmid:ProtocolVersion>` — URI-идентификатор поддерживаемой версии WS-Management;
- дополнительный элемент `<wsmid:ProductVendor>` — идентификатор производителя реализации WS-Management;
- дополнительный элемент `<wsmid:ProductVersion>` — версия реализации WS-Management;
- дополнительный элемент `<wsmid:InitiativeSupport>` — технологии, поддерживаемые реализацией WS-Management с помощью дочерних элементов `<wsmid:InitiativeName>` (имя технологии) и `<wsmid:InitiativeVersion>` (версия технологии);
- дополнительный элемент `<wsmid:SecurityProfiles>` — профили безопасности, поддерживаемые реализацией WS-Management с помощью дочерних элементов `<wsmid:SecurityProfileName>` (имя профиля);

- дополнительный элемент `<wsmid:AddressingVersionURI>` — URI-идентификатор версии WS-Addressing, поддерживаемой реализацией WS-Management.

WS-Discovery

Спецификация WS-Discovery предлагает альтернативный технологии UDDI механизм публикации и поиска Web-сервисов.

В технологии UDDI создаются реестры, которые хранят информацию о Web-сервисах и управляются UDDI-серверами. Для публикации и поиска конкретного Web-сервиса достаточно связаться по определенному адресу с UDDI-сервером.

Публикация и поиск Web-сервисов технологии WS-Discovery основаны на групповой передаче сообщений по протоколу UDP (User Datagram Protocol) Multicast.

Для объявления своего присутствия в сети Web-сервис передает для определенной IP-адресной группы сообщение Hello, содержащее информацию о Web-сервисе. Соответственно данное сообщение получают слушатели, присоединенные к адресной группе. При прекращении своей работы в сети Web-сервис передает сообщение Bye.

Для того чтобы клиент смог найти Web-сервис, определено сообщение Probe, которое также передается по протоколу UDP Multicast. Слушающий Web-сервис, после получения такого сообщения посылает клиенту ответное UDP Unicast-сообщение ProbeMatch с информацией о Web-сервисе. Клиент может уточнить информацию о Web-сервисе с помощью UDP Multicast-сообщения Resolve и ответного UDP Unicast-сообщения ResolveMatch от Web-сервиса.

Спецификация WS-Discovery определяет два режима публикации и поиска Web-сервисов.

Первый режим Managed Mode подразумевает наличие Proxy-сервиса, который знает информацию о нужном Web-сервисе, и клиент посылает свои Probe- и Resolve-сообщения Proxy-сервису. Для своей публикации Web-сервис посылает Hello-сообщение Proxy-сервису. Сообщение Bye также посылается Proxy-сервису. В данном режиме все сообщения посылаются по протоколу Unicast, т. е. от точки к точке.

Во втором режиме Ad hoc Mode клиент посылает сообщения Probe и Resolve по протоколу Multicast для запрашиваемого Web-сервиса, который посылает Multicast сообщения Hello и Bye для клиентов. Если при этом Proxy-сервис все же присутствует, то он выступает как посредник при передаче Multicast-сообщений Probe/Resolve и Hello/Bye. Возможно переключение из режима Ad hoc Mode в режим Managed Mode, при этом Proxy-сервис в ответ на Multicast-сообщения Probe и Resolve клиента посылает ему Unicast-сообщение Hello, и режим становится Managed Mode. При переключении режимов Proxy-сервис слушает Multicast-сообщения Hello и Bye и сохраняет информацию о Web-сервисе, чтобы затем работать с клиентом в режиме Managed Mode.

SOAP-сообщение Hello имеет следующие элементы:

- заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Hello`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:RelatesTo>` присутствует при переключении режимов и содержит идентификатор Multicast-сообщения Probe или Resolve клиента;
- заголовок `<wsa:To>` со значением `urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01` в режиме Ad hoc Mode или адресом конечной точки в режиме Managed Mode;
- заголовок `<d:AppSequence/>` (пространство имен `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01`), в режиме Ad hoc Mode позволяет получателю упорядочивать сообщения. Элемент `<d:AppSequence/>` имеет атрибуты `InstanceId` (счетчик, увеличивающийся при перезагрузке Web-сервиса и при его возврате в сеть после потери состояния), `SequenceId` (идентификатор последовательности сообщений, по умолчанию `0`), `MessageNumber` (идентификатор сообщения в последовательности);
- в элементе `<SOAP:Body>` дочерний элемент `<d:Hello>`, содержащий следующие элементы:
  - `<wsa:EndpointReference>` — информация о конечной точке Web-сервиса, указываемая с помощью элементов `<wsa:Address>`, `<wsa:ReferenceParameters>`, `<wsa:Metadata>`. Элемент `<wsa:Address>` необязательно содержит транспортный адрес конечной точки в виде "http://...", это может быть URN-идентификатор в виде "urn:uuid: ...";
  - `<d:Types>` — тип Web-сервиса;
  - `<d:Scopes>` — URI-идентификатор области применения Web-сервиса;
  - `<d:XAddrs>` — транспортный адрес Web-сервиса;
  - `<d:MetadataVersion>` — счетчик изменений метаданных Web-сервиса.

SOAP-сообщение Bye имеет следующие элементы:

- заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Bye`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:To>` со значением `urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01` в режиме Ad hoc Mode или адресом конечной точки в режиме Managed Mode;
- заголовок `<d:AppSequence/>`, в режиме Ad hoc Mode позволяет получателю упорядочивать сообщения;
- в элементе `<SOAP:Body>` дочерний элемент `<d:Bye>`, содержащий следующие элементы:

- `<wsa:EndpointReference>` — информация о конечной точке Web-сервиса, указываемая с помощью элементов `<wsa:Address>`, `<wsa:ReferenceParameters>`, `<wsa:Metadata>`. Элемент `<wsa:Address>` необязательно содержит транспортный адрес конечной точки в виде "http://...", это может быть URN-идентификатор в виде "urn:uuid: ...";
- `<d:Types>` — тип Web-сервиса;
- `<d:Scopes>` — URI-идентификатор области применения Web-сервиса;
- `<d:XAddrs>` — транспортный адрес Web-сервиса;
- `<d:MetadataVersion>` — счетчик изменений метаданных Web-сервиса.

SOAP-сообщение Probe имеет следующие элементы:

- заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Probe`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:ReplyTo>` — адрес конечной точки для ответа, по умолчанию `http://www.w3.org/2005/08/addressing/anonymous`;
- заголовок `<wsa:To>` со значением `urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01` для запрашиваемого Web-сервиса или адресом конечной точки для Proxy-сервиса;
- в элементе `<SOAP:Body>` дочерний элемент `<d:Probe>`, содержащий следующие элементы:
  - `<d:Types>` — тип Web-сервиса;
  - `<d:Scopes>` — URI-идентификатор области применения Web-сервиса. Атрибут `MatchBy` указывает тип идентификатора с помощью следующих возможных значений:
    - `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/rfc3986` (по умолчанию) — спецификация RFC 3986;
    - `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/uuid` — спецификация RFC 4122;
    - `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ldap` — спецификация RFC 3986, RFC 4514 и RFC 4516;
    - `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/strcmp0` — строка;
    - `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/none` — схема не определена.

Ответное SOAP-сообщение ProbeMatch имеет следующие элементы:

- заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ProbeMatch`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;

- заголовок `<wsa:RelatesTo>` — идентификатор сообщения Probe;
- заголовок `<wsa:To>` — адрес конечной точки;
- заголовок `<d:AppSequence/>`, в режиме Ad hoc Mode позволяет получателю упорядочивать сообщения;
- в элементе `<SOAP:Body>` дочерний элемент `<d:ProbeMatches>`, содержащий следующие элементы:
  - `<wsa:EndpointReference>` — информация о конечной точке Web-сервиса, указываемая с помощью элементов `<wsa:Address>`, `<wsa:ReferenceParameters>`, `<wsa:Metadata>`. Элемент `<wsa:Address>` необязательно содержит транспортный адрес конечной точки в виде "http://...", это может быть URN-идентификатор в виде "urn:uuid: ...";
  - `<d:Types>` — тип Web-сервиса;
  - `<d:Scopes>` — URI-идентификатор области применения Web-сервиса;
  - `<d:XAddrs>` — транспортный адрес Web-сервиса;
  - `<d:MetadataVersion>` — счетчик изменений метаданных Web-сервиса.

Сообщения Hello и ProbeMatch могут содержать только элемент `<wsa:Endpoint Reference>`, который, как уже было сказано ранее, в элементе `<wsa:Address>` может указывать не транспортный адрес Web-сервиса, а его URN-идентификатор. В этом случае клиенту требуется запросить дополнительную информацию о Web-сервисе с помощью сообщения Resolve, включающего в себя следующие элементы:

- заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Resolve`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:ReplyTo>` — адрес конечной точки для ответа, по умолчанию `http://www.w3.org/2005/08/addressing/anonymous`;
- заголовок `<wsa:To>` со значением `urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01` для запрашиваемого Web-сервиса или адресом конечной точки для Proxy-сервиса;
- в элементе `<SOAP:Body>` дочерний элемент `<d:Resolve>`, содержащий элемент `<wsa:EndpointReference>`.

В ответ на Resolve-запрос посылается сообщение ResolveMatch со следующими элементами:

- заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ResolveMatch`;
- заголовок `<wsa:MessageID>` — идентификатор сообщения;
- заголовок `<wsa:RelatesTo>` — идентификатор Resolve-сообщения клиента;
- заголовок `<wsa:To>` — адрес конечной точки;

- заголовок `<d:AppSequence/>`, в режиме Ad hoc Mode позволяет получателю упорядочивать сообщения;
- в элементе `<SOAP:Body>` дочерний элемент `<d:ResolveMatches>`, содержащий следующие элементы:
  - `<wsa:EndpointReference>` — информация о конечной точке Web-сервиса, указываемая с помощью элементов `<wsa:Address>`, `<wsa:ReferenceParameters>`, `<wsa:Metadata>`;
  - `<d:Types>` — тип Web-сервиса;
  - `<d:Scopes>` — URI-идентификатор области применения Web-сервиса;
  - `<d:XAddrs>` — транспортный адрес Web-сервиса;
  - `<d:MetadataVersion>` — счетчик изменений метаданных Web-сервиса.

Для указания информации о цифровой подписи сообщений спецификация WS-Discovery определяет заголовок `<d:Security>` как альтернативу заголовка `<wsse:Security>`. Заголовок `<d:Security>` может иметь дочерний элемент `<d:Sig>`, который указывает информацию о цифровой подписи с помощью атрибутов:

- обязательный атрибут `Scheme` — схема цифровой подписи. Предопределенное значение атрибута — `http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/rsa`;
- дополнительный атрибут `KeyId` — идентификатор ключа для цифровой подписи;
- обязательный атрибут `Refs` — идентификаторы подписанных частей сообщения. Части сообщения идентифицируются с помощью глобального атрибута `d:Id`;
- дополнительный атрибут `PrefixList` — список префиксов включаемых пространств имен, передаваемых при канонизации документа;
- обязательный атрибут `Sig` — Base64-значение цифровой подписи.

WS-ReliableMessaging

Спецификация WS-ReliableMessaging определяет механизм, обеспечивающий надежность передачи сообщений между компонентами распределенной системы, предотвращая потерю, дублирование и изменение порядка сообщений.

Протокол надежного обмена сообщениями спецификации WS-ReliableMessaging реализуется с помощью агентов RM Source и RM Destination, являющихся частью компонентов распределенной системы.

Агент RM Source отвечает за:

- запрос на создание и окончание контекста надежного обмена сообщениями;
- добавление соответствующих заголовков в сообщения;
- повторную отправку сообщений в случае их потери.

Агент RM Destination отвечает за:

- ответ на запрос создания и прекращения контекста надежного обмена сообщениями;

- прием сообщений и отправку подтверждений получения сообщений;
- исключение дублирующих сообщений;
- восстановление порядка сообщений.

Если сообщения являются однонаправленными, тогда запрашивающая сторона имеет только агента RM Source, а запрашиваемая сторона — RM Destination. В случае двунаправленного обмена каждая сторона содержит по два агента.

Для обмена сообщениями с привлечением агентов RM Source и RM Destination вначале должна быть создана сессия надежного обмена. Сессия надежного обмена сообщениями создается с помощью формирования последовательности сообщений Sequence.

SOAP-сообщение, содержащее CreateSequence-запрос агента RM Source агенту RM Destination, включает в теле сообщения элемент `<wsrm:CreateSequence>` (рис. 2.10).

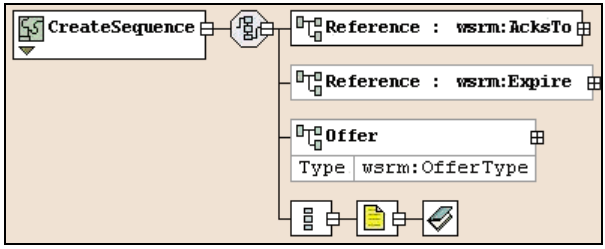


Рис. 2.10. Общая схема элемента `<wsrm:CreateSequence>`

Спецификация WS-ReliableMessaging имеет пространство имен:

- `http://schemas.xmlsoap.org/ws/2005/02/rm` — версия 1.0;
- `http://docs.oasis-open.org/ws-rx/wsrm/200702` — версии 1.1 и 1.2.

Элемент `<wsrm:CreateSequence>` имеет следующие дочерние элементы:

- обязательный элемент `<wsrm:AcksTo>` — элемент типа элемента `<wsa:EndpointReferenceType>` (спецификация WS-Addressing), указывает адрес той конечной точки, в которую должны отправляться подтверждения о получении сообщений агентом RM Destination;
- дополнительный элемент `<wsrm:Expires>` — запрашиваемое время жизни последовательности;
- дополнительный элемент `<wsrm:Offer>` — с помощью данного элемента агент RM Source предлагает свою последовательность для надежного обмена сообщениями. Элемент `<wsrm:Offer>` имеет дочерние элементы:
  - обязательный элемент `<wsrm:Identifier>` — URI-идентификатор последовательности;
  - обязательный элемент `<wsrm:Endpoint>` — адрес конечной точки, по которому агент RM Source может посылать запросы на завершение последовательности, запросы получения подтверждения доставки сообщений, а также отправлять сообщения об ошибках;

- дополнительный элемент `<wsrm:Expires>` — определяет время жизни последовательности;
- дополнительный элемент `<wsrm:IncompleteSequenceBehavior>` — поведение агента RM Destination при принудительном завершении последовательности. Возможные значения элемента `<wsrm:IncompleteSequenceBehavior>`: `DiscardEntireSequence` (последовательность завершается без отправки оставшихся подтверждений о получении сообщений), `DiscardFollowingFirstGap` (последовательность завершается с отправкой только первого оставшегося подтверждения), `NoDiscard` (последовательность завершается с отправкой всех оставшихся подтверждений).

В ответ на CreateSequence-запрос агент RM Destination посылает SOAP-сообщение, содержащее в теле сообщения элемент `<wsrm:CreateSequenceResponse>` (рис. 2.11), который имеет следующие дочерние элементы:

- обязательный элемент `<wsrm:Identifier>` — URI-идентификатор последовательности;
- дополнительный элемент `<wsrm:Expires>` — время жизни последовательности;

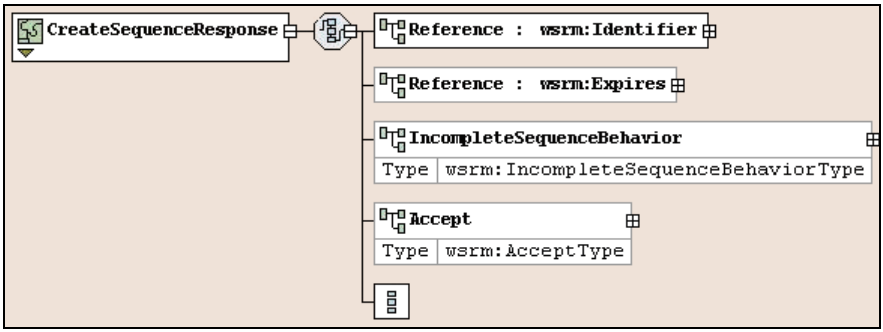


Рис. 2.11. Общая схема элемента `<wsrm:CreateSequenceResponse>`

- дополнительный элемент `<wsrm:IncompleteSequenceBehavior>` — поведение агента RM Destination при принудительном завершении последовательности. Возможные значения элемента `<wsrm:IncompleteSequenceBehavior>`: `DiscardEntireSequence` (последовательность завершается без отправки оставшихся подтверждений о получении сообщений), `DiscardFollowingFirstGap` (последовательность завершается с отправкой только первого оставшегося подтверждения), `NoDiscard` (последовательность завершается с отправкой всех оставшихся подтверждений);
- дополнительный элемент `<wsrm:Accept>` — принятие предлагаемой агентом RM Source последовательности. Элемент `<wsrm:Accept>` содержит элемент `<wsrm:AcksTo>`, указывающий адрес конечной точки, по которому будут отправляться подтверждения о получении сообщений и сообщения об ошибках.

Спецификация WS-ReliableMessaging версии 1.0 не определяет элементы `<wsrm:Endpoint>` и `<wsrm:IncompleteSequenceBehavior>`.

Элемент `<wsrm:CreateSequence>` может дополнительно содержать элемент `<wsse:SecurityTokenReference>` для ссылки на маркер защиты, используемый для обеспечения безопасности последовательности. В этом случае CreateSequence-запрос должен нести заголовок `<wsrm:UsesSequenceSTR soap:mustUnderstand="true"/>`, гарантирующий понимание агентом RM Destination элемента `<wsse:SecurityTokenReference>`. Если же агент RM Destination не может обработать дочерний элемент `<wsse:SecurityTokenReference>` элемента `<wsrm:CreateSequence>`, он должен сгенерировать ошибку `soap:MustUnderstand` заголовка `<wsrm:UsesSequenceSTR>`.

Другой способ защиты последовательности — это создание защищенной сессии SSL/TLS. При этом CreateSequence-запрос должен нести заголовок `<wsrm:UsesSequenceSSL soap:mustUnderstand="true"/>`, гарантирующий использование протокола SSL/TLS агентом RM Destination.

Спецификация WS-ReliableMessaging версии 1.0 не определяет заголовки `<wsrm:UsesSequenceSTR>` и `<wsrm:UsesSequenceSSL>`.

В случае двунаправленного обмена сообщениями каждая сторона имеет агентов RM Source и RM Destination, и должны быть созданы две независимые последовательности сообщений.

Агент RM Source или агент RM Destination могут принудительно завершить последовательность двумя способами. Первый способ — это отправка CloseSequence-сообщения, при этом агент RM Destination не сбрасывает состояние закрытой последовательности, и агент RM Source может запросить подтверждение получения сообщений или послать запрос на завершение последовательности вторым способом. Второй способ — это отправка TerminateSequence-сообщения. В этом случае агент RM Destination очищает состояние последовательности и никакие дальнейшие запросы в данной последовательности невозможны.

Если агент RM Source посылает запрос на завершение последовательности, с элементом `<wsrm:CloseSequence>` или элементом `<wsrm:TerminateSequence>` в теле сообщения, тогда агент RM Destination в ответ посылает сообщение, содержащее заголовок `<wsrm:SequenceAcknowledgement>` с элементом `<wsrm:Final/>` и элемент `<wsrm:CloseSequenceResponse>` или элемент `<wsrm:TerminateSequenceResponse>` в теле сообщения.

Если же завершение последовательности инициирует агент RM Destination, тогда он посылает сообщение с заголовком `<wsrm:SequenceAcknowledgement>`, включающим элемент `<wsrm:Final/>`, и элементом `<wsrm:CloseSequence>` или элементом `<wsrm:TerminateSequence>` в теле сообщения.

Элементы `<wsrm:CloseSequence>` и `<wsrm:TerminateSequence>` имеют дочерние элементы:

- обязательный элемент `<wsrm:Identifier>` — идентификатор последовательности;
- дополнительный элемент `<wsrm:LastMsgNumber>` — номер последнего сообщения, посланного агенту RM Destination.

Элементы `<wsrm:CloseSequenceResponse>` и `<wsrm:TerminateSequenceResponse>` имеют дочерний элемент `<wsrm:Identifier>`, указывающий идентификатор завершенной последовательности.

Спецификация WS-ReliableMessaging версии 1.0 не определяет элементы `<wsrm:CloseSequence>`, `<wsrm:CloseSequenceResponse>` и `<wsrm:TerminateSequence Response>`, а в элементе `<wsrm:TerminateSequence>` определяет только дочерний элемент `<wsrm:Identifier>`.

Агент RM Destination уведомляет агента RM Source об успешном получении сообщения с помощью отправки ему сообщения, содержащего заголовок `<wsrm:SequenceAcknowledgement>`, который имеет следующие дочерние элементы:

- обязательный элемент `<wsrm:Identifier>` — идентификатор последовательности;
- дополнительный элемент `<wsrm:AcknowledgementRange>` — номера успешно полученных сообщений с помощью атрибутов `Upper` (наибольший номер успешно полученного сообщения) и `Lower` (наименьший номер успешно полученного сообщения);
- дополнительный элемент `<wsrm:None/>` указывает, что ни одно сообщение не было успешно получено в данной последовательности;
- дополнительный элемент `<wsrm:Final/>` указывает, что агент RM Destination не будет больше принимать сообщения. Данный элемент посылается при завершении последовательности;
- дополнительный элемент `<wsrm:Nack>` — альтернатива элементам `<wsrm:AcknowledgementRange>` и `<wsrm:None/>`, содержит номер не принятого сообщения в последовательности. Агент RM Source должен заново послать данное сообщение агенту RM Destination.

Спецификация WS-ReliableMessaging версии 1.0 не определяет элементы `<wsrm:None/>` и `<wsrm:Final/>`.

Каждое сообщение в последовательности имеет свой номер, указываемый в заголовке `<wsrm:Sequence>` сообщения. Заголовок `<wsrm:Sequence>` имеет следующие дочерние элементы:

- `<wsrm:Identifier>` — идентификатор последовательности;
- `<wsrm:MessageNumber>` — номер сообщения.

Спецификация WS-ReliableMessaging версии 1.0 дополнительно определяет элемент `<wsrm:LastMessage/>` заголовка `<wsrm:Sequence>`, указывающий, что данное сообщение является последним в последовательности.

Агент RM Source может запросить уведомление об успешном получении сообщений с помощью запроса, содержащего заголовок `<wsrm:AckRequested>`, который имеет дочерний элемент `<wsrm:Identifier>` — идентификатор последовательности. В ответ агент RM Destination посылает сообщение с заголовком `<wsrm:SequenceAcknowledgement>`.

Спецификация WS-ReliableMessaging версии 1.0 дополнительно определяет элемент `<wsrm:MessageNumber>` заголовка `<wsrm:AckRequested>`, указывающий наиболь-

ший номер сообщения, посланного агентом RM Source в данной последовательности.

Спецификация WS-ReliableMessaging версий 1.1 и 1.2 определяет значения заголовка `<wsa:Action>` для соответствующих сообщений как:

- `http://docs.oasis-open.org/ws-rx/wsrm/200702/CreateSequence`;
- `http://docs.oasis-open.org/ws-rx/wsrm/200702/SequenceAcknowledgement`;
- `http://docs.oasis-open.org/ws-rx/wsrm/200702/AckRequested`.

Спецификация WS-ReliableMessaging версии 1.0 определяет значения заголовка `<wsa:Action>` для соответствующих сообщений как:

- `http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence`;
- `http://schemas.xmlsoap.org/ws/2005/02/rm/SequenceAcknowledgement`;
- `http://schemas.xmlsoap.org/ws/2005/02/rm/AckRequested`.

Спецификация WS-ReliableMessaging версий 1.0, 1.1 и 1.2 поддерживает спецификацию Web Services Reliable Messaging Policy Assertion (WS-RM Policy) соответственно версий 1.0, 1.1 и 1.2.

WS-ReliableMessaging Policy

Спецификация Web Services Reliable Messaging Policy Assertion (WS-RM Policy) определяет утверждения политики для спецификаций WS-Policy и WS-ReliableMessaging.

Спецификация WS-RM Policy поддерживает пространство имен:

- `http://docs.oasis-open.org/ws-rx/wsrmp/200702` — версии 1.1 и 1.2;
- `http://schemas.xmlsoap.org/ws/2005/02/rm/policy` — версия 1.0.

Спецификация WS-RM Policy версии 1.0 поддерживает спецификацию WS-Policy 2004 г., WS-RM Policy версии 1.1 — WS-Policy версии 1.2 и WS-RM Policy версии 1.2 — WS-Policy версии 1.5.

Утверждение политики, требующее поддержку спецификации WS-ReliableMessaging, представлено элементом `<wsrmp:RMAssertion>`, имеющим следующие атрибуты и дочерние элементы (WS-RM Policy 1.1 и 1.2):

- необязательный атрибут `wsp:Optional` — если `true`, тогда утверждение необязательно к исполнению, по умолчанию — `false`;
- обязательный элемент `<wsp:Policy>` — политика утверждения, описываемая с помощью дочерних элементов:
  - дополнительный элемент `<wsrmp:SequenceSTR/>` требует поддержки дочернего элемента `<wsse:SecurityTokenReference>` элемента `<wsrm:CreateSequence>`;
  - дополнительный элемент `<wsrmp:SequenceTransportSecurity/>` — альтернатива элементу `<wsrmp:SequenceSTR/>`, требует поддержки протокола SSL/TLS для защиты последовательности;

- дополнительный элемент `<wsrmp:DeliveryAssurance>` — требования к агентам RM Source и RM Destination со стороны программных компонентов, которые их включают в себя. Элемент `<wsrmp:DeliveryAssurance>` имеет дочерний элемент `<wsp:Policy>` со следующими дочерними элементами:
  - дополнительный элемент `<wsrmp:ExactlyOnce/>` — каждое сообщение доставляется только один раз. Агент RM Source повторяет передачу сообщения до тех пор, пока не получит подтверждение о доставке сообщения от агента RM Destination, который исключает дублирующие сообщения и должен передать своему приложению только один экземпляр успешно полученного сообщения;
  - дополнительный элемент `<wsrmp:AtLeastOnce/>` — альтернатива элементу `<wsrmp:ExactlyOnce/>` — каждое сообщение доставляется, по меньшей мере, один раз. Агент RM Source повторяет передачу сообщения до тех пор, пока не получит подтверждение о доставке сообщения от агента RM Destination, который не обязан исключать дублирующие сообщения;
  - дополнительный элемент `<wsrmp:AtMostOnce/>` — альтернатива элементам `<wsrmp:ExactlyOnce/>` и `<wsrmp:AtLeastOnce/>` — каждое сообщение доставляется не более одного раза. Агент RM Source не обязан повторять передачу сообщения, а агент RM Destination должен исключать дублирующие сообщения;
  - дополнительный элемент `<wsrmp:InOrder/>` — агент RM Source должен сохранять порядок передачи сообщений агенту RM Destination относительно порядка передачи сообщений от приложения агенту RM Source. Агент RM Destination должен восстанавливать исходный порядок передачи сообщений при доставке их своему приложению.

Элемент `<wsrm:RMAssertion>` спецификации WS-RM Policy версии 1.0 имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `wsp:Optional` — если `true`, тогда утверждение необязательно к исполнению, по умолчанию — `false`;
- дополнительный элемент `<wsrm:InactivityTimeout/>` — период деактивации последовательности, в течение которого не должны передаваться никакие сообщения, использующие атрибут `Milliseconds`;
- дополнительный элемент `<wsrm:BaseRetransmissionInterval/>` указывает, используя атрибут `Milliseconds`, период ожидания агентом RM Source подтверждения о доставке сообщения, по истечении которого агент RM Source заново отправляет сообщение, если подтверждение не было получено;
- дополнительный элемент `<wsrm:ExponentialBackoff/>` указывает, что период ожидания агента RM Source формируется согласно алгоритму Exponential Backoff Algorithm;
- дополнительный элемент `<wsrm:AcknowledgementInterval>` указывает, используя атрибут `Milliseconds`, период, после которого агент RM Destination посылает подтверждение о доставке сообщения.

Спецификация WS-RM Policy версии 1.0 поддерживает спецификацию WS-PolicyAttachment 2004 г., WS-RM Policy версии 1.1 — WS-PolicyAttachment версии 1.2 и WS-RM Policy версии 1.2 — WS-PolicyAttachment версии 1.5.

WS-MakeConnection

Спецификация WS-MakeConnection определяет механизм, позволяющий осуществлять обмен сообщениями между двумя конечными точками, одна из которых не имеет доступного адреса в результате, например, использования межсетевого экрана или прокси-сервера.

Спецификация WS-Addressing определяет анонимный URI-адрес для идентификации неадресуемой конечной точки (`http://www.w3.org/2005/08/addressing/anonymous`). Использование такого URI-адреса подразумевает, что ответное сообщение будет послано по тому же HTTP-каналу, по которому пришел запрос. Если же HTTP-канал запроса по каким-либо причинам закрывается, тогда необходимо установить новое соединение между двумя конечными точками, одна из которых не имеет адреса.

Для создания нового транспортного канала анонимная конечная точка посылает SOAP-сообщение, содержащее заголовок `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-rx/wsmc/200702/MakeConnection` и элемент `<wsmc:MakeConnection>` (пространство имен `http://docs.oasis-open.org/ws-rx/wsmc/200702`) в теле сообщения.

Элемент `<wsmc:MakeConnection>` имеет следующие дочерние элементы:

- `<wsmc:Address>` — URI-идентификатор анонимной конечной точки, который формируется как `http://docs.oasis-open.org/ws-rx/wsmc/200702/anonymous?id=`{*unique-String*}, где параметр `id` — идентификатор анонимной конечной точки для создания транспортного канала;
- `<wsrm:Identifier>` (WS-ReliableMessaging) — идентификатор последовательности надежного обмена сообщениями.

Возвращаемое по созданному каналу сообщение содержит заголовок `<wsmc:MessagePending>`, указывающий с помощью атрибута `pending` (`true`/`false`) есть ли еще сообщения, ожидающие отправки анонимной конечной точки.

Спецификация WS-MakeConnection также определяет утверждение политики, требующее поддержки MakeConnection-протокола, представленное элементом `<wsmc:MCSupported>`.

WS-Coordination

При выполнении определенной задачи компоненты распределенной системы могут вступать между собой в сложную цепочку взаимодействий. Иногда требуется объединение такой цепочки в единое действие как, например, в случае создания

отдельной транзакции, когда объединенное действие должно либо приводить к заданному результату, либо, при возникновении ошибки, изменения, произошедшие в результате каждого взаимодействия в цепочке, должны быть аннулированы и система должна вернуться в исходное состояние. Для выполнения объединенного действия требуется механизм, координирующий участие каждого компонента распределенной системы. Спецификация WS-Coordination как раз и определяет такой механизм координации взаимодействий между компонентами распределенной системы.

Для координации выполнения группой Web-сервисов общей задачи создается специальный сервис Coordinator, состоящий, в свою очередь, из трех сервисов:

- Activation-сервис отвечает за создание контекста объединенного действия. Именно создание такого контекста координации и позволяет объединить отдельные взаимодействия в единое действие компонентов системы. Контекст координации, как правило, содержит идентификатор объединенного действия, время жизни контекста и другую координирующую информацию;
- Registration-сервис отвечает за регистрацию компонентов как участников объединенного действия, что дает им возможность получать координирующие сообщения;
- Protocol-сервис отвечает за отправку координирующих сообщений по определенному координирующему протоколу.

Чтобы начать создание координирующего действия, инициирующий Web-сервис должен обратиться к Activation-сервису, который создаст контекст координации. После получения контекста координации инициирующий Web-сервис передает его остальным участникам объединенного действия, и все Web-сервисы, включая инициирующий Web-сервис, регистрируются как участники путем обращения к Registration-сервису. Теперь они могут получать координирующие сообщения от Coordinator-сервиса для выполнения общей задачи.

Для выполнения объединенных действий распределенная система может состоять из сложной композиции Coordinator-сервисов и Web-сервисов участников, при этом Coordinator-сервисы могут образовывать иерархию с последовательным и параллельным между собой взаимодействием.

Для запроса контекста координации инициирующий Web-сервис посылает Activation-сервису SOAP-сообщение с заголовком `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-tx/wscoor/2006/06/CreateCoordinationContext` (или `http://schemas.xmlsoap.org/ws/2004/10/wscoor/CreateCoordinationContext`) и элементом `<wscoor:CreateCoordinationContext>` (пространство имен `http://docs.oasis-open.org/ws-tx/wscoor/2006/06` или `http://schemas.xmlsoap.org/ws/2004/10/wscoor`), имеющим следующие дочерние элементы:

- дополнительный элемент `<wscoor:Expires>` — запрашиваемое время жизни контекста координации;
- дополнительный элемент `<wscoor:CurrentContext>` — при отсутствии данного элемента Activation-сервис создает контекст для нового независимого действия.

Если же данный элемент присутствует, тогда он содержит идентификатор контекста действия, с которым связан создаваемый контекст нового действия;

- обязательный элемент `<wscoor:CoordinationType>` содержит URI-идентификатор типа требуемой координации, например поддержку отдельной транзакции.

В ответ Activation-сервис посылает SOAP-сообщение с заголовком `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-tx/wscoor/2006/06/CreateCoordinationContextResponse` и элементом `<wscoor:CreateCoordinationContextResponse>`, содержащим дочерний элемент `<wscoor:CoordinationContext>`, который может иметь следующие элементы:

- `<wscoor:Identifier>` — URI-идентификатор контекста координации;
- `<wscoor:CoordinationType>` — URI-идентификатор типа координации;
- `<wscoor:RegistrationService>` — адрес конечной точки сервиса регистрации.

Контекст координации рассылается в SOAP-сообщениях в виде заголовка `<wscoor:CoordinationContext>`, который содержит идентификатор контекста координации, время жизни контекста координации, тип координации, адрес конечной точки сервиса регистрации.

Для регистрации участники объединенного действия посылают Registration-сервису SOAP-сообщение с заголовком `<wsa:Action>` со значением `http://docs.oasis-open.org/ws-tx/wscoor/2006/06/Register` и элементом `<wscoor:Register>`, содержащим следующие элементы:

- `<wscoor:ProtocolIdentifier>` — URI-идентификатор координационного протокола для данного типа координации, определенного в контексте координации;
- `<wscoor:ParticipantProtocolService>` — адрес конечной точки для получения координационных сообщений.

В ответ Registration-сервис посылает SOAP-сообщение с заголовком `<wsa:Action>`, имеющим значение `http://docs.oasis-open.org/ws-tx/wscoor/2006/06/RegisterResponse`, и элементом `<wscoor:RegisterResponse>`, содержащим элемент `<wscoor:CoordinatorProtocolService>` с адресом конечной точки Coordinator-сервиса для взаимодействия по протоколу координации.

WS-AtomicTransaction

Спецификация Web Services Atomic Transaction (WS-AtomicTransaction) обеспечивает поддержку отдельных транзакций для распределенных систем.

Серия последовательных изменений в результате взаимодействий компонентов распределенной системы может быть объединена в одну логическую единицу работы — транзакцию, характеризующуюся следующими свойствами:

- атомность — транзакция либо выполняется целиком и успешно, либо не выполняется совсем и все изменения отменяются с восстановлением системы в первоначальное состояние;

- согласованность — результат транзакции не должен нарушать действительность связанных данных, в противном случае производится откат транзакции;
- изолированность — параллельные транзакции не должны взаимодействовать друг с другом;
- долговечность — результат успешной транзакции должен сохраняться независимо от последующих сбоев.

Основой для спецификации WS-AtomicTransaction является спецификация WS-Coordination.

Спецификация WS-AtomicTransaction определяет для элемента `<wscoor:CoordinationType>` значение `http://docs.oasis-open.org/ws-tx/wsat/2006/06` или `http://schemas.xmlsoap.org/ws/2004/10/wsat`, а для элемента `<wscoor:ProtocolIdentifier>` следующие значения:

- `http://docs.oasis-open.org/ws-tx/wsat/2006/06/Completion` или `http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion` — данный протокол регистрируется инициирующим Web-сервисом и позволяет ему подать Coordinator-сервису сигнал завершения транзакции. В результате транзакция либо подтверждается, либо откатывается, соответственно Coordinator-сервис передает инициирующему Web-сервису уведомление либо об успешном завершении транзакции, либо о ее откате. После получения запроса на завершение транзакции Coordinator-сервис использует протокол двухфазного подтверждения для взаимодействия с остальными участниками транзакции. Сначала идет взаимодействие с участниками по протоколу Volatile2PC, а затем по протоколу Durable2PC;
- `http://docs.oasis-open.org/ws-tx/wsat/2006/06/Volatile2PC` или `http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC` — протокол ненадежного двухфазного подтверждения. С данным протоколом регистрируются участники, управляющие несохраняемыми ресурсами, например кэшами;
- `http://docs.oasis-open.org/ws-tx/wsat/2006/06/Durable2PC` или `http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC` — протокол надежного двухфазного подтверждения. Для данного протокола регистрируются участники, управляющие надежными ресурсами, например базами данных.

Протокол двухфазного подтверждения — это протокол, по которому Coordinator-сервис вначале уведомляет всех участников приготовиться, а затем послать Coordinator-сервису сообщения либо об успешном завершении, либо об ошибке. После получения сообщений от участников транзакции Coordinator-сервис принимает решение либо об ее успешном завершении, либо об откате. Если было получено хотя бы одно сообщение об ошибке, тогда Coordinator-сервис посылает всем участникам транзакции запрос на откат всех изменений.

Таким образом, в протоколе двухфазного подтверждения участники получают от Coordinator-сервиса Prepare-, Rollback- и Commit-сообщения, а Coordinator-сервис получает от участников Prepared- или ReadOnly- (если участник не изменял данные), Aborted- и Committed-сообщения.

Спецификация WS-AtomicTransaction также определяет утверждение политики, требующее поддержки атомных транзакций, представленное элементом `<wsat:ATAssertion>` (пространство имен `xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06"` или `xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"`).

WS-BusinessActivity

Спецификация WS-BusinessActivity определяет механизм координации бизнес-действий, основываясь на спецификации WS-Coordination.

Бизнес-действие представляет собой продолжительное во времени выполнение бизнес-задачи с использованием многочисленных атомных транзакций и большого числа Web-сервисов.

Протокол бизнес-действия, в отличие от атомной транзакции, в случае возникновения на каком-либо этапе ошибки не требует полного отката. Вместо полного отката сервисом координации может запускаться альтернативный сценарий выполнения бизнес-задачи (компенсация). При выполнении бизнес-задачи участники и сервис координации проходят через цепочку состояний. При изменении каждого состояния участники и сервис координации обмениваются уведомлениями, и в отличие от атомной транзакции каждое такое промежуточное состояние должно сохраняться. В отличие от атомной транзакции, если отдельный Web-сервис полностью выполнил свою задачу в общем бизнес-действии, он может прекратить свое участие в бизнес-действии. Кроме того, бизнес-действие, в отличие от атомной транзакции, имеет слабую изоляцию, и промежуточный результат одного бизнес-действия может использоваться другим бизнес-действием.

Спецификация WS-BusinessActivity определяет для элемента `<wscoor:CoordinationType>` следующие значения:

- `http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome` или `http://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome` — тип координации, по которому сервис координации должен управлять всеми участниками для завершения или компенсации;
- `http://docs.oasis-open.org/ws-tx/wsba/2006/06/MixedOutcome` или `http://schemas.xmlsoap.org/ws/2004/10/wsba/MixedOutcome` — тип координации, по которому сервис координации должен управлять всеми участниками, но может и управлять каждым отдельным участником для завершения или компенсации.

Для элемента `<wscoor:ProtocolIdentifier>` спецификация WS-BusinessActivity определяет следующие значения:

- `http://docs.oasis-open.org/ws-tx/wsba/2006/06/ParticipantCompletion` или `http://schemas.xmlsoap.org/ws/2004/10/wsba/ParticipantCompletion` — участники, зарегистрированные для этого протокола, сами определяют свое завершение в бизнес-действии;
- `http://docs.oasis-open.org/ws-tx/wsba/2006/06/CoordinatorCompletion` или `http://schemas.xmlsoap.org/ws/2004/10/wsba/CoordinatorCompletion` — участники, заре-

гистрированные для этого протокола, полностью управляются сервисом координации для выполнения своих задач в общем бизнес-действии.

Оба вышеперечисленных протокола могут использоваться для каждого типа координации.

Спецификация WS-BusinessActivity также определяет утверждения политики, требующие поддержку типов координации AtomicOutcome и MixedOutcome, представленные элементами `<wsba:BAAtomicOutcomeAssertion>` и `<wsba:BAMixedOutcome Assertion>` соответственно (пространство имен `http://docs.oasis-open.org/ws-tx/wsba/2006/06` или `http://schemas.xmlsoap.org/ws/2004/10/wsba`).

ГЛАВА 3

Java Web-сервисы

Платформа Java предлагает ряд программных интерфейсов реализации технологии Web-сервисов. Созданный с их применением набор классов, представляющий Web-сервис, не является самостоятельным приложением для операционной системы, а должен разворачиваться в определенной среде выполнения. Такую среду выполнения предоставляют реализации платформ Java SE и Java EE. Для платформы Java SE это комплект разработки JDK 6, а для платформы Java EE это серверы приложений, такие как GlassFish, Oracle ApplicationServer, JBoss, WebLogic, WebSphere и др. Платформа Java ME позволяет разворачивать только клиента для Web-сервиса. Кроме того, Web-сервисы и клиенты Web-сервисов могут быть созданы и развернуты на платформах Web-сервисов, таких как Apache CXF и Axis2, которые могут работать на платформе Java SE или поверх сервера приложений.

Платформа Java SE 6 реализует поддержку технологий Web-сервисов — XML, SOAP, WSDL и WS-Addressing в таких спецификациях, как Java API for XML Processing (JAXP), Java API for XML Web Services (JAX-WS), Java Architecture for XML Binding (JAXB), SOAP with Attachments API for Java (SAAJ), Java Web Services Metadata.

Платформа Java EE, дополнительно к реализациям технологии Web-сервисов платформой Java SE, обеспечивает поддержку технологий Web-сервисов в таких спецификациях, как Java API for RESTful Web Services (JAX-RS), Implementing Enterprise Web Services, Java API for XML-Based RPC (JAX-RPC), Java API for XML Registries (JAXR), Java API for XML messaging (JAXM).

Также сами серверы приложений платформы Java EE обеспечивают поддержку спецификаций второго уровня технологии Web-сервисов в различных комбинациях, составляющих определенные стеки Web-сервисов, которые формируют и платформы Web-сервисов, имеющие свои реализации спецификаций первого и второго уровня технологии Web-сервисов.

Таким образом, для создания Web-сервисов и клиентов Web-сервисов можно использовать программные интерфейсы платформы Java SE, платформы Java EE, платформ Web-сервисов, а для их развертывания — среду выполнения платформы Java SE, серверы приложений Java EE и среды выполнения платформ Web-сервисов.

JAXM и SAAJ

Спецификация Java API for XML messaging (JAXM) представляет низкоуровневый программный интерфейс для создания, обработки и обмена SOAP-сообщений. Спецификация JAXM основывается на спецификациях SOAP 1.1 и SOAP with Attachments.

Изначально JAXM API версии 1.0 состоял из двух пакетов: `javax.xml.soap` и `javax.xml.messaging`. Следующая версия 1.1 спецификации разделила эти два пакета на две спецификации JAXM — пакет `javax.xml.messaging` и SOAP with Attachments API for Java (SAAJ) — пакет `javax.xml.soap`.

> ***Примечание***
> Спецификация SAAJ версии 1.3 поддерживает оба протокола SOAP 1.1 и SOAP 1.2.

Для создания Web-сервиса, основанного на JAXM и SAAJ, достаточно создать сервлет или EJB-компонент, использующий пакеты `javax.xml.soap` и `javax.xml.messaging`.

Клиент такого Web-сервиса может быть двух типов:

- отдельное приложение платформы Java SE, использующее пакет `javax.xml.soap`;
- приложение платформы Java EE с применением пакетов `javax.xml.soap` и `javax.xml.messaging`, которое может использовать поставщика сообщений Messaging Provider.

Для использования пакета `javax.xml.messaging` необходимо, чтобы сервер приложений включал в себя библиотеку `jaxm-api.jar`. Использование поставщика сообщений Messaging Provider требует наличия пакета `jaxm-runtime.jar` — реализации спецификации JAXM.

Поставщик сообщений Messaging Provider представляет собой промежуточный сервис среды выполнения JAXM, обеспечивающий передачу сообщения от клиентского приложения конечному получателю. Поставщик сообщений позволяет выполнять асинхронную отправку сообщения от клиента конечному получателю через промежуточные адреса, определенные в заголовке SOAP-сообщения. Так как поставщик сообщений способен хранить сообщения, то он может обеспечивать надежную доставку сообщений.

Классы и интерфейсы, составляющие программные интерфейсы JAXM API и SAAJ API, рассмотрены в приложении "JAXM API и SAAJ API" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Пример Web-сервиса и клиента на основе JAXM и SAAJ

Для создания Web-сервиса и его клиента воспользуемся средой разработки Eclipse, для платформы Java EE она доступна по адресу **http://www.eclipse.org/downloads/**.

1. Откроем среду Eclipse и в правом верхнем углу главного окна переключимся на **Java EE perspective**.

2. В меню **File** выберем пункты **New | Dynamic Web Project**. В появившемся диалоговом окне создания проекта введем имя проекта JAXMService.

   На компакт-диске
   Проект JAXMService расположен в папке Примеры\Глава3 компакт-диска.

3. Далее необходимо выбрать среду выполнения для Web-сервиса. Нажмем кнопку **New Runtime** и в появившемся окне щелкнем по ссылке **Download additional server adapters**. После загрузки из сети дополнительных адаптеров выберем и инсталлируем **Oracle GlassFish Server Tool**. Теперь в окне **New Server Runtime Environment** выберем **GlassFish 2.1 Java EE 5**.
4. Для того чтобы использовать сервер приложений GlassFish 2.1, требуется предварительно его установить. Сервер GlassFish 2.1 доступен по адресу **http://www.oracle.com/technetwork/java/javaee/downloads/index.html** в составе Java EE 5 SDK. Кроме этого, должен быть инсталлирован набор JDK 6 (**http://www.oracle.com/technetwork/java/javase/downloads/index.html**) с установленной переменной среды `JAVA_HOME`.
5. После выбора GlassFish 2.1 укажем **Create a new local server**, для того чтобы добавить сервер в среду Eclipse, и нажмем кнопку **Next**. Далее, нажав кнопку **Browse**, определим каталог установленного сервера и нажмем кнопку **Finish**.

   Выбор сервера GlassFish 2.1 обусловлен тем, что данный сервер содержит библиотеку JAXM (файл \imq\lib\jaxm-api.jar).

6. После установки среды выполнения в окне создания проекта дважды нажмем кнопку **Next**, выберем **Generate web.xml deployment descriptor** и нажмем кнопку **Finish**. В результате в левом окне **Project Explorer** появится основа проекта.

   В нижнем окне **Servers** отобразится сервер GlassFish 2.1, управлять которым можно теперь с помощью меню, появляющемся при щелчке правой кнопки мыши на названии сервера.

7. Щелкнем правой кнопкой мыши на названии проекта JAXMService в окне **Project Explorer** и выберем пункты **Run As | Run on Server**. В появившемся диалоговом окне нажмем кнопку **Finish** — в окне редактора появится страничка приветствия "Hello World!".
8. Наполним сгенерированную основу проекта JAXM Web-сервисом, синхронно взаимодействующим с Java-клиентом, который использует для соединения с Web-сервисом SOAPConnection.
9. Щелкнем правой кнопкой мыши на узле **Java Resources: src** и выберем пункты **New | Class**.
10. В появившемся диалоговом окне введем имя пакета `serviceresponse` и имя класса `ServiceResponseReqResp` и нажмем кнопку **Finish**.
11. В окне редактора дополним сгенерированный код класса `serviceresponse.ServiceResponseReqResp` (листинг 3.1).

Листинг 3.1. Код класса `serviceresponse.ServiceResponseReqResp`

```
package serviceresponse;
import javax.xml.soap.*;
import javax.xml.messaging.*;

@SuppressWarnings("serial")
public class ServiceResponseReqResp extends JAXMServlet implements
ReqRespListener
{ public SOAPMessage onMessage(SOAPMessage request) {
    try {
      if(request.getSOAPBody().getTextContent().equals("getData")){
        MessageFactory mf = MessageFactory.newInstance();
        SOAPMessage response = mf.createMessage();
        SOAPPart soapPart = response.getSOAPPart();
        SOAPEnvelope envelope = soapPart.getEnvelope();
        SOAPBody body = envelope.getBody();
        body.addTextNode("Data");
        AttachmentPart attachment = response.createAttachmentPart();
        String Data = "This is data for response";
        attachment.setContent(Data, "text/plain");
        attachment.setContentId("data_for_response");
        response.addAttachmentPart(attachment);
        return(response);
      }
      return(null);
    } catch(Exception e) {
      log(e.getMessage());
      return(null);
    }
  }
}
```

Класс `ServiceResponseReqResp` расширяет класс `JAXMServlet` для работы в Servlet-контейнере и реализует интерфейс `ReqRespListener` для получения запроса и отправки ответа в одной операции `onMessage`.

Метод `onMessage()` сначала проверяет соответствие входящего сообщения ожидаемому запросу, а затем формирует тело ответного SOAP-сообщения и его вложения.

Использование пакета `javax.xml.messaging` требует добавления файла jaxm-api.jar в `classpath` приложения. Для того чтобы добавить библиотеку jaxm-api.jar в `classpath` приложения, щелкнем правой кнопкой мыши на узле проекта **Libraries | EAR Libraries** и выберем пункты **Build Path | Configure Build Path**. В появившемся диалоговом окне нажмем кнопку **Add External JARs** и добавим библиотеку \imq\lib\jaxm-api.jar из каталога сервера GlassFish 2.1.

Для того чтобы определить URI-шаблон Web-сервиса ServiceResponseReqResp, изменим дескриптор web.xml узла **WebContent | WEB-INF** (листинг 3.2).

Листинг 3.2. Дескриптор web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <display-name>JAXMService</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>ServiceResponseReqResp</servlet-name>
    <servlet-class>serviceresponse.ServiceResponseReqResp
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServiceResponseReqResp</servlet-name>
    <url-pattern>/ServiceResponseReqResp</url-pattern>
  </servlet-mapping>
</web-app>
```

Теперь создадим синхронного клиента для Web-сервиса ServiceResponseReqResp.

1. Для этого щелкнем правой кнопкой мыши на узле **Java Resources: src** и выберем пункты **New | Class**.
2. В появившемся диалоговом окне введем имя пакета `servicerequest` и имя класса `ServiceRequestSaaj` и нажмем кнопку **Finish**.
3. В окне редактора дополним сгенерированный код класса `servicerequest.ServiceRequestSaaj` (листинг 3.3).

Листинг 3.3. Код класса `servicerequest.ServiceRequestSaaj`

```
package servicerequest;
import java.net.URL;
import javax.xml.soap.*;

public class ServiceRequestSaaj {
  public static void main(String[] args) {
    try{
      SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
```

```
      SOAPConnection con = scf.createConnection();
      MessageFactory mf = MessageFactory.newInstance();
      SOAPMessage request = mf.createMessage();
      SOAPPart soapPart = request.getSOAPPart();
      SOAPEnvelope envelope = soapPart.getEnvelope();
      SOAPBody body = envelope.getBody();
      body.addTextNode("getData");
      request.saveChanges();
      request.writeTo(System.out);
      System.out.println("");
      URL urlEndpoint =
       new URL("http://localhost:8080/JAXMService/ServiceResponseReqResp");
      SOAPMessage response = con.call(request, urlEndpoint);
      System.out.println(response.getSOAPBody().getTextContent());
      @SuppressWarnings("rawtypes")
      java.util.Iterator it = response.getAttachments();
      while (it.hasNext()) {
        AttachmentPart attachment = (AttachmentPart)it.next();
        System.out.println("Attachment " + attachment.getContentId() +
                 " contains: " + attachment.getContent());
      }
      System.out.println("");
      response.writeTo(System.out);
      con.close();
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Класс `ServiceRequestSaaj` представляет собой отдельное Java-приложение платформы Java SE с точкой входа — методом `main()`.

Метод `main()` использует метод `call()` класса `SOAPConnection` для отправки сообщения Web-сервису ServiceResponseReqResp и получения от него ответного сообщения.

В методе `main()` исходящее сообщение заполняется текстом запроса, который после получения проверяется Web-сервисом ServiceResponseReqResp.

Также метод `main()` извлекает из ответного сообщения текст и вложения и выводит их в консоль. Кроме того, как исходящее, так и ответное сообщения полностью выводятся в консоль с помощью метода `writeTo()` класса `SOAPMessage`.

1. Для развертывания Web-сервиса ServiceResponseReqResp в сервере GlassFish 2.1 щелкнем правой кнопкой мыши на названии проекта JAXMService в окне **Project Explorer** и выберем пункты **Run As | Run on Server**. В появившемся диалоговом окне нажмем кнопку **Finish**.

2. Запустим клиента, щелкнув правой кнопкой мыши на узле **ServiceRequestSaaj.java** проекта и выбрав пункты **Run As | Java Application**. В результате в консоль будет выведено исходящее SOAP-сообщение, текст и вложения ответного сообщения и полностью ответное SOAP-сообщение (рис. 3.1):

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/><SOAP-
ENV:Body>getData</SOAP-ENV:Body></SOAP-ENV:Envelope>
Data
Attachment data_for_response contains: This is data for response

------=_Part_0_5506056.1292989566607
Content-Type: text/xml; charset=utf-8

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/><SOAP-
ENV:Body>Data</SOAP-ENV:Body></SOAP-ENV:Envelope>
------=_Part_0_5506056.1292989566607
Content-Type: text/plain
Content-ID: data_for_response

This is data for response
```

Создадим теперь не отдельное клиентское приложение Java SE, а клиента для развертывания в сервере GlassFish 2.1, который будет асинхронно взаимодействовать с JAXM Web-сервисом.

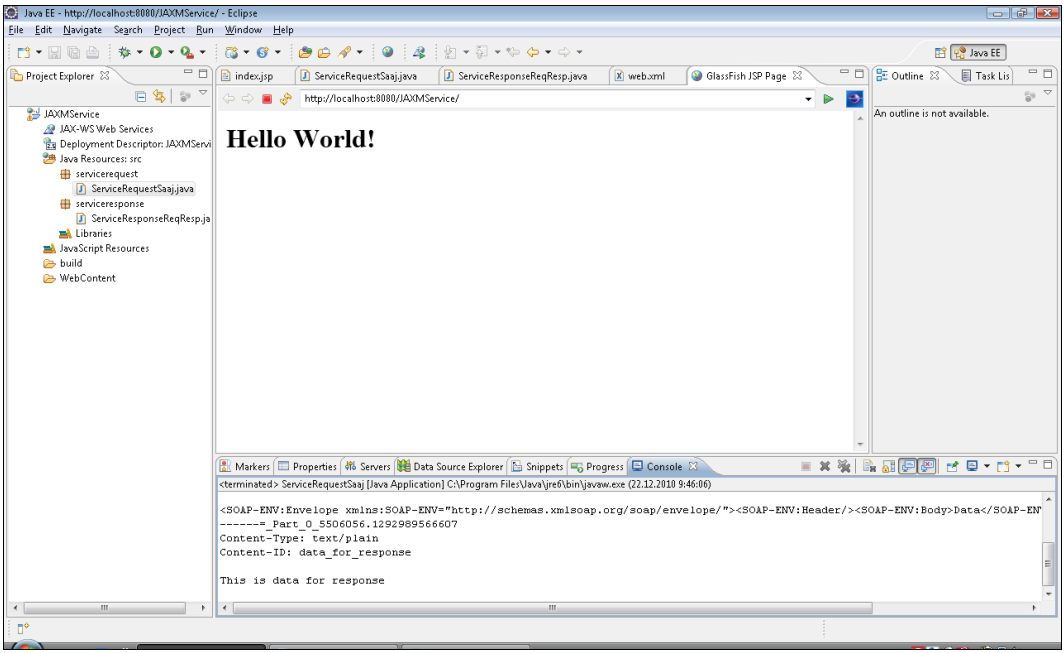


Рис. 3.1. Среда Eclipse после запуска приложения

Сервер GlassFish 2.1 не поддерживает JAXM-поставщика сообщений Messaging Provider, но содержит похожий сервис сообщений Java Message Service (JMS). Поэтому будем использовать программный интерфейс JMS API, являющийся стандартом платформы Java EE, для отправки и получения асинхронных SOAP-сообщений.

Библиотека JMS API и ее реализация содержатся в файлах \imq\lib\jms.jar, \imq\lib\imq.jar и \imq\lib\imqxm.jar каталога сервера GlassFish 2.1. Поэтому с помощью опции **Build Path | Configure Build Path** добавим файлы \imq\lib\jms.jar, \imq\lib\imq.jar, \imq\lib\imqxm.jar в `classpath` приложения.

Сервис сообщений JMS поддерживает две модели отправки-получения сообщений:

- модель point-to-point (PTP), в которой отправитель посылает свое сообщение в очередь (queue), а получатель извлекает сообщение из очереди. При этом очередь queue представлена администрируемым объектом `javax.jms.Queue` сервера, задача которого — хранить сообщения, и каждое сообщение имеет только одного получателя;
- модель publish/subscribe (pub/sub), в которой отправитель публикует свое сообщение в администрируемом объекте `javax.jms.Topic` сервера, сохраняющем сообщения, а многочисленные получатели могут подписаться на рассылку сообщения из объекта `javax.jms.Topic` сервера.

Соединение с JMS-реализацией осуществляется с помощью объекта `javax.jms.Connection` (`QueueConnection` или `TopicConnection`), который получается с применением администрируемого объекта `javax.jms.ConnectionFactory` (`QueueConnectionFactory` или `TopicConnectionFactory`) сервера. Поэтому для начала работы создадим объекты `QueueConnectionFactory`, `TopicConnectionFactory`, `QueueConnection`, `TopicConnection` сервера.

1. Для этого щелкнем правой кнопкой мыши на значке сервера GlassFish 2.1 в окне **Servers** и, выбрав **GlassFish | View Admin Console**, откроем страничку консоли сервера.
2. В узле **Resourses | JMS Resourses** откроем **Connection Factories** и **Destination Resourses** и создадим объект `javax.jms.TopicConnectionFactory` с JNDI-именем JAXMTopicConnectionFactory и объекты `javax.jms.Topic` с JNDI-именами JAXMTopicRequest и JAXMTopicResponse, а также объект `javax.jms.QueueConnectionFactory` с JNDI-именем JAXMQueueConnectionFactory и объекты `javax.jms.Queue` с JNDI-именами JAXMQueueRequest и JAXMQueueResponse.
3. Для создания отправителя сообщений в очередь JAXMQueueRequest щелкнем правой кнопкой мыши на узле **Java Resources: src | servicerequest** и выберем пункты **New | Class**.
4. В появившемся диалоговом окне введем имя класса `ServiceRequestJAXMwithJMSQueue` и нажмем кнопку **Finish**.
5. В окне редактора дополним сгенерированный код класса `servicerequest.ServiceRequestJAXMwithJMSQueue` (см. папку Примеры\Глава3\JAXMService\src\servicerequest компакт-диска).

Класс `ServiceRequestJAXMwithJMSQueue` развертывается в сервере GlassFish 2.1 как сервлет JAXMServlet и реализует интерфейс `javax.jms.MessageListener`, который имеет единственный метод `onMessage()`, вызываемый сервером при получении сообщения.

В методе `init()` инициализации сервлета ServiceRequestJAXMwithJMSQueue вводится объект `javax.jms.QueueConnectionFactory` для создания соединения с сервисом JMX-сервера и вводятся объекты `javax.jms.Queue` для создания объектов `javax.jms.QueueSender` и `javax.jms.QueueReceiver`, отправляющих и получающих соответственно сообщения.

Сообщение запроса создается и отправляется в методе `doGet()`, который вызывается HTTP-запросом GET из страницы браузера. При этом на страницу браузера в ответ выводится SOAP-сообщение запроса и SOAP-сообщение ответа.

В методе `doGet()` SOAP-сообщение запроса конвертируется в JMS-сообщение с помощью метода `MessageTransformer.SOAPMessageIntoJMSMessage()` и затем отправляется методом `send()` интерфейса `javax.jms.QueueSender`.

Ответное сообщение из очереди JAXMQueueResponse получается в методе `onMessage()` и затем конвертируется в SOAP-сообщение ответа методом `MessageTransformer.SOAPMessageFromJMSMessage()`.

1. Для создания получателя сообщений из очереди JAXMQueueRequest щелкнем правой кнопкой мыши на узле **Java Resources: src | serviceresponse** и выберем пункты **New | Class**.
2. В появившемся диалоговом окне введем имя класса `ServiceResponse JAXMwithJMSQueue` и нажмем кнопку **Finish**.
3. В окне редактора дополним сгенерированный код класса `serviceresponse. ServiceResponseJAXMwithJMSQueue` (см. папку Примеры\Глава3\JAXMService\src\ serviceresponse компакт-диска).

Класс `ServiceResponseJAXMwithJMSQueue` развертывается в сервере GlassFish 2.1 как сервлет JAXMServlet и реализует интерфейс `javax.jms.MessageListener`, который имеет единственный метод `onMessage()`, вызываемый сервером при получении сообщения.

В методе `init()` инициализации сервлета ServiceResponseJAXMwithJMSQueue вводится объект `javax.jms.QueueConnectionFactory` для создания соединения с сервисом JMX сервера и вводятся объекты `javax.jms.Queue` для создания объектов `javax.jms.QueueSender` и `javax.jms.QueueReceiver`, отправляющих и получающих сообщения.

Отличие класса `ServiceResponseJAXMwithJMSQueue` от класса `ServiceRequest JAXMwithJMSQueue` в том, что первый упомянутый класс получает сообщения из очереди JAXMQueueRequest, а отправляет в очередь JAXMQueueResponse. Класс же `ServiceRequestJAXMwithJMSQueue`, наоборот, получает сообщения из очереди JAXMQueueResponse, а отправляет в очередь JAXMQueueRequest.

В методе `onMessage()` сообщение запроса получается и конвертируется в SOAP-сообщение, затем на основе SOAP-сообщения запроса формируется SOAP-

сообщение ответа, конвертируется в JMX-сообщение и отправляется в очередь JAXMQueueResponse.

Метод `doGet()` используется для запуска инициализации сервлета ServiceResponseJAXMwithJMSQueue из страницы браузера.

Для создания JAXMServlet-сервлета, публикующего сообщения в объекте `JAXMTopicRequest`:

1. Щелкнем правой кнопкой мыши на узле **Java Resources: src | servicerequest** и выберем пункты **New | Class**.
2. В появившемся диалоговом окне введем имя класса `ServiceRequestJAXMwith JMSTopic` и нажмем кнопку **Finish**.
3. В окне редактора дополним сгенерированный код класса `servicerequest. ServiceRequestJAXMwithJMSTopic` (см. папку Примеры\Глава3\JAXMService\src\ servicerequest компакт-диска).

Класс `ServiceRequestJAXMwithJMSTopic` является JAXMServlet-сервлетом и реализует интерфейс `javax.jms.MessageListener` для автоматического получения сообщений из объекта `JAXMTopicResponse` сервера в методе `onMessage()`, который вызывается сервером при получении сообщения.

В методе `init()` инициализации сервлета ServiceRequestJAXMwithJMSTopic вводится объект `javax.jms.TopicConnectionFactory` для создания соединения с сервисом JMX сервера и вводятся объекты `javax.jms.Topic` для создания объектов `javax.jms.TopicPublisher` и `javax.jms.TopicSubscriber`, отвечающих за публикацию сообщений и подписку на получение сообщений.

Сообщение запроса создается и публикуется в методе `doGet()`, который вызывается HTTP-запросом GET из страницы браузера. При этом на страницу браузера в ответ выводится SOAP-сообщение запроса и SOAP-сообщение ответа.

В методе `doGet()` SOAP-сообщение запроса конвертируется в JMS-сообщение с помощью метода `MessageTransformer.SOAPMessageIntoJMSMessage()` и затем публикуется в объекте `JAXMTopicRequest` методом `publish()` интерфейса `javax.jms.Topic Publisher`.

Ответное сообщение из объекта `JAXMTopicResponse` получается в методе `onMessage()` и затем конвертируется в SOAP-сообщение ответа методом `MessageTransformer. SOAPMessageFromJMSMessage()`.

1. Для создания получателя сообщений из объекта `JAXMTopicRequest` щелкнем правой кнопкой мыши на узле **Java Resources: src | serviceresponse** и выберем пункты **New | Class**.
2. В появившемся диалоговом окне введем имя класса `ServiceResponseJAXMwith JMSTopic` и нажмем кнопку **Finish**.
3. В окне редактора дополним сгенерированный код класса `serviceresponse. ServiceResponseJAXMwithJMSTopic` (см. папку Примеры\Глава3\JAXMService\src\ serviceresponse компакт-диска).

Класс `ServiceResponseJAXMwithJMSTopic` также является JAXMServlet-сервлетом и реализует интерфейс `javax.jms.MessageListener`, но для автоматического получения сообщений из объекта `JAXMTopicRequest`, а не из объекта `JAXMTopicResponse`, как класс `ServiceRequestJAXMwithJMSTopic`.

В методе `init()` инициализации сервлета ServiceResponseJAXMwithJMSTopic вводится объект `javax.jms.TopicConnectionFactory` для создания соединения с сервисом JMX сервера и вводятся объекты `javax.jms.Topic` для создания объектов `javax.jms.TopicPublisher` и `javax.jms.TopicSubscriber`, отвечающих за публикацию сообщений и подписку на получение сообщений.

Метод `onMessage()` отвечает за получение сообщения запроса и его конвертацию в SOAP-сообщение, и затем на основе SOAP-сообщения запроса — за формирование SOAP-сообщения ответа, которое конвертируется в JMX-сообщение и публикуется в объекте `JAXMTopicResponse`.

Метод `doGet()` используется для запуска инициализации сервлета ServiceResponseJAXMwithJMSTopic из страницы браузера.

Для запуска созданных сервлетов отредактируем страницу приветствия index.jsp узла **WebContent** проекта (листинг 3.4).

Листинг 3.4. Код страницы index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>GlassFish JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <a
href="http://localhost:8080/JAXMService/ServiceResponseJAXMwithJMSTopic">Servic
eRequestJAXMwithJMSTopic</a>
    <p><a
href="http://localhost:8080/JAXMService/ServiceResponseJAXMwithJMSQueue">Servic
eRequestJAXMwithJMSQueue</a>
  </body>
</html>
```

Для того чтобы определить URI-шаблоны созданных сервлетов, изменим дескриптор развертывания web.xml узла **WebContent | WEB-INF** (см. папку Примеры\Глава3\JAXMService\WebContent\WEB-INF компакт-диска).

Для развертывания и запуска приложения в сервере GlassFish 2.1 щелкнем правой кнопкой мыши на названии проекта JAXMService в окне **Project Explorer** и выбе-

рем пункты **Run As | Run on Server**. В появившемся диалоговом окне нажмем кнопку **Finish**.

В результате появится страница приветствия с гиперссылками, инициирующими отправление сообщения запроса (рис. 3.2).

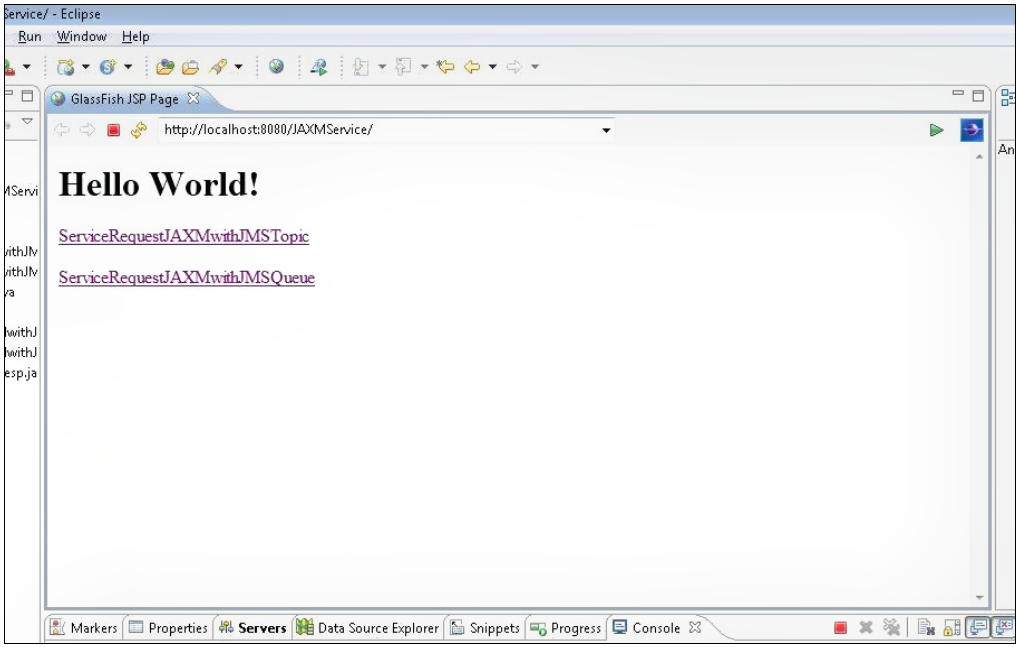


Рис. 3.2. Страничка приветствия приложения

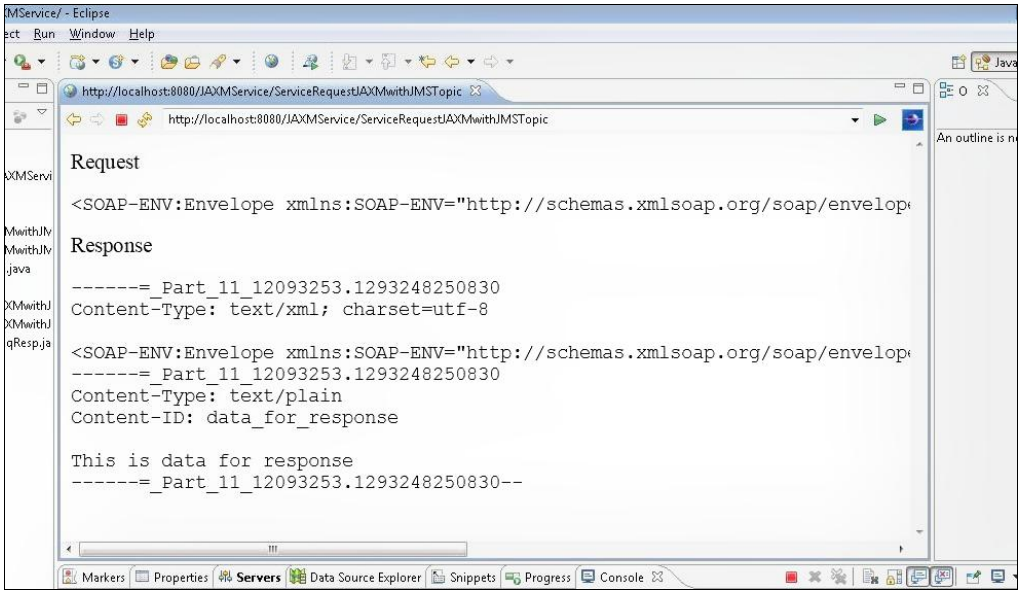


Рис. 3.3. Страничка браузера с сообщениями запроса и ответа

После отправки сообщения запроса в страницу браузера будет выведено сообщение запроса и ответное сообщение (рис. 3.3).

JAXP

Спецификация Java API for XML Processing (JAXP) представляет программный интерфейс для взаимодействия с анализаторами XML-документов, основанными на технологиях SAX (Simple API for XML) и DOM (Document Object Model). Использование JAXP позволяет осуществлять разбор XML-документов, участвующих в обмене с Web-сервисами. Кроме того, JAXP поддерживает технологию Extensible Stylesheet Language Transformations (XSLT), дающую возможность трансформации XML-документов в другие форматы, и реализует стандарт Streaming API for XML (StAX), обеспечивающий механизм, альтернативный SAX и DOM, анализа XML-документов.

Технология SAX-анализа основана на последовательном чтении XML-документа, при этом SAX-анализатор вызывает в использующем его приложении функции-обработчики событий на появление в XML-документе определенных элементов и атрибутов.

Технология DOM-анализа основана на построении в памяти дерева узлов XML-документа, представляющих элементы, атрибуты, текстовые или другие объекты документа, что позволяет осуществлять произвольный доступ к содержимому XML-документа с возможностью изменения его структуры и оформления.

Технология StAX-анализа — это поточная обработка XML-документа как потока событий, где, в отличие от SAX-анализа, приложение само запрашивает события одно за другим вместо предоставления обработчиков событий, которые вызывает SAX-анализатор. Кроме того, в отличие от SAX, программный интерфейс StAX позволяет не только читать XML-документ, но и записывать его.

Программный интерфейс JAXP API состоит из следующих пакетов:

- `javax.xml.parsers` — обработка XML-документов с помощью SAX- или DOM-анализатора;
- `javax.xml.stream`, `javax.xml.stream.events` и `javax.xml.stream.util` — поддержка StAX;
- `org.w3c.dom`, `org.w3c.dom.bootstrap`, `org.w3c.dom.events` и `org.w3c.dom.ls` — поддержка DOM;
- `org.xml.sax`, `org.xml.sax.ext` и `org.xml.sax.helpers` — поддержка SAX;
- `javax.xml.transform`, `javax.xml.transform.sax`, `javax.xml.transform.dom` и `javax.xml.transform.stream` — поддержка XSLT;
- `javax.xml.xpath` — поддержка XPath;
- `javax.xml.validation` — проверка XML-документа с помощью его XML-схемы;
- `javax.xml.datatype` — обеспечение соответствия Java-объектов XML-типам данных, относящихся к временным константам.

Классы и интерфейсы, составляющие программный интерфейс JAXP API, подробно рассматриваются в приложении "JAXP API" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Пример использования JAXP

Для примера использования технологии JAXP понадобится какой-либо XML-файл, требующий анализа. Возьмем реальные XML-данные, которые возвращаются GET-запросом от сервиса Zvents.

Сервис Zvents (**http://www.zvents.com/**) предоставляет информацию о развлечениях различного рода согласно выбранной локализации. Для использования данных программным способом Zvents предлагает интерфейс REST API (**http://corporate.zvents.com/developers/documentation.html**), обеспечивающий набор методов для программного доступа к информации о культурных и спортивных событиях, индексируемых Zvents. Для того чтобы воспользоваться Zvents REST API, необходимо зарегистрироваться на сайте и сгенерировать ключ API Key доступа к сервису Zvents (страница **https://secure.zvents.com/user/edit**).

В качестве примера используем Zvents-метод `/search()` для поиска событий с параметрами. Сформируем следующий GET-запрос:

```
http://www.zvents.com/rest/search?what=dancing&where=New+York%2CNY&when=today&limit=1&key=XEDLOCWFDMUPQBGYRPSQBIRMLAEEMNNNLGVLBYJRGEFYROCTQORCTGVWDHQJOBFB
```

Данный запрос возвращает XML-файл, содержащий информацию об одной вечеринке, которая будет сегодня в Нью-Йорке:

```
<?xml version="1.0" encoding="UTF-8"?>
<rsp status="ok">
<stream_count>21</stream_count>
<event id="125934105">
<name>Radio City Christmas Spectacular</name>
<description>Every performance of the Radio City Christmas Spectacular&#174;
. . .
</description>

<summary>Every performance of the Radio City Christmas Spectacular&#174;
presented by Capital One Bank is a glittering present full of fun and surprises
for the whole family! ...</summary>
<link>http://www.zvents.com/new-york-ny/events/show/125934105-radio-city-
christmas-spectacular</link>
<url>http://www.radiocity.com/events/christmas-spectacular-2010.html</url>
<phone />
<starttime>201012260900</starttime>
<endtime />
<creator_id>126134</creator_id>
<price>$45.00 — $250.00</price>
<age_suitability>All Ages</age_suitability>
```

```
<venue_id>11540</venue_id>
</event>
<venue id="11540">
<name>Radio City Music Hall</name>
<summary>America's most popular entertainers have thrilled audiences at Radio
City Music Hall since its doors opened December 27, 1932. Frank Sinatra, Ella
Fitzgerald, Sammy Davis Jr. ...</summary>
<description>America's most popular entertainers have thrilled audiences at
Radio City Music Hall since its doors opened December 27, 1932.
. . .
</description>
<phone>(212) 247-4777</phone>

<link>http://www.zvents.com/new-york-ny/venues/show/11540-radio-city-music-
hall</link>
<url>http://www.radiocity.com</url>
<creator_id>26945</creator_id>
<address>1260 Avenue of the Americas</address>
<city>New York</city>
<zipcode>10020</zipcode>
<state>NY</state>
<country>United States</country>
<timezone>US/Eastern</timezone>

<latitude>40.75994</latitude>
<longitude>-73.98036</longitude>
</venue>
<search_info>
<identifier>st=event&amp;what=dancing&amp;when=today&amp;where=
New+York%2CNY&amp;ssi=0&amp;ssrss=5&amp;srss=1</identifier>
<sst>1293336000</sst>
<set>1293422399</set>

<srss>1</srss>
<what>dancing</what>
<when>today</when>
<where>New York,NY</where>
<radius>75</radius>
<st>event</st>
<result_size>21</result_size>
<point>
<latitude>40.7561</latitude>

<longitude>-73.987</longitude>
<city>New York</city>
<state>NY</state>
<country>United States</country>
```

```
</point>
</search_info>
</rsp>
```

Для разбора XML-данных, получаемых от сервиса Zvents, создадим JAXP-приложение, использующее SAX, DOM и StAX технологии.

1. Откроем среду разработки Eclipse IDE for Java Developers (**http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/heliossr1**), в меню **File** последовательно выберем пункты **New | Project | Java | Java Project**, нажмем кнопку **Next**, введем имя проекта JAXPExample и нажмем кнопку **Finish**.

   На компакт-диске

   Проект JAXPExample расположен в папке Примеры\Глава3 компакт-диска.

2. В окне **Package Explorer** щелкнем правой кнопкой мыши на узле **src** и выберем пункты **New | Class**. В появившемся диалоговом окне введем имя пакета `jaxpexample` и имя класса `JAXPExample` и нажмем кнопку **Finish**.

В классе `JAXPExample` используем SAX-анализ для фильтрации и вывода информации, содержащейся в элементах `<name>`, `<description>`, `<url>`, `<phone>` и `<address>` XML-файла, возвращаемого сервисом Zvents. Для этого в окне редактора наполним сгенерированную основу класса `jaxpexample.JAXPExample` (листинг 3.5).

Листинг 3.5. Код класса `JAXPExample` с использованием SAX-анализа

```
package jaxpexample;
import java.io.*;
import java.net.*;
import javax.xml.parsers.*;
public class JAXPExample {
  public static void main(String [] args){
    try {
      URL url = new URL
("http://www.zvents.com/rest/search?what=dancing&where=New+York%2CNY&when=today
&limit=1&key=XEDLOCWFDMUPQBGYRPSQBIRMLAEEMNNNLGVLBYJRGEFYROCTQORCTGVWDHQJOBFB");
      InputStream in = url.openStream();
      SAXParserFactory factory = SAXParserFactory.newInstance();
      SAXParser parser = factory.newSAXParser();
      parser.parse(in, new SAXHandler());
      in.close();
    } catch (Exception e) {
      System.out.println(e.getMessage());
}}}
```

В классе `JAXPExample` в качестве первого шага открывается возвращаемый поток HTTP-запроса сервиса Zvents. Затем создается SAX-парсер и производится анализ входящего потока с использованием пользовательского обработчика `SAXHandler`.

Класс `SAXParser` является оберткой парсера `XMLReader`, поэтому экземпляр парсера parser может быть получен альтернативным способом с помощью метода `XMLReaderFactory.createXMLReader()`, при этом SAX-обработчик устанавливается методом `XMLReader.setContentHandler()`.

1. Для создания обработчика `SAXHandler` в окне **Package Explorer** щелкнем правой кнопкой мыши на узле **src | jaxpexample** и выберем пункты **New | Class**. В появившемся диалоговом окне введем имя класса `SAXHandler` и нажмем кнопку **Finish**.
2. В окне редактора дополним сгенерированную основу класса `jaxpexample.SAXHandler` (листинг 3.6).

Листинг 3.6. Код класса `SAXHandler`

```
package jaxpexample;
import org.xml.sax.*;
public class SAXHandler extends org.xml.sax.helpers.DefaultHandler {
  StringBuffer strbf;
  String QName;
  public void startDocument ()
  { strbf = new StringBuffer(); }

  public void endDocument ()
  { System.out.println(strbf); }

  public void characters(char[] ch, int start, int length){
    if(QName.equals("name") | QName.equals("description") |
       QName.equals("url")| QName.equals("phone")|
       QName.equals("address")){
      strbf.append(ch, start, length);
    }
  }

  public void startElement(String uri, String localName, String qName,
                           Attributes attributes){
    QName=qName;
  }
}
```

Класс `SAXHandler` является SAX-обработчиком, т. к. он расширяет класс `org.xml.sax.helpers.DefaultHandler`, переопределяя его методы `startDocument()`, `endDocument()`, `startElement()` и `characters()`, которые вызываются средой выполнения при встрече SAX-парсера с началом XML-документа, его окончанием, началом XML-элемента и его текстовым содержимым.

В методе `characters()` текстовое содержимое накапливается в буферной строке с фильтрацией по XML-элементам `<name>`, `<description>`, `<url>`, `<phone>` и `<address>`, а метод `endDocument()` выводит результирующую строку в консоль.

Щелкнув правой кнопкой мыши на узле **JAXPExample.java** проекта в окне **Package Explorer** и выбрав пункты **Run As | Java Application**, в окне **Console** можно увидеть требуемый текст, извлеченный из XML-данных сервиса Zvents (рис. 3.4).

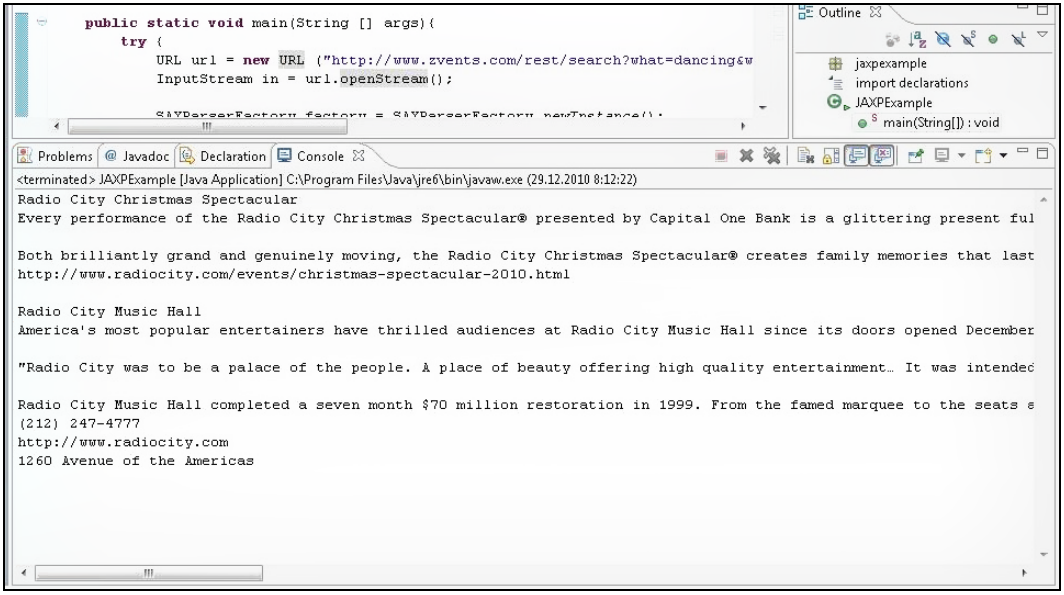


Рис. 3.4. Результат запуска программы JAXPExample

Чтобы применить теперь DOM-анализ для разбора тех же самых XML-данных от сервиса Zvents, дополним класс `JAXPExample` следующим кодом:

```
import org.w3c.dom.*;
. . .
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse(in);
NodeList nodes = doc.getElementsByTagName("name");
for(int i = 0; i < nodes.getLength(); i++) {
  if (nodes.item(i).hasChildNodes()){
      String text=nodes.item(i).getFirstChild().getNodeValue();
      System.out.println(text); }
}
nodes = doc.getElementsByTagName("description");
for(int i = 0; i < nodes.getLength(); i++) {
  if (nodes.item(i).hasChildNodes()){
    String text=nodes.item(i).getFirstChild().getNodeValue();
    System.out.println(text); }
}
```

```
nodes = doc.getElementsByTagName("url");
for(int i = 0; i < nodes.getLength(); i++) {
  if (nodes.item(i).hasChildNodes()){
    String text=nodes.item(i).getFirstChild().getNodeValue();
    System.out.println(text);}
}
nodes = doc.getElementsByTagName("address");
for(int i = 0; i < nodes.getLength(); i++) {
  if (nodes.item(i).hasChildNodes()){
    String text=nodes.item(i).getFirstChild().getNodeValue();
    System.out.println(text);}
}
nodes = doc.getElementsByTagName("phone");
for(int i = 0; i < nodes.getLength(); i++) {
  if (nodes.item(i).hasChildNodes()){
    String text=nodes.item(i).getFirstChild().getNodeValue();
    System.out.println(text);}
}
```

Здесь DOM-парсер инициализируется с помощью метода `DocumentBuilderFactory.newInstance().newDocumentBuilder()`, а дерево XML-документа получается методом `parse()` парсера `DocumentBuilder`.

Далее из дерева документа извлекаются требуемые узлы и выводится на печать содержимое их дочерних текстовых узлов.

Запустив приложение JAXPExample, в окне **Console** можно увидеть текст, извлеченный из XML-данных сервиса Zvents.

Произведем теперь разбор XML-данных сервиса Zvents с помощью StAX-анализа, при этом используем как Cursor API, так и Event Iterator API.

Для этого дополним класс `JAXPExample` следующим кодом:

```
import javax.xml.stream.*;
import javax.xml.stream.events.*;
. . .
//Cursor API
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
XMLStreamReader str = inputFactory.createXMLStreamReader(in);
while(str.hasNext()){
  str.next();
  if(str.isStartElement()){
    String QName=str.getName().toString();
    if(QName.equals("name") | QName.equals("description") |
       QName.equals("url") | QName.equals("phone") |
       QName.equals("address")){
         System.out.println(str.getElementText());
}}}
```

```
str.close();
in.close();
//Event Iterator API
in = url.openStream();
XMLEventReader evr = inputFactory.createXMLEventReader(in);
while (evr.hasNext()) {
  XMLEvent event = evr.nextEvent();
  if (event.isStartElement()) {
    StartElement startElement = event.asStartElement();
    String QName = startElement.getName().toString();
    if(QName.equals("name") | QName.equals("description") |
       QName.equals("url") | QName.equals("phone") |
       QName.equals("address")){
      event = evr.nextEvent();
      StringBuffer st = new StringBuffer();
      while(event.isCharacters()){
        Characters ch = event.asCharacters();
        st.append(ch.getData());
        event = evr.nextEvent();
      }
      System.out.println(st);
}}}
evr.close();
```

Здесь для обработки входящего потока XML-данных создается курсор `XMLStreamReader` и итератор `XMLEventReader`, которые передвигаются по потоку с помощью методов `next()` и `nextEvent()` соответственно. При встрече с нужными XML-элементами их текстовое содержимое извлекается и выводится в консоль.

Запустив приложение JAXPExample, в окне **Console** можно увидеть требуемый текст, извлеченный из XML-данных сервиса Zvents.

JAXB[1]

Спецификация Java Architecture for XML Binding (JAXB) предлагает другой, по сравнению с JAXP, подход к обработке XML-документов. В технологии JAXB основой является XML-схема документа, а сами XML-документы рассматриваются как экземпляры соответствующей XML-схемы. Таким образом, в технологии JAXB из XML-схемы создаются Java-классы, а соответствующие XML-схеме XML-документы представляются в виде наборов Java-объектов созданных Java-классов. Технология JAXB позволяет не только создавать из XML-схемы Java-классы и десериализовать из XML-данных Java-объекты, но и наоборот — создавать из Java-классов XML-схему и сериализовать Java-объекты в XML-данные. Поэтому технология JAXB — это механизм двухстороннего преобразования между XML-представлением и классами языка Java.

[1] Далее рассматривается спецификация JAXB версии 2.2.

Основное отличие технологии JAXB от технологии JAXP состоит в том, что JAXP не требует обязательного наличия схемы XML-документа для его анализа. SAX-анализ разбирает XML-данные как поток и позволяет взять только необходимую часть данных из всего XML-документа. DOM-анализ представляет в памяти XML-документ в виде дерева объектов и позволяет управлять деревом, изменяя его структуру. Кроме того, JAXP дает возможность преобразования формата XML-документа с помощью XSLT API. Дополнительно с помощью JAXP можно проверять соответствие XML-документа его XML-схеме. Однако технология JAXB позволяет создавать объектное представление XML-документа, обходя ограничения памяти DOM-анализа, т. к. JAXB-анализ не предоставляет возможность управления деревом объектов документа. Также объектная модель JAXB разрешает преобразование данных к различным типам и оперирует только верными данными, основываясь на схеме XML-документа. Отличие объектной модели JAXB от DOM-модели заключается в том, что DOM-дерево состоит из объектов-узлов, представляющих элементы, атрибуты, секции `CDATA` и т. д., а JAXB-дерево состоит из объектов, представляющих XML-элементы и содержащих атрибуты и контент как значения своих полей.

Для двухстороннего преобразования "XML-схема ↔ Java-классы" технология JAXB предоставляет два инструмента: xjc и schemagen. Инструмент xjc конвертирует XML-схему типа XML Schema (по умолчанию), RELAX NG, XML DTD и WSDL (компилируются XML-схемы, содержащиеся в WSDL-описании) в Java-классы, маркированные аннотациями пакета `javax.xml.bind.annotation` и обеспечивающие доступ к содержимому компонентов XML-схемы с помощью `get`- и `set`-методов. Инструмент schemagen, наоборот, создает XML-схему типа XML Schema из маркированных аннотациями Java-классов. При генерации Java-классов компилятор xjc использует ряд правил по умолчанию для связи XML-имен с Java-именами. В некоторых случаях необходимо контролировать связывание XML-имен с Java-именами для предотвращения конфликта имен или удовлетворения требований приложения. Для этого существует язык Binding Declaration Language, позволяющий создавать объявления связей (binding declaration) для переопределения правил по умолчанию. Объявления связей могут быть включены непосредственно в XML-схему или же использоваться в виде отдельного файла.

Программный интерфейс JAXB API обеспечивает маршализацию (сериализацию) дерева Java-объектов в XML-документ и демаршализацию (десериализацию) XML-документа в дерево Java-объектов Java-классов, связанных с соответствующей XML-схемой. Процесс маршализации/демаршализации может дополнительно включать в себя проверку на соответствие XML-схеме.

Инструменты xjc и schemagen

Инструменты xjc и schemagen находятся в папке bin каталога JDK. Использование инструментов xjc и schemagen в командной строке выглядит следующим образом:

```
xjc [-options ...] <schema file/URL/dir/jar> ... [-b <bindinfo>] ...
schemagen [-options ...] <java files>
```

Таким образом, инструменты xjc и schemagen позволяют управлять процессом преобразования путем включения дополнительных опций командной строки, описанных в табл. 3.1 и 3.2.

Таблица 3.1. *Опции инструмента xjc (версия 2.2.1)*

Опция	**Описание**
-nv	Компилятор не будет выполнять строгую проверку корректности обрабатываемой XML-схемы
-extension	Компилятор запускается в расширенном режиме, включающем в себя поддержку в объявлениях связей schema component designator (можно указывать в binding declaration компоненты XML-схемы с помощью SCD-выражений), выражений <xjc:superClass>, <xjc:superInterface>, <xjc:javaType>, <xjc:simple/>, <xjc:treatRestrictionLikeNewType/>, <xjc:substitutable>
-b <*file*>	Указывает один или несколько файлов binding declaration
-d <*dir*>	Определяет каталог, в который будут помещены сгенерированные Java-классы
-p <*pkg*>	Переопределяет имя пакета для Java-классов
-httpproxy <*proxy*> -httpproxyfile <*f*>	Определяет HTTP/HTTPS прокси для доступа к ресурсам при компиляции
-classpath <*arg*>	Указывает путь класс-файла, используемого <jxb:javaType> и <xjc:superClass>
-catalog <*file*>	Определяет catalog-файл, связывающий публичные и системные идентификаторы ресурсов с URI-адресами
-readOnly	Компилятор маркирует сгенерированные Java-классы только для чтения
-npa	Подавляет генерацию аннотаций @javax.xml.bind.annotation.XmlSchema в файле package-info.java
-no-header	Подавляет генерацию комментариев в заголовках создаваемых файлов
-target 2.0	Генерируемый код соответствует спецификации JAXB 2.0
-xmlschema	Обрабатывает XML-схему как W3C XML Schema (по умолчанию)
-relaxng	Обрабатывает XML-схему как RELAX NG
-relaxng-compact	Обрабатывает XML-схему как RELAX NG compact syntax
-dtd	Обрабатывает XML-схему как XML DTD
-wsdl	Обрабатывает XML-схему как WSDL
-quiet	Подавляет вывод компилятором информации о процессе компиляции и предупреждения
-verbose	Указывает компилятору выводить дополнительную информацию в процессе компиляции
-help	Вывод документации компилятора
-version	Вывод версии компилятора
<*schema file/URL/dir*>	Определяет XML-схему к обработке

Таблица 3.1 *(окончание)*

Опция	Описание
-Xlocator	Обеспечивает поддержку использования для генерируемых Java-классов объекта org.xml.sax.Locator в процессе демаршализации
-Xsync-methods	Включает ключевое слово synchronized для генерируемых методов
-mark-generated	Включает аннотацию @Generated для генерируемого кода
-episode <*file*>	Компилятор генерирует episode-файл, позволяющий использовать созданные Java-классы при компиляции другой XML-схемы, ссылающейся на данную XML-схему. При компиляции другой XML-схемы episode-файл указывается с помощью опции -b

Таблица 3.2. *Опции инструмента schemagen (версия 2.2.1)*

Опция	Описание
-d <*path*>	Определяет исходящий каталог для компилятора
-cp <*path*> -classpath <*path*>	Указывает расположение связанных Java-классов
-episode <*file*>	Компилятор генерирует episode-файл, обеспечивающий раздельную компиляцию
-version	Выводит версию компилятора
-help	Вывод документации компилятора

Binding Declaration

Как уже было сказано, для переопределения правил по умолчанию для связи XML-имен с Java-именами используются объявления связей — binding declaration (рис. 3.5).

Объявления связей включаются в XML-схему (inline annotated schema) с помощью элемента <xs:appinfo>, являющегося дочерним элементом элемента <xs:annotation> (пространство имен http://www.w3.org/2001/XMLSchema).

Отдельный файл объявлений связей (external binding declaration) содержит корневой элемент <jaxb:bindings> (пространство имен http://java.sun.com/xml/ns/jaxb). Элемент <jaxb:bindings> имеет необязательный атрибут schemaLocation, указывающий URI-адрес XML-схемы, и может иметь дочерний элемент <jaxb:bindings> с необязательным атрибутом node, указывающим узел XML-схемы, для которого применяются объявления связей.

Спецификация JAXB также определяет глобальный атрибут jaxb:version, указывающий версию спецификации (возможные значения — 1.0 и 2.0) и применяемый в элементе <xs:schema> (inline annotated schema) или <jaxb:bindings> (external binding declaration).

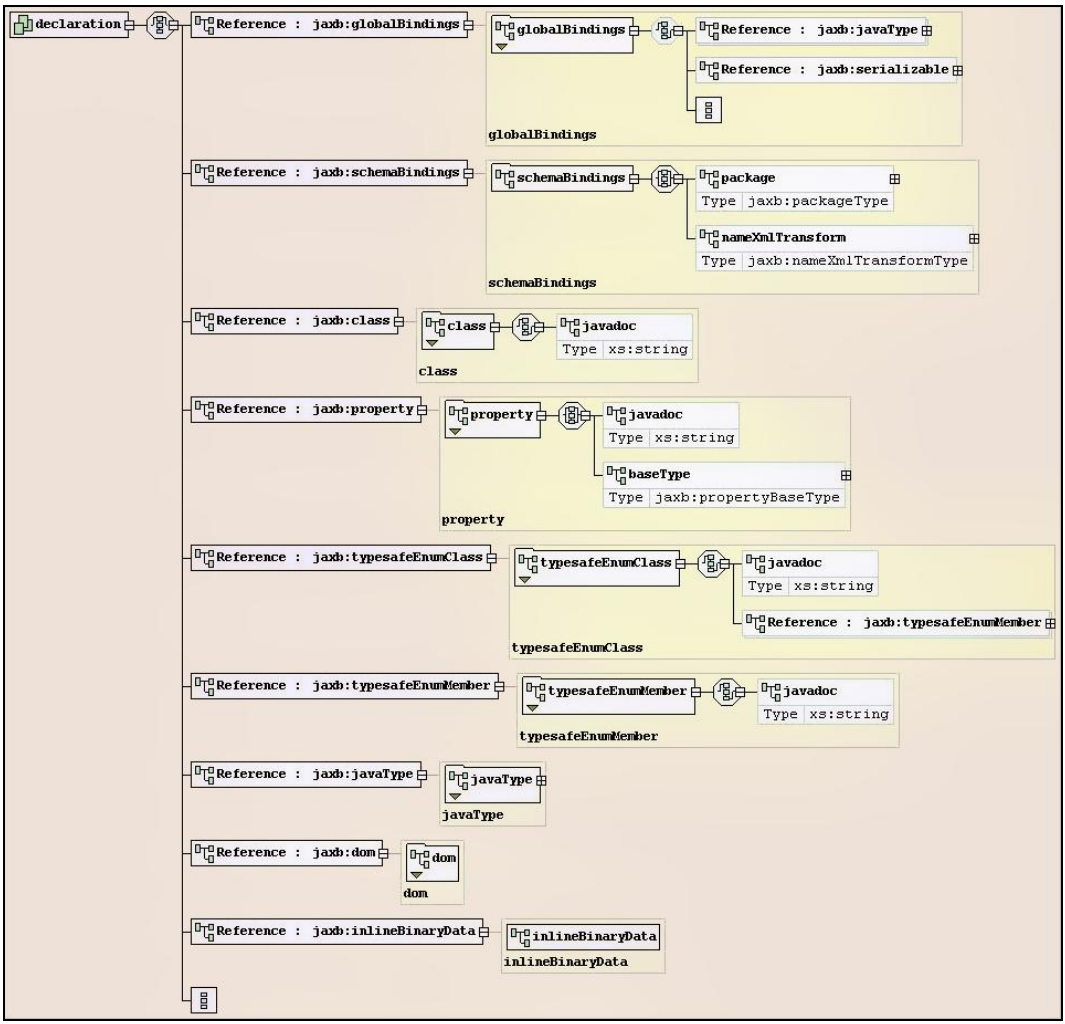


Рис. 3.5. Общая структура JAXB-объявления связей

В случае модификации самой XML-схемы объявлениями связей (inline annotated schema) XML-схема может иметь следующую структуру:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
            jaxb:version="2.0">
  <xsd:annotation>
  <xsd:appinfo>
<jaxb:globalBindings
[ collectionType = "indexed" | "<fully qualified java class name>"]
[ fixedAttributeAsConstantProperty= "true" | "false" | "1" | "0" ]
[ generateIsSetMethod= "true" | "false" | "1" | "0" ]
[ enableFailFastCheck = "true" | "false" | "1" | "0" ]
[ choiceContentProperty = "true" | "false" | "1" | "0" ]
```

```
[ underscoreBinding  = "asWordSeparator" | "asCharInWord" ]
[ typesafeEnumBase = "typesafeEnumBase" ]
[ typesafeEnumMemberName = "skipGeneration" |
                           "generateName" | "generateError" ]
[ typesafeEnumMaxMembers = "xxxx"]
[ enableJavaNamingConventions = "true" | "false" | "1" | "0" ]
[ generateElementClass = "false" | "true" | "0" | "1" ]
[ generateElementProperty = "false" | "true" | "0" | "1" ]
[ generateValueClass = "true" | "true" | "0" | "1" ]
[ optionalProperty = "wrapper" | "primitive" | "isSet" ]
[ mapSimpleTypeDef = "true" | "false" | "1" | "0" ]
[ localScoping = "nested" | "toplevel" ] >
[ <jaxb:javaType> ... </jaxb:javaType> ]*
[ <jaxb:serializable uid="xxxx"/> ]*
</jaxb:globalBindings>
. . .
<jaxb:schemaBindings [ map="boolean" ]>
[<jaxb:package [ name = "packageName"]
[<jaxb:javadoc> ... </jaxb:javadoc>]
</jaxb:package>]
[<jaxb:nameXmlTransform>
  [<jaxb:typeName  [ suffix="suffix" ] [ prefix="prefix" ] />]
  [<jaxb:elementName [ suffix="suffix" ] [ prefix="prefix" ] />]
  [<jaxb:modelGroupName [ suffix="suffix" ] [ prefix="prefix" ] />]
  [<jaxb:anonymousTypeName [ suffix="suffix" ] [ prefix="prefix" ] />]
</jaxb:nameXmlTransform>]*
</jaxb:schemaBindings>
</xsd:appinfo>
</xsd:annotation>
. . .
<xsd:complexType . . .>
<xsd:annotation><xsd:appinfo>
  <jaxb:class[ name = "className" ]
             [ implClass= "implClass" ]>
  [ ref = "className" ]
  [ <jaxb:javadoc> ... </jaxb:javadoc> ]
  </jaxb:class>
</xsd:appinfo></xsd:annotation>
. . .
</xsd:complexType>
. . .
<xsd:simpleType . . .>
<xsd:annotation> <xsd:appinfo>
<jaxb:class . . ./>
</xsd:appinfo></xsd:annotation>
. . .
</xsd:simpleType>
. . .
```

```
<xsd:element . . .>
      <xsd:annotation><xsd:appinfo>
            <jaxb:class . . ./>
         </xsd:appinfo></xsd:annotation>
   </xsd:element>
. . .
<xsd:sequence>
<xsd:element . . .>
  <xsd:annotation><xsd:appinfo>
<jaxb:property[ name = "propertyName" ]
[ collectionType = "propertyCollectionType" ]
[ fixedAttributeAsConstantProperty= "true" | "false" | "1" | "0"
]
[ generateIsSetMethod= "true" | "false" | "1" | "0" ]
[ enableFailFastCheck="true" | "false" | "1" | "0" ]
[ generateElementProperty= "true" | "false" | "1" | "0" ]
[ attachmentRef = "resolve" | "doNotResolve" | "default" ]
[<jaxb:baseType name="fully qualified Java class">
<jaxb:javaType> ... </jaxb:javaType>
</jaxb:baseType>]
[ <jaxb:javadoc> ... </jaxb:javadoc> ]
</jaxb:property>
  </xsd:appinfo></xsd:annotation>
. . .
    </xsd:element>
. . .
</xsd:sequence>
. . .
<xsd:attribute . . .>
  <xsd:annotation><xsd:appinfo>
  <jaxb:property . . ./>
  </xsd:appinfo></xsd:annotation>
. . .
</xsd:attribute>
. . .
<xsd:simpleType . . .>
<xsd:annotation><xsd:appinfo>
<jaxb:javaType> ... </jaxb:javaType>
</xsd:appinfo></xsd:annotation>
. . .
</xsd:simpleType>
. . .
<xsd:simpleType . . .>
<xsd:annotation><xsd:appinfo>
<jaxb:typesafeEnumClass>
[ name = "enumClassName" ]
[ map = "true" | "false" | "1" | "0" ]
```

```
[ ref = "enumClassName" ]
[<jaxb:typesafeEnumMember name = "enumMemberName">
[ value = "enumMemberValue"]
[ <jaxb:javadoc> enumMemberJavadoc </jaxb:javadoc> ]
</jaxb:typesafeEnumMember>]*
[ <jaxb:javadoc> enumClassJavadoc </jaxb:javadoc> ]
</jaxb:typesafeEnumClass> </xsd:appinfo></xsd:annotation>
. . .
</xsd:simpleType>
. . .
</xsd:schema>
```

Элемент `<jaxb:globalBindings>`, аннотирующий элемент `<xsd:schema>`, имеет глобальную область действия и применяется ко всем элементам XML-схемы, включая элементы связанных XML-схем. Элемент `<jaxb:globalBindings>` может иметь следующие атрибуты:

- `collectionType` определяет, что все списки в генерируемых Java-классах должны быть представлены как индексируемые JavaBeans-свойства (массивы свойств или объектов) или как объекты указанного класса, реализующего `java.util.List`;
- `fixedAttributeAsConstantProperty` — если `true`, тогда определяет, что все элементы XML-схемы `<xsd:attribute>` с атрибутом `fixed` должны быть преобразованы в Java-константы (по умолчанию `false`):

  ```
  @XmlAttribute
  public final static type NAME = value;
  ```

- `generateIsSetMethod` — если `true`, тогда определяет генерацию метода `public boolean isSetNameMethod()` (по умолчанию `false`);
- `enableFailFastCheck` — если `true`, тогда при установке некорректного значения свойства объекта сгенерированного Java-класса методом `set()` будет создаваться исключение, не дожидаясь отдельной проверки соответствия относительно XML-схемы (по умолчанию `false`);
- `choiceContentProperty` — если `true`, тогда обеспечивает реализацию элемента `<xsd:choice>`, позволяя объектам Java-класса принимать одно из свойств с различными типами данных (по умолчанию `false`);
- `underscoreBinding` — если `asCharInWord`, тогда Java-имена формируются символом "_", если `asWordSeparator` (по умолчанию) — Java-имена формируются заглавными буквами, например `asCharInWord` — `myPrize_one`, `asWordSeparator` — `myPrizeOne`;
- `enableJavaNamingConventions` — если `true` (по умолчанию), тогда Java-имена генерируются согласно соглашению об именах документа Code Conventions for the Java Programming Language;
- `typesafeEnumBase` содержит список QName-имен (по умолчанию `xs:string`), базовых для перечислений `<xsd:enumeration>` элемента `<xsd:simpleType>`, и опре-

деляет, что данные перечисления должны быть преобразованы согласно Java-шаблону Type-Safe Enumerations;

- typesafeEnumMemberName, возможные значения: skipGeneration (по умолчанию) — если конфликт имен не допускает генерацию Java-типа enum, компилятор пропускает генерацию и выдает предупреждение; generateError — в случае конфликта компилятор генерирует ошибку; generateName — в случае конфликта имен компилятор генерирует имена членов перечисления согласно шаблону VALUE_<N>;
- typesafeEnumMaxMembers — максимальное количество членов перечисления (по умолчанию 256);
- generateElementClass — если true, тогда компилятор генерирует для каждого объявления элемента <xsd:element> в XML-схеме Java-класс, расширяющий javax.xml.bind.JAXBElement<T> и метод класса-фабрики, возвращающий экземпляр JAXBElement<T>. По умолчанию значение false, компилятор генерирует только метод класса-фабрики, возвращающий экземпляр JAXBElement<T>;
- generateElementProperty — если true, тогда методы get() и set() Java-класса будут оперировать его свойствами в виде объектов JAXBElement<T>, в случае, если XML-схема использует в объявлении элементов minOccurs="0" и nillable="true";
- generateValueClass — если true (по умолчанию), тогда для каждого объявления комплексного типа <xsd:complexType> в XML-схеме генерируется Java-класс. Если же значение атрибута — false, тогда генерируется интерфейс и класс его реализации;
- optionalProperty устанавливает представление дополнительного ненулевого элемента или атрибута свойством Java-класса с простым базовым типом. Возможные значения: wrapper — базовый тип свойства Java-класса генерируется как класс-обертка для простого типа; primitive — базовый тип свойства Java-класса генерируется как простой тип; isSet — базовый тип свойства Java-класса генерируется как простой тип и дополнительно генерируется метод isSet();
- mapSimpleTypeDef — если true, тогда компилятор генерирует для каждого объявления простого типа <xsd:simpleType> свой Java-класс, по умолчанию — false (простой XSD-тип связывается с простым Java-типом);
- localScoping — если toplevel, тогда генерируются отдельные Java-классы, а не внутренние Java-классы класса, представляющего корневой элемент XML-схемы, как в случае значения по умолчанию — nested.

Дочерний элемент <jaxb:javaType> элемента <jaxb:globalBindings> обеспечивает переопределение связывания простых XML-типов данных с Java-типами данных. Элемент <jaxb:javaType> имеет следующие атрибуты:

- name — имя Java-типа, к которому преобразуется XML-тип;
- xmlType — преобразуемый XML-тип;
- parseMethod — метод Java-класса, вызываемый JAXB-реализацией при демаршализации;

- printMethod — метод Java-класса, вызываемый JAXB-реализацией при маршализации.

Для определения методов parseMethod() и printMethod() может использоваться интерфейс javax.xml.bind.DatatypeConverterInterface и класс javax.xml.bind.DatatypeConverter JAXB API.

Дочерний элемент <jaxb:serializable> элемента <jaxb:globalBindings> обеспечивает реализацию создаваемыми Java-классами интерфейса java.io.Serializable. Атрибут @uid элемента <jaxb:serializable> указывает значение поля serialVersionUID, используемое для контроля версий класса.

Элемент <jaxb:schemaBindings>, аннотирующий элемент <xsd:schema>, имеет область действия XML-схемы и применяется ко всем элементам целевого пространства имен XML-схемы. Элемент <jaxb:schemaBindings> может иметь дочерний элемент <jaxb:package>, определяющий имя Java-пакета, в который будут генерироваться Java-классы, и элементы <jaxb:nameXmlTransform>, определяющие преобразование имен при формировании пакета.

Атрибут map элемента <jaxb:schemaBindings>, в случае значения false, предотвращает генерацию Java-классов для указанного пакета. По умолчанию значение атрибута — true.

Элемент <jaxb:package> имеет атрибут name — указывает имя Java-пакета, и дочерний элемент <jaxb:javadoc> — если присутствует, тогда определяет содержимое генерируемой javadoc-документации для раздела пакета. Содержимое элемента <jaxb:javadoc> должно использовать символ < или секцию CDATA для применения HTML-тегов.

Элемент <jaxb:nameXmlTransform> может иметь дочерние элементы:

- <jaxb:typeName> — атрибуты suffix и prefix определяют суффикс и префикс Java-имен, представляющих XML-элементы <xsd:complexType> или <xsd:simpleType> (в случае генерации Java-класса);
- <jaxb:elementName> — атрибуты suffix и prefix определяют суффикс и префикс Java-имен, представляющих XML-элементы <xsd:element>;
- <jaxb:modelGroupName> — атрибуты suffix и prefix определяют суффикс и префикс Java-имен, представляющих XML-элементы <xsd:all>, <xsd:choice> или <xsd:sequence>;
- <jaxb:anonymousTypeName> — по умолчанию для XML-объявления анонимного типа данных элемента <xsd:element> генерируется Java-интерфейс с именем элемента плюс суффикс Type, атрибуты suffix и prefix элемента <jaxb:anonymousTypeName> определяют суффикс и префикс Java-имен, представляющих XML-объявления анонимного типа данных элементов <xsd:element>.

Элемент <jaxb:class>, аннотирующий элементы <xsd:complexType>, <xsd:simpleType> и <xsd:element>, имеет область действия компонентов XML-схемы и используется для переопределения связывания компонентов XML-схемы с Java-классами, представляющими элемент и его тип данных, а также с Java-

интерфейсами и классами их реализации. Элемент `<jaxb:class>` может иметь дочерний элемент `<jaxb:javadoc>` и следующие атрибуты:

- `name` — Java-имя для генерации;
- `implClass` — имя Java-класса реализации для `name`;
- `ref` — имя внешнего Java-класса, и компилятор не генерирует новый класс.

Элемент `<jaxb:property>`, аннотирующий элементы `<xsd:attribute>`, `<xsd:choice>`, `<xsd:all>` или `<xsd:sequence>`, имеет область действия компонентов XML-схемы и используется для переопределения связывания компонентов XML-схемы с Java-свойствами. Элемент `<jaxb:property>` может иметь дочерние элементы `<jaxb:javadoc>` и `<jaxb:baseType>`, который переопределяет базовый тип Java-свойства и имеет атрибут `name` (имя базового Java-типа по умолчанию для данного свойства) и дочерний элемент `<jaxb:javaType>`, а также следующие атрибуты:

- `name` — генерируемое имя Java-свойства;
- `collectionType` — Java-тип группы свойств, возможные значения — `indexed` или имя Java-класса, реализующего `java.util.List`;
- `fixedAttributeAsConstantProperty`, `generateIsSetMethod`, `enableFailFastCheck`, `generateElementProperty` — см. элемент `<jaxb:globalBindings>`;
- `attachmentRef` — возможные значения: `resolve` — если XML-тип `xsd:anyURI` или его производный, тогда Java-свойство представляет ссылку на вложение XML-документа и базовый тип Java-свойства генерируется как `javax.activation.DataHandler`, а само Java-свойство аннотируется `@XmlAttachmentRef`; `doNotResolve` — если XML-тип `ref:swaRef`, тогда базовый тип Java-свойства генерируется как `String`, а аннотация `@XmlAttachmentRef` не генерируется; `default` — если XML-тип `ref:swaRef`, тогда Java-свойство представляет ссылку на вложение XML-документа и базовый тип Java-свойства генерируется как `javax.activation.DataHandler`, а само Java-свойство аннотируется `@XmlAttachmentRef`.

Элемент `<jaxb:typesafeEnumClass>`, аннотирующий элементы `<xsd:simpleType>`, имеет область действия компонентов XML-схемы и используется для переопределения связывания компонентов XML-схемы с Java-перечислениями. Элемент `<jaxb:typesafeEnumClass>` может иметь дочерние элементы `<jaxb:javadoc>` и `<jaxb:typesafeEnumMember>`, которые переопределяют связывание элементов `<xsd:enumeration>` с членами Java-перечисления с помощью атрибутов `name` (имя члена перечисления) и `value` (значение члена перечисления). Элемент `<jaxb:typesafeEnumMember>` может непосредственно аннотировать элементы `<xsd:enumeration>`. Элемент `<jaxb:typesafeEnumClass>` также может иметь следующие атрибуты:

- `name` — имя Java-класса перечисления;
- `map` — если `false`, тогда преобразование для данного `<xsd:simpleType>` не производится, по умолчанию `true`;
- `ref` — имя внешнего Java-класса, и компилятор не генерирует новый класс.

Спецификация JAXB также определяет элементы `<jaxb:dom>`, `<jaxb:inline BinaryData>` и `<jaxb:factoryMethod>`.

Элемент `<jaxb:dom>` может аннотировать элементы `<xsd:element>`, `<xsd:complexType>`, `<xsd:simpleType>`, `<xsd:any>`, `<xsd:choice>`, `<xsd:all>`, `<xsd:sequence>`, `<xsd:group>` и определяет связывание XML-компонентов с Java-представлением DOM. Элемент `<jaxb:dom>` может иметь атрибут `type`, указывающий тип DOM, по умолчанию это W3C DOM и преобразование осуществляется к экземпляру `org.w3c.dom.Element`.

Элемент `<jaxb:inlineBinaryData>` может аннотировать элементы `<xsd:element>`, `<xsd:complexType>`, `<xsd:simpleType>` и отключает MTOM/XOP-обработку бинарных данных XML-компонентов, при этом Java свойства или классы аннотируются `@XmlInlineBinaryData`.

Элемент `<jaxb:factoryMethod>` аннотирует объявления элемента или его типа данных и позволяет контролировать имя метода-фабрики класса `ObjectFactory` с помощью атрибута `name`.

Отдельный файл объявлений связей (external binding declaration) имеет следующую структуру:

```
<jaxb:bindings [schemaLocation = "xs:any
  <jaxb:bindings [node = "xs:string"]>*
  <binding declaration>
  <jaxb:bindings>
</jaxb:bindings>
```

Где объявления связей представлены элементами `<jaxb:globalBindings>`, `<jaxb:schemaBindings>`, `<jaxb:class>`, `<jaxb:property>`, `<jaxb:javaType>`, `<jaxb:typesafeEnumClass>`, `<jaxb:dom/>`, `<jaxb:inlineBinaryData>` и `<jaxb:factoryMethod>`.

JAXB API

Программный интерфейс JAXB API состоит из пакетов:

- `javax.xml.bind` обеспечивает платформу для демаршализации, маршализации и проверки XML-документов в Java-приложениях;
- `javax.xml.bind.annotation` и `javax.xml.bind.annotation.adapters` определяют аннотации для компонентов Java-приложений;
- `javax.xml.bind.attachment` обеспечивает маршализацию-демаршализацию XML-документов с вложениями бинарных данных;
- `javax.xml.bind.helpers` предназначен для разработчиков реализаций JAXB;
- `javax.xml.bind.util` позволяет комбинировать JAXB с другими Java/XML-технологиями.

Ключевым классом пакета `javax.xml.bind` является класс `JAXBContext`, обеспечивающий точку входа Java-приложения в JAXB API, предоставляя методы для маршализации, демаршализации и проверки XML-документов.

Приложение создает экземпляры класса `JAXBContext` с помощью статического метода `newInstance()`, который имеет две формы. Первый вид метода `newInstance()` получает в качестве аргумента список имен Java-пакетов, содержащих Java-классы, полученных из XML-схем, и/или полные имена созданных класс-файлов, маркированных JAXB-аннотациями. Каждый пакет Java-классов, полученных из XML-схемы, сопровождается файлом jaxb.properties, который указывает значение свойства `javax.xml.bind.context.factory` — имя класса JAXB-реализации, отвечающего за создание объектов `JAXBContext`, и класс-файлом ObjectFactory.class, содержащим методы создания объектов Java-классов, связанных с XML-представлением. В случае созданных Java-классов, маркированных JAXB-аннотациями, можно не перечислять в методе `newInstance()` класс-файлы, а включить в Java-пакет файл jaxb.index, указывающий неполные имена класс-файлов. Кроме того, Java-пакет может содержать файл package-info.java, обеспечивающий доступ к аннотациям уровня пакета во время исполнения (runtime). Второй вид метода `newInstance()` получает в качестве аргумента список класс-файлов, связанных с XML-представлением.

Классы и интерфейсы, составляющие программный интерфейс JAXB API, подробно рассматриваются в приложении "JAXB API" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Пример использования JAXB

В качестве примера использования технологии JAXB рассмотрим создание XML-данных сервиса Zvents из XML-схемы.

XML-схема для XML-данных, возвращаемых методом `/search` интерфейса Zvents API (**http://corporate.zvents.com/developers/documentation.html**), будет выглядеть так, как показано в листинге 3.7.

Листинг 3.7. XML-схема данных сервиса Zvents

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
  <xs:element name="rsp">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="stream_count"/>
        <xs:element ref="event"/>
        <xs:element ref="venue"/>
        <xs:element ref="search_info"/>
      </xs:sequence>
      <xs:attribute name="status" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="stream_count" type="xs:integer"/>
```

```
<xs:element name="event">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="description"/>
      <xs:element ref="summary"/>
      <xs:element ref="link"/>
      <xs:element ref="url"/>
      <xs:element ref="phone"/>
      <xs:element ref="starttime"/>
      <xs:element ref="endtime"/>
      <xs:element ref="creator_id"/>
      <xs:element ref="price"/>
      <xs:element ref="age_suitability"/>
      <xs:element ref="venue_id"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
<xs:element name="starttime" type="xs:integer"/>
<xs:element name="endtime">
<xs:complexType/>
</xs:element>
<xs:element name="price" type="xs:string"/>
<xs:element name="age_suitability" type="xs:string"/>
<xs:element name="venue_id" type="xs:integer"/>
<xs:element name="venue">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="summary"/>
      <xs:element ref="description"/>
      <xs:element ref="phone"/>
      <xs:element ref="link"/>
      <xs:element ref="url"/>
      <xs:element ref="creator_id"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="zipcode"/>
      <xs:element ref="state"/>
      <xs:element ref="country"/>
      <xs:element ref="timezone"/>
      <xs:element ref="latitude"/>
      <xs:element ref="longitude"/>
    </xs:sequence>
    <xs:attribute name="id" use="required" type="xs:integer"/>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="address" type="xs:string"/>
<xs:element name="zipcode" type="xs:integer"/>
<xs:element name="timezone" type="xs:string"/>
<xs:element name="search_info">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="identifier"/>
      <xs:element ref="sst"/>
      <xs:element ref="set"/>
      <xs:element ref="srss"/>
      <xs:element ref="what"/>
      <xs:element ref="when"/>
      <xs:element ref="where"/>
      <xs:element ref="radius"/>
      <xs:element ref="st"/>
      <xs:element ref="result_size"/>
      <xs:element ref="point"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="identifier" type="xs:string"/>
<xs:element name="sst" type="xs:integer"/>
<xs:element name="set" type="xs:integer"/>
<xs:element name="srss" type="xs:integer"/>
<xs:element name="what" type="xs:NCName"/>
<xs:element name="when" type="xs:NCName"/>
<xs:element name="where" type="xs:string"/>
<xs:element name="radius" type="xs:integer"/>
<xs:element name="st" type="xs:NCName"/>
<xs:element name="result_size" type="xs:integer"/>
<xs:element name="point">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="latitude"/>
      <xs:element ref="longitude"/>
      <xs:element ref="city"/>
      <xs:element ref="state"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="summary" type="xs:string"/>
<xs:element name="link" type="xs:anyURI"/>
<xs:element name="url" type="xs:anyURI"/>
<xs:element name="phone" type="xs:string"/>
```

```
  <xs:element name="creator_id" type="xs:integer"/>
  <xs:element name="city" type="xs:string"/>
  <xs:element name="state" type="xs:NCName"/>
  <xs:element name="country" type="xs:string"/>
  <xs:element name="latitude" type="xs:decimal"/>
  <xs:element name="longitude" type="xs:decimal"/>
</xs:schema>
```

Для создания JAXB-классов из XML-схемы воспользуемся средой разработки NetBeans, доступной для скачивания по адресу **http://netbeans.org/index_ru.html**.

1. Откроем среду NetBeans и в меню **Файл** последовательно выберем пункты **Создать проект | Java | Приложение Java**, нажмем кнопку **Далее**, введем имя проекта JAXBExample и нажмем кнопку **Завершить**.

   На компакт-диске

   Проект JAXBExample расположен в папке Примеры\Глава3 компакт-диска.

2. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAXBExample** и выберем пункты **Создать | Другие | XML | Связывание JAXB**, нажмем кнопку **Далее**, введем имя связывания jaxbexample (**Binding Name**). Нажав кнопку **Обзор**, укажем XML-схему (**Schema File**), введем имя пакета для генерируемых JAXB-классов `jaxbexample` (**Имя пакета**) и нажмем кнопку **Завершить**.

   В результате среда NetBeans сгенерирует для указанной XML-схемы Java-классы, представляющие данные сервиса Zvents (узел **Созданные исходные файлы (jaxb)** проекта JAXBExample).

3. Откроем в узле **Пакеты исходных файлов | jaxbexample** файл Main.java и в окне редактора дополним его код (см. папку Примеры\Глава3\JAXBExample\src\jaxbexample компакт-диска).

В методе `main()` класса `Main` (точке входа в приложение) создаются Java-объекты `Event` и `Venue`, которые затем заполняются данными. После этого Java-объекты маршализуются в XML-данные и выводятся в консоль методом `void marshal(Object jaxbElement, OutputStream os)` интерфейса `javax.xml.bind.Marshaller`.

Для запуска приложения щелкнем правой кнопкой мыши на узле **JAXBExample** и выберем пункт **Выполнить**. В результате в окне **Вывод** среды NetBeans появятся XML-данные сервиса Zvents.

JAX-RPC

Спецификация Java API for XML-based RPC JAX-RPC 1.1 обеспечивает создание XML Web-сервисов и их клиентов, использующих технологии удаленного вызова процедур (Remote Procedure Calls, RPC) и XML. Технология JAX-RPC является частью платформы Java EE и реализует технологию Web-сервисов, обеспечивая совместимость различных платформ и языков программирования.

Платформа Java SE также позволяет создавать распределенные системы с применением пакетов `java.net` и RMI API (`java.rmi`, `java.rmi.activation`, `java.rmi.dgc`,

java.rmi.registry, java.rmi.server). Пакет java.net обеспечивает обмен данными по сети между клиентом и сервером с использованием протокола TCP или UDP, а программный интерфейс RMI API — вызов методов удаленных Java-объектов. Однако такие распределенные системы работают только на платформе Java и не совместимы с компонентами других платформ. Технология же JAX-RPC дает возможность клиенту JAX-RPC взаимодействовать с Web-сервисом другой платформы, а JAX-RPC Web-сервису отвечать на запросы клиента, созданного на основе другой платформы. На базе технологии CORBA, поддерживаемой платформой Java SE, также возможно создание распределенных систем программных компонентов, написанных на разных языках программирования и работающих на различных платформах, но технология CORBA, в отличие от XML Web-сервисов, не использует основанный на XML протокол передачи данных, а ее реализация достаточно сложна по сравнению с JAX-RPC. Кроме того, технология CORBA ориентирована на локальные сети, а не на Интернет, как технология Web-сервисов.

JAX-RPC Web-сервис описывается с использованием языка WSDL, а взаимодействие с ним осуществляется по SOAP/HTTP-протоколу.

Спецификация JAX-RPC 1.1 основывается на спецификациях WSDL 1.1, SOAP 1.1 with attachments и HTTP 1.1.

JAX-RPC Web-сервис может создаваться для развертывания в Servlet-контейнере платформы Java EE или для развертывания в EJB-контейнере сервера приложений платформы Java EE. Далее рассмотрим базирующийся на модели Servlet JAX-RPC Web-сервис.

Создание JAX-RPC Web-сервиса может быть основано на двух различных подходах.

Первый способ — это написание WSDL-документа для будущего Web-сервиса. При наличии WSDL-документа можно воспользоваться JAX-RPC инструментом wscompile, чтобы сгенерировать интерфейс сервиса (service endpoint interface, SEI) и шаблон класса, реализующего этот интерфейс (service implementation class), из элемента <portType> WSDL-определения:

```
wscompile -gen:server -import <config.xml>
```

где config.xml — конфигурационный файл, указывающий расположение WSDL-документа и параметры связывания WSDL/XML и Java Mapping.

Далее следует отредактировать шаблон класса реализации SEI-интерфейса, и мы будем иметь основу для генерации необходимых искусственных объектов и развертывания Web-сервиса.

Второй способ — это кодирование SEI-интерфейса и класса его реализации, а уже потом генерация из них WSDL-документа и других искусственных объектов. При этом SEI-интерфейс должен расширять интерфейс java.rmi.Remote, а его методы должны использовать (throw) исключение java.rmi.RemoteException и иметь параметры и возвращаемые значения поддерживаемых Java-типов. SEI-интерфейс не должен иметь объявления типа public final static. Также класс реализации SEI-интерфейса должен иметь публичный конструктор по умолчанию и может допол-

нительно реализовывать интерфейс `javax.xml.rpc.server.ServiceLifecycle` для управления жизненным циклом конечной точки Web-сервиса. Вспомогательные классы, представляющие передаваемые данные, должны иметь публичные конструкторы по умолчанию, не должны реализовывать интерфейс `java.rmi.Remote`, но должны реализовывать интерфейс `java.io.Serializable`.

После создания или генерации SEI-интерфейса и класса его реализации можно воспользоваться инструментом wscompile (при этом нужно написать конфигурационный файл config.xml) для генерации необходимых для работы Web-сервиса искусственных объектов, например:

```
wscompile -gen:both -d [...] -classpath [...] config.xml
```

где файл config.xml описывает Web-сервис, а инструмент wscompile генерирует искусственные объекты как для сервера, так и для клиента Web-сервиса.

Сгенерированные искусственные объекты на стороне сервера состоят из:

- классов-заглушек Tie;
- WSDL-документа;
- классов, отвечающих за связь Java-представления с XML-представлением.

Сгенерированные искусственные объекты на стороне JAX-RPC-клиента состоят из:

- классов-заглушек Stub, реализующих интерфейс `javax.xml.rpc.Stub`;
- Service-интерфейса и класса его реализации, реализующего интерфейс `javax.xml.rpc.Service`;
- классов, отвечающих за связь Java-представления с XML-представлением.

При взаимодействии клиента с Web-сервисом клиентский запрос и ответ от Web-сервиса проходят следующие этапы (рис. 3.6):

1. После получения JAX-RPC клиентским приложением, использующим для этого интерфейс `javax.xml.rpc.Service` и класс его реализации, объекта Stub-класса, служащего посредником между клиентом и JAX-RPC средой выполнения, клиент вызывает метод объекта Stub-класса, представляющий удаленную процедуру.
2. Объект Stub-класса управляет маршализацией клиентского запроса в SOAP-сообщение, используя классы, отвечающие за связь Java-представления с XML-представлением, и посылает SOAP-сообщение по HTTP-протоколу конечной точке Web-сервиса. Классы, отвечающие за связь Java-представления с XML-представлением, включают в себя классы сериализации-десериализации, конвертирующие Java-тип в XML-представление и обратно, и классы RPC-структур, моделирующие вызов удаленных методов в виде структур для их дальнейшего преобразования в XML-представление. Для каждого удаленного метода есть свой класс RPC-структуры запроса и свой класс RPC-структуры ответа.
3. После получения SOAP-сообщения сервер связывает его с соответствующим объектом Tie-класса. Контейнер Servlet при развертывании Web-сервиса обеспечивает для Tie-класса сервлет, который использует Tie-класс, чтобы управлять

демаршализацией клиентского запроса, применяя классы, отвечающие за связь Java-представления с XML-представлением, и вызывать требуемый метод SEI-интерфейса и класса его реализации, результат которого обратно маршализуется с помощью Tie-класса в SOAP-сообщение.

4. Ответное SOAP-сообщение передается по HTTP-протоколу клиентскому приложению, Stub-класс которого демаршализует SOAP-сообщение в Java-результат для клиента.

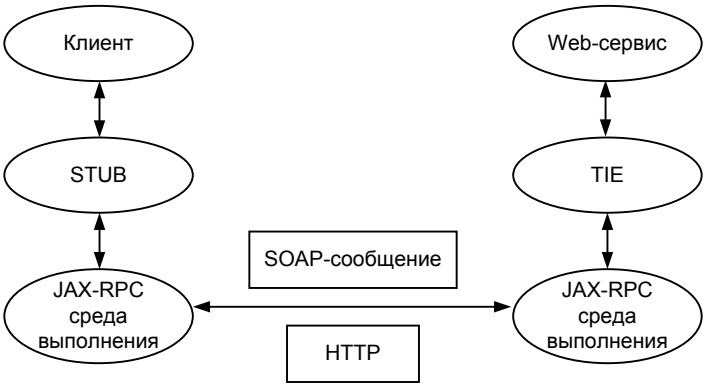


Рис. 3.6. Взаимодействие клиента и Web-сервиса

Таким образом, технология JAX-RPC скрывает от разработчика все взаимодействие по SOAP/HTTP-протоколу — за все отвечает JAX-RPC-реализация. На стороне сервера разработчику достаточно создать интерфейс Web-сервиса, класс его реализации и конфигурационный файл config.xml. Затем сгенерировать артефакты и развернуть Web-сервис. На стороне JAX-RPC-клиента нужно создать Java-приложение, подключить необходимые JAR-файлы и развернуть клиентское приложение.

Клиент JAX-RPC Web-сервиса может быть любым приложением, так же, как и JAX-RPC-клиент, может взаимодействовать с любым Web-сервисом. Далее рассмотрим создание JAX-RPC-клиента.

Существуют три вида JAX-RPC-клиента:

- клиент, использующий статический Stub-объект;
- клиент, использующий динамический объект заглушки (Proxy);
- клиент, использующий динамический вызов Dynamic Invocation Interface (DII) интерфейса `javax.xml.rpc.Call`.

Клиентское приложение, использующее статический Stub-объект, должно иметь предварительно сгенерированные артефакты, включая Stub-класс и Service-класс, которые можно получить с помощью инструмента wscompile из SEI-интерфейса или WSDL-описания Web-сервиса.

Зная имя Stub-класса и имя Service-класса, клиентское приложение с помощью конструктора создает Service-объект и получает Stub-объект с помощью метода

getPort() интерфейса javax.xml.rpc.Service. После этого необходимо методом _setProperty() интерфейса javax.xml.rpc.Stub установить свойство javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY (URI-адрес конечной точки Web-сервиса) для Stub-объекта, что позволит вызвать удаленный метод Stub-объекта.

Статический JAX-RPC-клиент должен иметь все необходимые JAX-RPC артефакты, знать адрес конечной точки, имя вызываемого метода и его параметры.

Клиенту, использующему динамический Proxy-объект, не нужен предварительно сгенерированный Stub-класс, он получает объект заглушки программным образом из WSDL-документа Web-сервиса. Для этого клиентское приложение:

1. Создает объект javax.xml.rpc.ServiceFactory статическим методом newInstance().
2. Создает Service-объект методом createService(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName), зная адрес WSDL-документа и имя Web-сервиса.
3. Получает Proxy-объект методом getPort(javax.xml.namespace.QName portName, java.lang.Class serviceEndpointInterface) интерфейса javax.xml.rpc.Service, зная имя WSDL-элемента <port> и имя SEI-интерфейса. При этом JAX-RPC-реализация использует Dynamic Proxy Class API — класс java.lang.reflect.Proxy.

   Примечание

   WSDL-элемент <port> представляет экземпляр Web-сервиса, созданный средой выполнения.

4. Вызывает удаленный метод Proxy-объекта.

Клиенту, использующему интерфейс javax.xml.rpc.Call, также не нужен предварительно сгенерированный Stub-класс, ему нужно знать имена WSDL-элементов <port> и <service>, адрес конечной точки Web-сервиса, имя и детали удаленного метода, чтобы произвести следующие действия:

1. Создать объект javax.xml.rpc.ServiceFactory статическим методом newInstance.
2. Создать Service-объект методом createService(javax.xml.namespace.QName serviceName), зная имя Web-сервиса.
3. Создать Call-объект методом createCall(javax.xml.namespace.QName portName, java.lang.String operationName) Service-объекта, зная имя порта и имя удаленного метода.
4. Методами Call-объекта setTargetEndpointAddress, setReturnType, setProperty установить адрес конечной точки, тип возвращаемого методом значения, такие свойства, как ENCODINGSTYLE_URI_PROPERTY и OPERATION_STYLE_PROPERTY.
5. Вызвать удаленный метод с помощью метода invoke(java.lang.Object[] inputParams) или invokeOneWay(java.lang.Object[] inputParams) Call-объекта.

Метод invoke() обеспечивает синхронное взаимодействие запроса и ответа, а метод invokeOneWay() — синхронный запрос, который завершается при получении от сервера HTTP-кода 200 (OK) или кода ошибки.

Таким образом, технология JAX-RPC поддерживает только синхронное взаимодействие клиента с Web-сервисом.

Инструменты wscompile и wsdeploy

Существуют два JAX-RPC инструмента — wscompile и wsdeploy, используемых для генерации артефактов, необходимых для работы Web-сервиса и его клиента.

Общий синтаксис инструмента wscompile, который способен генерировать все необходимые артефакты как на стороне клиента, так и на стороне сервера:

```
wscompile [options] <configuration-file>
```

а его опции описаны в табл. 3.3.

Таблица 3.3. *Опции инструмента wscompile*

Опция	Описание
`-classpath <path>` или `-cp <path>`	Указывает расположение класс-файлов Java Web-сервиса при генерации WSDL-документа
`-d <directory>`	Указывает каталог для генерируемых артефактов
`-define`	Генерирует WSDL-документ
`-f:<features>` или `-features:<features>`	Включает дополнительные опции компилятора (см. табл. 3.4)
`-g`	Выводит информацию при компиляции исходных Java-файлов
`-gen` или `-gen:client`	Генерирует искусственные объекты на стороне клиента
`-gen:server`	Генерирует артефакты на стороне сервера
`-httpproxy:<host>:<port>`	Определяет HTTP-прокси при использовании внешних ресурсов — WSDL-документа или XML-схем
`-import`	Генерирует интерфейс Web-сервиса SEI и шаблон класса, реализующего этот интерфейс, из элемента `<portType>` WSDL-определения
`-keep`	Сохраняет исходные сгенерированные Java-файлы после их компиляции. По умолчанию сгенерированные исходные Java-файлы удаляются после компиляции и остаются только класс-файлы
`-mapping <file>`	Генерирует файл JAX-RPC mapping file, являющийся XML-файлом, который определяет связывание между WSDL-документом и Java-интерфейсом Web-сервиса SEI
`-model <file>`	Записывает бинарное представление Web-сервиса, являющееся внутренним для компилятора и используемое в дальнейшем инструментом wsdeploy, в указанный файл
`-nd <directory>`	Определяет каталог для размещения генерируемого WSDL-документа
`-O`	Оптимизирует код при компиляции, удаляя неиспользуемые поля, ненужный код и т. д.
`-s <directory>`	Определяет каталог для размещения генерируемых исходных Java-файлов

Таблица 3.3 *(окончание)*

Опция	Описание
`-verbose`	Выводит информацию о процессе работы wscompile
`-version`	Выводит версию wscompile

Таблица 3.4. *Дополнительные опции* `-f` *инструмента wscompile*

Опция	Описание	Тип входящего файла
`datahandleronly`	Если данная опция отсутствует, тогда JAX-RPC-реализация связывает значения типа `java.awt.Image`, `java.lang.String`, `javax.mail.internet.MimeMultipart` и `javax.xml.transform.Source` с SOAP-вложениями MIME-типа `image/gif` или `image/jpeg`, `text/plain`, `multipart/*` и `text/xml` или `application/xm`. Если же данная опция включена, тогда содержимое SOAP-вложений всегда связывается с Java-типом `javax.activation.DataHandler`	WSDL
`documentliteral`	Тело SOAP-сообщений описывается XML-схемой	Java-интерфейс Web-сервиса SEI
`donotoverride`	Запрещает генерацию классов, уже присутствующих в `classpath`	WSDL, Java-интерфейс Web-сервиса SEI
`donotunwrap`	При использовании кодировки SOAP-сообщений document/literal wrapped (т. е. когда тело SOAP-сообщений описывается XML-схемой, но параметры метода Web-сервиса определяются внутри специальных элементах-обертках) в процессе генерации Java-интерфейса SEI из WSDL-файла элементы-обертки могут либо игнорироваться, либо учитываться. Включение же данной опции запрещает удаление оберток (по умолчанию)	WSDL
`explicitcontext`	При генерации Java-интерфейса SEI из WSDL-файла блоки заголовков SOAP-сообщений, определенные в элементах `<soap:header message="..." part="..."...>` WSDL-элемента `<soap:operation>`, связываются с дополнительными аргументами методов Java-интерфейса Web-сервиса SEI, что дает возможность программной обработки контекста Web-сервиса	WSDL
`infix:<`*name*`>` или `infix=<`*name*`>`	Определяет строку, включаемую в имена генерируемых Java-классов заглушек и сериализаторов	WSDL, Java-интерфейс Web-сервиса SEI
`jaxbenumtype`	По умолчанию анонимное XML-перечисление связывается с Java-перечислением. Данная опция связывает анонимное XML-перечисление с базовыми типами данных	WSDL
`nodatabinding`	Аргументы и возвращаемые значения методов Java-интерфейса Web-сервиса SEI конструируются как объекты `javax.xml.soap.SOAPElement`	WSDL

Таблица 3.4 *(окончание)*

Опция	**Описание**	**Тип входящего файла**
`noencodedtypes`	Атрибут `xsi:type` кодировки SOAP encoding style для массивов и сложных элементов подавляется	WSDL, Java-интерфейс Web-сервиса SEI, файл model
`nomultirefs`	Все типы данных сериализуются в теле SOAP-сообщения как встроенные (inline), а не как использующие ссылки	WSDL, Java-интерфейс Web-сервиса SEI, файл model
`norpcstructures`	Применяется вместе с `-import`, запрещая генерацию Java-классов реализации, используемых средой выполнения JAX-RPC	WSDL
`novalidation`	Выключает проверку WSDL-документа	WSDL
`resolveidref`	По умолчанию JAX-RPC-реализация не поддерживает тип данных `xsd:IDREF`. Данная опция включает поддержку `xsd:IDREF`	WSDL
`rpcliteral`	Тело SOAP-сообщения формируется согласно правилам RPC — содержит элемент с именем вызываемого метода Web-сервиса, который, в свою очередь, содержит элементы для параметров метода. При этом сериализация данных осуществляется согласно XML-схеме	Java-интерфейс Web-сервиса SEI
`searchschema`	При создании артефактов на стороне клиента из WSDL-документа генерируются классы для всех типов XML-схемы, связанной с WSDL-документом	WSDL
`serializeinterfaces`	Включает прямую сериализацию интерфейса	WSDL, Java-интерфейс Web-сервиса SEI, файл model
`strict`	Генерируемый код строго соответствует спецификации JAX-RPC без расширений реализации	WSDL, Java-интерфейс Web-сервиса SEI
`unwrap`	При использовании кодировки SOAP-сообщений document/literal wrapped, в процессе генерации Java-интерфейса SEI из WSDL-документа, элементы-обертки игнорируются	WSDL
`useonewayoperations`	Допускается генерация операций one-way для WSDL-документа	Java-интерфейс Web-сервиса SEI
`wsi`	Включает поддержку спецификации WSI-Basic Profile	WSDL

Конфигурационный файл config.xml содержит информацию о Web-сервисе, используемую инструментом wscompile, и имеет следующую структуру (рис. 3.7):

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
   <service>|<wsdl>|<modelfile>
</configuration>
```

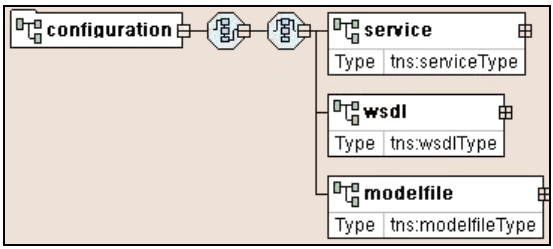


Рис. 3.7. Общая схема файла config.xml

Если конфигурационный файл содержит элемент `<service>`, тогда wscompile генерирует артефакты, исходя из Java-интерфейса Web-сервиса.

Элемент `<service>` (рис. 3.8) имеет следующие атрибуты и дочерние элементы:

- обязательный атрибут `name` — имя Web-сервиса, применяемое при генерации properties-файла, который используется средой выполнения JAX-RPC для передачи запроса классу-заглушке;
- обязательный атрибут `targetNamespace` — целевое пространство имен для генерируемого WSDL-документа;
- обязательный атрибут `typeNamespace` — целевое пространство имен для XML-схемы, связанной с WSDL-документом;
- обязательный атрибут `packageName` — имя Java-пакета Web-сервиса;

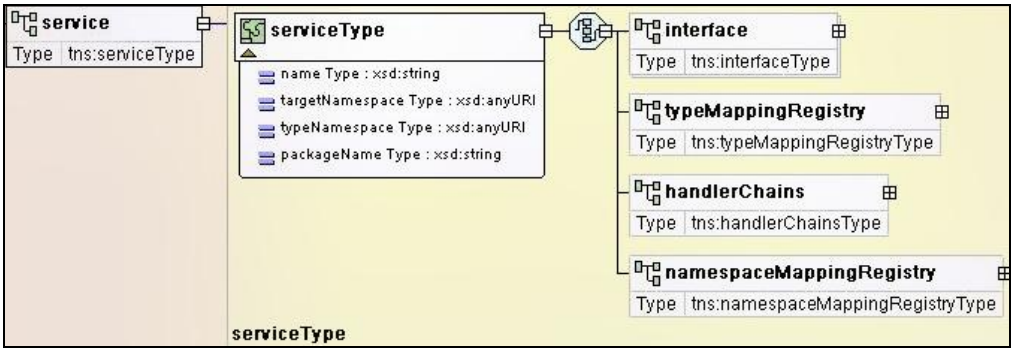


Рис. 3.8. Схема элемента `<service>`

- дополнительные элементы `<interface>` — описание Java-интерфейса Web-сервиса SEI с помощью следующих атрибутов и элементов:
  - обязательный атрибут `name` — имя Java-интерфейса Web-сервиса SEI;
  - дополнительный атрибут `servantName` — имя класса реализации Java-интерфейса Web-сервиса SEI;
  - дополнительный атрибут `soapAction` — значение атрибута `soapAction` WSDL-элементов `<soap:operation>`;

- дополнительный атрибут `soapActionBase` (взаимоисключающий с атрибутом `soapAction`) — базовый URI для значений атрибута `soapAction` WSDL-элементов `<soap:operation>`;
- дополнительный элемент `<handlerChains>` — описание цепочек обработчиков SOAP-сообщений. Обработчики SOAP-сообщений — это классы, реализующие интерфейс `javax.xml.rpc.handler.Handler` и способные один за другим модифицировать SOAP-запросы и ответы как на стороне клиента, так и на стороне сервера. Элемент `<handlerChains>` содержит элементы `<chain>`, имеющие атрибуты `runAt` (обязательный, указывает, где работает обработчик — на стороне клиента или сервера) и `roles` (дополнительный, указывает список SOAP-ролей для цепочки). Элемент `<chain>` также может содержать элементы `<handler>`, определяющие Handler-классы для цепочки. Элемент `<handler>` имеет атрибуты `className` (обязательный, указывает имя класса-обработчика), `headers` (дополнительный, указывает список обрабатываемых SOAP-заголовков) и дополнительные элементы `<property>`, определяющие свойства обработчика (атрибуты `name` и `value`);

- дополнительный элемент `<typeMappingRegistry>` — правила кодировки Java-типов с помощью дочерних элементов `<import>`, `<typeMapping>` и `<additionalTypes>`. Элемент `<import>` указывает список XML-схем, описывающих типы данных для сериализации, используя элементы `<schema>` — атрибуты `namespace` и `location`. Элемент `<typeMapping>` определяет связь типов для определенной кодировки — атрибут `encodingStyle` (URI кодировки) и элементы `<entry>` (атрибуты `schemaType` — схема, `javaType` — Java-класс, `serializerFactory` — класс-фабрика сериализатора, `deserializerFactory` — класс-фабрика десериализатора). Элемент `<additionalTypes>` указывает дополнительные Java-типы, которые должны обрабатываться, — дочерние элементы `<class>` (атрибут `name` — имя класса);
- дополнительный элемент `<handlerChains>` — цепочки обработчиков SOAP-сообщений по умолчанию. Элемент `<handlerChains>` содержит элементы `<chain>`, имеющие атрибуты `runAt` (обязательный, указывает, где работает обработчик — на стороне клиента или сервера) и `roles` (дополнительный, указывает список SOAP-ролей для цепочки). Элемент `<chain>` может также содержать элементы `<handler>`, определяющие `javax.xml.rpc.handler.Handler`-классы для цепочки. Элемент `<handler>` имеет атрибуты `className` (обязательный, указывает имя класса-обработчика), `headers` (дополнительный, указывает список обрабатываемых SOAP-заголовков) и дополнительные элементы `<property>`, определяющие свойства обработчика (атрибуты `name` и `value`);
- дополнительный элемент `<namespaceMappingRegistry>` — связь XML-пространства имен с именем Java-пакета с помощью дочерних элементов `<namespaceMapping>`, имеющих атрибуты `namespace` и `packageName`.

Если конфигурационный файл содержит элемент `<wsdl>`, тогда wscompile генерирует артефакты, исходя из WSDL-документа Web-сервиса. В этом случае артефакты на стороне JAX-RPC клиента включают в себя также SEI-интерфейс.

Элемент <wsdl> (рис. 3.9) имеет следующие атрибуты и дочерние элементы:

- обязательный атрибут location — URL-адрес WSDL-документа;
- обязательный атрибут packageName — имя Java-пакета по умолчанию;
- дополнительный элемент <typeMappingRegistry> — см. элемент <service>;
- дополнительный элемент <handlerChains> — см. элемент <service>;
- дополнительный элемент <namespaceMappingRegistry> — см. элемент <service>.

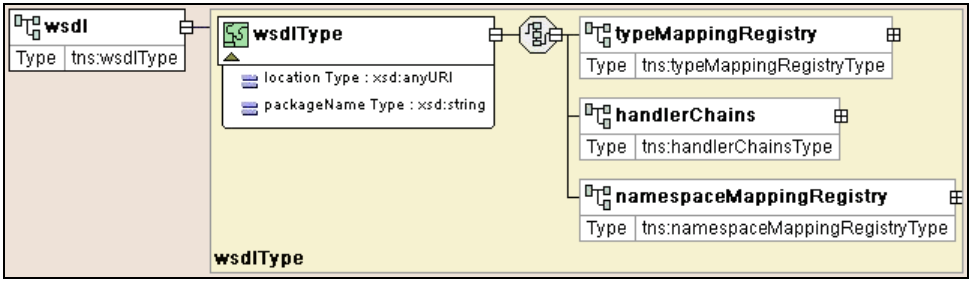


Рис. 3.9. Схема элемента <wsdl>

Если конфигурационный файл содержит элемент <service> или <wsdl>, тогда wscompile генерирует файл model, содержащий внутренние данные, описывающие Web-сервис. Указав элемент <modelfile> в конфигурационном файле, можно запустить wscompile с уже предварительно созданным файлом model. Элемент <modelfile> указывает расположение файла model с помощью атрибута location.

Инструмент wscompile может генерировать файл JAX-RPC mapping file (рис. 3.10), связывающий Java-представление Web-сервиса с его WSDL-описанием. Этот конфигурационный файл используется вместе с WSDL-документом для генерации артефактов при развертывании переносимого WAR-модуля Web-сервиса с помощью инструментов развертывания платформы Java EE и имеет следующую структуру (при развертывании возможно использование вместо mapping-файла model-файла):

```
<java-wsdl-mapping xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://www.ibm.com/webservices/xsd/j2ee_jaxrpc_mapping_1_1.xsd"
    version="1.1">
    ...
</java-wsdl-mapping>
```

Примечание

Переносимый WAR-модуль Web-сервиса — это архивный файл, содержащий Java-интерфейс Web-сервиса, класс его реализации, вспомогательные классы, WSDL-документ, конфигурационные файлы и дескрипторы развертывания, т. е. не содержит файлы, специфичные для конкретной JAX-RPC-реализации.

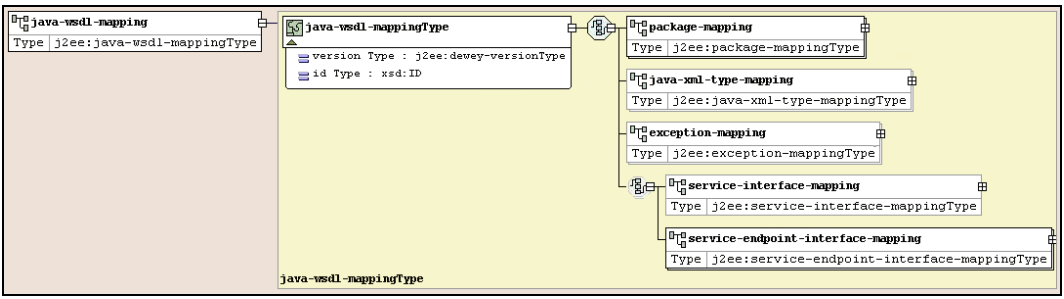


Рис. 3.10. Общая схема mapping-файла

Корневой элемент mapping-файла `<java-wsdl-mapping>` имеет следующие атрибуты и дочерние элементы:

- обязательный атрибут `version` — версия спецификации — `1.1`;
- дополнительный атрибут `id` — идентификатор элемента;
- обязательный элемент `<package-mapping>` — связь между именем Java-пакета и XML-пространством имен в WSDL-документе. Этот элемент имеет необязательный атрибут `id` и обязательные дочерние элементы `<package-type>` (имя Java-пакета) и `<namespaceURI>` (URI-пространства имен);
- дополнительный элемент `<java-xml-type-mapping>` — связь между Java-типами и XML-типами данных. Этот элемент имеет необязательный атрибут `id` и следующие дочерние элементы:
  - обязательный элемент `<java-type>` — полное имя Java-класса;
  - обязательный элемент `<root-type-qname>` или `<anonymous-type-qname>` — WSDL QName-имя XML-типа или его строка-идентификатор;
  - обязательный элемент `<qname-scope>` — возможные значения: `simpleType`, `complexType` или `element`;
  - дополнительный элемент `<variable-mapping>` имеет дополнительный атрибут `id`, определяет связь между Java-свойствами/полями и XML-элементами/атрибутами с помощью дочерних элементов: `<java-variable-name>` (обязательный, указывает имя Java-свойства/поля); `<data-member>` (дополнительный, если `true`, тогда Java-переменная — поле, если `false`, тогда — JavaBeans-свойство); `<xml-attribute-name>` или `<xml-element-name>`, или `<xml-wildcard>` (обязательный, указывает XML-имя или связь с элементом `<xsd:any>`);
- дополнительный элемент `<exception-mapping>` имеет дополнительный атрибут `id`, определяет связь между Java-исключениями и WSDL-описаниями ошибок с помощью дочерних элементов:
  - обязательный элемент `<exception-type>` — имя Java-класса исключения;
  - обязательный элемент `<wsdl-message>` — QName-имя WSDL-элемента `<message>`;

- дополнительный элемент `<wsdl-message-part-name>` — QName-имя WSDL-элемента `<part>`;
- дополнительный элемент `<constructor-parameter-order>` имеет дополнительный атрибут `id`, указывает порядок применения значений типа `complexType` в конструкторе Java-исключения с помощью дочерних элементов `<element-name>`;

❒ дополнительный элемент `<service-interface-mapping>` имеет дополнительный атрибут `id`, определяет связь между именем интерфейса `javax.xml.rpc.Service` или его расширением и WSDL-элементом `<service>`. Интерфейс `Service` реализуется генерируемым на стороне клиента классом-артефактом, объект которого обеспечивает для клиента доступ к Web-сервису с помощью своего метода, возвращающего экземпляр заглушки для интерфейса SEI Web-сервиса. Элемент `<service-interface-mapping>` имеет следующие дочерние элементы:

- обязательный элемент `<service-interface>` — имя интерфейса `javax.xml.rpc.Service` или его расширения;
- обязательный элемент `<wsdl-service-name>` — QName-имя WSDL-элемента `<service>`;
- дополнительный элемент `<port-mapping>` связывает имя WSDL-элемента `<port>` с именем экземпляра заглушки для интерфейса SEI Web-сервиса с помощью дочерних элементов `<port-name>` и `<java-port-name>`;

❒ обязательный элемент `<service-endpoint-interface-mapping>` имеет дополнительный атрибут `id`, определяет связь между интерфейсом SEI Web-сервиса и WSDL-описанием. Элемент `<service-endpoint-interface-mapping>` имеет следующие дочерние элементы:

- обязательный элемент `<service-endpoint-interface>` — полное имя SEI-интерфейса;
- обязательный элемент `<wsdl-port-type>` — QName-имя WSDL-элемента `<portType>`;
- обязательный элемент `<wsdl-binding>` — QName-имя WSDL-элемента `<binding>`;
- дополнительный элемент `<service-endpoint-method-mapping>` (рис. 3.11) связывает Java-методы SEI-интерфейса с WSDL-элементами `<operation>` с помощью дочерних обязательных элементов `<java-method-name>` и `<wsdl-operation>`, а также дополнительных элементов:
  - `<wrapped-element>` — элемент-обертка для кодировки document/literal wrapped;
  - `<method-param-parts-mapping>` связывает параметры Java-метода с WSDL-элементами `<message>` с помощью дочерних элементов: `<param-position>` — позиция параметра в методе; `<param-type>` — Java-тип параметра; `<wsdl-message-mapping>` — дочерние элементы `<wsdl-message>`,

<wsdl-message-part-name>, <parameter-mode> (режим параметра IN, OUT, INOUT); <soap-header> — если присутствует, то указывает, что параметр связан с SOAP-заголовком;

- <wsdl-return-value-mapping> связывает возвращаемые значения Java-метода с WSDL-элементами <message> с помощью дочерних элементов: <method-return-value> — Java-тип возвращаемого значения; <wsdl-message> — имя WSDL-элемента <message>; wsdl-<message-part-name> — если есть, указывает имя WSDL-элемента <part>.

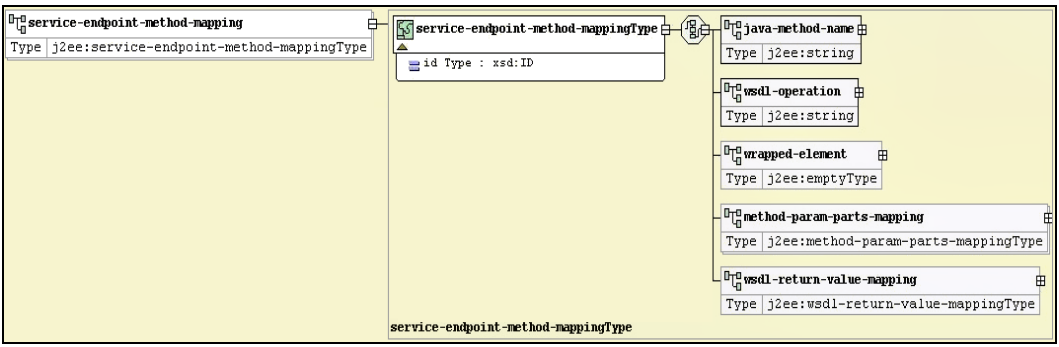


Рис. 3.11. Схема элемента <service-endpoint-method-mapping>

Разворачивать Web-сервис на стороне сервера можно также с помощью JAX-RPC-инструмента wsdeploy, который считывает переносимый WAR-модуль Web-сервиса и создает новый WAR-файл со всеми необходимыми для работы Web-сервиса артефактами и готовый для развертывания. В этом случае переносимый WAR-модуль Web-сервиса содержит SEI-интерфейс, класс его реализации, дескриптор развертывания и конфигурационный файл jaxrpc-ri.xml, описывающий конечную точку Web-сервиса. Общий синтаксис инструмента wsdeploy:

```
wsdeploy -o <output-war-file> <input-war-file> <options>
```

Опции инструмента wsdeploy описаны в табл. 3.5.

Таблица 3.5. *Опции инструмента wsdeploy*

Опция	Описание
-classpath <*path*>	Определяет дополнительный classpath
-keep	Сохраняет временные файлы
-o <*output WAR file*>	Указывает расположение исходящего WAR-файла
-source <*version*>	Генерирует код для указанной JAX-RPC версии — 1.0.1, 1.0.3 и 1.1 (по умолчанию)
-tmpdir <*directory*>	Указывает каталог для временных файлов
-verbose	Выводит информацию о процессе компиляции
-version	Выводит версию компилятора

Конфигурационный файл jaxrpc-ri.xml описывает Web-сервис и имеет следующую структуру (рис. 3.12):

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
             version="..."
             targetNamespaceBase="..."
             typeNamespaceBase="..."
             urlPatternBase="...">
  <endpoint>...</endpoint>
  <endpointMapping/>
</webServices>
```

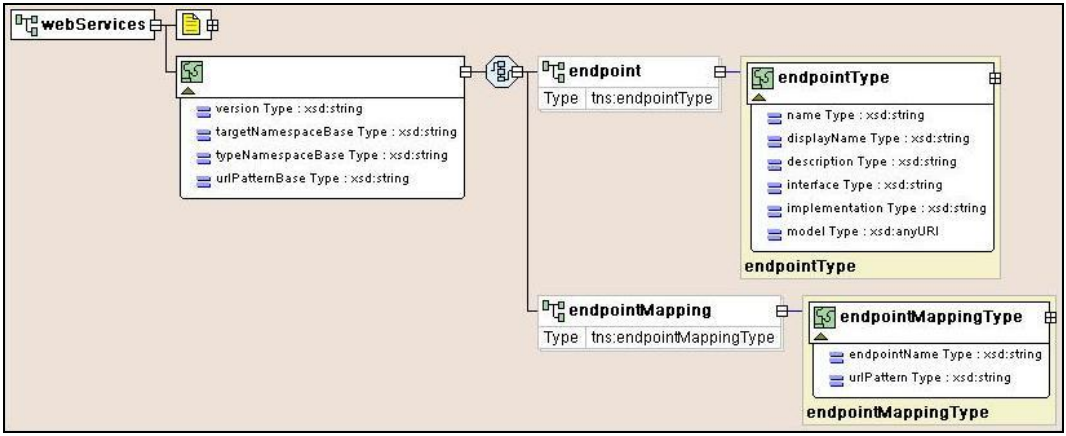


Рис. 3.12. Общая схема файла jaxrpc-ri.xml

Корневой элемент `<webServices>` файла jaxrpc-ri.xml имеет следующие атрибуты и дочерние элементы:

- обязательный атрибут `version` — номер версии;
- дополнительный атрибут `targetNamespaceBase` — базовый URI целевого пространства имен генерируемого WSDL-документа;
- дополнительный атрибут `typeNamespaceBase` — базовый URI целевого пространства имен XML-схемы WSDL-документа;
- дополнительный атрибут `urlPatternBase` — базовый URL для конечной точки Web-сервиса;
- дополнительный элемент `<endpoint>` — описание конечной точки Web-сервиса с помощью следующих атрибутов и дочерних элементов:
  - обязательный атрибут `name` — имя конечной точки;
  - дополнительный атрибут `displayName` — отображаемое инструментами разработки имя конечной точки;
  - дополнительный атрибут `description` — описание конечной точки;

- дополнительный атрибут `interface` — имя SEI-интерфейса Web-сервиса;
- дополнительный атрибут `implementation` — имя класса реализации SEI-интерфейса;
- дополнительный атрибут `model` — URI-адрес model-файла;
- дополнительный элемент `<handlerChains>` описывает цепочки обработчиков SOAP-сообщений по умолчанию. Этот элемент содержит элементы `<chain>`, имеющие атрибуты `runAt` (обязательный, указывает, где работает обработчик — на стороне клиента или сервера), и `roles` (дополнительный, указывает список SOAP-ролей для цепочки). Элемент `<chain>` также может содержать элементы `<handler>`, определяющие `javax.xml.rpc.handler.Handler`-классы для цепочки. Элемент `<handler>` имеет атрибуты `className` (обязательный, указывает имя класса-обработчика), `headers` (дополнительный, указывает список обрабатываемых SOAP-заголовков) и дополнительные элементы `<property>`, определяющие свойства обработчика (атрибуты `name` и `value`);

- дополнительный элемент `<endpointMapping>` связывает конечную точку Web-сервиса с URL-шаблоном клиентского запроса с помощью атрибутов `endpointName` (имя конечной точки) и `urlPattern` (URL-шаблон клиентского запроса).

JAX-RPC API

Программный интерфейс JAX-RPC API состоит из следующих пакетов:

- `javax.xml.rpc` содержит классы и интерфейсы для создания JAX-RPC клиента;
- `javax.xml.rpc.encoding` обеспечивает связывание Java-типов с XML-типами;
- `javax.xml.rpc.handler` и `javax.xml.rpc.handler.soap` содержат классы и интерфейсы для создания обработчиков SOAP-сообщений;
- `javax.xml.rpc.holders` обеспечивает для клиента Web-сервиса поддержку параметров `OUT` и `INOUT` удаленного метода;
- `javax.xml.rpc.server` для JAX-RPC Web-сервиса, развернутого в Servlet-контейнере обеспечивает управление жизненным циклом и контекстом конечной точки;
- `javax.xml.rpc.soap` содержит Java-исключение `SOAPFaultException`, представляющее SOAP-ошибку.

Подробно программный интерфейс JAX-RPC API рассмотрен в приложении "JAX-RPC API" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Пример использования JAX-RPC

Рассмотрим создание простого JAX-RPC Web-сервиса и его клиента с использованием среды разработки Eclipse IDE for Java EE Developers (**http://www.eclipse.org/downloads/**).

1. Откроем среду Eclipse и переключимся на **<Java EE> perspective**.
2. В меню **File** выберем **New | Other | Web | Dynamic Web Project** и нажмем кнопку **Next**. Введем имя проекта JAX_RPC_Example и, нажав кнопку **New Runtime**, выберем среду выполнения Apache Tomcat v7.0, предварительно инсталлировав сервер Apache Tomcat v7.0, доступный по адресу **http://tomcat.apache.org/download-70.cgi**. При этом не забудем указать **Create a new local server**.
3. После нажатия кнопок **Next** и **Browse** укажем каталог установленного сервера Apache Tomcat v7.0. Нажав кнопку **Installed JREs**, зададим каталог JDK 6 и нажмем кнопку **Finish**.
4. В диалоговом окне **New Dynamic Web Project** дважды нажмем кнопку **Next**, укажем **Generate web.xml deployment descriptor** и нажмем кнопку **Finish**.

   На компакт-диске

   Проект JAX_RPC_Example расположен в папке Примеры\Глава3\JAX_RPC_Example компакт-диска.

   В результате в окне **Project Explorer** появится сгенерированная средой Eclipse основа Web-приложения.
5. Щелкнем правой кнопкой мыши на узле **Java Resources: src** проекта и выберем пункты **New | Class**, введем имя пакета `jaxrpcexample` и имя класса `HelloWebService` и нажмем кнопку **Finish**.
6. В окне редактора дополним код класса `HelloWebService`:

```
package jaxrpcexample;
public class HelloWebService {
  public String sayHello (String name){
    String str = "Hello"+"  "+name;
    return str;
}}
```

7. Щелкнем правой кнопкой мыши на узле **JAX_RPC_Example** проекта, выберем пункты **New | Other | Web Services | Web Service** и нажмем кнопку **Next**. Далее в строке **Service implementation** введем `jaxrpcexample.HelloWebService` и нажмем кнопку **Finish**.

В результате будет сгенерирован WSDL-документ (см. папку Примеры\Глава3\JAX_RPC_Example\WebContent\wsdl компакт-диска) и Web-сервис HelloWebService будет развернут в Web-сервере Apache Tomcat v7.0. Проверить это можно, открыв встроенный браузер нажатием кнопки с изображением значка браузера в верхней панели среды Eclipse и введя адрес **http://localhost:8080/JAX_RPC_Example/services/HelloWebService**, при переходе по которому откроется страница приветствия (рис. 3.13).

Для создания простого JAX-RPC-клиента Web-сервиса HelloWebService:

1. В меню **File** выберем пункты **New | Other | Web | Dynamic Web Project** и нажмем кнопку **Next**.

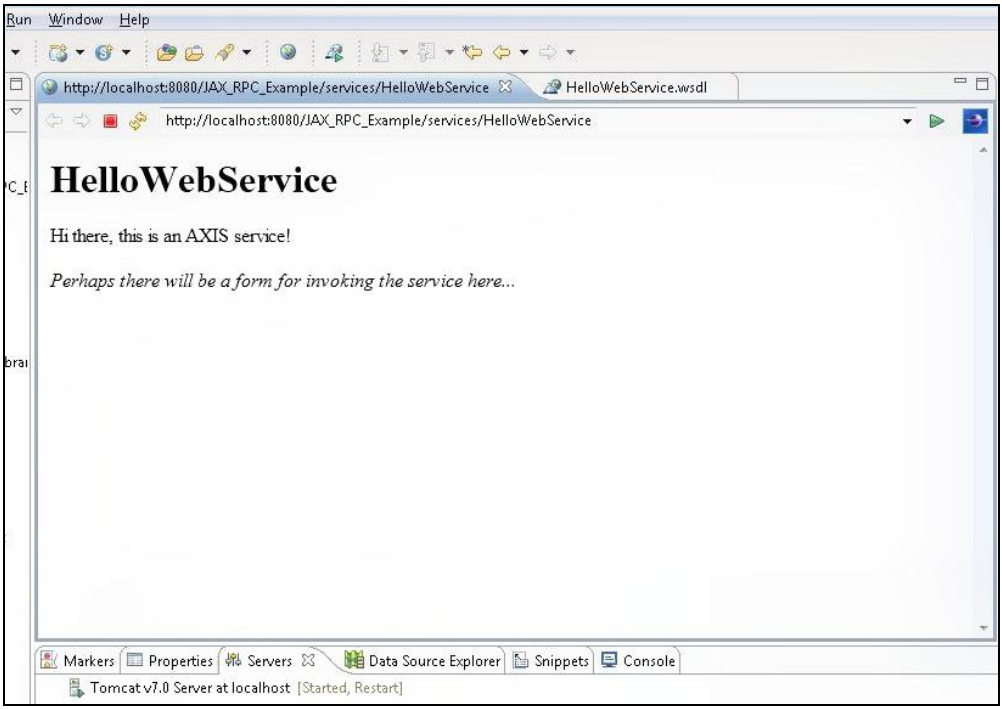


Рис. 3.13. Страница приветствия Web-сервиса HelloWebService

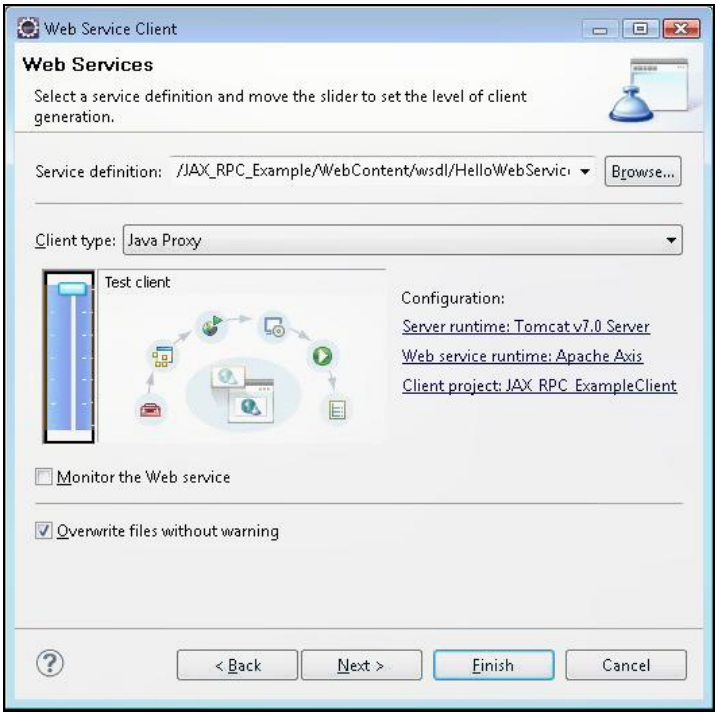


Рис. 3.14. Диалоговое окно создания клиента Web-сервиса

2. Введем имя проекта JAX_RPC_ExampleClient и нажмем кнопку **Finish**.
3. Щелкнем правой кнопкой мыши на узле **JAX_RPC_ExampleClient** проекта и выберем пункты **New | Other | Web Services | Web Service Client**. Нажмем кнопку **Next**. Далее с помощью кнопки **Browse** укажем файл HelloWebService.wsdl, установим бегунок в положение **Test client** (рис. 3.14) и нажмем кнопку **Finish**.

На компакт-диске

Проект JAX_RPC_ExampleClient расположен в папке Примеры\Глава3 компакт-диска.

В результате средой Eclipse будет сгенерирован и развернут клиент Web-сервиса (рис. 3.15).

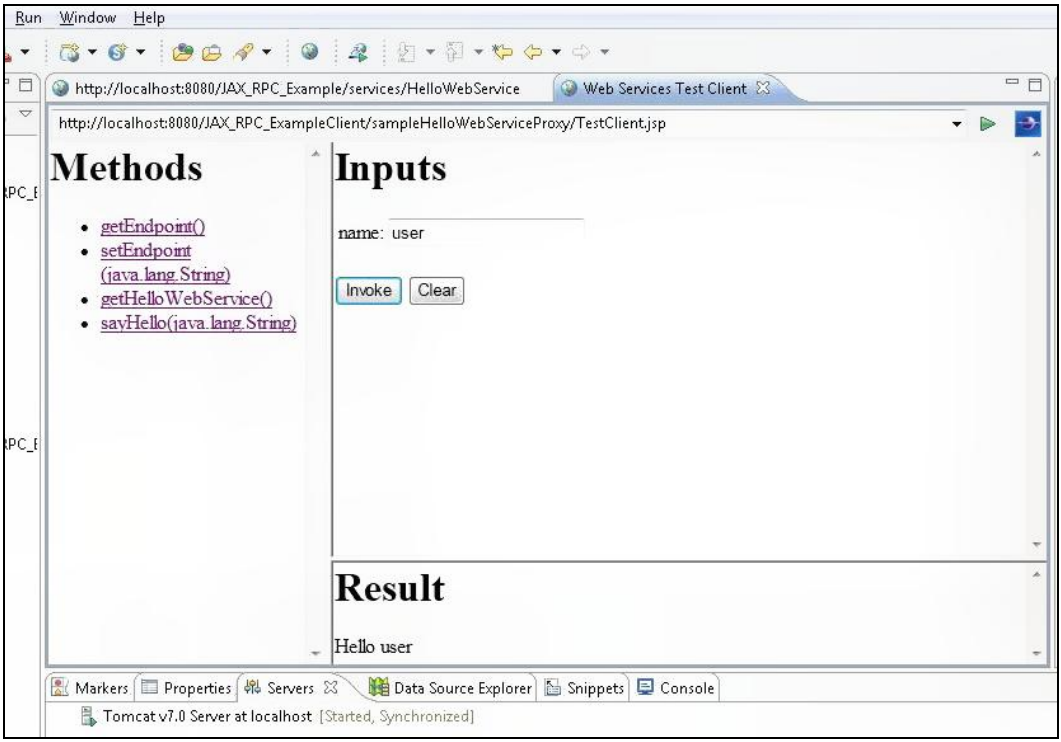


Рис. 3.15. Клиент Web-сервиса HelloWebService

JAX-WS

Спецификация JAX-WS — это следующий шаг, по сравнению со спецификацией JAX-RPC, на пути реализации технологии Web-сервисов платформой Java. Технология JAX-WS еще больше упрощает разработку Web-сервисов и их клиентов, взаимодействующих по SOAP-протоколу, скрывая при этом от разработчика все детали обработки SOAP-сообщений.

Основные отличия технологии JAX-WS от JAX-RPC:

- спецификация JAX-RPC 1.1 поддерживает SOAP 1.1 и WSDL 1.1, а спецификация JAX-WS 2.0 добавляет поддержку SOAP 1.2. Спецификация JAX-WS также декларирует поддержку WSDL 2.0;
- спецификация JAX-WS, в отличие от JAX-RPC, обеспечивает создание RESTful Web-сервисов, поддерживая WSDL-расширение HTTP binding;
- спецификация JAX-RPC 1.1 поддерживает WS-I BasicProfile 1.0, а спецификация JAX-WS 2.0 — WS-I BasicProfile 1.1;

> ***Примечание***
>
> Спецификация WS-I BasicProfile определяет правила использования спецификаций SOAP, WSDL и UDDI, обеспечивающих совместимость систем, созданных на их основе. Спецификация WS-I BasicProfile 1.1 основывается на спецификации WS-I BasicProfile 1.0 с добавлением вложений SOAP-сообщений SOAP Messages with Attachments (SwA) и WSDL 1.1 расширения MIME Binding.

- спецификация JAX-WS добавляет аннотации JSR 175 (A Metadata Facility for the Java Programming Language) и JSR 181 (Web Services Metadata for the Java Platform);
- спецификация JAX-RPC использует собственный механизм связывания Java-представления с XML-представлением. Спецификация JAX-WS описывает связывание "WSDL ↔ Java", но связывание данных определяет с помощью спецификации JAXB;
- спецификация JAX-WS добавляет асинхронные клиентские вызовы Web-сервиса;
- динамическая модель JAX-WS отличается от модели Dynamic Invocation Interface (DII) JAX-RPC;
- спецификация JAX-WS позволяет создавать Web-сервисы, ориентированные как на сообщения, так и на модель RPC;
- спецификация JAX-WS упрощает доступ к SOAP-сообщениям как на стороне клиента, так и на стороне сервера;
- спецификация JAX-RPC определяет управление сессией с помощью HTTP-протокола. Спецификация JAX-WS добавляет управление сессией с помощью SOAP-сообщений;
- технология JAX-WS включена в платформу Java SE;
- спецификация JAX-WS с помощью JAXB добавляет поддержку MTOM;
- обработчики SOAP-сообщений JAX-RPC основываются на SAAJ 1.2, а обработчики JAX-WS — на SAAJ 1.3;
- технология JAX-WS поддерживает транспортные протоколы, отличные от HTTP.

JAX-WS API

Программный интерфейс JAX-WS API состоит из следующих пакетов:

- `javax.xml.ws` содержит основные классы и интерфейсы, используемые как на стороне клиента, так и на стороне сервера;
- `javax.xml.ws.handler` и `javax.xml.ws.handler.soap` позволяют создавать обработчиков SOAP-сообщений;
- `javax.xml.ws.http` обеспечивает идентификацию связывания XML/HTTP;
- `javax.xml.ws.soap` обеспечивает идентификацию связывания SOAP/HTTP;
- `javax.xml.ws.spi` используется внутри JAX-WS API и обеспечивает связь с JAX-WS-реализацией;
- `javax.xml.ws.spi.http` обеспечивает развертывание JAX-WS Web-сервисов в HTTP-контейнере;
- `javax.xml.ws.wsaddressing` обеспечивает поддержку WS-Addressing;
- `javax.jws` и `javax.jws.soap` обеспечивают аннотации, упрощающие разработку Web-сервисов и их клиентов;
- `javax.annotation` определяет общие аннотации.

Программный интерфейс JAX-WS API подробно рассмотрен в приложении "JAX-WS API" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Модель программирования JAX-WS

Модель программирования на стороне сервера

Для создания JAX-WS Web-сервиса можно, но необязательно, создать SEI-интерфейс, маркированный аннотацией `@WebService`, и необходимо создать SEI-класс — класс реализации конечной точки Web-сервиса, который может быть промаркирован либо аннотацией `@WebService`, либо аннотацией `@WebServiceProvider`. SEI-класс Web-сервиса не должен быть абстрактным и финальным, должен иметь публичный конструктор по умолчанию и не должен иметь finalize-методов. Для управления жизненным циклом SEI-класс может иметь методы, промаркированные аннотациями `@PostConstruct` и `@PreDestroy`.

Аннотация `@WebService` применяется для создания Web-сервисов, ориентированных на вызов удаленных процедур RPC, а аннотация `@WebServiceProvider` позволяет создавать ориентированные на сообщения RESTful Web-сервисы.

При использовании аннотации `@WebService` бизнес-методы Web-сервиса маркируются аннотацией `@WebMethod`, при этом они должны быть публичными и не должны быть статическими или финальными, а параметры и возвращаемые значения бизнес-методов должны быть совместимы с JAXB-типами. Параметры бизнес-методов могут маркироваться аннотациями `@WebParam`. Параметры `OUT` и `INOUT` должны быть Holder-типа.

Бизнес-методы Web-сервиса могут быть дополнительно промаркированы аннотациями `@WebResult`, `@Oneway`, `@Action`, `@RequestWrapper`, `@ResponseWrapper`.

Совместно с аннотацией `@WebService` могут использоваться аннотации `@HandlerChain`, `@SOAPBinding`, `@BindingType`, `@RespectBinding`, `@Addressing`, `@MTOM`.

SEI-класс, промаркированный аннотацией `@WebService`, может использовать объект `WebServiceContext`, введенный с помощью аннотации `@Resource`, для доступа к таким объектам, как `java.security.Principal`, `javax.servlet.ServletContext` и `javax.servlet.http.HttpServletRequest`, обеспечивающим безопасность и управление сессией Web-сервиса.

Вместе с SEI-классом могут быть созданы классы обработчиков сообщений, реализующих интерфейсы `SOAPHandler<T extends SOAPMessageContext>` или `LogicalHandler<C extends LogicalMessageContext>`.

Аннотация `@WebServiceProvider` маркирует класс Web-сервиса, реализующий интерфейс `Provider<T>`. В этом случае за анализ входящего сообщения и за создание исходящего сообщения отвечает сам класс Web-сервиса, а не JAX-WS-реализация.

Совместно с аннотацией `@WebServiceProvider` используются аннотации `@ServiceMode` и `@BindingType`. В случае применения аннотации `@BindingType(value=HTTPBinding.HTTP_BINDING)` класс, реализующий интерфейс `Provider<T>`, представляет RESTful Web-сервис. При этом объект `WebServiceContext` обеспечивает информацию о HTTP-методе запроса.

Для развертывания Web-сервиса на платформе Java SE необходимо создать Java-класс с `main`-методом, использующим Endpoint API.

Вызов Web-сервиса на стороне сервера состоит из следующих шагов, выполняемых JAX-WS-реализацией:

1. Контейнер, в котором развернута конечная точка Web-сервиса как Servlet-объект, прослушивающий URL-адрес конечной точки, получает SOAP-сообщение.
2. Создается объект `MessageContext`, обеспечивающий SAAJ-представление сообщения.
3. Вызывается цепочка обработчиков входящего сообщения.
4. Определяется вызываемая WSDL-операция и, исходя из ее связи с бизнес-методом, определяется вызываемый Java-метод класса Web-сервиса.
5. SOAP-сообщение десериализуется JAXB-реализацией в JavaBean-компонент запроса, содержащий Java-объекты, передаваемые бизнес-методу в качестве параметров.
6. Вызывается Java-метод класса Web-сервиса, возвращающий в качестве результата Java-объект, с помощью которого создается JavaBean-компонент ответа.
7. JAXB-реализация производит сериализацию JavaBean-компонента ответа в исходящее SOAP-сообщение.

8. Обновляется объект `MessageContext` и вызывается цепочка обработчиков исходящего сообщения.
9. Исходящее SOAP-сообщение посылается по транспортному протоколу как ответ на запрос.

Модель программирования на стороне клиента

Существуют два типа JAX-WS клиента — Proxy-клиент и Dispatch-клиент.

Для работы Proxy-клиента используются предварительно сгенерированные артефакты, содержащие класс реализации `javax.xml.ws.Service`, промаркированный аннотацией `@WebServiceClient`.

Код Proxy-клиента использует конструктор Service-класса реализации или аннотацию `@WebServiceRef` для создания объекта `Service`. Для JAX-WS-клиента может быть установлена цепочка обработчиков сообщений с помощью метода `setHandlerResolver()` объекта `Service`, использующего переопределенный метод `getHandlerChain()` объекта `javax.xml.ws.handler.HandlerResolver`.

Для получения Proxy-объекта применяется метод `getPort()` Service-объекта. Так как Proxy-объект реализует интерфейс `javax.xml.ws.BindingProvider`, то далее можно конфигурировать Proxy-объект с помощью методов объекта `BindingProvider`, полученного следующим образом:

```
javax.xml.ws.BindingProvider bp=(javax.xml.ws.BindingProvider)proxy;
```

После конфигурации Proxy-объекта вызывается требуемый удаленный метод Web-сервиса.

Proxy-клиент может вызывать Web-сервис асинхронно. Для этого WSDL-описание Web-сервиса должно содержать объявление связей (binding declarations) `jaxws:enableAsyncMapping` со значением `true`. Тогда при генерации артефактов на стороне клиента SEI-интерфейс будет содержать методы типа `Response<T> invokeAsync(T msg)` и `java.util.concurrent.Future<?> invokeAsync(T msg, AsyncHandler<T> handler)`. Использование объекта `AsyncHandler` потребует реализации класса `AsyncHandler`-обработчика на стороне клиента.

Для работы Dispatch-клиента не требуются предварительно сгенерированные артефакты.

Service-объект при этом создается динамически с помощью статического метода `create()`. После этого добавляется порт Web-сервиса, используя метод `addPort()` Service-объекта. Далее методом `createDispatch()` получается объект Dispatch, который обеспечивает низкоуровневую обработку и создание XML-сообщений.

Объект Dispatch, так же как и объект Proxy, реализует интерфейс `BindingProvider`, и поэтому может быть сконфигурирован с помощью его методов.

После конфигурации Dispatch-объекта можно вызывать Web-сервис синхронно или асинхронно, используя polling-метод или callback-метод.

Таким образом, вызов Web-сервиса на стороне JAX-WS-клиента состоит из следующих шагов:

1. Создание экземпляра реализации конечной точки Web-сервиса.
2. Вызов конечной точки Web-сервиса.
3. Сериализация параметров вызываемого SEI-метода в XML-элементы и создание сообщения.
4. Вызов обработчиков сообщения.
5. Отправка сообщения по транспортному протоколу.
6. Получение ответного сообщения.
7. Десериализация ответного сообщения в Java-объекты.

Развертывание JAX-WS Web-сервисов и JAX-WS-клиентов

После создания SEI-класса, Handler-классов, handler-файла развернуть Web-сервис можно как на платформе Java SE — с помощью Endpoint API, так и на платформе Java EE — используя deploy-инструмент сервера приложений.

Для подготовки к развертыванию Web-сервиса существует JAX-WS-инструмент wsgen, генерирующий переносимые артефакты, необходимые для работы Web-сервиса.

Общий синтаксис командной строки инструмента wsgen:

`wsgen [options] service_implementation_class`

а его опции описаны в табл. 3.6.

Таблица 3.6. *Опции инструмента wsgen*

Опция	**Описание**
`-cp path` или `-classpath path`	Расположение входящих класс-файлов
`-d directory`	Расположение генерируемых файлов
`-extension true|false`	Использование расширений спецификации JAX-WS
`-help`	Вывод документации инструмента
`-keep`	Сохранение генерируемых исходных файлов
`-portname name`	Определяет имя генерируемого элемента `<wsdl:port>`
`-r directory`	Расположение генерируемого WSDL-документа
`-s directory`	Расположение генерируемых исходных файлов
`-servicename name`	Определяет имя генерируемого элемента `<wsdl:service>`
`-verbose`	Вывод информации о процессе генерации
`-version`	Вывод версии
`-wsdl [:protocol]`	Генерация WSDL-документа с указанием протокола для элемента `<wsdl:binding>`, по умолчанию — `soap1.1`

После генерации необходимых артефактов, таких как JavaBean-компоненты запроса и ответа, SEI-класс, все вспомогательные Java-классы, сгенерированные артефакты и дескрипторы развертывания упаковываются в готовый для развертывания WAR-модуль.

После создания Proxy-клиента требуется сгенерировать необходимые для его работы артефакты. Для этого можно воспользоваться JAX-WS-инструментом wsimport, генерирующим из WSDL-документа переносимые артефакты, такие как SEI-интерфейс, Service-класс, классы Java-исключений, асинхронный JavaBean-компонент ответа и классы JAXB-типов данных. Использование инструмента wsimport позволяет также создать Web-сервис, исходя из WSDL-документа. При этом все сгенерированные артефакты вместе с SEI-классом и дескрипторами развертывания упаковываются в готовый для развертывания WAR-модуль.

Общий синтаксис командной строки инструмента wsimport:

`wsimport [options] wsdl_file`

а его опции описаны в табл. 3.7.

Таблица 3.7. *Опции инструмента wsimport*

Опция	**Описание**
`-b directory`	Расположение внешних JAX-WS- или JAXB-файлов объявлений связей (binding declarations)
`-catalog`	Определяет файл XML-каталога, связывающий внешние ссылки ресурсов с локальными объектами
`-d directory`	Расположение генерируемых файлов
`-extension`	Использование расширений спецификации JAX-WS
`-help`	Вывод документации инструмента
`-httpproxy:host:port`	Определяет сервер HTTP proxy, по умолчанию порт 8080
`-keep`	Сохранение генерируемых исходных файлов
`-p`	Переопределяет имя пакета
`-s directory`	Расположение генерируемых исходных файлов
`-verbose`	Вывод информации о процессе генерации
`-version`	Вывод версии
`-wsdllocation URI`	Определяет значение элементов `@WebService.wsdlLocation` и `@WebServiceClient.wsdlLocation`

При генерации артефактов инструментом wsimport можно переопределить связывание "WSDL → Java" по умолчанию с помощью JAX-WS- или JAXB-объявлений связей. Объявления связей спецификации JAXB для XML-схемы, используемой WSDL-документом, мы уже рассматривали в *разд. "JAXB" ранее в этой главе*. Далее рассмотрим объявления связей, определенные спецификацией JAX-WS для самого WSDL-документа.

Объявления связей (binding declarations) JAX-WS имеют пространство имен `http://java.sun.com/xml/ns/jaxws` и могут быть сгруппированы в отдельный файл или быть встроены в существующий WSDL-документ.

Корневым элементом JAX-WS объявлений связей является элемент `<jaxws:bindings>`, имеющий следующие необязательные атрибуты:

- `wsdlLocation` — URI-адрес WSDL-документа, к которому применяются объявления связей;
- `node` — XPath-выражение, указывающее WSDL-элемент, к которому применяются объявления связей;
- `version` — версия JAX-WS спецификации, по умолчанию 2.0.

Корневой элемент `<jaxws:bindings>` может содержать несколько элементов `<jaxws:bindings>`, образующих блоки объявлений связей.

Элементы `<jaxws:bindings>` могут содержать помимо JAX-WS-объявлений связей также JAXB-объявления связей.

Спецификация JAX-WS определяет следующие дочерние элементы элемента `<jaxws:bindings>`, применяемые к WSDL-документу в целом:

- `<jaxws:package name="xs:string">` — имя Java-пакета. Дочерний элемент `<jaxws:javadoc>` содержит документацию уровня пакета;
- `<jaxws:enableWrapperStyle>` — если `true` (по умолчанию), тогда для операций предусмотрен wrapper-стиль;
- `<jaxws:enableAsyncMapping>` — если `true` (по умолчанию), тогда для операций генерируются методы типа `Response<T> invokeAsync(T msg)` и `java.util.concurrent.Future<?> invokeAsync(T msg, AsyncHandler<T> handler)`;
- `<jaxws:enableMIMEContent>` — если `true` (по умолчанию), тогда для операций используется информация WSDL-элемента `<mime:content>`, чтобы связать MIME-тип данных с Java-типом; если значение элемента — `false`, тогда данные связываются с байтовым массивом.

Спецификация JAX-WS определяет следующие дочерние элементы элемента `<jaxws:bindings>`, применяемые к элементу `<wsdl:portType>`:

- `<jaxws:class name="xs:string">` — имя SEI-интерфейса. Дочерний элемент `<jaxws:javadoc>` содержит документацию уровня класса;
- `<jaxws:enableWrapperStyle>` и `<jaxws:enableAsyncMapping>`.

Спецификация JAX-WS определяет следующие дочерние элементы элемента `<jaxws:bindings>`, применяемые к элементу `<wsdl:portType>`/`<wsdl:operation>`:

- `<jaxws:method name="xs:string">` — имя Java-метода SEI-интерфейса. Дочерний элемент `<jaxws:javadoc>` содержит документацию уровня метода;
- `<jaxws:enableWrapperStyle>` и `<jaxws:enableAsyncMapping>`;
- `<jaxws:parameter>` — связывание элементов `<wsdl:message>`/`<wsdl:part>` с Java-параметрами метода с помощью атрибутов `part`, `childElementName` и `name`.

Спецификация JAX-WS определяет дочерний элемент `<jaxws:class name="xs:string">` (имеет дочерний элемент `<jaxws:javadoc>`) элемента `<jaxws:bindings>`, применяемый к элементу `<wsdl:portType>`/`<wsdl:operation>`/`<wsdl:fault>`.

Спецификация JAX-WS определяет дочерний элемент `<jaxws:enableMIMEContent>` элемента `<jaxws:bindings>`, применяемый к элементу `<wsdl:binding>`.

Спецификация JAX-WS определяет следующие дочерние элементы элемента `<jaxws:bindings>`, применяемые к элементу `<wsdl:binding>`/`<wsdl:operation>`:

- `<jaxws:enableMIMEContent>` и `<jaxws:parameter part="xs:string" childElementName="xs:QName"? name="xs:string"/>`;
- `<jaxws:exception part="xs:string">` — связывание ошибки с Java-классом исключения с помощью дочернего элемента `<jaxws:class name="xs:string">` (имеет дочерний элемент `<jaxws:javadoc>`).

Спецификация JAX-WS определяет дочерний элемент `<jaxws:class name="xs:string">` (имеет дочерний элемент `<jaxws:javadoc>`) элемента `<jaxws:bindings>`, применяемый к элементу `<wsdl:service>`.

Спецификация JAX-WS определяет дочерние элементы `<jaxws:methodname="xs:string">` (имеет дочерний элемент `<jaxws:javadoc>`) и `<jaxws:provider/>` (указывает, что порт реализует интерфейс `javax.xml.ws.Provider`, при этом не генерируется SEI-интерфейс) элемента `<jaxws:bindings>`, применяемые к элементу `<wsdl:port>`.

Пример создания JAX-WS Web-сервиса и JAX-WS-клиента

Рассмотрим создание JAX-WS Web-сервиса и клиента JAX-WS с использованием среды NetBeans (**http://netbeans.org/**).

1. Откроем среду NetBeans и в меню **Файл** выберем пункты **Создать проект | Java Web | Веб-приложение**, нажмем кнопку **Далее**.
2. Введем имя проекта JAX_WS_WebService и нажмем кнопку **Далее**.
3. В раскрывающемся списке **Сервер** выберем вариант **GlassFish Server 3** и нажмем кнопку **Завершить**.

   На компакт-диске

   Проект JAX_WS_WebService расположен в папке Примеры\Глава3\JAX_WS_NetBeans компакт-диска.

4. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebService**, выберем пункты **Создать | Другие | Веб-службы | Веб-служба** и нажмем кнопку **Далее**, введем имя Web-сервиса JAXWSExample и имя пакета `jaxwsexample` и нажмем кнопку **Завершить**.
5. Дополним код класса `JAXWSExample`, щелкнув на узле **Веб-службы | JAXWSExample** и выбрав пункт **Добавить операцию**:

```
package jaxwsexample;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService()
public class JAXWSExample {
/**
  * Операция веб-службы
*/
  @WebMethod(operationName = "getHello")
  public String getHello(@WebParam(name = "name")
  String name) {
    //TODO write your implementation code here:
    return "Hello"+" "+ name;
  }
}
```

6. Чтобы развернуть созданный Web-сервис, в окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebService**, выберем пункт **Построить** и затем — **Развернуть**.
7. Протестировать Web-сервис можно, щелкнув в окне **Проекты** правой кнопкой мыши на узле **JAX_WS_WebService | Веб-службы | JAXWSExample** и выбрав пункт **Тестировать веб-сервис**.

Для создания клиента Web-сервиса:

1. В меню **Файл** выберем пункты **Создать проект | Java Web | Веб-приложение**, нажмем кнопку **Далее**, введем имя проекта JAX_WS_WebClient и нажмем кнопку **Далее**.
2. В раскрывающемся списке **Сервер** выберем вариант **GlassFish Server 3** и нажмем кнопку **Завершить**.

> ***На компакт-диске***
> Проект JAX_WS_WebClient расположен в папке Примеры\Глава3\JAX_WS_NetBeans компакт-диска.

3. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebClient**, выберем пункты **Создать | Другие | Веб-службы | Клиент веб-службы** и нажмем кнопку **Далее**. С помощью кнопки **Обзор** выберем Web-сервис JAXWSExample и нажмем кнопку **OK**, а затем кнопку **Завершить**.

В результате для клиента Web-сервиса средой NetBeans будут сгенерированы следующие артефакты:

- в папке JAX_WS_WebClient\web\WEB-INF\wsdl\localhost_8080\JAX_WS_WebService копии WSDL-описания и XML-схемы — JAXWSExampleService.wsdl и JAXWSExampleService.xsd_1.xsd;

- в папке JAX_WS_WebClient\web\WEB-INF конфигурационный файл jax-ws-catalog.xml — каталог, используемый средой выполнения JAX-WS для разрешения URI-ссылок на XML-схему и WSDL-описание Web-сервиса. В данном случае каталог связывает URL-адреса документов Web-сервиса с их локальными копиями;
- в папке JAX_WS_WebClient\build\generated-sources Java-файлы:
  - GetHello.java — код JavaBean-класса, представляющего данные запроса;
  - GetHelloResponse.java — код JavaBean-класса, представляющего данные ответа;
  - JAXWSExample.java — SEI-интерфейс Web-сервиса;
  - JAXWSExampleService.java — Service-класс для клиента Web-сервиса;
  - ObjectFactory.java — класс-фабрика, отвечающий за создание экземпляров Java-представлений XML-данных;
  - package-info.java — конфигурационный файл, связывающий XML-пространство имен с именем Java-пакета.

Сгенерированные артефакты будут использоваться сервлетом для взаимодействия с Web-сервисом. Для создания сервлета:

1. Щелкнем правой кнопкой мыши в окне **Проекты** на узле **JAX_WS_WebClient** и выберем пункты **Создать | Другие | Веб | Сервлет**, нажмем кнопку **Далее**.
2. Введем имя класса `WebServiceClient` и имя пакета `webserviceclient`, нажмем кнопку **Далее**, укажем **Добавление информации к дескриптору развертывания (web.xml)** и нажмем кнопку **Завершить**.
3. В окне редактора, в коде класса `WebServiceClient`, щелкнем правой кнопкой мыши и выберем пункты **Вставить код | Вызов операции веб-службы | JAX_WS_WebClient | JAXWSExampleService | JAXWSExamplePort | getHello**. Нажмем кнопку **OK**.

В результате в класс `WebServiceClient` будет добавлен код:

```
import javax.xml.ws.WebServiceRef;
. . .
@WebServiceRef(wsdlLocation = "WEB-
INF/wsdl/localhost_8080/JAX_WS_WebService/JAXWSExampleService.wsdl")
private JAXWSExampleService service;
. . .
private String getHello(java.lang.String name) {
        jaxwsexample.JAXWSExample port = service.getJAXWSExamplePort();
        return port.getHello(name);  }
```

Модифицируем код JSP-страницы index.jsp узла **Веб-страницы** проекта JAX_WS_WebClient и код метода `processRequest()` класса `WebServiceClient` (листинги 3.8 и 3.9).

Листинг 3.8. Код JSP-страницы index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Service Hello</h1>
    <form name="Name" method="post" action="WebServiceClient">
      <p><textarea cols="1" rows="1" name="TextAreaName"
                  ID="TextareaName" ></textarea>
      <p>
      <input type="submit" value="Отправить" name="namebutton">
    </form>
  </body>
</html>
```

Листинг 3.9. Код метода `processRequest()` класса `WebServiceClient`

```
protected void processRequest(HttpServletRequest request,
       HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      String Name = request.getParameter("TextAreaName");
      String Response = getHello (Name);
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet WebServiceClient</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>"+Response+"</h1>");
      out.println("</body>");
      out.println("</html>");
    } finally {
    out.close();
  }
}
```

После развертывания Web-сервиса и его клиента с помощью нажатия правой кнопкой мыши в окне **Проекты** и выбора пунктов **Построить** и **Развернуть** запустим клиента Web-сервиса выбором команды **Выполнить**. В результате в окне браузера

появится страница приветствия Web-приложения JAX_WS_WebClient с полем ввода имени, заполнив которое и нажав кнопку **Отправить** в ответ можно увидеть строку, возвращаемую методом `getHello()` Web-сервиса JAXWSExample.

Для создания JAX-WS-клиента платформы Java SE:

1. В среде NetBeans в меню **Файл** последовательно выберем пункты **Создать проект | Java | Приложение Java**, нажмем кнопку **Далее**, введем имя проекта JAX_WS_SEClient и нажмем кнопку **Завершить**.

   На компакт-диске

   Проект JAX_WS_SEClient расположен в папке Примеры\Глава3\JAX_WS_NetBeans компакт-диска.

2. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_SEClient**, выберем пункты **Создать | Другие | Веб-службы | Клиент веб-службы** и нажмем кнопку **Далее**. С помощью кнопки **Обзор** выберем Web-сервис JAXWSExample и нажмем кнопку **OK**, а затем кнопку **Завершить**.
3. В окне редактора в коде класса `Main` щелкнем правой кнопкой мыши, выберем пункты **Вставить код | Вызов операции веб-службы | JAX_WS_WebClient | JAXWSExampleService | JAXWSExamplePort | getHello** и нажмем кнопку **OK**. В результате класс `Main` дополнится методом `getHello()`.
4. Введем в метод `main()` класса `Main` код:

   ```
   System.out.println(getHello("User"));
   ```

Теперь, после запуска приложения JAX_WS_SEClient выбором опции **Выполнить**, в окне **Вывод** среды NetBeans можно увидеть строку, возвращаемую методом `getHello()` Web-сервиса JAXWSExample.

Для того чтобы Proxy-клиент мог асинхронно взаимодействовать с Web-сервисом, в окне **Проекты** щелкнем правой кнопкой мыши на узле **Ссылки на веб-службы | JAXWSExampleService** проекта JAX_WS_SEClient и выберем пункт **Правка атрибутов веб-службы**. В появившемся диалоговом окне на вкладке **Настройка WSDL** выберем опцию **Глобальная настройка**, укажем **Включить асинхронный клиент** и нажмем кнопку **OK**.

На компакт-диске

Измененный проект JAX_WS_SEClient находится в папке Примеры\Глава3\JAX_WS_NetBeans\Async компакт-диска.

В результате будет сгенерирован SEI-интерфейс, содержащий, помимо метода `public String getHello`, методы `public Response<GetHelloResponse> getHelloAsync` и `public Future<?> getHelloAsync`.

Для использования асинхронных методов в окне редактора в коде класса `Main` щелкнем правой кнопкой мыши и выберем пункты **Вставить код | Вызов операции веб-службы | JAX_WS_WebClient | JAXWSExampleService | JAXWSExamplePort | getHello[Асинхронный опрос]** и **getHello[Асинхронный обратный вызов]**, нажмем кнопку **OK**. В итоге класс `Main` дополнится методами, которые обозна-

чим как `private static void getHelloAsyncPolling()` и `private static void getHelloAsyncCallback()` и изменим код (см. папку Примеры\Глава3\JAX_WS_NetBeans\Async\JAX_WS_SEClient\src\jax_ws_seclient компакт-диска).

Вызвав в методе `main()` методы `getHelloAsyncPolling()` и `getHelloAsyncCallback()` и запустив приложение JAX_WS_SEClient выбором опции **Выполнить**, в окне **Вывод** среды NetBeans можно увидеть строку, асинхронно возвращаемую методом `getHello()` Web-сервиса JAXWSExample.

Для того чтобы продемонстрировать использование параметров типа `OUT` и `INOUT` вызова Web-сервиса, изменим код класса `JAXWSExample` следующим образом:

```
package jaxwsexample;
import javax.jws.*;
import javax.xml.ws.*;

@WebService()
public class JAXWSExample {
/**
  * Операция веб-службы
  */
  @WebMethod(operationName = "getHello")
  public String getHello(@WebParam(name = "name",
                                   mode=WebParam.Mode.INOUT)
  Holder<String> name,@WebParam(name="word", mode=WebParam.Mode.OUT)
                         Holder<String> word) {
    //TODO write your implementation code here:
    word.value="Hello";
    name.value="User";
    return name.value;
}}
```

На компакт-диске

Измененный проект JAX_WS_WebService находится в папке Примеры\Глава3\JAX_WS_NetBeans\Holder компакт-диска.

Для развертывания измененного Web-сервиса в окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebService** и выберем пункт **Очистить и построить**, а затем — пункт **Развернуть**.

Создадим JAX-WS-клиента платформы Java SE для измененного Web-сервиса.

1. Для этого в среде NetBeans в меню **Файл** последовательно выберем пункты **Создать проект | Java | Приложение Java**, нажмем кнопку **Далее**, введем имя проекта JAX_WS_SEClient и нажмем кнопку **Завершить**.

На компакт-диске

Проект JAX_WS_SEClient находится в папке Примеры\Глава3\JAX_WS_NetBeans\Holder компакт-диска.

2. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_SEClient** и выберем пункты **Создать | Другие | Веб-службы | Клиент веб-службы**, нажмем кнопку **Далее**. С помощью кнопки **Обзор** выберем Web-сервис JAXWSExample и нажмем кнопку **OK** и кнопку **Завершить**.
3. В окне редактора в коде класса `Main` щелкнем правой кнопкой мыши и выберем пункты **Вставить код | Вызов операции веб-службы | JAX_WS_WebClient | JAXWSExampleService | JAXWSExamplePort | getHello**, нажмем кнопку **OK**. В результате класс `Main` дополнится методом `getHello()`.
4. Изменим код класса `Main` так, как показано в листинге 3.10.

Листинг 3.10. Код класса `Main`

```
package jax_ws_seclient;
import javax.xml.ws.*;
public class Main {
/**
  * @param args the command line arguments
  */
  public static void main(String[] args) {
    // TODO code application logic here
    Holder<java.lang.String> word=new Holder();
    Holder<java.lang.String> name=new Holder("Dick");
    String str2=getHello(name,word);
    String str1=word.value;
    System.out.println(str1+" "+str2); }

  private static String getHello(javax.xml.ws.Holder<java.lang.String> name,
javax.xml.ws.Holder<java.lang.String> word) {
    jaxwsexample.JAXWSExampleService service =
      new jaxwsexample.JAXWSExampleService();
    jaxwsexample.JAXWSExample port = service.getJAXWSExamplePort();
    return port.getHello(name, word);
}}
```

После запуска приложения JAX_WS_SEClient выбором опции **Выполнить** в окне **Вывод** среды NetBeans можно увидеть строку "Hello User".

До сих пор JAX-WS-реализация скрывала все детали использования SOAP-протокола. Однако получить доступ к SOAP-сообщениям перед их демаршализацией или после их маршализации позволяют обработчики сообщений.

Создадим обработчики сообщений для Web-сервиса и его клиента.

1. Для этого в окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebService** и выберем пункты **Создать | Другие | Веб-службы | Обработчик сообщений**, нажмем кнопку **Далее**, введем имя обработчика

`ServiceMessageHandler` и имя пакета `servicemessagehandler` и нажмем кнопку **Завершить**.

2. В окне редактора изменим код метода `handleMessage()` класса `ServiceMessageHandler`, как показано в листинге 3.11.

Листинг 3.11. Код метода `handleMessage()`

```
public boolean handleMessage(SOAPMessageContext messageContext) {
  SOAPMessage msg = messageContext.getMessage();
  Boolean outboundProperty = (Boolean) messageContext.get
(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
  if (outboundProperty.booleanValue()) {
    System.out.println("\nOutbound message:");
  } else {
    System.out.println("\nInbound message:");
  }
  try {
    msg.writeTo(System.out);
    System.out.println("");
  } catch (Exception e) {
    System.out.println(e.getMessage());
  }
    return true;
  }
```

3. Для настройки обработчика щелкнем правой кнопкой мыши на узле **Веб-службы | JAXWSExample** и выберем пункт **Настройка обработчиков**. В появившемся диалоговом окне с помощью кнопки **Добавить** выберем обработчик `ServiceMessageHandler` и дважды нажмем кнопку **OK**. В результате в пакет `jaxwsexample` добавится конфигурационный файл JAXWSExample_handler.xml.
4. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebService** и выберем пункт **Очистить и построить**, а затем — **Развернуть**.

После запуска приложения JAX_WS_SEClient выбором опции **Выполнить** в появившемся окне **Вывод** на вкладке **GlassFish Server 3** среды NetBeans можно будет увидеть входящее и исходящее SOAP-сообщения Web-сервиса:

```
INFO: Inbound message:
INFO: <S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header/><S:Body><ns2:
getHello
xmlns:ns2="http://jaxwsexample/"><name>Dick</name></ns2:getHello></S:Body></S:E
nvelope>
INFO: Outbound message:
INFO: <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body><ns2:getHelloResponse xmlns:ns2="http://jaxwsexample/">
<return>User</return><name>User</name><word>Hello</word></ns2:getHelloResponse>
</S:Body></S:Envelope>
```

Для создания обработчика клиентских SOAP-сообщений:

1. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_SEClient** и выберем пункты **Создать | Другие | Веб-службы | Обработчик сообщений**, нажмем кнопку **Далее**, введем имя обработчика `ClientMessageHandler` и имя пакета `clientmessagehandler` и нажмем кнопку **Завершить**.
2. Изменим код метода `handleMessage()` класса `ClientMessageHandler`, как показано в листинге 3.11.

Для настройки клиентского обработчика щелкнем правой кнопкой мыши на узле **Ссылки на веб-службы | JAXWSExampleService** и выберем пункт **Настройка обработчиков**. В появившемся диалоговом окне, нажав кнопку **Добавить**, выберем обработчик `ClientMessageHandler` и дважды нажмем кнопку **OK**. В результате после запуска приложения JAX_WS_SEClient выбором опции **Выполнить** в окне **Вывод** на вкладке **run** среды NetBeans можно будет увидеть входящее и исходящее SOAP-сообщения клиента Web-сервиса:

```
Outbound message:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:
getHello xmlns:ns2="http://jaxwsexample/"><name>Dick</name>
</ns2:getHello></S:Body></S:Envelope>


Inbound message:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/><S:Body><ns2:getHelloResponse
xmlns:ns2="http://jaxwsexample/"><return>User</return><name>User</name><word>
Hello</word></ns2:getHelloResponse></S:Body></S:Envelope>
Hello User
```

На компакт-диске

Проекты JAX_WS_WebService и JAX_WS_SEClient, использующие обработчики сообщений, находятся в папке Примеры\Глава3\JAX_WS_NetBeans\Handler компакт-диска.

Для настройки клиентского обработчика программным способом необходимо изменить код метода `getHello()` класса `Main` следующим образом:

```
private static String getHello(javax.xml.ws.Holder<java.lang.String> name,
javax.xml.ws.Holder<java.lang.String> word) {
  jaxwsexample.JAXWSExampleService service =
    new jaxwsexample.JAXWSExampleService();
  service.setHandlerResolver(new javax.xml.ws.handler.HandlerResolver() {
    public java.util.List<javax.xml.ws.handler.Handler>
getHandlerChain(javax.xml.ws.handler.PortInfo info) {
      java.util.List<javax.xml.ws.handler.Handler> handlerList =
        new java.util.ArrayList<javax.xml.ws.handler.Handler>();
      javax.xml.ws.handler.Handler handler =
        new clientmessagehandler.ClientMessageHandler();
      handlerList.add(handler);
      return handlerList;
  }});
```

```
  jaxwsexample.JAXWSExample port = service.getJAXWSExamplePort();
  return port.getHello(name, word);
}
```

Рассмотрим обеспечение безопасности для JAX-WS Web-сервисов на примере реализации базовой аутентификации клиента Web-сервиса.

1. Откроем проект JAX_WS_WebService и изменим дескриптор развертывания web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
<security-constraint>
  <display-name>SecurityConstraint</display-name>
  <web-resource-collection>
    <web-resource-name>JAX_WS_WebService</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Client</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>file</realm-name>
  </login-config>
</web-app>
```

2. Изменим дескриптор развертывания sun-web.xml приложения JAX_WS_WebService, связав роль Client с пользователем User сервера приложений:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD GlassFish
Application Server 3.0 Servlet 3.0//EN"
"http://www.sun.com/software/appserver/dtds/sun-web-app_3_0-0.dtd">
<sun-web-app error-url="">
  <context-root>/JAX_WS_WebService</context-root>
  <class-loader delegate="true"/>
```

```
  <security-role-mapping>
    <role-name>Client</role-name>
    <principal-name>User</principal-name>
  </security-role-mapping>
  <jsp-config>
    <property name="keepgenerated" value="true">
    <description>Keep a copy of the generated servlet class' java
code.</description>
    </property>
  </jsp-config>
</sun-web-app>
```

3. В окне **Службы** щелкнем правой кнопкой мыши на узле **GlassFish Server 3** и выберем пункты **Запустить | Просмотр консоли администратора**.
4. На странице консоли администратора выберем узел **Security | Realms | file**, нажмем кнопку **Manage Users | New**, введем имя пользователя `User` и его пароль `jaxwsuser` и нажмем кнопку **OK**.
5. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_WebService** и выберем пункты **Очистить и построить | Развернуть**.

   Теперь при запуске приложения JAX_WS_SEClient в консоль будет выводиться сообщение об ошибке:

```
Exception in thread "main"
com.sun.xml.internal.ws.client.ClientTransportException: The server sent
HTTP status code 401: Unauthorized
```

6. Откроем проект JAX_WS_SEClient и изменим код метода `getHello()` класса `Main`:

```
private static String getHello(javax.xml.ws.Holder<java.lang.String> name,
javax.xml.ws.Holder<java.lang.String> word) {
  jaxwsexample.JAXWSExampleService service =
    new jaxwsexample.JAXWSExampleService();
  service.setHandlerResolver(new javax.xml.ws.handler.HandlerResolver() {
    public java.util.List<javax.xml.ws.handler.Handler>
getHandlerChain(javax.xml.ws.handler.PortInfo info) {
      java.util.List<javax.xml.ws.handler.Handler> handlerList =
        new java.util.ArrayList<javax.xml.ws.handler.Handler>();
      javax.xml.ws.handler.Handler handler =
        new clientmessagehandler.ClientMessageHandler();
      handlerList.add(handler);
      return handlerList;
}});
  jaxwsexample.JAXWSExample port = service.getJAXWSExamplePort();
  BindingProvider provider = (BindingProvider) port;
  provider.getRequestContext().put(BindingProvider.USERNAME_PROPERTY,
                                   "User");
```

```
    provider.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY,
                                     "jaxwsuser");
    return port.getHello(name, word);
  }
```

7. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_SEClient** и выберем пункты **Очистить и построить | Выполнить**. В результате в окне **Вывод** на вкладке **run** среды NetBeans можно будет увидеть входящее и исходящее SOAP-сообщения клиента Web-сервиса.

На компакт-диске

Проекты JAX_WS_WebService и JAX_WS_SEClient, использующие базовую аутентификацию-авторизацию, находятся в папке Примеры\Глава3\JAX_WS_NetBeans\Security компакт-диска.

До сих пор в качестве JAX-WS клиента рассматривался Proxy-клиент, а в качестве JAX-WS Web-сервиса — Web-сервис, ориентированный на вызов удаленных процедур. Далее рассмотрим создание JAX-WS Provider-сервиса, ориентированного на сообщения, и JAX-WS Dispatch-клиента.

Для создания Provider-сервиса возьмем WSDL-документ и XML-схему из приложения JAX_WS_WebService и модифицируем их так, как показано в листинге 3.12.

Листинг 3.12. WSDL-описание Provider-сервиса JAXWSProviderImpService.wsdl

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns=
    "http://jaxwsprovider/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://jaxwsprovider/"
  name="JAXWSProviderImpService">

<types>
  <xs:schema targetNamespace="http://jaxwsprovider/">
    <xs:element name="getHello" type="tns:getHello" />
    <xs:element name="getHelloResponse" type="tns:getHelloResponse" />
    <xs:complexType name="getHello">
      <xs:sequence>
        <xs:element name="name" type="xs:string" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="getHelloResponse">
      <xs:sequence>
        <xs:element name="return" type="xs:string" minOccurs="0" />
```

```
        <xs:element name="name" type="xs:string" minOccurs="0" />
        <xs:element name="word" type="xs:string" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
</types>
<message name="getHello">
  <part name="parameters" element="tns:getHello" />
</message>
<message name="getHelloResponse">
  <part name="parameters" element="tns:getHelloResponse" />
</message>
<portType name="JAXWSProvider">
  <operation name="getHello">
    <input
       wsam:Action="http://jaxwsprovider/JAXWSProvider/getHelloRequest"
       message="tns:getHello" />
    <output
       wsam:Action="http://jaxwsprovider/JAXWSProvider/getHelloResponse"
       message="tns:getHelloResponse" />
  </operation>
</portType>
<binding name="JAXWSProviderPortBinding" type="tns:JAXWSProvider">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
              style="document" />
<operation name="getHello">
  <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
</operation>
</binding>
  <service name="JAXWSProviderImpService">
    <port name="JAXWSProviderPort" binding="tns:JAXWSProviderPortBinding">
      <soap:address
location="http://localhost:8080/JAX_WS_ProviderService/JAXWSProviderImpService"
/>
    </port>
  </service>
</definitions>
```

После создания WSDL-описания будущего Web-сервиса:

1. Откроем среду NetBeans и в меню **Файл** выберем пункты **Создать проект | Java Web | Веб-приложение**, нажмем кнопку **Далее**, введем имя проекта

JAX_WS_ProviderService и нажмем кнопку **Далее**. В раскрывающемся списке **Сервер** выберем **GlassFish Server 3** и нажмем кнопку **Завершить**.

На компакт-диске

Проект JAX_WS_ProviderService находится в папке Примеры\Глава3\JAX_WS_NetBeans компакт-диска.

2. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_ ProviderService** и выберем пункты **Создать | Другие | Веб-службы | Веб-служба на основе WSDL**, нажмем кнопку **Далее**, введем имя Web-сервиса JAXWSProviderImp и имя пакета `jaxwsprovider`, кнопкой **Обзор** определим созданный ранее WSDL-документ JAXWSProviderImpService.wsdl, выберем опцию **Использовать провайдер** и нажмем кнопку **Завершить**.

В результате средой NetBeans из WSDL-документа будет сгенерирован класс `jaxwsprovider.JAXWSProviderImp`, промаркированный аннотацией `@WebServiceProvider` и реализующий интерфейс `javax.xml.ws.Provider<javax.xml.transform.Source>`.

Изменим код класса `JAXWSProviderImp` так, как показано в листинге 3.13.

Листинг 3.13. Код класса `JAXWSProviderImp`

```
package jaxwsprovider;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.dom.DOMResult;
import javax.xml.soap.*;
import javax.xml.ws.*;

@ServiceMode(value = javax.xml.ws.Service.Mode.PAYLOAD)
@WebServiceProvider(serviceName = "JAXWSProviderImpService",
                    portName = "JAXWSProviderPort",
                    targetNamespace = "http://jaxwsprovider/",
wsdlLocation = "WEB-INF/wsdl/JAXWSProviderImp/JAXWSProviderImpService.wsdl")
public class JAXWSProviderImp implements
javax.xml.ws.Provider<javax.xml.transform.Source> {
private String Name;
    @Override
public javax.xml.transform.Source invoke(javax.xml.transform.Source source) {
  try {
    Transformer tr = TransformerFactory.newInstance().newTransformer();
    //tr.transform(source, new StreamResult(System.out));
    DOMResult DomSource=new DOMResult();
    tr.transform(source,DomSource);
    org.w3c.dom.Node root=DomSource.getNode();
    org.w3c.dom.Node payload=root.getFirstChild();
    org.w3c.dom.NodeList list=payload.getChildNodes();
```

```
    for(int i = 0; i < list.getLength(); i++) {
      if (list.item(i).getLocalName().equals("name")){
        if (list.item(i).hasChildNodes()){
          String text=list.item(i).getFirstChild().getNodeValue();
          Name=text;
    }}}
    MessageFactory mf = MessageFactory.newInstance();
    SOAPMessage message = mf.createMessage();
    SOAPPart soapPart = message.getSOAPPart();
    SOAPEnvelope envelope = soapPart.getEnvelope();
    SOAPBody body = envelope.getBody();
    SOAPElement bodyElement =
      body.addChildElement(envelope.createName("getHelloResponse",
        "ns2", "http://jaxwsprovider"));
    bodyElement.addChildElement("return").addTextNode(Name);
    bodyElement.addChildElement("name").addTextNode(Name);
    bodyElement.addChildElement("word").addTextNode("Hello");
    message.saveChanges();
    message.writeTo(System.out);
    return soapPart.getContent();
  } catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
  return null;
}
}
```

В переопределенном методе `invoke()` класса `JAXWSProviderImp` входящее сообщение в виде объекта `javax.xml.transform.Source` трансформируется в объект `javax.xml.transform.dom.DOMResult`, из которого затем извлекается значение XML-элемента `<name>` входящего сообщения.

Полученное значение используется для формирования ответного SOAP-сообщения, которое в итоге трансформируется в возвращаемый объект `Source` методом `javax.xml.soap.SOAPPart.getContent()`.

1. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_ProviderService** и выберем пункты **Очистить и построить | Развернуть**.
2. Для создания Dispatch-клиента в меню **Файл** выберем пункты **Создать проект | Java | Приложение Java**, нажмем кнопку **Далее**, введем имя проекта JAX_WS_DispatchClient и нажмем кнопку **Завершить**.

   На компакт-диске

   Проект JAX_WS_DispatchClient находится в папке Примеры\Глава3\JAX_WS_NetBeans компакт-диска.

3. В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_DispatchClient** и выберем пункты **Создать | Другие | Веб-службы | Клиент веб-**

службы, нажмем кнопку **Далее**, кнопкой **Обзор** выберем Web-сервис JAXWSProviderImpService, укажем опцию **Создать код диспетчеризации** и нажмем кнопку **Завершить**.

4. В окне редактора в коде класса `Main` щелкнем правой кнопкой мыши и выберем пункты **Вставить код | Вызов операции веб-службы**, выберем операцию **getHello** и нажмем кнопку **OK**.
5. Изменим код класса `Main` так, как показано в листинге 3.14.

Листинг 3.14. Код класса `Main`

```
package jax_ws_dispatchclient;

import javax.xml.namespace.QName;
import javax.xml.transform.Source;
import javax.xml.ws.Dispatch;
import javax.xml.transform.stream.StreamSource;
import javax.xml.ws.Service;
import java.io.StringReader;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;

public class Main {

private static void requestService(){
jaxwsprovider.JAXWSProviderImpService service =
  new jaxwsprovider.JAXWSProviderImpService();
  QName portQName =
    new QName("http://jaxwsprovider/", "JAXWSProviderPort");
String req ="<getHello  xmlns=\"http://jaxwsprovider/\">
<name>User</name></getHello>";

try { // Call Web Service Operation
      Dispatch<Source> sourceDispatch = null;
      sourceDispatch = service.createDispatch(portQName,
        Source.class, Service.Mode.PAYLOAD);
      Source result = sourceDispatch.invoke(new StreamSource(new
        StringReader(req)));
      Transformer tr = TransformerFactory.newInstance().newTransformer();
      tr.transform(result, new StreamResult(System.out));
      System.out.println("/n");
    } catch (Exception ex) {
      // TODO handle custom exceptions here
      System.out.println(ex.getMessage());
    }
  }
```

```
  public static void main(String[] args) {
    // TODO code application logic here
    requestService();
  }
}
```

В классе `Main` из сгенерированного средой NetBeans кода создан метод `requestService()`, вызываемый в методе `main()` класса `Main`.

В методе `requestService()` формируется исходящее сообщение, отправляемое Web-сервису методом `javax.xml.ws. Dispatch<T>.invoke`.

Возвращаемое методом `invoke()` ответное от Web-сервиса сообщение в виде объекта `javax.xml.transform.Source` трансформируется в объект `javax.xml.transform.stream.StreamResult`, содержимое которого выводится в консоль.

В окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_WS_ DispatchClient** и выберем пункты **Очистить и построить | Выполнить**.

В результате в окне **Вывод** на вкладке **run** среды NetBeans можно будет увидеть ответное сообщение Web-сервиса.

По умолчанию среда NetBeans использует в качестве среды выполнения реализацию технологии Web-сервисов Metro (**http://metro.java.net/**). Однако среда NetBeans обеспечивает поддержку и для создания Apache Axis2 Web-сервисов (**http://axis.apache.org/axis2/java/core/**).

Среда выполнения Apache Axis2 доступна для скачивания по адресу **http://axis.apache.org/axis2/java/core/download.cgi**.

Для установки среды Apache Axis2 в контейнере Apache Tomcat вместе со средой NetBeans установим не только сервер GlassFish v3, но и сервер Apache Tomcat.

1. Откроем среду NetBeans и в окне **Службы** щелкнем правой кнопкой мыши на узле **Серверы | Apache Tomcat**, выберем пункты **Запуск | Остановить | Свойства** и в появившемся диалоговом окне посмотрим, где находится **Базовая папка Catalina**.
2. Скопируем предварительно скачанный файл axis2.war в папку \apache-tomcat-xxx_base\webapps.
3. В окне **Службы** щелкнем правой кнопкой мыши на узле **Серверы | Apache Tomcat**, выберем пункт **Запуск**.

В результате среда выполнения Apache Axis2 будет развернута в контейнере Apache Tomcat.

1. В меню **Файл** среды NetBeans выберем пункты **Создать проект | Java | Библиотека классов Java**, нажмем кнопку **Далее**, введем имя проекта Axis2_WebService и нажмем кнопку **Завершить**.

На компакт-диске

Проект Axis2_WebService находится в папке Примеры\Глава3\Axis2NetBeans компакт-диска.

2. В меню **Сервис** выберем пункт **Плагины**, на вкладке **Доступные плагины** найдем и выберем **Axis2 Support** и нажмем кнопку **Установить**.
3. В меню **Сервис** выберем пункт **Настройки**, откроем вкладку **Axis2** и, нажав кнопку **Browse**, определим каталог \apache-tomcat-xxx_base\webapps\axis2.
4. Убедимся, что в строке **Axis2 URL** (**http://localhost:8084/axis2**) порт совпадает с портом, указанным в диалоговом окне свойств сервера Apache Tomcat. Укажем **Use Tomcat Manager for Deployment**, введем имя пользователя `ide` и пароль, указанный в диалоговом окне свойств сервера Apache Tomcat, и нажмем кнопку **OK**.
5. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Axis2_WebService** и выберем пункты **Создать | Другие | Веб-службы | Axis2 Service from Java**, дважды нажмем кнопку **Далее**. Введем имя Web-сервиса Axis2Example и имя пакета `axis2sexample` и нажмем кнопку **Завершить**.
6. В окне **Службы** запустим сервер Apache Tomcat, в окне **Проекты** щелкнем правой кнопкой мыши на узле **Axis2_WebService | Axis2 Web Service | Axis2Example** и выберем **Deploy to Server**, щелкнем правой кнопкой мыши на узле **hello: String** и выберем пункт **Test Operation in Browser**.

В результате в окне браузера будет показано ответное сообщение от Web-сервиса.

Рассмотрим создание Apache Axis2 Web-сервисов с использованием среды Eclipse IDE for Java EE Developers (**http://www.eclipse.org/downloads/**).

Для создания Web-сервиса и его клиента используем среду выполнения Apache Axis2, доступную для скачивания по адресу **http://axis.apache.org/axis2/java/core/download.cgi**.

1. Откроем среду Eclipse, в окне **Servers** щелкнем правой кнопкой мыши и выберем пункты **New | Server**, выберем **Tomcat v7.0 Server** и нажмем кнопку **Next**. Нажав кнопку **Installed JREs**, определим установленный набор JDK 6, в строке **JRE** укажем добавленный JDK 6, с помощью кнопки **Browse** укажем каталог предварительно установленного сервера Apache Tomcat v7.0 (**http://tomcat.apache.org/download-70.cgi**) и нажмем кнопку **Finish**.

   В результате сервер Apache Tomcat v7.0 будет добавлен в среду Eclipse в качестве локального сервера.
2. В меню **File** выберем пункты **New | Dynamic Web Project** и введем имя проекта Axis2_WebService. С помощью кнопки **New Runtime** выберем **Apache Tomcat v7.0**, в строке **Dynamic web module version** укажем **2.5** и нажмем кнопку **Finish**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebService** и выберем пункты **New | Class**, введем имя Web-сервиса Axis2Example и имя пакета `axis2sexample` и нажмем кнопку **Finish**.
4. Дополним код класса `Axis2Example` методом:

```
public String getHello(String name) {
  return "Hello"+" "+ name;
}
```

5. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Axis2 Preferences**, на вкладке **Axis2 Runtime**, нажав кнопку **Browse**, определим каталог предварительно инсталлированной среды выполнения Apache Axis2 (Binary Distribution, **http://axis.apache.org/axis2/java/core/download.cgi**).

6. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Server and Runtime**, в раскрывающемся списке **Server runtime** выберем Tomcat v7.0 Server, в списке **Web service runtime** — **Apache Axis2** и нажмем кнопку **OK**.

7. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Axis2 Web Services** и нажмем кнопку **OK**.

8. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebService** и выберем пункты **New | Other | Web Services | Web Service**, нажмем кнопку **Next** и в строке **Service implementation** введем `axis2sexample.Axis2Example`.

9. Перейдем по ссылке **Web service runtime: Apache Axis**, выберем Apache Axis2 и нажмем кнопку **OK**.

10. В окне **Web Service** нажмем кнопку **Finish** (рис. 3.16).

11. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebService**, выберем пункты **Run As | Run on Server** и нажмем кнопку **Finish**. В результате в окне браузера появится страница приветствия Axis2, на которой, перейдя по ссылке **Services**, можно увидеть информацию о развернутом Web-сервисе Axis2Example (рис. 3.17).

Для создания клиента Axis2 Web-сервиса:

1. В меню **File** среды Eclipse выберем пункты **New | Dynamic Web Project** и введем имя проекта Axis2_WebServiceClient. Нажав кнопку **New Runtime**, выберем **Apache Tomcat v7.0**, в строке **Dynamic web module version** укажем **2.5** и нажмем кнопку **Finish**.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Axis2 Web Services** и нажмем кнопку **OK**.

3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClient** и выберем пункты **New | Other | Web Services | Web Service Client**, нажмем кнопку **Next**, в строке **Service definition** введем `http://localhost:8080/Axis2_WebService/services/Axis2Example?wsdl` и нажмем кнопку **Next** (рис. 3.18). В раскрывающемся списке **Port Name** выберем конечную точку, например **Axis2ExampleHttpSoap12Endpoint**, отметим флажок **Generate all types for all elements referred to by schemas** и нажмем кнопку **Finish** (рис. 3.19).

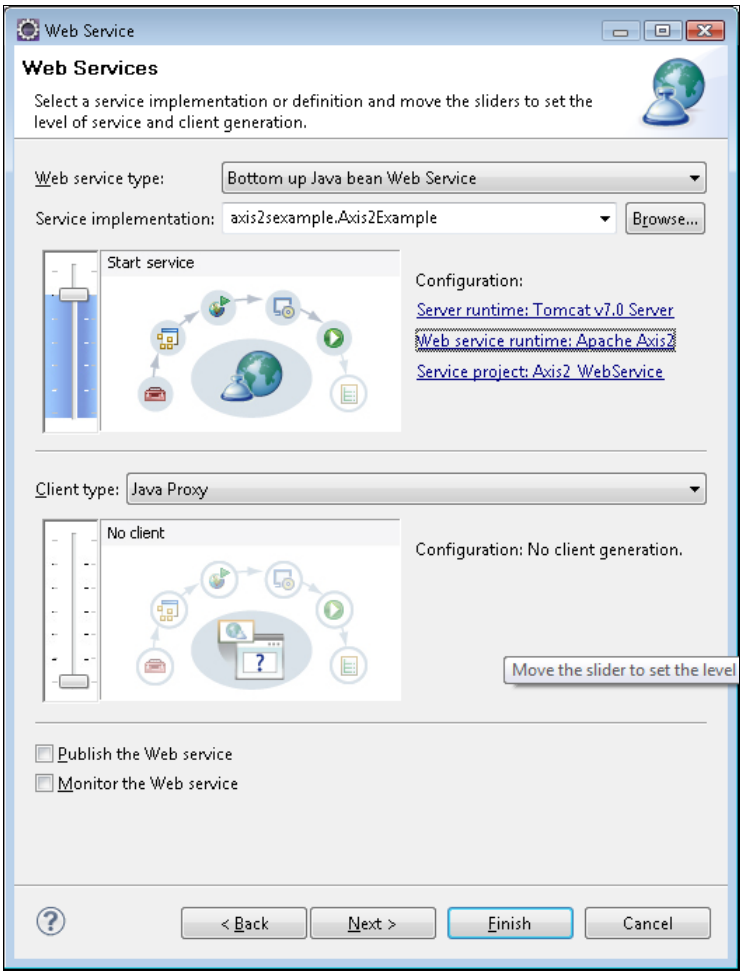


Рис. 3.16. Диалоговое окно создания Web-сервиса

4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClient | Java Resources: src | axis2sexample** и выберем пункты **New | Class**, введем имя класса `Axis2Client` и нажмем кнопку **Finish**.
5. Изменим код класса `Axis2Client` так, как показано в листинге 3.15.

Листинг 3.15. Код класса `Axis2Client`

```
package axis2sexample;
public class Axis2Client {
  public static void main(String[] args) {
    Axis2ExampleStub.GetHello getHello=new Axis2ExampleStub.GetHello();
    getHello.setName("User");
    Axis2ExampleStub.GetHelloResponse res=
      new Axis2ExampleStub.GetHelloResponse();
```

```
        try {
          Axis2ExampleStub stub=new Axis2ExampleStub();
          res=stub.getHello(getHello);
          System.out.println(res.get_return());
        } catch (Exception e) {
          System.out.println(e.getMessage());
}}}
```

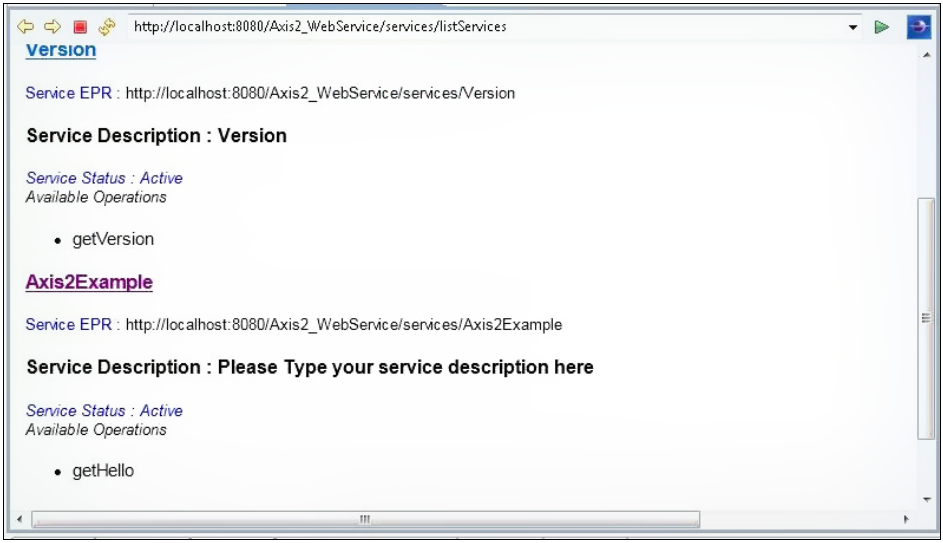


Рис. 3.17. Страница Axis2

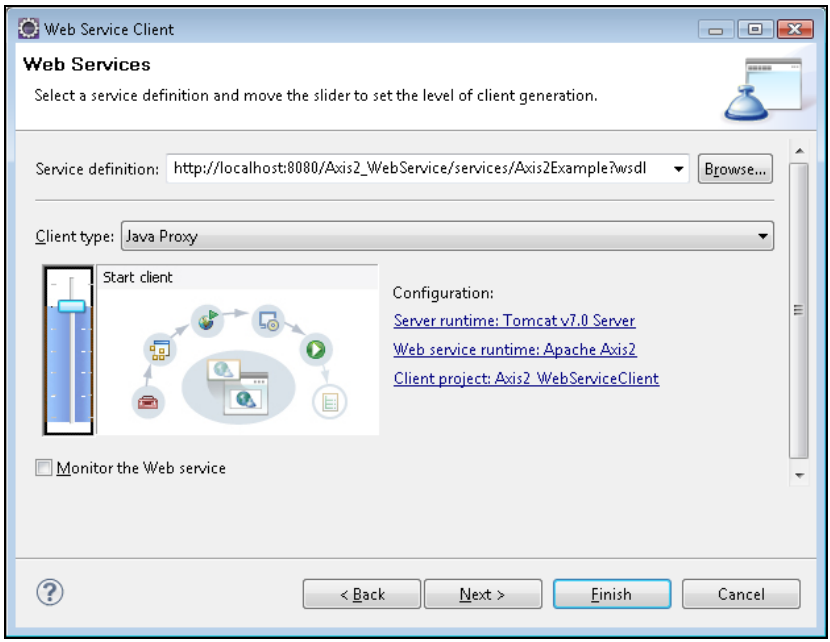


Рис. 3.18. Диалоговое окно создания клиента Web-сервиса Axis2Example: определение сервиса

В методе `main()` класса `Axis2Client` используется сгенерированный JavaBean-компонент `GetHello` для установки параметра запроса `name`. Вызов Web-сервиса обеспечивает объект сгенерированного класса-заглушки `Axis2ExampleStub`, метод `getHello()` которого возвращает ответное сообщение от Web-сервиса, содержащееся в экземпляре JavaBean-компонента `GetHelloResponse`.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClient | Java Resources: src | axis2sexample | Axis2Client** и выберем пункты **Run As | Java Application**. В результате в окно **Console** будет выведено ответное сообщение Web-сервиса "Hello User".

На компакт-диске

Проекты Axis2_WebService и Axis2_WebServiceClient находятся в папке Примеры\Глава3\Axis2Eclipse компакт-диска.

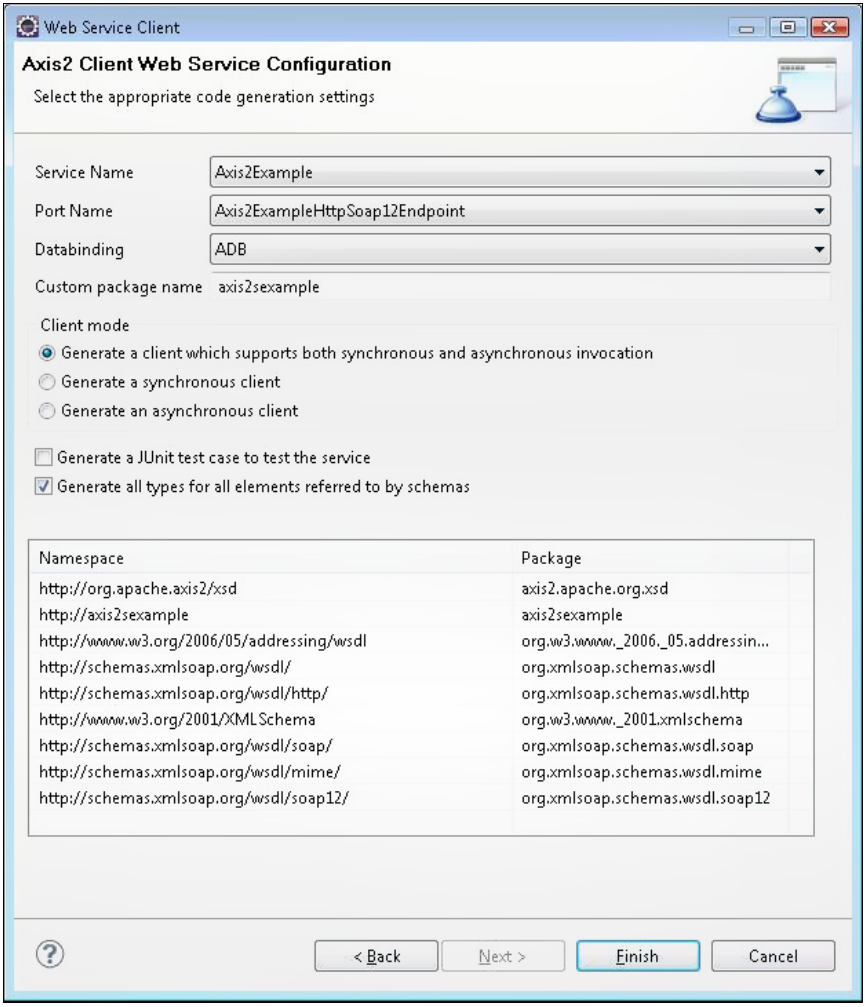


Рис. 3.19. Диалоговое окно создания клиента Web-сервиса Axis2Example: настройка клиента сервиса

Из рассмотренного примера видно, что платформа Axis2 дает возможность разворачивать простой класс Plain Old Java Object (POJO) как Web-сервис. Помимо POJO Web-сервисов платформа Axis2 позволяет разворачивать и аннотированные JAX-WS Web-сервисы. Для создания JAX-WS Web-сервиса на платформе Axis2 достаточно заменить код класса `Axis2Example` на аннотированный код:

```
package axis2sexample;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService()
public class Axis2Example {
  @WebMethod(operationName = "getHello")
  public String getHello(@WebParam(name = "name")
    String name) {
      return "Hello"+" "+ name;
}}
```

Дальше развертывание Web-сервиса ничем не отличается от рассмотренного примера.

Для создания Web-сервиса и его клиента платформа Axis2 обеспечивает полное использование спецификации JAX-WS.

Рассмотрим создание и развертывание Web-сервисов и их клиентов еще на одной платформе — Apache CXF (**http://cxf.apache.org/**).

1. Откроем среду Eclipse, в меню **File** выберем пункты **New | Dynamic Web Project** и введем имя проекта CXF_WebService. С помощью кнопки **New Runtime** выберем **Apache Tomcat v7.0** и укажем **Create a new local server**. Нажмем кнопку **Next**. С помощью кнопки **Browse** определим каталог предварительно инсталлированного сервера Apache Tomcat v7.0 (**http://tomcat.apache.org/**). Нажав кнопку **Installed JREs**, определим каталог JDK 6 и в строке **JRE** укажем JDK 6. Нажмем кнопку **Finish**, в окне **Dynamic Web Project** нажмем кнопку **Finish**.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **CXF 2.x Web Services** и перейдем по ссылке **Further configuration available | Configure installed runtimes**. На вкладке **CXF Runtime**, нажав кнопки **Add** и **Browse**, укажем каталог предварительно инсталлированной среды выполнения Apache CXF (**http://cxf.apache.org/download.html**). Нажмем кнопку **Finish**. На вкладке **CXF Runtime** отметим добавленную среду Apache CXF и трижды нажмем кнопку **OK**.

3. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Server and Runtime**, в раскрывающемся списке **Server runtime** выберем **Tomcat Server v7.0**, в раскрывающемся списке **Web service runtime** укажем **Apache CXF 2.x** и нажмем кнопку **OK**.

4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService** и выберем пункты **New | Class**, введем имя класса `CXFExample` и имя пакета `cxfexample` и нажмем кнопку **Finish**.
5. В редакторе исходного кода дополним код класса `CXFExample` методом `getHello()`:

```
public String getHello(String name) {
  return "Hello"+" "+ name;
}
```

6. Закроем класс `CXFExample`, сохранив изменения. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService** и выберем пункты **New | Other | Web Services | Web Service**, нажмем кнопку **Next** и в строке **Service implementation** введем `cxfexample.CXFExample`. Убедимся, что в ссылке указано **Web service runtime: Apache CXF 2.x**, выставим бегунок в положение **Assemble service** и нажмем кнопку **Finish** (рис. 3.20).

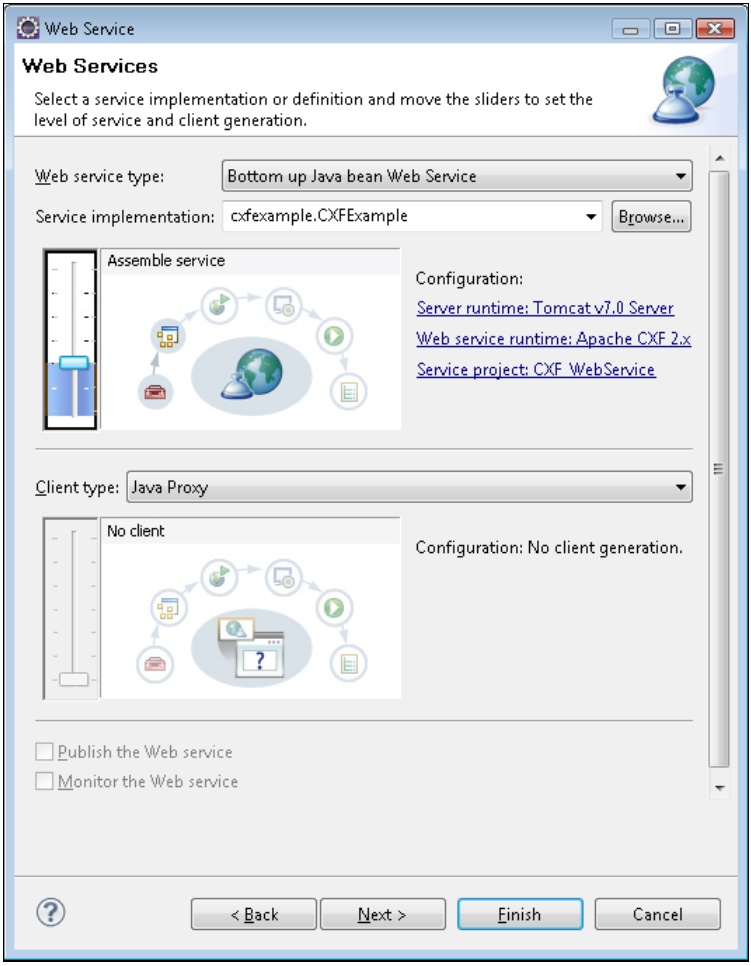


Рис. 3.20. Диалоговое окно создания Web-сервиса CXFExample

В результате класс `CXFExample` будет промаркирован JAX-WS-аннотацией `javax.jws.WebService`, будут сгенерированы JavaBean-классы `cxfexample.jaxws.GetHello` и `cxfexample.jaxws.GetHelloResponse`, а также WSDL-документ cxfexample.wsdl и XML-схема cxfexample_schema1.xsd.

1. Для развертывания Web-сервиса CXFExample в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebService**, выберем пункты **Run As | Run on Server** и нажмем кнопку **Finish**.
2. Для создания клиента CXF Web-сервиса в меню **File** среды Eclipse выберем пункты **New | Dynamic Web Project**, введем имя проекта CXF_WebServiceClient и нажмем кнопку **Finish**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **CXF 2.x Web Services** и нажмем кнопку **OK**.
4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebServiceClient** и выберем пункты **New | Other | Web Services | Web Service Client**, нажмем кнопку **Next**. В строке **Service definition** введем `http://localhost:8080/CXF_WebService/services/CXFExamplePort?wsdl` и нажмем кнопку **Finish**.
5. Откроем узел **CXF_WebServiceClient | Java Resources: src | cxfexample | CXFExample_CXFExamplePort_Client.java** и в редакторе модифицируем код метода `main()`:

   ```
   java.lang.String _getHello_arg0 = "User";
   ```

6. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXF_WebServiceClient | Java Resources: src | cxfexample | CXFExample_CXFExamplePort_Client.java** и выберем пункты **Run As | Java Application**.

В результате в окно **Console** среды Eclipse будет выведено сообщение:

```
Invoking getHello...
getHello.result=Hello User
```

На компакт-диске

Проекты CXF_WebService и CXF_WebServiceClient находятся в папке Примеры\Глава3 компакт-диска.

JAX-RS

Целью создания спецификации JAX-RS (Java API for RESTful Web Services) было упрощение разработки Web-сервисов, реализующих архитектуру REST.

Архитектура Representational State Transfer (REST) — это набор ограничений, определяющих стиль архитектуры распределенной системы ресурсов. Система, реали-

зующая архитектуру REST, или RESTful-система, — это система, удовлетворяющая следующим требованиям:

- система является клиент-серверной системой;
- система является Stateless-системой — каждый запрос клиента является независимым от предыдущего;
- система поддерживает клиентское кэширование ответов;
- система способна загружать код по требованию;
- система может быть многоуровневой, т. е. состоящей из клиента, промежуточных серверов и конечного сервера;
- между клиентом и сервером существует универсальный интерфейс, в котором каждый ресурс имеет свой уникальный идентификатор, обеспечивающий доступ к нему для клиента. При этом ресурс отделен от своего представления, возвращаемого клиенту. Представление или формат данных, представляющих состояние ресурса, может быть разным — HTML, XML, JSON и т. д. — и содержит достаточно информации для модификации самого ресурса.

Таким образом, Web-сервисы RESTful — это Web-сервисы, представляющие удаленные ресурсы, доступные для клиента с помощью HTTP-методов GET, PUT, POST и DELETE. В *разд. "JAX-WS" ранее в этой главе* было показано, что Web-сервисы RESTful могут быть созданы с помощью низкоуровневого интерфейса `javax.xml.ws.Provider<T>`. Задача спецификации JAX-RS — упростить разработку RESTful Web-сервисов для развертывания их на платформе Java EE.

JAX-RS API

Программный интерфейс JAX-RS API состоит из следующих пакетов:

- `javax.ws.rs` содержит аннотации верхнего уровня для создания JAX-RS-приложений;
- `javax.ws.rs.core` обеспечивает низкоуровневый API для создания JAX-RS-приложений;
- `javax.ws.rs.ext` обеспечивает связывание между представлениями ресурса и Java-типами.

Программный интерфейс JAX-RS API подробно рассмотрен в приложении "JAX-RS API" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

Модель программирования и развертывания JAX-RS Web-сервисов

JAX-RS-приложение состоит из одного или нескольких ресурсов (resources) и может включать в себя поставщиков (providers).

Ресурсы и поставщики конфигурируются классом, расширяющим абстрактный класс `javax.ws.rs.core.Application`. Этот класс содержит переопределяемый метод

getClasses(), который возвращает набор классов ресурсов и поставщиков. Класс реализации Application может быть промаркирован аннотацией @ApplicationPath.

При развертывании в контейнере Java EE JAX-RS-приложение упаковывается в качестве сервлета в WAR-файл. Класс Application, классы ресурсов и поставщиков упаковываются в папку WEB-INF/classes, необходимые библиотеки — в папку WEB-INF/lib. При этом класс реализации Application указывается в дескрипторе развертывания web.xml JAX-RS-приложения (на примере JAX-RS реализации Jersey):

```
<web-app>
  <servlet>
    <servlet-name>имя приложения</servlet-name>
    <servlet-class> com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>имя подкласса Application</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name> имя приложения </servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Если класс реализации Application отсутствует, тогда в дескрипторе web.xml имя сервлета указывается как javax.ws.rs.core.Application.

Класс ресурса — это POJO-класс Web-сервиса, использующий аннотации спецификации JAX-RS для обеспечения соответствующего Web-ресурса.

Класс ресурса маркируется аннотацией @Path, а его публичные методы — аннотациями @GET, @PUT, @POST, @DELETE, @HEAD и @OPTIONS. Для инициализации средой выполнения JAX-RS-класс ресурса должен иметь публичный конструктор, который может содержать параметры, маркированные аннотациями @Context, @HeaderParam, @CookieParam, @MatrixParam, @QueryParam или @PathParam.

Поля и свойства класса ресурса также могут маркироваться аннотациями @Context, @HeaderParam, @CookieParam, @MatrixParam, @QueryParam или @PathParam. Аннотация @DefaultValue может быть использована при этом для установки значений по умолчанию. Аннотация @Encoded может использоваться для отключения автоматического декодирования значений, извлекаемых аннотациями @MatrixParam, @QueryParam и @PathParam.

Параметры методов ресурсов могут маркироваться аннотациями @FormParam, @Context, @HeaderParam, @CookieParam, @MatrixParam, @QueryParam или @PathParam, а также аннотациями @DefaultValue и @Encoded.

Метод ресурса может иметь единственный немаркированный параметр, представляющий тело запроса.

Метод ресурса может возвращать `void`, `Response`, `GenericEntity` или другой Java-тип, при этом:

- `void` формирует пустое тело ответа с HTTP-кодом 204;
- `Response` формирует тело ответа своим полем `entity` и имеет свой установленный HTTP-код. Если тело ответа пустое, тогда HTTP-код — 204, если HTTP-код не установлен для `Response`, тогда HTTP-код — 200;
- `GenericEntity` формирует тело ответа своим полем `entity` и имеет HTTP-код 200 для непустого тела и HTTP-код 204 для пустого тела;
- Java-типы формируют тело ответа из экземпляра Java-класса с HTTP-кодом 200 для непустого тела и HTTP-кодом 204 для пустого тела.

Связывание между телом ответа и Java-объектом обеспечивают поставщики Entity Providers.

Если выполнение метода ресурса вызывает исключение `WebApplicationException`, тогда ответ формируется из поля `response` исключения `WebApplicationException`. Если же выполнение метода ресурса вызывает другой тип исключения, тогда ответ формируется из объекта `Response`, создаваемого `ExceptionMapper`-классом поставщика.

В случае клиентского запроса `HEAD` или `OPTIONS` JAX-RS-реализация вызывает метод, маркированный аннотацией `@HEAD` или `@OPTIONS`, или, если такие методы отсутствуют, вызывает метод `@GET` с отбрасыванием тела ответа или автоматически возвращает метаданные соответственно.

Аннотации `@Consumes` и `@Produces`, применяемые к классу ресурса или к его методам, могут ограничивать MIME-типы, потребляемые или производимые ресурсом. При этом в случае несовпадения HTTP-заголовков `Accept` и `Content-Type` со значениями аннотаций `@Produces` и `@Consumes` соответствующие методы ресурса вызываться не будут.

Класс поставщика — это Java-класс, промаркированный аннотацией `@Provider` и реализующий один или несколько интерфейсов `MessageBodyReader<T>`, `MessageBodyWriter<T>`, `ContextResolver<T>` и `ExceptionMapper<E extends Throwable>`.

Для инициализации средой выполнения JAX-RS класс поставщика должен иметь публичный конструктор, который может содержать параметры, маркированные аннотацией `@Context`.

Классы `MessageBodyReader` и `MessageBodyWriter` могут маркироваться аннотациями `@Consumes` и `@Produces` для ограничения MIME-типов.

Формат JSON

Сообщения, участвующие в обмене между клиентом и RESTful Web-сервисом, могут содержать различные представления одного и того же ресурса. Это могут быть данные формата JPG, text, XML, JSON и т. д.

Формат JSON (JavaScript Object Notation) является очень популярным для обмена данными с RESTful Web-сервисами.

MIME-тип формата JSON определен как "application/json".

Формат JSON — это облегченный, по сравнению с XML, текстовый, кросс-платформенный формат обмена данными. Символьная кодировка формата JSON — всегда Unicode.

Формат JSON основан на литералах языка JavaScript, определенных в стандарте ECMAScript Programming Language Standard, Third Edition (ECMA).

Данные в формате JSON представлены двумя структурными типами — объектами и массивами и четырьмя простыми типами — строками, числами, логическими значениями и нулем, где строка — это последовательность Unicode-символов (UTF-8, UTF-16 или UTF-32).

Объект — это неупорядоченная коллекция пар "имя/значение", где имя — строка, а значение — строка, число, `true`/`false`, `null`, объект или массив. Объект ограничивается фигурными скобками (`{}`), имя отделяется от значения двоеточием (`:`), а пары отделяются друг от друга запятыми (`,`).

Массив — это упорядоченная последовательность значений — строк, чисел, `true`/`false`, `null`, объектов или массивов. Массив ограничивается квадратными скобками (`[]`), значения отделяются друг от друга запятыми (`,`).

Согласно соглашению BadgerFish следующие XML-данные:

```
<customer id="[ID]">
  <first>firstName</first>
  <last>lastName</last>
  <phone>number1</phone>
  <phone>number2</phone>
</customer>
```

могут быть представлены как JSON-данные:

```
{ "customer" :
  { "@id" : [ID],
    "first" : { "$" : "firstName"},
    "last" :  { "$" : "lastName" },
    "phone" : [ { "$", "number1"}, { "$", "number2"} ]
  }
}
```

где символ `$` означает пространство имен по умолчанию (в случае объявления пространства имен используется `@xmlns`).

Таким образом, JSON-данные — это комбинация объектов и массивов, которая предназначена для хранения данных и не содержит переменных.

Для преобразования Java-объектов в JSON-данные и обратно на стороне сервера существует большой набор различных Java-библиотек (см. **http://www.json.org/java/index.html**).

Преобразование JSON-данных на стороне клиента легко может быть осуществлено с помощью JavaScript-функции `eval()`, которая, принимая в качестве аргумента JSON-данные, создаст JavaScript-объект:

```
var JSON_object = eval("(" + JSON_data + ")");
```

Структура JSON-документа может быть описана с помощью схемы JSON Schema. JSON Schema для приведенного выше документа будет выглядеть следующим образом:

```
{ "description":"A customer",
  "type":"object",
  "properties":
  { "@id"   : {"type" : "integer", "maximum":125},
    "first" : {"type" : "string"},
    "last"  : {"type" : "string"},
    "phone" : {"type" : "array", "items": {"type":"string"}}
  }
}
```

WADL

Web Application Description Language (WADL) — облегченный, по сравнению с WSDL 2.0, основанный на XML язык описания HTTP Web-приложений, таких как RESTful Web-сервисы.

Язык WADL не так широко распространен, как язык WSDL, его поддержка обеспечивается компанией Sun Microsystems — разработчиком этого языка. Реализация Jersey спецификации JAX-RS (JSR 311) RI от компании Sun Microsystems автоматически генерирует WADL-документ для всего JAX-RS-приложения при его запуске, так что WADL-документ JAX-RS-приложения можно получить с помощью запроса GET по адресу **http://root/application.wadl**. Кроме того, Jersey автоматически генерирует WADL-документы для индивидуальных ресурсов, которые можно получить с помощью запроса `OPTIONS` по адресу ресурса. Среда разработки NetBeans использует WADL-документ для генерации простого тестового HTML-клиента.

Документ WADL может содержать информацию о наборе ресурсов HTTP-приложения, описание связей между ресурсами, методов, применимых к каждому ресурсу, и форматов представления ресурсов.

Пространство имен WADL-элементов определено как `http://wadl.dev.java.net/2009/02`. Структура WADL-докумета описывается XML-схемой `http://wadl.dev.java.net/2009/02 wadl.xsd`.

Корневым элементом документа WADL является элемент `<application>`. Общая схема элемента `<application>` показана на рис. 3.21.

Элемент `<doc>` может быть дочерним элементом любого WADL-элемента и содержать его документацию. Элемент `<doc>` может иметь атрибуты: `xml:lang` — язык документации, и `title` — короткое описание WADL-элемента.

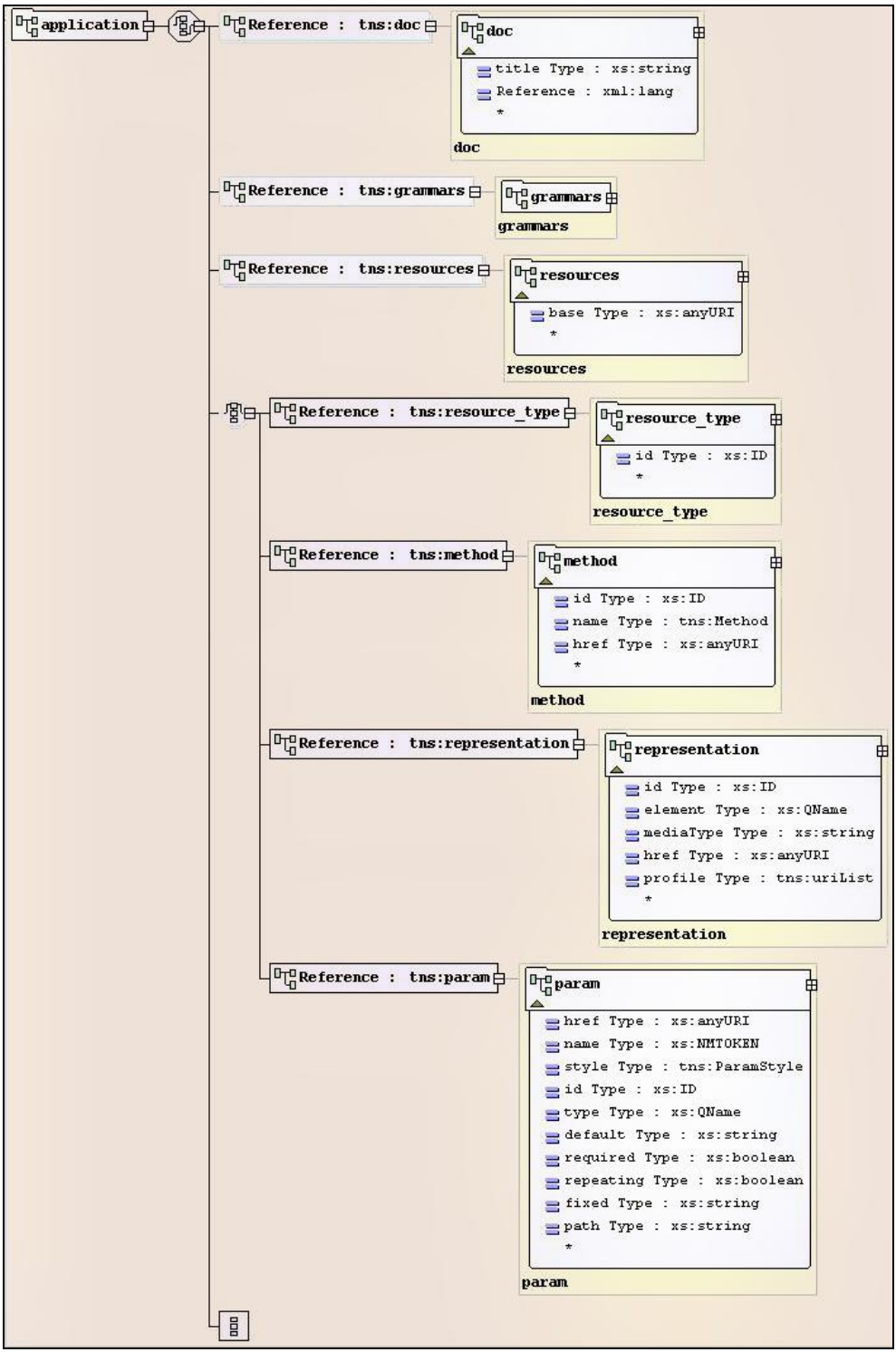


Рис. 3.21. Общая схема элемента `<application>`

Элемент `<grammars>` служит контейнером для определения формата данных, передаваемых между клиентом и HTTP-приложением. Элемент `<grammars>` может содержать непосредственно XML-схемы или включать в себя элементы `<include>`, указывающие URI-адреса XML-схем с помощью атрибута `href`.

Элемент `<resources>` служит контейнером для элементов `<resource>`, описывающих ресурсы HTTP-приложения. Элемент `<resources>` может иметь атрибут `base`, указывающий базовый URI-адрес для URI-адресов ресурсов.

Элемент `<resource>` (рис. 3.22) описывает ресурс HTTP-приложения и может иметь следующие атрибуты:

- `id` — идентификатор элемента;
- `path` — относительный URI-путь ресурса;
- `type` — список ссылок на элементы `<resource_type>`, определяющие методы, поддерживаемые ресурсом;
- `queryType` — MIME-тип HTTP-запроса ресурса, по умолчанию `application/x-www-form-urlencoded`;

Элемент `<resource>` может иметь дочерние элементы `<doc>`, `<param>`, `<method>` и `<resource>`.

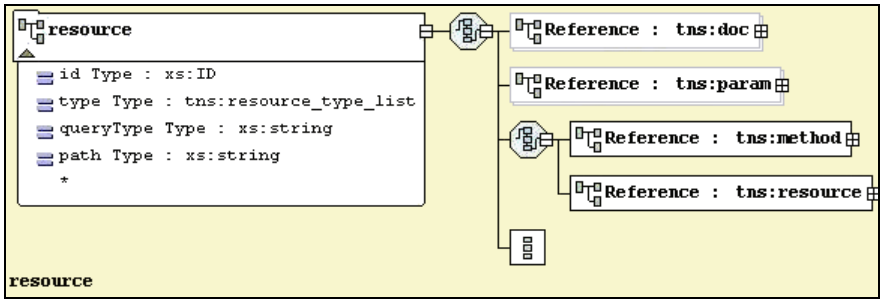


Рис 3.22. Общая схема элемента `<resource>`

Элемент `<param>` описывает параметры HTTP-запроса ресурса и имеет следующие атрибуты и дочерние элементы:

- дополнительный атрибут `href` — ссылка на элемент `<param>`;
- дополнительный атрибут `id` — идентификатор элемента;
- обязательный атрибут `name` — имя параметра как `xsd:NMTOKEN`;
- дополнительный атрибут `style` — тип параметра, возможные значения: `template`, `matrix`, `query`, `header`;
- дополнительный атрибут `type` — тип значения параметра, по умолчанию `xsd:string`;
- дополнительный атрибут `default` — значение по умолчанию параметра;
- дополнительный атрибут `required` — обязательность параметра, по умолчанию `false`;

- дополнительный атрибут `repeating` — параметр может иметь многократные значения (`true`) или единственное значение (`false`, по умолчанию);
- дополнительный атрибут `fixed` — фиксированное значение параметра;
- дополнительные элементы `<option>` указывает возможные значения параметра с помощью атрибутов `value` (обязательный, указывает значение параметра) и `mediaType` (дополнительный, указывает MIME-тип значения параметра).

Элемент `<method>` описывает методы ресурса и имеет следующие атрибуты и дочерние элементы:

- дополнительный атрибут `href` — ссылка на элемент `<method>`;
- дополнительный атрибут `name` — HTTP-метод запроса;
- дополнительный атрибут `id` — идентификатор элемента;
- дополнительный элемент `<request>` описывает входящие параметры метода и представления ресурса. Элемент `<request>` может содержать дочерние элементы `<param>` со значениями атрибута `style`, равными `query` или `header`, и дочерние элементы `<representation>` в случае HTTP-метода PUT или POST. Элемент `<representation>` описывает представление ресурса и имеет следующие атрибуты и дочерние элементы:
  - дополнительный атрибут `href` — ссылка на элемент `<representation>`;
  - дополнительный атрибут `id` — идентификатор элемента;
  - дополнительный атрибут `mediaType` — MIME-тип представления;
  - дополнительный атрибут `element` — корневой элемент XML-представления ресурса;
  - дополнительный атрибут `profile` — расположение документа, содержащего информацию о значениях `rel` и `rev` атрибутов элемента `<link>` *(см. далее)*;
  - дополнительные элементы `<param>` описывают элементы представления. Дочерние элементы `<param>` элементов `<representation>` могут иметь, дополнительно к уже ранее перечисленным: атрибут `style` со значениями `query` (`application/x-www-form-urlencoded` или `multipart/form-data`) или `plain`; атрибут `path` — путь значения параметра в представлении; элемент `<link>` — связи с ресурсами в представлении, которые указываются с помощью дополнительных атрибутов `resource_type` (ссылка на элемент `<resource_type>`), `rel` и `rev` (указывают соотношение элемента представления относительно пути параметра);
- дополнительный элемент `<response>` — результат метода. Этот элемент может содержать атрибут `status` — перечень HTTP-кодов ответа, дочерние элементы `<param>` со значением `header` атрибута `style` и дочерние элементы `<representation>`, описывающие возможные исходящие представления ресурса.

Дочерние элементы `<resource>` элемента `<resource>` описывают подресурсы ресурса.

Элемент `<resource_type>` описывает набор методов, который определяет поведение ресурса, и имеет следующие атрибуты и дочерние элементы:

- обязательный атрибут `id` — идентификатор элемента, обеспечивающий ссылки на него;
- дополнительные элементы `<param>` — параметры HTTP-запроса ресурса со значениями `query` или `header` атрибута `style`;
- дополнительные элементы `<method>` — методы ресурса;
- дополнительные элементы `<resource>` описывают подресурсы ресурса.

Глобальные элементы `<param>`, `<method>` и `<representation>` имеют обязательный атрибут `id`, обеспечивающий ссылки на глобально определенные элементы для их многократного использования.

Применение технологии JAX-RS

Архитектура REST широко используется для создания ресурсов Интернета. Такие интернет-площадки, как Google, Twitter, Flickr и др., предоставляют интерфейс REST API для программного доступа к своим RESTful Web-сервисам.

Наиболее часто RESTful Web-сервисы применяются для обработки клиентских HTTP-запросов к базам данных. Поэтому в качестве примера рассмотрим создание Web-приложения, ориентированного на данные и использующего технологии JAX-RS и JPA.

Рассмотрим разработку JAX-RS приложения с использованием среды NetBeans (**http://netbeans.org/**). Поддержка средой NetBeans разработки JAX-RS приложений основывается на реализации Jersey и продвинутых шаблонах кода.

Для создания JAX-RS-проекта откроем среду NetBeans, в меню **Файл** выберем пункты **Создать проект | Java Web | Веб приложение** и нажмем кнопку **Далее**. Введем имя проекта JAX_RS_Example и последовательно нажмем кнопки **Далее** и **Завершить**.

> ***На компакт-диске***
>
> Проект JAX_RS_Example находится в папке Примеры\Глава3\JAX_RS_NetBeans компакт-диска.

В качестве базы данных для проекта используем сервер базы данных MySQL, которая доступна по адресу **http://www.mysql.com/downloads/**. После инсталляции MySQL в окне **Службы** среды NetBeans щелкнем правой кнопкой мыши на узле **Базы данных** и выберем пункт **Зарегистрировать сервер MySQL**. В появившемся диалоговом окне **Свойства сервера MySQL** на вкладке **Базовые свойства** введем пароль администратора, а на вкладке **Свойства администратора**, нажав кнопки **Обзор**, укажем пути к предварительно инсталлированному приложению администрирования MySQL Workbench (**http://www.mysql.com/downloads/workbench/**), команде запуска `mysqld` (аргументы `-u root`) и команде остановки `mysqladmin` (аргументы `-u root -p[password] shutdown`), расположенных в папке bin каталога сервера

MySQL. Для соединения с базой данных MySQL среда NetBeans содержит JDBC-драйвер MySQL Connector/J, отображаемый в узле **Драйверы**.

Создадим простую базу данных, содержащую две таблицы. Одна таблица `DATA` будет хранить данные, запрашиваемые пользователем, а другая `API_KEY` будет содержать ключи API key, как правило, требуемые для доступа к RESTful Web-сервисам интернет-площадки.

Для создания базы данных щелкнем правой кнопкой мыши на узле **Базы данных | Сервер MySQL** и выберем пункт **Создать базу данных**, введем имя базы данных `RESTExample` и нажмем кнопку **OK**. В результате будет создано новое соединение `jdbc:mysql://localhost:3306/RESTExample`.

Щелкнем правой кнопкой мыши на узле **jdbc:mysql://localhost:3306/RESTExample | restexample | Таблицы** и выберем пункт **Создать Таблицу**, введем имя таблицы `DATA`, добавим столбцы `ID_DATA` (первичный ключ типа `INT`), `NAME` (тип `VARCHAR`) и `FIELD` (тип `BLOB`) и нажмем кнопку **OK**.

Для создания таблицы `API_KEY` также щелкнем правой кнопкой мыши на узле **jdbc:mysql://localhost:3306/RESTExample | restexample | Таблицы** и выберем пункт **Создать Таблицу**, введем имя таблицы `API_KEY`, добавим столбцы `ID_API_KEY` (первичный ключ типа `INT`), `APIKEY` (тип `VARCHAR`) и `SECRETKEY` (тип `VARCHAR`) и нажмем кнопку **OK**. Подразумевается, что при регистрации пользователя на интернет-площадке ему выдается постоянный секретный ключ SECRETKEY, на основе которого пользователь может сгенерировать ключ доступа к сервисам APIKEY.

Заполним созданные таблицы данными, щелкнув правой кнопкой мыши на узле **Таблицы** и выбрав пункт **Выполнить Команду**:

```
INSERT INTO restexample.api_key
VALUES (1, 'f47ac10b-58cc-4372-a567-0e02b2c3d479', 'k89fd89v-45br-4jh7-hk87-
jh76rtj4rnjf');
INSERT INTO restexample.`data`
VALUES (1, 'name_1', 'data_user')
```

Проверить ввод данных можно, щелкнув правой кнопкой мыши на узлах таблиц и выбрав пункт **Посмотреть Данные**.

Для создания JPA-классов, представляющих данные базы данных `RESTExample`, в окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_RS_Example** проекта и выберем пункты **Создать | Классы Сущности из Базы Данных | Источник данных | Новый источник данных | Подключение к базе данных | jdbc:mysql://localhost:3306/RESTExample**, введем JNDI-имя jdbc/RESTExample и нажмем кнопку **OK**. Перенесем таблицы `api_key` и `data` из списка **Доступные Таблицы** в список **Выбранные Таблицы** кнопкой **Добавить >** и нажмем кнопку **Далее**. Введем имя пакета `restexamplemysql` и нажмем кнопку **Завершить**.

На основе созданных JPA-классов создадим теперь JAX-RS Web-сервис. Для этого в окне **Проекты** щелкнем правой кнопкой мыши на узле **JAX_RS_Example** проекта и выберем пункты **Создать | Веб службы RESTful на основе классов сущно-**

стей. Перенесем класс DATA из списка **Доступные классы сущностей** в список **Выбранные классы сущностей** кнопкой **Добавить >** и нажмем кнопку **Завершить**.

Соберем и развернем проект, щелкнув правой кнопкой мыши на узле **JAX_RS_Example** проекта и выбрав пункты **Построить** и **Развернуть**.

Протестировать созданный Web-сервис можно, выбрав **Тестировать веб службы RESTful**, при этом откроется страница браузера, с помощью которой можно осуществить запросы к сервису (рис. 3.23).

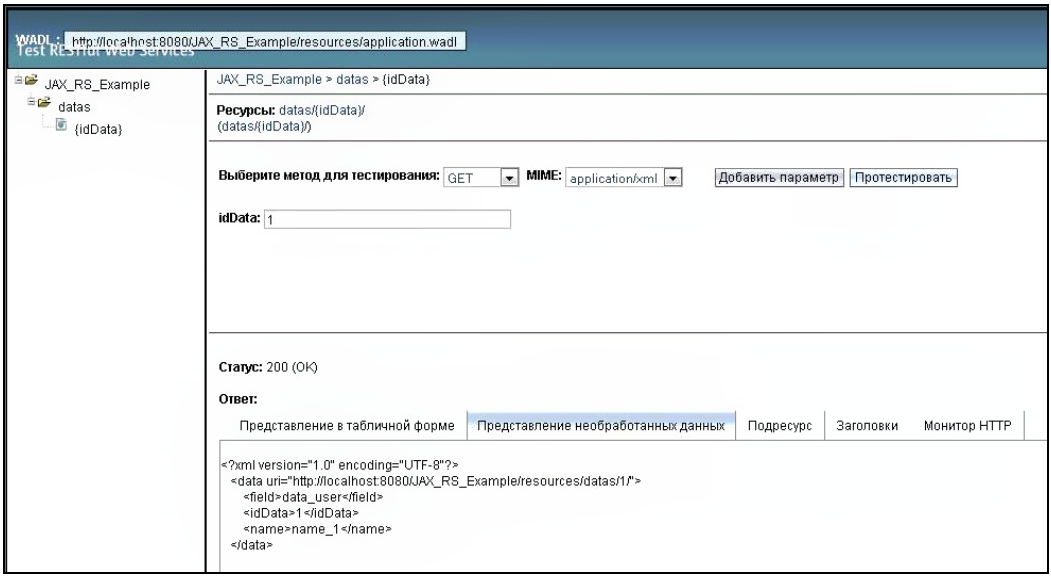


Рис. 3.23. Страница тестирования JAX-RS Web-сервиса

Защитим доступ к Web-сервису DatasResource проверкой ключа API key. Для этого дополним код класса `service.DatasResource` (листинг 3.16).

Листинг 3.16. Код класса `service.DatasResource`

```
package service;

import java.util.Collection;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;
import javax.ws.rs.PathParam;
import javax.ws.rs.QueryParam;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Context;
```

```
import javax.ws.rs.core.UriInfo;
import javax.persistence.EntityManager;
import converter.DatasConverter;
import converter.DataConverter;
import javax.persistence.PersistenceContext;
import javax.ejb.Stateless;
import restexamplemysql.Data;

@Path("/datas/")
@Stateless
public class DatasResource {
  @javax.ejb.EJB
  private DataResource dataResource;
  @Context
  protected UriInfo uriInfo;
  @PersistenceContext(unitName = "JAX_RS_ExamplePU")
  protected EntityManager em;

  public DatasResource() { }

  @GET
  @Produces({"application/xml", "application/json"})
  public DatasConverter get(@QueryParam("api_key")
                            @DefaultValue("") String api_key,
          @QueryParam("start") @DefaultValue("0") int start,
          @QueryParam("max")   @DefaultValue("10") int max,
          @QueryParam("expandLevel") @DefaultValue("1") int expandLevel,
          @QueryParam("query") @DefaultValue("SELECT e FROM Data e")
    String query) throws Exception {
      if(!validate(api_key)){throw new Exception("Missing API key");}
        return new DatasConverter(getEntities(start, max, query),
          uriInfo.getAbsolutePath(), expandLevel);
      }
    @POST
    @Consumes({"application/xml", "application/json"})
    public Response post(DataConverter data, String api_key)
      throws Exception {
      if(!validate(api_key)){throw new Exception("Missing API key");}
      Data entity = data.resolveEntity(em);
      createEntity(data.resolveEntity(em));
      return Response.created(uriInfo.getAbsolutePath().
              resolve(entity.getIdData() + "/")).build();
    }

    @Path("{idData}/")
    public DataResource getDataResource(@QueryParam("api_key")
                            @DefaultValue("") String api_key,
                            @PathParam("idData")
```

```
    Integer id throws Exception {
        if(!validate(api_key)){throw new Exception("Missing API key");}
        dataResource.setId(id);
        dataResource.setEm(em);
        return dataResource;
    }

    protected Collection<Data> getEntities(int start, int max,
      String query) {
      return em.createQuery(query).setFirstResult(start).
        setMaxResults(max).getResultList();
    }
    protected void createEntity(Data entity) {
        em.persist(entity);
    }
    protected boolean validate(String api_key){
      Long d= (Long)em.createQuery("SELECT COUNT(e) FROM ApiKey e WHERE
e.apikey = :apikey").setParameter("apikey",api_key).getSingleResult();
      if(d==0){return false;}
      return true;
    }
}
```

Теперь для получения данных от Web-сервиса требуется включение в запрос правильного ключа API key (рис. 3.24).

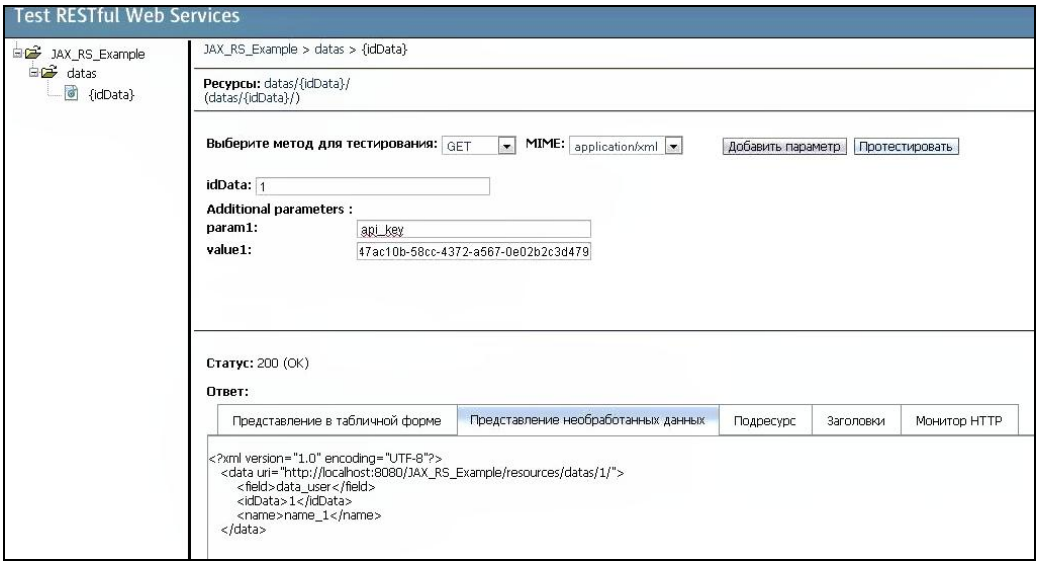


Рис. 3.24. Доступ к Web-сервису с использованием ключа API key

Разработка JAX-RS Web-сервисов с использованием среды Eclipse не так тривиальна, как в среде NetBeans. Для создания JAX-RS-приложения в среде Eclipse IDE for

Java EE Developers (**http://www.eclipse.org/downloads/**) в меню **File** выберем пункты **New | Dynamic Web Project**, введем имя проекта JAX_RS_Example, нажмем кнопку **New Runtime** и перейдем по ссылке **Download additional server adapters**. Выберем **Oracle GlassFish Server Tools**, загрузим и установим выбранный адаптер. Теперь в окне **New Dynamic Web Project** кнопкой **New Runtime** можно выбрать предварительно инсталлированный сервер приложений GlassFish Server Open Source Edition 3 (**http://glassfish.java.net/public/downloadsindex.html#top**). Далее, указав **Create a new local server**, нажмем кнопку **Next** и кнопкой **Browse** определим каталог сервера GlassFish v3. Используя ссылку **Installed JRE preferences**, укажем установленный JDK 6, введем пароль администратора сервера и нажмем кнопку **Finish**. После двухкратного нажатия кнопки **Next** укажем **Generate web.xml deployment descriptor** и нажмем кнопку **Finish**.

На компакт-диске

Проект JAX_RS_Example находится в папке Примеры\Глава3\JAX_RS_Eclipse компакт-диска.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **JAX_RS_Example** проекта и выберем пункт **Properties**. В появившемся диалоговом окне выберем раздел **Project Facets** и установим флажок **JAX-RS (REST Web Services)** (рис. 3.25).

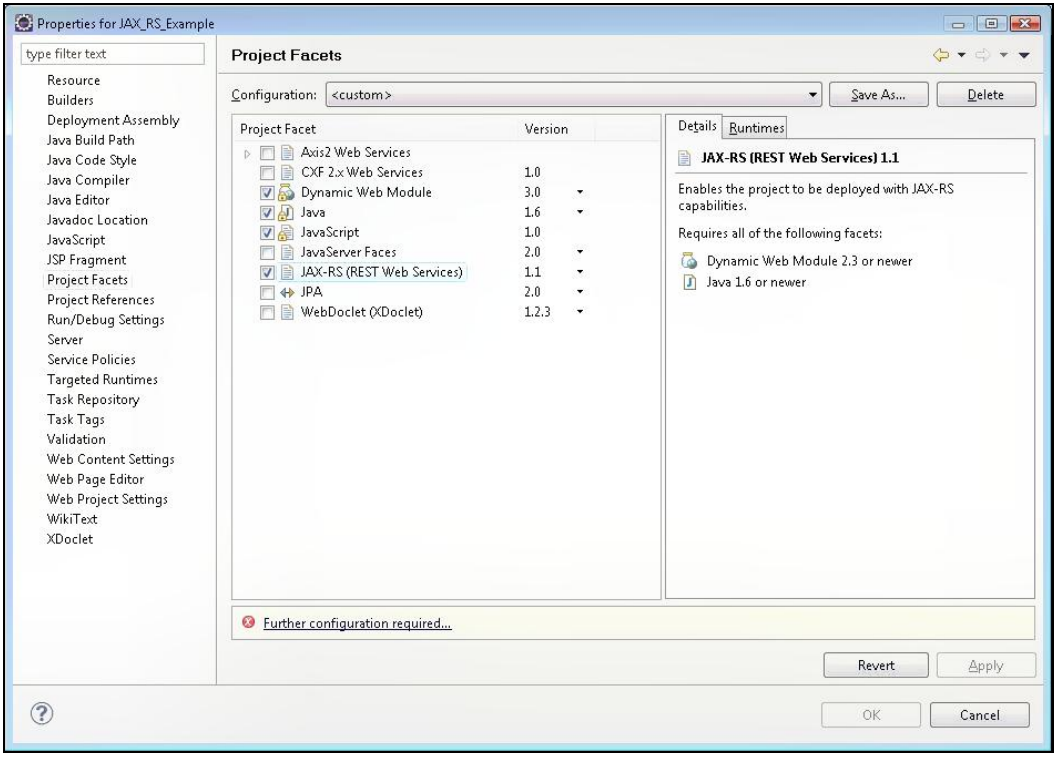


Рис. 3.25. Диалоговое окно определения свойств проекта

Внизу появится ссылка **Futher configuration required**, щелкнув по которой необходимо определить конфигурацию JAX-RS-реализации. Введем имя сервлета JAX_RS_Example, укажем URL-шаблон — `/*` и нажмем кнопку **OK** (рис. 3.26).

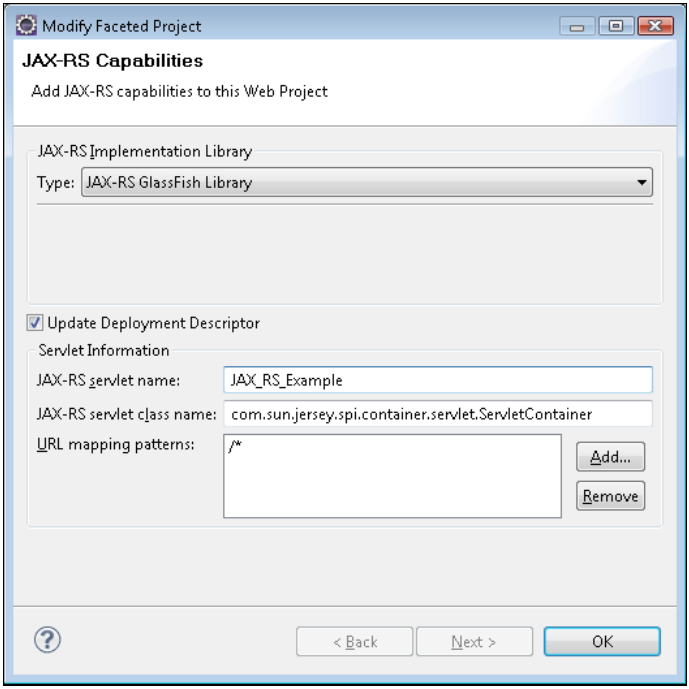


Рис. 3.26. Определение свойств JAX-RS реализации

1. Для использования спецификации JPA в окне **Project Facets** укажем **JPA**, перейдем по ссылке **Futher configuration required**, выберем **Disable Library Configuration**, **Annotated classes must be listed in persistence.xml** и перейдем по ссылке **Add connection**. Выберем **MySQL**, нажмем кнопку **Next** и кнопку **New Driver Definitions**. На вкладке **Name/Type** выберем версию **MySQL JDBC Driver**, а на вкладке **JAR List** укажем предварительно инсталлированный драйвер MySQL Connector/J (**http://www.mysql.com/downloads/connector/j/**) mysql-connector-java-bin.jar. На вкладке **Properties** укажем пароль для соединения с базой данных и нажмем кнопку **OK**. В окне **New Connection Profile** на вкладке **General** укажем URL-адрес соединения с базой данных **jdbc:mysql://localhost:3306/RESTExample**, отдельно запустим сервер MySQL и нажмем кнопку **Test Connection** — в результате должно появиться сообщение "Ping succeded!". Нажмем кнопку **Finish** и кнопку **OK**.
2. Закроем диалоговое окно определения свойств проекта кнопками **Apply** и **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **JAX_RS_Example** проекта и выберем пункты **New | Other | JPA | Entities from Table**, нажмем кнопку **Next** и выберем таблицы `api_key` и `data`, нажмем дважды кнопку **Next**, введем имя пакета `restexamplemysql` и нажмем кнопку **Finish**.

4. Щелкнем правой кнопкой мыши на узле **Java Resources: src** и выберем пункты **New | Package**, введем имя пакета `service` и нажмем кнопку **Finish**. Щелкнем правой кнопкой мыши на узле **service** и выберем пункты **New | Class**, введем имя класса `DatasResource` и нажмем кнопку **Finish**. В окне редактора введем код класса `DatasResource` из NetBeans-проекта. Таким же образом создадим классы `service.DataResource`, `converter.DataConverter`, `converter.DatasConverter` и `converter.UriResolver`.
5. Откроем узел **JPA Content | persistence.xml** проекта и в окне редактора кода изменим файл persistence.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="JAX_RS_ExamplePU" transaction-type="JTA">
    <jta-data-source>jdbc/RESTExample</jta-data-source>
      <class>restexamplemysql.ApiKey</class>
  <class>restexamplemysql.Data</class>
    <properties/>
  </persistence-unit>
</persistence>
```

6. В узле **Java Resources: src** проекта создадим пакет `rest.application.config` и класс `ApplicationConfig` со следующим кодом:

```
package rest.application.config;
import java.util.*;
import service.*;
@javax.ws.rs.ApplicationPath("resources")
public class ApplicationConfig extends javax.ws.rs.core.Application {
  @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(DatasResource.class);
        classes.add(DataResource.class);
        return classes;
    }
}
```

7. Откроем узел **WebContent | WEB-INF | web.xml** проекта и в окне редактора кода изменим дескриптор web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
```

```
  <display-name>JAX_RS_Example</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <description>JAX-RS Tools Generated — Do not modify</description>
    <servlet-name>JAX_RS_Example</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>rest.application.config.ApplicationConfig
      </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JAX_RS_Example</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

8. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **JAX_RS_Example** проекта и выберем пункты **Run As | Run on Server**.
9. В браузере введем строку запроса:

```
http://localhost:8080/JAX_RS_Example/resources/datas/1/?api_key=f47ac10b-
58cc-4372-a567-0e02b2c3d479
```

В результате на странице браузера отобразятся данные запроса к таблице `DATA` базы данных `RESTExample` (рис. 3.27).

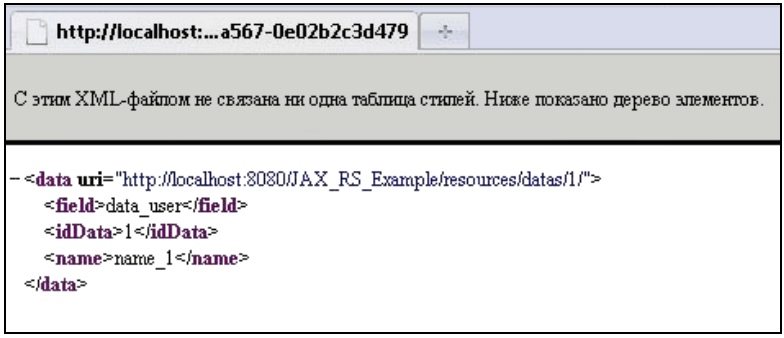


Рис. 3.27. Результат GET-запроса к JAX-RS Web-сервису DatasResource

ГЛАВА 4

Проект Metro

Проект Metro, часть проекта GlassFish, обеспечивает стек Web-сервисов, состоящий из технологии WSIT и реализаций спецификаций JAX-WS и JAXB. Стек Metro позволяет создавать и разворачивать безопасные, надежные, с поддержкой транзакций, совместимые Web-сервисы и клиенты Web-сервисов.

Технология Web Services Interoperability Technology (WSIT) — проект компании Sun Microsystems, позволяющий создавать основанные на Java Web-сервисы, полностью совместимые с WCF Web-сервисами платформы Microsoft .NET 3.*x*. Технология WSIT (или проект Tango — имя, символизирующее танец между двумя стеками Web-сервисов) является частью проекта Metro и представляет собой продукт совместной работы компаний Sun Microsystems и Microsoft, гарантирующий полную совместимость технологий создания Web-сервисов в части безопасности, надежности обмена сообщениями и обеспечения транзакций.

> ***Примечание***
>
> Технология Microsoft Windows Communication Foundation (WCF) является частью платформы Microsoft .NET и содержит библиотеки API для создания безопасных, надежных, поддерживающих транзакции Web-сервисов, способных взаимодействовать с приложениями других платформ. Компании Sun Microsystems и Microsoft обеспечили полную совместимость двух стеков, Metro и WCF, таким образом, что клиенты Metro могут взаимодействовать с Web-сервисами WCF, а клиенты WCF — с Web-сервисами Metro.

Технология WSIT реализует технологии безопасности, надежности и транзакций (свойства Security, Reliability и Transactions) Web-сервисов, описанные такими основными спецификациями, как WS-MetadataExchange, WS-Policy, WS-PolicyAttachment, WS-ReliableMessaging, WS-ReliableMessaging Policy, WS-AtomicTransaction, WS-Coordination, WS-Security, WS-SecurityPolicy, WS-Trust, WS-SecureConversation.

Спецификации, поддерживаемые стеком Metro, в зависимости от версий, представлены в табл. 4.1.

Стек Metro версий 1.0—1.2 поддерживает совместимость с .NET 3.0, а стек Metro версии 1.3 и выше — с .NET 3.5.

Таблица 4.1. *Спецификации, реализованные стеком Metro*

Спецификации	Metro 1.0	Metro 1.3	Metro 2.0
WS-MetadataExchange	Версия 1.1	Версия 1.1	Версия 1.1
WS-Policy	Версия 1.2	Версия 1.5	Версия 1.5
WS-PolicyAttachment	Версия 1.2	Версия 1.5	Версия 1.5
WS-ReliableMessaging	Версия 1.0	Версия 1.1	Версия 1.2
WS-ReliableMessaging Policy	Версия 1.0	Версия 1.1	Версия 1.2
WS-MakeConnection	—	—	Версия 1.1
WS-AtomicTransaction	Версия 1.0	Версия 1.0	Версия 1.0
WS-Coordination	Версия 1.0	Версия 1.0	Версия 1.0
WS-Security	Версии 1.0 и 1.1	Версии 1.0 и 1.1	Версии 1.0 и 1.1
WS-SecurityPolicy	Версия 1.1	Версия 1.2	Версия 1.2
WS-Trust	Версия 1.2	Версия 1.3	Версия 1.4
WS-SecureConversation	Версия 1.2	Версия 1.3	Версия 1.4
JAX-WS	Версия 2.0	Версия 2.1	Версия 2.2
JAXB	Версия 2.0	Версия 2.1	Версия 2.2

Стек Metro идет в комплекте с такими серверами приложений, как GlassFish, Oracle WebLogic и JBoss. Отдельно стек Metro можно получить по адресу **http://metro.java.net/**.

Для создания WSIT Web-сервиса нет необходимости вносить изменения в существующий код JAX-WS Web-сервиса. Свойства Security, Reliability и Transactions Web-сервиса определяются с помощью специального конфигурационного файла приложения wsit-*.xml. Среда выполнения Metro использует конфигурационный WSIT-файл для генерации WSDL-документа, содержащего соответствующие утверждения политик. Также клиент Web-сервиса определяет свои WSIT-свойства с помощью своего конфигурационного WSIT-файла.

Тестирование стека Metro

1. Откроем среду NetBeans и в меню **Файл** выберем пункты **Создать проект | Java Web | Веб-приложение**, нажмем кнопку **Далее**, введем имя проекта Metro_WebService и нажмем кнопку **Далее**. В раскрывающемся списке **Сервер** выберем **GlassFish Server 3** и нажмем кнопку **Завершить**.

   Примечание

   Сервер приложений GlassFish v3 поставляется вместе со стеком Metro версии 2.0.

2. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService** и выберем пункты **Создать | Другие | Веб-службы | Веб-служба** и нажмем кнопку

Далее. Введем имя Web-сервиса MetroTest и имя пакета `metrotest` и нажмем кнопку **Завершить**.

3. Дополним код класса `MetroTest` методом `getHello()`:

```
package metrotest;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
@WebService()
public class MetroTest {
/**
  * Операция веб-службы
  */
  @WebMethod(operationName = "getHello")
  public String getHello(@WebParam(name = "name")
  String name) {
    //TODO write your implementation code here:
    return "Hello"+" "+ name;
  }
}
```

4. Чтобы развернуть созданный Web-сервис, в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService**, выберем пункт **Построить**, а затем пункт **Развернуть**.

Протестировать Web-сервис можно, щелкнув в окне **Проекты** правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выбрав пункт **Тестировать веб-сервис** (рис. 4.1 и 4.2).

Рис. 4.1. Страница тестирования Web-сервиса

Среда NetBeans позволяет включать опции безопасности, надежности и транзакций с помощью вкладки **Качество обслуживания** диалогового окна, которое открывается при выборе пункта **Правка атрибутов веб-службы** после щелчка правой кнопкой мыши в окне **Проекты** на узле **[Веб-приложение] | Веб-службы | [Веб-**

служба]. При этом в папке WEB-INF приложения генерируется WSIT-файл, на основе которого при развертывании Web-сервиса сервером GlassFish создается WSDL-описание Web-сервиса.

Method parameter(s)

Type	Value
java.lang.String	User

Method returned

java.lang.String : "**Hello User**"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:getHello xmlns:ns2="http://metrotest/">
      <name>User</name>
    </ns2:getHello>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getHelloResponse xmlns:ns2="http://metrotest/">
      <return>Hello User</return>
    </ns2:getHelloResponse>
  </S:Body>
</S:Envelope>
```

Рис. 4.2. Результат тестирования Web-сервиса

Оптимизация передачи двоичных данных (MTOM)

1. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Правка атрибутов веб-службы**.
2. В диалоговом окне **MetroTest** на вкладке **Качество обслуживания** (рис. 4.3) отметим флажок **Оптимизировать передачу двоичных данных (MTOM)** и нажмем кнопку **OK**.

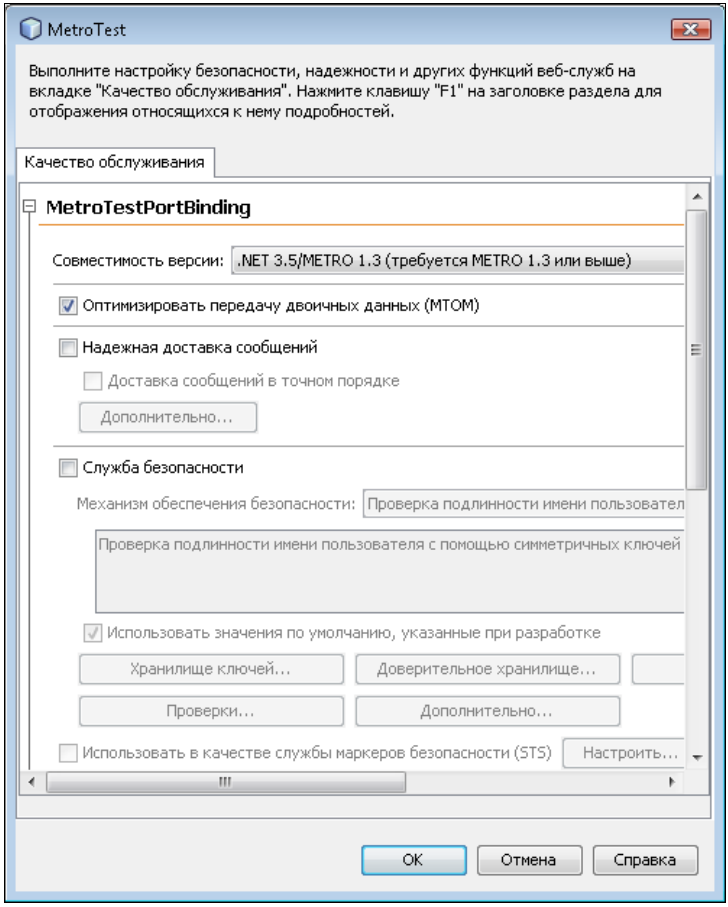


Рис. 4.3. Диалоговое окно установки качества обслуживания Web-сервиса

Примечание

Спецификация SOAP Message Transmission Optimization Mechanism (MTOM) рассматривалась в *разд. "SOAP 1.2" главы 1*.

В результате в узле **Metro_WebService | Веб-страницы | WEB-INF** будет создан конфигурационный WSIT-файл wsit-metrotest.MetroTest.xml, содержащий в WSDL-элементе `<binding>` ссылку `wsp:PolicyReference` на политику, определяющую элемент `<wsoma:OptimizedMimeSerialization>`.

Элемент `<wsoma:OptimizedMimeSerialization>` (пространство имен `xmlns:wsoma="http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization"`) описан спецификацией MTOM Serialization Policy Assertion (WS-MTOMPolicy) и указывает, что в исходящих и входящих сообщениях Web-сервиса должен использоваться механизм MTOM.

1. Добавим в узел **Metro_WebService | Пакеты исходных файлов** папку resources и скопируем в нее файл изображения image.jpg.

2. Дополним код класса `MetroTest` методом `getImage()`:

```
@WebMethod(operationName = "getImage")
public java.awt.Image getImage() {
  java.net.URL url=this.getClass().getResource("/resources/image.jpg");
  try {
    return (java.awt.Image)javax.imageio.ImageIO.read(url);
  } catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
  return null;
}
```

3. В окне **Службы** среды NetBeans щелкнем правой кнопкой мыши на узле **Серверы | GlassFish Server 3** и выберем пункт **Просмотр консоли администратора**.
4. После открытия странички консоли администратора в окне браузера в узле **JVM Settings** откроем вкладку **JVM Options**, нажмем кнопку **Add JVM Option** и введем опцию:

   `-Dcom.sun.xml.ws.transport.http.HttpAdapter.dump=true`

5. Нажмем кнопку **Save** и перезагрузим сервер, щелкнув правой кнопкой мыши на узле **Серверы | GlassFish Server 3** и выбрав пункт **Перезапустить**.

   Теперь в окне **GlassFish Server 3** среды NetBeans будут отображаться входящие и исходящие сообщения Web-сервиса.
6. В окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле **Metro_WebService** и выберем пункты **Очистить и построить | Развернуть**.
7. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Тестировать веб-сервис**.
8. В окне браузера нажмем кнопку **getImage** (рис. 4.4).

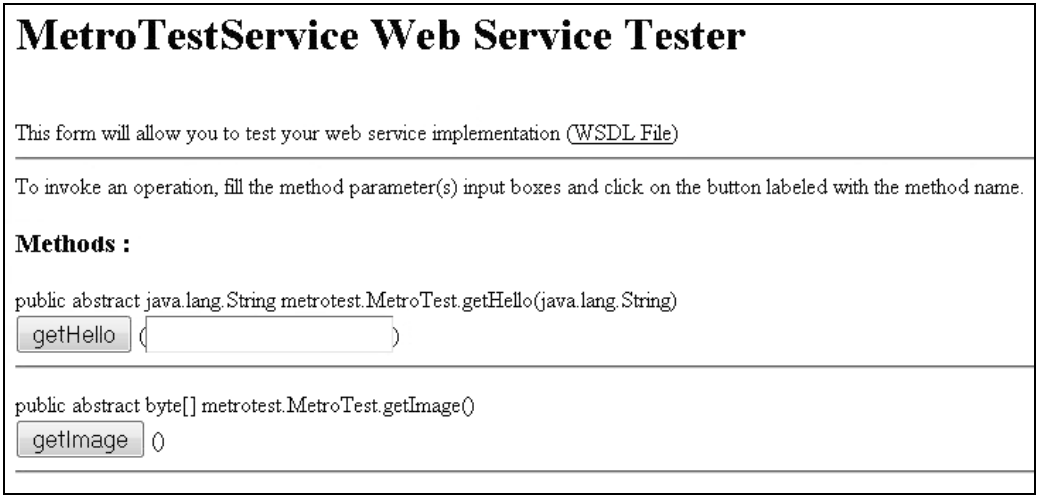


Рис. 4.4. Страница тестирования Web-сервиса MetroTest

В результате в окне вывода **GlassFish Server 3** среды NetBeans появятся входящее и исходящее сообщения Web-сервиса MetroTest:

```
INFO: ---[HTTP request]---
INFO: accept: text/xml, multipart/related
INFO: connection: keep-alive
INFO: content-length: 454
INFO: content-type: multipart/related;start="<rootpart*ce2855b1-de9d-4b53-a84f-
869323318d55@example.jaxws.sun.com>";type="application/xop+xml";boundary="uuid:
ce2855b1-de9d-4b53-a84f-869323318d55";start-info="text/xml"
INFO: host: localhost:8080
INFO: soapaction: "http://metrotest/MetroTest/getImageRequest"
INFO: user-agent: JAX-WS RI 2.2.1-hudson-28-
INFO: --uuid:ce2855b1-de9d-4b53-a84f-869323318d55
Content-Id: <rootpart*ce2855b1-de9d-4b53-a84f-
869323318d55@example.jaxws.sun.com>
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary
<?xml version='1.0' encoding='UTF-8'?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:getImage
xmlns:ns2="http://metrotest/"/></S:Body></S:Envelope>
--uuid:ce2855b1-de9d-4b53-a84f-869323318d55--
INFO: --------------------
INFO: ---[HTTP response 200]---
INFO: --uuid:0a87c539-a6b4-4932-8209-457f4e33ad2f
Content-Id: <rootpart*0a87c539-a6b4-4932-8209-
457f4e33ad2f@example.jaxws.sun.com>
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary
<?xml version='1.0' encoding='UTF-8'?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:
getImageResponse xmlns:ns2="http://metrotest/"><return><xop:Include
xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:952f116d-4a68-4366-
aa01-4b67dda599b1@example.jaxws.sun.com"/></return>
</ns2:getImageResponse></S:Body></S:Envelope>
--uuid:0a87c539-a6b4-4932-8209-457f4e33ad2f
Content-Id: <952f116d-4a68-4366-aa01-4b67dda599b1@example.jaxws.sun.com>
Content-Type: image/png
Content-Transfer-Encoding: binary
%PNG
...
```

Видно, что как входящее, так и исходящее сообщение имеет MIME-тип `application/xop+xml`, а в исходящем сообщении создан XOP-пакет, состоящий из основного SOAP-сообщения и вложения, представляющего двоичные данные возвращаемого изображения.

Перейдя по ссылке **WSDL File**, на странице тестирования Web-сервиса можно увидеть, что при развертывании Web-сервиса на основе его WSIT-файла в WSDL-документ была включена политика:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <ns1:OptimizedMimeSerialization/>
</wsp:Policy>
```

определяющая использование механизма MTOM обработки сообщений.

> ***На компакт-диске***
>
> Пример использования механизма MTOM находится в папке Примеры\Глава4\MTOM компакт-диска.

Для включения механизма MTOM необязательно создавать WSIT-файл. Использование MTOM обеспечивает также аннотация `@javax.xml.ws.soap.MTOM`, при этом есть возможность регулировки порога включения механизма MTOM.

Для демонстрации применения аннотации `@javax.xml.ws.soap.MTOM` в диалоговом окне **MetroTest** на вкладке **Качество обслуживания** сбросим флажок **Оптимизировать передачу двоичных данных (MTOM)** и добавим в код класса `MetroTest` перед аннотацией `@WebService()` аннотацию:

```
@javax.xml.ws.soap.MTOM(threshold=1000000000)
```

После выбора пунктов **Очистить и построить | Развернуть | Тестировать веб-сервис | getImage** в окне **GlassFish Server 3** среды NetBeans можно будет увидеть, что механизм MTOM не был включен. Бинарные данные изображения были закодированы в символьные `base64Binary`-последовательности и включены непосредственно в тело исходящего SOAP-сообщения Web-сервиса. Процесс кодирования/декодирования пересылаемых бинарных данных большого размера занимает большие ресурсы. Использование механизма MTOM позволяет увеличить эффективность передачи бинарных данных в SOAP-сообщении путем включения их в качестве вложения без кодирования в `base64Binary`-формат.

При изменении значения `threshold=0` (по умолчанию) аннотации `@javax.xml.ws.soap.MTOM` бинарные данные изображения снова будут прикреплены к основному SOAP-сообщению как вложение.

Адресация

1. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Правка атрибутов веб-службы**.
2. В диалоговом окне **MetroTest** на вкладке **Качество обслуживания** отметим флажок **Адресовано** и нажмем кнопку **OK**.

В результате WSIT-файл Web-сервиса MetroTest дополнится политикой WSDL-элемента `<binding>`:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsam:Addressing wsp:Optional="true"/>
```

```
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Элемент `<wsam:Addressing>` значением своего атрибута `wsp:Optional="true"` указывает, что конечная точка Web-сервиса должна поддерживать спецификацию WS-Addressing. Для демонстрации этого в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Тестировать веб-сервис**.

После вызова операции Web-сервиса `getHello` в окне браузера отобразятся входящее и исходящее сообщения Web-сервиса, содержащие в своем заголовке элементы `To`, `Action`, `ReplyTo` и `MessageID` спецификации WS-Addressing (рис. 4.5 и 4.6).

Method returned

java.lang.String : "**Hello User**"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing">http://localhost:8080/Metro_WebService/MetroTestService</To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/MetroTest/getHelloRequest</Action>
    <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
  <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
</ReplyTo>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:1691cf20-295b-4910-ae46-8b84a95f8426</MessageID>
  </S:Header>
  <S:Body>
    <ns2:getHello xmlns:ns2="http://metrotest/">
      <name>User</name>
    </ns2:getHello>
  </S:Body>
</S:Envelope>
```

Рис. 4.5. Входящее сообщение Web-сервиса с поддержкой спецификации WS-Addressing

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing/anonymous</To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/MetroTest/getHelloResponse</Action>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:2145d4a5-bb8f-4b98-a0d8-2abebe74f97f</MessageID>
    <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:1691cf20-295b-4910-ae46-8b84a95f8426</RelatesTo>
  </S:Header>
  <S:Body>
    <ns2:getHelloResponse xmlns:ns2="http://metrotest/">
      <return>Hello User</return>
    </ns2:getHelloResponse>
  </S:Body>
</S:Envelope>
```

Рис. 4.6. Исходящее сообщение Web-сервиса с поддержкой спецификации WS-Addressing

НА КОМПАКТ-ДИСКЕ

Пример использования спецификации WS-Addressing находится в папке Примеры\Глава4\Addressing компакт-диска.

Надежная доставка сообщений

Для тестирования надежности доставки сообщений в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Правка атрибутов веб-службы**.

Диалоговое окно **MetroTest** на вкладке **Качество обслуживания** предлагает следующие опции для обеспечения надежности доставки сообщений.

- **Надежная доставка сообщений** — включает механизм надежной доставки сообщений. При этом сообщения группируются в последовательности, каждая из которых соответствует своему клиентскому Proxy-объекту. Для создания последовательности отправитель и конечная точка обмениваются handshake-сообщениями. Каждое сообщение последовательности содержит заголовок с идентификатором последовательности и идентификатором сообщения в последовательности. Конечная точка Web-сервиса группирует сообщения в последовательности, используя идентификатор последовательности, содержащийся в заголовке входящих сообщений. Если установлен флажок **Доставка сообщений в точном порядке**, тогда конечная точка Web-сервиса обрабатывает сообщения, используя идентификаторы сообщений в последовательности. При получении сообщения конечная точка посылает отправителю сообщение-уведомление, содержащее идентификаторы последовательности и сообщения в последовательности. Если отправитель не получает такого уведомления, он повторно посылает утерянное сообщение и требует уведомление о доставке. Отправитель хранит копии сообщений до тех пор, пока не получит уведомление о доставке. При получении всех уведомлений отправитель закрывает последовательность.
- **Доставка сообщений в точном порядке** — конечная точка реконструирует порядок сообщений в последовательности, используя идентификаторы сообщений в последовательности и буферизацию сообщений.
- **Гарантия доставки сообщения** — при включении опции **Только один раз** сообщение доставляется конечному приложению не более одного раза (дублирующие сообщения исключаются конечной точкой), при включении опции **Не менее одного раза** сообщение доставляется конечному приложению по меньшей мере один раз (дублирующие сообщения не исключаются).
- **Управление потоком** — опция **Максимальный размер буфера управления потоком** позволяет контролировать максимальное число сообщений, хранящихся в буфере конечной точки для реконструкции порядка сообщений в последовательности. Если порог буфера превышается, тогда более поздние входящие сообщения последовательности игнорируются. Опция **Тайм-аут бездействия** позволяет определить время ожидания сообщений в последовательности. Если порог ожидания превышается, тогда конечная точка принудительно закрывает последовательность. Для поддержания последовательности отправитель должен периодически посылать конечной точке AckRequested-сообщения.

В диалоговом окне **MetroTest** на вкладке **Качество обслуживания** отметим флажки **Надежная доставка сообщений** и **Доставка сообщений в точном порядке**, нажмем кнопку **Дополнительно**, укажем **Не менее одного раза** и нажмем кнопку **OK**.

В результате во WSIT-файл Web-сервиса будет включена политика, которую можно посмотреть в файле wsit-metrotest.MetroTest.xml папки Примеры\Глава4\RM\Metro_WebService\web\WEB-INF.

Утверждения политики `<wsrm:RMAssertion>`, `<wsrm:DeliveryAssurance>`, `<wsrm:AtLeastOnce/>`, `<wsrm:InOrder/>` были рассмотрены в *разд. "WS-ReliableMessaging Policy" главы 2*, а соответствующие элементы SOAP-сообщений — в *разд. "WS-ReliableMessaging" главы 2*.

После выбора пунктов **Тестировать веб-сервис | getHello** в окне **GlassFish Server 3** среды NetBeans можно будет увидеть обмен сообщениями между клиентом и Web-сервисом.

Создание последовательности:

1. Запрос:

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <To
xmlns="http://www.w3.org/2005/08/addressing">http://localhost:8080/Metro_Web
Service/MetroTestService</To>
    <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-
open.org/ws-rx/wsrm/200702/CreateSequence</Action>
    <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
      <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
    </ReplyTo>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:49e6d183-
1976-4e81-9694-bd038fbd802f</MessageID>
  </S:Header>
  <S:Body>
<ns2:CreateSequence xmlns="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:ns3="http://docs.oasis-open.org/ws-rx/wsmc/200702"
xmlns:ns4="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:ns5="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:ns6="http://schemas.microsoft.com/ws/2006/05/rm">
      <ns2:AcksTo>
      <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
      </ns2:AcksTo>
      <ns2:Offer>
<ns2:Identifier>uuid:25078777-c776-4816-9ca6-d028126f82f9</ns2:Identifier>
        <ns2:Endpoint>
      <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
```

```
        </ns2:Endpoint>
      </ns2:Offer>
    </ns2:CreateSequence>
  </S:Body>
</S:Envelope>
```

2. Ответ:

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header>
<To>http://www.w3.org/2005/08/addressing/anonymous</To>
<Action>http://docs.oasis-open.org/ws-
rx/wsrm/200702/CreateSequenceResponse</Action>
<MessageID>uuid:aecadb21-d1ad-4400-9a95-2a28e7043a78</MessageID>
<RelatesTo>uuid:49e6d183-1976-4e81-9694-bd038fbd802f</RelatesTo>
</S:Header>
<S:Body xmlns:ns6="http://schemas.microsoft.com/ws/2006/05/rm"
xmlns:ns5="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:ns4="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:ns3="http://docs.oasis-open.org/ws-rx/wsmc/200702"
xmlns:ns2="http://docs.oasis-open.org/ws-rx/wsrm/200702">
<ns2:CreateSequence>
<ns2:Identifier>uuid:47dd8256-fe6b-45a0-86a3-2bd03e78cf82</ns2:Identifier>
<ns2:Accept>
<ns2:AcksTo>
<Address>http://localhost:8080/Metro_WebService/MetroTestService
</Address>
</ns2:AcksTo>
</ns2:Accept>
</ns2:CreateSequence>
</S:Body>
</S:Envelope>
```

Вызов метода Web-сервиса:

1. SOAP-запрос:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
<To
xmlns="http://www.w3.org/2005/08/addressing">http://localhost:8080/Metro_Web
Service/MetroTestService</To>
<Action
xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/MetroTest/getH
elloRequest</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
  <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
</ReplyTo>
```

```
<MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:43aaa6ad-b086-
425a-af51-87134f445de1</MessageID>
<ns2:Sequence xmlns="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:ns3="http://docs.oasis-open.org/ws-rx/wsmc/200702"
xmlns:ns4="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:ns5="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:ns6="http://schemas.microsoft.com/ws/2006/05/rm"
xmlns:ns7="http://schemas.xmlsoap.org/soap/envelope/"
ns7:mustUnderstand="true">
<ns2:Identifier>uuid:41aca66f-85db-448f-b03b-49ea8905a6a1</ns2:Identifier>
<ns2:MessageNumber>1</ns2:MessageNumber>
</ns2:Sequence>
<ns2:AckRequested xmlns="http://www.w3.org/2005/08/addressing"
xmlns:ns2="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:ns3="http://docs.oasis-open.org/ws-rx/wsmc/200702"
xmlns:ns4="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:ns5="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:ns6="http://schemas.microsoft.com/ws/2006/05/rm">
<ns2:Identifier>uuid:41aca66f-85db-448f-b03b-49ea8905a6a1</ns2:Identifier>
  </ns2:AckRequested>
  </S:Header>
  <S:Body>
    <ns2:getHello xmlns:ns2="http://metrotest/"/>
  </S:Body>
</S:Envelope>
```

2. SOAP-ответ:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header>
<To
xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addre
ssing/anonymous</To>
<Action
xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/MetroTest/getH
elloResponse</Action>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:3c231d04-9455-
4153-bead-c638f6800561</MessageID>
<RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:43aaa6ad-b086-
425a-af51-87134f445de1</RelatesTo>
</S:Header>
  <S:Body>
    <ns2:getHelloResponse xmlns:ns2="http://metrotest/">
      <return>Hello null</return>
    </ns2:getHelloResponse>
  </S:Body>
</S:Envelope>
```

При изменении опции **Тайм-аут бездействия** на 60 мс в ответ на SOAP-запрос будет выведено предупреждение:

```
The sequence [ ] is in state [ TERMINATING ] and cannot is not ready to
register messages
```

и конечной точкой будет отправлено сообщение, содержащее код ошибки SequenceTerminated, т. к. время между созданием последовательности и получением конечной точкой SOAP-запроса превысит время ожидания сообщений в последовательности, и конечная точка принудительно закроет последовательность.

Система безопасности

Для тестирования системы безопасности обмена сообщениями в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Правка атрибутов веб-службы**. Появившееся при этом диалоговое окно **MetroTest** на вкладке **Качество обслуживания** при установке флажка **Служба безопасности** предлагает широкий выбор настроек для обеспечения безопасности обмена сообщениями.

Опция **Механизм обеспечения безопасности** предлагает следующий набор параметров.

- **Проверка подлинности имени пользователя с помощью симметричного ключа** содержит следующие настройки:
  - **Маркер проверки подлинности** содержит значения **Имя пользователя**, **X509**;
  - **Набор алгоритмов** содержит значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
  - **Макет заголовка безопасности** содержит значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
  - **Требуются производные ключи**;
  - **Установить безопасное соединение (защищенный диалог)**;
  - **Требуются производные ключи для безопасного сеанса**;
  - **Требуется подтверждение подписи**;
  - **Подпись шифрования**;
  - **Зашифровать перед подписанием**;
  - **Поддержка хэш-паролей**;
- **Username Authentication with Password Derived Key** предлагает следующие настройки:
  - **Набор алгоритмов** содержит значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15,

192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;

- **Макет заголовка безопасности** содержит значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
- **Требуются производные ключи**;
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Protect Tokens**;

❒ **Безопасность совместных сертификатов** содержит следующие настройки:

- **Набор алгоритмов** содержит значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
- **Макет заголовка безопасности** содержит значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

❒ **Симметричная привязка к маркеру Kerberos** содержит следующие настройки:

- **Набор алгоритмов** содержит значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
- **Макет заголовка безопасности** содержит значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
- **Требуются производные ключи**;
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

❒ **Безопасность транспорта (SSL)** содержит настройку **Требуется сертификат клиента**;

- **Проверка подлинности сообщения по SSL** предлагает следующие настройки:
  - **Маркер проверки подлинности** содержит значения **Имя пользователя**, **X509**;
  - **Версия WSS** — значения **1.0**, **1.1**;
  - **Набор алгоритмов** содержит значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
  - **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
  - **Установить безопасное соединение (защищенный диалог)**;
  - **Требуются производные ключи для безопасного сеанса**;
  - **Требуется подтверждение подписи**;
- **Проверка подлинности SAML по SSL** содержит следующие настройки:
  - **Версия SAML** — значения **1.0 (профиль 1.0)**, **1.0 (профиль 1.1)**, **1.1 (профиль 1.0)**, **1.1 (профиль 1.1)**, **2.0 (профиль 1.1)**;
  - **Версия WSS** — выбор **1.0**, **1.1**;
  - **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
  - **Макет заголовка безопасности** — выбор **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
  - **Требуется сертификат клиента**;
  - **Требуется подтверждение подписи**;
- **Одобрение сертификата** содержит следующие настройки:
  - **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
  - **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
  - **Требуются производные ключи**;
  - **Установить безопасное соединение (защищенный диалог)**;
  - **Требуются производные ключи для безопасного сеанса**;

- **Требуется подтверждение подписи**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

❒ **Подтверждение подлинности отправителя SAML сертификатом** содержит следующие настройки:

- **Версия SAML** — значения **1.0 (профиль 1.0)**, **1.0 (профиль 1.1)**, **1.1 (профиль 1.0)**, **1.1 (профиль 1.1)**, **2.0 (профиль 1.1)**;
- **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
- **Макет заголовка безопасности** — выбор **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
- **Требуются производные ключи**;
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

❒ **Держатель ключа SAML** содержит следующие настройки:

- **Версия SAML** — значения **1.0 (профиль 1.0)**, **1.0 (профиль 1.1)**, **1.1 (профиль 1.0)**, **1.1 (профиль 1.1)**, **2.0 (профиль 1.1)**;
- **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
- **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

❒ **Выпущенный STS маркер** содержит следующие настройки:

- **Адрес источника**;
- **Метаданные адреса источника**;

- **Тип маркера** — значения **1.0**, **1.1**, **2.0**;
- **Тип ключа** — значения **Симметричный ключ**, **Публичный ключ**;
- **Размер ключа** — значения **128**, **192**, **256**;
- **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
- **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
- **Требуются производные ключи для выпущенного маркера**;
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Требуется подтверждение подписи**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

❒ **Выпущенный STS маркер с сертификатом службы** содержит следующие настройки:

- **Адрес источника**;
- **Метаданные адреса источника**;
- **Тип маркера** — значения **1.0**, **1.1**, **2.0**;
- **Тип ключа** — значения **Симметричный ключ**, **Публичный ключ**;
- **Размер ключа** — значения **128**, **192**, **256**;
- **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
- **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь);**
- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Требуется подтверждение подписи**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**;

- **Выпущенный STS маркер одобрения** содержит следующие настройки:
  - **Адрес источника**;
  - **Метаданные адреса источника**;
  - **Тип маркера** — значения **1.0**, **1.1**, **2.0**;
  - **Тип ключа** — значения **Симметричный ключ**, **Публичный ключ**;
  - **Размер ключа** — значения **128**, **192**, **256**;
  - **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
  - **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь);**
  - **Требуются производные ключи для маркера X509**;
  - **Требуются производные ключи для выпущенного маркера**;
  - **Установить безопасное соединение (защищенный диалог)**;
  - **Требуются производные ключи для безопасного сеанса**;
  - **Требуется подтверждение подписи**;
  - **Подпись шифрования**;
  - **Зашифровать перед подписанием**;
- **Выпущенный STS маркер поддержки** содержит следующие настройки:
  - **Адрес источника**;
  - **Метаданные адреса источника**;
  - **Тип маркера** — значения **1.0**, **1.1**, **2.0**;
  - **Тип ключа** — значения **Симметричный ключ**, **Публичный ключ**;
  - **Размер ключа** — значения **128**, **192**, **256**;
  - **Набор алгоритмов** — значения **Basic**: 256bit, 192bit, 128bit, 256Rsa15, 192Rsa15, 128Rsa15, 256Sha256, 192Sha256, 128Sha256, 256Sha256Rsa15, 192Sha256Rsa15, 128Sha256Rsa15; TriplDes; TriplDesRsa15; TriplDesSha256; TriplDesSha256Rsa15;
  - **Макет заголовка безопасности** — значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)**, **Свободно (метка времени в последнюю очередь)**;
  - **Требуются производные ключи для маркера X509**;
  - **Требуются производные ключи для выпущенного маркера**;

- **Установить безопасное соединение (защищенный диалог)**;
- **Требуются производные ключи для безопасного сеанса**;
- **Требуется подтверждение подписи**;
- **Подпись шифрования**;
- **Зашифровать перед подписанием**.

Создание клиента Web-сервиса

Для тестирования системы безопасности обмена сообщениями создадим клиента Web-сервиса.

1. Для этого в меню **Файл** среды NetBeans выберем пункты **Создать проект | Java Web | Веб-приложение**, нажмем кнопку **Далее**, введем имя проекта Metro_WebClient и нажмем кнопку **Далее**. В раскрывающемся списке **Сервер** выберем **GlassFish Server 3** и нажмем кнопку **Завершить**.
2. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient** и выберем пункты **Создать | Другие | Веб-службы | Клиент веб-службы**, нажмем кнопку **Далее**. С помощью кнопки **Обзор** выберем Web-сервис MetroTest и последовательно нажмем кнопки **OK** и **Завершить**.

В результате для клиента Web-сервиса средой NetBeans будут сгенерированы артефакты, которые будут использоваться сервлетом клиента для взаимодействия с Web-сервисом.

Для создания сервлета клиента:

1. Щелкнем правой кнопкой мыши в окне **Проекты** на узле **Metro_WebClient** и выберем пункты **Создать | Другие | Веб | Сервлет**, нажмем кнопку **Далее**, введем имя класса `MetroClient` и имя пакета `metroclient`, нажмем кнопку **Далее**, укажем **Добавление информации к дескриптору развертывания (web.xml)** и нажмем кнопку **Завершить**.
2. В окне редактора в коде класса `MetroClient` щелкнем правой кнопкой мыши и выберем пункты **Вставить код | Вызов операции веб-службы | Metro_WebClient | MetroTestService | MetroTestPort | getHello**, нажмем кнопку **OK**.

   В результате в класс `MetroClient` будет добавлен код:

```
import javax.xml.ws.WebServiceRef;
. . .
@WebServiceRef(wsdlLocation = "WEB-
INF/wsdl/localhost_8080/Metro_WebService/MetroTestService.wsdl")
. . .
private MetroTestService service;
private String getHello(java.lang.String name) {
  metrotest.MetroTest port = service.getMetroTestPort();
  return port.getHello(name);
}
```

3. Модифицируем код JSP-страницы index.jsp узла **Веб-страницы** проекта Metro_WebClient и код метода `processRequest()` класса `MetroClient` (листинги 4.1 и 4.2).

Листинг 4.1. Код JSP-страницы index.jsp проекта Metro_WebClient

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Service Hello</h1>
    <form name="Name" method="post" action="MetroClient">
      <p><textarea cols="1" rows="1" name="TextAreaName"
                   ID="TextareaName" ></textarea>
      <p>
      <input type="submit" value="Отправить" name="namebutton">
    </form>
  </body>
</html>
```

Листинг 4.2. Код метода `processRequest()` класса `MetroClient`

```
protected void processRequest(HttpServletRequest request,
   HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      String Name = request.getParameter("TextAreaName");
      String Response = getHello (Name);
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet MetroClient</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>"+Response+"</h1>");
      out.println("</body>");
      out.println("</html>");

    } finally {
      out.close();
    }
}
```

После развертывания проекта Metro_WebClient с помощью щелчка правой кнопкой мыши в окне **Проекты** и выбора пунктов **Построить** и **Развернуть** запустим клиента Web-сервиса командой **Выполнить**. В результате в окне браузера появится страница приветствия Web-приложения Metro_WebClient с полем ввода имени, заполнив которое и нажав кнопку **Отправить**, в ответ можно увидеть строку, возвращаемую методом `getHello()` Web-сервиса MetroTest.

Настройку качества обслуживания клиента Web-сервиса будем осуществлять щелчком правой кнопкой мыши в окне **Проекты** на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выбором пункта **Правка атрибутов веб-службы**.

Опция *Проверка подлинности имени пользователя с помощью симметричного ключа*

Настроим качество обслуживания Web-сервиса MetroTest.

Для этого в опции **Механизм обеспечения безопасности** выберем **Проверка подлинности имени пользователя с помощью симметричного ключа**. Нажав кнопку **Настройка**, для параметра **Маркер проверки подлинности** укажем значение **Имя пользователя**, для параметра **Набор алгоритмов** — значение **Basic 256bit**, для параметра **Макет заголовка безопасности** — значение **Строго** и дважды нажмем кнопку **OK**.

В результате будет сгенерирован WSIT-файл, содержащий политику, связанную с WSDL-элементом `<binding>`, и политики, связанные с WSDL-элементами `<input>` и `<output>`.

> ***На компакт-диске***
>
> Сгенерированный WSIT-файл Web-сервиса можно посмотреть в папке Примеры\Глава4\Security\SymmetricBinding\Metro_WebService\web\WEB-INF компакт-диска.

Элементы пространства имен `xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"` были рассмотрены в *разд. "WS-SecurityPolicy" главы 2*.

Элемент `<sp:SymmetricBinding>` политики WSDL-элемента `<binding>` является утверждением требования защиты сообщений как для клиента, так и для конечной точки, с помощью цифровой подписи и шифрования симметричным ключом согласно спецификации WS-Security.

Здесь в элемент `<sp:SymmetricBinding>` включены элементы `<sp:ProtectionToken>`, `<sp:Layout>`, `<sp:IncludeTimestamp/>`, `<sp:OnlySignEntireHeadersAndBody/>` и `<sp:AlgorithmSuite>`.

Элемент `<sp:ProtectionToken>` определяет требования к шифрованию и цифровой подписи сообщений. В данном случае элемент `<sp:ProtectionToken>` содержит элемент `<sp:X509Token>` со значением атрибута `sp:IncludeToken`, указывающим, что включение в сообщение маркера защиты `<wsse:BinarySecurityToken>` не требуется, а используется внешняя ссылка на него.

Элемент `<sp:Layout>` требует определенного порядка элементов в заголовке `<wsse:Security>` сообщения. В данном случае элемент `<sp:Layout>` содержит элемент `<sp:Strict/>`, указывающий, что элементы заголовка `<wsse:Security>` должны быть продекларированы перед их использованием.

Элемент `<sp:IncludeTimestamp/>` требует наличие в сообщении элемента `<wsu:Timestamp>`, определяющего время создания заголовка `<wsse:Security>` и время его окончания действия.

Элемент `<sp:OnlySignEntireHeadersAndBody/>` указывает, что цифровой подписью должно снабжаться все тело SOAP-сообщения и полностью весь заголовок сообщения. Дочерние элементы отдельно не снабжаются цифровой подписью, исключение при этом составляет заголовок `<wsse:Security>`.

Элемент `<sp:AlgorithmSuite>` определяет набор алгоритмов для выполнения операций шифрования. В данном случае элемент `<sp:AlgorithmSuite>` содержит элемент `<sp:Basic256/>` — применяется набор алгоритмов Sha1, Aes256, KwAes256, KwRsaOaep, PSha1L256, PSha1L192, минимальная длина симметричного ключа — 256.

Элемент `<sp:Wss11>` политики WSDL-элемента `<binding>` определяет поддержку свойств спецификации WS-Security версии 1.1. В данном случае элемент `<sp:Wss11>` содержит элементы `<sp:MustSupportRefIssuerSerial/>` (требует поддержку ссылок на маркеры защиты с помощью элемента `<ds:X509IssuerSerial>`), `<sp:MustSupportRefThumbprint/>` (требует при ссылках поддержки URI-идентификатора `http://docs.oasis-open.org/wss/oasiswss-soap-message-security-1.1#ThumbPrintSHA1`) и `<sp:MustSupportRefEncryptedKey/>` (требует при ссылках поддержки URI-идентификатора `http://docs.oasis-open.org/wss/oasis-wss-soap-message-security-1.1#EncryptedKeySHA`).

Элемент `<sp:SignedEncryptedSupportingTokens>` политики WSDL-элемента `<binding>` требует включения в сообщение одного или нескольких маркеров защиты, которые для защиты должны быть зашифрованы и подписаны цифровой подписью. В данном случае элемент `<sp:SignedEncryptedSupportingTokens>` содержит элемент `<sp:UsernameToken>` со значением атрибута `sp:IncludeToken`, указывающим, что все входящие сообщения конечной точки должны содержать маркер защиты `<wsse:UsernameToken>`, при этом маркер защиты соответствует спецификации UsernameTokenProfile версии 1.0 (элемент `<sp:WssUsernameToken10/>`).

Элемент `<sc:KeyStore>` политики WSDL-элемента `<binding>` указывает расположение хранилища ключей и сертификатов.

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** в окне **MetroTest** на вкладке **Качество обслуживания**, используя кнопку **Хранилище ключей**, можно переопределить расположение по умолчанию хранилища ключей и сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик `com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidator`, отвечающий за проверку имени/пароля во входящих сообщениях на стороне сервера.

Кнопка **Дополнительно** позволяет установить:

- опцию **Максимальный фазовый сдвиг синхронизирующих импульсов (мс)** — максимальное допустимое различие между системными часами сервера и клиента, используемое при проверке значения элемента `<wsu:Timestamp>`;
- опцию **Предел старения метки времени (мс)** — максимальное значение элемента `<wsu:Timestamp>`;
- опцию **Использовать механизм отмены сертификатов по умолчанию** — использование механизма по умолчанию проверки устаревших сертификатов.

Политика WSDL-элементов `<input>` и `<output>` определяет, что в исходящих и входящих сообщениях конечной точки тело сообщения шифруется и подписывается, а перечисленные заголовки сообщения только снабжаются цифровой подписью.

При использовании опции **Проверка подлинности имени пользователя с помощью симметричного ключа** клиент посылает серверу имя/пароль для аутентификации, соответственно сервер должен иметь предварительно зарегистрированного пользователя. Посылаемый клиентом маркер защиты `<wsse:UsernameToken>` зашифрован симметричным ключом, который генерируется средой выполнения клиента. Симметричный ключ, в свою очередь, также шифруется и отсылается серверу. Шифрование симметричного ключа осуществляется с помощью публичного ключа сертификата сервера, который хранится в соответствующем доверительном хранилище. При получении зашифрованного симметричного ключа, сервер расшифровывает его с помощью приватного ключа своего сертификата и, используя расшифрованный симметричный ключ, расшифровывает полученный маркер защиты. Затем сервер сравнивает расшифрованный маркер защиты с данными хранилища ключей и, в случае успешной аутентификации, разрешает доступ клиенту к Web-сервису. Симметричный ключ также используется для шифрования и подписи тела и заголовков сообщений.

Для регистрации пользователя сервера GlassFish в окне **Службы** среды NetBeans щелкнем правой кнопкой мыши на узле **Серверы | GlassFish Server 3** и выберем пункт **Просмотр консоли администратора**. На странице консоли администратора выберем узел **Security | Realms | file**, нажмем кнопки **Manage Users** и **New**, введем имя пользователя и его пароль и нажмем кнопку **OK**.

Так как WSDL-описание Web-сервиса MetroTest изменилось с изменением его WSIT-файла, в проекте Metro_WebClient необходимо обновить клиента Web-сервиса MetroTest, щелкнув правой кнопкой мыши на узле **Ссылки на веб-службы | MetroTestService** и выбрав пункт **Обновить**.

Чтобы сконфигурировать клиента:

1. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.
2. В диалоговом окне **MetroTestService** на вкладке **Качество обслуживания** (рис. 4.7) сбросим флажок **Использовать значения по умолчанию, указанные при разработке** и в полях **Имя пользователя по умолчанию** и **Пароль по**

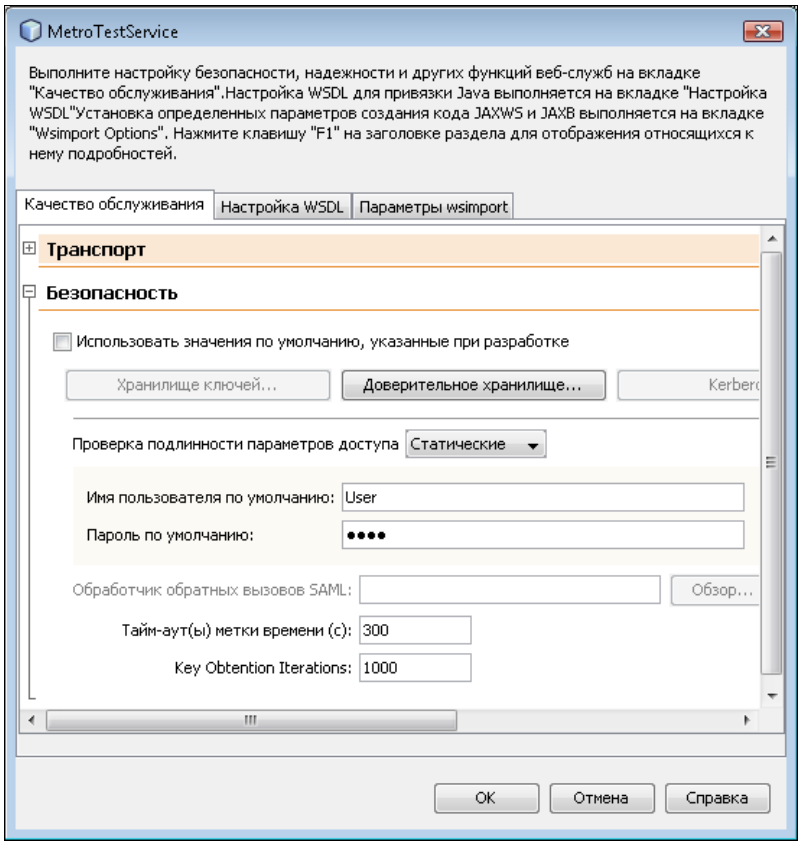


Рис. 4.7. Диалоговое окно настройки качества обслуживания клиента Web-сервиса

умолчанию введем имя пользователя и пароль зарегистрированного пользователя сервера, нажмем кнопку **OK**.

В результате в папке META-INF проекта Metro_WebClient будет создан WSIT-файл клиента Web-сервиса, содержащий политику:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:TrustStore wspp:visibility="private"
         location="D:\Program Files\glassfish-
3.0.1\glassfish\domains\domain1\config\cacerts.jks" type="JKS"
storepass="changeit" peeralias="xws-security-server"/>
        <sc:CallbackHandlerConfiguration wspp:visibility="private">
        <sc:CallbackHandler default="User" name="usernameHandler"/>
        <sc:CallbackHandler default="1234" name="passwordHandler"/>
      </sc:CallbackHandlerConfiguration>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Элемент `<sc:TrustStore>` указывает расположение доверительного хранилища сертификата сервера, а элемент `<sc:CallbackHandlerConfiguration>` определяет для обработчиков имени и пароля `javax.security.auth.callback.NameCallback` и `javax.security.auth.callback.PasswordCallback`, отвечающих за запрос у клиента имени/пароля, значения по умолчанию как данные авторизированного пользователя сервера.

Кнопка **Доверительное хранилище** окна **MetroTestService** на вкладке **Качество обслуживания** позволяет переопределить расположение по умолчанию доверительного хранилища сертификата сервера. Изменение опции **Проверка подлинности параметров доступа** со значения **Статические** на значение **Динамический** позволяет указать пользовательскую реализацию обработчиков имени и пароля. Поле **Тайм-аут(ы) метки времени (с)** дает возможность установить значение элемента `<wsu:Timestamp>` для клиентских сообщений. С помощью поля **Key Obtention Iterations** можно установить количество итераций шифрования ключа.

В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient** и выберем пункты **Очистить и построить | Выполнить**, заполним поле ввода имени и нажмем кнопку **Отправить**. В результате в окне браузера появится строка, возвращаемая методом `getHello()` Web-сервиса MetroTest, а в окне **GlassFish Server 3** — входящее и исходящее сообщения Web-сервиса.

Входящее сообщение Web-сервиса содержит зашифрованный симметричный ключ, зашифрованный маркер защиты, зашифрованное и подписанное тело сообщения и подписанные цифровой подписью заголовки сообщения. Исходящее сообщение Web-сервиса содержит зашифрованное и подписанное тело сообщения и подписанные цифровой подписью заголовки сообщения.

Кнопка **Настройка** опции **Проверка подлинности имени пользователя с помощью симметричного ключа** окна **MetroTest** на вкладке **Качество обслуживания** Web-сервиса позволяет переопределить набор алгоритмов с помощью опции **Набор алгоритмов**. При этом изменяется дочерний элемент элемента `<sp:AlgorithmSuite>`, являющегося дочерним элементом элемента `<sp:SymmetricBinding>` политики WSDL-элемента `<binding>`.

При изменении, с помощью кнопки **Настройка**, опции **Макет заголовка безопасности** изменяется дочерний элемент элемента `<sp:Layout>` политики WSDL-элемента `<binding>`. В случае выбора значений **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)** расположение дочерних элементов заголовка `<wsse:Security>` по факту может не меняться. При выборе варианта **Свободно (метка времени в последнюю очередь)** элемент `<wsu:Timestamp>` становится последним дочерним элементом заголовка `<wsse:Security>`, а клиент Web-сервиса может выдавать ошибку при попытке дешифрования ответного сообщения от Web-сервиса из-за несовпадения порядка расположения дочерних элементов `<xenc:ReferenceList>` и `<ds:Signature>` заголовка `<wsse:Security>` во входящем и исходящем сообщениях.

При нажатии кнопки **Настройка** и выбора опции **Требуются производные ключи**, в политике WSDL-элемента `<binding>` появится дочерний элемент `<sp:Require`

DerivedKeys/> элемента <sp:X509Token>, устанавливая тем самым, что для шифрования маркера защиты, а также для шифрования и подписи тела сообщения и заголовков сообщения будет использоваться не сам симметричный ключ, а его производные ключи. Соответственно в заголовке <wsse:Security> появятся элементы <wsc:DerivedKeyToken>, содержащие информацию о производных ключах.

В случае нажатия кнопки **Настройка** и выбора опции **Требуется подтверждение подписи** в политике WSDL-элемента <binding> появится дочерний элемент <sp:RequireSignatureConfirmation/> элемента <sp:Wss11>, указывая поддержку элемента <wsse11:SignatureConfirmation>. При этом в исходящее сообщение Web-сервиса будет включен дочерний элемент <wsse11:SignatureConfirmation> заголовка <wsse:Security>, содержащий полученную Web-сервисом от клиента цифровую подпись входящего сообщения.

В случае нажатия кнопки **Настройка** и выбора опции **Подпись шифрования** в политике WSDL-элемента <binding> появится дочерний элемент <sp:EncryptSignature/> элемента <sp:SymmetricBinding>, определяя, что цифровая подпись должна быть зашифрована. При этом дочерние элементы <ds:Signature> заголовка <wsse:Security> сообщений будут зашифрованы.

В случае нажатия кнопки **Настройка** и выбора опции **Зашифровать перед подписанием** в политике WSDL-элемента <binding> появится дочерний элемент <sp:EncryptBeforeSigning/> элемента <sp:SymmetricBinding>, указывая, что цифровая подпись сообщения создается после его шифрования, при этом используются симметричные ключи.

При нажатии кнопки **Настройка** и выбора опции **Поддержка хэш-паролей** в политике WSDL-элемента <binding> появится дочерний элемент <sp:HashPassword/> элемента <sp:UsernameToken>, указывая, что пароль маркера защиты должен быть отдельно зашифрован. В этом случае элемент <wsse:Password> маркера защиты имеет значение PasswordDigest атрибута Type и дочерние элементы <Nonce> и <Created>. При использовании цифрового пароля необходимо создать собственный класс, расширяющий com.sun.xml.wss.RealmAuthenticationAdapter, который будет производить аутентификацию клиента относительно Custom Realm на стороне сервера, т. к. DefaultRealmAuthenticationAdapter сервера GlassFish не поддерживает цифровую аутентификацию.

В случае нажатия кнопки **Настройка** и последовательного выбора опций **Маркер проверки подлинности**, **X509**, **Установить безопасное соединение (защищенный диалог)**, **Требуются производные ключи для безопасного сеанса** политика WSDL-элемента <binding> дополнится элементами, которые можно посмотреть в файле wsit-metrotest.MetroTest.xml папки Примеры\Глава4\Security\SecureConversation\Metro_WebService\web\WEB-INF компакт-диска.

Дочерний элемент <sp:SecureConversationToken> элемента <sp:ProtectionToken> указывает требование получения маркера защиты <wsc:SecurityContextToken> *(см. разд. "WS-SecureConversation" главы 2)*. Маркер защиты <wsc:Security ContextToken> используется для цифровой подписи и шифрования SOAP-сообщений и является маркером защиты контекста (Security Context Token, SCT) для

создания безопасной сессии обмена многочисленными SOAP-сообщениями между несколькими сторонами с применением сессионных ключей, общими для всех участников сессии. Стандартный механизм получения такого маркера защиты описывается спецификацией WS-Trust.

Дочерний элемент `<sp:RequireDerivedKeys/>` элемента `<sp:SecureConversationToken>` устанавливает обязательность использования производного ключа защищенной сессии `<wsc:DerivedKeyToken>`, а дочерний элемент `<sp:BootstrapPolicy>` описывает политику получения и использования маркера защиты `<wsc:SecurityContextToken>`.

Элемент `<sp:Trust13>` политики WSDL-элемента `<binding>` определяет поддержку свойств спецификации WS-Trust версии 1.3. Его дочерние элементы `<sp:RequireClientEntropy/>`, `<sp:RequireServerEntropy/>` и `<sp:MustSupportIssuedTokens/>` требуют наличия элемента `<wst:Entropy>` в RST-запросе, наличия элемента `<wst:Entropy>` в RSTR-ответе и поддержки заголовка `<wst:IssuedTokens>`.

Для запуска клиента Web-сервиса MetroTest с измененным WSIT-файлом и соответственно измененным WSDL-описанием необходимо в проекте Metro_WebClient обновить клиента Web-сервиса MetroTest.

При таком WSIT-файле Web-сервиса клиент сначала посылает RST-запрос серверу для выдачи SCT-маркера. В ответ сервер посылает клиенту RSTR-ответ с SCT-маркером. При этом для шифрования и подписи тела и заголовков RST-запроса и RSTR-ответа используется симметричный ключ, сгенерированный средой выполнения клиента и зашифрованный публичным ключом сертификата сервера.

После получения клиентом SCT-маркера устанавливается защищенная сессия, и входящее сообщение Web-сервиса содержит маркер защиты `<wsc:SecurityContextToken>` в заголовке `<wsse:Security>`, а также элементы `<wsc:DerivedKeyToken>`, описывающие производные ключи, полученные из маркера защиты `<wsc:SecurityContextToken>` и используемые для шифрования маркера защиты `<wsse:UsernameToken>`, а также для шифрования и подписи тела сообщения и его заголовков. Исходящее сообщение Web-сервиса содержит элементы `<wsc:DerivedKeyToken>` в заголовке `<wsse:Security>`, описывающие производные ключи SCT-маркера, используемые для шифрования и подписи тела сообщения и его заголовков.

Вкладка **Качество обслуживания** окна Web-сервиса также дает возможность настраивать политику WSIT-файла Web-сервиса и соответственно его WSDL-описания на уровне входящего и исходящего сообщений с помощью опций узлов **Входное сообщение** и **Выходное сообщение** (рис. 4.8).

При выборе вместе с опцией **Маркер проверки подлинности** опций **Зашифровано**, **Подписано** и **Одобрение** во входящее сообщение Web-сервиса дополнительно включается указанный маркер защиты, который может быть зашифрован, подписан и использован для создания вторичной цифровой подписи сообщения соответственно.

Кнопка **Части сообщения** позволяет регулировать наличие заголовков в сообщении, а также подпись и шифрование заголовков и тела сообщения (рис. 4.9).

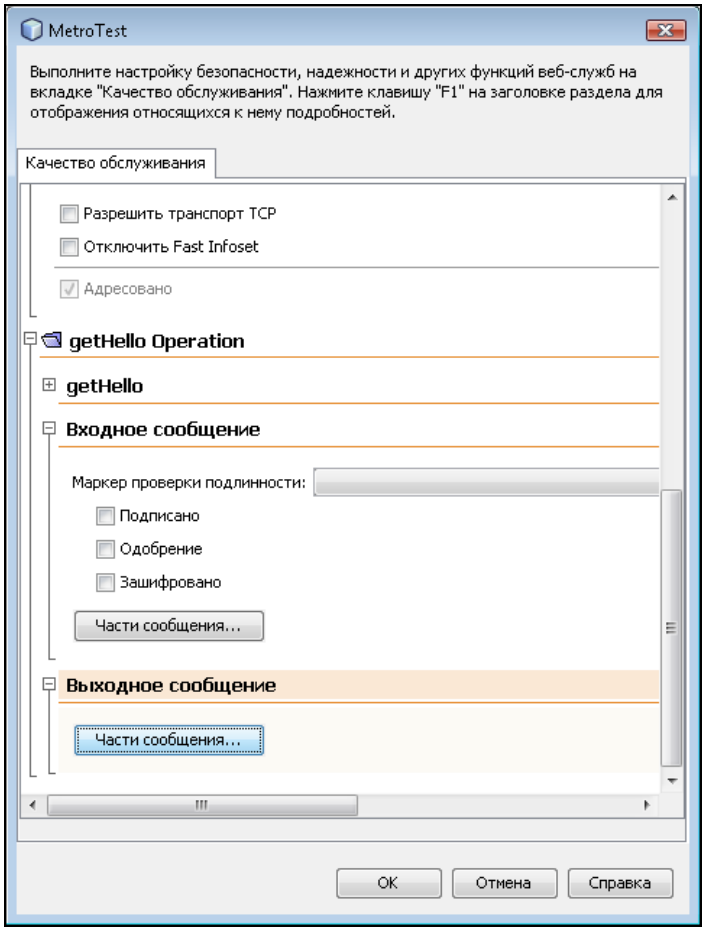


Рис. 4.8. Настройка политик входящего и исходящего сообщений Web-сервиса

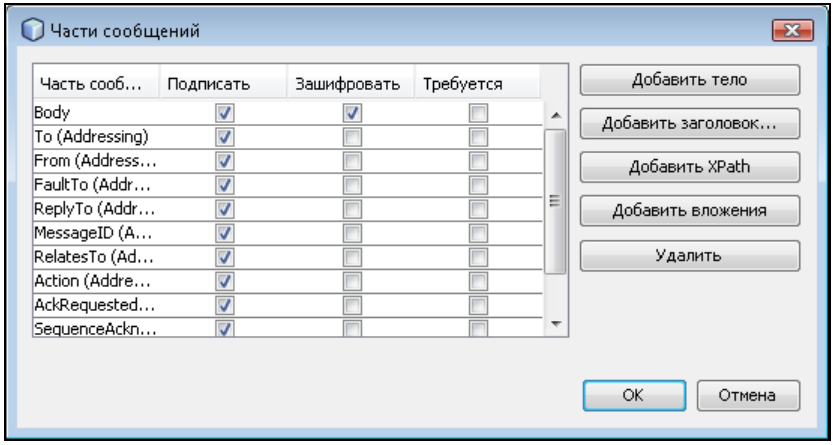


Рис. 4.9. Диалоговое окно настройки защиты частей сообщения

При настройке опций узлов **Входное сообщение** и **Выходное сообщение** соответственно меняются политики WSDL-элементов `<input>` и `<output>`.

Опция **Транзакция** узла **getHello Operation | getHello** обеспечивает поддержку спецификации WS-AtomicTransaction (рис. 4.10).

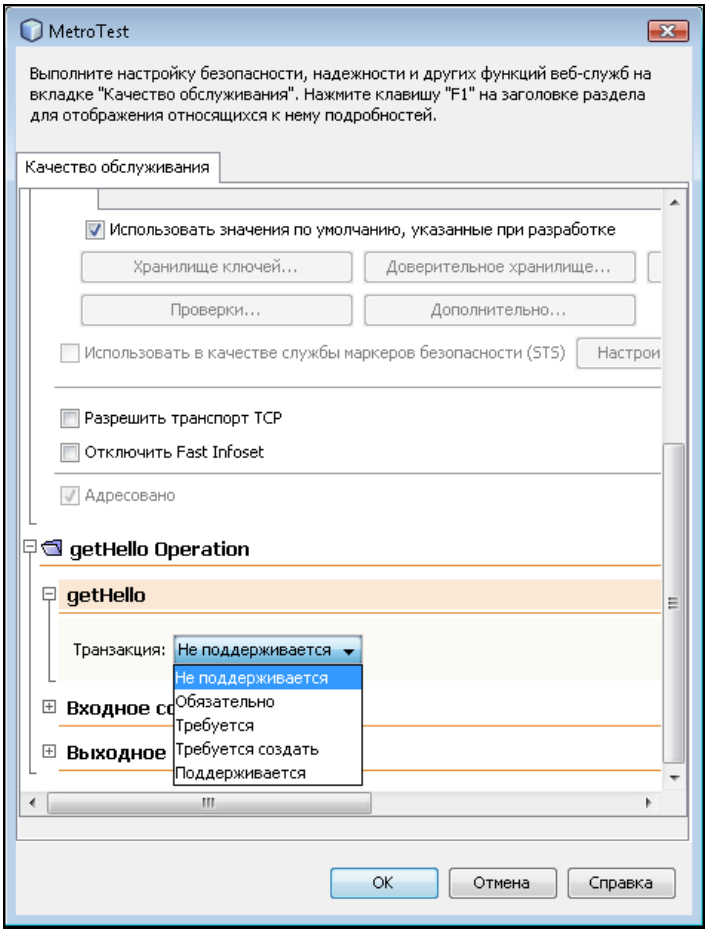


Рис. 4.10. Настройка поддержки атомных транзакций

Опция *Username Authentication with Password Derived Key*

Механизм применения данной опции близок к механизму опции **Проверка подлинности имени пользователя с помощью симметричного ключа**. Отличие состоит в том, что для шифрования и подписи сообщений используется не симметричный ключ, генерируемый средой выполнения клиента, а производный ключ, полученный из пароля клиента (формула `SHA-1(nonce + created + password)`).

Для тестирования данной опции в окне **MetroTest** на вкладке **Качество обслуживания** Web-сервиса, в опции **Механизм обеспечения безопасности** выберем **Username Authentication with Password Derived Key**. Нажав кнопку **Настройка**,

для опции **Набор алгоритмов** укажем **Basic 256bit**, а для параметра **Макет заголовка безопасности** выберем вариант **Строго** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\PasswordValidator\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sc:ValidatorConfiguration>` указывает класс реализации интерфейса `javax.security.auth.callback.Callback` для обработки производного ключа пароля на стороне сервера.

Основу данного класса среда NetBeans сгенерирует в папке \src\java\validators проекта Metro_WebService.

Откроем узел **Пакеты исходных файлов | validators | DerivedKeyPasswordValidator.java** и отредактируем код класса `DerivedKeyPasswordValidator` (листинг 4.3).

Листинг 4.3. Код класса `DerivedKeyPasswordValidator`

```
package validators;

import com.sun.xml.wss.impl.callback.PasswordValidationCallback;
import
com.sun.xml.wss.impl.callback.PasswordValidationCallback.DerivedKeyPasswordRequest;
import
com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidationException;
import com.sun.xml.wss.impl.callback.PasswordValidationCallback.Request;

public class DerivedKeyPasswordValidator extends
PasswordValidationCallback.DerivedKeyPasswordValidator {
  public DerivedKeyPasswordValidator() {}

  @Override
  public void setPassword(Request request) {
    if (request instanceof DerivedKeyPasswordRequest) {
       ((DerivedKeyPasswordRequest)request).setPassword("1234");
    }
  }

  @Override
  public boolean validate(Request rqst) throws PasswordValidationException
  { return true; }
}
```

После построения и развертывания проекта Metro_WebService, а также после обновления клиента Web-сервиса MetroTest в проекте Metro_WebClient, в результате его запуска произойдет обмен сообщениями между клиентом и Web-серви-

сом. Входящее сообщение Web-сервиса будет содержать маркер защиты `<wsse:UsernameToken>` с дочерними элементами `<wsse:Username>`, `<wsse:Salt>` и `<wsse:Iterations>`. При этом тело входящего сообщения будет зашифровано и подписано, а его заголовки подписаны производным ключом пароля. Исходящее сообщение Web-сервиса будет также зашифровано (в части тела сообщения) и подписано (в части тела и заголовков сообщения) тем же производным ключом пароля.

При изменении, с помощью кнопки **Настройка**, опции **Набор алгоритмов** изменяется дочерний элемент элемента `<sp:AlgorithmSuite>`, являющегося дочерним элементом элемента `<sp:SymmetricBinding>` политики WSDL-элемента `<binding>`.

При изменении, с помощью кнопки **Настройка**, опции **Макет заголовка безопасности** изменяется дочерний элемент элемента `<sp:Layout>` политики WSDL-элемента `<binding>`. В случае выбора вариантов **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)** расположение дочерних элементов заголовка `<wsse:Security>` по факту не меняется. При выборе значения **Свободно (метка времени в последнюю очередь)** элемент `<wsu:Timestamp>` становится последним дочерним элементом заголовка `<wsse:Security>`. При этом Web-сервис может выдавать ошибку при попытке проверки цифровой подписи входящего сообщения из-за несовпадения ожидаемого порядка расположения дочерних элементов `<xenc:ReferenceList>` и `<ds:Signature>` заголовка `<wsse:Security>`.

В случае выбора кнопкой **Настройка** опции **Требуются производные ключи** в политике WSDL-элемента `<binding>` появится дочерний элемент `<sp:Require DerivedKeys/>` элемента `</sp:UsernameToken>`, устанавливая тем самым, что будет использоваться не сам производный ключ пароля, а его производные ключи. Соответственно в заголовке `<wsse:Security>` входящего и исходящего сообщений Web-сервиса появятся элементы `<wsc:DerivedKeyToken>`, содержащие информацию о производных ключах.

В случае нажатия кнопки **Настройка** и выбора опций **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса**, в политике WSDL-элемента `<binding>` появятся элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `<sp:BootstrapPolicy>` и `<sp:Trust13>`.

В этом случае для цифровой подписи и шифрования SOAP-сообщений будут использоваться производные ключи SCT-маркера защиты `<wsc:SecurityContext Token>`.

Для получения SCT-маркера клиент сначала посылает RST-запрос серверу. В ответ сервер посылает клиенту RSTR-ответ с SCT-маркером. При этом для шифрования и подписи тела и заголовков RST-запроса и RSTR-ответа используется производный ключ пароля.

После получения клиентом SCT-маркера устанавливается защищенная сессия, и входящее сообщение Web-сервиса содержит маркер защиты `<wsc:Security ContextToken>` в заголовке `<wsse:Security>`, а также элементы `<wsc:Derived KeyToken>`, описывающие производные ключи, полученные из маркера защиты `<wsc:SecurityContextToken>` и используемые для шифрования и подписи тела сооб-

щения и его заголовков. Исходящее сообщение Web-сервиса содержит элементы `<wsc:DerivedKeyToken>` в заголовке `<wsse:Security>`, описывающие производные ключи SCT-маркера, используемые для шифрования и подписи тела исходящего сообщения и его заголовков.

Опция **Protect Tokens** после нажатия кнопки **Настройка** в версии NetBeans 6.9.1 не поддерживается.

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** в окне **MetroTestService** на вкладке **Качество обслуживания** Web-сервиса, используя кнопку **Хранилище ключей**, можно переопределить расположение по умолчанию хранилища ключей и сертификатов сервера. Кнопкой **Доверительное хранилище** — переопределить расположение по умолчанию доверительного хранилища ключей и сертификатов сервера. Кнопкой **Дополнительно** — указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик имени/пароля во входящих сообщениях на стороне сервера, реализующий интерфейс `com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidator`, пользовательский обработчик метки времени, реализующий интерфейс `com.sun.xml.wss.impl.callback.TimestampValidationCallback.TimestampValidator`, пользовательский обработчик сертификатов, реализующий интерфейс `com.sun.xml.wss.impl.callback.CertificateValidationCallback.CertificateValidator`, и пользовательский обработчик SAML, реализующий интерфейс `com.sun.xml.wss.impl.callback.SAMLAssertionValidator`.

Опция *Безопасность совместных сертификатов*

В окне **MetroTest** на вкладке **Качество обслуживания** Web-сервиса, в опции **Механизм обеспечения безопасности** выберем **Безопасность совместных сертификатов**. Нажав кнопку **Настройка**, для опции **Набор алгоритмов** укажем **Basic 256bit**, а для параметра **Макет заголовка безопасности** выберем вариант **Строго** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\AsymmetricBinding\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sp:AsymmetricBinding>` политики WSDL-элемента `<binding>` заменил элемент `<sp:SymmetricBinding>`, определяя требования защиты сообщений с помощью цифровой подписи и шифрования с применением асимметричных ключей. Его дочерний элемент `<sp:InitiatorToken>` определяет требования цифровой подписи и шифрования к запрашивающей стороне, а дочерний элемент `<sp:RecipientToken>` — требования цифровой подписи и шифрования к запрашиваемой стороне.

Элемент `<sp:Wss10>` политики WSDL-элемента `<binding>` заменил элемент `<sp:Wss11>`, определяя поддержку свойств спецификации WS-Security версии 1.0.

Его дочерний элемент `<sp:MustSupportRefIssuerSerial/>` указывает поддержку ссылок на маркеры защиты с помощью элемента `<ds:X509IssuerSerial>`.

После развертывания проекта Metro_WebService обновим клиента Web-сервиса MetroTest в проекте Metro_WebClient, в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.

В окне **MetroTestService** на вкладке **Качество обслуживания** сбросим флажок **Использовать значения по умолчанию, указанные при разработке**. Для применения опции **Безопасность совместных сертификатов** не требуется имя/пароль авторизированного пользователя сервера, поэтому очистим строки **Имя пользователя по умолчанию** и **Пароль по умолчанию**. Нажмем кнопки **Хранилище ключей** и **Доверительное хранилище** и кнопку **OK**. Закроем окно **MetroTestService** нажатием кнопки **OK**.

В результате во WSIT-файле клиента Web-сервиса появится политика:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:KeyStore wspp:visibility="private" location="D:\Program
Files\glassfish-3.0.1\glassfish\domains\domain1\config\keystore.jks" type="JKS"
storepass="changeit" alias="xws-security-client"/>
      <sc:TrustStore wspp:visibility="private" location="D:\Program
Files\glassfish-3.0.1\glassfish\domains\domain1\config\cacerts.jks" type="JKS"
storepass="changeit" peeralias="xws-security-server"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

После запуска клиента Web-сервиса MetroTest с помощью кнопки **Отправить** страницы приветствия проекта Metro_WebClient между клиентом и Web-сервисом произойдет соответствующий обмен сообщениями.

При этом может быть выведено сообщение об ошибке:

```
Invalid key provided for encryption/decryption.
java.security.InvalidKeyException: Illegal key size or default parameters[/b]
```

Для устранения данной ошибки необходимо обновить файлы Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6, скачав файл jce_policy-6.zip по адресу

https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jce_policy-6-oth-JPR@CDS-CDS_Developer

и поместив его содержимое в папку \jre\lib\security каталога JRE.

Входящее сообщение Web-сервиса будет содержать элемент `<wsse:BinarySecurityToken>` заголовка `<wsse:Security>` с сертификатом клиента, расположенным по умолчанию в хранилище ключей keystore.jks сервера GlassFish под псевдонимом

xws-security-client. Тело входящего сообщения будет зашифровано публичным ключом сертификата сервера, который клиент возьмет в доверительном хранилище cacerts.jks под псевдонимом xws-security-server. Публичный ключ сертификата сервера будет описан элементом `<xenc:EncryptedKey>` заголовка `<wsse:Security>`. Заголовки входящего сообщения будут подписаны приватным ключом сертификата клиента.

После получения входящего сообщения сервер находит публичный ключ сертификата клиента в хранилище ключей keystore.jks и проверяет цифровую подпись заголовков входящего сообщения. Для расшифровки тела входящего сообщения сервер находит приватный ключ своего сертификата в хранилище ключей keystore.jks.

Исходящее сообщение Web-сервиса будет зашифровано в части его тела публичным ключом сертификата клиента и подписано в части заголовков и тела приватным ключом сертификата сервера.

При изменении с помощью кнопки **Настройка** опции **Набор алгоритмов** на вкладке **Качество обслуживания** в окне тестирования Web-сервиса MetroTest изменяется дочерний элемент элемента `<sp:AlgorithmSuite>`, являющегося, в свою очередь, дочерним элементом элемента `<sp:AsymmetricBinding>` политики WSDL-элемента `<binding>`.

При изменении с помощью кнопки **Настройка** опции **Макет заголовка безопасности** изменяется дочерний элемент элемента `<sp:AsymmetricBinding>\<sp:Layout>` политики WSDL-элемента `<binding>`. В случае выбора значения **Строго**, **Свободно**, **Свободно (метка времени в первую очередь)** расположение дочерних элементов заголовка `<wsse:Security>` по факту не меняется. При выборе варианта **Свободно (метка времени в последнюю очередь)** элемент `<wsu:Timestamp>` становится последним дочерним элементом заголовка `<wsse:Security>`. При этом Web-сервис может выдавать ошибку:

```
com.sun.xml.wss.impl.PolicyViolationException:
com.sun.xml.wss.XWSSecurityException: Encryption Policy verification error:
Looking for an Encryption Element  in Security header, but found
com.sun.xml.wss.impl.policy.mls.SignaturePolicy@17d0063.
```

из-за несовпадения ожидаемого порядка расположения дочерних элементов `<xenc:EncryptedKey>` и `<ds:Signature>` заголовка `<wsse:Security>`.

В случае нажатия кнопки **Настройка** и выбора опции **Подпись шифрования** в политике WSDL-элемента `<binding>` появится дочерний элемент `<sp:EncryptSignature/>` элемента `<sp:AsymmetricBinding>`, определяя, что цифровая подпись должна быть зашифрована. При этом дочерние элементы `<ds:Signature>` заголовка `<wsse:Security>` сообщений будут зашифрованы публичным ключом сертификата сервера во входящем сообщении Web-сервиса и публичным ключом сертификата клиента в исходящем сообщении Web-сервиса.

В случае нажатия кнопки **Настройка** и выбора опции **Зашифровать перед подписанием** в политике WSDL-элемента `<binding>` появится дочерний элемент `<sp:EncryptBeforeSigning/>` элемента `<sp:AsymmetricBinding>`, указывая, что цифровая подпись сообщения создается после его шифрования, при этом используются

асимметричные ключи. В этом случае во входящем и исходящем сообщениях Web-сервиса изменится порядок элементов `<ds:Signature>` и `<xenc:EncryptedKey>` так, что заработает опция **Свободно (метка времени в последнюю очередь)**.

При нажатии кнопки **Настройка** и выбора опций **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** в политике WSDL-элемента `<binding>` появятся элементы `<sp:SymmetricBinding>/<sp:SecureConversationToken>`, `<sp:SymmetricBinding>/<sp:SecureConversationToken>/<sp:RequireDerivedKeys/>`, `<sp:SymmetricBinding>/<sp:SecureConversationToken>/<sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>`.

В этом случае для цифровой подписи и шифрования SOAP-сообщений будут использоваться производные ключи SCT-маркера защиты `<wsc:SecurityContextToken>`, который клиент получит с помощью RST-запроса серверу. При этом для шифрования тела RST-запроса используется публичный ключ сертификата сервера, а для подписи тела и заголовков RST-запроса — приватный ключ сертификата клиента. Для шифрования тела RSTR-ответа применяется публичный ключ сертификата клиента, а для подписи тела и заголовков RSTR-ответа — приватный ключ сертификата сервера.

После получения клиентом SCT-маркера устанавливается защищенная сессия, и входящее сообщение Web-сервиса содержит маркер защиты `<wsc:SecurityContextToken>` в заголовке `<wsse:Security>`, а также элементы `<wsc:DerivedKeyToken>`, описывающие производные ключи, полученные из маркера защиты `<wsc:SecurityContextToken>` и используемые для шифрования и подписи тела входящего сообщения и его заголовков. Исходящее сообщение Web-сервиса содержит элементы `<wsc:DerivedKeyToken>` в заголовке `<wsse:Security>`, описывающие производные ключи SCT-маркера, используемые для шифрования и подписи тела исходящего сообщения и его заголовков.

Опция *Симметричная привязка к маркеру Kerberos*

При применении данной опции аутентификация клиента, подпись и шифрование сообщений производится сервером на основе Kerberos-билетов, выдаваемых службой Key Distribution Center (KDC), состоящей из двух логических частей — сервера Authentication Server (AS) и сервера Ticket Granting Server (TGS) — и содержащей базу данных секретных ключей клиентов и сервисов.

В случае использования протокола Kerberos клиент проходит аутентификацию в службе KDC в качестве авторизированного пользователя с помощью идентификатора, связанного с секретным ключом клиента, и получает от службы KDC первичный Kerberos-билет. При этом секретный ключ клиента используется для шифрования первичного Kerberos-билета службой KDC и для его расшифровки клиентом. Далее клиент использует первичный Kerberos-билет и идентификатор запрашиваемого сервиса для получения от службы KDC Kerberos-билетов, позволяющих клиенту получить доступ к запрашиваемому сервису.

Использование протокола Kerberos состоит из следующих этапов:

1. После получения идентификатора и пароля пользователя клиентская среда выполнения генерирует с помощью хэширования пароля секретный ключ клиента и посылает запрос, содержащий идентификатор, серверу AS для получения Kerberos-билета Ticket Granting Ticket (TGT).
2. Сервер AS находит в базе данных по идентификатору секретный ключ клиента и посылает клиенту два сообщения. Первое сообщение содержит сессионный ключ "клиент — TGT", зашифрованный секретным ключом клиента. Второе сообщение передает билет TGT с сессионным ключом "клиент — TGT", зашифрованный ключом сервера TGS.
3. Для получения доступа к сервису клиент посылает серверу TGS билет TGT, зашифрованный ключом сервера TGS, идентификатор сервиса, аутентификатор клиента, содержащий идентификатор клиента и зашифрованный сессионным ключом "клиент — TGT", который клиент расшифровал с помощью своего секретного ключа.
4. Сервер TGS извлекает сессионный ключ "клиент — TGT" из билета TGT, расшифровывает и проверяет аутентификатор клиента, по идентификатору сервиса находит секретный ключ сервиса и посылает клиенту два сообщения. Первое сообщение содержит сессионный ключ "клиент — сервер" для доступа к сервису, зашифрованный сессионным ключом "клиент — TGT". Второе сообщение передает Kerberos-билет с сессионным ключом "клиент — сервер", зашифрованный секретным ключом сервиса.
5. Клиент расшифровывает сессионный ключ "клиент — сервер" и посылает серверу сервиса Kerberos-билет, зашифрованный секретным ключом сервиса, и аутентификатор клиента, зашифрованный сессионным ключом "клиент — сервер".
6. Сервер извлекает из Kerberos-билета сессионный ключ "клиент — сервер", расшифровывает и проверяет аутентификатор клиента и отправляет подтверждение клиенту, зашифрованное сессионным ключом "клиент — сервер".
7. Защита дальнейшей сессии "клиент — сервер" обеспечивается с помощью сессионного ключа "клиент — сервер".

Для применения опции **Симметричная привязка к маркеру Kerberos** в операционной системе сервера GlassFish требуется наличие инсталлированной службы Kerberos KDC. Для семейства операционных систем Windows служба Kerberos KDC поставляется с операционными системами Window Server 2000, 2003 и 2008 как часть Active Directory.

Для начала работы со службой Kerberos KDC необходимо создать учетные записи клиента и сервиса в базе данных KDC (опция `addprinc` Kerberos-утилиты `kadmin`). При этом добавим учетную запись сервиса в файл /etc/krb5.keytab (опция `ktadd` Kerberos-утилиты `kadmin`). Далее Kerberos-утилитой `kinit` получим и кэшируем для клиента TGT-билет.

Определим логин-модули JAAS для сервера GlassFish, включив в конфигурационный файл /domains/domain1/config/login.conf каталога сервера GlassFish:

```
KerberosClient {
  com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
}
KerberosServer {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true keyTab="/etc/krb5.keytab"
  doNotPrompt=true storeKey=true principal="[идентификатор сервиса]";
}
```

В окне **MetroTest** на вкладке **Качество обслуживания** Web-сервиса, в опции **Механизм обеспечения безопасности** выберем **Симметричная привязка к маркеру Kerberos**. Нажав кнопку **Настройка**, для параметра **Набор алгоритмов** укажем **Basic 256bit**, а для параметра **Макет заголовка безопасности** выберем вариант **Строго** и нажмем кнопку **OK**. Кнопкой **Kerberos** в раскрывающемся списке **Модуль входа в систему** укажем вариант **KerberosServer** и нажмем кнопку **OK**.

В результате политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса примет следующий вид:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsam:Addressing wsp:Optional="false"/>
        <sc:KeyStore wspp:visibility="private"
                     location="D:\Program Files\glassfish-
3.0.1\glassfish\domains\domain1\config\keystore.jks"
                     type="JKS" storepass="changeit"
                     alias="xws-security-server"/>
        <sp:SymmetricBinding>
          <wsp:Policy>
            <sp:ProtectionToken>
              <wsp:Policy>
<sp:KerberosToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludToken/Once">
                <wsp:Policy>
                  <sp:WssGssKerberosV5ApReqToken11/>
                </wsp:Policy>
              </sp:KerberosToken>
            </wsp:Policy>
            </sp:ProtectionToken>
            <sp:Layout>
              <wsp:Policy>
                <sp:Strict/>
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp/>
            <sp:OnlySignEntireHeadersAndBody/>
            <sp:AlgorithmSuite>
```

```
            <wsp:Policy>
              <sp:Basic128/>
            </wsp:Policy>
          </sp:AlgorithmSuite>
        </wsp:Policy>
      </sp:SymmetricBinding>
      <sp:Wss11>
        <wsp:Policy>
          <sp:MustSupportRefIssuerSerial/>
          <sp:MustSupportRefThumbprint/>
          <sp:MustSupportRefEncryptedKey/>
        </wsp:Policy>
      </sp:Wss11>
      <sc:KerberosConfig wspp:visibility="private"
                         loginModule="KerberosServer"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Элемент `<sp:KerberosToken>` указывает включение в сообщение маркера защиты `<wsse:BinarySecurityToken>`, содержащего Kerberos-билет. Атрибут `sp:IncludeToken` элемента `<sp:KerberosToken>` своим значением определяет, что маркер защиты должен быть включен только в одно сообщение, посылаемое запрашиваемой стороне.

Дочерний элемент `<sp:WssGssKerberosV5ApReqToken11/>` указывает требование равенства атрибута `ValueType` элемента `<wsse:BinarySecurityToken>` соответствующему билету GSS Kerberos Version 5 AP-REQ.

Элемент `<sc:KerberosConfig>` определяет имя логин-модуля JAAS для конфигурации Kerberos Web-сервиса.

После обновления клиента Web-сервиса MetroTest в проекте Metro_WebClient щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.

На вкладке **Качество обслуживания** нажмем кнопку **Kerberos** и в строке **Модуль входа в систему** укажем **KerberosClient**, в строке **Участник службы** введем идентификатор сервиса, укажем **Делегирование параметров доступа** и нажмем кнопку **OK**.

В результате во WSIT-файле клиента Web-сервиса появится политика:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:KerberosConfig wspp:visibility="private"
                         loginModule="KerberosClient"
                         servicePrincipal="[идентификатор сервиса]"
                         credentialDelegation="true"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Здесь элемент `<sc:KerberosConfig>` определяет имя логин-модуля JAAS для конфигурации Kerberos клиента Web-сервиса. Атрибут `servicePrincipal` указывает идентификатор сервиса, для которого клиент запрашивает Kerberos-билет у службы Kerberos KDC. Атрибут `credentialDelegation="true"` указывает, что сервер может инициировать создание контекста безопасности для клиента.

После запуска клиента Web-сервиса MetroTest выбором пункта **Выполнить** проекта Metro_WebClient и нажатием кнопки **Отправить** страницы приветствия проекта Metro_WebClient между клиентом и Web-сервисом произойдет соответствующий обмен сообщениями.

Кнопка **Настройка** опции **Симметричная привязка к маркеру Kerberos** окна **MetroTest** на вкладке **Качество обслуживания** Web-сервиса позволяет переопределить:

- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>` политики WSDL-элемента `<binding>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>` политики WSDL-элемента `<binding>`);
- опцией **Требуются производные ключи** — определить использование не сессионного ключа "клиент — сервер" для шифрования и подписи сообщений, а его производного ключа (элемент `<sp:RequireDerivedKeys/>` политики WSDL-элемента `<binding>`);
- опциями **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** — определить использование для шифрования и подписи сообщений производные ключи SCT-маркера защиты `<wsc:SecurityContextToken>` (элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `</sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>` политики WSDL-элемента `<binding>`);
- опцией **Подпись шифрования** — установить шифрование цифровой подписи (элемент `<sp:EncryptSignature/>` политики WSDL-элемента `<binding>`);
- опцией **Зашифровать перед подписанием** — указать создание цифровой подписи после шифрования сообщения (элемент `<sp:EncryptBeforeSigning/>` политики WSDL-элемента `<binding>`).

Опция *Безопасность транспорта (SSL)*

Протокол SSL (Secure Sockets Layer) обеспечивает защиту передачи сообщения от клиента серверу и, наоборот, на транспортном уровне. То есть защита действует от момента отправки сообщения по транспортному протоколу и до момента прибытия сообщения. Сообщение становится незащищенным по прибытии адресату. Таким образом, применение опции **Безопасность транспорта (SSL)** обеспечивает создание защищенного транспортного канала обмена сообщениями между клиентом и сервером.

Защита передачи сообщения с помощью протокола SSL основывается на шифровании и подписи сообщения отправителем и расшифровки сообщения и проверки подписи получателем с применением публичного и приватного ключей сертификата. В случае применения одностороннего протокола SSL используется только сертификат сервера. В случае двухстороннего протокола SSL используется как сертификат сервера, так и сертификат клиента.

Для тестирования опции **Безопасность транспорта (SSL)** на вкладке **Качество обслуживания** окна Web-сервиса MetroTest, в опции **Механизм обеспечения безопасности** выберем **Безопасность транспорта (SSL)**. Нажав кнопку **Настройка**, выберем **Требуется сертификат клиента** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\SSL\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sp:TransportBinding>` политики WSDL-элемента `<binding>` требует защиты сообщений с помощью транспортного протокола. Его дочерний элемент `<sp:TransportToken>` определяет требования к транспортному протоколу с помощью элемента `<sp:HttpsToken>`, который требует использование HTTPS-протокола, указывая значением своего атрибута `RequireClientCertificate="true"`, что должна производиться аутентификация запрашивающей стороны с помощью сертификата.

Откроем узел **Веб-страницы | WEB-INF** проекта Metro_WebService и модифицируем дескриптор развертывания web.xml следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
<security-constraint>
  <display-name>SSL</display-name>
  <web-resource-collection>
    <web-resource-name>MetroTest</web-resource-name>
    <description/>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <description/>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
</login-config>
</web-app>
```

Элемент `<security-constraint>` вводит ограничение доступа к Web-сервису, основанное на использовании протокола SSL (элемент `<transport-guarantee>`) и аутентификации клиента с помощью его сертификата (элемент `<auth-method>`).

После развертывания проекта Metro_WebService в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Удалить**. Щелкнем правой кнопкой мыши на узле **Metro_WebClient** и выберем пункты **Создать | Клиент веб-службы**. В появившемся диалоговом окне установим переключатель **URL-адрес файла WSDL**, введем:

```
https://localhost:8181/Metro_WebService/MetroTestService?wsdl
```

и нажмем кнопку **Завершить** (рис. 4.11).

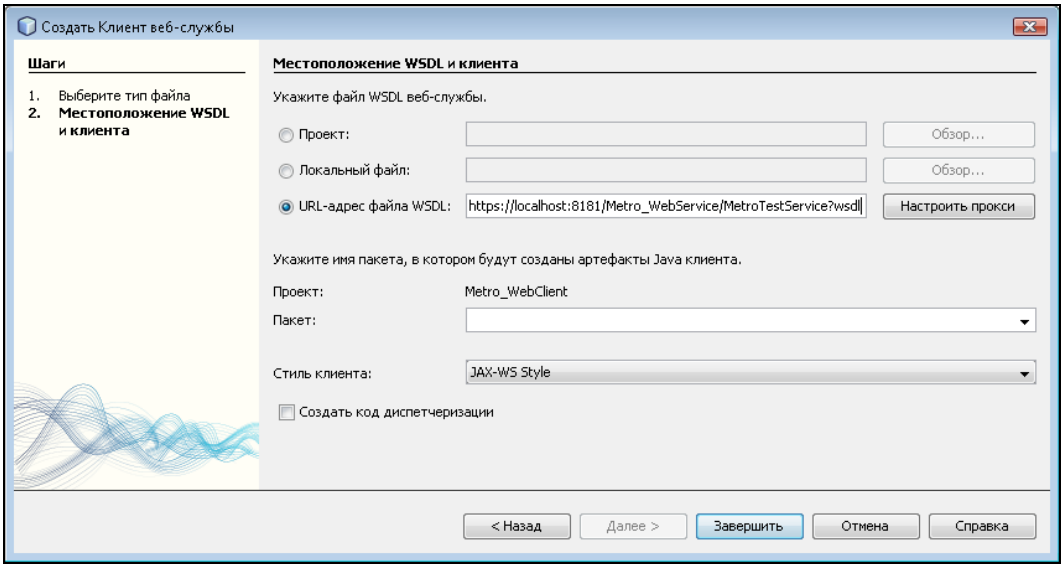


Рис. 4.11. Диалоговое окно создания клиента Web-сервиса

После запуска клиента Web-сервиса MetroTest выбором опции **Выполнить** проекта Metro_WebClient и нажатием кнопки **Отправить** страницы приветствия проекта Metro_WebClient между клиентом и Web-сервисом произойдет обмен сообщениями.

Входящее сообщение Web-сервиса:

```
INFO: ---[HTTP request]---
INFO: accept: text/xml, multipart/related
INFO: cache-control: no-cache
```

```
INFO: connection: keep-alive
INFO: content-length: 1304
INFO: content-type: text/xml; charset=utf-8
INFO: host: localhost:8181
INFO: pragma: no-cache
INFO: soapaction: "http://metrotest/MetroTest/getHelloRequest"
INFO: user-agent: JAX-WS RI 2.2.1-hudson-28-
INFO: <?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<S:Header>
<To
xmlns="http://www.w3.org/2005/08/addressing">https://localhost:8181/Metro_WebSe
rvice/MetroTestService</To>
<Action
xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/MetroTest/getHell
oRequest</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
    <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
</ReplyTo>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:ebe83f3d-1f77-
4fc3-abde-65b40c6bbf50</MessageID>
<wsse:Security S:mustUnderstand="1">
<wsu:Timestamp
xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity"
xmlns:ns14="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
xmlns:ns13="http://www.w3.org/2003/05/soap-envelope" wsu:Id="_1">
<wsu:Created>2011-02-11T02:22:31Z</wsu:Created>
<wsu:Expires>2011-02-11T02:27:31Z</wsu:Expires>
</wsu:Timestamp>
</wsse:Security>
</S:Header>
<S:Body>
<ns2:getHello xmlns:ns2="http://metrotest/">
<name></name>
</ns2:getHello>
</S:Body>
```

Исходящее сообщение Web-сервиса:

```
INFO: ---[HTTP response 200]---
INFO: <?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<S:Header>
<To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/
addressing/anonymous</To>
```

```
<Action xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/
MetroTest/getHelloResponse</Action>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:8eab1ed4-af20-
4838-a966-27bb6564e2c3</MessageID>
<RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:ebe83f3d-1f77-
4fc3-abde-65b40c6bbf50</RelatesTo>
<wsse:Security S:mustUnderstand="1">
<wsu:Timestamp
xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity"
xmlns:ns14="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
xmlns:ns13="http://www.w3.org/2003/05/soap-envelope" wsu:Id="_1">
<wsu:Created>2011-02-11T02:22:31Z</wsu:Created>
<wsu:Expires>2011-02-11T02:27:31Z</wsu:Expires>
</wsu:Timestamp>
</wsse:Security>
</S:Header>
<S:Body>
<ns2:getHelloResponse xmlns:ns2="http://metrotest/">
<return>Hello </return>
</ns2:getHelloResponse>
</S:Body>
</S:Envelope>
```

Видно, что по прибытии адресату сообщение никак не защищено.

По умолчанию, как клиент, так и сервер используют сертификат s1as (сокращенно Sun One Application Server) домена сервера GlassFish, находящийся в хранилище ключей \domains\domain1\config\keystore.jks и доверительном хранилище \domains\domain1\config\cacerts.jks сервера.

Для создания клиента Web-сервиса платформы Java SE создадим в среде NetBeans проект Java-приложения Metro_SEClient с главным классом `metro_seclient.Main`.

1. Щелкнем правой кнопкой мыши на узле **Metro_SEClient** и выберем пункты **Создать | Клиент веб-службы**. В появившемся диалоговом окне установим переключатель **Локальный файл**. Нажав кнопку **Обзор**, укажем предварительно сохраненный WSDL-документ Web-сервиса MetroTest и нажмем кнопку **Завершить**.

2. В коде класса `Main` щелкнем правой кнопкой мыши, выберем пункты **Вставить код | Вызов операции веб-службы | Metro_SEClient | MetroTestService | MetroTestPort | getHello** и нажмем кнопку **OK**.

   В результате код класса `Main` дополнится методом `getHello()`:

   ```
   private static String getHello(java.lang.String name) {
   metrotest.MetroTestService service = new metrotest.MetroTestService();
     metrotest.MetroTest port = service.getMetroTestPort();
     return port.getHello(name);
   }
   ```

3. Модифицируем код метода `main()` класса `Main` (листинг 4.4).

Листинг 4.4. Код метода `main()` класса `metro_seclient.Main`

```
public static void main(String[] args) {
  // Определяем доверительное хранилище
  System.setProperty("javax.net.ssl.trustStore",
    "D:\\Program Files\\glassfish-
3.0.1\\glassfish\\domains\\domain1\\config\\cacerts.jks");
  System.setProperty("javax.net.ssl.trustStorePassword", "changeit");
  // Определяем хранилище ключей
  System.setProperty("javax.net.ssl.keyStore",
    "D:\\Program Files\\glassfish-
3.0.1\\glassfish\\domains\\domain1\\config\\keystore.jks" );
  System.setProperty("javax.net.ssl.keyStorePassword", "changeit");
  // Вызываем Web-сервис
  System.out.println(getHello("User"));
}
```

Для применения протокола SSL необходимо перед вызовом Web-сервиса определить хранилище ключей с сертификатом клиента и доверительное хранилище с сертификатом сервера.

В окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле **Metro_SEClient | Библиотеки** и выберем пункт **Добавить библиотеку**. В появившемся диалоговом окне выберем библиотеку **Metro 2.0** и нажмем кнопку **Добавление библиотеки**.

Щелкнем правой кнопкой мыши на узле **Metro_SEClient** и выберем пункт **Выполнить**. В результате в окне **Вывод** среды NetBeans появится строка "Hello User", возвращаемая методом `getHello()` Web-сервиса MetroTest.

Опция *Проверка подлинности сообщения по SSL*

Опция **Проверка подлинности сообщения по SSL** добавляет к опции **Безопасность транспорта (SSL)** защиту входящих сообщений Web-сервиса с помощью включения в них маркера защиты `<wsse:UsernameToken>` или `<wsse:BinarySecurityToken>`.

Для тестирования данной опции на вкладке **Качество обслуживания** окна Web-сервиса MetroTest, в опции **Механизм обеспечения безопасности** выберем значение **Проверка подлинности сообщения по SSL**. Нажав кнопку **Настройка**, укажем для параметра **Маркер проверки подлинности** значение **Имя пользователя**, для параметра **Версия WSS** — значение **1.0**, для параметра **Набор алгоритмов** укажем **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Свободно** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\UsernameTokenSSL\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sp:SignedEncryptedSupportingTokens>` политики WSDL-элемента `<binding>` требует включения в сообщение одного или нескольких маркеров защиты, которые для защиты должны быть зашифрованы и снабжены цифровой подписью. В данном случае это маркер защиты `<wsse:UsernameToken>` (элемент `<sp:UsernameToken>`).

При отсутствии требования сертификата клиента (атрибут `RequireClientCertificate="false"` элемента `<sp:HttpsToken>`) дескриптор развертывания web.xml проекта Metro_WebService будет выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <security-constraint>
    <display-name>SSL</display-name>
    <web-resource-collection>
      <web-resource-name>MetroTest</web-resource-name>
      <description/>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
      <description/>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
</web-app>
```

После развертывания проекта Metro_WebService создадим для проекта Metro_WebClient клиента Web-сервиса MetroTestService, используя WSDL-описание по адресу **https://localhost:8181/Metro_WebService/MetroTestService?wsdl**.

В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**. В появившемся диалоговом окне на вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке**, кнопкой **Хранилище ключей** подтвердим настройку хранилища ключей keystore.jks, в строке **Имя пользователя по умолчанию** введем имя предварительно созданного пользователя сервера GlassFish, а в строке **Пароль по умолчанию** введем его пароль и нажмем кнопку **OK**.

В результате во WSIT-файл клиента будет включена политика:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
     <wsp:All>
```

```
        <sc:CallbackHandlerConfiguration wspp:visibility="private">
        <sc:CallbackHandler default="User" name="usernameHandler"/>
        <sc:CallbackHandler default="1234" name="passwordHandler"/>
        </sc:CallbackHandlerConfiguration>
        <sc:KeyStore wspp:visibility="private" storepass="changeit"
           type="JKS" location="D:\Program Files\glassfish-
3.0.1\glassfish\domains\domain1\config\keystore.jks"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

После запуска клиента Web-сервиса MetroTest выбором опции **Выполнить** проекта Metro_WebClient и нажатием кнопки **Отправить** страницы приветствия проекта Metro_WebClient между клиентом и Web-сервисом произойдет обмен сообщениями.

Входящее сообщение Web-сервиса:

```
INFO: ---[HTTP request]---
INFO: accept: text/xml, multipart/related
INFO: cache-control: no-cache
INFO: connection: keep-alive
INFO: content-length: 1767
INFO: content-type: text/xml; charset=utf-8
INFO: host: localhost:8181
INFO: pragma: no-cache
INFO: soapaction: "http://metrotest/MetroTest/getHelloRequest"
INFO: user-agent: JAX-WS RI 2.2.1-hudson-28-
INFO: <?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
<S:Header>
<To
xmlns="http://www.w3.org/2005/08/addressing">https://localhost:8181/Metro_WebSe
rvice/MetroTestService</To>
<Action
xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/MetroTest/getHell
oRequest</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
  <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
</ReplyTo>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:0f936e48-8b36-
40fd-b820-55bc536d7a8c</MessageID>
    <wsse:Security S:mustUnderstand="1">
```

```
      <wsu:Timestamp xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/
addressingidentity" xmlns:ns14="http://docs.oasis-open.org/ws-sx/ws-
secureconversation/200512" xmlns:ns13="http://www.w3.org/2003/05/soap-envelope"
wsu:Id="_1"><wsu:Created>2011-02-11T04:03:50Z</wsu:Created><wsu:Expires>2011-
02-11T04:08:50Z</wsu:Expires></wsu:Timestamp>
      <wsse:UsernameToken xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/
addressingidentity" xmlns:ns14="http://docs.oasis-open.org/ws-sx/ws-
secureconversation/200512" xmlns:ns13="http://www.w3.org/2003/05/soap-envelope"
wsu:Id="uuid_f76d7282-b0f5-4a27-b11d-7b2a6c5b3c93">
      <wsse:Username>User</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-username-token-profile-1.0#PasswordText">1234</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <ns2:getHello xmlns:ns2="http://metrotest/">
      <name></name>
    </ns2:getHello>
  </S:Body>
</S:Envelope>
```

Исходящее сообщение Web-сервиса:

```
INFO: ---[HTTP response 200]---
INFO: <?xml version='1.0' encoding='UTF-8'?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <S:Header>
<To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/
2005/08/addressing/anonymous</To>
<Action xmlns="http://www.w3.org/2005/08/addressing">http://metrotest/
MetroTest/getHelloResponse</Action>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:b8638d40-72b4-
48fa-8874-3b47729007a9</MessageID>
<RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:0f936e48-8b36-
40fd-b820-55bc536d7a8c</RelatesTo>
    <wsse:Security S:mustUnderstand="1">
      <wsu:Timestamp xmlns:ns15="http://schemas.xmlsoap.org/ws/2006/02/
addressingidentity" xmlns:ns14="http://docs.oasis-open.org/ws-sx/ws-
secureconversation/200512" xmlns:ns13="http://www.w3.org/2003/05/soap-envelope"
wsu:Id="_1">
      <wsu:Created>2011-02-11T04:03:50Z</wsu:Created>
      <wsu:Expires>2011-02-11T04:08:50Z</wsu:Expires>
      </wsu:Timestamp>
    </wsse:Security>
  </S:Header>
  <S:Body>
    <ns2:getHelloResponse xmlns:ns2="http://metrotest/">
```

```
    <return>Hello </return>
   </ns2:getHelloResponse>
  </S:Body>
</S:Envelope>
```

Во входящее сообщение Web-сервиса в соответствии с его политикой включен маркер защиты `<wsse:UsernameToken>` с дочерними элементами `<wsse:Username>` и `<wsse:Password>`, содержащими имя и пароль зарегистрированного пользователя сервера.

При изменении, с помощью кнопки **Настройка**, на вкладке **Качество обслуживания** окна Web-сервиса MetroTest опцией **Маркер проверки подлинности** со значением **X509** в политику WSDL-элемента `<binding>` WSIT-файла Web-сервиса вместо элемента `<sp:SignedEncryptedSupportingTokens>` будет включен элемент:

```
<sp:EndorsingSupportingTokens>
<wsp:Policy>
  <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
  <wsp:Policy>
    <sp:WssX509V3Token10/>
  </wsp:Policy>
</sp:X509Token>
</wsp:Policy>
</sp:EndorsingSupportingTokens>
```

Элемент `<sp:EndorsingSupportingTokens>` политики WSDL-элемента `<binding>` требует включения в сообщение одного или нескольких маркеров защиты, имеющих ключи, которые используются для цифровой подписи первоначальной цифровой подписи. В данном случае это маркер защиты `<wsse:BinarySecurityToken>` (элемент `<sp:X509Token>`).

В случае применения опции **Маркер проверки подлинности** со значением **X509** входящее сообщение Web-сервиса содержит дочерний элемент `<wsse:Binary SecurityToken>` заголовка `<wsse:Security>` с данными сертификата клиента и дочерний элемент `<ds:Signature>` заголовка `<wsse:Security>` с вторичной цифровой подписью.

Кнопка **Настройка** опции **Проверка подлинности сообщения по SSL** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить:

- опцией **Версия WSS** — поддержку свойств спецификации WS-Security версии 1.0 или 1.1 (элемент `<sp:Wss10/>` или `<sp:Wss11/>` политики WSDL-элемента `<binding>`);
- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>` политики WSDL-элемента `<binding>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>` политики WSDL-элемента `<binding>`);

- опциями **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** — определить использование для шифрования и подписи сообщений производные ключи SCT-маркера защиты `<wsc:SecurityContextToken>` (элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `</sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>` политики WSDL-элемента `<binding>`);
- опцией **Версия WSS** со значением **1.1** и опцией **Требуется подтверждение подписи** — определить включение в исходящее сообщение Web-сервиса дочернего элемента `<wsse11:SignatureConfirmation>` заголовка `<wsse:Security>` с вторичной цифровой подписью (значение атрибута `Value`, опция **X509**) или без нее (опция **Имя пользователя**).

Установив флажок **Использовать значения по умолчанию, указанные при разработке** на вкладке **Качество обслуживания** окна Web-сервиса, нажав кнопку **Доверительное хранилище**, можно переопределить расположение по умолчанию доверительного хранилища сертификата клиента. Кнопкой **Дополнительно** указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик имени/пароля во входящих сообщениях на стороне сервера, реализующий интерфейс `com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidator`, и пользовательский обработчик сертификатов, реализующий интерфейс `com.sun.xml.wss.impl.callback.CertificateValidationCallback.CertificateValidator`.

Опция *Проверка подлинности SAML по SSL*

Применение опции **Проверка подлинности SAML по SSL** добавляет во входящее сообщение Web-сервиса SAML-утверждение об аутентификации клиента. Передача сообщений при этом осуществляется по защищенному транспортному каналу в соответствии с протоколом SSL.

В данном случае SAML-утверждение клиентского запроса содержит информацию, подтверждающую и описывающую факт аутентификации клиента в центре аутентификации, которому доверяет сервер GlassFish. На основании информации, содержащейся в SAML-утверждении, сервер разрешает доступ клиенту к Web-сервису.

Для тестирования данной опции на вкладке **Качество обслуживания** окна Web-сервиса MetroTest, в опции **Механизм обеспечения безопасности** выберем **Проверка подлинности SAML по SSL**. Нажав кнопку **Настройка**, для параметра **Версия SAML** укажем **1.0 (профиль 1.0)**, для параметра **Версия WSS** — значение 1.0, для параметра **Набор алгоритмов** — значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — **Свободно**, укажем **Требуется сертификат клиента** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\SAMLSSL\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sp:SignedSupportingTokens>` политики WSDL-элемента `<binding>` требует включения в сообщение одного или нескольких маркеров защиты, которые для защиты должны быть снабжены цифровой подписью. В данном случае это маркер защиты `<saml:Assertion>` (элемент `<sp:SamlToken>`).

Чтобы соответствовать такой политики Web-сервиса, дескриптор развертывания web.xml проекта Metro_WebService должен выглядеть следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
<security-constraint>
  <display-name>SSL</display-name>
  <web-resource-collection>
    <web-resource-name>MetroTest</web-resource-name>
    <description/>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <description/>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
</login-config>
</web-app>
```

После развертывания проекта Metro_WebService создадим для проекта Metro_WebClient клиента Web-сервиса MetroTestService, используя WSDL-описание по адресу **https://localhost:8181/Metro_WebService/MetroTestService?wsdl**.

В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**. В появившемся диалоговом окне на вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и нажмем кнопку **OK**.

В результате средой NetBeans будет сгенерирован WSIT-файл клиента Web-сервиса, содержащий политику:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
<wsp:ExactlyOne>
```

```
  <wsp:All>
    <sc:CallbackHandlerConfiguration wspp:visibility="private">
    <sc:CallbackHandler name="samlHandler"
          classname="samlcb.Saml11SVCallbackHandler"/>
    </sc:CallbackHandlerConfiguration>
  </wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Элемент `<sc:CallbackHandler>` указывает класс реализации интерфейса `javax.security.auth.callback.CallbackHandler`, который будет отвечать за включение SAML-утверждения в клиентский запрос серверу.

Средой NetBeans также будет сгенерирован сам класс `samlcb.Saml11SVCallbackHandler`, отображаемый узлом **Пакеты исходных файлов | samlcb | Saml11SV CallbackHandler.java** проекта Metro_WebClient.

Для успешной компиляции класса `Saml11SVCallbackHandler` в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient** и выберем пункты **Свойства | Библиотеки**. На вкладке **Компиляция** нажмем кнопку **Добавить библиотеку** и добавим библиотеку **Metro 2.0**, установим переключатель **Пакет** для предотвращения пакетирования библиотеки в проект Metro_WebClient, т. к. сервер GlassFish изначально с ней интегрирован.

В коде класса `Saml11SVCallbackHandler` задокументируем запрещенные методы:

```
//  assertion.setMajorVersion(BigInteger.ONE);
//  assertion.setMinorVersion(BigInteger.ONE);
```

Выбором **Очистить и построить | Выполнить** запустим проект Metro_WebClient и кнопкой **Отправить** страницы приветствия вызовем Web-сервис MetroTest.

В результате между клиентом и Web-сервисом произойдет обмен сообщениями, при этом входящее сообщение Web-сервиса будет содержать маркер защиты `<saml:Assertion>` с дочерними элементами `<saml:Conditions>` и `<saml:Attribute Statement>`.

Кнопка **Настройка** опции **Проверка подлинности SAML по SSL** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить:

- опцией **Версия SAML** — поддержку комбинации спецификации SAML Token Profile версий 1.0 и 1.1 и стандарта SAML версий 1.0, 1.1 и 2.0 (дочерний элемент элемента `<sp:SamlToken>`). При этом будет изменяться генерируемый средой NetBeans класс `SamlCallbackHandler`;
- опцией **Версия WSS** — поддержку свойств спецификации WS-Security версии 1.0 или 1.1 (элемент `<sp:Wss10/>` или `<sp:Wss11/>` политики WSDL-элемента `<binding>`);
- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>`);

- опцией **Версия WSS** со значением **1.1** и опцией **Требуется подтверждение подписи** — определить включение в исходящее сообщение Web-сервиса дочернего элемента `<wsse11:SignatureConfirmation>` заголовка `<wsse:Security>`.

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** на вкладке **Качество обслуживания** окна Web-сервиса, нажав кнопку **Дополнительно**, можно указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик SAML на стороне сервера, реализующий интерфейс `com.sun.xml.wss.impl.callback.SAMLAssertionValidator`.

Опция *Одобрение сертификата*

При применении опции **Одобрение сертификата** для шифрования и подписи тела сообщений и подписи заголовков сообщений используется сертификат сервера. При этом первоначальная подпись тела и заголовков входящего сообщения Web-сервиса подписывается вторичной подписью с использованием одобренного сервером сертификата клиента, тем самым обеспечивается идентификация и авторизация клиента.

Для тестирования данной опции на вкладке **Качество обслуживания** окна Web-сервиса MetroTest, в опции **Механизм обеспечения безопасности** выберем **Одобрение сертификата**. Нажав кнопку **Настройка**, для параметра **Набор алгоритмов** укажем значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Свободно** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\Endorsing\Metro_WebService\web\WEB-INF компакт-диска.

После развертывания проекта Metro_WebService создадим для проекта Metro_WebClient клиента Web-сервиса MetroTestService, используя кнопку **Обзор** диалогового окна **Создать Клиент веб-службы**.

1. В окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.
2. В появившемся диалоговом окне на вклдадке **Качество обслуживания** сбросим флажок **Использовать значения по умолчанию, указанные при разработке** и подтвердим расположение хранилища сертификата клиента и расположение доверительного хранилища сертификата сервера с помощью кнопок **Хранилище ключей** и **Доверительное хранилище**. Закроем окно нажатием кнопки **OK**.

В результате во WSIT-файле клиента Web-сервиса появится политика:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
```

```
      <sc:KeyStore wspp:visibility="private" location="D:\Program
           Files\glassfish-3.0.1\glassfish\domains\domain1\config\keystore.jks"
        type="JKS" storepass="changeit" alias="xws-security-client"/>
      <sc:TrustStore wspp:visibility="private" location="D:\Program
            Files\glassfish-3.0.1\glassfish\domains\domain1\config\cacerts.jks"
        type="JKS" storepass="changeit" peeralias="xws-security-server"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

После запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом и Web-сервисом произойдет обмен сообщениями, при этом входящее сообщение Web-сервиса будет содержать элемент `<xenc:EncryptedKey>`, представляющий публичный ключ сертификата сервера, которым будет зашифровано и подписано тело входящего сообщения и подписаны его заголовки, и элемент `<wsse:BinarySecurityToken>` с сертификатом клиента, приватным ключом которого будет снабжена первоначальная подпись тела и заголовков входящего сообщения. С помощью публичного ключа сертификата сервера также будет зашифровано и подписано тело исходящего сообщения Web-сервиса и подписаны его заголовки.

Таким образом, в данном случае публичный ключ сертификата сервера будет использован в качестве симметричного ключа шифрования и подписи входящего и исходящего сообщений Web-сервиса.

Кнопка **Настройка** опции **Одобрение сертификата** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить:

- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>`);
- опцией **Требуются производные ключи** — определить использование не публичного ключа сертификата сервера для шифрования и подписи сообщений, а его производных ключей (элемент `<sp:SymmetricBinding>`/ `<sp:ProtectionToken>`/ `<sp:X509Token>`/`<wsp:Policy>`/`<sp:RequireDerivedKeys/>` политики WSDL-элемента `<binding>`);
- опциями **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** — определить использование для шифрования и подписи сообщений производные ключи SCT-маркера защиты `<wsc:SecurityContextToken>` (элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `</sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>` политики WSDL-элемента `<binding>`);
- опцией **Требуется подтверждение подписи** — определить включение в исходящее сообщение Web-сервиса дочернего элемента `<wsse11:Signature Confirmation>` заголовка `<wsse:Security>` с первоначальной подписью входящего сообщения;

- опцией **Подпись шифрования** — установить шифрование цифровой подписи публичным ключом сертификата сервера (элемент `<sp:EncryptSignature/>` политики WSDL-элемента `<binding>`);
- опцией **Зашифровать перед подписанием** — указать создание цифровой подписи после шифрования сообщения (элемент `<sp:EncryptBeforeSigning/>`).

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** на вкладке **Качество обслуживания** окна Web-сервиса, нажав кнопку **Хранилище ключей**, можно переопределить расположение по умолчанию хранилища ключей и сертификатов сервера. Кнопкой **Доверительное хранилище** — переопределить расположение по умолчанию доверительного хранилища ключей и сертификатов сервера. Кнопкой **Дополнительно** — указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик сертификатов, реализующий интерфейс `com.sun.xml.wss.impl.callback.CertificateValidationCallback.CertificateValidator`.

Опция *Подтверждение подлинности отправителя SAML сертификатом*

В случае применения данной опции клиент включает в свой запрос Web-сервису SAML-утверждение, обеспечивающее ему авторизацию на сервере. Тело клиентского запроса, а также SAML-утверждение шифруется с помощью публичного ключа сертификата сервера. Подпись клиентского запроса осуществляется с помощью приватного ключа сертификата клиента, представленного в клиентском запросе маркером защиты `<wsse:BinarySecurityToken>`. Ответное сообщение от Web-сервиса подписывается приватным ключом сертификата сервера, а его тело шифруется публичным ключом сертификата клиента.

Для тестирования данной опции на вкладке **Качество обслуживания** окна Web-сервиса MetroTest в опции **Механизм обеспечения безопасности** выберем **Подтверждение подлинности отправителя SAML сертификатом**. Нажав кнопку **Настройка**, для параметра **Версия SAML** укажем значение **1.0 (профиль 1.0)**, для параметра **Набор алгоритмов** — значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Строго** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\SamlSVHandler\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sp:AsymmetricBinding>` определяет защиту сообщений с использованием асимметричных ключей, причем в данном случае сертификат клиента должен быть представлен во входящем сообщении Web-сервиса соответствующим маркером защиты (элемент `<sp:InitiatorToken>`/`<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">`), а на сертификат сервера можно только ссылаться (элемент `<sp:RecipientToken>`/

`<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">`).

Элемент `<sp:SignedEncryptedSupportingTokens>` своим дочерним элементом `<sp:SamlToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">` требует включение во входящее сообщение Web-сервиса SAML-утверждения, которое должно быть зашифровано и подписано цифровой подписью.

Политика сообщений Web-сервиса примет следующий вид:

```
<wsp:Policy wsu:Id="">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:EncryptedParts>
        <sp:Body/>
      </sp:EncryptedParts>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

1. После обновления клиента Web-сервиса MetroTest в проекте Metro_WebClient щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.
2. В появившемся диалоговом окне на вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и нажмем кнопку **OK**.

В результате средой NetBeans будет сгенерирован WSIT-файл клиента Web-сервиса, содержащий политику:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:KeyStore wspp:visibility="private" location="D:\Program
Files\glassfish-3.0.1\glassfish\domains\domain1\config\keystore.jks" type="JKS"
storepass="changeit" alias="xws-security-client"/>
      <sc:TrustStore wspp:visibility="private" location="D:\Program
Files\glassfish-3.0.1\glassfish\domains\domain1\config\cacerts.jks" type="JKS"
storepass="changeit" peeralias="xws-security-server"/>
      <sc:CallbackHandlerConfiguration wspp:visibility="private">
      <sc:CallbackHandler name="samlHandler"
           classname="samlcb.Saml11SVCallbackHandler"/>
      </sc:CallbackHandlerConfiguration>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Элемент `<sc:CallbackHandler>` указывает класс реализации интерфейса `javax.security.auth.callback.CallbackHandler`, отвечающий за включение SAML-утверждения в клиентский запрос серверу. Данный класс `samlcb.Saml11SVCallbackHandler`, отображаемый узлом **Пакеты исходных файлов | samlcb | Saml11SVCallbackHandler.java** проекта Metro_WebClient, также будет сгенерирован средой NetBeans.

Для успешной компиляции класса `Saml11SVCallbackHandler` в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient** и выберем **Свойства | Библиотеки**. На вкладке **Компиляция** кнопкой **Добавить библиотеку** добавим библиотеку **Metro 2.0** и отменим переключатель **Пакет** для предотвращения пакетирования библиотеки в проект Metro_WebClient, т. к. сервер GlassFish изначально с ней интегрирован.

В коде класса `Saml11SVCallbackHandler` задокументируем запрещенные методы:

```
//  assertion.setMajorVersion(BigInteger.ONE);
//  assertion.setMinorVersion(BigInteger.ONE);
```

После запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом и Web-сервисом произойдет обмен сообщениями, при этом входящее сообщение Web-сервиса в заголовке `<wsse:Security>` будет содержать элементы:

- `<xenc:EncryptedKey>` — представляет публичный ключ сертификата сервера;
- `<xenc:EncryptedData>` — представляет SAML-утверждение, зашифрованное публичным ключом сертификата сервера;
- `<wsse:BinarySecurityToken>` — представляет сертификат клиента.

Тело входящего сообщения Web-сервиса будет зашифровано публичным ключом сертификата сервера, а его заголовки, включая SAML-утверждение, и тело подписаны приватным ключом сертификата клиента.

Тело исходящего сообщения Web-сервиса будет зашифровано публичным ключом сертификата клиента, а его заголовки и тело подписаны приватным ключом сертификата сервера.

Кнопка **Настройка** опции **Подтверждение подлинности отправителя SAML сертификатом** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить:

- опцией **Версия SAML** — поддержку комбинации спецификации SAML Token Profile версий 1.0 и 1.1 и стандарта SAML версий 1.0, 1.1 и 2.0 (дочерний элемент элемента `<sp:SamlToken>`). При этом будет изменяться генерируемый средой NetBeans класс `SamlCallbackHandler`;
- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>`);
- опцией **Требуются производные ключи** — определить использование не публичного ключа сертификата сервера для шифрования входящего сообщения

Web-сервиса и не публичного ключа сертификата клиента для шифрования исходящего сообщения Web-сервиса, а их производных ключей (элементы `<sp:X509Token>`/`<sp:RequireDerivedKeys/>`);

- опциями **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** — определить использование для шифрования и подписи сообщений производные ключи SCT-маркера защиты `<wsc:SecurityContextToken>` (элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `</sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>` политики WSDL-элемента `<binding>`);
- опцией **Подпись шифрования** — установить шифрование цифровой подписи публичным ключом сертификата сервера во входящем сообщении Web-сервиса и шифрование цифровой подписи публичным ключом сертификата клиента в исходящем сообщении Web-сервиса (элемент `<sp:EncryptSignature/>` политики WSDL-элемента `<binding>`);
- опцией **Зашифровать перед подписанием** — указать создание цифровой подписи после шифрования сообщения (элемент `<sp:EncryptBeforeSigning/>`).

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** на вкладке **Качество обслуживания** окна Web-сервиса, используя кнопку **Хранилище ключей**, можно переопределить расположение по умолчанию хранилища ключей и сертификатов сервера. Кнопкой **Доверительное хранилище** — переопределить расположение по умолчанию доверительного хранилища ключей и сертификатов сервера. Кнопкой **Дополнительно** — указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик SAML на стороне сервера, реализующий интерфейс `com.sun.xml.wss.impl.callback.SAMLAssertionValidator`.

Опция *Держатель ключа SAML*

При применении данной опции клиентский запрос Web-сервису включает в себя подписанное цифровой подписью SAML-утверждение, причем цифровая подпись и информация о публичном ключе цифровой подписи SAML-утверждения содержатся в самом SAML-утверждении. Таким образом, авторизация клиента производится на основании цифровой подписи SAML-утверждения. Тело клиентского запроса шифруется с помощью публичного ключа сертификата сервера, а его заголовки и тело подписываются приватным ключом цифровой подписи SAML-утверждения. Тело ответного сообщения от Web-сервиса шифруется публичным ключом цифровой подписи SAML-утверждения, а его заголовки и тело подписываются приватным ключом сертификата сервера.

Для тестирования данной опции на вкладке **Качество обслуживания** окна Web-сервиса MetroTest в опции **Механизм обеспечения безопасности** выберем **Держатель ключа SAML**. Нажав кнопку **Настройка**, для параметра **Версия SAML** укажем значение **1.0 (профиль 1.0)**, для параметра **Набор алгоритмов** — значение

Basic 128bit, для параметра **Макет заголовка безопасности** — значение **Строго** и дважды нажмем кнопку **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\SamlHOKHandler\Metro_WebService\web\WEB-INF компакт-диска.

Элемент `<sp:AsymmetricBinding>` определяет защиту сообщений с использованием асимметричных ключей, причем в данном случае асимметричные ключи предоставляются SAML-утверждением (элемент `<sp:InitiatorToken>`/`<sp:SamlToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">`) и сертификатом сервера (элемент `<sp:RecipientToken>`/`<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Never">`).

1. После обновления клиента Web-сервиса MetroTest в проекте Metro_WebClient щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.
2. В появившемся диалоговом окне на вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и нажмем кнопку **OK**.

В результате средой NetBeans будет сгенерирован WSIT-файл клиента Web-сервиса, содержащий политику:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:KeyStore wspp:visibility="private" location="D:\Program
Files\glassfish-3.0.1\glassfish\domains\domain1\config\keystore.jks" type="JKS"
storepass="changeit" alias="xws-security-client"/>
      <sc:TrustStore wspp:visibility="private" location="D:\Program
Files\glassfish-3.0.1\glassfish\domains\domain1\config\cacerts.jks" type="JKS"
storepass="changeit" peeralias="xws-security-server"/>
      <sc:CallbackHandlerConfiguration wspp:visibility="private">
      <sc:CallbackHandler name="samlHandler"
          classname="samlcb.Saml11HOKCallbackHandler"/>
      </sc:CallbackHandlerConfiguration>
    </wsp:All>
  </wsp:ExactlyOne>

</wsp:Policy>
```

Элемент `<sc:CallbackHandler>` указывает класс реализации интерфейса `javax.security.auth.callback.CallbackHandler`, отвечающий за включение SAML-утверждения с ключом в клиентский запрос серверу. Данный класс `samlcb.Saml11HOKCallbackHandler`, отображаемый узлом **Пакеты исходных файлов | samlcb | Saml11HOKCallbackHandler.java** проекта Metro_WebClient, автоматически создается средой NetBeans при конфигурировании атрибутов ссылки на Web-сервис на стороне клиента.

1. Для успешной компиляции класса `Saml11HOKCallbackHandler` в окне **Проекты** щелкнем правой кнопкой мыши на узле **Metro_WebClient** и выберем пункты **Свойства | Библиотеки**. На вкладке **Компиляция** кнопкой **Добавить библиотеку** добавим библиотеку **Metro 2.0** и отменим переключатель **Пакет** для предотвращения пакетирования библиотеки в проект Metro_WebClient, т. к. сервер GlassFish изначально с ней интегрирован.
2. В коде класса `Saml11HOKCallbackHandler` задокументируем запрещенные методы:

```
//  assertion.setMajorVersion(BigInteger.ONE);
//  assertion.setMinorVersion(BigInteger.ONE);
```

После запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом и Web-сервисом произойдет обмен сообщениями, при этом входящее сообщение Web-сервиса в заголовке `<wsse:Security>` будет содержать элементы:

- `<saml:Assertion>` — SAML-утверждение с публичным ключом (дочерний элемент `<ds:KeyInfo>`) и цифровой подписью SAML-утверждения (дочерний элемент `<ds:Signature>`);
- `<xenc:EncryptedKey>` — публичный ключ сертификата сервера.

Тело входящего сообщения Web-сервиса будет зашифровано публичным ключом сертификата сервера, а его заголовки и тело подписаны приватным ключом цифровой подписи SAML-утверждения. Тело исходящего сообщения Web-сервиса будет зашифровано публичным ключом цифровой подписи SAML-утверждения, а его заголовки и тело подписаны приватным ключом сертификата сервера.

Кнопка **Настройка** опции **Держатель ключа SAML** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить:

- опцией **Версия SAML** — поддержку комбинации спецификации SAML Token Profile версий 1.0 и 1.1 и стандарта SAML версий 1.0, 1.1 и 2.0 (дочерний элемент элемента `<sp:SamlToken>`). При этом будет изменяться генерируемый средой NetBeans класс `SamlCallbackHandler`;
- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>`);
- опциями **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** — определить использование для шифрования и подписи сообщений производные ключи SCT-маркера защиты `<wsc:SecurityContextToken>` (элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `</sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>` политики WSDL-элемента `<binding>`);
- опцией **Подпись шифрования** — установить шифрование цифровой подписи публичным ключом сертификата сервера во входящем сообщении Web-сервиса и шифрование цифровой подписи публичным ключом цифровой подписи

SAML-утверждения в исходящем сообщении Web-сервиса (элемент `<sp:EncryptSignature/>`);

- опцией **Зашифровать перед подписанием** — указать создание цифровой подписи после шифрования сообщения (элемент `<sp:EncryptBeforeSigning/>`).

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** на вкладке **Качество обслуживания** окна Web-сервиса, нажав кнопку **Хранилище ключей**, можно переопределить расположение по умолчанию хранилища ключей и сертификатов сервера. Кнопкой **Доверительное хранилище** — переопределить расположение по умолчанию доверительного хранилища ключей и сертификатов сервера. Кнопкой **Дополнительно** — указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик SAML на стороне сервера, реализующий интерфейс `com.sun.xml.wss.impl.callback.SAMLAssertionValidator`.

Опция *Выпущенный STS маркер*

При применении данной опции клиент Web-сервиса получает SAML-маркер у центра аутентификации, представленного Web-приложением, которое реализует сервис Security Token Service (STS). После получения SAML-маркера от STS-сервиса клиент включает его в свой запрос Web-сервису и проходит на запрашиваемом сервере, доверяющему данному STS-сервису, авторизацию доступа к Web-сервису с помощью SAML-маркера. В результате клиент получает от Web-сервиса ответ на свой запрос. При этом SAML-маркер содержит информацию о том, что клиент успешно прошел аутентификацию у доверенного STS-сервиса.

Таким образом, в данном случае требуется наличие Web-приложения, представляющего доверенный STS-сервис.

Для создания STS-сервиса:

1. В меню **Файл** среды NetBeans выберем пункты **Создать проект | Java Web | Веб-приложение**, нажмем кнопку **Далее**, введем имя проекта STSProject и нажмем кнопку **Далее**, а затем кнопку **Завершить**.
2. В окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле **STSProject** и выберем **Создать | Другие | Веб-службы | Служба маркеров безопасности (STS)**, нажмем кнопку **Далее**, введем имя сервиса STSExample и имя пакета stsexample и нажмем кнопку **Завершить**.

В результате средой NetBeans будет создан STS-сервис STSExample.

STS-сервис будет реализован как Provider-класс, промаркированный аннотацией `@WebServiceProvider`, расширяющий класс `com.sun.xml.ws.security.trust.sts.BaseSTSImpl` и реализующий интерфейс `javax.xml.ws.Provider<Source>`.

Класс `BaseSTSImpl` также является Provider-классом, метод `invoke()` которого отвечает за разбор элемента `<wst:RequestSecurityToken>` тела RST-запроса SAML-

маркера защиты и формирование элемента `<wst:RequestSecurityTokenResponse>` RSTR-ответа, содержащего запрашиваемый SAML-маркер.

Класс `STSExample` вносит конкретные детали своей реализацией класса `BaseSTSImpl` в виде имени Service-класса, имени порта, целевого пространства имен и адреса WSDL-описания STS-сервиса.

Щелкнув правой кнопкой мыши на узле **STSProject | Веб-службы | STSExample-Service** и выбрав пункт **Правка атрибутов веб-службы**, можно увидеть, что в появившемся окне на вкладке **Качество обслуживания** при установленном флажке **Служба безопасности** для параметра **Механизм обеспечения безопасности** по умолчанию выбраны опции: **Проверка подлинности имени пользователя с помощью симметричного ключа**, для параметра **Маркер проверки подлинности** — значение **Имя пользователя**, для параметра **Набор алгоритмов** — значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Свободно**, для параметра **Требуются производные ключи** — значение **Подпись шифрования**.

Данный набор опций означает, что для аутентификации и получения SAML-маркера у STS-сервиса клиент должен предварительно зарегистрировать на сервере свой логин/пароль и включить в RST-запрос маркер защиты `<wsse:Username Token>` с логином и паролем. При этом маркер защиты `<wsse:UsernameToken>` должен быть зашифрован симметричным ключом, сгенерированным средой выполнения клиента. Симметричный ключ, содержащийся в RST-запросе, в свою очередь, шифруется с помощью публичного ключа сертификата сервера, хранящегося в доверительном хранилище клиента. Тело RST-запроса и цифровая подпись его заголовков и тела шифруются с помощью производного ключа, полученного из симметричного ключа клиента. Тело RST-запроса и его заголовки подписываются также с помощью производного ключа от симметричного ключа клиента. Тело RSTR-ответа и цифровая подпись его заголовков и тела шифруются, тело RSTR-ответа и его заголовки подписываются с помощью производных ключей от симметричного ключа клиента.

Уточнить размещение приватного ключа сертификата сервера, с помощью которого сервер будет расшифровывать симметричный ключ клиента, можно кнопкой **Хранилище ключей** на вкладке **Качество обслуживания** окна STS-сервиса.

Также при установленном флажке **Служба безопасности** окна **Качество обслуживания** STS-сервиса по умолчанию отмечен флажок **Использовать в качестве службы маркеров безопасности (STS)**. Нажав кнопку **Настроить** данной опции, можно увидеть диалоговое окно настройки STS-сервиса (рис. 4.12).

В поле **Источник** указан идентификатор STS-сервиса, в поле **Класс реализации контракта** — класс, отвечающий за выпуск, обновление, удаление и проверку маркера безопасности. Поле **Время жизни выпущенных маркеров (мс)** определяет время действия выпущенного STS-сервисом маркера защиты. Установка флажка **Шифрование выпущенного ключа** указывает, что ключ цифровой подписи SAML-маркера, выпущенного STS-сервисом, должен быть зашифрован сертификатом Web-сервиса. Установка флажка **Шифрование выпущенного маркера** указы-

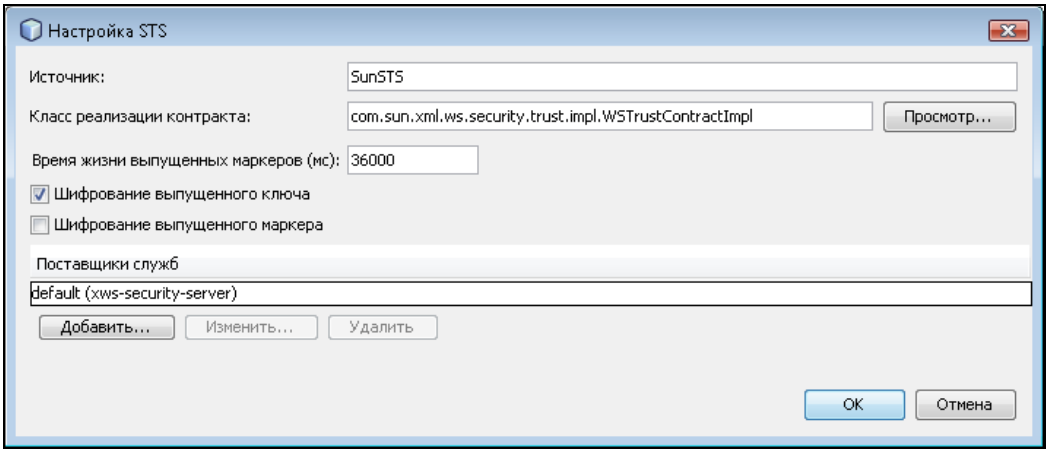


Рис. 4.12. Диалоговое окно настройки STS-сервиса

вает, что весь SAML-маркер, представленный элементом `<saml:Assertion>` сообщения, должен быть зашифрован сертификатом Web-сервиса.

Кнопка **Добавить** окна **Настройка STS** позволяет переопределить по умолчанию Web-сервис, с которым установлены доверительные отношения у STS-сервиса (рис. 4.13).

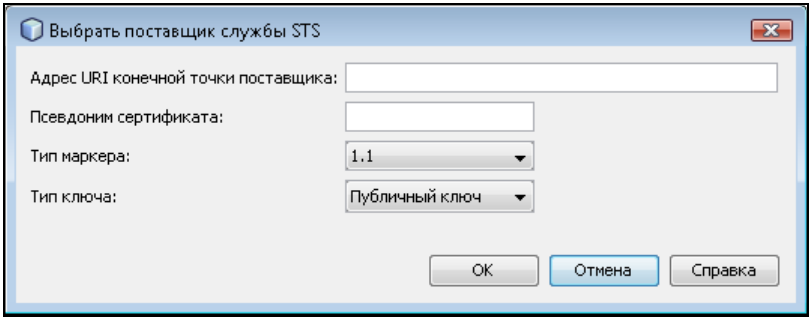


Рис. 4.13. Окно настройки потребителя выпущенного STS-сервисом маркера защиты

Все опции на вкладке **Качество обслуживания** окна STS-сервиса реализуются в его WSDL-описании, представленном узлом **Файлы настройки | xml-resources | web-services | STSExample | wsdl | STSExampleService.wsdl** проекта STSProject (см. папку Примеры\Глава4\Security\STS\STSProject\src\conf\xml-resources\web-services\STSExample\wsdl компакт-диска).

Дескриптор развертывания web.xml STS-сервиса по умолчанию имеет следующий вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
```

```
  <servlet>
    <servlet-name>stsexample.STSExample</servlet-name>
    <servlet-class>stsexample.STSExample</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>com.sun.xml.ws.mex.server.MEXEndpoint</servlet-name>
    <servlet-class>com.sun.xml.ws.mex.server.MEXEndpoint</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>stsexample.STSExample</servlet-name>
    <url-pattern>/STSExampleService</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>com.sun.xml.ws.mex.server.MEXEndpoint</servlet-name>
    <url-pattern>/STSExampleService/mex</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

Из дескриптора развертывания web.xml STS-сервиса видно, что Web-приложение STSProject предоставляет два сервиса. Один сервис, stsexample.STSExample, отвечает за передачу SAML-маркера защиты клиенту, а другой сервис, com.sun.xml.ws.mex.server.MEXEndpoint, отвечает за передачу клиенту метаданных STS-сервиса, представленных его WSDL-описанием, для того, чтобы клиент смог правильно оформить свой RST-запрос SAML-маркера защиты.

Таким образом, клиент, прежде чем отправлять RST-запрос сервису STSExample, отправляет Get-запрос, согласно спецификации WS-Transfer, сервису MEXEndpoint для получения WSDL-описания сервиса STSExample.

После развертывания проекта STSProject в сервере GlassFish v3, в окне **Проекты** среды NetBeans щелкнем правой кнопкой мыши на узле **Metro_WebService | Веб-службы | MetroTest** и выберем пункт **Правка атрибутов веб-службы**.

В появившемся диалоговом окне на вкладке **Качество обслуживания** в опции **Механизм обеспечения безопасности** выберем значение **Выпущенный STS маркер** и с помощью кнопки **Настройка** укажем:

- в строке **Адрес источника** — адрес конечной точки сервиса STSExample **http://localhost:8080/STSProject/STSExampleService**;
- в строке **Метаданные адреса источника** — адрес конечной точки сервиса **MEXEndpoint http://localhost:8080/STSProject/STSExampleService/mex**;
- в опции **Тип маркера** — **1.1** (версия SAML-маркера);

- в опции **Тип ключа — Симметричный ключ** (тип ключа цифровой подписи SAML-маркера);
- в опции **Размер ключа — 128**;
- в опции **Набор алгоритмов — Basic 128bit**;
- в опции **Макет заголовка безопасности — Свободно**.

На вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и закроем окно кнопкой **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\STS\Metro_WebService\web\WEB-INF компакт-диска.

В данном случае политика Web-сервиса MetroTest определяет, что входящее сообщение Web-сервиса должно содержать SAML-маркер `<saml:Assertion>`, выпущенный сервисом SunSTS, имеющим адрес конечной точки **http://localhost:8080/STSProject/STSExampleService** и информация о котором находится по адресу **http://localhost:8080/STSProject/STSExampleService/mex**. При этом SAML-маркер включает в себя симметричный ключ цифровой подписи SAML-маркера, зашифрованный сертификатом Web-сервиса (элемент `<sc:KeyStore>`) и используемый для шифрования тела входящего и исходящего сообщения Web-сервиса и цифровой подписи заголовков и тела входящего и исходящего сообщений Web-сервиса.

После развертывания проекта Metro_WebService в проекте Metro_WebClient создадим клиента Web-сервиса STSExampleService и клиента Web-сервиса MetroTestService.

1. Щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | MetroTestService** и выберем пункт **Правка атрибутов веб-службы**.
2. В диалоговом окне на вкладке **Качество обслуживания** в узле **Безопасность** укажем кнопкой **Доверительное хранилище** размещение сертификата сервера на стороне клиента, в узле **Служба маркеров безопасности** зададим:
   - в строке **Конечная точка** — адрес **http://localhost:8080/STSProject/STSExampleService**;
   - в строке **Местоположение файла WSDL** — адрес **http://localhost:8080/STSProject/STSExampleService?wsdl**;
   - в строке **Метаданные** — адрес **http://localhost:8080/STSProject/STSExampleService/mex**

   и нажмем кнопку **OK**.

В результате средой NetBeans будет сгенерирован WSIT-файл клиента Web-сервиса MetroTestService, содержащий политику:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:TrustStore wspp:visibility="private" storepass="changeit"
```

```
        type="JKS" location="D:\Program Files\glassfish-3.0.1\glassfish\
         domains\domain1\config\cacerts.jks" peeralias="xws-security-server"/>
      <tc:PreconfiguredSTS wspp:visibility="private"
         endpoint="http://localhost:8080/STSProject/STSExampleService"
    wsdlLocation="http://localhost:8080/STSProject/STSExampleService?wsdl"
    metadata="http://localhost:8080/STSProject/STSExampleService/mex"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

1. Щелкнем правой кнопкой мыши на узле **Metro_WebClient | Ссылки на веб-службы | STSExampleService** и выберем пункт **Правка атрибутов веб-службы**.
2. В появившемся диалоговом окне на вкладке **Качество обслуживания** в узле **Безопасность** укажем кнопкой **Доверительное хранилище** размещение сертификата сервера на стороне клиента, в строке **Имя пользователя по умолчанию** введем логин зарегистрированного пользователя сервера GlassFish, а в строке **Пароль по умолчанию** — его пароль.

В результате политика WSIT-файла клиента Web-сервиса STSExampleService примет следующий вид:

```
<wsp:Policy wsu:Id="ISTSExampleService_BindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sc:CallbackHandlerConfiguration wspp:visibility="private">
       <sc:CallbackHandler default="User" name="usernameHandler"/>
       <sc:CallbackHandler default="1234" name="passwordHandler"/>
      </sc:CallbackHandlerConfiguration>
      <sc:TrustStore wspp:visibility="private"
         peeralias="xws-security-server" storepass="changeit"
         type="JKS" location="D:\Program Files\glassfish-
3.0.1\glassfish\domains\domain1\config\cacerts.jks"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

После запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом, STS-сервисом и Web-сервисом произойдет обмен сообщениями:

1. Клиентский запрос метаданных у сервиса MEXEndpoint **http://schemas.xmlsoap.org/ws/2004/09/transfer/Get**.
2. Ответ **http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse** сервиса MEXEndpoint клиенту, содержащий в теле элемент `<mex:Metadata>` с WSDL-описанием сервиса STSExample.
3. Клиентский запрос **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue** STS-сервису STSExample.

4. Ответ **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal** STS-сервиса STSExample клиенту.
5. Клиентский запрос **http://metrotest/MetroTest/getHelloRequest** Web-сервису MetroTest.
6. Ответ **http://metrotest/MetroTest/getHelloResponse** Web-сервиса MetroTest клиенту.

Кнопка **Настройка** опции **Выпущенный STS маркер** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить:

- опцией **Тип маркера** — версию 1.0, 1.1 и 2.0 стандарта SAML (элемент `<t:TokenType>`);
- опцией **Тип ключа** — использование в качестве ключа подписи SAML-маркера симметричного или публичного ключа (элемент `<t:KeyType>`);
- опцией **Размер ключа** — размер ключа подписи SAML-маркера (элемент `<t:KeySize>`);
- опцией **Набор алгоритмов** — набор алгоритмов шифрования (дочерний элемент элемента `<sp:AlgorithmSuite>`);
- опцией **Макет заголовка безопасности** — порядок расположения дочерних элементов заголовка `<wsse:Security>` (дочерний элемент элемента `<sp:Layout>`);
- опцией **Требуются производные ключи для выпущенного маркера** — использование не самого ключа подписи SAML-маркера для шифрования и подписи сообщений, а его производных ключей (элемент `<sp:RequireDerivedKeys/>`);
- опциями **Установить безопасное соединение (защищенный диалог)** и **Требуются производные ключи для безопасного сеанса** — использование для шифрования и подписи сообщений производных ключей SCT-маркера защиты `<wsc:SecurityContextToken>` (элементы `<sp:SecureConversationToken>`, `<sp:RequireDerivedKeys/>`, `</sp:BootstrapPolicy>`, `<sp:Wss11>` и `<sp:Trust13>` политики WSDL-элемента `<binding>`);
- опцией **Требуется подтверждение подписи** — включение в исходящее сообщение Web-сервиса дочернего элемента `<wsse11:SignatureConfirmation>` заголовка `<wsse:Security>` с первоначальной подписью входящего сообщения (элемент `<sp:RequireSignatureConfirmation/>`);
- опцией **Подпись шифрования** — установить шифрование цифровой подписи ключом цифровой подписи SAML-утверждения во входящем и исходящем сообщениях Web-сервиса (элемент `<sp:EncryptSignature/>`);
- опцией **Зашифровать перед подписанием** — указать создание цифровой подписи после шифрования сообщения (элемент `<sp:EncryptBeforeSigning/>`).

Сбросив флажок **Использовать значения по умолчанию, указанные при разработке** на вкладке **Качество обслуживания** окна Web-сервиса, нажав кнопку **Хранилище ключей**, можно переопределить расположение по умолчанию хранилища ключей и сертификатов сервера. Кнопкой **Доверительное хранилище** — переоп-

ределить расположение по умолчанию доверительного хранилища ключей и сертификатов сервера. Кнопкой **Дополнительно** — указать максимальное допустимое различие между системными часами сервера и клиента и максимальное значение элемента `<wsu:Timestamp>`, установить использование механизма по умолчанию проверки устаревших сертификатов. С помощью кнопки **Проверки** можно указать пользовательский обработчик имени/пароля во входящих сообщениях на стороне сервера, реализующий интерфейс `com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidator`, пользовательский обработчик метки времени, реализующий интерфейс `com.sun.xml.wss.impl.callback.TimestampValidationCallback.TimestampValidator`, пользовательский обработчик сертификатов, реализующий интерфейс `com.sun.xml.wss.impl.callback.CertificateValidationCallback.CertificateValidator`, и пользовательский обработчик SAML, реализующий интерфейс `com.sun.xml.wss.impl.callback.SAMLAssertionValidator`.

Опция *Выпущенный STS маркер с сертификатом службы*

Отличие использования данной опции от опции **Выпущенный STS маркер** заключается в том, что при применении опции **Выпущенный STS маркер** тело входящего и исходящего сообщений Web-сервиса шифруется, а их заголовки и тело подписываются ключом цифровой подписи SAML-маркера, выпущенного для клиента STS-сервисом.

При применении же опции **Выпущенный STS маркер с сертификатом службы** для шифрования и подписи сообщений используются разные ключи. Тело и заголовки входящего сообщения Web-сервиса подписываются ключом цифровой подписи SAML-маркера, а его тело шифруется с использованием публичного ключа сертификата сервера. Тело и заголовки исходящего сообщения Web-сервиса, наоборот, подписываются с использованием приватного ключа сертификата сервера, а его тело шифруется ключом цифровой подписи SAML-маркера.

Для тестирования данной опции:

1. На вкладке **Качество обслуживания** окна Web-сервиса MetroTest в опции **Механизм обеспечения безопасности** выберем вариант **Выпущенный STS маркер с сертификатом службы**. Нажав кнопку **Настройка**, для параметра **Тип маркера** укажем значение **1.1**, для параметра **Тип ключа** — значение **Симметричный ключ**, для параметра **Размер ключа** — значение **128**, для параметра **Набор алгоритмов** — значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Свободно** и нажмем кнопку **OK**.
2. На вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и закроем окно кнопкой **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\STSCertificate\Metro_WebService\web\WEB-INF компакт-диска.

После развертывания проекта Metro_WebService и обновления клиента Web-сервиса MetroTestService в проекте Metro_WebClient в результате запуска проекта

Metro_WebClient и вызова Web-сервиса MetroTest между клиентом, STS-сервисом и Web-сервисом произойдет обмен сообщениями:

1. Клиентский запрос метаданных у сервиса MEXEndpoint **http://schemas.xmlsoap.org/ws/2004/09/transfer/Get**.
2. Ответ **http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse** сервиса MEXEndpoint клиенту, содержащий в теле элемент `<mex:Metadata>` с WSDL-описанием сервиса STSExample.
3. Клиентский запрос http**://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue** STS-сервису STSExample.
4. Ответ **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal** STS-сервиса STSExample клиенту.
5. Клиентский запрос **http://metrotest/MetroTest/getHelloRequest** Web-сервису MetroTest. Тело и заголовки запроса подписываются ключом цифровой подписи SAML-маркера, а его тело шифруется с использованием публичного ключа сертификата сервера.
6. Ответ **http://metrotest/MetroTest/getHelloResponse** Web-сервиса MetroTest клиенту. Тело и заголовки ответа подписываются с использованием приватного ключа сертификата сервера, а его тело шифруется ключом цифровой подписи SAML-маркера.

Кнопка **Настройка** опции **Выпущенный STS маркер с сертификатом службы** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить те же опции, что и кнопка **Настройка** опции **Выпущенный STS маркер**, за исключением опции **Требуются производные ключи для выпущенного маркера**.

Опция *Выпущенный STS маркер одобрения*

Отличие данной опции от опции **Выпущенный STS маркер с сертификатом службы** показано в табл. 4.2.

Таблица 4.2. *Различия применения опций* ***Выпущенный STS маркер одобрения*** *и* ***Выпущенный STS маркер с сертификатом службы***

Опция	Подпись запроса	Шифрование запроса	Подпись ответа	Шифрование ответа	Вторичная подпись запроса
Выпущенный STS маркер с сертификатом службы	Ключ SAML-маркера	Ключ, зашифрованный сертификатом сервера	Ключ, зашифрованный сертификатом сервера	Ключ SAML-маркера	Нет
Выпущенный STS маркер одобрения	Ключ, зашифрованный сертификатом сервера	Ключ, зашифрованный сертификатом сервера	Ключ, зашифрованный сертификатом сервера	Ключ, зашифрованный сертификатом сервера	Ключ SAML-маркера

Таким образом, при применении опции **Выпущенный STS маркер одобрения** для шифрования и подписи тела сообщений и подписи заголовков сообщений исполь-

зуется сертификат сервера. При этом первоначальная подпись тела и заголовков входящего сообщения Web-сервиса снабжается вторичной подписью с использованием одобренного сервером SAML-маркера клиента, тем самым обеспечивается идентификация и авторизация клиента.

Для тестирования данной опции:

1. На вкладке **Качество обслуживания** окна Web-сервиса MetroTest в опции **Механизм обеспечения безопасности** выберем вариант **Выпущенный STS маркер одобрения**. Нажав кнопку **Настройка**, для параметра **Тип маркера** укажем значение **1.1**, для параметра **Тип ключа** — значение **Симметричный ключ**, для параметра **Размер ключа** — значение **128**, для параметра **Набор алгоритмов** — значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Свободно** и нажмем кнопку **OK**.
2. На вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и закроем окно кнопкой **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\STSEndorsing\Metro_WebService\web\WEB-INF компакт-диска.

После развертывания проекта Metro_WebService и обновления клиента Web-сервиса MetroTestService в проекте Metro_WebClient в результате запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом, STS-сервисом и Web-сервисом произойдет обмен сообщениями:

1. Клиентский запрос метаданных у сервиса MEXEndpoint **http://schemas.xmlsoap.org/ws/2004/09/transfer/Get**.
2. Ответ **http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse** сервиса MEXEndpoint клиенту, содержащий в теле элемент `<mex:Metadata>` с WSDL-описанием сервиса STSExample.
3. Клиентский запрос **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue** STS-сервису STSExample.
4. Ответ **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal** STS-сервиса STSExample клиенту.
5. Клиентский запрос **http://metrotest/MetroTest/getHelloRequest** Web-сервису MetroTest. Запрос подписывается и шифруется временным ключом, зашифрованным сертификатом сервера. Первичная подпись подписывается ключом SAML-маркера.
6. Ответ **http://metrotest/MetroTest/getHelloResponse** Web-сервиса MetroTest клиенту. Ответ подписывается и шифруется временным ключом, зашифрованным сертификатом сервера.

Кнопка **Настройка** опции **Выпущенный STS маркер одобрения** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить те же опции, что и кнопка **Настройка** опции **Выпущенный STS маркер**, а также предлагает дополнительную опцию **Требуются производные ключи для маркера X509**, ко-

торая определяет использование не временного ключа, зашифрованного сертификатом сервера, а его производных ключей.

Опция *Выпущенный STS маркер поддержки*

При применении данной опции входящее и исходящее сообщения Web-сервиса шифруются и подписываются временным ключом, зашифрованным сертификатом сервера. При этом авторизация клиента осуществляется с помощью SAML-маркера, выпущенного для клиента доверенным STS-сервисом. SAML-маркер, содержащийся во входящем сообщении Web-сервиса, зашифрован и подписан временным ключом, зашифрованным сертификатом сервера.

Для тестирования данной опции:

1. На вкладке **Качество обслуживания** окна Web-сервиса MetroTest в опции **Механизм обеспечения безопасности** выберем вариант **Выпущенный STS маркер поддержки**. Нажав кнопку **Настройка**, для параметра **Тип маркера** укажем значение **1.1**, для параметра **Тип ключа** — значение **Симметричный ключ**, для параметра **Размер ключа** — значение **128**, для параметра **Набор алгоритмов** — значение **Basic 128bit**, для параметра **Макет заголовка безопасности** — значение **Свободно** и нажмем кнопку **OK**.
2. На вкладке **Качество обслуживания** установим флажок **Использовать значения по умолчанию, указанные при разработке** и закроем окно кнопкой **OK**.

В результате изменится политика WSDL-элемента `<binding>` WSIT-файла Web-сервиса, который можно посмотреть в папке Примеры\Глава4\Security\STSSupporting\Metro_WebService\web\WEB-INF компакт-диска.

После развертывания проекта Metro_WebService и обновления клиента Web-сервиса MetroTestService в проекте Metro_WebClient в результате запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом, STS-сервисом и Web-сервисом произойдет обмен сообщениями:

1. Клиентский запрос метаданных у сервиса MEXEndpoint **http://schemas.xmlsoap.org/ws/2004/09/transfer/Get**.
2. Ответ **http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse** сервиса MEXEndpoint клиенту, содержащий в теле элемент `<mex:Metadata>` с WSDL-описанием сервиса STSExample.
3. Клиентский запрос **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue** STS-сервису STSExample.
4. Ответ **http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal** STS-сервиса STSExample клиенту.
5. Клиентский запрос **http://metrotest/MetroTest/getHelloRequest** Web-сервису MetroTest. Запрос подписывается и шифруется временным ключом, зашифрованным сертификатом сервера. Запрос содержит зашифрованный и подписанный временным ключом SAML-маркер, необходимый для авторизации клиента.

6. Ответ **http://metrotest/MetroTest/getHelloResponse** Web-сервиса MetroTest клиенту. Ответ подписывается и шифруется временным ключом, зашифрованным сертификатом сервера.

Кнопка **Настройка** опции **Выпущенный STS маркер поддержки** на вкладке **Качество обслуживания** окна Web-сервиса позволяет переопределить те же опции, что и кнопка **Настройка** опции **Выпущенный STS маркер одобрения**.

Поддержка протокола SOAP/TCP

Стандартный протокол передачи SOAP-сообщений — это протокол HTTP, работающий поверх сетевого протокола TCP. При передаче SOAP-сообщений по протоколу HTTP в исходящее сообщение включаются различные HTTP-заголовки, обработка которых требует дополнительных затрат. Передача SOAP-сообщений напрямую по транспортному протоколу TCP ускоряет обмен сообщениями между клиентом и Web-сервисом.

Для оптимизации транспортного протокола обмена сообщениями вкладка **Качество обслуживания** окна Web-сервиса содержит флажок **Разрешить транспорт TCP** (рис. 4.14), а вкладка **Качество обслуживания** окна клиента Web-сервиса — флажок **Автоматически выбрать оптимальный транспорт (HTTP/TCP)** (рис. 4.15).

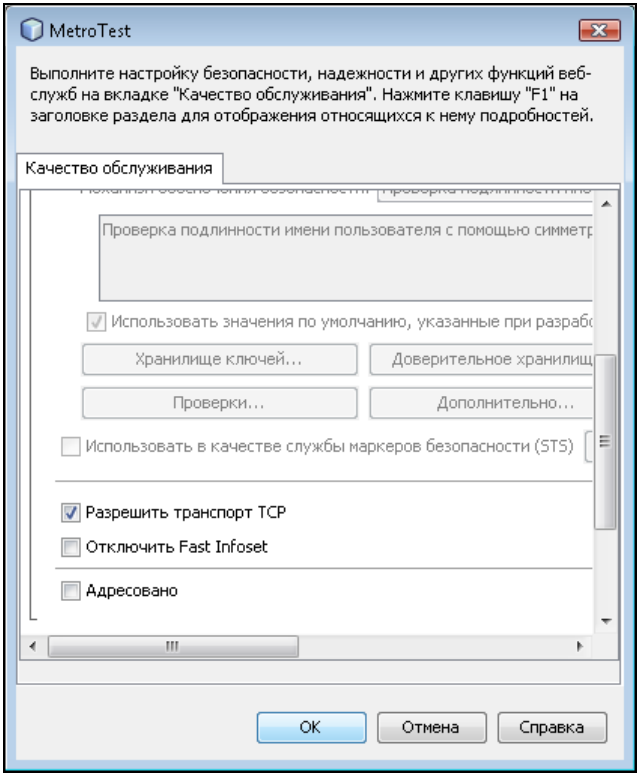


Рис. 4.14. Оптимизация транспортного протокола Web-сервиса

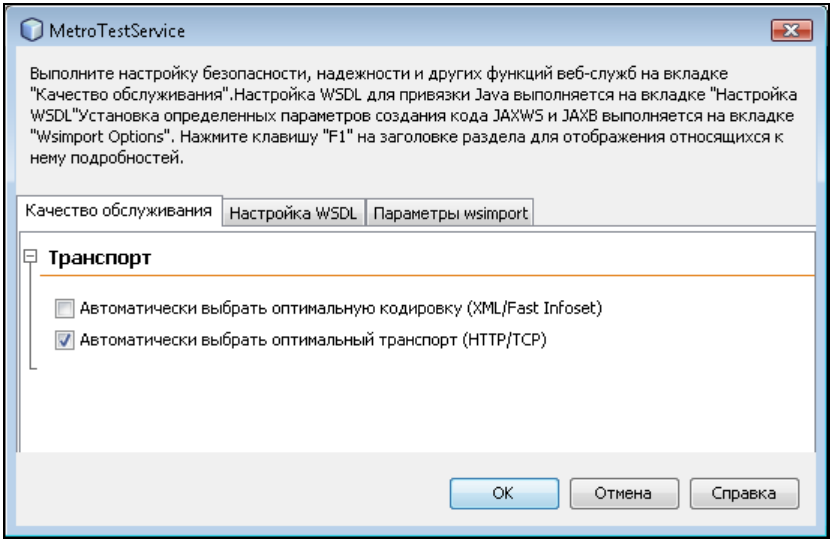


Рис. 4.15. Оптимизация транспортного протокола клиента Web-сервиса

При этом WSIT-файл Web-сервиса дополняется политикой:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tcp:OptimizedTCPTransport enabled="true"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

а WSIT-файл клиента Web-сервиса — политикой:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <tcp:AutomaticallySelectOptimalTransport/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

В данном случае клиент Web-сервиса автоматически выбирает протокол TCP для передачи SOAP-сообщений, если Web-сервис настроен на использование транспортного протокола TCP.

Поддержка кодировки Fast Infoset

XML-формат сообщений обеспечивает кроссплатформенный формат передаваемых данных, однако передача данных в виде простого текста требует больших ресурсов при его пересылке и анализе. Технология Fast Infoset позволяет сохранять XML-

данные в более эффективном бинарном формате, который существенно сокращает размер данных и значительно уменьшает время разбора и сериализации данных.

При использовании кодировки Fast Infoset XML-документ преобразуется в бинарные данные и наоборот в соответствии с его информационной моделью XML Information Set (рис. 4.16).

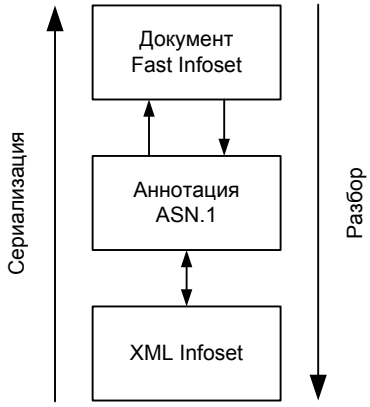


Рис. 4.16. Преобразование информационного набора документа XML Information Set в бинарные данные Fast Infoset и наоборот

Процесс преобразования XML-данных в бинарные данные состоит из создания значения ASN.1 из структуры XML Infoset документа и последующего кодирования значения ASN.1 в документ Fast Infoset.

Примечание

Abstract Syntax Notation One (ASN.1) — машинно-независимый язык описания структур данных.

Для оптимизации передачи, разбора и сериализации SOAP-сообщений с помощью технологии Fast Infoset вкладка **Качество обслуживания** в окне клиента Web-сервиса содержит флажок **Автоматически выбрать оптимальную кодировку (XML/Fast Infoset)**. При выборе данной опции WSIT-файл клиента Web-сервиса дополняется политикой:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <fi:AutomaticallySelectOptimalEncoding/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Если запрашиваемый Web-сервис изначально конфигурирован для использования Fast Infoset, тогда как входящее, так и исходящее сообщения Web-сервиса будут иметь кодировку Fast Infoset. При этом входящее сообщение Web-сервиса будет содержать HTTP-заголовки:

```
accept: application/fastinfoset, text/xml, multipart/related
content-type: application/fastinfoset
```

Если же запрашиваемый Web-сервис изначально имеет политику, запрещающую использование Fast Infoset, тогда при установленном флажке **Автоматически выбрать оптимальную кодировку (XML/Fast Infoset)** клиент Web-сервиса автоматически выбирает XML-формат для передачи сообщения.

Если политика запрашиваемого Web-сервиса явно не указывает поддержку Fast Infoset, тогда при установленном флажке **Автоматически выбрать оптимальную кодировку (XML/Fast Infoset)** входящее сообщение Web-сервиса содержит HTTP-заголовки:

```
accept: application/fastinfoset, text/xml, multipart/related
content-type: text/xml; charset=utf-8
```

и имеет XML-формат, а исходящее сообщение Web-сервиса имеет кодировку Fast Infoset.

Поддержка кодировки Fast Infoset Web-сервисом определяется политикой его WSIT-файла:

```
<wsp:Policy wsu:Id="MetroTestPortBindingPolicy">
  <wsp:ExactlyOne>
    <wsp:All>
      <fi:OptimizedFastInfosetSerialization enabled="true"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Для отключения кодировки Fast Infoset вкладка **Качество обслуживания** окна Web-сервиса содержит флажок **Отключить Fast Infoset**, при этом значение атрибута `enabled` элемента `<fi:OptimizedFastInfosetSerialization>` изменяется с `true` на `false`.

Поддержка WS-MakeConnection

В окне **Проекты** среды NetBeans откроем узел **Metro_WebService | Веб-страницы | WEB-INF | wsit-metrotest.MetroTest.xml** и включим в политику WSIT-файла Web-сервиса утверждение `<wsmc:MCSupported/>` (пространство имен `xmlns:wsmc="http://docs.oasis-open.org/ws-rx/wsmc/200702"`) (см. папку Примеры\Глава4\MakeConnection\Metro_WebService\web\WEB-INF компакт-диска).

После развертывания проекта Metro_WebService и обновления клиента Web-сервиса MetroTestService в проекте Metro_WebClient в результате запуска проекта Metro_WebClient и вызова Web-сервиса MetroTest между клиентом, STS-сервисом и Web-сервисом произойдет обмен сообщениями:

1. Клиентский запрос **http://metrotest/MetroTest/getHelloRequest**.
2. Клиентский запрос **http://docs.oasis-open.org/ws-rx/wsmc/200702/MakeConnection** создания нового транспортного канала для возврата ответа от Web-сервиса, содержащий заголовок

```
<Action xmlns="http://www.w3.org/2005/08/addressing">http://
docs.oasis-open.org/ws-rx/wsmc/200702/MakeConnection</Action>
```

и тело сообщения

```
<S:Body><MakeConnection xmlns="http://docs.oasis-open.org/ws-rx/wsmc/200702"
xmlns:ns2="http://www.w3.org/2005/08/addressing"><Address>http://docs.oasis-
open.org/ws-rx/wsmc/200702/anonymous?id=bb5f93e8-29ef-47d7-9448-
dbd28b215739</Address></MakeConnection></S:Body>,
```

указывающее элементом `<Address>` URI-идентификатор конечной точки клиента.

3. Ответ **http://metrotest/MetroTest/getHelloResponse** Web-сервиса MetroTest клиенту. Ответ содержит заголовок

```
<MessagePending xmlns="http://docs.oasis-open.org/ws-rx/wsmc/200702"
xmlns:ns2="http://www.w3.org/2005/08/addressing" pending="false"/>,
```

указывающий, что сообщения, ожидающие отправки клиенту, отсутствуют.

ГЛАВА 5

Проект Apache CXF

Проект Apache CXF (**http://cxf.apache.org/**) представляет собой открытую платформу для разработки и развертывания Web-сервисов. Он создан на основе двух проектов — Celtix и XFire, и его название является комбинацией их начальных букв.

Для скачивания платформа CXF доступна по адресу
http://cxf.apache.org/download.html.

Платформа CXF поддерживает следующие технологии и стандарты:

- спецификации первого уровня технологии Web-сервисов SOAP и WSDL;
- спецификации JAX-WS, JAXB и SAAJ для создания SOAP Web-сервисов;
- технологию MTOM для обмена сообщениями, содержащими бинарные данные;
- спецификации второго уровня технологии Web-сервисов WS-Basic Profile, WS-Addressing, WS-Policy, WS-ReliableMessaging, WS-Security;
- спецификацию JAX-RS для создания RESTful Web-сервисов;
- формат JSON для взаимодействия с JAX-RS Web-сервисами;
- транспортные протоколы HTTP, JMS и Local (локальный транспортный протокол, действующий в пределах одной виртуальной Java-машины и позволяющий ускорить обмен сообщениями без сериализации объектов);
- технологию CORBA;
- технологию Spring для объявления конечных точек и определения клиента Web-сервиса;
- технологии Aegis Databinding и XMLBeans связывания данных;
- хорошо задокументированный программный интерфейс CXF API для создания Web-сервисов на основе классов POJO и расширяющий функциональность стандартов технологии Web-сервисов.

Платформа CXF поддерживает разработку трех типов Web-сервисов — SOAP, RESTful и CORBA Web-сервисов. Помимо SOAP-связывания (по умолчанию), платформа CXF поддерживает чистое XML-связывание для обмена XML-сообщениями и HTTP-связывание для реализации REST-архитектуры.

Платформа CXF также включает в себя JavaScript-модуль, дающий возможность создавать сервисы с использованием библиотеки Java Rhino.

Основное отличие стека CXF Web-сервисов от стека Metro заключается в том, что стек CXF предлагает более широкую поддержку различных технологий в части разработки самих Web-сервисов и их клиентов, а стек Metro охватывает более широкую поддержку в части спецификаций второго уровня технологии Web-сервисов для обеспечения безопасности, надежности и транзакций взаимодействия с Web-сервисами.

Стек Metro реализует спецификации JAX-WS и JAXB и технологию WSIT для поддержки спецификаций второго уровня технологии Web-сервисов. При этом технология WSIT основана на использовании декларативного включения опций безопасности, надежности и транзакций как для Web-сервиса, так и для клиента Web-сервиса, с помощью WSIT-файла и гарантирует совместимость с WCF Web-сервисами платформы Microsoft .NET.

Стек CXF также реализует спецификации JAX-WS и JAXB. Однако помимо этого стек CXF предлагает свою реализацию спецификации JAX-RS и помимо JAXB — связывание данных Aegis и XMLBeans. Кроме того, платформа CXF имеет свой, хорошо задокументированный программный интерфейс CXF API, расширяющий стандарты технологии Web-сервисов. В части поддержки безопасности, надежности и транзакций платформа CXF не имеет такой общей технологии, как автоматическая генерация WSDL-описания на основе конфигурационного WSIT-файла, и так полно не поддерживает технологии второго уровня, как стек Metro.

Архитектура платформы CXF

Архитектура платформы CXF состоит из следующих частей:

- Bus;
- Frontend;
- Messaging & Interceptors;
- Service Model;
- Pluggable Data Bindings;
- Protocol Bindings;
- Transports.

Шина Bus пакета `org.apache.cxf` платформы CXF является поставщиком общих ресурсов и сервисов для среды выполнения CXF и обеспечивает общий контекст приложения, который создается объектом `SpringBusFactory` на основе конфигурационных Spring-файлов платформы CXF cxf.xml, cxf-extension.xml и cxf-property-editors.xml.

Интерфейс программирования Frontend платформы CXF представлен реализациями технологий JAX-WS, JAX-RS, Simple и JavaScript. Программный интерфейс Simple Frontend дает возможность создавать SOAP Web-сервисы и их клиентов без использования аннотаций спецификации JAX-WS.

Для низкоуровневой обработки SOAP-сообщений платформа CXF предлагает программные интерфейсы JAX-WS Handler API и Interceptor CXF API.

Объекты Interceptor являются основой обработки сообщений платформы CXF.

При вызове Web-сервиса, развернутого на платформе CXF, сначала создается и вызывается цепочка InterceptorChain объектов Interceptor, которые отвечают за разбор, преобразование, обработку заголовков и т. д. входящих сообщений Web-сервиса. Исходящие сообщения Web-сервиса также обрабатываются цепочкой объектов Interceptor. Цепочки объектов Interceptor являются частью процесса обмена сообщениями как на стороне CXF Web-сервиса, так и на стороне CXF-клиента.

Конечная точка Web-сервиса имеет три цепочки объектов Interceptor — цепочка обработки входящих сообщений, цепочка обработки исходящих сообщений и цепочка обработки ошибок.

В цепочке каждый объект Interceptor может быть связан с определенной фазой процесса обмена сообщениями. Обработка входящего сообщения Web-сервиса разделена на следующие фазы:

- RECEIVE — обработка на транспортном уровне;
- (PRE/USER/POST)_STREAM — обработка-трансформация входящего потока;
- READ — чтение заголовков входящего сообщения;
- (PRE/USER/POST)_PROTOCOL — обработка на уровне SOAP-протокола;
- UNMARSHAL — демаршализация запроса;
- (PRE/USER/POST)_LOGICAL — обработка запроса;
- PRE_INVOKE — подготовка к запросу Web-сервиса;
- INVOKE — вызов Web-сервиса;
- POST_INVOKE — вызов цепочки объектов Interceptor обработки исходящего сообщения.

Обработка исходящего сообщения Web-сервиса разделена на следующие фазы:

- SETUP — конфигурирование;
- (PRE/USER/POST)_LOGICAL — обработка объектов, подлежащих маршализации;
- PREPARE_SEND — создание соединения с адресатом;
- PRE_STREAM — подготовка создания исходящего потока;
- PRE_PROTOCOL — подготовка поддержки протокола;
- WRITE — создание основы исходящего сообщения согласно протоколу;
- MARSHAL — маршализация объектов;
- (USER/POST)_PROTOCOL — создание исходящего сообщения;
- (USER/POST)_STREAM — создание исходящего потока;
- SEND — отправка сообщения и закрытие исходящего потока.

По умолчанию платформа CXF создает следующие объекты Interceptor на стороне сервера:

- `AttachmentInInterceptor` — разбирает MIME-структуру входящего сообщения, отделяя тело сообщения от его вложений;

- `StaxInInterceptor` — создает Stax-парсер входящего потока;
- `ReadHeadersInterceptor` — разбирает и обрабатывает SOAP-заголовки;
- `SoapActionInInterceptor` — разбирает и обрабатывает SOAP-заголовки `Action`;
- `MustUnderstandInterceptor` — разбирает и обрабатывает SOAP-заголовки `MustUnderstand`;
- `SOAPHandlerInterceptor` — JAX-WS-обработчик, обеспечивает доступ к блоку SOAP-заголовков;
- `LogicalHandlerInInterceptor` — JAX-WS-обработчик, обеспечивает доступ к телу входящего сообщения;
- `CheckFaultInterceptor` — проверяет ошибки и вызывает обработчиков ошибок;
- `URIMappingInterceptor` — обрабатывает HTTP-запрос GET;
- `DocLiteralnInterceptor` — осуществляет разбор первого элемента тела входящего сообщения для определения вызываемой операции Web-сервиса;
- `SoapHeaderInterceptor` — связывает SOAP-заголовки с параметрами вызываемых операций;
- `WrapperClassInInterceptor` — в случае стиля wrapped doc/lit в результате связывания данных создает массив `Object[]`, необходимый для вызова операций;
- `SwAInInterceptor` — разбирает SwA-вложения сообщения;
- `HolderInInterceptor` — для параметров OUT и IN/OUT создает объекты `Holder`;
- `ServiceInvokerInInterceptor` — вызывает Web-сервис;
- `HolderOutInterceptor` — для параметров OUT и IN/OUT извлекает значения объектов `Holder` и добавляет их в список параметров исходящего сообщения;
- `SwAOutInterceptor` — определяет вложения для исходящего сообщения;
- `WrapperClassOutInterceptor` — в случае стиля wrapped doc/lit создает объект `JAXB`, представляющий исходящее сообщение;
- `SoapHeaderOutFilterInterceptor` — удаляет ненужные заголовки;
- `SoapActionOutInterceptor` — устанавливает SOAP-заголовок `Action`;
- `MessageSenderInterceptor` — подготавливает исходящее сообщение на транспортном уровне;
- `SoapPreProtocolOutInterceptor` — формирует общие SOAP-заголовки;
- `AttachmentOutInterceptor` — подготавливает маршализацию вложений исходящего сообщения;
- `StaxOutInterceptor` — создает объект `XMLStreamWriter`;
- `SoapHandlerInterceptor` — JAX-WS-обработчик, формирует SOAP-заголовки исходящего сообщения;
- `SoapOutInterceptor` — создает начальный элемент элементов `<soap:envelope>` и `<soap:body>` и завершает формирование SOAP-заголовков;

- `LogicalHandlerOutInterceptor` — JAX-WS-обработчик, формирует тело сообщения;
- `WrapperOutInterceptor` — создает оберточные элементы для исходящего сообщения;
- `BareOutInterceptor` — создает элементы исходящих параметров;
- `SoapOutInterceptor.SoapOutEndingInterceptor` — завершает формирование элементов `<soap:body>` и `<soap:envelope>` исходящего сообщения;
- `StaxOutInterceptor.StaxOutEndingInterceptor` — закрывает исходящий поток;
- `MessageSenderInt.MessageSenderEnding` — завершает обмен между клиентом и сервером.

Помимо объектов Interceptor по умолчанию, в цепочку обработчиков входящих и исходящих сообщений могут быть включены пользовательские объекты Interceptor, созданные с помощью пакета `org.apache.cxf.interceptor` CXF API.

Добавлять пользовательские объекты Interceptor можно различными способами:

- программным способом методом `add()` объекта `org.apache.cxf.endpoint.Server.getEndpoint().getInInterceptor()` или объекта `org.apache.cxf.endpoint.Client.getInInterceptor()`;
- с помощью аннотаций `@org.apache.cxf.interceptor.InInterceptors`, `@org.apache.cxf.interceptor.InFaultInterceptors`, `@org.apache.cxf.interceptor.OutInterceptors`, `@org.apache.cxf.interceptor.InFaultInterceptors`;
- используя конфигурационный Spring-файл `<beans>`.

Программный интерфейс Invoker API платформы CXF дает возможность контролировать вызов методов Web-сервиса.

Объект `Invoker` перехватывает сообщение перед вызовом метода Web-сервиса и таким образом позволяет фильтровать его функциональность. Создавать объекты `Invoker` позволяет пакет `org.apache.cxf.service.invoker` CXF API. Определять объект `Invoker` для конечной точки Web-сервиса дает возможность метод `setInvoker(Invoker invoker)` интерфейса `org.apache.cxf.service.Service`.

Метод `setExecutor(Executor executor)` интерфейса `org.apache.cxf.service.Service` определяет для конечной точки Web-сервиса объект `Executor`, отвечающий за регулировку потоков выполнения операций Web-сервиса.

Программный интерфейс платформы CXF внутри себя использует модель Service Model для создания Web-сервиса. Модель Service Model является WSDL-подобным представлением структуры Web-сервиса и составляется из объектов пакета org.`apache.cxf.service.model`, таких как `ServiceInfo`, `InterfaceInfo`, `OperationInfo` и др.

Связывание данных Pluggable Data Bindings отвечает за связь между XML-элементами и Java-объектами, обеспечивая маршализацию/демаршализацию XML-данных, создание XML-схем и генерацию Java-кода из WSDL-описания. Для свя-

зывания Java-объектов с XML-элементами платформа CXF поддерживает технологии JAXB (по умолчанию), XMLBeans и Aegis. Рекомендуется применять связывание JAXB вместе с программным интерфейсом JAX-WS, а связывание Aegis — совместно с интерфейсом Simple Frontend.

Связывание протокола Protocol Bindings определяет формат сообщений, участвующих в обмене с Web-сервисом. Платформа CXF поддерживает протоколы SOAP 1.1, SOAP 1.2, REST/HTTP, чистый XML и CORBA.

Компонент Transports архитектуры платформы CXF обеспечивает транспортировку сообщений между клиентом и Web-сервисом. Платформа CXF поддерживает такие транспортные протоколы, как HTTP, JMS, Camel и Local.

Создание SOAP Web-сервисов с использованием CXF API

Разработка SOAP Web-сервисов платформы CXF может осуществляться двумя способами — с помощью CXF-реализации спецификации JAX-WS и программного интерфейса Simple Frontend.

Разработка клиента SOAP Web-сервиса также может осуществляться различными способами:

- создание клиентской заглушки JAX-WS из копии WSDL-описания Web-сервиса с помощью инструмента WSDL2Java;
- создание объекта JAX-WS Proxy из удаленного WSDL-описания Web-сервиса с помощью методов `create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)` и `getPort(java.lang.Class<T> serviceEndpointInterface)` класса `javax.xml.ws.Service`;
- создание динамического JAX-WS Dispatch-клиента;
- создание клиентской заглушки с помощью программного интерфейса `ClientProxyFactoryBean` API платформы CXF;
- создание динамического клиента с помощью CXF-интерфейса `org.apache.cxf.endpoint.Client`.

В *разд. "Пример создания JAX-WS Web-сервиса и JAX-WS клиента" главы 3* уже рассматривалась разработка CXF JAX-WS Web-сервиса и его клиента из POJO-класса в среде Eclipse.

В процессе создания клиента Web-сервиса *главы 3* в проекте CXF_WebServiceClient среда Eclipse автоматически генерировала, используя адрес WSDL-описания Web-сервиса, все необходимые JAX-WS-артефакты на стороне клиента. При разработке проекта CXF_WebService среда Eclipse автоматически маркировала класс Web-сервиса JAX-WS-аннотациями и создавала WSDL-документ и XSD-схему Web-сервиса. При этом в диалоговом окне, открываемом с помощью меню **Window | Preferences | CXF 2.x Preferences** среды Eclipse, на вкладке **Endpoint Config** можно включать или выключать использование контекста ApplicationContext контейне-

ра Spring для развертывания Web-сервиса (опции **Use Spring Application Context** и **Use CXF Servlet** соответственно).

В случае использования контекста ApplicationContext контейнера Spring в папке WEB-INF проекта средой Eclipse автоматически создается конфигурационный Spring-файл beans.xml, содержащий элемент `<jaxws:endpoint>`, который определяет конфигурацию конечной точки Web-сервиса. Дескриптор развертывания web.xml, при использовании контекста ApplicationContext контейнера Spring, содержит элемент `<context-param>`, указывающий в качестве значения контекстного параметра `contextConfigLocation` сервлета org.apache.cxf.transport.servlet.CXFServlet — конфигурационный файл WEB-INF\beans.xml.

> ***Примечание***
>
> Стандартный сервлет CXFServlet платформы CXF служит адаптером для конечной точки Web-сервиса и позволяет Web-сервису быть развернутым в контейнере Servlet.

Дескриптор развертывания web.xml также включает в себя элемент `<listener>`, указывающий класс `org.springframework.web.context.ContextLoaderListener` платформы Spring, который обеспечивает загрузку контекста Web-приложения.

Если же контекст ApplicationContext контейнера Spring не используется для развертывания Web-сервиса, тогда в папке WEB-INF проекта генерируется конфигурационный Spring-файл cxf-servlet.xml, содержащий элемент `<jaxws:endpoint>`, который определяет конфигурацию конечной точки Web-сервиса.

Использование контекста ApplicationContext контейнера Spring делает развертывание Web-сервиса быстрее, т. к. контекст ApplicationContext загружается перед созданием Spring-шины среды выполнения Bus. Только после инициализации Spring-шины Bus загружается конфигурационный файл cxf-servlet.xml, публикующий конечную точку Web-сервиса. Поэтому без использования контекста ApplicationContext загружаются все возможные CXF-модули для конечной точки. Контекст ApplicationContext позволяет определить, какие конкретно CXF-модули требуются.

В данном примере Web-сервис развертывался в Servlet-контейнере сервера Apache Tomcat, т. е. платформа CXF работала поверх сервера Apache Tomcat. Однако платформа CXF позволяет развертывать JAX-WS Web-сервисы и на платформе Java SE с помощью метода `publish(String address, Object implementor)` класса `javax.xml.ws.Endpoint` или с помощью методов класса `org.apache.cxf.jaxws.JaxWsServerFactoryBean` интерфейса CXF API.

Клиент Web-сервиса в данном примере получал Proxy-объект методом `getPort()` класса `javax.xml.ws.Service`. Для JavaSE-клиента платформа CXF также предлагает альтернативный способ получения Proxy-объекта для вызова Web-сервиса с помощью метода `create()` класса `org.apache.cxf.jaxws.JaxWsProxyFactoryBean`.

Для создания динамического JAX-WS клиента платформа CXF предлагает интерфейс `org.apache.cxf.endpoint.Client`, объект которого может быть получен методом `createClient()` класса `org.apache.cxf.jaxws.endpoint.dynamic.JaxWsDynamicClientFactory`.

Далее рассмотрим альтернативный JAX-WS технологии способ создания Web-сервиса и его клиента на основе программного интерфейса Simple Frontend платформы CXF и их развертывание на платформе Java SE.

Программный интерфейс Simple Frontend платформы CXF позволяет создавать Web-сервисы и их клиентов без применения JAX-WS-аннотаций. Рекомендуется при этом использовать технологию Aegis для связывания данных при взаимодействии с Web-сервисом.

Платформа CXF построена таким образом, что ее классы реализации JAX-WS являются подклассами классов реализации Simple Frontend. Поэтому реализация JAX-WS дает больше возможностей для создания Web-сервисов и их клиентов по сравнению с реализацией Simple Frontend, которая является более общей и может быть эффективно использована в простых случаях.

Web-сервисы Simple Frontend могут быть опубликованы двумя способами — с помощью объекта `org.apache.cxf.frontend.ServerFactoryBean` или посредством конфигурационного элемента `<simple:server>`.

Для создания Proxy-клиента Simple Frontend может использоваться программный интерфейс `ClientProxyFactoryBean` API пакета `org.apache.cxf.frontend` или же интерфейс `org.apache.cxf.endpoint.Client` — для динамического вызова Web-сервиса. Объект Client может быть создан методом `createClient()` класса `org.apache.cxf.endpoint.dynamic.DynamicClientFactory`.

Для создания Web-сервиса Simple Fronten:

1. Откроем среду Eclipse, переключимся на перспективу Java, в меню **File** выберем пункты **New | Java Project**, введем имя проекта SimpleWebService, выберем в опции **Use a project specific JRE** установленный на компьютере набор JDK и нажмем кнопку **Finish**.

   На компакт-диске

   Проект SimpleWebService находится в папке Примеры\Глава5 компакт-диска.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module** и **CXF 2.x Web Services** и перейдем по ссылке **Further configuration available | Configure installed runtimes**. На вкладке **CXF Runtime**, нажав кнопки **Add** и **Browse**, укажем каталог предварительно инсталлированной среды выполнения Apache CXF (**http://cxf.apache.org/download.html**). Нажмем кнопку **Finish**. На вкладке **CXF Runtime** отметим добавленную среду Apache CXF и трижды нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebService** и выберем пункты **New | Interface**, введем имя интерфейса `CXFSimple` и имя пакета `cxfsimple` и нажмем кнопку **Finish**.
4. В редакторе исходного кода дополним код интерфейса `CXFSimple` методом `getHello()`:

   ```
   public String getHello(String name);
   ```

5. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebService | src | cxfsimple** и выберем пункты **New | Class**, введем имя класса `CXFSimpleImp`. Нажмем кнопку **Add**, расположенную рядом с полем **Interfaces**, и введем имя интерфейса `CXFSimple`. Нажмем кнопку **OK** и закроем окно **New Java Class** кнопкой **Finish**.
6. В редакторе исходного кода дополним код метода `getHello()` класса `CXFSimpleImp`:

```
public String getHello(String name) {
  return "Hello"+" "+ name;
}
```

7. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebService | src | cxfsimple** и выберем пункты **New | Class**. Введем имя класса `CXFSimpleServer`, отметим опцию **public static void main (String [] args)** и нажмем кнопку **Finish**.
8. В редакторе исходного кода дополним код метода `main()` класса `CXFSimple Server`:

```
CXFSimpleImp service=new CXFSimpleImp();
org.apache.cxf.frontend.ServerFactoryBean svrFactory =
  new org.apache.cxf.frontend.ServerFactoryBean();
svrFactory.setServiceClass(CXFSimple.class);
svrFactory.setAddress("http://localhost:8080/CXFSimple");
svrFactory.setServiceBean(service);
svrFactory.create();
```

9. Для развертывания Web-сервиса CXFSimple в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebService | src | cxfsimple | CXFSimpleServer.java** и выберем пункты **Run As | Java Application**.

В результате, набрав в строке браузера адрес **http://localhost:8080/CXFSimple?wsdl**, можно увидеть WSDL-описание Web-сервиса CXFSimple.

Для создания клиента CXF Web-сервиса:

1. В меню **File** среды Eclipse выберем пункты **New | Java Project**, введем имя проекта SimpleWebServiceClient и нажмем кнопку **Finish**.

   На компакт-диске
   Проект SimpleWebServiceClient находится в папке Примеры\Глава5 компакт-диска.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module** и **CXF 2.x Web Services** и нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebServiceClient** и выберем пункты **New | Class**, введем имя класса `CXFSimpleClient` и имя пакета `cxfsimpleclient`, отметим опцию **public static void main (String [] args)** и нажмем кнопку **Finish**.

4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Java Build Path | Libraries**, нажмем кнопку **Add Class Folder**, укажем **SimpleWebService | bin** и дважды нажмем кнопку **OK**.
5. В редакторе исходного кода дополним код метода `main()` класса `CXFSimpleClient`:

```
org.apache.cxf.frontend.ClientProxyFactoryBean factory =
  new org.apache.cxf.frontend.ClientProxyFactoryBean();
factory.setServiceClass(cxfsimple.CXFSimple.class);
factory.setAddress("http://localhost:8080/CXFSimple");
cxfsimple.CXFSimple client = (cxfsimple.CXFSimple) factory.create();
String str=client.getHello("User");
System.out.println(str);
```

6. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleWebServiceClient | Java Resources: src | cxfsimpleclient | CXFSimpleClient.java** и выберем пункты **Run As | Java Application**.

В результате в окно **Console** среды Eclipse будет выведено сообщение:

```
Hello User
```

Ранее упоминалось, что Web-сервисы Simple Frontend могут быть опубликованы двумя способами — с помощью объекта `org.apache.cxf.frontend.ServerFactoryBean` или посредством конфигурационного элемента `<simple:server>`.

Использование конфигурационного элемента `<simple:server>` (пространство имен `xmlns:simple="http://cxf.apache.org/simple"`) аналогично применению элемента `<jaxws:endpoint>` в конфигурационном Spring-файле, т. е. элемент `<simple:server>` описывает конечную точку Simple Frontend Web-сервиса для его развертывания во встроенном Jetty-сервере или в Servlet-контейнере какого-либо Web-сервера, например Apache Tomcat, и работы платформы CXF поверх Web-сервера. При этом также возможны комбинации дескриптора развертывания web.xml с конфигурационным Spring-файлом beans.xml или cxf-servlet.xml.

Элемент `<simple:server>` может содержать следующие атрибуты и дочерние элементы:

- атрибут `endpointName` — имя конечной точки в формате "`ns:`*`ENDPOINT_NAME`", соответствует элементу `<wsdl:port>`;
- атрибут `serviceName` — имя сервиса в формате "`ns:`*`SERVICE_NAME`", соответствует элементу `<wsdl:service>`;
- атрибут `wsdlLocation` — адрес WSDL-описания;
- атрибут `bindingId` — URI-идентификатор типа связывания;
- атрибут `transported` — URI-идентификатор транспортного протокола;
- атрибут `address` — адрес конечной точки;
- атрибут `bus` — идентификатор элемента, конфигурирующего шину Bus;

- атрибут `serviceBean` — SEI-класс;
- атрибут `serviceClass` — SEI-интерфейс;
- атрибут `start` — если `true` (по умолчанию), тогда конечная точка публикуется сразу;
- элемент `<simple:executor>` — класс `Executor`;
- элемент `<simple:inInterceptors>` — список обработчиков входящих сообщений Interceptor;
- элемент `<simple:inFaultInterceptors>` — список обработчиков входящих ошибок Interceptor;
- элемент `<simple:outInterceptors>` — список обработчиков исходящих сообщений Interceptor;
- элемент `<simple:outFaultInterceptors>` — список обработчиков исходящих ошибок Interceptor;
- элемент `<simple:properties>` — свойства конечной точки;
- элемент `<simple:dataBinding>` — класс `DataBinding`;
- элемент `<simple:binding>` — класс `BindingFactory`;
- элемент `<simple:features>` — опции конечной точки;
- элемент `<simple:invoker>` — класс `Invoker`;
- элемент `<simple:schemaLocations>` — адрес схемы данных;
- элемент `<simple:serviceFactory>` — класс `ServiceFactory`;
- элемент `<simple:serviceBean>` — SEI-класс.

Клиент Simple Frontend также может быть конфигурирован с помощью Spring-файла, который будет уже содержать элемент `<simple:client>`. При этом для получения объекта клиентской заглушки из контекста приложения необходимо применить Spring-интерфейс `getBean` API.

Элемент `<simple:client>` может содержать следующие атрибуты и дочерние элементы:

- атрибут `endpointName` — аналогично `<simple:server>`;
- атрибут `serviceName` — аналогично `<simple:server>`;
- атрибут `wsdlLocation` — аналогично `<simple:server>`;
- атрибут `bindingId` — аналогично `<simple:server>`;
- атрибут `address` — аналогично `<simple:server>`;
- атрибут `bus` — аналогично `<simple:server>`;
- атрибут `serviceClass` — аналогично `<simple:server>`;
- атрибут `username` — логин;
- атрибут `password` — пароль;

- элемент `<simple:inInterceptors>` — аналогично `<simple:server>`;
- элемент `<simple:inFaultInterceptors>` — аналогично `<simple:server>`;
- элемент `<simple:outInterceptors>` — аналогично `<simple:server>`;
- элемент `<simple:outFaultInterceptors>` — аналогично `<simple:server>`;
- элемент `<simple:properties>` — аналогично `<simple:server>`;
- элемент `<simple:dataBinding>` — аналогично `<simple:server>`;
- элемент `<simple:binding>` — аналогично `<simple:server>`;
- элемент `<simple:features>` — аналогично `<simple:server>`;
- элемент `<simple:conduitSelector>` — класс `ConduitSelector`.

Рассмотрим создание динамического клиента Simple Frontend с использованием интерфейса `org.apache.cxf.endpoint.Client`.

Для создания динамического клиента Simple Frontend:

1. В меню **File** среды Eclipse выберем пункты **New | Java Project** и введем имя проекта SimpleCXFClient и нажмем кнопку **Finish**.

   На компакт-диске
   Проект SimpleCXFClient находится в папке Примеры\Глава5 компакт-диска.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleCXFClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module** и **CXF 2.x Web Services** и нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleCXFClient** и выберем пункты **New | Class**, введем имя класса `SimpleClient` и имя пакета `simpleclient`, отметим опцию **public static void main (String [] args)** и нажмем кнопку **Finish**.
4. В редакторе исходного кода дополним код метода `main()` класса `SimpleClient`:

```
org.apache.cxf.endpoint.dynamic.DynamicClientFactory clientFactory =
  org.apache.cxf.endpoint.dynamic.DynamicClientFactory.newInstance();
org.apache.cxf.endpoint.Client client =
  clientFactory.createClient("http://localhost:8080/CXFSimple?wsdl");
try {
  Object[] response=client.invoke("getHello","User");
  System.out.println(response[0]);
} catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
}
```

5. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **SimpleCXFClient | Java Resources: src | simpleclient | SimpleClient.java** и выберем пункты **Run As | Java Application**.

В результате в окно **Console** среды Eclipse будет выведено сообщение:

```
Hello User
```

Связывание данных Aegis

Система Aegis представляет собой систему связывания Java-объектов с XML-документами, определяемыми своими XML-схемами, и наоборот, основанную на технологии StAX.

Система связывания данных Aegis является частью проекта XFire.

Основным преимуществом и целью применения системы Aegis является возможность уточнения связывания без добавления аннотаций в Java-код, т. е. контролировать сериализацию Java-объектов без модификации исходного кода. Осуществляется такой контроль в системе Aegis с применением конфигурационного файла [*имя Java-класса*].aegis.xml.

Работа системы Aegis осуществляется на трех уровнях. Сначала система Aegis связывает Java-объекты, исходя из WSDL-описания и XML-схем, затем идет обработка Java-аннотаций и после этого уточнение связывания с помощью конфигурационного Aegis-файла.

Система Aegis идеально работает вместе с программным интерфейсом Simple Frontend, т. к. при этом не возникают конфликты между JAX-WS-аннотациями и связыванием Aegis.

Связывание данных Aegis устанавливается для Web-сервиса с помощью конфигурационного элемента `<simple:server>`, например, путем включения в него следующего блока:

```
<simple:dataBinding>
  <bean class="org.apache.cxf.aegis.databinding.AegisDatabinding" />
</simple:dataBinding>
```

или методом

```
org.apache.cxf.frontend.ServerFactoryBean.getServiceFactory().setDataBinding
(new org.apache.cxf.aegis.databinding.AegisDatabinding ()).
```

На стороне клиента связывание Aegis определяется конфигурационным элементом `<simple:client>`, или методом

```
org.apache.cxf.frontend.ClientProxyFactoryBean.getServiceFactory().
setDataBinding(new org.apache.cxf.aegis.databinding.AegisDatabinding()).
```

Система Aegis создает для Web-сервиса набор связей между Java-типами и XML-типами схемы, причем система Aegis изначально имеет определенный набор Java-классов, обеспечивающих такое связывание по умолчанию. При сериализации Java-объектов в XML-элементы система Aegis ищет соответствие Java-объекта связыванию. Если система Aegis не находит соответствие Java-объекта связыванию по умолчанию, она вызывает специальные объекты Type Creators, отвечающие за создание связывания типов Java-XML путем создания XML-схем из Java-объектов.

Новое связывание сохраняется в виде специального Java-класса, обеспечивающего сериализацию такого Java-объекта.

При десериализации XML-элементов в Java-объекты платформа CXF вызывает систему Aegis с использованием QName-имен, исходя из XML-схем WSDL-описания Web-сервиса. Далее система Aegis связывает существующие Java-типы с XML-типами и создает Java-объекты. При этом система Aegis не может динамически создавать Java-классы исходя из XML-схемы, поэтому все Java-классы должны уже существовать на момент десериализации.

Если Web-сервис содержит Java-типы, не отображаемые прямо SEI-интерфейсом, например, в свойствах SEI-класса, тогда требуется конфигурировать объект `AegisContext`, посылаемый объекту `AegisDatabinding`, с помощью определения его свойств `rootClasses`, `writeXsiTypes` и `beanImplementationMap`.

Для контроля над связыванием по умолчанию существует класс `org.apache.cxf.aegis.type.TypeCreationOptions`, объект которого посылается объекту `AegisContext`. Класс `TypeCreationOptions` имеет свойства `defaultNillable`, `defaultMinOccurs`, `defaultExtensibleElements` и `defaultExtensibleAttributes`, позволяющие переопределить опции связывания по умолчанию.

Система Aegis также предусматривает наличие конфигурационного Aegis-файла, уточняющего детали связывания Java-типов с XML-типами. Aegis-файл размещается в том же месте, что и Java-класс, связывание которого он уточняет.

Корневым элементом Aegis-файла является элемент `<mappings>`, определяющий связывания с помощью дочерних элементов `<mapping>` (рис. 5.1).

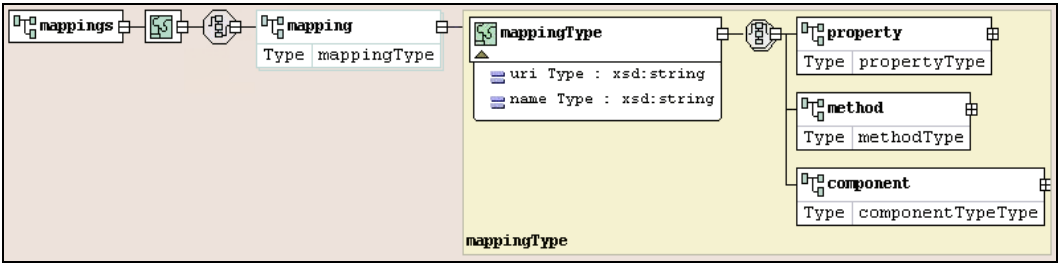


Рис. 5.1. Общая схема Aegis-файла

Элемент `<mapping>` может иметь следующие атрибуты и дочерние элементы:

- атрибут `uri` уточняет пространство имен сервиса, для которого определяется связывание;
- атрибут `name` указывает имя Java-типа, для которого определяется связывание;
- элемент `<property>` определяет связывание Java-свойства;
- элемент `<method>` определяет связывание Java-метода;
- элемент `<component>` определяет связывание коллекции.

Элемент `<property>` может иметь следующие атрибуты:

- `name` — имя Java-свойства;
- `type` — имя Java-класса, представляющего свойство;
- `typeName` — тип данных свойства (`xsd:string`);
- `mappedName` — имя XML-элемента;
- `nillable` — свойство `nillable` XML-элемента (`xsd:Boolean`);
- `flag` — контролирует использование `xsi:type` при обработке Java-объекта (`xsd:Boolean`);
- `ignore` — предотвращает сериализацию (`xsd:Boolean`);
- `componentType` — тип данных коллекции;
- `keyType` — для Map-объекта тип ключа;
- `valueType` — для Map-объекта тип значения;
- `minOccurs` — минимальное количество XML-элементов данного типа;
- `maxOccurs` — максимальное количество XML-элементов данного типа;
- `style` — XML-стиль — сериализация в атрибут или элемент.

Элемент `<method>` указывает уточняемый Java-метод с помощью атрибута `name` и определяет его связывание дочерними элементами `<return-type>` и `<parameter>`.

Элемент `<return-type>`, определяющий связывание возвращаемого Java-методом значения, может иметь те же атрибуты, что и элемент `<property>`, а элемент `<parameter>`, определяющий связывание параметров Java-метода, помимо тех же атрибутов, что и атрибуты элемента `<property>`, имеет атрибуты `index` и `class`, указывающие индекс параметра и Java-класс, его представляющий.

Элемент `<component>` имеет помимо тех же атрибутов, что и атрибуты элемента `<property>`, атрибут `class`, указывающий Java-класс, представляющий тип коллекции.

Так же как и связывание JAXB, связывание Aegis поддерживает такие Java-аннотации, как `XmlAttribute`, `XmlType`, `XmlElement`, `XmlReturnType`, `XmlIgnore`, `javax.jws.WebParam`, `javax.jws.WebResult`, `javax.xml.bind.annotation.XmlAttribute`, `javax.xml.bind.annotation.XmlElement`, `javax.xml.bind.annotation.XmlSchema`, `javax.xml.bind.annotation.XmlType`, `javax.xml.bind.annotation.XmlTransient`.

Поддержка MTOM отличается в системах Aegis и JAXB. Система JAXB всегда поддерживает XOP-пакетирование, а система Aegis — только если поддержка MTOM определена методом `AegisDatabinding.setMtomEnabled()`.

На компакт-диске

Пример использования системы Aegis совместно с программным интерфейсом Simple Frontend можно посмотреть в папке Примеры\Глава5\Aegis компакт-диска.

В данном примере на стороне клиента добавляются обработчики `org.apache.cxf.interceptor.LoggingInInterceptor` и `org.apache.cxf.interceptor.LoggingOutInterceptor` входящих и исходящих сообщений, которые выводят содержимое со-

общений в консоль. Конфигурационный Aegis-файл Web-сервиса, в данном примере, определяет с помощью атрибута `mappedName` имя XML-элемента, содержащего результат вызова Web-сервиса. Изменяя значение атрибута `mappedName` в Aegis-файле, можно видеть, как изменяется имя дочернего элемента тела входящего сообщения в консоли среды Eclipse.

Связывание данных XMLBeans

Платформа XMLBeans (**http://xmlbeans.apache.org/**) представляет собой инструмент связывания Java и XML и является частью проекта Apache Software Foundation XML.

Основной целью системы XMLBeans является наиболее полное связывание конструкций XML и XML Schema с конструкциями языка Java. Преимущество системы XMLBeans — легкость и простота доступа и обработки XML-данных Java-кодом. Система XMLBeans гарантирует обработку XML-документа, т. к. он существует, без потери его целостности.

Система XMLBeans генерирует Java-интерфейсы и Java-классы на основе схемы XML Schema для сериализации-десериализации XML-данных с их помощью. Система XMLBeans также предлагает программный интерфейс API для полного доступа к информационной модели XML Infoset XML-данных.

Программный интерфейс XMLBeans API состоит из трех основных частей:

- `XmlObject` — базовый класс Java-классов, генерируемых системой XMLBeans из схемы XML Schema;
- `XmlCursor` — класс, обеспечивающий доступ к набору XML Infoset и представляющий курсор информационной модели XML-данных;
- `SchemaType` — обеспечивает объектную модель схемы XML Schema.

После инсталляции платформы XMLBeans ее инструмент `scomp` позволит компилировать XSD-файл в JAR-файл, который будет содержать Java-типы, полностью соответствующие XML-типам скомпилированной XML-схемы. Далее XMLBeans API обеспечит манипуляцию XML-документами, соответствующими скомпилированной XML-схеме.

Поддержка системы XMLBeans платформой CXF представлена пакетами `org.apache.cxf.xmlbeans` и `org.apache.cxf.xmlbeans.tools` и может реализовываться на двух уровнях:

- опция `-db xmlbeans` CXF-инструмента `wsdl2java` позволяет генерировать XMLBeans-классы из XML-схем WSDL-описания вместо JAXB-классов;
- для использования XMLBeans-связывания вместо JAXB-связывания средой выполнения CXF на стороне сервера конструкция

```
<jaxws:dataBinding>
  <bean class="org.apache.cxf.xmlbeans.XmlBeansDataBinding" />
</jaxws:dataBinding>
```

включается в конфигурационный элемент `<jaxws:server>` или `<jaxws:endpoint>`. На стороне клиента такая же конструкция включается в конфигурационный элемент `<jaxws:client>`. Программным способом XMLBeans-связывание определяется методом `org.apache.cxf.frontend.ServerFactoryBean.getServiceFactory().setDataBinding(new org.apache.cxf.xmlbeans.XmlBeansDataBinding())` на стороне сервера и методом `org.apache.cxf.frontend.ClientProxyFactoryBean.getServiceFactory().setDataBinding(new org.apache.cxf.xmlbeans.XmlBeansDataBinding())` на стороне клиента.

Опции *features* и обработчики Interceptors

Рассмотрим пример создания пользовательских обработчиков Interceptor входящих и исходящих сообщений на стороне клиента и Web-сервиса.

Для создания Web-сервиса:

1. Откроем среду Eclipse, в меню **File** выберем пункты **New | Dynamic Web Project** и введем имя проекта CXFService. Нажав кнопку **New Runtime**, выберем **Apache Tomcat v7.0**.
2. Если сервер Tomcat еще не был интегрирован в среду Eclipse, укажем **Create a new local server**. Нажмем кнопку **Next** и кнопкой **Browse** определим каталог предварительно инсталлированного сервера Apache Tomcat v7.0 (**http://tomcat.apache.org/**). Кнопкой **Installed JREs** определим каталог JDK 6, в строке **JRE** укажем JDK 6 и нажмем кнопку **Finish**.
3. В окне **Dynamic Web Project** нажмем кнопку **Finish**.
4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **CXF 2.x Web Services**.
5. Если среда выполнения CXF еще не была интегрирована в среду Eclipse, перейдем по ссылке **Further configuration available | Configure installed runtimes**. На вкладке **CXF Runtime**, нажав кнопки **Add** и **Browse**, укажем каталог предварительно инсталлированной среды выполнения Apache CXF (**http://cxf.apache.org/download.html**). Нажмем кнопку **Finish** и на вкладке **CXF Runtime** отметим добавленную среду Apache CXF. На вкладке **JAX-WS** отметим **JAX-WS Annotation Generation** и дважды нажмем кнопку **OK**.
6. Закроем окно свойств проекта кнопкой **OK**.
7. В меню **Window** среды Eclipse выберем **Preferences | Web Services | Server and Runtime**, в раскрывающемся списке **Server runtime** выберем **Tomcat Server v7.0**, в раскрывающемся списке **Web service runtime** выберем **Apache CXF 2.x** и нажмем кнопку **OK**.
8. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFService** и выберем пункты **New | Class**, введем имя класса `CXFService` и имя пакета `cxfservice` и нажмем кнопку **Finish**.

9. В редакторе исходного кода дополним код класса `CXFService` методом `getHello()`:

```
public String getHello(String name) {
  return "Hello"+" "+ name;
}
```

10. Сохраним изменения кода класса `CXFService`, в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFService** и выберем пункты **New | Other | Web Services | Web Service**, нажмем кнопку **Next** и в строке **Service implementation** введем `cxfservice.CXFService`. Запустим генерацию CXF JAX-WS сервиса кнопкой **Finish**.

В результате CXF Web-сервис будет создан и развернут на платформе CXF поверх сервера Tomcat.

Созданный Web-сервис имеет сгенерированный конфигурационный Spring-файл beans.xml, содержащий конструкцию:

```
<jaxws:features>
  <bean class="org.apache.cxf.feature.LoggingFeature" />
</jaxws:features>
```

которая определяет для конечной точки включение CXF-опции `LoggingFeature`.

Опция `LoggingFeature` автоматически добавляет в цепочки обработчиков конечной точки классы реализации класса `org.apache.cxf.interceptor.AbstractLoggingInterceptor`, отвечающего за печать входящих и исходящих сообщений. Включение опции `LoggingFeature` аналогично добавлению обработчиков `org.apache.cxf.interceptor.LoggingInInterceptor` и `org.apache.cxf.interceptor.LoggingOutInterceptor` в соответствующие цепочки.

Таким образом, при вызове Web-сервиса CXFService, в консоль сервера Tomcat будут выводиться входящие и исходящие сообщения Web-сервиса. В среде Eclipse переключиться на нужную консоль можно кнопкой **Display Selected Console**.

Для просмотра сообщений, участвующих в обмене между клиентом и Web-сервисом, существуют, помимо опции `LoggingFeature`, другие не-CXF-инструменты, такие как Tcpmon (**http://java.net/projects/tcpmon/**), Web service monitoring tool (**http://java.net/projects/wsmonitor/**), soapUI (**http://soapui.org/**) и др.

Классы опций платформы CXF расширяют класс `org.apache.cxf.feature.AbstractFeature`, который предоставляет дополнительные возможности настройки объектов `Server`, `Client` и `Bus`. Для создания пользовательской опции необходимо написать класс, расширяющий класс `AbstractFeature`, и добавить опцию на стороне клиента или сервиса с помощью аннотации `@org.apache.cxf.feature.Features`, метода `setFeatures()` классов `ServerFactoryBean` и `ClientFactoryBean`, или используя конфигурационный Spring-файл.

Платформа CXF предлагает, помимо опции `LoggingFeature`, следующие опции, которые могут быть включены в качестве дочерних элементов конфигурационного элемента `< features>`:

- org.apache.cxf.clustering.FailoverFeature позволяет, в случае падения первоначальной конечной точки, перейти клиенту к другой конечной точке Web-сервиса (элемент <failover> пространства имен http://cxf.apache.org/clustering);
- org.apache.cxf.feature.FastInfosetFeature включает использование FastInfoset с первого сообщения (элемент <bean>);
- org.apache.cxf.feature.StaxTransformFeature позволяет динамически преобразовывать входящие и исходящие сообщения с помощью следующих свойств (дочерний элемент <property name="..."> элемента <bean id="..." class="org.apache.cxf.feature.StaxTransfromFeature">):
  - outTransformElements изменяет имена исходящих элементов и идентификаторы пространств имен с помощью дочерних элементов <entry key="..." value="..."/> элемента <map>;
  - inTransformElements изменяет имена входящих элементов и идентификаторы пространств имен;
  - outAppendElements добавляет новые исходящие элементы и значения;
  - inAppendElements добавляет новые входящие элементы и значения;
  - outDropElements исключает исходящие элементы с помощью дочерних элементов <value>...</value> элемента <list>;
  - inDropElements исключает входящие элементы;
  - attributesAsElements преобразует атрибуты в элементы;
- org.apache.cxf.transport.http.gzip.GZIPFeature позволяет отправлять сообщения в GZIP-формате;
- org.apache.cxf.binding.coloc.feature.ColocFeature позволяет Web-сервису работать с различными транспортными протоколами (элемент <enableColoc> пространства имен http://cxf.apache.org/binding/coloc);
- org.apache.cxf.databinding.stax.StaxDataBindingFeature — связывание данных выполняется с помощью технологии Stax (элемент <bean> пространства имен http://springframework.org);
- org.apache.cxf.management.interceptor.ResponseTimeFeature включает управление временем ответа конечной точки (элемент <bean> пространства имен http://springframework.org);
- org.apache.cxf.ws.addressing.WSAddressingFeature включает поддержку WS-Addressing (элемент <addressing> пространства имен http://cxf.apache.org/ws/addressing);
- org.apache.cxf.ws.policy.WSPolicyFeature — включает поддержку WS-Policy (элемент <policies> пространства имен http://cxf.apache.org/policy-config);
- org.apache.cxf.ws.rm.feature.RMFeature включает поддержку WS-RM (элемент <reliableMessaging> пространства имен http://cxf.apache.org/ws/rm/manager).

Для создания клиента CXF Web-сервиса:

1. В меню **File** среды Eclipse выберем пункты **New | Dynamic Web Project**, введем имя проекта CXFClient и нажмем кнопку **Finish**.
2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **CXF 2.x Web Services** и нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFClient** и выберем пункты **New | Class**, введем имя класса `CXFClient` и имя пакета `cxfclient` и нажмем кнопку **Finish**.
4. В окне редактора исходного кода модифицируем код класса `CXFClient` (листинга 5.1).

Листинг 5.1. Код класса `CXFClient`

```
package cxfclient;
import org.apache.cxf.jaxws.endpoint.dynamic.JaxWsDynamicClientFactory;
import org.apache.cxf.endpoint.Client;
public class CXFClient {
  public static void main(String[] args) {
    JaxWsDynamicClientFactory dcf =
      JaxWsDynamicClientFactory.newInstance();
    Client client = dcf.createClient(
      "http://localhost:8080/CXFService/services/cxfservice?wsdl");
    InClientInterceptor inintr=new InClientInterceptor();
    OutClientInterceptor outintr=new OutClientInterceptor();
    client.getInInterceptors().add(inintr);
    client.getOutInterceptors().add(outintr);
    try {
      Object[] response=client.invoke("getHello","User");
      System.out.println(response[0]);
    } catch (Exception e) {
      e.printStackTrace();
}  }}
```

В коде метода `main()` класса `CXFClient` применяется динамическое создание клиента Web-сервиса и добавление к клиенту обработчиков Interceptor входящих и исходящих сообщений.

Для создания обработчиков Interceptor:

1. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFClient | Java Resources src: | cxfclient** и выберем пункты **New | Class**, введем имя класса `InClientInterceptor` и нажмем кнопку **Finish**.
2. Дополним код класса `InClientInterceptor` (листинг 5.2).

Листинг 5.2. Код класса `InClientInterceptor`

```
package cxfclient;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.message.Message;
public class InClientInterceptor extends AbstractPhaseInterceptor <Message>
{ InClientInterceptor(){
  super(Phase.RECEIVE);}
public void handleMessage(Message message) throws Fault {
  System.out.println("INInterceptor");}}
```

3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFClient | Java Resources src: | cxfclient** и выберем пункты **New | Class**, введем имя класса `OutClientInterceptor` и нажмем кнопку **Finish**.
4. Дополним код класса `OutClientInterceptor` (листинг 5.3).

Листинг 5.3. Код класса `OutClientInterceptor`

```
package cxfclient;
import org.apache.cxf.phase.AbstractPhaseInterceptor;
import org.apache.cxf.phase.Phase;
import org.apache.cxf.interceptor.Fault;
import org.apache.cxf.message.Message;
public class OutClientInterceptor extends AbstractPhaseInterceptor <Message> {
  OutClientInterceptor(){
     super(Phase.SEND);  }
public void handleMessage(Message message) throws Fault {
  System.out.println("OUTIntercptor");}}
```

5. Щелкнув правой кнопкой мыши на узле **CXFClient | Java Resources src: | cxfclient | CXFClient.java** и выбрав пункты **Run As | Java Application**, можно увидеть в консоли среды Eclipse, что обработчики Interceptor действительно срабатывают при получении и отправке сообщения.

Создадим такие же обработчики Interceptor в пакете `cxfservice` на стороне Web-сервиса CXFService и определим их конфигурацию с помощью включения в Spring-файл дочерних элементов `<jaxws:inInterceptors>` и `<jaxws:outInterceptors>` элемента `<jaxws:endpoint>`:

```
<jaxws:inInterceptors>
  <bean class="cxfservice.InServiceInterceptor" />
</jaxws:inInterceptors>
<jaxws:outInterceptors>
  <bean class="cxfservice.OutServiceInterceptor" />
</jaxws:outInterceptors>
```

Заново развернем сервис CXFService и, вызвав его с помощью клиента CXFClient, увидим в консоли среды Eclipse (кнопка **Display Selected Console**), что обработчики Interceptor также срабатывают и на стороне Web-сервиса при получении и отправке сообщения.

На компакт-диске

Проекты CXFService и CXFClient примера использования обработчиков Interceptor находятся в папке Примеры\Глава5\Interceptors компакт-диска.

Применение обработчиков Interceptor обеспечивает низкоуровневый доступ к входящим и исходящим сообщениям на различных стадиях обработки сообщения — начиная от транспортного уровня и заканчивая уровнем приложения. Создание пользовательских обработчиков Interceptor дает возможность извлекать различные части сообщения, включая его заголовки (`org.apache.cxf.binding.soap.SoapMessage`) и вложения (`org.apache.cxf.attachment.AttachmentDeserializer`). Интерфейс `org.apache.cxf.message.Message` своим методом `<T> T getContent(Class<T> format)` позволяет получить сообщение в виде таких объектов, как `InputStream`, `XMLStreamReader`, `List <Object>` и др. В частности, с помощью пользовательских обработчиков Interceptor можно организовать базовую аутентификацию клиента, используя на стороне клиента обработчик `org.apache.cxf.binding.soap.interceptor.AbstractSoapInterceptor` исходящих сообщений в фазе Phase.WRITE, добавляющий в объект `org.apache.cxf.binding.soap.SoapMessage` заголовок с маркером защиты, содержащим логин/пароль клиента. На стороне сервиса обработчик `AbstractSoapInterceptor` в фазе Phase.PRE_INVOKE может извлекать маркер защиты из входящего объекта `SoapMessage` и производить авторизацию клиента.

Конфигурационный элемент `<jaxws:endpoint>` (рис. 5.2) Spring-файла позволяет устанавливать конфигурацию конечной точки JAX-WS Web-сервиса для развертывания и может иметь следующие атрибуты и дочерние элементы:

- атрибут `endpointName` — имя конечной точки, соответствует WSDL-элементу `<port>`, в формате "`ns:`*`ENDPOINT_NAME`*";
- атрибут `publish` — если `true`, тогда конечная точка публикуется сразу;
- атрибут `serviceName` — имя сервиса, соответствует WSDL-элементу `<service>`, в формате "`ns:`*`SERVICE_NAME`*";
- атрибут `wsdlLocation` — адрес WSDL-документа;
- атрибут `bindingUri` — идентификатор протокола связывания;
- атрибут `address` — адрес публикации конечной точки;
- атрибут `bus` — идентификатор элемента, конфигурирующего шину Bus;
- атрибут `implementor` — имя SEI-класса;
- атрибут `implementorClass` — имя SEI-класса, если значение атрибута `implementor` является ссылкой на элемент `<bean>`, определенный согласно Spring AOP;
- атрибут `createdFromAPI` — если `true`, тогда конечная точка публикуется программным способом;

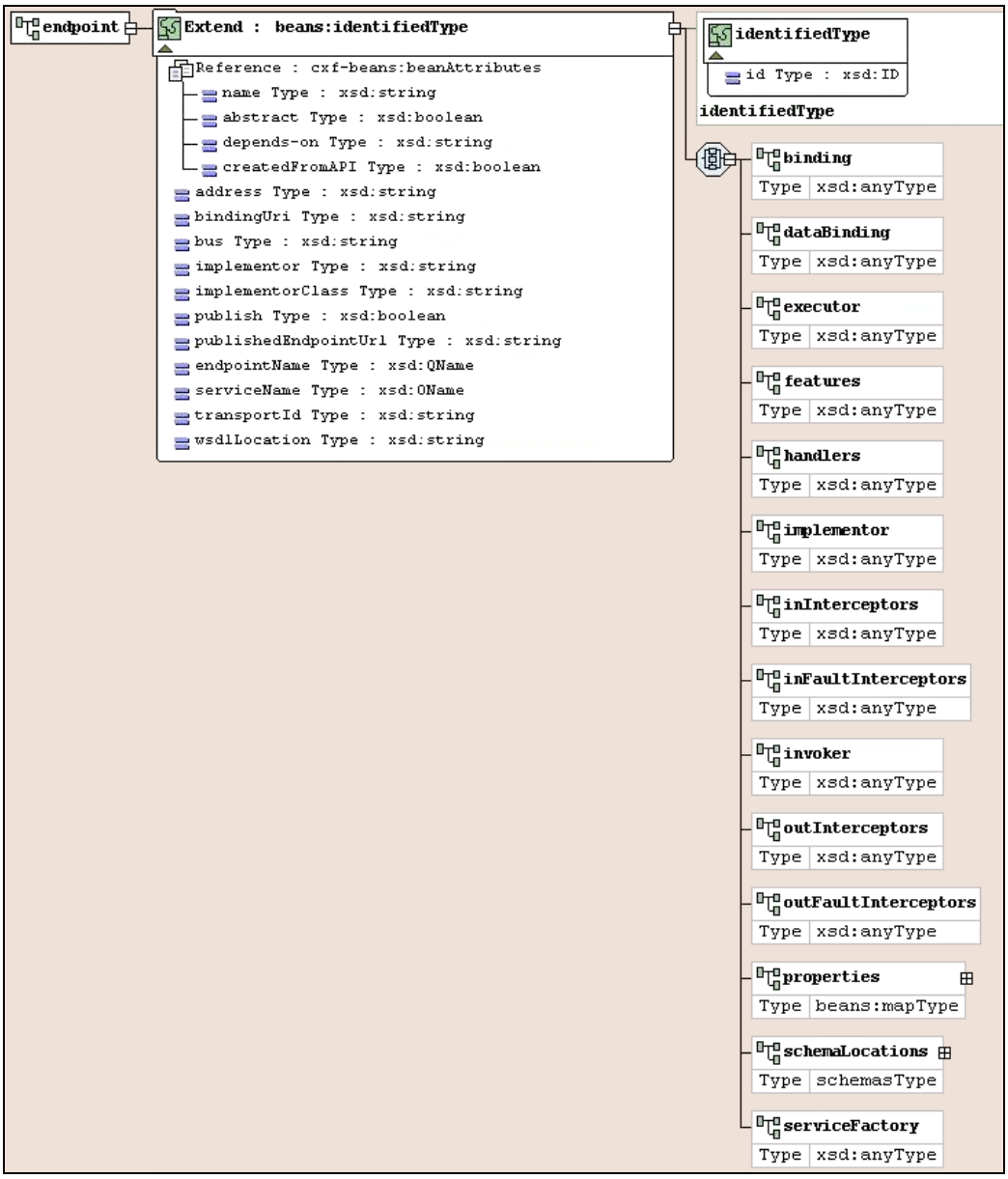


Рис. 5.2. Общая схема элемента `<jaxws:endpoint>`

- атрибут `publishedEndpointUrl` — публичный адрес конечной точки;
- элемент `<jaxws:executor>` — имя класса `Executor` в форме `<bean class="..."/>`;
- элемент `<jaxws:inInterceptors>` — обработчики входящих сообщений в форме `<bean class="..."/>` или `<ref bean="..."/>`;
- элемент `<jaxws:inFaultInterceptors>` — обработчики входящих ошибок в форме `<bean class="..."/>` или `<ref bean="..."/>`;

- элемент `<jaxws:outInterceptors>` — обработчики исходящих сообщений в форме `<bean class="..."/>` или `<ref bean="..."/>`;
- элемент `<jaxws:outFaultInterceptors>` — обработчики исходящих ошибок в форме `<bean class="..."/>` или `<ref bean="..."/>`;
- элемент `<jaxws:handlers>` — JAX-WS обработчики в форме `<bean class="..."/>` или `<ref bean="..."/>`;
- элемент `<jaxws:properties>` — свойства конечной точки;
- элемент `<jaxws:dataBinding>` — класс `DataBinding` в форме `<bean class="..."/>`;
- элемент `<jaxws:binding>` — класс `BindingFactory` в форме `<bean class="..."/>`;
- элемент `<jaxws:features>` — опции `Features` конечной точки в форме `<bean class="..."/>` или `<ref bean="..."/>`;
- элемент `<jaxws:invoker>` — класс `Invoker` в форме `<bean class="..."/>`;
- элемент `<jaxws:schemaLocations>` — адреса XML-схем, дочерние элементы `<schemaLocation>`;
- элемент `<jaxws:serviceFactory>` — класс `ServiceFactory` в форме `<bean class="..."/>`.

Конфигурация JAX-WS Web-сервиса также может быть определена с помощью конфигурационного элемента `<jaxws:server>` (рис. 5.3).

Развертывание клиента JAX-WS Web-сервиса также может быть настроено Spring-файлом посредством конфигурационного элемента `<jaxws:client>` (рис. 5.4) со следующими атрибутами и дочерними элементами:

- атрибут `id` — идентификатор элемента;
- атрибут `address` — адрес вызываемого Web-сервиса;
- атрибут `serviceClass` — имя SEI-интерфейса;
- атрибут `serviceName` — аналогично `<jaxws:endpoint>`;
- атрибут `endpointName` — аналогично `<jaxws:endpoint>`;
- атрибут `bindingId` — аналогично `<jaxws:endpoint>`;
- атрибут `bus` — аналогично `<jaxws:endpoint>`;
- атрибут `username` — логин клиента;
- атрибут `password` — пароль клиента;
- атрибут `wsdlLocation` — аналогично `<jaxws:endpoint>`;
- атрибут `createdFromAPI` — если `true`, тогда клиент создан программным способом;
- элемент `<jaxws:inInterceptors>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:inFaultInterceptors>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:outInterceptors>` — аналогично `<jaxws:endpoint>`;

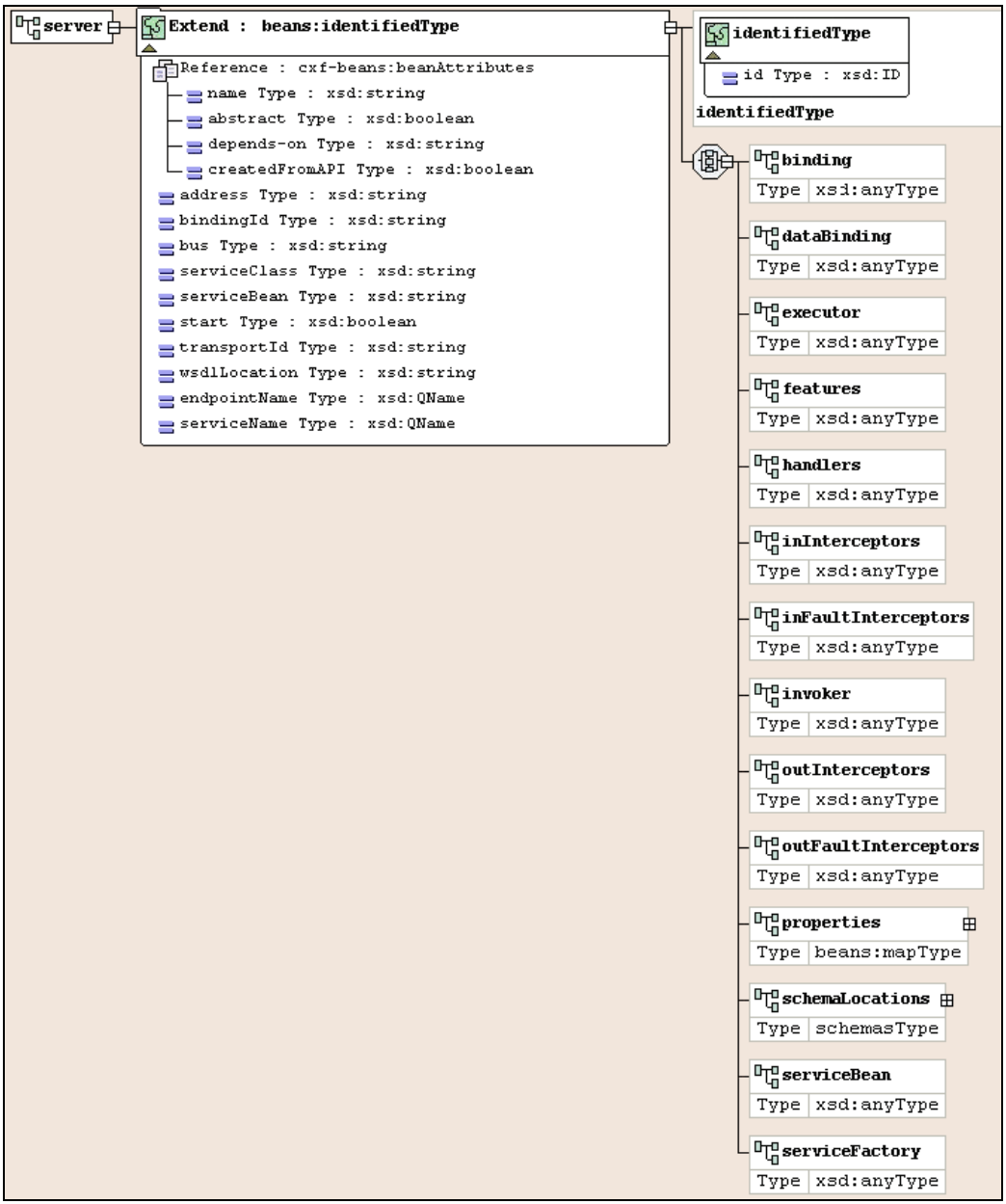


Рис. 5.3. Общая схема элемента `<jaxws:server>`

- элемент `<jaxws:outFaultInterceptors>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:features>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:handlers>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:properties>` — свойства клиента, например `<entry key="mtom-enabled" value="true"/>`;

- элемент `<jaxws:dataBinding>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:binding>` — аналогично `<jaxws:endpoint>`;
- элемент `<jaxws:conduitSelector>` — класс `ConduitSelector`, выбирающий транспортный канал для исходящего сообщения.

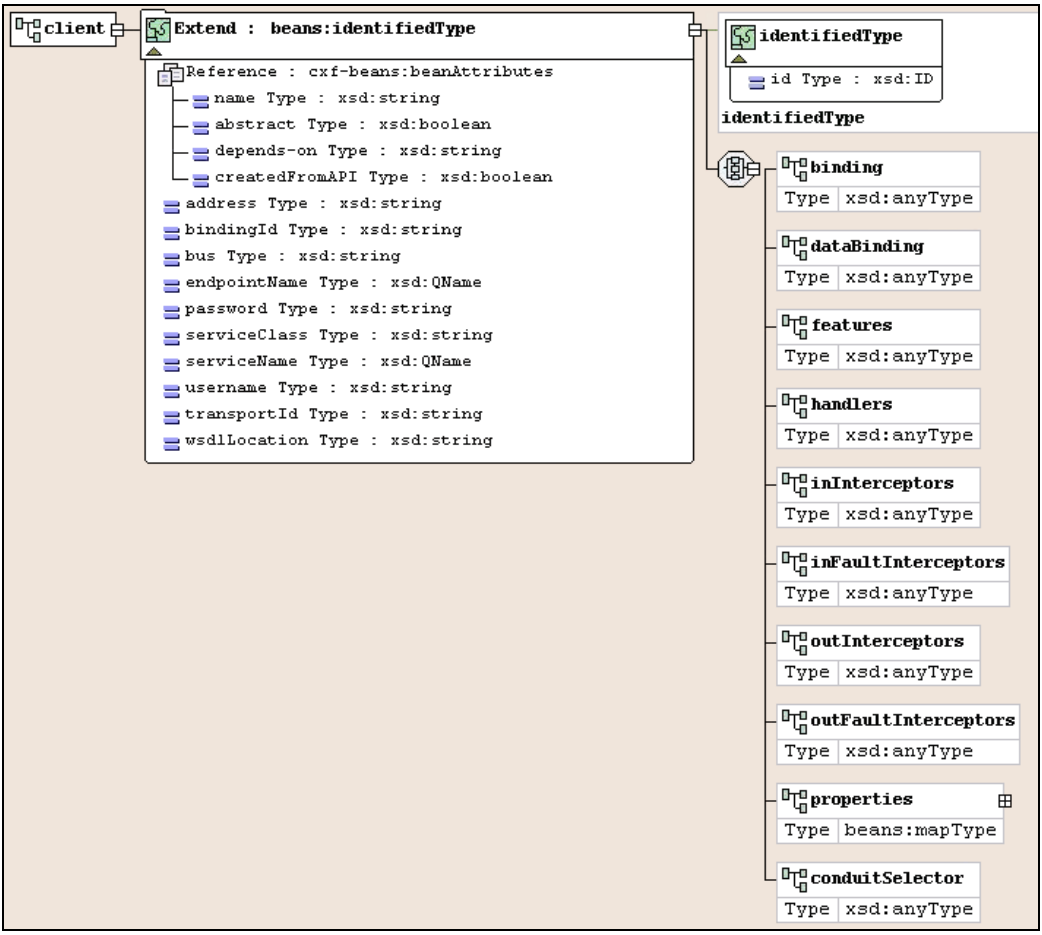


Рис. 5.4. Общая схема элемента `<jaxws:client>`

Протоколы передачи сообщений

Поддержка протокола SOAP/HTTP

Платформа CXF обеспечивает поддержку протокола SOAP версии 1.1 и протокола SOAP версии 1.2.

Для определения протокола передачи сообщений SOAP 1.1/HTTP на стороне Web-сервиса используются WSDL-элементы `<soap:binding>`, `<soap:operation>`, `<soap:body>` и `<soap:address>` пространства имен `xmlns:soap="http://schemas.`

xmlsoap.org/wsdl/soap/", являющиеся дочерними элементами элемента <wsdl:binding> и указывающие транспортный протокол, стиль и адрес конечной точки.

Для определения протокола передачи сообщений SOAP 1.2/HTTP на стороне Web-сервиса также используются дочерние WSDL-элементы <soap12:binding>, <soap12:operation>, <soap12:body> и <soap12:address> элемента <wsdl:binding>, только уже пространства имен xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/".

В среде Eclipse использование протокола SOAP 1.1 или SOAP 1.2 может быть определено двумя способами.

В диалоговом окне, открываемом из меню **Window** выбором **Preferences | Web Services | CXF 2.x Preferences**, на вкладке **Java2WS** конфигурации CXF-инструмента java2ws в раскрывающемся списке **Default SOAP Binding** можно выбрать **SOAP 1.1** или **SOAP 1.2**.

В процессе создания Web-сервиса для проекта Dynamic Web Project при выборе **New | Other | Web Service | Next | Next | Next | Next** также можно указать в раскрывающемся списке **Default SOAP Binding** значение **SOAP 1.1** или **SOAP 1.2** определения опций CXF-инструмента java2ws.

При этом будет сгенерирован WSDL-документ необходимой конфигурации.

CXF-клиент Web-сервиса применяет протокол SOAP 1.1/HTTP по умолчанию, поэтому для его принудительного использования протокола SOAP 1.1 ничего делать не нужно.

Чтобы CXF-клиент Web-сервиса использовал протокол SOAP 1.2, необходимо его дополнительное конфигурирование, которое можно осуществить с помощью конфигурационного Spring-файла beans.xml, включающего в себя дочерний элемент

```
<jaxws:binding>
    <soap:soapBinding version="1.2"/>
</jaxws:binding>
```

элемента <jaxws:client>.

Для создания контекста клиентского приложения, определяемого Spring-файлом, в коде класса клиента необходимо использовать конструктор класса org.springframework.context.support.ClassPathXmlApplicationContext с указанием адреса Spring-файла beans.xml. Сам же клиент Web-сервиса создается методом getBean() объекта ClassPathXmlApplicationContext.

При вызове Web-сервиса его клиентом с использованием протокола SOAP 1.1/HTTP или протокола SOAP 1.2/HTTP содержимое входящих и исходящих сообщений будет отличаться.

Сообщения протокола SOAP 1.1/HTTP будут иметь заголовок Content-Type: text/xml и пространство имен xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/", а сообщения протокола SOAP 1.2/HTTP — заголовок Content-Type: application/soap+xml и пространство имен xmlns:soap="http://www.w3.org/2003/05/soap-envelope".

На компакт-диске

Примеры Web-сервиса и его клиента, использующих протоколы SOAP 1.1/HTTP и SOAP 1.2/HTTP, можно посмотреть в папках Примеры\Глава5\Soap11 и Примеры\Глава5\Soap12 компакт-диска.

Поддержка протокола XML/HTTP

Использование протокола XML/HTTP позволяет передавать и получать XML-сообщения, не соответствующие SOAP-протоколу.

Протокол XML/HTTP обмена сообщениями между Web-сервисом и его клиентом можно установить двумя способами. Первый способ — это создание Web-сервиса и его клиента, использующих протокол xformat платформы CXF. Второй способ — это создание JAX-WS Dispatch-клиента и Provider-сервиса, использующих HTTP-связывание.

Протокол xformat платформы CXF имеет идентификатор `http://cxf.apache.org/bindings/xformat`, и его применение на стороне Web-сервиса можно определить с помощью дочерних WSDL-элементов `<xformat:binding>` и `<xformat:body>` (пространство имен `xmlns:xformat="http://cxf.apache.org/bindings/xformat"`) элемента `<wsdl:binding>`.

Программным способом применение протокола xformat на стороне Web-сервиса можно установить с помощью аннотации `@javax.xml.ws.BindingType(value = "http://cxf.apache.org/bindings/xformat")` SEI-класса или при публикации конечной точки методом `setBindingId()` класса `org.apache.cxf.jaxws.JaxWsServerFactoryBean`.

На стороне клиента Web-сервиса применение протокола xformat можно установить методом `addPort()` класса `javax.xml.ws.Service` или методом `setBindingId()` класса `org.apache.cxf.jaxws.JaxWsProxyFactoryBean`.

Определить использование протокола xformat можно также конфигурационным Spring-файлом.

На компакт-диске

Пример Web-сервиса и его клиента, использующих протокол xformat, находится в папке Примеры\Глава5\HTTP компакт-диска. Данные проекты были созданы в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

При вызове Web-сервиса его клиентом, в данном примере, между клиентом и Web-сервисом производится обмен сообщениями, имеющими чистый XML-формат без SOAP-элементов:

```
Входящее сообщение Web-сервиса:
Encoding: UTF-8
Content-Type: text/xml
Headers: {Accept=[*/*]}
Payload: <ns2:getHello
xmlns:ns2="http://cxfservice/"><arg0>User</arg0></ns2:getHello>
Исходящее сообщение Web-сервиса:
Response-Code: 200
```

```
Encoding: UTF-8
Content-Type: text/xml;charset=UTF-8
Headers: {content-type=[text/xml;charset=UTF-8], Content-Length=[103],
Server=[Jetty(7.2.2.v20101205)]}
Payload: <ns2:getHelloResponse xmlns:ns2="http://cxfservice/"><return>Hello
User</return></ns2:getHelloResponse>
```

Использование протокола XML/HTTP обеспечивает также HTTP-связывание технологии JAX-WS.

HTTP-связывание имеет идентификатор `http://www.w3.org/2004/08/wsdl/http` и представлено полем `HTTP_BINDING` интерфейса `javax.xml.ws.http.HTTPBinding`. Для его применения необходимо создать Web-сервис и его клиента, использующих низкоуровневую обработку входящих и исходящих сообщений. Такой низкоуровневый доступ к сообщениям обеспечивают интерфейсы `javax.xml.ws.Provider<T>` и `javax.xml.ws.Dispatch<T>` технологии JAX-WS.

Установить HTTP-связывание на стороне Web-сервиса можно с помощью аннотации `@javax.xml.ws.BindingType` или при публикации конечной точки методом `create()` класса `javax.xml.ws.Endpoint`. На стороне клиента HTTP-связывание можно указать методом `addPort()` класса `javax.xml.ws.Service`.

В WSDL-документе HTTP-связывание представлено элементами `<http:binding>`, `<http:operation>` и `<http:address>`, указывающими использование HTTP-протокола, адрес операции и адрес конечной точки.

Платформа CXF дает возможность устанавливать конфигурацию обмена сообщениями по транспортному протоколу HTTP, определяя понятие HTTP Conduit или HTTP-канала доставки сообщений от клиента Web-сервису и понятие HTTP Destination или слушателя клиентских запросов.

Конфигурацию свойств Conduit-канала на стороне клиента обеспечивает элемент `<http-conf:conduit>` (пространство имен `xmlns:http-conf="http://cxf.apache.org/transports/http/configuration"`) конфигурационного Spring-файла. Элемент `<http-conf:conduit>` имеет следующие атрибуты и дочерние элементы:

- атрибут `name` связывает Conduit-канал с WSDL-портом в форме `"{WSDL Namespace}portName.http-conduit"`. Значением атрибута также могут быть выражения `"*.http-conduit"` и `http://.../.*`;
- элемент `<http-conf:client>` определяет свойства HTTP-соединения с помощью следующих атрибутов:
  - `ConnectionTimeout` — максимальное время установки соединения, по умолчанию 30 000 миллисекунд. Значение 0 определяет бесконечный предел для установки соединения;
  - `ReceiveTimeout` — максимальное время ожидания ответа, по умолчанию 60 000 миллисекунд. Значение 0 определяет бесконечное время ожидания ответа;
  - `AutoRedirect` — если `true`, тогда клиент автоматически следует перенаправлению запроса сервером, по умолчанию `false`;

- `MaxRetransmits` — максимальное количество раз отправки клиентом перенаправленного запроса, по умолчанию –1 — бесконечное количество раз;
- `AllowChunking` — если `true` (по умолчанию), тогда клиент посылает запрос по частям;
- `ChunkingThreshold` — порог переключения на отправку запроса по частям, по умолчанию пороговый размер сообщения равен 4 Кбайт;
- `Accept` — соответствует HTTP-заголовку `Accept property`;
- `AcceptLanguage` — соответствует HTTP-заголовку `AcceptLanguage`;
- `AcceptEncoding` — соответствует HTTP-заголовку `AcceptEncoding`;
- `ContentType` — соответствует HTTP-заголовку `ContentType`, по умолчанию `text/xml`;
- `Host` — соответствует HTTP-заголовку `Host`;
- `Connection` — возможные значения: `Keep-Alive` (по умолчанию) — соединение поддерживается после первоначального запроса, `close` — соединение закрывается после первоначального обмена сообщениями;
- `CacheControl` — параметры кэширования;
- `Cookie` — cookies для запросов;
- `BrowserType` — тип браузера клиента;
- `Referer` — соответствует HTTP-заголовку `Referer`;
- `DecoupledEndpoint` — URL-адрес для получения ответа на запрос;
- `ProxyServer` — URL-адрес Proxy-сервера;
- `ProxyServerPort` — номер порта Proxy-сервера;
- `ProxyServerType` — тип Proxy-сервера, возможные значения: `HTTP` (по умолчанию), `SOCKS`;

❒ элемент `<http-conf:authorization>` — параметры базовой аутентификации, дочерние элементы — `<conf-sec:UserName>` и `<conf-sec:Password>`;

❒ элемент `<http-conf:proxyAuthorization>` — параметры базовой аутентификации для HTTP Proxy-сервера;

❒ элемент `<http-conf:tlsClientParameters>` — параметры использования протокола SSL/TLS с помощью атрибутов:

- `certConstraints` — ограничение на сертификат протокола SSL;
- `cipherSuites` — идентификатор набора алгоритмов шифрования протокола SSL;
- `cipherSuitesFilter` — ограничение на возможные наборы алгоритмов шифрования протокола SSL;

- disableCNcheck — если true (что позволяет использовать локальный хост), тогда отключается проверка имени хоста в HTTPS URL-запросе относительно имени Common Name (CN) сертификата, по умолчанию false;
- jsseProvider — JSSE-реализация SSL-протокола;
- keyManagers — объект javax.net.ssl.KeyManager, обеспечивающий X509-сертификат;
- secureRandomParameters — объект java.security.SecureRandom;
- secureSocketProtocol — имя протокола, например "SSL", "TLS" (по умолчанию), "TLSv1";
- trustManagers — объект javax.net.ssl.TrustManager;
- useHttpsURLConnectionDefaultSslSocketFactory — если true, тогда используется объект HttpsURLConnection.getDefaultSSLSocketFactory() для создания защищенного соединения, по умолчанию false;
- useHttpsURLConnectionDefaultHostnameVerifier — если true, тогда используется объект HttpsURLConnection.getDefaultHostnameVerifier() для создания защищенного соединения, по умолчанию false;

❒ элемент <http-conf:basicAuthSupplier> — имя класса, обеспечивающего базовую аутентификацию;

❒ элемент <http-conf:trustDecider> — имя класса, использующего объект java.net.URLConnection для HTTPS-протокола.

Конфигурацию свойств Conduit-канала клиента можно также установить с помощью дочернего WSDL-элемента <http-conf:client> элемента <port> или программным способом с использованием объектов org.apache.cxf.transport.http.HTTPConduit и org.apache.cxf.transports.http.configuration.HTTPClientPolicy.

Конфигурацию свойств Destination-слушателя на стороне сервера обеспечивает элемент <http:destination> (пространство имен xmlns:http-conf="http://cxf.apache.org/transports/http/configuration) конфигурационного Spring-файла. Элемент <http:destination> имеет следующие атрибуты и дочерние элементы:

❒ атрибут name указывает слушателя клиентских запросов в форме "{WSDL_endpoint_target_namespace}PortName.http-destination";

❒ элемент <http-conf:server> определяет свойства HTTP-соединения с помощью атрибутов:

- ReceiveTimeout — максимальное время ожидания запроса, по умолчанию 30 000 миллисекунд;
- SuppressClientSendErrors — если false (по умолчанию), тогда при возникновении ошибки получения запроса исключения обрабатываются;
- SuppressClientReceiveErrors — если false (по умолчанию), тогда при возникновении ошибки отправки ответа исключения обрабатываются;

- `HonorKeepAlive` — если `true` (по умолчанию), тогда запросы типа keep-alive принимаются;
- `RedirectURL` — соответствует HTTP-заголовку `RedirectURL`;
- `CacheControl` — параметры кэширования;
- `ContentLocation` — URL-адрес ресурса для ответа;
- `ContentType` — соответствует HTTP-заголовку `ContentType`;
- `ContentEncoding` — соответствует HTTP-заголовку `ContentEncoding`;
- `ServerType` — тип сервера, например Apache/1.2.5;

❒ элемент `<http-conf:contextMatchStrategy>` определяет параметры обработки HTTP-запросов;

❒ `http-conf:fixedParameterOrder` — если `true`, тогда порядок параметров HTTP-запроса фиксирован.

Конфигурацию свойств Destination-слушателя сервера можно также установить с помощью дочернего WSDL-элемента `<http-conf:server>` элемента `<port>`.

Поддержка протокола HTTPS

Платформа CXF может обеспечивать защиту сообщений, передаваемых по протоколу HTTP, их шифрованием по протоколу SSL или TLS. При этом может быть реализовано как шифрование без аутентификации клиента, так и шифрование с аутентификацией клиента.

При HTTPS-шифровании без аутентификации клиента, на стороне клиента генерируется ключ, с помощью которого производится шифрование сообщений, участвующих в обмене между клиентом и сервером. Сгенерированный ключ шифруется публичным ключом сертификата сервера и передается серверу, который расшифровывает его приватным ключом своего сертификата.

В случае HTTPS-шифровании с аутентификацией клиента на стадии рукопожатия сервер запрашивает сертификат клиента и, после его получения, сверяет сертификат клиента со списком доверенных сертификатов. Только после успешной аутентификации клиента между клиентом и сервером устанавливается защищенная сессия.

Таким образом, для использования протокола HTTP/SSL или TLS требуются сертификаты сервера и клиента.

> ***На компакт-диске***
>
> Пример Web-сервиса и его клиента, обменивающихся сообщениями по протоколу HTTP/SSL, находится в папке Примеры\Глава5\HTTPS компакт-диска.

В данном примере Web-сервис представлен проектом CXFHTTPSService, а его клиент — проектом CXFHTTPSClient. Эти проекты были созданы в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Для создания хранилища ключей, доверительного хранилища ключей и генерации публичных/приватных ключей можно воспользоваться Java-инструментом `keytool`. Для более удобной работы можно использовать также приложение с графическим интерфейсом, например, KeyTool IUI (**http://www.softpedia.com/get/Security/Security-Related/KeyTool-IUI.shtml**).

Для проекта CXFHTTPSService было создано хранилище ключей cxfservice.jks, содержащее сгенерированные ключи сервера, и доверительное хранилище trustservice.jks, содержащее импортированный публичный ключ клиента. Для проекта CXFHTTPSClient было создано хранилище ключей cxfclient.jks, содержащее сгенерированные ключи клиента, и доверительное хранилище trustclient.jks, содержащее импортированный публичный ключ сервера.

Проект CXFHTTPSService содержит в пакете `cxfservice`:

- SEI-интерфейс `CXFService`;
- SEI-класс `CXFServiceImp`;
- главный класс `CXFHTTPSServer`, который в методе `main()` публикует конечную точку по адресу **https://localhost:8080/CXFHTTPSService/services/CXFServiceImpPort**;
- конфигурационный Spring-файл beans.xml, используемый методом `org.apache.cxf.bus.spring.SpringBusFactory.createBus()` для создания объекта `org.apache.cxf.Bus` класса `CXFHTTPSServer`, определяющего контекст приложения.

Конфигурационный Spring-файл beans.xml проекта CXFHTTPSService включает в себя элемент `<http:destination>`, указывающий порт Web-сервиса CXFService как слушателя HTTP-запросов, и элемент `<httpj:engine-factory>` (пространство имен `xmlns:httpj="http://cxf.apache.org/transports/http-jetty/configuration`), определяющий конфигурацию экземпляра Jetty-сервера, встроенного в среду выполнения платформы CXF. При этом дочерний элемент `<sec:clientAuthentication want="true" required="true"/>` элемента `<httpj:tlsServerParameters>` определяет требование к аутентификации клиента с использованием его сертификата.

Элемент `<httpj:engine-factory>` может иметь следующие атрибуты и дочерние элементы:

- атрибут `bus` — тип Bus-шины, по умолчанию `"cxf"`;
- атрибут `id` — идентификатор элемента;
- элемент `identifiedTLSServerParameters` — набор свойств обеспечения безопасности HTTP-соединения с помощью дочернего элемента `<tlsServerParameters>` (рис. 5.5), имеющего атрибуты и дочерние элементы:
  - атрибут `jsseProvider` — имя Java Secure Socket Extension (JSSE) реализации, например `"SunJSSE"`;
  - атрибут `secureSocketProtocol` — имя протокола шифрования: `"SSL"`, `"TLS"` или `"TLSv1"`;
  - элемент `<keyManagers>` — определяет конфигурацию хранилища ключей с помощью атрибутов `keyPassword`, `provider`, `factoryAlgorithm` ("PKIX") и до-

чернего элемента <keyStore>. Элемент <keyStore> имеет атрибуты type, password, provider ("SUN"), url, file и resource;

- элемент <trustManagers> — определяет конфигурацию доверительного хранилища ключей и сертификатов с помощью атрибутов provider, factoryAlgorithm и дочерних элементов <keyStore> *(см. ранее)* и <certStore> (хранилище сертификатов, атрибуты url, file, resource);
- элемент <cipherSuites> — определяет поддерживаемые наборы алгоритмов шифрования с помощью дочерних элементов <cipherSuite>;
- элемент <cipherSuitesFilter> — шаблоны, фильтрующие возможные наборы алгоритмов шифрования с помощью дочерних элементов <include> и <exclude>;
- элемент <secureRandomParameters> — определяет SecureRandom-параметры;
- элемент <clientAuthentication> — требования к аутентификации клиента, определяемые с помощью атрибутов want и required;
- элемент <certConstraints> — ограничения для сертификатов, определяемые с помощью дочерних элементов <SubjectDNConstraints> и <IssuerDN Constraints>;

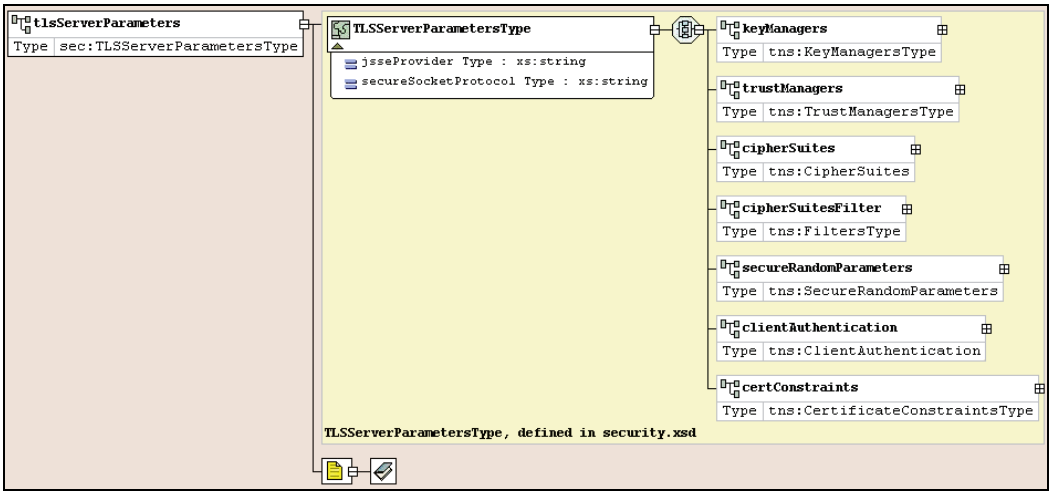


Рис. 5.5. Общая схема элемента <tlsServerParameters>

- элемент <identifiedThreadingParameters> — набор свойств пула потоков экземпляра Jetty-сервера. Имеет атрибут id и дочерний элемент <threadingParameters> с атрибутами minThreads и maxThreads;
- элемент <engine> — конфигурация среды выполнения Jetty, имеет следующие атрибуты и дочерние элементы:
  - атрибут port — номер порта экземпляра Jetty-сервера;
  - атрибут host — адрес хоста сервера;

- атрибут `continuationsEnabled` — если `true`, тогда интерфейс Jetty Continuations поддерживается;
- элемент `<tlsServerParameters>` или ссылку на него — *(см. ранее)*;
- элемент `<threadingParameters>` или ссылку на него — *(см. ранее)*;
- элемент `<connector>` — класс реализации HTTP Connector;
- элемент `<handlers>` — обработчики Jetty Handler;
- элемент `<sessionSupport>` — если `true`, тогда инициируется создание сессии, по умолчанию `false`;
- элемент `<reuseAddress>` — если `true` (по умолчанию), тогда устанавливается флаг `SO_REUSEADDR`.

Конфигурацию экземпляра Jetty-сервера можно также устанавливать программным способом, используя CXF API.

Проект CXFHTTPSClient содержит в пакете `cxfclient` главный класс клиентского приложения `CXFHTTPSClient` и конфигурационный Spring-файл beans.xml, используемый объектом `org.springframework.context.support.ClassPathXmlApplicationContext` класса `CXFHTTPSClient` для создания контекста клиентского приложения.

Клиентский Spring-файл beans.xml включает в себя, помимо элемента `<jaxws:client>`, определяющего конфигурацию клиента Web-сервиса, элемент `<http-conf:conduit>`, который указывает протокол шифрования и параметры хранилища ключей и доверительного хранилища ключей клиента.

Apache Camel, JMS и Apache ActiveMQ

Проект Apache Camel

Проект Apache Camel (**http://camel.apache.org/**) представляет открытую Java-платформу интеграции приложений, основанную на шаблонах Enterprise Integration Patterns (**http://www.enterpriseintegrationpatterns.com/toc.html**) и призванную упростить задачу создания комплексных систем.

Платформа Apache Camel обеспечивает использование шаблонов Enterprise Integration Patterns (EIP) для реализации правил маршрутизации и обработки сообщений, связывающих различные приложения в одну систему. Фактически платформа Apache Camel представляет собой движок маршрутизации, дающий возможность разработчику реализовывать правила, по которым сообщение проходит путь от отправителя до конечного адресата, работающие вне зависимости от формата сообщения и транспортного протокола. Обеспечивают такую универсальность по отношению к типам данных и протоколам компоненты платформы Apache Camel.

Примечание

Шаблоны Enterprise Integration Patterns — это шаблоны проектирования интеграции приложений с помощью системы сообщений, используемые архитектурой Enterprise Application Integration (EAI) интеграции приложений и инфраструктурой Message-oriented middleware (MOM) обмена сообщениями. Реализация архитектуры EAI может быть создана на основе

инфраструктуры MOM. Примером реализации архитектуры EAI может служить платформа Apache Camel, а примером реализации инфраструктуры MOM — различные реализации JMS (Java Message Service), например система сообщений Apache ActiveMQ. Платформа Apache Camel содержит в качестве одного из компонентов компонент JMS Component, позволяющий отправлять и получать сообщения, используя JMS Queue или Topic.

Платформа Apache Camel может работать на платформе Java SE или в любом Spring-контейнере, может быть развернута поверх сервера приложений, кроме того, платформа Apache Camel интегрирована с такими платформами, как Apache ServiceMix, Apache ActiveMQ, Apache MINA и Apache CXF.

Платформа Apache Camel позволяет реализовывать шаблоны EIP на языке Spring XML или Java Domain Specific Language (DSL), который может расширяться за счет включения выражений, созданных с использованием большого набора языков, таких как JXPath, Groovy, PHP, Scala DSL и др. С Apache Camel конкретный EIP-шаблон просто транслируется в Camel-элемент. Кроме того, маршрутизацию и обработку можно реализовывать программным способом, используя Camel API и Camel-аннотации.

Любое Camel-приложение использует контекст `org.apache.camel.CamelContext`, представляющий набор правил маршрутизации передачи данных и обеспечивающий среду выполнения Camel. Метод `void addRoutes(RoutesBuilder builder)` интерфейса `CamelContext` позволяет создать определенный набор правил с помощью класса `org.apache.camel.builder.RouteBuilder`, переопределяя его метод `abstract void configure()`, который автоматически вызывается при запуске контекста CamelContext методом `start()`. При этом объект `CamelContext` создается с помощью класса `org.apache.camel.impl.DefaultCamelContext`, а метод `configure()` включает в себя Java DSL-код реализации EIP-шаблона.

Java DSL-код содержит реализацию EIP-шаблона, которая имеет общий вид маршрута:

```
from("[Component]").[блок обработки сообщения].to("[Component]");
```

Каждая реализация EIP-шаблона состоит из одного или нескольких маршрутов. *Маршрут* — это интеграционный путь передачи данных между двумя или более конечными точками, идентифицируемыми URL-адресами.

Перечень EIP-шаблонов и их синтаксис можно посмотреть по адресу **http://camel.apache.org/camel-dsl.html**.

Camel-компоненты реализуют интерфейс `org.apache.camel.Component` и обеспечивают конкретный протокол и формат передаваемых данных, представляя собой фабрики для экземпляров конечной точки. Дистрибутив платформы Apache Camel содержит более 100 готовых компонентов (**http://camel.apache.org/components.html**), которые имеют общий синтаксис в виде URI-идентификатора конечной точки.

Каждый Camel-элемент `from().to()`, представляющий EIP-шаблон, является классом Camel API, реализующим интерфейс `org.apache.camel.Processor` обработки сообщений. Поэтому в любой маршрут можно встроить объект `Processor` с его переопределенным методом `void process(Exchange exchange)`, используя синтаксис:

```
from().process(new Processor(){public void process(Exchange exchange)
throws Exception {...}}).to();
```

Интерфейс `org.apache.camel.Exchange` дает возможность доступа к сообщениям запроса, ответа и ошибки через интерфейс `org.apache.camel.Message`, обеспечивающий доступ к идентификатору, заголовкам и телу сообщения.

Среду выполнения CamelContext также можно конфигурировать с помощью Spring-файла:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://camel.apache.org/schema/spring
       http://camel.apache.org/schema/spring/camel-spring.xsd">
  [bean-определения конечных точек]
  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri=". . ."/>
      <process ref=". . ."/>
      <to uri=". . ."/>
    </route>
  </camelContext>
</beans>
```

Spring-элементы, используемые для реализации EIP-шаблонов, показаны в табл. 5.1.

Таким образом, для конфигурирования правил маршрутизации и обработки, добавляемых в контекст CamelContext для реализации EIP-шаблонов, может использоваться как основанный на Java язык DSL, так и язык XML.

При добавлении правил маршрутизации в контекст CamelContext, он становится контейнером для коллекции экземпляров Component — фабрик конечных точек.

Таблица 5.1. *Элементы Camel Spring*

Spring-элемент	**Описание**
`<camelContext>`	Автоматически запускает объект `org.apache.camel.spring SpringCamelContext`, расширяющий `DefaultCamelContext`
`<routeBuilder ref="..."/>`	Ссылка на Bean-определение класса `RouteBuilder`
`<route>`	Определяет маршрут
`<from uri="..."/>`	Определяет Camel-компонент `Component` начальной конечной точки
`<to uri="..."/>`	Определяет Camel-компонент `Component` финальной конечной точки
`<process ref="..."/>`	Ссылка на Bean-определение класса `Processor`

Таблица 5.1 *(продолжение)*

Spring-элемент	Описание
`<packageScan>`	Загружает все RouteBuilder-классы из указанного элементом `<package>` пакета. Для фильтрации могут использоваться элементы `<excludes>` и `<includes>`
`<contextScan/>`	Загружает все RouteBuilder-классы, маркированные аннотацией `@org.springframework.stereotype.Component`
`<endpoint id="..." uri="..."/>`	Определяет конечную точку
`<import resource="[...].xml"/>`	Загружает внешний XML-файл, содержащий определения маршрутов с корневым элементом `<routeContext>`
`<routeContextRef ref="..."/>`	Ссылка на элемент `<routeContext>`
`<choice>`, `<when>`, `<otherwise>`, `<stop/>`	Определяют условия маршрутизации
`<pipeline>`	Блок последовательного прохождения сообщения
`<filter>`	Определяет условия фильтрации сообщений
`<multicast>`	Блок параллельного прохождения сообщения, отправляет копии сообщения нескольким адресатам
`<setHeader>` и `<setBody>`	Устанавливают заголовок и тело сообщения
`<recipientList>`	Фильтрует получателей сообщения по его содержимому
`<wireTap>`	Посылает копию сообщения другому адресату без влияния на последующую маршрутизацию исходного сообщения
`<bean ref="..."/>`	Ссылка на bean-определение класса, обрабатывающего сообщение
`<transform>`	Указывает метод класса, преобразующий сообщение, с помощью элемента `<method bean="..." method="..."/>` или содержит выражение преобразования в элементе `<simple>`
`<aggregate>`	Собирает вместе и комбинирует несколько сообщений в одно сообщение
`<enrich>`	Дополняет текущее сообщение данными из другого источника
`<dataFormats>`	Определяет формат данных
`<marshal>` и `<unmarshal>`	Осуществляет маршализацию-демаршализацию данных
`<split>`	Разделяет одно сообщение на несколько сообщений
`<routingSlip>`	Определяет часть сообщения, содержащую правила маршрутизации для дальнейшей маршрутизации
`<dynamicRouter>`	Указывает метод класса, который определяет динамическую маршрутизацию
`<loadBalance>`	Распределяет нагрузку маршрутизации согласно указанной стратегии
`<resequence>`	Изменяет порядок сообщений
`<throttle>`	Предохраняет конечную точку от перегрузки
`<sample>`	Предохраняет конечную точку от перегрузки, используя период остановки обмена при превышении порога

Таблица 5.1 *(окончание)*

Spring-элемент	Описание
`<inOut>` и `<inOnly>`	Определяют MEP-шаблоны
`<loop>`	Определяет цикл обработки сообщения
`<log>`	Обеспечивает регистрацию сообщения
`<idempotentConsumer>`	Обеспечивает фильтрацию дублированных сообщений
`<validate>`	Определяет проверку содержимого сообщения
`<sort>`	Сортирует содержимое сообщения
`<convertBodyTo type="..."/>`	Конвертирует тело сообщения в указанный тип
`<transacted/>`	Устанавливает транзакцию для маршрута
`<threads>`	Определяет пул потоков
`<delay>`	Указывает задержку выполнения
`<errorHandler>`	Определяет обработку ошибок
`<onException>`	Определяет обработку исключений

Совместное использование платформ Apache CXF и Apache Camel может быть двух видов. Во-первых, CXF Web-сервисы могут использовать специальный вид транспортного протокола Camel transport for CXF для обмена сообщениями с клиентом. Во-вторых, для реализации обмена сообщениями с Web-сервисом платформа Apache Camel предоставляет такие компоненты, как CXF (**http://camel.apache.org/cxf.html**), CXF Bean (**http://camel.apache.org/cxf-bean-component.html**) и CXFRS (**http://camel.apache.org/cxfrs.html**). Оба вида интеграции обеспечивают применение богатых возможностей платформы Apache Camel по созданию правил маршрутизации и обработки сообщений.

Протокол Camel transport for CXF (идентификатор протокола **http://cxf.apache.org/transports/camel**) представляет собой реализацию интерфейса CXF Transport API на основе библиотеки Camel Core и позволяет присоединить CXF Web-сервис к Camel-конечной точке. При этом имя транспортного протокола будет указываться не как **http://...**, а как **camel://direct:...**.

Для присоединения CXF Web-сервиса к Camel-конечной точке с адресом вида **camel://direct:...** необходимо включить объект `org.apache.camel.component.cxf.transport.CamelTransportFactory` в конфигурацию объекта Bus приложения. Сделать это можно как с помощью конфигурационного Spring-файла, так и используя CXF API.

На компакт-диске

Примеры использования протокола Camel transport for CXF расположены в папке Примеры\Глава5\CamelTransport компакт-диска.

Проекты примеров созданы в среде Eclipse в перспективе Java. В данных проектах как Web-сервис, так и его клиент разворачиваются на платформе Java SE в среде

выполнения CXF с присоединенными в путь приложения JAR-файлами camel-core.jar, camel-cxf.jar, camel-spring.jar и commons-management.jar платформы Apache Camel (**http://camel.apache.org/download.html**).

Для подключения платформы Apache CXF необходимо в меню **Properties | Project Facets** проекта выбрать **Dynamic Web Module** и **CXF 2.x Web Services**.

Для подключения платформы Apache Camel необходимо в меню **Properties | Java Build Path** проекта добавить нужные JAR-файлы, нажав кнопку **Add External JARs**.

Проект CamelTransport_1 содержит три пакета: `cxfservice`, `cxfclient` и `main`.

Пакет `cxfservice` включает в себя следующие классы и файлы:

- JAX-WS SEI-интерфейс `CXFService`, объявляющий метод `public String getHello(String name)`;
- JAX-WS SEI-класс `CXFServiceImp`, реализующий SEI-интерфейс `CXFService`;
- конфигурационный Spring-файл beans.xml, объявляющий Bean-компонент класса `CamelTransportFactory`. При этом Bean-компонент связан с заявленным контекстом CamelContext;
- класс `CXFCamelTransportServer`, создающий в своем конструкторе объект Bus приложения на основе Spring-файла beans.xml и в своем методе `start()` публикующий конечную точку Web-сервиса CXFServiceImp по адресу **camel://direct:CXFCamelTransportService**.

Пакет `cxfclient` содержит класс `CXFCamelTransportClient`, который в своем конструкторе создает Proxy-объект Web-сервиса с указанием адреса конечной точки **camel://direct:CXFCamelTransportService** и в методе `invoke()` вызывает метод Web-сервиса `getHello`.

Пакет `main` содержит класс `Main` с методом `main()` — точкой входа в приложение, который создает объект `CXFCamelTransportServer` и вызывает его метод `start()`, создает объект `CXFCamelTransportClient` и вызывает его метод `invoke()` и выводит результат вызова Web-сервиса в консоль.

Запустив приложение, можно увидеть, что обмен между клиентом и Web-сервисом по протоколу Camel transport for CXF производится стандартными SOAP-сообщениями.

В проекте CamelTransport_2 контекст CamelContext, объявленный в Spring-файле beans.xml, содержит маршрут от конечной точки по адресу **camel:// direct:CXFCamelTransportServiceProxy**, который устанавливает для Proxy-объекта клиент Web-сервиса, до конечной точки по адресу **camel:// direct:CXFCamelTransportService**, по которому публикуется Web-сервис, с выводом входящих и исходящих сообщений в консоль.

В проекте CamelTransport_3 клиент Web-сервиса создается с помощью конфигурационного Spring-файла, содержащего объявление `<jaxws:client>`, используемое при создании контекста приложения ClassPathXmlApplicationContext, на основе которого создается Proxy-объект Web-сервиса. Для публикации Web-сервиса также созда-

ется объект `ClassPathXmlApplicationContext`, используя Spring-файл Web-сервиса. Spring-файл Web-сервиса объявляет конечную точку с помощью элемента `<jaxws:endpoint>`.

В проекте CamelTransport_4 для подключения протокола Camel transport for CXF используется не Bean-компонент класса `CamelTransportFactory`, а элемент `<camel:destination>` (пространство имен `xmlns:camel="http://cxf.apache.org/transports/camel"`).

Элемент `<camel:destination>` используется для конфигурации Camel-транспорта на стороне сервера и имеет единственный атрибут `name`, определяющий WSDL-элемент `<port>` конечной точки слушателя на стороне сервера. Элемент `<camel:destination>` также имеет дочерние элементы `<camel-spring:camelContext>` и `<camel:camelContextRef>`, определяющие или ссылающиеся на контекст CamelContext слушателя.

На стороне клиента для конфигурации Camel-транспорта может использоваться элемент `<camel:conduit>`, также имеющий атрибут `name` и дочерние элементы `<camel-spring:camelContext>` и `<camel:camelContextRef>`.

На компакт-диске

Пример использования CXF Camel-компонента (**http://camel.apache.org/cxf.html**) находится в папке Примеры\Глава5 компакт-диска.

Проект CXFCamelComponent примера содержит два пакета: `cxfservice` и `cxfclient`.

Пакет `cxfservice` включает в себя следующие классы и файлы:

- JAX-WS SEI-интерфейс `CXFService`, объявляющий метод `public void Hello(String name)` запроса Web-сервиса в одну сторону;
- конфигурационный Spring-файл beans.xml, объявляющий Camel-компонент CXF с помощью элемента `<cxfEndpoint>` (рис. 5.6). Объявленный Camel-компонент CXF используется в маршруте контекста CamelContext, который отправляет сообщения от конечной точки Camel-компонента CXF Bean-компоненту ProcessBean с выводом входящих и исходящих сообщений в консоль. Элемент `<cxfEndpoint>` является фабрикой конечных точек и при запуске маршрута обеспечивает регистрацию конечной точки Web-сервиса по указанному адресу. При этом конечная точка Web-сервиса фактически представляет собой только почтовый адрес, который принимает сообщения от клиента и пересылает их дальше по маршруту, ничего с ними не делая. Всю обработку сообщений должен взять на себя сам маршрут. По зарегистрированному адресу конечной точки Web-сервиса публикуется WSDL-описание Web-сервиса, которое используется клиентом Web-сервиса для создания Proxy-объекта. Однако такой зарегистрированный адрес конечной точки Web-сервиса не дает возможности вызывать какие-либо методы Web-сервиса. Поэтому для использования Camel-компонента CXF достаточно иметь только SEI-интерфейс без его реализации;
- класс `ProcessBean`, обрабатывающий клиентское сообщение, пересланное Camel-конечной точкой;

- класс `CXFCamelTransportServer`, создающий объект ClassPathXmlApplicationContext на основе Spring-файла и являющийся точкой входа для запуска маршрута.

Пакет `cxfclient` включает в себя следующие классы и файлы:

- конфигурационный Spring-файл beans.xml, объявляющий клиента Web-сервиса с помощью элемента `<jaxws:client>`, который указывает адрес Camel-конечной точки;
- класс `CXFCamelTransportClient`, создающий объект `ClassPathXmlApplicationContext` с использованием Spring-файла и на его основе Proxy-объект Web-сервиса с вызовом его метода, что инициирует отправку сообщения Camel-конечной точке и далее Bean-компоненту, его обрабатывающему.

После запуска последовательно классов `CXFCamelTransportServer` и `CXFCamelTransportClient` как Java-приложений, в консоли среды Eclipse можно увидеть входящее и исходящее сообщения Bean-компонента.

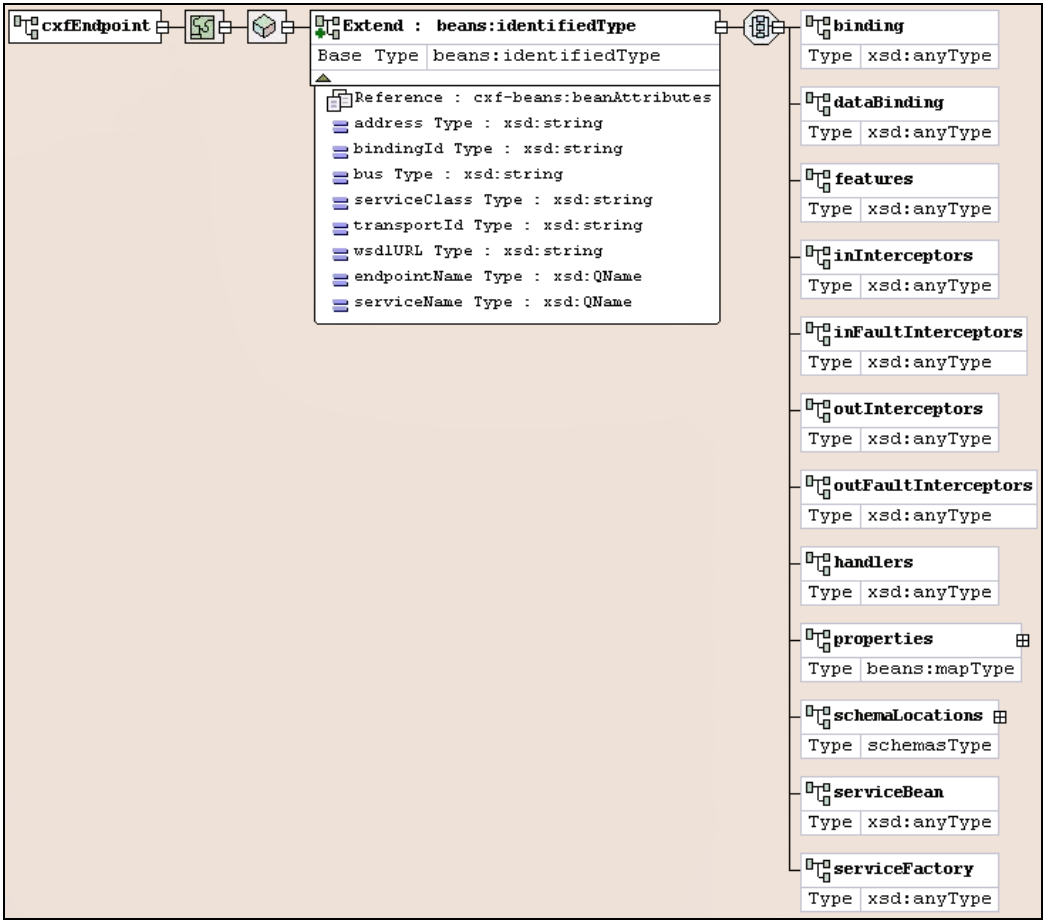


Рис. 5.6. Общая схема элемента `<cxfEndpoint>`

Проект Apache ActiveMQ

Проект Apache ActiveMQ (**http://activemq.apache.org/**) представляет открытую платформу, обеспечивающую сервис сообщений и использование шаблонов Enterprise Integration Patterns с помощью библиотеки Apache Camel. Фактически Apache ActiveMQ — это высокопроизводительный брокер сообщений, обеспечивающий связь между приложениями на основе спецификации JMS (Java Message Service), расширяя ее дополнительными возможностями.

Apache ActiveMQ является поставщиком JMS Provider, реализуя спецификацию JMS 1.1 и обеспечивая JMS-интерфейсы для доступа к системе сообщений, и поддерживает большой набор транспортных протоколов доставки сообщений — HTTP, SSL, TCP, UDP и др. Система сообщений Apache ActiveMQ может быть развернута на платформе Java SE, а также в различных контейнерах, таких как Tomcat, Geronimo, Jetty и др., и предоставляет клиентский интерфейс, помимо Java, для таких платформ, как .NET, Perl, PHP, Ruby и др.

Система сообщений Apache ActiveMQ является системой Message-oriented middleware (MOM) и выступает в качестве посредника между отправителями сообщений и получателями сообщений, позволяя создавать слабо связанные сложные системы программных компонентов. Основное отличие реализации технологии MOM от реализации технологии Web-сервисов, которая также дает возможность создавать сложные системы приложений, заключается в том, что в MOM-системе отправитель и получатель сообщений могут ничего не знать друг о друге, т. к. между ними существует брокер, а между клиентом Web-сервиса и самим Web-сервисом всегда должен быть установлен прямой и обратный транспортные каналы для отправки запроса и получения ответа. Поэтому зачастую системы, включающие в себя Web-сервисы, используют также и технологию MOM.

Для скачивания система сообщений Apache ActiveMQ доступна по адресу **http://activemq.apache.org/download.html**.

После инсталляции запуск ActiveMQ-брокера, отвечающего за прием и отправку сообщений, как отдельного приложения платформы Java SE, осуществляется с помощью сценария activemq.bat папки bin каталога инсталлированной системы Apache ActiveMQ. Открывающуюся при этом консоль закрывать не надо, т. к. это приведет к остановке ActiveMQ-брокера. Завершить работу ActiveMQ-брокера можно так же, нажав комбинацию клавиш <Ctrl>+<C> в консоли. Для мониторинга запущенного ActiveMQ-брокера существует Web-консоль по адресу **http://localhost:8161/admin**.

Для использования ActiveMQ на стороне клиента необходимо в путь приложения добавить архив activemq-all.jar с необходимыми библиотеками, расположенный в каталоге дистрибутива ActiveMQ.

При запуске ActiveMQ-брокера по умолчанию загружается Spring-файл activemq.xml папки conf каталога ActiveMQ, определяющий конфигурацию ActiveMQ-брокера.

Элементы Spring-файла activemq.xml, устанавливающие конфигурацию ActiveMQ-брокера, принадлежат пространству имен `http://activemq.apache.org/schema/core`,

схему которого можно посмотреть в приложении "XML-схема конфигурации ActiveMQ" справки "Приложения" и PDF-файле в каталоге Приложения компакт-диска.

В частности, для установки транспортного протокола и адреса, по которому брокер будет после запуска принимать и слушать запросы от клиента, предназначен элемент `<transportConnectors>` и его дочерние элементы `<transportConnector>` с основными атрибутами `name` (имя протокола) и `uri` (адрес брокера для запросов). Так, для установки протокола TCP используется URI-адрес вида **tcp://hostname:port?key=*value***, а для определения протокола HTTP — URI-адрес вида **http://host:*port***.

Для вывода входящих и исходящих сообщений брокера можно включить в элемент `<plugins>` Spring-файла activemq.xml элемент `<loggingBrokerPlugin>`, устанавливающий logging-обработчик брокера.

Для ограничения доступа к ActiveMQ-брокеру в конфигурационный файл можно включить элемент `<plugins>` с дочерним элементом `<simpleAuthenticationPlugin>`, содержащим элемент `<users>` с дочерними элементами `<authenticationUser>`, устанавливающими имена авторизированных пользователей (атрибут `username`) и их пароли (атрибут `password`) (см. конфигурационный файл activemq-security.xml папки conf каталога ActiveMQ). Для определения прав доступа к объектам Destination в элемент `<plugins>` конфигурационного файла включается элемент `authorizationPlugin/map/authorizationMap/authorizationEntries` с дочерними элементами `<authorizationEntry queue="..." read="..." write="..." admin="..." />` или `<authorizationEntry topic="..." read="..." write="..." admin="..." />`, определяющими права для групп пользователей брокера.

Таким образом, для создания системы клиент-ActiveMQ-брокер достаточно запустить ActiveMQ-брокер сценарием activemq.bat, а на стороне клиента включить следующий код:

```
import org.apache.activemq.ActiveMQConnectionFactory;
. . .
ActiveMQConnectionFactory factory =
  new ActiveMQConnectionFactory("tcp://0.0.0.0:61616");
connection = factory.createConnection("[имя пользователя]","[пароль]");
session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
topic = session.createTopic("...");
publisher = session.createProducer(topic);
//или
session.createConsumer(topic).setMessageListener(this);
public void onMessage(Message message) {
  . . .
  connection.close();
}
//
connection.start();
```

```
publisher.send(session.createTextMessage("..."));
connection.stop();
connection.close();
```

Помимо запуска ActiveMQ-брокера как отдельного приложения, его можно встроить непосредственно в Java-приложение, которое будет управлять жизненным циклом ActiveMQ-брокера.

Для создания встроенного Embedded ActiveMQ-брокера достаточно создать Java-приложение со следующим кодом:

```
import org.apache.activemq.broker.BrokerService;
. . .
public static void main(String[] args) throws Exception {
        BrokerService broker = new BrokerService();
        broker.addConnector("tcp://localhost:61616");
        broker.start();
```

Методы класса `BrokerService` позволяют установить конфигурацию ActiveMQ-брокера программным способом. Для определения конфигурации ActiveMQ-брокера с помощью конфигурационного файла можно воспользоваться методом `public static BrokerService createBroker(String` *`brokerURI`*`)` класса `org.apache.activemq.broker.BrokerFactory`, где *`brokerURI`* — URI-адрес конфигурационного файла.

Если в Java-код включить:

```
broker.setBrokerName("[имя Embedded ActiveMQ-брокера]");
```

тогда клиент, работающий на той же JVM-машине, что и Embedded ActiveMQ-брокер, может вызывать его, используя локальный транспорт **vm://[*имя Embedded ActiveMQ-брокера*]**.

Embedded ActiveMQ-брокера также можно создать с помощью класса `ActiveMQConnectionFactory` на основе конфигурационного файла:

```
ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory
  ("vm://localhost?brokerConfig=xbean:activemq.xml ");
```

Использование только JMS API системы сообщений ActiveMQ позволяет создать Web-сервис и его клиента на основе низкоуровневого программного интерфейса для формирования и чтения сообщений. Пример такого Web-сервиса уже рассматривался в *разд. "JAXM и SAAJ" главы 3*.

Создать систему, состоящую из CXF JAX-WS Web-сервиса, JMS-брокера (ActiveMQ-брокера) и клиента Web-сервиса, можно несколькими способами.

> ***НА КОМПАКТ-ДИСКЕ***
>
> Примеры использования ActiveMQ-брокера расположены в папке Примеры\Глава5\JMSService компакт-диска.

Проекты примеров созданы в среде Eclipse в перспективе Java. В данных проектах как Web-сервис, так и его клиент разворачиваются на платформе Java SE в среде выполнения CXF с присоединенным в путь приложения JAR-файлом activemq-all.jar платформы Apache ActiveMQ.

Первый способ основан на применении транспортного плагина JMS-транспорта платформы CXF, обеспечивающего связь с JMS-брокером.

> ***На компакт-диске***
>
> Пример такого Web-сервиса и его клиента, использующих ActiveMQ-брокера через JMS-транспорт, представлен проектом CXFJMSTransport_1, расположенным в папке Примеры\Глава5\JMSService компакт-диска.

Проект CXFJMSTransport_1 содержит три пакета: `cxfservice`, `cxfclient` и `main` и подключенные библиотеки платформы Apache CXF и Apache ActiveMQ.

Пакет `cxfservice` содержит следующие классы и файлы:

- SEI-интерфейс `CXFService`;
- SEI-класс `CXFServiceImp`;
- конфигурационный Spring-файл beans.xml, определяющий с помощью элемента `<jaxws:endpoint>` конечную точку Web-сервиса для публикации по адресу **jms://** и использующий элемент `<jms:destination>` (пространство имен `xmlns:jms="http://cxf.apache.org/transports/jms"`) для конфигурации JMS-транспорта конечной точки Web-сервиса. Элемент `<jms:destination>` указывает поставщика конечной точки с помощью атрибута `name` и конфигурирует JMS-транспорт конечной точки Web-сервиса с помощью дочернего элемента `<jms:address>`, имеющего следующие атрибуты и дочерние элементы:
  - атрибут `destinationStyle`, возможные значения `queue` или `topic`. В данном случае это очередь `queue`;
  - атрибут `jndiConnectionFactoryName` указывает JNDI-имя объекта `org.apache.activemq.ActiveMQConnectionFactory`. В данном случае это имя фабрики по умолчанию ConnectionFactory;
  - атрибут `jndiDestinationName` указывает JNDI-имя объекта Destination для запроса. В данном случае это имя создаваемой очереди динамического контекста dynamicQueues;
  - атрибут `jndiReplyDestinationName` указывает JNDI-имя объекта Destination для ответа. В данном случае это имя создаваемой очереди динамического контекста dynamicQueues;
  - атрибут `connectionUserName` — имя пользователя брокера;
  - атрибут `connectionPassword` — пароль пользователя брокера;
  - элемент `JMSNamingProperty` определяет JNDI-свойства с помощью атрибутов `name` и `value`. В данном случае элементы JMSNamingProperty указывают объект `org.apache.activemq.jndi.ActiveMQInitialContextFactory`, создающий объекты `ActiveMQConnectionFactory` и `Destination`, и адрес ActiveMQ-брокера;
- класс `CXFJMSTransportServer`, создающий в конструкторе объект `ClassPathXmlApplicationContext` на основе Spring-файла beans.xml.

Пакет `cxfclient` включает в себя следующие классы и файлы:

- конфигурационный Spring-файл beans.xml, объявляющий клиента Web-сервиса с помощью элемента `<jaxws:client>`, который указывает адрес **jms://** конечной точки Web-сервиса, и использующий элемент `<jms:conduit>` (пространство имен `xmlns:jms="http://cxf.apache.org/transports/jms"`) для конфигурации JMS-транспорта Proxy-объекта Web-сервиса. Элемент `<jms:conduit>` указывает потребителя конечной точки с помощью атрибута `name` и конфигурирует JMS-транспорт потребителя конечной точки Web-сервиса посредством дочернего элемента `<jms:address>`;
- класс `CXFJMSTransportClient`, создающий в конструкторе объект `ClassPathXmlApplicationContext` с использованием Spring-файла beans.xml и на его основе Proxy-объект Web-сервиса, а также объявляющий метод, который отвечает за вызов метода Proxy-объекта Web-сервиса.

Пакет `main` содержит класс `Main` с методом `main()` — точкой входа в приложение, который создает объект `CXFJMSTransportServer` и объект `CXFJMSTransportClient` с вызовом его метода `invoke()` и выводом результата вызова Web-сервиса в консоль.

После запуска ActiveMQ-брокера сценарием activemq.bat папки bin каталога установленной системы Apache ActiveMQ и запуска класса `Main` как Java-приложения, в консоли среды Eclipse можно увидеть исходящее и входящее сообщения и строку, возвращаемую методом Web-сервиса.

Второй способ применения ActiveMQ-брокера также основан на использовании JMS-транспорта, только с помощью опции **JMSConfigFeature** платформы CXF, обеспечивающей связь с JMS-брокером.

> ***На компакт-диске***
>
> Пример такого Web-сервиса и его клиента, взаимодействующих друг с другом через ActiveMQ-брокера, представлен проектом CXFJMSTransport_2, расположенным в папке Примеры\Глава5\JMSService компакт-диска.

Проект CXFJMSTransport_2 содержит также три пакета: `cxfservice`, `cxfclient` и `main` и подключенные библиотеки платформы Apache CXF и Apache ActiveMQ.

В отличие от предыдущего проекта, в проекте CXFJMSTransport_2 в конфигурационных Spring-файлах Web-сервиса и клиента Web-сервиса для установки конфигурации JMS-транспорта используются не элементы `<jms:destination>` и `<jms:conduit>`, а дочерний элемент `<jaxws:features>` элементов `<jaxws:endpoint>` и `<jaxws:client>`, определяющий Bean-компонент:

```
<bean class="org.apache.cxf.transport.jms.JMSConfigFeature"
      p:jmsConfig-ref="jmsConfig"/>
```

где атрибут `jmsConfig-ref` ссылается на Bean-компонент конфигурации `org.apache.cxf.transport.jms.JMSConfiguration`:

```
<bean id="jmsConfig" class="org.apache.cxf.transport.jms.JMSConfiguration"
      p:connectionFactory-ref="cxfConnectionFactory"
      p:targetDestination="cxfQueue" />
```

Конфигурация JMSConfiguration устанавливается с помощью следующих атрибутов:

- `connectionFactory-ref` — ссылка на Bean-компонент, определяющий JMS-объект ConnectionFactory;
- `wrapInSingleConnectionFactory` — если `true` (по умолчанию), тогда используется Spring-фабрика `org.springframework.jms.connection.SingleConnectionFactory`;
- `reconnectOnException` — если `false` (по умолчанию), тогда при возникновении ошибки не будет делаться попытка повторного соединения;
- `targetDestination` — JNDI-имя создаваемого объекта Destination для запроса;
- `replyDestination` — JNDI-имя создаваемого объекта Destination для ответа;
- `destinationResolver` — ссылка на объект `org.springframework.jms.support.destination.DestinationResolver` (по умолчанию `DynamicDestinationResolver`);
- `transactionManager` — ссылка на объект `org.springframework.transaction.support.AbstractPlatformTransactionManager`;
- `taskExecutor` — ссылка на объект `org.springframework.core.task.TaskExecutor` (по умолчанию `SimpleAsyncTaskExecutor`);
- `useJms11` — если `true`, тогда поддерживается спецификация JMS 1.1 (по умолчанию `false`);
- `messageIdEnabled` — если `true` (по умолчанию), тогда производится идентификация сообщений;
- `messageTimestampEnabled` — если `true` (по умолчанию), тогда запросы имеют время действия;
- `cacheLevel` — определяет уровень кэширования (по умолчанию `-1`);
- `pubSubNoLocal` — если `true`, тогда не допускается использование локального объекта `Topic` (по умолчанию `false`);
- `receiveTimeout` — указывает время ожидания ответа (по умолчанию `0`);
- `explicitQosEnabled` — если `true`, тогда QoS-параметры устанавливаются для каждого сообщения (по умолчанию `false`);
- `deliveryMode` — режим доставки сообщений (по умолчанию `NON_PERSISTENT=1`);
- `priority` — приоритет сообщений (по умолчанию `4`);
- `timeToLive` — время, после которого сообщение отбрасывается (по умолчанию `0`);
- `sessionTransacted` — если `true`, тогда используются JMS-транзакции (по умолчанию `false`);
- `concurrentConsumers` — минимальное число параллельных клиентов (по умолчанию `1`);
- `maxConcurrentConsumers` — максимальное число параллельных клиентов (по умолчанию `1`);

- `maxConcurrentTasks` — максимальное число потоков, обрабатывающих запросы (по умолчанию `10`);
- `messageSelector` — фильтр сообщений;
- `subscriptionDurable` — если `true`, тогда брокер хранит сообщения, когда клиент неактивен (по умолчанию `false`);
- `durableSubscriptionName` — имя сохраняемой подписки;
- `messageType` — тип сообщения: `text` (по умолчанию), `binary` и `byte`;
- `pubSubDomain` — если `false` (по умолчанию), тогда объект SingleConnectionFactory использует Queue-очереди;
- `jmsProviderTibcoEms` — если `true`, тогда JMS-брокер является сервисом TIBCO Enterprise Message Service;
- `useMessageIDAsCorrelationID` — если `true`, тогда идентификатор сообщений используется для их корреляции между собой (по умолчанию `false` — используется специальный корреляционный идентификатор).

Третий способ применения ActiveMQ-брокера основан на реализации платформой CXF спецификации SOAP over Java Message Service. В данном случае JMS-транспорт подключается с помощью определения JMS URI-адреса в качестве адреса конечной точки Web-сервиса и установки идентификатора транспорта **http://www.w3.org/2010/soapjms/** (например, с помощью метода `setTransportId(org.apache.cxf.transport.jms.spec.JMSSpecConstants.SOAP_JMS_SPECIFICATION_TRANSPORTID)` фабрик `org.apache.cxf.jaxws.JaxWsProxyFactoryBean` и `org.apache.cxf.jaxws.JaxWsServerFactoryBean`).

Адрес JMS URI имеет следующий синтаксис:

- `jms:jndi:[`*имя Destination*`]?[`*свойства key=value*`]`;
- `jms:queue:[`*имя Queue*`]?[`*свойства key=value*`]`;
- `jms:topic:[`*имя Topic*`]?[`свойства *key=value*`]`,

где свойства могут быть следующими:

- `deliveryMode` — режим доставки сообщений, возможные значения: `PERSISTENT` (по умолчанию), `NON_PERSISTENT`;
- `jndiConnectionFactoryName` — имя фабрики для создания соединения (по умолчанию JNDI-имя фабрики ActiveMQ-брокера — ConnectionFactory);
- `jndiInitialContextFactory` — имя объекта `InitialContextFactory` (для ActiveMQ-брокера — `org.apache.activemq.jndi.ActiveMQInitialContextFactory`);
- `jndiURL` — адрес брокера;
- `replyToName` — JNDI-имя объекта `Destination` для ответа;
- `priority` — приоритет сообщений (по умолчанию `4`);
- `timeToLive` — время, после которого сообщения отбрасываются (по умолчанию `0`).

На компакт-диске

Проект CXFJMSTransport_3 примера использования SOAP over JMS находится в папке Примеры\Глава5\JMSService компакт-диска.

Для запуска примера необходимо сначала запустить ActiveMQ-брокер сценарием activemq.bat папки bin каталога установленной системы Apache ActiveMQ.

Проект CXFJMSTransport_3 содержит три пакета: `cxfservice`, `cxfclient` и `main` и подключенные библиотеки платформы Apache CXF и Apache ActiveMQ.

Класс `CXFJMSTransportServer` пакета `cxfservice` публикует конечную точку Web-сервиса в конструкторе с помощью следующего кода:

```
String address = "jms:jndi:dynamicQueues/cxfQueue"
          + "?jndiInitialContextFactory"
          + "=org.apache.activemq.jndi.ActiveMQInitialContextFactory"
          + "&jndiConnectionFactoryName=ConnectionFactory&jndiURL=
                                                    tcp://0.0.0.0:61616";
      CXFServiceImp implementor = new CXFServiceImp();
      JaxWsServerFactoryBean svrFactory = new JaxWsServerFactoryBean();
      svrFactory.setServiceClass(CXFServiceImp.class);
      svrFactory.setAddress(address);
      svrFactory.setTransportId(JMSSpecConstants.SOAP_JMS_SPECIFICATION_
                                                          TRANSPORTID);
svrFactory.setServiceBean(implementor);
svrFactory.create();
```

А класс `CXFJMSTransportClient` обеспечивает создание Proxy-объекта Web-сервиса в конструкторе с помощью следующего кода:

```
String address = "jms:jndi:dynamicQueues/cxfQueue"
        + "?jndiInitialContextFactory"
        + "=org.apache.activemq.jndi.ActiveMQInitialContextFactory"
        + "&jndiConnectionFactoryName=ConnectionFactory&jndiURL="
        + "tcp://0.0.0.0:61616";
JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setTransportId
 (JMSSpecConstants.SOAP_JMS_SPECIFICATION_TRANSPORTID);
factory.setServiceClass(cxfservice.CXFService.class);
factory.setAddress(address);
factory.getInInterceptors().add(new LoggingInInterceptor());
factory.getOutInterceptors().add(new LoggingOutInterceptor());
client = (cxfservice.CXFService)factory.create();
```

Четвертый способ применения ActiveMQ-брокера основан на использовании Camel-компонента `org.apache.camel.component.jms.JmsComponent` (**http://camel.apache.org/jms.html**) или `org.apache.activemq.camel.component.ActiveMQComponent` (**http://camel.apache.org/activemq.html**).

Использование Camel-компонента дает возможность совместного применения Camel-транспорта, маршрутизации и ActiveMQ-брокера.

На компакт-диске

Пример Web-сервиса и его клиента, взаимодействующих через Camel-компонент ActiveMQ-брокера по Camel-транспорту, можно посмотреть в папке Примеры\Глава5\JMSService как проект CXFJMSCamel_1 (JmsComponent) и проект CXFJMSCamel_2 (ActiveMQComponent).

Использование Camel-транспорта и Camel-компонента требует подключения, помимо библиотек платформы CXF и библиотечного файла системы сообщений ActiveMQ, библиотечных файлов платформы Camel — camel-core.jar, camel-cxf.jar, camel-spring.jar, camel-jms.jar и commons-management.jar.

Перед запуском примеров также необходимо предварительно запустить ActiveMQ-брокер сценарием activemq.bat папки bin каталога установленной системы Apache ActiveMQ.

Проекты CXFJMSCamel_1 и CXFJMSCamel_2, как и предыдущие примеры, содержат пакеты `cxfservice`, `cxfclient` и `main`. Отличие заключается в содержимом конфигурационных Spring-файлах Web-сервиса и клиента Web-сервиса.

Конфигурационный Spring-файл Web-сервиса содержит следующие объявления:

- конечной точки с помощью элемента `<jaxws:endpoint>` по адресу **camel://direct:**;
- Camel-компонента с помощью элементов:

```
<bean id="jms" class="org.apache.camel.component.jms.JmsComponent">
```

или

```
<bean id="activemq"
      class="org.apache.activemq.camel.component.ActiveMQComponent">
  <property name="connectionFactory" ref="cxfConnectionFactory" />
</bean>
<bean id="cxfConnectionFactory"
      class="org.apache.activemq.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp://0.0.0.0:61616" />
</bean>
```

- Camel-транспорта с помощью элемента:

```
<bean class="org.apache.camel.component.cxf.transport.CamelTransportFactory">
  <property name="camelContext" ref="camelContext" />
</bean>
```

- Camel-маршрута:

```
<camelContext id="camelContext"
              xmlns="http://camel.apache.org/schema/spring">
<route>
<from uri="jms://queue:cxfQueue" />
```

или

```
<from uri="activemq://queue:cxfQueue" />
<to uri="direct:CXFJMSCamel" />
</route>
</camelContext>
```

Конфигурационный Spring-файл клиента Web-сервиса содержит следующие объявления:

- Proxy-объекта Web-сервиса с помощью элемента `<jaxws:client>` по адресу **camel://direct:**;
- Camel-компонента *(см. ранее)*;
- Camel-транспорта *(см.ранее)*;
- Camel-маршрута:

```
<camelContext id="camelContext"
              xmlns="http://camel.apache.org/schema/spring">
<route>
<from uri="direct:CXFJMSCamel" />
<to uri="jms://queue:cxfQueue" />
```

или

```
<to uri="activemq://queue:cxfQueue" />
</route>
</camelContext>
```

Конфигурационные Spring-файлы, в данном случае, требуют включения объявлений:

`http://camel.apache.org/schema/spring`
`http://camel.apache.org/schema/spring/camel-spring.xsd`
`http://cxf.apache.org/transports/camel`
`http://cxf.apache.org/transports/camel.xsd`
`<import resource="classpath:META-INF/cxf/cxf-extension-camel.xml"/>` (на стороне Web-сервиса).

Локальный транспорт

Платформа CXF обеспечивает локальный транспорт для более эффективного обмена сообщениями между клиентом и Web-сервисом, работающих на одной и той же JVM-машине. При этом адрес для конечной точки Web-сервиса и его Proxy-объекта устанавливается в виде **local://[*имя конечной точки*]**.

> ***На компакт-диске***
>
> Пример использования локального транспорта находится в папке Примеры\Глава5\ как проект CXFLocalTransport. Данный проект был создан в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFLocalTransport содержит три пакета: `cxfservice`, `cxfclient` и `main` и подключенные библиотеки платформы Apache CXF.

Пакет `cxfservice` включает в себя следующие классы и файлы:

- SEI-интерфейс `CXFService`;
- SEI-класс `CXFServiceImp`;

- конфигурационный Spring-файл beans.xml, определяющий с помощью элемента `<jaxws:endpoint>` конечную точку Web-сервиса для публикации по адресу **local://CXFLocalSevice** и подключающий локальный транспорт с помощью Bean-компонента `<bean class="org.apache.cxf.transport.local.LocalTransportFactory"/>`. Как альтернатива использованию Spring-файла, существует возможность публикации конечной точки методом `Endpoint.publish("local://CXFLocalSevice", new CXFServiceImp());`
- класс `CXFLocalTransportServer`, создающий в конструкторе объект `ClassPathXmlApplicationContext` на основе Spring-файла beans.xml.

Пакет `cxfclient` включает в себя следующие классы и файлы:

- конфигурационный Spring-файл beans.xml, объявляющий клиента Web-сервиса с помощью элемента `<jaxws:client>`, который указывает адрес **local:// CXFLocalSevice** конечной точки Web-сервиса;
- класс `CXFLocalTransportClient`, создающий в конструкторе объект `ClassPathXmlApplicationContext` с использованием Spring-файла beans.xml и на его основе Proxy-объект Web-сервиса, а также объявляющий метод, который отвечает за вызов метода Proxy-объекта Web-сервиса.

Пакет `main` содержит класс `Main` с методом `main()` — точкой входа в приложение, который создает объект `CXFLocalTransportServer` и объект `CXFLocalTransportClient` с вызовом его метода `invoke()` и выводом результата вызова Web-сервиса в консоль.

После запуска класса `Main` как Java-приложения в консоли среды Eclipse можно увидеть исходящее и входящее сообщения и строку, возвращаемую методом Web-сервиса.

Поддержка MTOM

Поддержка технологии SOAP Message Transmission Optimization Mechanism (MTOM) обеспечивает оптимизацию передачи SOAP-сообщений, включающих в себя двоичные данные, закодированные в формате `base64Binary`.

При включении механизма MTOM для SOAP-сообщения создается пустой XOP-пакет (XML-binary Optimized Packaging) и в SOAP-сообщении определяются данные формата `base64Binary` для оптимизации на основании размера данных. После определения данных для оптимизации производится их декодирование из формата `base64Binary` обратно в двоичный формат с удалением данных из SOAP-элемента `Envelope`. Затем формируется для пересылки окончательный XOP-пакет, состоящий из основного SOAP-сообщения и вложений, представляющих собой декодированные при оптимизации данные. В случае если оптимизации подвергается SOAP-сообщение с вложениями, первоначальные вложения просто копируются в XOP-пакет.

Для включения механизма MTOM оптимизации обмена сообщениями между клиентом и Web-сервисом необходимо произвести следующие действия:

1. Необязательно — промаркировать типы данных, предназначенных для преобразования в MTOM-вложение, с помощью аннотации `@XmlMimeType("application/octet-stream")` или WSDL-атрибута `xmime:expectedContentTypes="application/octet-stream"`.
2. Включить механизм MTOM на стороне клиента и Web-сервиса. Включение механизма MTOM среды выполнения CXF возможно двумя способами:
   - программным способом с помощью следующего кода:

     ```
     javax.xml.ws.soap.SOAPBinding binding =
      (javax.xml.ws.soap.SOAPBinding)[объект Endpoint или Proxy].getBinding();
     binding.setMTOMEnabled(true);
     ```

     Или аннотации на стороне Web-сервиса

     ```
     @javax.xml.ws.soap.MTOM(threshold=...)
     ```

     и на стороне клиента

     ```
     [объект javax.xml.ws.Service].getPort(new MTOMFeature(...))
     ```

   - в конфигурационном Spring-файле, используя элемент:

     ```
     <jaxws:properties>
       <entry key="mtom-enabled" value="true"/>
     </jaxws:properties>
     ```

НА КОМПАКТ-ДИСКЕ

Пример использования механизма MTOM оптимизации обмена сообщениями между клиентом и Web-сервисом можно посмотреть в папке Примеры\Глава5 как проект CXFMTOM.

Проект CXFMTOM создан в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFMTOM состоит из трех пакетов: `cxfservice`, `cxfclient` и `main`.

Пакет `cxfservice` содержит следующие классы и файлы:

- SEI-интерфейс `CXFService`, объявляющий метод `public java.awt.Image getImage()`;
- SEI-класс `CXFServiceImp`, переопределяющий метод `getImage()` (в проекте в узле **src** создана папка resources с файлом изображения image.jpg):

```
@Override
public java.awt.Image  getImage() {
  java.net.URL url=this.getClass().getResource("/resources/image.jpg");
  try {
    return (java.awt.Image)javax.imageio.ImageIO.read(url);
  } catch (Exception ex) {
    System.out.println(ex.getMessage());
  }
  return null;    }
```

- конфигурационный Spring-файл beans.xml, определяющий с помощью элемента `<jaxws:endpoint>` конечную точку Web-сервиса для публикации по адресу **http://localhost:9090/CXFServiceImpPort**, включающий механизм MTOM дочерним элементом:

  ```
  <jaxws:properties>
    <entry key="mtom-enabled" value="true"/>
  </jaxws:properties>
  ```

  элемента `<jaxws:endpoint>` и подключающий транспортный протокол SOAP/HTTP с помощью элемента `<import resource="classpath:META-INF/cxf/cxf-extension-http-jetty.xml"/>`;
- класс `CXFMTOMServer`, создающий в конструкторе объект `ClassPathXmlApplicationContext` на основе Spring-файла beans.xml.

Пакет `cxfclient` включает в себя следующие классы и файлы:

- конфигурационный Spring-файл beans.xml, объявляющий клиента Web-сервиса с помощью элемента `<jaxws:client>`, который указывает адрес **http://localhost:9090/CXFServiceImpPort** конечной точки Web-сервиса;
- класс `CXFMTOMClient`, создающий в конструкторе объект `ClassPathXmlApplicationContext` с использованием Spring-файла beans.xml и на его основе Proxy-объект Web-сервиса, а также определяющий метод, который отвечает за вызов метода Proxy-объекта Web-сервиса.

Пакет `main` содержит класс `Main` с методом `main()` — точкой входа в приложение, который создает объект `CXFMTOMServer` и объект `CXFMTOMClient` с вызовом его метода `invoke()`.

После запуска класса `Main` как Java-приложения в консоли среды Eclipse можно увидеть исходящее и входящее сообщения. Видно, что исходящее сообщение Web-сервиса имеет MIME-тип `application/xop+xml` и XOP-пакет, состоящий из основного SOAP-сообщения и вложения, представляющего двоичные данные возвращаемого изображения.

Поддержка спецификаций WS-*

WS-Addressing

Включить поддержку спецификации WS-Addressing для CXF Web-сервиса и его клиента можно несколькими способами.

Включить адресацию сообщений, участвующих в обмене между Web-сервисом и клиентом Web-сервиса, можно программным способом. На стороне Web-сервиса это позволяет сделать метод `getFeatures().add(new org.apache.cxf.ws.addressing.WSAddressingFeature())` класса `org.apache.cxf.jaxws.EndpointImpl`. На стороне клиента включить поддержку WS-Addressing можно, используя метод `getFeatures().add(new WSAddressingFeature())` класса `org.apache.cxf.jaxws.JaxWsProxyFactoryBean`.

С помощью конфигурационного Spring-файла поддержка WS-Addressing включается дочерним элементом `<wsa:addressing xmlns:wsa="http://cxf.apache.org/ws/addressing"/>` элемента `<jaxws:features>` или определением политики конечной точки.

Для определения политики конечной точки на стороне Web-сервиса в Spring-файл необходимо включить следующие элементы:

- `<import resource="classpath:META-INF/cxf/cxf-extension-policy.xml"/>` — подключает конфигурационный файл реализации платформой CXF спецификации WS-Policy. CXF-реализация WS-Policy основана на платформе Apache Neethi с использованием AssertionBuilder API и PolicyInterceptorProvider API;
- `<import resource="classpath:META-INF/cxf/cxf-extension-addr.xml"/>` — подключает конфигурационный файл CXF-реализации утверждения политики `<wsam:Addressing>`;
- `<p:policies>` (пространство имен `xmlns:p="http://cxf.apache.org/policy"`, схема **http://cxf.apache.org/schemas/policy.xsd**) как дочерний элемент элемента `<jaxws:features>` — подключает CXF-реализацию WS-Policy;
- ссылку на политику или политику как дочерний элемент элемента `<p:policies>`:

```
<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
<wsam:Addressing xmlns:wsam="http://www.w3.org/2007/02/addressing/metadata">
<wsp:Policy/>
</wsam:Addressing>
</wsp:Policy>
```

На стороне клиента в Spring-файл необходимо включить элемент `<p:policies>` как дочерний элемент элемента `<jaxws:features>`, содержащий ссылку на политику или политику с утверждением `<wsam:Addressing>`.

> ***На компакт-диске***
>
> Примеры использования спецификации WS-Addressing находятся в папке Примеры\Глава5\WS компакт-диска как проекты CXFAddressing_1 и CXFAddressing_2.

Проекты CXFAddressing_1 и CXFAddressing_2 созданы в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

В проекте CXFAddressing_1 продемонстрирован программный способ подключения WS-Addressing и использование дочернего элемента `<wsa:addressing xmlns:wsa="http://cxf.apache.org/ws/addressing"/>` элемента `<jaxws:features>` конфигурационного Spring-файла.

В проекте CXFAddressing_2 показано определение политики конечной точки элементом `<wsp:Policy xmlns:wsp="http://www.w3.org/ns/ws-policy">`, содержащемся в дочернем элементе `<p:policies>` элемента `<jaxws:features>` конфигурационного Spring-файла.

После запуска класса `Main` проектов в консоли среды Eclipse можно увидеть, что входящее и исходящее сообщения Web-сервиса содержат заголовок `<soap:Header>` с элементами `<Action xmlns="http://www.w3.org/2005/08/addressing">`, `<MessageID xmlns="http://www.w3.org/2005/08/addressing">`,

`<To xmlns="http://www.w3.org/2005/08/addressing">`,
`<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">` или
`<RelatesTo xmlns="http://www.w3.org/2005/08/addressing">`.

WS-ReliableMessaging

Поддержка спецификации WS-ReliableMessaging обеспечивает механизм надежной и гарантированной передачи сообщений между клиентом и Web-сервисом, предотвращающий потерю, дублирование и изменение порядка сообщений. При этом между сторонами, участвующими в таком обмене сообщениями, создается сессия надежного и гарантированного обмена сообщениями с помощью формирования последовательности сообщений Sequence.

Поддержка WS-ReliableMessaging реализуется платформой CXF, так же как и другие CXF-опции, с помощью обработчиков Interceptor пакета `org.apache.cxf.ws.rm`. Поэтому для включения поддержки WS-ReliableMessaging необходимо добавить WS-ReliableMessaging Interceptor-обработчики и WS-Addressing Interceptor-обработчики в цепочки обработчиков сообщений. С помощью конфигурационного Spring-файла это можно сделать, используя опции `<wsa:addressing/>` и `<wsrm-mgr:reliableMessaging/>` или опцию `<p:policies>`, определяющую политику с утверждениями `<wsam:Addressing>` и `<wsrmp:RMAssertion>`.

> ***На компакт-диске***
>
> Примеры использования спецификации WS-ReliableMessaging находятся в папке Примеры\Глава5\WS компакт-диска как проекты CXFRM_1 и CXFRM_2.

Проекты CXFRM_1 и CXFRM_2 созданы в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFRM_1 демонстрирует применение опции `<p:policies>` и содержит на стороне Web-сервиса конфигурационный Spring-файл beans.xml, на основе которого создается контекст среды выполнения Web-сервиса.

Spring-файл Web-сервиса для поддержки WS-ReliableMessaging включает в себя:

- определение пространства имен опции `<p:policies>` — `xmlns:p="http://cxf.apache.org/policy"`;
- адрес схемы пространства имен опции `<p:policies>` — `http://cxf.apache.org/policy http://cxf.apache.org/schemas/policy.xsd`;
- импорт конфигурационных файлов поддержки утверждений политики `<import resource="classpath:META-INF/cxf/cxf-extension-policy.xml"/>`, `<import resource="classpath:META-INF/cxf/cxf-extension-addr.xml"/>`, `<import resource="classpath:META-INF/cxf/cxf-extension-rm.xml"/>`;
- дочерний элемент `<p:policies>` элемента `<jaxws:endpoint>`/`<jaxws:features>` с утверждениями `<wsam:Addressing xmlns:wsam="http://www.w3.org/2007/02/addressing/metadata">` и `<wsrmp:RMAssertion xmlns:wsrmp="http://schemas.xmlsoap.org/ws/2005/02/rm/policy">`.

Spring-файл контекста клиента Web-сервиса проекта CXFRM_1 для поддержки WS-ReliableMessaging включает в себя те же самые элементы, что и Spring-файл Web-сервиса (элемент `<p:policies>` содержится в элементе `<jaxws:client>`/ `<jaxws:features>`).

Проект CXFRM_2 демонстрирует применение опций `<wsa:addressing/>` и `<wsrm-mgr:reliableMessaging/>` и содержит на стороне Web-сервиса конфигурационный Spring-файл beans.xml, на основе которого создается объект Bus Web-сервиса.

Spring-файл Web-сервиса проекта CXFRM_2 для поддержки WS-ReliableMessaging включает в себя:

- дочерние элементы `<wsa:addressing xmlns:wsa="http://cxf.apache.org/ws/addressing"/>` и `<wsrm-mgr:reliableMessaging xmlns:wsrm-mgr= "http://cxf.apache.org/ws/rm/manager"/>` элемента `<cxf:bus>`;
- адрес схемы пространства имен элемента `<wsrm-mgr:reliableMessaging>` — `http://cxf.apache.org/ws/rm/manager` `http://cxf.apache.org/schemas/configuration/wsrm-manager.xsd`.

Spring-файл конфигурации объекта Bus клиента Web-сервиса проекта CXFRM_2 для поддержки WS-ReliableMessaging включает в себя те же самые элементы, что и Spring-файл Web-сервиса.

Элемент `<wsrm-mgr:reliableMessaging>` (рис. 5.7) может включать в себя следующие элементы:

- `<RMAssertion>` (пространство имен `http://schemas.xmlsoap.org/ws/2005/02/rm/policy`) определяет политику использования WS-ReliableMessaging с помощью дочерних элементов:
  - `<InactivityTimeout>` — период деактивации последовательности, в течение которого не должны передаваться никакие сообщения; использует атрибут `Milliseconds`;
  - `<BaseRetransmissionInterval>` — период ожидания подтверждения о доставке сообщения, по истечении которого заново отправляется сообщение, если подтверждение не было получено; использует атрибут `Milliseconds`;

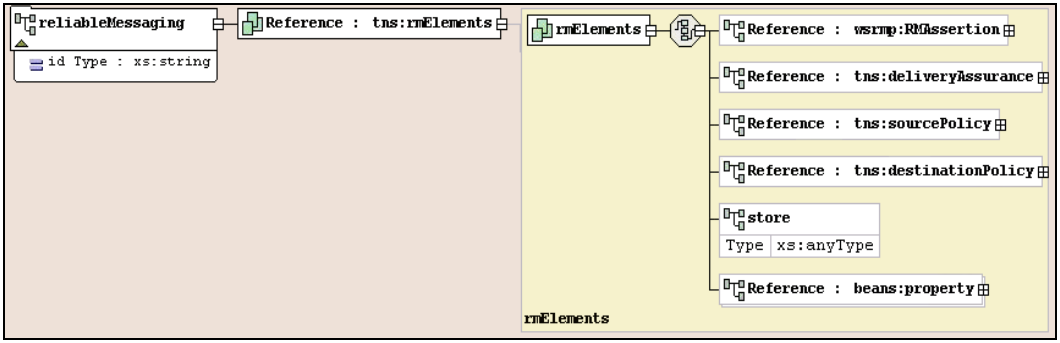


Рис. 5.7. Общая схема элемента `<wsrm-mgr:reliableMessaging>`

- `<ExponentialBackoff>` — указывает, что период ожидания формируется согласно алгоритму Exponential Backoff Algorithm;
- `<AcknowledgementInterval>` — период, после которого посылается подтверждение о доставке сообщения; использует атрибут `Milliseconds`;

❒ `<deliveryAssurance>` конфигурирует гарантию доставки сообщений с помощью дочерних элементов:

- `<ExactlyOnce>` — каждое сообщение доставляется только один раз;
- `<AtLeastOnce>` — альтернатива элементу `<ExactlyOnce>`, каждое сообщение доставляется, по меньшей мере, один раз;
- `<AtMostOnce>` — альтернатива элементам `<ExactlyOnce>` и `<AtLeastOnce>`, каждое сообщение доставляется не более одного раза;
- `<InOrder>` — должен сохранять порядок передачи сообщений;

❒ `<sourcePolicy>` определяет конфигурацию для отправителя сообщений с помощью следующих атрибутов и элементов:

- атрибут `sequenceExpiration` — предлагаемое время жизни последовательности сообщений, включается в CreateSequence-запрос;
- атрибут `acksTo` — адрес конечной точки, по которому будут отправляться подтверждения о получении сообщений и сообщения об ошибках;
- атрибут `includeOffer` — если `true`, тогда в CreateSequence-запрос всегда включается элемент `<wsrm:Offer>`, предлагающий свою последовательность для надежного обмена сообщениями;
- атрибут `offeredSequenceExpiration` — время жизни предлагаемой последовательности;
- элемент `<sequenceTerminationPolicy>` — параметры завершения последовательности сообщений, определяемые с помощью атрибутов `maxLength`, `maxRanges`, `maxUnacknowledged` и `terminateOnShutdown`;

❒ `<destinationPolicy>` определяет конфигурацию для получателя сообщений с помощью следующих атрибутов и элементов:

- атрибут `sequenceExpiration` — время жизни последовательности сообщений;
- атрибут `acceptOffers` — если `true`, тогда предлагаемая отправителем последовательность принимается;
- элемент `<acksPolicy>` — параметры отправки подтверждений доставки сообщений, определяемые с помощью атрибутов `intraMessageThreshold` и `ImmediaAcksTimeout`;

❒ `<store>` определяет хранение сообщений с помощью указания Bean-компонента класса, реализующего `org.apache.cxf.ws.rm.persistence.RMStore`, или ссылки на него. По умолчанию сообщения хранятся в памяти.

После запуска класса `Main` проектов в консоли среды Eclipse можно увидеть, что между клиентом и Web-сервисом состоялся обмен сообщениями, включающий в

себя запрос на создание последовательности надежного обмена сообщениями, подтверждение создания последовательности, сообщение вызова метода Web-сервиса, сообщение с результатом вызова Web-сервиса.

WS-Security

Спецификация WS-Security определяет SOAP-заголовок `<wsse:Security>`, предназначенный для передачи информации, относящейся к обеспечению безопасности обмена SOAP-сообщениями, путем включения в него элементов, представляющих маркеры защиты, такие как логин/пароль или сертификат безопасности, шифрование маркеров защиты и данных сообщения, а также использование цифровой подписи для обеспечения целостности SOAP-сообщения в процессе передачи.

Платформа CXF реализует спецификацию WS-Security на основе проекта Apache WSS4J (**http://ws.apache.org/wss4j/**). Для включения WSS4J-реализации необходимо определить на стороне клиента и Web-сервиса соответствующие Interceptor-обработчики сообщений, отвечающие за добавление и обработку SOAP-заголовка `<wsse:Security>`. Требуемые Interceptor-обработчики обеспечены классами `WSS4JInInterceptor` и `WSS4JOutInterceptor` пакета `org.apache.cxf.ws.security.wss4j`.

Обработчики `WSS4JInInterceptor` и `WSS4JOutInterceptor` конфигурируются с помощью набора свойств, представленных полями класса `org.apache.ws.security.handler.WSHandlerConstants` (табл. 5.2).

Таблица 5.2. *Набор свойств WSS4J-обработчиков*

Ключ	**Значение**
`WSHandlerConstants.ACTION`	`WSHandlerConstants.USERNAME_TOKEN`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.NO_SECURITY`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.SAML_TOKEN_UNSIGNED`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.SAML_TOKEN_SIGNED`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.SIGNATURE`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.ENCRYPT`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.TIMESTAMP`
`WSHandlerConstants.ACTION`	`WSHandlerConstants.SIGN_WITH_UT_KEY`
`WSHandlerConstants.ACTOR`	Имя роли
`WSHandlerConstants.USER`	Логин
`WSHandlerConstants.ENCRYPTION_USER`	Логин для шифрования
`WSHandlerConstants.SIGNATURE_USER`	Логин для подписи
`WSHandlerConstants.ENCRYPTION_USER`	`WSHandlerConstants.USE_REQ_SIG_CERT`
`WSHandlerConstants.PW_CALLBACK_CLASS`	Ссылка на класс реализации `javax.security.auth.callback.CallbackHandler`, отвечающий за получение пароля
`WSHandlerConstants.PW_CALLBACK_REF`	Ссылка на CallbackHandler-объект пароля

Таблица 5.2 *(продолжение)*

Ключ	Значение
WSHandlerConstants.SAML_CALLBACK_CLASS	Ссылка на класс реализации CallbackHandler, отвечающий за получение SAML-утверждения
WSHandlerConstants.SAML_CALLBACK_REF	Ссылка на CallbackHandler-объект SAML-утверждения
WSHandlerConstants.ENC_CALLBACK_CLASS	Ссылка на класс реализации CallbackHandler, отвечающий за получение ключа шифрования
WSHandlerConstants.ENC_CALLBACK_REF	Ссылка на CallbackHandler-объект ключа шифрования
WSHandlerConstants.SIG_PROP_FILE	Адрес конфигурационного файла реализации org.apache.ws.security.components.crypto.Crypto для подписи. Например: org.apache.ws.security.crypto.provider =org.apache.ws.security.components.crypto.Merlin
WSHandlerConstants.SIG_PROP_REF_ID	Ссылка на объект java.util.Properties Crypto-конфигурации подписи
WSHandlerConstants.DEC_PROP_FILE	Адрес файла Crypto-конфигурации расшифровки
WSHandlerConstants.DEC_PROP_REF_ID	Ссылка на объект java.util.Properties Crypto-конфигурации расшифровки
WSHandlerConstants.ENC_PROP_FILE	Адрес файла Crypto-конфигурации шифрования
WSHandlerConstants.ENC_PROP_REF_ID	Ссылка на объект java.util.Properties Crypto-конфигурации шифрования
WSHandlerConstants.SAML_PROP_FILE	Адрес файла конфигурации SAML Issuer
WSHandlerConstants. ENABLE_SIGNATURE_CONFIRMATION	true/false (по умолчанию)
WSHandlerConstants.MUST_UNDERSTAND	true (по умолчанию)/false
WSHandlerConstants.IS_BSP_COMPLIANT	true (по умолчанию)/false
WSHandlerConstants. HANDLE_CUSTOM_PASSWORD_TYPES	true/false (по умолчанию)
WSHandlerConstants.PASSWORD_TYPE_STRICT	true/false (по умолчанию)
WSHandlerConstants. ALLOW_NAMESPACE_QUALIFIED_PASSWORD_TYPES	true/false (по умолчанию)
WSHandlerConstants.USE_ENCODED_PASSWORDS	true/false (по умолчанию)
WSHandlerConstants. USE_SINGLE_CERTIFICATE	true (по умолчанию)/false
WSHandlerConstants.USE_DERIVED_KEY	true (по умолчанию)/false
WSHandlerConstants. USE_DERIVED_KEY_FOR_MAC	true (по умолчанию)/false
WSHandlerConstants.TIMESTAMP_PRECISION	true (по умолчанию)/false
WSHandlerConstants.TIMESTAMP_STRICT	true (по умолчанию)/false
WSHandlerConstants.ENC_SYM_ENC_KEY	true (по умолчанию)/false
WSHandlerConstants.ENC_KEY_NAME	KeyInfo-имя ключа шифрования

Таблица 5.2 *(окончание)*

Ключ	Значение
WSHandlerConstants.PASSWORD_TYPE	WSConstants.PW_DIGEST или WSConstants.PW_TEXT
WSHandlerConstants.ADD_UT_ELEMENTS	Список элементов, добавляемых в маркер UsernameToken
WSHandlerConstants.SIG_KEY_ID	IssuerSerial (по умолчанию) или DirectReference
WSHandlerConstants.SIG_ALGO и WSHandlerConstants.SIG_DIGEST_ALGO	Идентификаторы алгоритмов подписи
WSHandlerConstants.SIGNATURE_PARTS	Список SOAP-элементов для подписи
WSHandlerConstants.WSE_SECRET_KEY_LENGTH	Длина WSE UT_SIGN-ключа (по умолчанию 16 байт)
WSHandlerConstants.DERIVED_KEY_ITERATIONS	Количество итераций производного ключа (по умолчанию 1000)
WSHandlerConstants.ENC_KEY_ID	IssuerSerial, X509KeyIdentifier, DirectReference, Thumbprint, SKIKeyIdentifier или EmbeddedKeyName
WSHandlerConstants.ENC_SYM_ALGO	WSConstants.TRIPLE_DES, WSConstants.AES_128, WSConstants.AES_256 или WSConstants.AES_192
WSHandlerConstants.ENC_KEY_TRANSPORT	WSConstants.KEYTRANSPORT_RSA15
WSHandlerConstants.ENCRYPTION_PARTS	Список SOAP-элементов для шифрования
WSHandlerConstants.TTL_TIMESTAMP	Время Time-To-Live
WSHandlerConstants.TTL_FUTURE_TIMESTAMP	Создаваемое Timestamp-время

На компак-диске

Пример использования WSS4J-реализации находится в папке Примеры\Глава5\WS компакт-диска как проект CXFSecurity.

Проект CXFSecurity создан в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFSecurity демонстрирует включение SOAP-заголовка `<wsse:Security>` с маркером защиты `<wsse:UsernameToken>` в клиентский запрос и передачу запроса Web-сервису по SSL-протоколу на основе определения конфигурации обработчиков `WSS4JInInterceptor` и `WSS4JOutInterceptor`.

Проект CXFSecurity содержит три пакета: `cxfservice`, `cxfclient` и `main` и подключенные библиотеки платформы Apache CXF.

Пакет `cxfservice` включает в себя следующие классы и файлы:

- SEI-интерфейс `CXFService`;
- SEI-класс `CXFServiceImp`;
- конфигурационный Spring-файл beans.xml, определяющий следующие элементы:
  - `<jaxws:endpoint>` объявляет конечную точку Web-сервиса для публикации по адресу **https://localhost:8080/CXFServicePort**. Дочерний элемент `<jaxws:`

inInterceptors> элемента <jaxws:endpoint> конфигурирует обработчик WSS4JInInterceptor для разбора заголовка <wsse:Security> входящего сообщения с помощью свойств <entry key="action" value="UsernameToken"/>, <entry key="passwordType" value="PasswordText"/> и <entry key="password CallbackClass" value="cxfservice.ServerPasswordCallback"/>, где класс ServerPasswordCallback отвечает за проверку входящего логина/пароля маркера защиты <wsse:UsernameToken>;

- <httpj:engine-factory> конфигурирует встроенный Jetty-сервер для использования протокола SSL;

- класс CXFSecurityServer, создающий в конструкторе объект ClassPathXml ApplicationContext на основе Spring-файла beans.xml;
- класс ServerPasswordCallback, реализующий интерфейс javax.security.auth. callback.CallbackHandler для проверки логина/пароля, содержащихся в маркере защиты <wsse:UsernameToken>.

Пакет cxfclient включает в себя следующие классы и файлы:

- конфигурационный Spring-файл beans.xml, определяющий следующие элементы:
  - <jaxws:client> объявляет клиента Web-сервиса с указанием адреса **https://localhost:8080/CXFServicePort** конечной точки Web-сервиса. Дочерний элемент <jaxws:outInterceptors> элемента <jaxws:client> конфигурирует обработчик WSS4JOutInterceptor для включения SOAP-заголовка <wsse:Security> с маркером защиты <wsse:UsernameToken> в исходящее сообщение с помощью свойств <entry key="action" value="UsernameToken"/>, <entry key="user" value="client"/>, <entry key="passwordType" value="PasswordText"/> и <entry key="passwordCallbackClass" value="cxfclient.ClientPasswordCallback"/>, где класс ClientPasswordCallback отвечает за установку пароля маркера <wsse:UsernameToken>;
  - <http-conf:conduit> конфигурирует клиента для использования протокола SSL;
- класс CXFSecurityClient, создающий в конструкторе объект ClassPathXml ApplicationContext с использованием Spring-файла beans.xml и на его основе Proxy-объект Web-сервиса, а также объявляющий метод, который отвечает за вызов метода Proxy-объекта Web-сервиса;
- класс ClientPasswordCallback, реализующий интерфейс javax.security.auth. callback.CallbackHandler для установки пароля маркера <wsse:UsernameToken>.

Пакет main содержит класс Main с методом main() — точкой входа в приложение, который создает объект CXFSecurityServer и объект CXFSecurityClient с вызовом его метода invoke() и выводом результата вызова Web-сервиса в консоль.

После запуска класса Main как Java-приложения в консоли среды Eclipse можно увидеть исходящее и входящее сообщения и строку, возвращаемую методом Web-

сервиса. При этом исходящее сообщение будет содержать SOAP-заголовок `<wsse:Security>` с маркером защиты `<wsse:UsernameToken>`.

WS-SecurityPolicy

Спецификация WS-SecurityPolicy определяет утверждения политики системы безопасности Web-сервисов, удовлетворяемые включением в SOAP-сообщение заголовка `<wsse:Security>` с информацией о маркерах защиты, цифровой подписи и шифровании SOAP-сообщения, передачей сообщения по защищенному протоколу и др.

Платформа CXF версии 2.2 обеспечивает реализацию спецификации WS-SecurityPolicy на основе анализа WSDL-документа Web-сервиса. Использование политики WSDL-документа Web-сервиса при публикации конечной точки Web-сервиса и для создания клиента Web-сервиса является альтернативным, по сравнению с CXF-реализацией WS-Security, способом установки конфигурации для WSS4J-реализации системы безопасности Web-сервисов.

Для определения свойств среды выполнения реализации WS-SecurityPolicy платформа CXF предлагает набор пар "ключ — значение" (табл. 5.3).

Таблица 5.3. *CXF-свойства WS-SecurityPolicy*

Ключ	Значение
`ws-security.username`	Логин маркера UsernameToken
`ws-security.password`	Пароль маркера UsernameToken
`ws-security.callback-handler`	CallbackHandler-объект, обеспечивающий пароль
`ws-security.signature.properties`	Файл или объект конфигурации WSS4J-свойств подписи
`ws-security.encryption.properties`	Файл или объект конфигурации WSS4J-свойств шифрования
`ws-security.signature.username`	Логин ключа в хранилище ключей для подписи
`ws-security.encryption.username`	Логин ключа в хранилище ключей для шифрования
`ws-security.signature.crypto`	Объект WSS4J Crypto подписи
`ws-security.encryption.crypto`	Объект WSS4J Crypto шифрования

НА КОМПАКТ-ДИСКЕ

Пример использования спецификации WS-SecurityPolicy находится в папке Примеры\Глава5\WS компакт-диска как проект CXFSecurityPolicy.

Проект CXFSecurityPolicy создан в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFSecurityPolicy демонстрирует включение SOAP-заголовка `<wsse:Security>` с маркером защиты `<wsse:UsernameToken>` в клиентский запрос и передачу запроса Web-сервису по SSL-протоколу на основе определения политики WSDL-документа Web-сервиса и CXF-свойств WS-SecurityPolicy.

Проект CXFSecurityPolicy содержит три пакета: `cxfservice`, `cxfclient` и `main`, папку wsdl с WSDL-документом Web-сервиса и подключенные библиотеки платформы Apache CXF.

Пакет `cxfservice` включает в себя следующие классы и файлы:

- SEI-интерфейс `CXFService`;
- SEI-класс `CXFServiceImp`;
- конфигурационный Spring-файл beans.xml, определяющий следующие элементы:
  - `<jaxws:endpoint>` объявляет конечную точку Web-сервиса с указанием адреса WSDL-документа. Дочерний элемент `<jaxws:properties>` элемента `<jaxws:endpoint>` указывает класс `ServerPasswordCallback`, отвечающий за проверку входящего логина/пароля маркера защиты `<wsse:UsernameToken>`, с помощью свойства `ws-security.callback-handler`;
  - `<httpj:engine-factory>` конфигурирует встроенный Jetty-сервер для использования протокола SSL;
  - `<import resource="classpath:META-INF/cxf/cxf-extension-policy.xml"/>` и `<import resource="classpath:META-INF/cxf/cxf-extension-ws-security.xml"/>` подключают конфигурацию модулей WS-Policy и WS-SecurityPolicy;
- класс `CXFSecurityPolicyServer`, создающий в конструкторе объект `ClassPathXmlApplicationContext` на основе Spring-файла beans.xml;
- класс `ServerPasswordCallback`, реализующий интерфейс `javax.security.auth.callback.CallbackHandler` для проверки логина/пароля, содержащихся в маркере защиты `<wsse:UsernameToken>`.

Пакет `cxfclient` включает в себя следующие классы и файлы:

- конфигурационный Spring-файл beans.xml, определяющий следующие элементы:
  - `<jaxws:client>` объявляет клиента Web-сервиса с атрибутом `createdFromAPI="true"`, указывающим, что Proxy-объект создается программным образом, а элемент `<jaxws:client>` используется для его дополнительного конфигурирования. Дочерний элемент `<jaxws:properties>` элемента `<jaxws:endpoint>` указывает логин и класс `ClientPasswordCallback`, отвечающий за установку пароля маркера `<wsse:UsernameToken>`, с помощью свойств `ws-security.username` и `ws-security.callback-handler`;
  - `<http-conf:conduit>` конфигурирует клиента для использования протокола SSL;
- класс `CXFSecurityPolicyClient`, создающий в конструкторе Bus-объект на основе конфигурационного файла beans.xml и Proxy-объект на основе WSDL-документа Web-сервиса, а также объявляющий метод, который отвечает за вызов метода Proxy-объекта Web-сервиса;
- класс `ClientPasswordCallback`, реализующий интерфейс `javax.security.auth.callback.CallbackHandler` для установки пароля маркера `<wsse:UsernameToken>`.

Папка wsdl содержит WSDL-документ Web-сервиса с политикой, устанавливающей использование маркера защиты `<wsse:UsernameToken>` для аутентификации/авторизации клиента и протокола HTTPS для защиты передачи сообщений.

Пакет `main` содержит класс `Main` с методом `main()` — точкой входа в приложение, который создает объект `CXFSecurityPolicyServer` и объект `CXFSecurityPolicyClient` с вызовом его метода `invoke()` и выводом результата вызова Web-сервиса в консоль.

После запуска класса `Main` как Java-приложения в консоли среды Eclipse можно увидеть исходящее и входящее сообщения и строку, возвращаемую методом Web-сервиса. При этом исходящее сообщение будет содержать SOAP-заголовок `<wsse:Security>` с маркером защиты `<wsse:UsernameToken>`.

WS-Trust

Спецификация Web Services Trust Language (WS-Trust) описывает стандартный механизм получения клиентом Web-сервиса маркеров защиты у сервиса Security Token Service (STS), которому доверяет запрашиваемый Web-сервис, для их дальнейшего использования для аутентификации, подписи и шифрования SOAP-сообщений.

Платформа CXF версии 2.2 обеспечивает поддержку WS-Trust частично — на стороне клиента, что дает возможность создания CXF-клиента, который способен получить необходимый маркер защиты у STS-сервиса для взаимодействия с Web-сервисом, развернутого вместе со своим STS-сервисом на другой платформе, например, Metro.

Механизм работы такого CXF-клиента состоит из следующих этапов:

1. CXF-клиент получает WSDL-документ запрашиваемого Web-сервиса.
2. CXF-реализация WS-Policy клиента анализирует утверждение `<sp:IssuedToken>` WSDL-политики запрашиваемого Web-сервиса и из элемента `<sp:Issuer>` получает адрес STS-сервиса.
3. CXF-клиент получает WSDL-документ STS-сервиса и генерирует Proxy-объект STS-сервиса.
4. CXF-клиент отправляет RST-запрос STS-сервису.
5. STS-сервис отправляет RSTR-ответ с маркером защиты CXF-клиенту.
6. CXF-клиент формирует запрос Web-сервису, используя полученный маркер защиты.

При обнаружении утверждения `<sp:IssuedToken>` средой выполнения CXF-клиента, CXF-клиент должен обеспечить объект `org.apache.cxf.ws.security.trust.STSClient`, отвечающий за взаимодействие с STS-сервисом. Сделать это можно двумя способами.

Первый способ — это создание экземпляра STSClient программным способом с помощью конструктора `STSClient(Bus b)`, затем конфигурирование созданного

экземпляра методами `set()` класса `STSClient` и включение экземпляра STSClient в контекст запроса методом `(BindingProvider)port).getRequestContext().put ("ws-security.sts.client", STSClient sts)`.

Второй способ — это определение экземпляра STSClient с помощью Spring-файла. В этом случае в конфигурационный Spring-файл клиента включаются элементы:

```
<jaxws:client name="{http://[...]/}[...]" createdFromAPI="true">
  <jaxws:properties>
    <entry key="ws-security.sts.client">
      <bean class="org.apache.cxf.ws.security.trust.STSClient">
        <constructor-arg ref="cxf"/>
        <property name="wsdlLocation" value="[...]/[...].wsdl"/>
        <property name="serviceName"
value="{http://cxf.apache.org/securitytokenservice}SecurityTokenService"/>
        <property name="endpointName"
value="{http://cxf.apache.org/securitytokenservice}SecurityTokenEndpoint"/>
        <property name="properties">
          <map>
            <entry key="ws-security.username" value="..."/>
            <entry key="ws-security.callback-handler"      value="..."/>
            <entry key="ws-security.signature.properties"  value="..."/>
            <entry key="ws-security.encryption.properties" value="..."/>
            <entry key="ws-security.encryption.username"   value="..."/>
          </map>
        </property>
      </bean>
    </entry>
  </jaxws:properties>
</jaxws:client>
```

WS-SecureConversation

Спецификация WS-SecureConversation описывает механизм создания безопасной сессии обмена многочисленными SOAP-сообщениями между несколькими сторонами с применением сессионных ключей, общих для всех участников сессии, на основе маркера защиты контекста (Security Context Token, SCT).

Реализация WS-SecureConversation платформой CXF основана на CXF-реализации WS-SecurityPolicy: чтобы установить использование маркера защиты `<wsc:Security ContextToken>` для цифровой подписи и шифрования SOAP-сообщений, необходимо определить соответствующую политику WSDL-документа конечной точки Web-сервиса, на основе которого будет создаваться клиент Web-сервиса.

Создание безопасных Web-сервисов на платформе CXF менее удобно по сравнению с использованием стека Metro. Среда NetBeans предлагает набор готовых опций, при выборе которых автоматически генерируется конфигурационный WSIT-файл с набором утверждений политики, полностью совместимым с Metro-

реализацией спецификаций WS-*. При работе с платформой CXF нужно вручную подбирать утверждения, совместимые с CXF-реализацией WS-SecurityPolicy.

Для определения свойств среды выполнения реализации WS-SecurityPolicy вместе с WS-SecureConversation платформа CXF предлагает такой же набор пар "ключ — значение" (см. табл. 5.3), отличие заключается в том, что для конфигурирования выполнения политики, определяемой дочерним элементом `<sp:BootstrapPolicy>` элемента `<sp:SecureConversationToken>`, свойства из табл. 5.3 имеют синтаксис `[...].sct`.

> ***На компакт-диске***
>
> Пример совместного использования спецификации WS-SecurityPolicy и WS-SecureConversation находится в папке Примеры\Глава5\WS компакт-диска как проект CXFSecureConversation.

Проект CXFSecureConversation создан в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFSecureConversation демонстрирует передачу сообщений по протоколу SSL с включением в исходящее сообщение маркера защиты `<wsc:SecurityContextToken>`, используемого для цифровой подписи тела сообщения. Для этого в WSDL-документ Web-сервиса включена политика, содержащая элемент `<sp:TransportBinding>` с дочерним элементом `<sp:HttpsToken/>` и элемент `<sp:SecureConversationToken p:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">`. Также политика Web-сервиса требует использование адресации сообщений с помощью утверждения:

```
<wsam:Addressing xmlns:wsam="http://www.w3.org/2007/02/addressing/metadata">
<wsp:Policy/>
</wsam:Addressing>
```

Поэтому в Spring-файл Web-сервиса включен элемент:

```
<import resource="classpath:META-INF/cxf/cxf-extension-addr.xml" />
```

В остальных деталях проект CXFSecureConversation основывается на проекте CXFSecurityPolicy из *разд. "WS-SecurityPolicy"*.

JAX-RS

Платформа CXF позволяет создавать RESTful Web-сервисы двумя способами — посредством программного интерфейса JAX-WS Provider и Dispatch API и с помощью CXF-реализации спецификации "JAX-RS: Java API for RESTful Web Services", упрощающей разработку Web-сервисов, которые основаны на архитектуре REST.

1. Для создания JAX-RS Web-сервиса и его клиента на платформе CXF в среде Eclipse в перспективе Java в меню **File** выберем пункты **New | Java Project**, введем имя проекта CXFJAXRSExample и нажмем кнопку **Finish**.

На компакт-диске

Проект CXFJAXRSExample находится в папке Примеры\Глава5\JAX-RS компакт-диска.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **CXFJAXRSExample** проекта и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets** и укажем **Dynamic Web Module** и **CXF 2.x Web Services**, определив инсталлированную среду выполнения CXF.
3. В проекте CXFJAXRSExample создадим три пакета: `cxfservice`, `cxfclient` и `main`.
4. В пакете `cxfservice` создадим класс `CXFRESTService`, представляющий CXF JAX-RS Web-сервис, и конфигурационный Spring-файл для его развертывания.

Класс ресурса `CXFRESTService` использует аннотацию `javax.ws.rs.Path` для уточнения URL-запроса и аннотации `javax.ws.rs.GET`, `javax.ws.rs.Produces` и `javax.ws.rs.QueryParam` для маркировки метода класса `CXFRESTService`, возвращающего данные в ответ на HTTP-запрос GET клиента (листинг 5.4).

Листинг 5.4. Код класса `CXFRESTService`

```
package cxfservice;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.DefaultValue;

@Path("/resources/")
public class CXFRESTService {
  @GET
  @Produces({"text/plain","text/html"})
  public String getHello(@QueryParam("name") @DefaultValue("")
                          String name) {
  String str="<html><body><h1>Hello!" + "  " +
             name +"</body></h1></html>";
  return str;
  }
}
```

Конфигурационный Spring-файл beans.xml пакета `cxfservice` помогает развертывать JAX-RS Web-сервис CXFRESTService во встроенном Jetty-сервере с помощью элемента `<jaxrs:server>` (листинг 5.5).

Листинг 5.5. Код Spring-файла beans.xml Web-сервиса CXFRESTService

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxrs
http://cxf.apache.org/schemas/jaxrs.xsd">
  <import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-jaxrs-binding.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-http-jetty.xml" />
  <jaxrs:server id="CXFRESTService" address="http://localhost:8000/">
    <jaxrs:serviceBeans>
      <bean class="cxfservice.CXFRESTService"/>
    </jaxrs:serviceBeans>
    <jaxrs:features>
      <bean class="org.apache.cxf.feature.LoggingFeature" />
    </jaxrs:features>
  </jaxrs:server>
</beans>
```

Элемент `<jaxrs:server>` относится к пространству имен `http://cxf.apache.org/jaxrs` и имеет структуру, схожую с элементом `<jaxws:server>` (рис. 5.8).

В пакете `cxfclient` создадим класс `CXFRESTClient` (листинг 5.6), представляющий клиента CXF JAX-RS Web-сервиса и использующий для его создания программный интерфейс CXF Client API, и конфигурационный Spring-файл как альтернативу классу `CXFRESTClient` (листинг 5.7).

Листинг 5.6. Код класса `CXFRESTClient`

```
package cxfclient;
import org.apache.cxf.jaxrs.client.JAXRSClientFactory;
import org.apache.cxf.jaxrs.client.WebClient;
public class CXFRESTClient {
private cxfservice.CXFRESTService client_factory;
private WebClient client_web;
public CXFRESTClient(){
  client_factory=JAXRSClientFactory.create("http://localhost:9000/",
    cxfservice.CXFRESTService.class);  }
public String invoke_factory(String name){
  String str=client_factory.getHello(name);
  return str;}
public String invoke_web(String name) throws Exception{
   client_web = WebClient.create("http://localhost:8000/resources?name=" +
     name);
   String str=client_web.accept("text/plain").get(String.class);
     return str;       }}
```

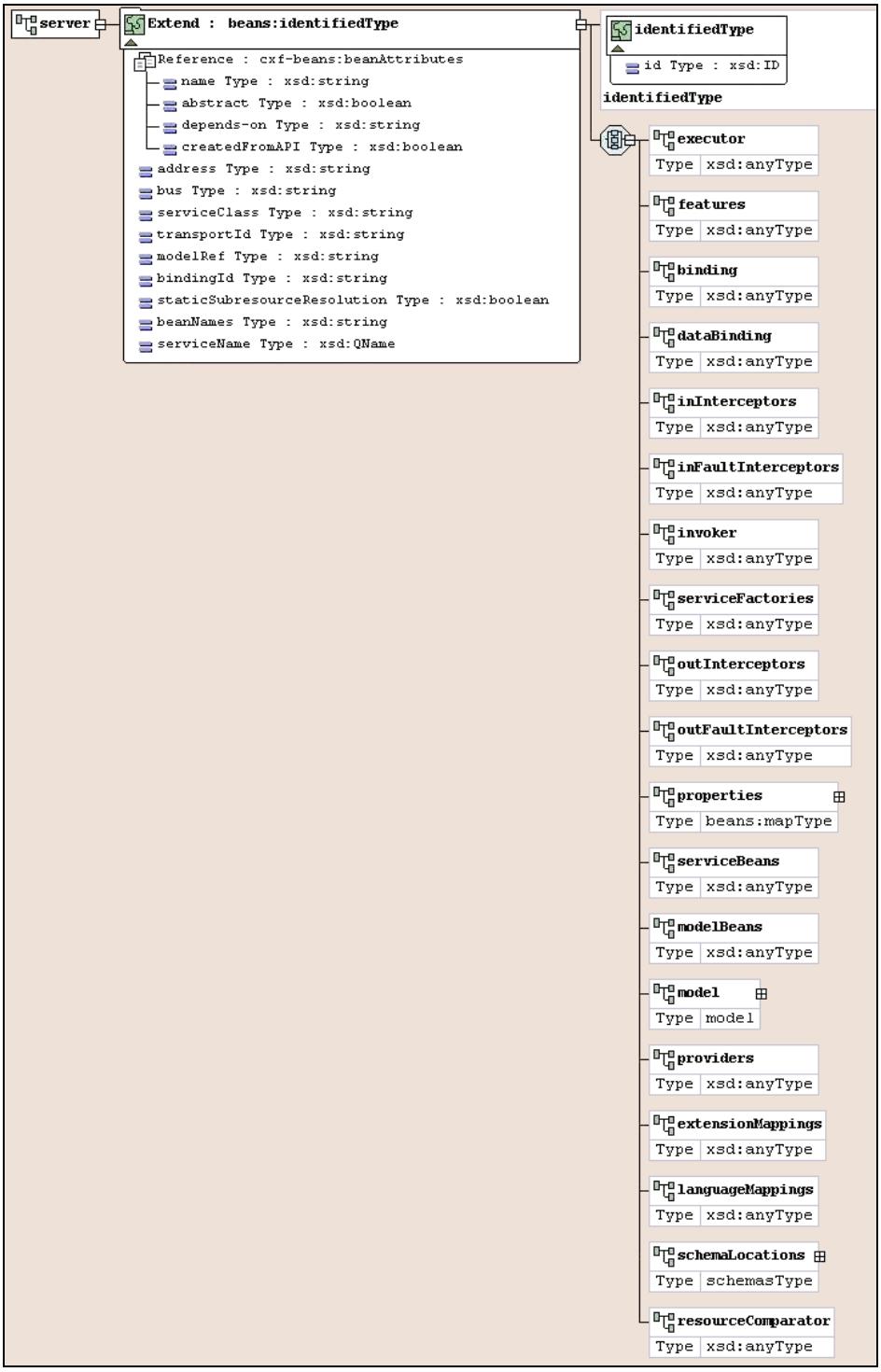


Рис. 5.8. Общая схема элемента `<jaxws:server>`

Листинг 5.7. Код Spring-файла beans.xml клиента Web-сервиса CXFRESTService

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxrs
http://cxf.apache.org/schemas/jaxrs.xsd">
  <jaxrs:client id="CXFRESTClient" address="http://localhost:8000/"
      serviceClass="cxfservice.CXFRESTService" inheritHeaders="true">
    <jaxrs:headers>
      <entry key="Accept" value="text/plain"/>
    </jaxrs:headers>
  </jaxrs:client>
</beans>
```

Платформа CXF предоставляет программный интерфейс Client API, упрощающий создание клиента для CXF JAX-RS Web-сервиса. Интерфейс Client API основывается на двух классах: `org.apache.cxf.jaxrs.client.JAXRSClientFactory` и `org.apache.cxf.jaxrs.client.WebClient`.

Класс `JAXRSClientFactory` дает возможность создать методом `create()` Proxy-объект CXF JAX-RS Web-сервиса, который содержит методы Web-сервиса для его вызова. В качестве аргумента метода `create()`, помимо URL-адреса, выступает интерфейс класса ресурса или сам класс ресурса. Если аргументом метода `create()` служит класс ресурса (как в данном случае), тогда создается CGLIB Proxy-объект и в путь приложения необходимо добавить библиотеку cglib.jar, доступную для скачивания по адресу **http://sourceforge.net/projects/cglib/files/**.

Класс `WebClient` имеет статический метод `create()` для создания объекта WebClient на основе URL-адреса ресурса и методы `get()`, `delete()`, `post()` и `put()`, представляющие HTTP-запросы GET, DELETE, POST и PUT.

Spring-файл beans.xml пакета `cxfclient` использует элемент `<jaxrs:client>` (рис. 5.9) пространства имен `http://cxf.apache.org/jaxrs` для создания Proxy-объекта на основе URL-адреса ресурса и класса ресурса.

В пакете `main` создадим класс `Main` (листинг 5.8), отвечающий за публикацию ресурса и запрос его представления.

Листинг 5.8. Код класса `Main`

```
package main;
import org.apache.cxf.jaxrs.JAXRSServerFactoryBean;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Main {
  public static void main(String[] args) throws Exception {
  JAXRSServerFactoryBean sf = new JAXRSServerFactoryBean();
  sf.setResourceClasses(cxfservice.CXFRESTService.class);
  sf.setAddress("http://localhost:9000/");
  sf.create();
```

```
  new ClassPathXmlApplicationContext(new String[]
     {"cxfservice/beans.xml" });
  cxfclient.CXFRESTClient client=new cxfclient.CXFRESTClient();
  System.out.println(client.invoke_factory("User_1"));
  System.out.println(client.invoke_web("User_2"));
  ClassPathXmlApplicationContext context = new
    ClassPathXmlApplicationContext(new String[] {"cxfclient/beans.xml"});
  cxfservice.CXFRESTService client_spring =
   (cxfservice.CXFRESTService)context.getBean("CXFRESTClient");
  System.out.println(client_spring.getHello("User_3"));
  System.exit(0);
}}
```

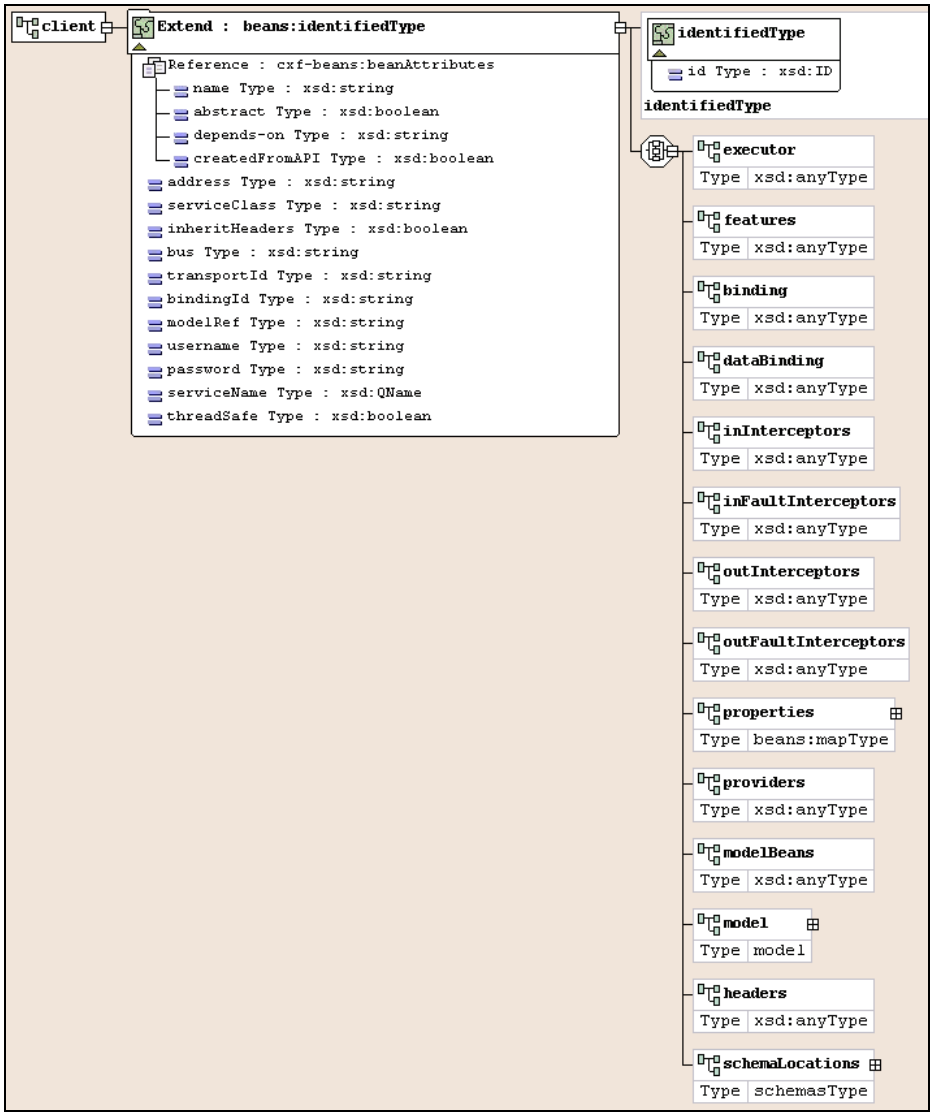


Рис. 5.9. Общая схема элемента `<jaxrs:client>`

Класс `Main` демонстрирует публикацию ресурса CXFRESTService программным способом с использованием класса `org.apache.cxf.jaxrs.JAXRSServerFactoryBean` и с помощью конфигурационного Spring-файла beans.xml пакета `cxfservice`. Клиент ресурса также создается в классе `Main` двумя способами — программным способом с применением класса `cxfclient.CXFRESTClient` и с помощью конфигурационного Spring-файла beans.xml пакета `cxfclient`.

После запуска класса `Main` как Java-приложения в консоли среды Eclipse можно увидеть строку, возвращаемую методом `getHello()` класса `CXFRESTService`.

JavaScript

Интегрированность платформы CXF с платформой Rhino (**http://www.mozilla.org/rhino/**) дает возможность создания Web-сервисов и клиентов Web-сервисов на языке JavaScript.

Платформа CXF позволяет разработку на языке JavaScript только низкоуровневых SOAP Web-сервисов, являющихся аналогами JAX-WS Provider-сервисов и использующих для обработки XML-сообщений интерфейс DOMSource API или язык ECMAScript for XML (E4X).

Аналогом аннотации `javax.xml.ws.WebServiceProvider` в такого рода Web-сервисах служит объявление переменной `var WebServiceProvider = {};` со следующими свойствами:

- `wsdlLocation` — URL-адрес WSDL-документа;
- `serviceName` — имя сервиса;
- `portName` — имя порта;
- `targetNamespace` — пространство имен сервиса;
- `ServiceMode` — режим обработки сообщений, по умолчанию PAYLOAD;
- `BindingMode` — режим связывания, по умолчанию SOAP 1.1/HTTP binding;
- `EndpointAddress` — URL-адрес клиента сервиса.

После определения свойств переменной `WebServiceProvider` реализация CXF JavaScript Web-сервиса заключается в объявлении функции `WebServiceProvider.invoke = function(document) {}`, обрабатывающей и возвращающей объект `javax.xml.transform.dom.DOMSource` или объект E4X XML.

Разворачиваются Web-сервисы, созданные на языке JavaScript, в среде выполнения CXF с помощью класса `org.apache.cxf.js.rhino.ServerApp` командой со следующим синтаксисом:

```
java org.apache.cxf.js.rhino.ServerApp [ -a addressURL ]
    [ -b baseAddressURL ] file.js [ file2.js file3.jsx ... ]
```

где *`addressURL`* — адрес публикации конечной точки сервиса, а файлы с расширением js — JavaScript-файлы, файлы с расширением jsx — E4X-файлы. Опция *`baseAddressURL`* позволяет публиковать несколько сервисов по одному базовому адресу.

Создание JavaScript-клиента Web-сервиса на основе платформы CXF возможно тремя способами:

- с помощью CXF-инструмента `wsdl2js`, который из WSDL-документа генерирует JavaScript-код клиента;
- с помощью CXF-инструмента `java2js`, который из Java-кода Web-сервиса генерирует JavaScript-код клиента;
- динамически, с использованием URL-запроса `http://host:port/port_and_or_service?js`.

> ***На компакт-диске***
>
> Пример CXF JavaScript-клиента Web-сервиса, динамически получающего JavaScript-код для создания Proxy-объекта Web-сервиса, находится в папке Примеры\Глава5 как проект CXFJavaScript.

Проект CXFJavaScript создан в среде Eclipse в перспективе Java с подключенными библиотеками платформы CXF.

Проект CXFJavaScript состоит из двух пакетов: `cxfservice` и `cxfclient`.

Пакет `cxfservice` содержит следующие классы и файлы:

- SEI-интерфейс `CXFService`;
- SEI-класс `CXFServiceImp`;
- конфигурационный Spring-файл beans.xml, определяющий следующие элементы:
  - `<jaxws:endpoint>` объявляет конечную точку Web-сервиса с указанием адреса для ее публикации **http://localhost:9000/CXFServicePort**;
  - `<import resource="classpath:META-INF/cxf/cxf-extension-javascript-client.xml"/>` подключают конфигурацию модуля `org.apache.cxf.javascript.JavascriptQueryHandler`, отвечающего за выдачу клиентского JavaScript-кода конечной точки по запросу;
- класс `CXFServer`, имеющий метод `main()` для запуска класса как Java-приложения и создающий в методе `main()` объект `ClassPathXmlApplicationContext` на основе Spring-файла beans.xml для публикации конечной точки Web-сервиса.

Пакет `cxfclient` содержит HTML-страницу JavascriptClient.html с кодом создания Proxy-объекта Web-сервиса и вызовом его метода (листинг 5.9).

Листинг 5.9. HTML-код клиента Web-сервиса

```
<html>
<head>
<script type="text/javascript"
        src="http://localhost:9000/CXFServicePort?js"></script>
<script type="text/javascript">
  var CXFService = new cxfservice__CXFService;
  var responseSpan;
```

```
  function getHelloResponse(response)
  { responseSpan = document.getElementById('Response');
    responseSpan.firstChild.nodeValue = response.getReturn();
  }
  function getHelloError(error)
  { alert('error ' + error);  }
  function invoke()
  { responseSpan = document.getElementById('Response');
    var name="User";
    CXFService.getHello(getHelloResponse, getHelloError, name);
  }
</script>
</head>
<body>
  <form>
    <input type="button" value="invoke"
           name="getHello" onClick="invoke()">
   <p/>
   <span id='Response'>Сервис не вызван</span>
  </form>
</body>
</html>
```

В клиентском коде элемент `<script type="text/javascript" src="http://localhost:9000/CXFServicePort?js"></script>` отвечает за динамическую загрузку JavaScript-кода, представляющего артефакты Web-сервиса на стороне клиента.

Далее в теге `<script type="text/javascript">` на основе загруженного JavaScript-кода создается Proxy-объект Web-сервиса CXFService и объекты `getHelloResponse`, `getHelloError`, отвечающие за получения ответа или ошибки от Web-сервиса.

Функция `invoke` отвечает за вызов метода `getHello()` Web-сервиса CXFService, в результате которого функцией `getHelloResponse` возвращаемая строка метода отобразится в той части HTML-странички, которая определяется тегом `<span id='Response'>`.

После запуска класса `CXFServer` как Java-приложения и страницы JavascriptClient.html как **Open With | Web Browser** с последующим нажатием кнопки **invoke** на странице браузера отобразится строка, возвращаемая методом `getHello()` Web-сервиса CXFService.

ГЛАВА 6

Проект Axis2

Проект Apache Axis2 (**http://axis.apache.org/axis2/java/core/**) представляет модульную платформу для создания и развертывания SOAP Web-сервисов и клиентов SOAP Web-сервисов и является преемником стека Apache Axis SOAP. Проект Apache Axis2 предоставляет две реализации технологии Web-сервисов — Apache Axis2/Java и Apache Axis2/C.

Стек Apache Axis2/Java поддерживает следующие технологии и стандарты:

- спецификации технологии Web-сервисов первого уровня SOAP 1.1 и SOAP 1.2, WSDL 1.1 и WSDL 2.0;
- спецификации технологии Web-сервисов второго уровня WS-Policy, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction, WS-Security, WS-Addressing;
- модель документов Apache Axiom;
- транспортные протоколы HTTP, Local, TCP, SMTP, JMS и др.;
- стандарты платформы Java реализации технологии Web-сервисов SAAJ 1.1, JAX-WS и JAXB;
- архитектуру Representational State Transfer (REST) на основе спецификации WSDL 2.0 HTTP Bindings;
- технологии передачи бинарных данных Message Transmission Optimization Mechanism (MTOM), XML Optimized Packaging (XOP) и SOAP with Attachments;
- связывание данных Axis Data Binding (ADB), XMLBeans, JibX, JaxBRI;
- платформу Spring Framework.

В отличие от платформы CXF платформа Axis2 не содержит своей реализации спецификации JAX-RS, а поддерживает архитектуру REST на основе WSDL 2.0 HTTP-связывания. Также стек Axis2 не так тесно интегрирован с платформой Spring, как стек CXF. Однако поддержка стеком Axis2 спецификаций WS-* является более полной по сравнению со стеком CXF.

Архитектура платформы Axis2 состоит из следующих основных компонентов и компонентов расширения.

- Основной компонент "Информационная модель" обеспечивает хранение информации, необходимой для обработки сообщений, включая информацию о конфигурации среды выполнения Axis2, Axis2-модулях, сервисах и их операциях, транспорте и т. д. Информационная модель состоит из двух связанных между собой иерархий объектов — Context и Description. Description-иерархия объектов `AxisConfiguration`, `AxisServiceGroup`, `AxisService`, `AxisOperation`, `AxisMessage`, `AxisModule` представляет статические данные, существующие в виде параметров (ключ — значение), составляемых из дескрипторов различных уровней — axis2.xml, services.xml и module.xml. Context-иерархия представляет динамические данные обработки сообщений средой выполнения Axis2, хранящиеся как свойства (в виде пар "ключ — значение") контекстов различного уровня — ConfigurationContext, ServiceGroupContext, ServiceContext, OperationContext и MessageContext. Context-иерархия объектов создается на основе Description-иерархии объектов.
- Основной компонент "Модель обработки XML-данных" обеспечивает XML InfoSet модель представления SOAP-сообщений на основе модели AXIs Object Model (AXIOM).
- Основной компонент "Модель обработки SOAP-протокола" обеспечивает обработку SOAP-сообщений с помощью Handler-обработчиков, при этом входящее сообщение проходит через "In" pipe-фазы (табл. 6.1), а исходящее сообщение — через "Out" pipe-фазы (табл. 6.2). Каждая фаза обработки сообщения определяется своим набором Handler-обработчиков. Основные Handler-обработчики платформы Axis2 — это Dispatcher-обработчики, связывающие поступающее в среду выполнения Axis2 SOAP-сообщение с конкретным развернутым сервисом и его операциями, обработчики Message Receiver, отвечающие за потребление SOAP-сообщения приложением, и обработчики Transport Sender, отвечающие за отправку SOAP-сообщения конечной точке. Поддержка спецификаций WS-* осуществляется Axis2-модулями, содержащими специфические наборы Handler-обработчиков, работающих на определенных фазах обработки сообщений. Платформа Axis2 реализует шаблоны обмена сообщениями Message Exchange Patterns (MEPs) на основе комбинаций pipe-процессов "In" и "Out".
- Основной компонент "Модель развертывания" дает возможность разворачивать и обновлять Web-сервисы в виде архивных aar-файлов на работающей платформе Axis2 без перезапуска за счет прослушивания папки services файловой системы платформы. При этом конфигурацию развертывания AAR-архива Web-сервиса определяет дескриптор services.xml. Конфигурацию развертывания самой платформы Axis2 определяет дескриптор axis2.xml. Дескрипторы module.xml определяют конфигурацию развертывания Axis2-модулей *.mar, расширяющих функциональность платформы Axis2.
- Основной компонент "Программный интерфейс Client API" обеспечивает создание клиента Web-сервиса с помощью программных интерфейсов ServiceClient API и OperationClient API.

- Основной компонент "Транспортные протоколы" обеспечивает различные транспортные протоколы передачи сообщений, такие как HTTP, Local, TCP, SMTP, JMS и др.
- Компонент расширения "Генерация WSDL-описания и Java-кода" обеспечивает инструменты WSDL2Java и Java2WSDL. Для интеграции этих инструментов со средой Eclipse необходимо скачать Eclipse-плагин Code Generator Wizard (**http://axis.apache.org/axis2/java/core/tools/index.html**) и поместить его в папку plugins каталога среды Eclipse. После этого в окне **Project Explorer** среды Eclipse при выборе пунктов **New | Other | Axis2 Wizards | Axis2 Code Generator** можно генерировать Java-код из WSDL-описания и наоборот.
- Компонент расширения "Связывание данных" обеспечивает поддержку различных технологий связывания данных, определяемых XML-схемой WSDL-документа Web-сервиса, с Java-объектами. Платформа Axis2 интегрирована с системой ADB (Axis Data Binding) и поддерживает такие технологии, как XMLbeans, JibX и JaxBRI.

Таблица 6.1. *Фазы pipe-процесса "In"*

Фаза	Описание
Transport	Обработка транспортной информации сообщения
Pre-Dispatch	Обработка адресной информации сообщения
Dispatch	Поиск сервиса и его операции, для которых предназначено сообщение
User Defined	Обработка сообщения пользовательскими обработчиками
Message Validation	Проверка правильности обработки SOAP-протокола
Message Processing	Выполнение бизнес-логики SOAP-сообщения

Таблица 6.2. *Фазы pipe-процесса"Out"*

Фазы	Описание
Message Initialize	Инициализация обработчиков сообщения
User	Обработка сообщения пользовательскими обработчиками
Transports	Подготовка и отправка сообщения конечной точке

Платформа Axis2 (**http://axis.apache.org/axis2/java/core/download.cgi**) поставляется четырьмя различными дистрибутивами.

Первый вид дистрибутива, Binary Distribution, позволяет запускать платформу Axis2 поверх платформы Java SE как отдельное приложение с помощью пакетного файла axis2server папки bin каталога дистрибутива (предварительно необходимо установить переменные среды `JAVA_HOME` и `AXIS2_HOME`). После запуска платформы Axis2 как отдельного сервера адрес **http://localhost:8080/axis2/services/** дает возможность посмотреть развернутые Web-сервисы. Дистрибутив Binary Distribution

по умолчанию содержит Axis2-модули ping, metadataExchange, addressing, script, soapmonitor, mtompolicy в папке repository\modules каталога дистрибутива. Остальные Axis2-модули для поддержки WS-RM, WS-Security и WS-SecureConversation необходимо загрузить дополнительно (**http://axis.apache.org/axis2/java/core/modules/index.html**).

Дистрибутив Source Distribution содержит исходный код платформы Axis2.

Дистрибутив WAR (Web Archive) Distribution дает возможность разворачивать платформу Axis2 поверх сервера приложений с помощью WAR-файла axis2.war. Например, чтобы развернуть платформу Axis2 поверх сервера Tomcat, необходимо поместить файл axis2.war в папку webapps дистрибутива сервера, тогда при запуске сервера Tomcat платформа Axis2 будет развернута как Web-приложение и станет доступна для администрирования по адресу **http://localhost:8080/axis2/**.

Дистрибутив Documents Distribution содержит документацию платформы Axis2.

Конфигурационный файл axis2.xml

Файл axis2.xml папки conf определяет всю необходимую конфигурацию развертывания платформы Axis2.

Корневым элементом файла axis2.xml служит элемент `<axisconfig>` (рис. 6.1), содержащий следующие атрибуты и дочерние элементы.

- Атрибут `name` — имя платформы (`name="AxisJava2.0"`).
- Элемент `<messageReceiver>` указывает класс обработчика `org.apache.axis2.engine.MessageReceiver`, отвечающего за вызов бизнес-логики сервиса, для определенного MEP-шаблона с помощью атрибутов `class` и `mep`.
- Элемент `<module>` определяет активацию модуля на глобальном уровне для всех сервисов системы, используя атрибуты `name`, `ref` и `version`. По умолчанию дескриптор axis2.xml активирует только один модуль — модуль адресации `<module ref="addressing"/>`.
- Элемент `<parameter>` определяет параметры конфигурации с помощью атрибута `locked` (`true`/`true`), который указывает, может ли значение параметра переопределяться, и атрибута `name`, который указывает имя параметра. Элемент `parameter` позволяет определять такие параметры, как `hotdeployment` (развертывание сервисов динамически), `hotupdate` (обновление сервисов динамически), `enableMTOM`, `enableSwA`, `userName` и `password` (логин/пароль модуля администрирования), `ModulesDirectory`, `disableREST`, `disableSOAP12` и др.
- Элемент `<defaultModuleVersions>` указывает версии модулей по умолчанию с помощью дочерних элементов `<module>`.
- Элемент `<deployer>` определяет развертывание файлов различного формата с помощью атрибутов `class` (имя Axis2-класса, отвечающего за развертывание), `directory` (каталог нахождения файла) и `extension` (расширение файла). В частности, по умолчанию дескриптор axis2.xml позволяет развертывать POJO Web-

сервисы в виде класс-файлов в папке pojo, JAX-WS Web-сервисы в виде JAR-файлов в папке servicejars.

- Элемент `<hostConfiguration>` определяет конфигурацию хоста с помощью атрибутов `ip` и `port`.
- Элемент `<listener>` указывает класс реализации интерфейса `org.apache.axis2.engine.AxisObserver`, отвечающий за обработку таких событий, как развертывание, удаление, активация/деактивация сервиса, развертывание и удаление модуля. Элемент имеет атрибут `class` и дочерние элементы `<parameter>`.

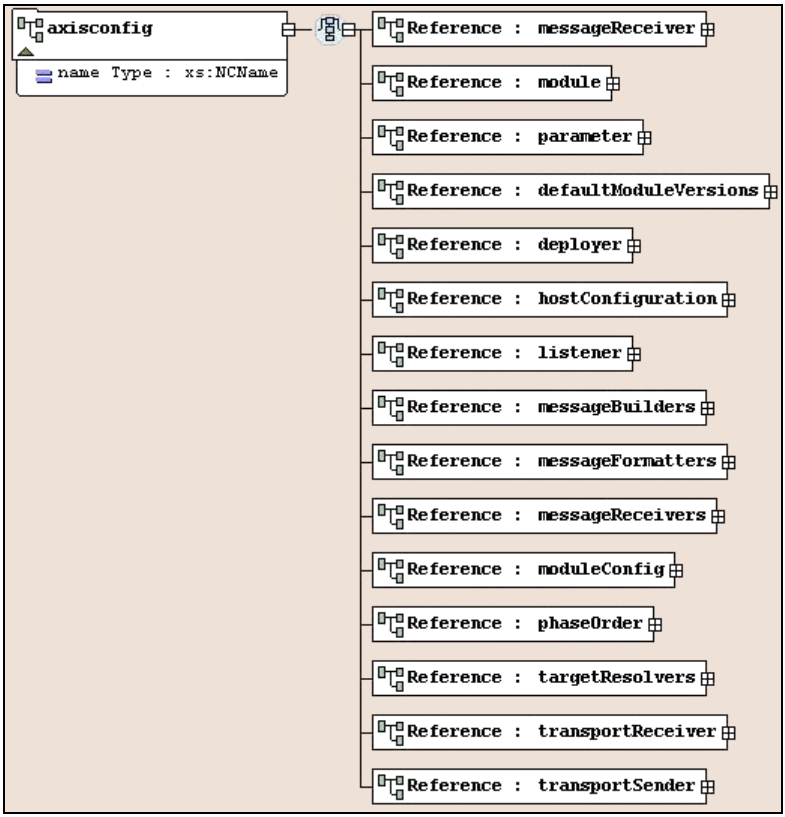


Рис. 6.1. Общая схема элемента `<axisconfig>`

- Элемент `<messageBuilders>` указывает с помощью дочерних элементов `<messageBuilder>` классы, отвечающие за конвертацию поступающих не-SOAP-сообщений в SOAP-сообщения, т. к. платформа Axis2 работает только с SOAP-сообщениями. Элемент имеет атрибуты `contentType` (тип содержимого поступающего сообщения, например, `application/xml`) и `class` (имя класса обработчика, например, `org.apache.axis2.builder.ApplicationXMLBuilder`).
- Элемент `<messageFormatters>` указывает с помощью дочерних элементов `<messageFormatter>` классы, отвечающие за сериализацию Java-объектов в опре-

деленные форматы исходящих сообщений. Элемент имеет атрибуты contentType (тип содержимого исходящего сообщения, например, application/xml) и class (имя класса обработчика, например, org.apache.axis2.transport.http.ApplicationXMLFormatter — производит сериализацию в сообщение формата "plain old XML" (POX)).

- Элемент <messageReceivers> определяет на глобальном уровне с помощью дочерних элементов <messageReceiver> Axis2 Handler-обработчики последней фазы обработки входящего сообщения, используя атрибуты class и mep.
- Элемент <moduleConfig> указывает параметры конфигурации модуля с помощью атрибута name (имя модуля) и дочерних элементов <parameter>.
- Элемент <phaseOrder> определяет порядок фаз и Handler-обработчики фаз для входящего и исходящего процессов обработки сообщений с помощью следующих атрибутов и дочерних элементов:
  - атрибут type — тип процесса обработки сообщений InFlow, OutFlow, OutFaultFlow, InFaultFlow;
  - элемент phase группирует Handler-обработчики для определенной фазы. Элемент имеет атрибуты name (имя фазы) и class (имя класса расширения org.apache.axis2.engine.Phase обработки фазы), а также дочерние элементы <handler> с атрибутами class (имя класса Handler-обработчика) и name (идентификатор обработчика).
- Элемент <targetResolvers> указывает с помощью атрибутов class дочерних элементов <targetResolver> классы реализации org.apache.axis2.util.TargetResolver, отвечающие за разрешение вызываемого URL-адреса.
- Элемент <transportReceiver> определяет класс обработки транспортного протокола входящих сообщений с помощью атрибутов class (имя класса) и name (идентификатор транспортного протокола), а также дочерних элементов <parameter>.
- Элемент <transportSender> определяет класс обработки транспортного протокола исходящих сообщений с помощью атрибутов class (имя класса) и name (идентификатор транспортного протокола), а также дочерних элементов <parameter>.

Архив AAR и развертывание Web-сервиса

Axis2-архив Web-сервиса *.aar предназначен для развертывания Web-сервиса в среде выполнения Axis2. Содержанием aar-архива является набор класс-файлов Web-сервиса и конфигурационный файл services.xml папки META-INF архива.

Конфигурационный файл services.xml содержит в качестве корневого элемента элемент <service>, в случае если архив включает в себя один Web-сервис, или элемент <serviceGroup> — если архив предназначен для развертывания нескольких Web-сервисов.

Элемент `<serviceGroup>` имеет следующие дочерние элементы:

- `<service>` — конфигурация отдельного Web-сервиса;
- `<parameter>` — разделяемый параметр группы Web-сервисов;
- `<module>` — активация совместного модуля группы Web-сервисов.

Элемент `<service>` (рис. 6.2) имеет следующие атрибуты и дочерние элементы:

- атрибут `class` — имя класса сервиса, управляющего его жизненным циклом;
- атрибут `name` — имя сервиса;
- атрибут `scope` — время жизни сессии сервиса. Возможные значения:
  - `application` — время жизни сессии определяется временем жизни системы;
  - `soapsession` — время жизни сессии определяется параметром SOAP-заголовка;
  - `transportsession` — время жизни сессии определяется временем жизни транспорта;
  - `request` (по умолчанию) — время жизни сессии определяется временем обработки запроса;

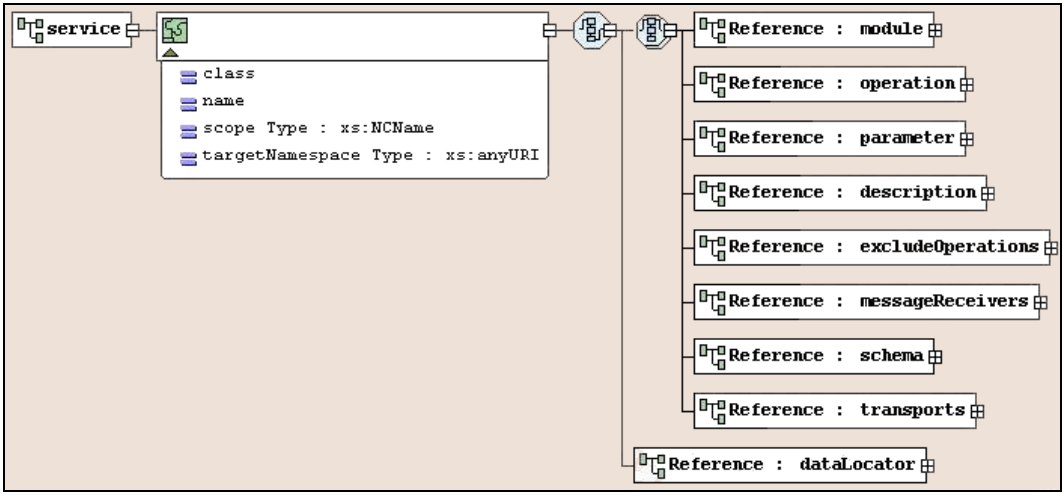


Рис. 6.2. Общая схема элемента `<service>`

- атрибут `targetNamespace` — целевое пространство имен сервиса;
- элемент `<module>` — активация Axis2-модуля на уровне сервиса, определяемая с помощью атрибута `ref`;
- элемент `<operation>` (рис. 6.3) — операция сервиса, описываемая с помощью следующих атрибутов и дочерних элементов:
  - атрибут `mep` указывает WSDL 2.0 шаблон MEP (Message Exchange Pattern) операции;

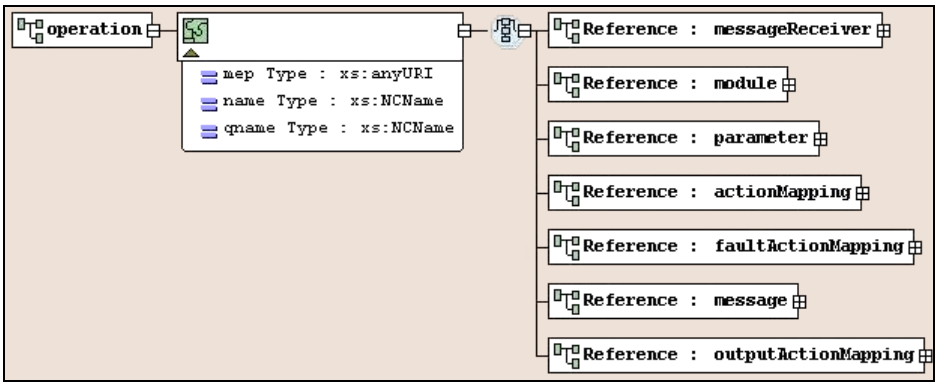


Рис. 6.3. Общая схема элемента `<operation>`

- атрибут `name` — имя операции;
- атрибут `qname` — QName-имя операции;
- элемент `<messageReceiver>` определяет Axis2 Handler-обработчик последней фазы обработки входящего сообщения с помощью атрибутов `class` и `mep`;
- элемент `<module>` определяет активацию Axis2-модуля на уровне операции с помощью атрибута `ref`;
- элемент `<parameter>` — параметр операции. Атрибут `locked` (`true`/`false`) указывает, может ли значение параметра переопределяться, атрибут `name` указывает имя параметра;
- элемент `<actionMapping>` — входящее Action-значение;
- элемент `<faultActionMapping>` — Action-значение ошибки;
- элемент `<message>` определяет сообщение для операции с помощью атрибута `label` (`in`, `out`) и дочернего элемента `<parameter>`;
- элемент `<outputActionMapping>` — исходящее Action-значение;

❒ элемент `<parameter>` определяет параметр сервиса, используя атрибуты `locked` и `name`;

❒ элемент `<description>` — описание сервиса;

❒ элемент `<excludeOperations>` исключает из использования операции сервиса с помощью дочерних элементов `<operation>`;

❒ элемент `<messageReceivers>` указывает для всех операций сервиса Axis2 Handler-обработчики последней фазы обработки входящего сообщения с помощью атрибутов `class` и `mep` дочерних элементов `<messageReceiver>`;

❒ элемент `<schema>` указывает целевое пространство имен XML-схемы данных сервиса с помощью атрибута `schemaNamespace`;

❒ элемент `<transports>` определяет транспортные протоколы сервиса с помощью дочерних элементов `<transport>`;

- элемент `<dataLocator>` указывает, используя атрибут `class` и дочерний элемент `<dialectLocator>` (атрибуты `class` и `dialect`), `AxisDataLocator`-класс, используемый для извлечения метаданных WSDL, Policy, Schema Web-сервиса.

В *разд. "Пример создания JAX-WS Web-сервиса и JAX-WS-клиента" главы 3* уже рассматривалось создание Axis2 Web-сервиса с использованием среды Eclipse.

В данном примере при определении в свойствах проекта Web-приложения **Properties**, в меню **Project Facets**, "включить генерацию Web-сервиса платформой Axis2" (вариант **Axis2 Web Services**), среда Eclipse включала в Web-приложение компоненты платформы Axis2 и Web-приложение фактически становилось средой выполнения Axis2, которая разворачивалась в сервере Tomcat. Далее при генерации Web-сервиса из POJO-класса (выбор **New | Other | Web Services | Web Service**) в папке WebContent\WEB-INF\services проекта создавалась папка Web-сервиса с класс-файлом Web-сервиса и папкой META-INF, содержащей Axis2-файл services.xml, реализуя тем самым Axis2-модель развертывания Web-сервисов — Web-приложение становилось средой выполнения Axis2 с включенным в нее Web-сервисом.

Если взять сгенерированную средой Eclipse папку Web-сервиса с класс-файлом Web-сервиса и папкой META-INF с Axis2-файлом services.xml и поместить ее в папку webapps\axis2\WEB-INF\services сервера Tomcat, в котором предварительно был развернут файл axis2.war (путем размещения файла axis2.war в папке webapps каталога сервера и запуском сервера утилитой `startup` папки bin), тогда Web-сервис будет автоматически развернут на сервере Tomcat, и это можно будет увидеть на странице браузера по адресу **http://localhost:8080/axis2/services/listServices**.

Поместив сгенерированную средой Eclipse папку Web-сервиса в папку repository\services бинарного дистрибутива платформы Axis2, после запуска утилиты `axis2server` папки bin Web-сервис будет развернут на Axis2-сервере поверх платформы Java SE.

Чтобы иметь возможность создания AAR-архива в среде Eclipse, необходимо предварительно скачать Eclipse-плагин Service Archive Generator (**http://axis.apache.org/axis2/java/core/tools/index.html**) и поместить его в папку dropins каталога среды Eclipse. После этого в окне **Project Explorer** среды Eclipse при выборе пунктов **New | Other | Axis2 Wizards | Axis2 Service Archiver** можно из класс-файлов Web-сервиса сгенерировать AAR-архив Web-сервиса. Разместив сгенерированный AAR-архив Web-сервиса в папке webapps\axis2\WEB-INF\services сервера Tomcat или папке repository\services платформы Axis2, Web-сервис будет развернут на сервере Tomcat или Axis2-сервере соответственно.

По умолчанию конфигурационный файл axis2.xml своими элементами `<deployer>` позволяет также разворачивать Web-сервис в виде набора класс-файлов или в виде JAR-файла.

Для развертывания Web-сервиса в виде класс-файла без дескриптора services.xml необходимо создать папку pojo в каталоге, содержащем папку services, и поместить в него класс-файл Web-сервиса, представленного POJO-классом пакета по умолчанию.

Для развертывания Web-сервиса в виде JAR-файла без дескриптора services.xml необходимо создать папку servicejars в каталоге, содержащем папку services, и поместить в него JAR-файл Web-сервиса, представленного POJO-классом, промаркированным JAX-WS-аннотациями.

Развертывание Web-сервиса возможно также программным способом с помощью Axis2 API.

Самый простой способ развертывания программным способом Web-сервиса с запуском Axis2-сервера — это создать класс с методом `main()`, содержащим следующий код:

```
new org.apache.axis2.engine.AxisServer().deployService([имя класса
Web-сервиса].class.getName());
```

После запуска созданного класса как Java-приложения при условии, что все необходимые библиотеки платформы Axis2 добавлены в путь приложения, URL-адрес **http://localhost:6060/axis2/services/** дает возможность доступа к развернутому сервису.

> ***На компакт-диске***
>
> Пример развертывания Web-сервиса различными способами находится в папке Примеры\Глава6\Deployment компакт-диска как проект Axis2_WebService среды Eclipse.

Конфигурационный файл services.xml, помимо определения конфигурации сервиса, транспортного протокола, Axis2-обработчиков, активации Axis2-модулей, может служить инструментом определения политики Web-сервиса. Включение в элемент `<service name="...">` элемента `<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">` с политикой Web-сервиса обеспечивает при его развертывании генерацию WSDL-документа с включенной в него политикой. При этом Axis2-инструмент WSDL2Java обеспечивает генерацию необходимым образом конфигурированной заглушки на стороне клиента Web-сервиса.

Модули Axis2

Модули платформы Axis2 позволяют расширять обеспечение качества QoS (Quality of Service) Web-сервисов в реализации таких опций, как надежность, безопасность и поддержка транзакций.

Axis2-модуль представляет собой MAR-архив, содержащий набор Handler-обработчиков и сопутствующих ресурсов для обработки сообщений на определенных фазах pipe-процесса. Конфигурацию развертывания Axis2-модуля определяет дескриптор module.xml папки META-INF MAR-архива.

Корневым элементом дескриптора module.xml служит элемент `<module>` (рис. 6.4), имеющий следующие атрибуты и дочерние элементы:

- атрибут `name` — имя модуля;
- атрибут `class` — класс реализации интерфейса `org.apache.axis2.modules.Module`, обеспечивающий жизненный цикл модуля;

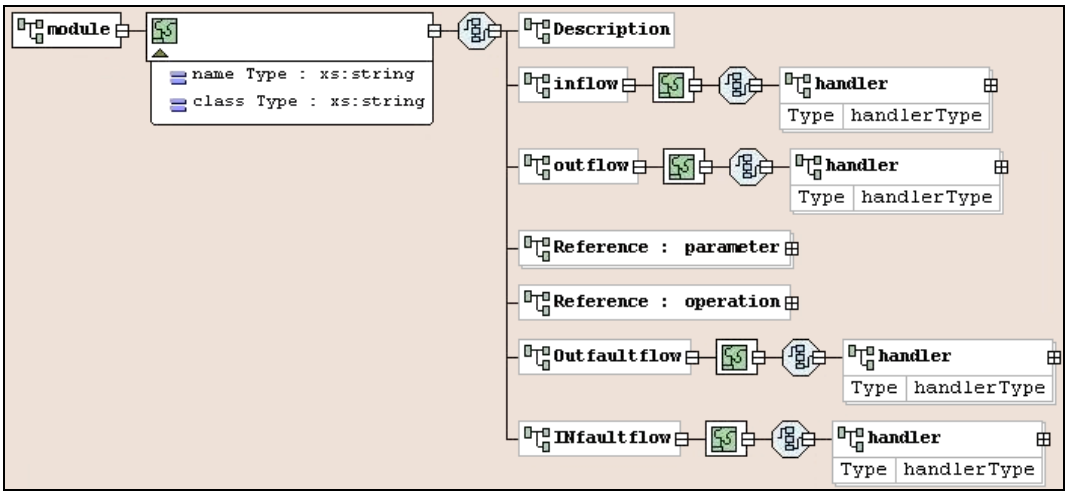


Рис. 6.4. Общая схема элемента `<module>`

- элемент `<Description>` — описание модуля;
- элемент `<InFlow>` определяет включение Handler-обработчиков в определенную фазу входящего pipe-процесса с помощью дочерних элементов `<handler>`, имеющих атрибуты `name` (идентификатор обработчика) и `class` (класс расширения `org.apache.axis2.handlers.AbstractHandler`, представляющий Handler-обработчик), а также дочерние элементы `<order>`. Элемент `<order>` указывает фазу включения Handler-обработчика с помощью следующих атрибутов:
  - `phase` — идентификатор фазы;
  - `after` — идентификатор обработчика фазы, перед которым производится включение;
  - `before` — идентификатор обработчика, после которого производится включение;
  - `phaseFirst` — если `true`, тогда данный Handler-обработчик является первым обработчиком фазы;
  - `phaseLast` — если `true`, тогда данный Handler-обработчик является последним обработчиком фазы;
- элемент `<OutFlow>` определяет включение Handler-обработчиков в определенную фазу исходящего pipe-процесса аналогично элементу `InFlow`;
- элемент `<InFaultFlow>` определяет включение Handler-обработчиков в определенную фазу входящего pipe-процесса обработки ошибки аналогично элементу `<InFlow>`;
- элемент `<OutFaultFlow>` определяет включение Handler-обработчиков в определенную фазу исходящего pipe-процесса обработки ошибки аналогично элементу `<InFlow>`;

- элемент `<parameter>` определяет свойства конфигурации модуля, используя атрибуты `name` и `locked`;
- элемент `<operation>` добавляет операцию сервису, для которого модуль активируется. Элемент `<operation>` определяет операцию сервиса аналогично элементу `<operation>` дескриптора services.xml. Добавление операции сервису актуально, например, в случае реализации WS-RM для определения операции `createSequence`.

Платформа Axis2 по умолчанию определяет развертывание Axis2-модулей ping, metadataExchange, addressing, script, soapmonitor, mtompolicy, находящихся в папке repository\modules или папке WEB-INF\modules каталога платформы.

Для развертывания Axis2-модулей, обеспечивающих поддержку WS-RM, WS-Security и WS-SecureConversation, необходимо скачать дистрибутивы модулей Sandesha2 и Rampart, воспользовавшись адресом **http://axis.apache.org/axis2/java/core/modules/index.html**.

Далее необходимо скопировать содержимое папки lib дистрибутива Rampart в папку lib или папку WEB-INF\lib каталога платформы Axis2, а содержимое папки modules дистрибутива Rampart — в папку repository\modules или папку WEB-INF\modules каталога платформы Axis2.

MAR-архив дистрибутива Sandesha2 необходимо скопировать в папку repository\modules или папку WEB-INF\modules каталога платформы Axis2, а библиотечные JAR-файлы дистрибутива Sandesha2 — в папку lib или папку WEB-INF\lib каталога платформы Axis2.

Теперь при запуске платформы Axis2 помимо модулей по умолчанию станут доступны модули:

- sandesha2, который обеспечивает поддержку спецификации WSRM 1.1 для платформы Axis2;
- rahas, добавляющий операцию RequestSecurityToken сервису, для которого активирован модуль;
- rampart, обеспечивающий поддержку спецификаций WS-Security и WS-SecureConversation для платформы Axis2 на основе реализаций Apache WSS4J, Apache XML-Security и Apache Rahas.

Axis2-модули имеют два состояния — Axis2-модуль может быть *развернут* (доступен) и *активирован*.

Axis2-модуль может быть активирован на различных уровнях:

- на глобальном уровне — обработчики модуля добавляются в соответствующие pipe-фазы, и сообщения всех развернутых сервисов Axis2-системы будут подвергаться обработке модулем на соответствующей ему pipe-фазе;
- на уровне группы сервисов — обработчики модуля добавляются в соответствующие pipe-фазы, и сообщения для определенной группы развернутых сервисов Axis2-системы будут подвергаться обработке модулем на соответствующей ему pipe-фазе;

- на уровне сервиса — обработчики модуля добавляются в соответствующие pipe-фазы, и сообщения для определенного развернутого сервиса Axis2-системы будут подвергаться обработке модулем на соответствующей ему pipe-фазе;
- на уровне операции сервиса — обработчики модуля добавляются в соответствующие pipe-фазы, и сообщения для определенной операции развернутого сервиса Axis2-системы будут подвергаться обработке модулем на соответствующей ему pipe-фазе.

Активация Axis2-модуля может быть произведена двумя способами — статически с помощью дескрипторов развертывания axis2.xml и services.xml и динамически, используя ссылку **Administration** Web-страницы Axis2-консоли администрирования (**http://localhost:8080/axis2/**).

Дескриптор axis2.xml активирует Axis2-модуль на глобальном уровне с помощью элемента `<module>`, а дескриптор services.xml позволяет активировать Axis2-модуль на уровне группы сервисов, отдельного сервиса и отдельной операции сервиса с помощью соответствующих элементов `<module>`.

Web-консоль Axis2 содержит административный модуль (по умолчанию для входа логин admin, пароль axis2) со следующими опциями:

- **System Components | Available Modules** — просмотр списка развернутых модулей;
- **System Components | Globally Engaged Modules** — просмотр модулей, активированных на глобальном уровне;
- **Engage Module | For all Services**, **Engage Module | For a Service Group**, **Engage Module | For a Service**, **Engage Module | For an Operation** — позволяют активировать модуль на глобальном уровне, на уровне группы сервисов, на уровне сервиса и на уровне операции сервиса.

Модель программирования Axis2 Web-сервисов

Платформа Axis2 поддерживает разработку трех видов Web-сервисов и клиентов Web-сервисов на основе различных моделей программирования:

- JAX-WS-модель программирования;
- POJO-модель программирования;
- Spring-модель программирования.

В этом состоит отличие платформы Axis2 от платформы CXF. В CXF поддержка спецификации JAX-WS строится на основе POJO-реализации Simple Frontend, при этом как реализация JAX-WS, так и реализация Simple Frontend тесно интегрированы с платформой Spring. В Axis2 реализация JAX-WS обеспечивается отдельным модулем, а развертывание POJO Web-сервисов и JAX-WS Web-сервисов по умолчанию базируется на модели, отличной от Spring.

Платформа Axis2 обеспечивает полную поддержку модели программирования JAX-WS+JAXB+SAAJ как для Web-сервисов, так и для клиентов Web-сервисов, описанную в *разд. "JAX-WS" главы 3*, с небольшими дополнениями. Для конфигурирования контекста клиентского запроса платформа Axis2 определяет свойство org.apache.axis2.jaxws.use.async.mep, включающее асинхронный вызов Web-сервиса.

Развертывание JAX-WS Web-сервисов осуществляется с помощью создания JAR-файла Web-сервиса и размещения его в каталоге servicejars платформы Axis2.

Наиболее простой и удобный способ создания Web-сервисов на платформе Axis2 — это использование модели программирования POJO, не требующей применения JAX-WS-аннотаций. Развертывание POJO Web-сервисов осуществляется с помощью конфигурационного файла services.xml или размещением класс-файла Web-сервиса в каталоге pojo платформы Axis2. Для управления жизненным циклом класс Web-сервиса может реализовывать интерфейс `org.apache.axis2.engine.ServiceLifeCycle`, предоставляющий методы `startUp()` и `shutDown()`, вызываемые средой выполнения при развертывании и деактивации Web-сервиса. Далее будет рассматриваться именно POJO-модель программирования как наиболее простая и удобная модель создания Web-сервисов и клиентов Web-сервисов.

Модель программирования Spring POJO Web-сервиса требует, помимо самого Web-сервиса, создания сервиса, реализующего интерфейс `org.apache.axis2.engine.ServiceLifeCycle` и отвечающего за инициализацию в методе `startUp()` контекста приложения Spring, основанного на конфигурационном Spring-файле applicationContext.xml:

```
org.springframework.context.support.ClassPathXmlApplicationContext appCtx =
new org.springframework.context.support.ClassPathXmlApplicationContext
(new String[]{"applicationContext.xml"}, false);
```

Соответственно библиотечные файлы платформы Spring должны быть добавлены в путь приложения Web-сервиса, а сервис инициализации Spring-контекста должен быть определен в конфигурационном файле services.xml:

```
<serviceGroup>
  <service name="[идентификатор сервиса инициализации Spring-контекста]"
class="[имя класса сервиса инициализации Spring-контекста]">
    <parameter name="ServiceClass">
       [имя класса сервиса инициализации Spring-контекста]
    </parameter>
    <parameter name="ServiceTCCL">composite</parameter>
    <parameter name="load-on-startup">true</parameter>
. . .
  </service>
  <service name="[идентификатор Web-сервиса]">
    <parameter name="ServiceClass">
        [имя класса Web-сервиса]
    </parameter>
```

```
<parameter name="ServiceObjectSupplier">
org.apache.axis2.extensions.spring.receivers.SpringAppContextAwareObjectSupplier
</parameter>
    <parameter name="SpringBeanName">
        [идентификатор Bean-компонента Web-сервиса]
    </parameter>
    <messageReceivers>
      <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
      class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
      <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
      class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>
  </service>
</serviceGroup>
```

Конфигурационный Spring-файл applicationContext.xml должен определять Bean-компонент Web-сервиса:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="applicationContext" class=
"org.apache.axis2.extensions.spring.receivers.ApplicationContextHolder" />
  <bean id="[идентификатор Bean-компонента Web-сервиса]"
        class="[имя класса Web-сервиса]">
. . .
  </bean>
</beans>
```

Axis2 XML-модель AXIOM

В основе системы сериализации/десериализации сообщений стеков Metro и CXF лежит технология Streaming API for XML (StAX). Не исключение и стек Axis2. Частью проекта Apache Axis2 является проект Apache Axiom (**http://ws.apache.org/axiom/**).

Среда выполнения Axis2 обрабатывает SOAP-сообщения, используя модель AXIs Object Model (AXIOM), базирующуюся на технологии StAX. При разборе SOAP-сообщения среда выполнения Axis2 создает в памяти объектную AXIOM-модель не сразу и полностью всего XML-документа, представляющего сообщение, а с приращением и по требованию. В этом основное отличие объектной AXIOM-модели от модели Document Object Model (DOM), которая основывается на изначальном построении в памяти дерева объектов документа. Применение технологий, использующих StAX-парсер, значительно улучшает производительность анализа XML-документов и снижает требуемые для этого ресурсы.

Помимо создания объектной модели документа на основе StAX-парсера технология AXIOM обеспечивает полную совместимость объектной модели документа с XML Infoset, поддержку SOAP-протокола и технологии XOP/MTOM.

Архитектура системы AXIOM состоит из следующих основных компонентов:

- программного интерфейса OM API, представленного пакетом `org.apache.axiom.om`;
- программного интерфейса Builder API, представленного пакетом `org.apache.axiom.om.impl.builder`;
- программного интерфейса SOAP API, представленного пакетом `org.apache.axiom.soap`;
- программного интерфейса SOAP Builder API, представленного пакетом `org.apache.axiom.soap.impl.builder`.

Программные интерфейсы OM API и Builder API обеспечивают создание и сериализацию, десериализацию и разбор XML-документов, а программные интерфейсы SOAP API и SOAP Builder API позволяют работать с документами, соответствующими SOAP-протоколу.

Для формирования XML-документа с помощью системы AXIOM сначала необходимо создать объект фабрики `org.apache.axiom.om.OMFactory` элементов XML-документа. Объект `OMFactory` можно создать статическим методом `org.apache.axiom.om.OMAbstractFactory.getOMFactory()`.

После создания объекта `OMFactory` его методы `create()` позволяют создавать объекты `OMDocument`, `OMElement`, `OMAttribute` и т. д. AXIOM-модели документа.

Интерфейсы пакета `org.apache.axiom.om`, представляющие объектную AXIOM-модель, имеют общие суперинтерфейсы `OMContainer` и `OMSerializable`, методы которых `serialize()` (сериализация с кэшированием) и `serializeAndConsume()` (сериализация без кэширования) обеспечивают сериализацию Java-объектов в XML-данные на основе объектов `java.io.OutputStream`, `java.io.Writer` и `javax.xml.stream.XMLStreamWriter`.

Для десериализации и разбора XML-документа сначала необходимо создать объект `org.apache.axiom.om.OMXMLParserWrapper`, отвечающий за создание объектной AXIOM-модели документа. Объект `OMXMLParserWrapper` можно создать с помощью статических методов `createOMBuilder()` класса-фабрики `org.apache.axiom.om.OMXMLBuilderFactory`, основанных на объектах `java.io.InputStream`, `java.io.Reader` и `javax.xml.stream.XMLStreamReader`.

После создания объекта `OMXMLParserWrapper` его методы `get()` позволяют получить объекты `OMDocument` и `OMElement` AXIOM-модели документа.

Для работы с SOAP-сообщениями вместо объекта `OMFactory` используется объект `org.apache.axiom.soap.SOAPFactory`, создаваемый статическим методом `getSOAP11Factory()` или `getSOAP12Factory()` класса `OMAbstractFactory`. Методы `create()` интерфейса `SOAPFactory` обеспечивают создание объектов `SOAPMessage`, `SOAPEnvelope`, `SOAPHeader`, `SOAPBody` и `SOAPFault` AXIOM-модели SOAP-сообщения.

Интерфейсы пакета `org.apache.axiom.soap`, представляющие объектную AXIOM-модель SOAP-сообщения, также имеют общие суперинтерфейсы `OMContainer` и `OMSerializable`, методы `serialize()` (сериализация с кэшированием) и `serializeAndConsume()` (сериализация без кэширования) которых обеспечивают сериализацию Java-объектов в SOAP-сообщение на основе объектов `java.io.OutputStream`, `java.io.Writer` и `javax.xml.stream.XMLStreamWriter`.

Для десериализации и разбора SOAP-сообщения необходимо использовать класс `org.apache.axiom.soap.impl.builder.StAXSOAPModelBuilder` или класс `org.apache.axiom.soap.impl.builder.MTOMStAXSOAPModelBuilder`, экземпляры которых создаются с помощью конструктора класса на основе объекта `javax.xml.stream.XMLStreamReader` и методы `get()` которых позволяют получить объекты AXIOM-модели SOAP-сообщения.

В качестве примера использования системы AXIOM рассмотрим создание Web-сервиса, применяющего низкоуровневую обработку SOAP-сообщений на основе AXIOM-модели.

1. Откроем среду Eclipse, в меню **File** выберем пункты **New | Dynamic Web Project** и введем имя проекта AXIOMWebService. Нажав кнопку **New Runtime**, выберем **Apache Tomcat v7.0**, в раскрывающемся списке **Dynamic web module version** выберем **2.5** и нажмем кнопку **Finish**.
2. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Axis2 Preferences**. На вкладке **Axis2 Runtime**, нажав кнопку **Browse**, определим каталог предварительно инсталлированной среды выполнения Apache Axis2 (Binary Distribution, **http://axis.apache.org/axis2/java/core/download.cgi**).
3. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Server and Runtime**. В раскрывающемся списке **Server runtime** выберем **Tomcat v7.0 Server**, а в раскрывающемся списке **Web service runtime** — выберем **Apache Axis2** и нажмем кнопку **OK**.
4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **AXIOMWebService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Axis2 Web Services** и нажмем кнопку **OK**.

   В результате среда выполнения Axis2 будет интегрирована в проект AXIOMWebService как Web-приложение сервера Tomcat.
5. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **AXIOMWebService** и выберем пункты **New | Class**, введем имя Web-сервиса AXIOMService и имя пакета `axiomservice`, нажмем кнопку **Finish**.
6. Дополним код класса `AXIOMService`, как показано в листинге 6.1.

Листинг 6.1. Код класса `AXIOMService`

```
package axiomservice;
import org.apache.axiom.om.*;
public class AXIOMService {
```

```
public OMElement getHello(OMElement root) throws Exception {
  System.out.println(root.toStringWithConsume());
  String name;
  OMElement element = root.getFirstElement();
  name=element.getText();
  System.out.println("Name"+" "+name);
  OMFactory factory = OMAbstractFactory.getOMFactory();
  OMNamespace ns1 =
    factory.createOMNamespace("http://axiomservice", "ns1");
  OMNamespace s1 = factory.createOMNamespace
    ("http://www.w3.org/2001/XMLSchema-instance", "s1");
  OMNamespace s2 =
    factory.createOMNamespace("http://www.w3.org/2001/XMLSchema", "s2");
  OMElement rootresponse =
    factory.createOMElement("getHelloResponse", ns1);
  OMElement returnresponse = factory.createOMElement("return", ns1);
  returnresponse.declareNamespace(s1);
  returnresponse.declareNamespace(s2);
  OMAttribute type = factory.createOMAttribute("type",s1,"s2:string");
  returnresponse.addAttribute(type);
  returnresponse.setText("Hello"+" "+name);
  rootresponse.addChild(returnresponse);
  System.out.println(rootresponse.toStringWithConsume());
  return rootresponse;
}}
```

Web-сервис AXIOMService имеет единственный метод `getHello()`, который получает в качестве аргумента XML-данные тела SOAP-сообщения запроса и возвращает XML-данные тела SOAP-сообщения ответа.

Метод `getHello()` предварительно извлекает из входящего SOAP-сообщения аргумент запроса и затем на его основе создает ответное SOAP-сообщение.

7. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **AXIOMWebService** и выберем пункты **New | Other | Web Services | Web Service**, нажмем кнопку **Next** и в строке **Service implementation** введем `axiomservice.AXIOMService`.
8. Установим бегунки диалогового окна **Web Service** в положение **Start service** и **Assemble client** (рис. 6.5) и нажмем кнопку **Finish**.

В результате среда Eclipse добавит в папку services среды выполнения Axis2 класс-файл Web-сервиса AXIOMService и его сгенерированный дескриптор развертывания services.xml и сгенерирует Web-приложение AXIOMWebServiceClient с основой клиентского кода Web-сервиса AXIOMService.

Дескриптор services.xml Web-сервиса AXIOMService указывает в качестве Axis2 Handler-обработчика последней фазы обработки входящего сообщения класс `org.apache.axis2.rpc.receivers.RPCMessageReceiver`, оперирующий экземплярами

JavaBean-классов и передающий в метод Web-сервиса значения его параметров как простые Java-типы. В рассматриваемом случае параметром метода Web-сервиса служат XML-данные клиентского SOAP-сообщения, поэтому в качестве Handler-обработчика последней фазы, в данном случае, должен быть определен класс `org.apache.axis2.receivers.RawXMLINOutMessageReceiver`, оперирующий объектами `OMElement`.

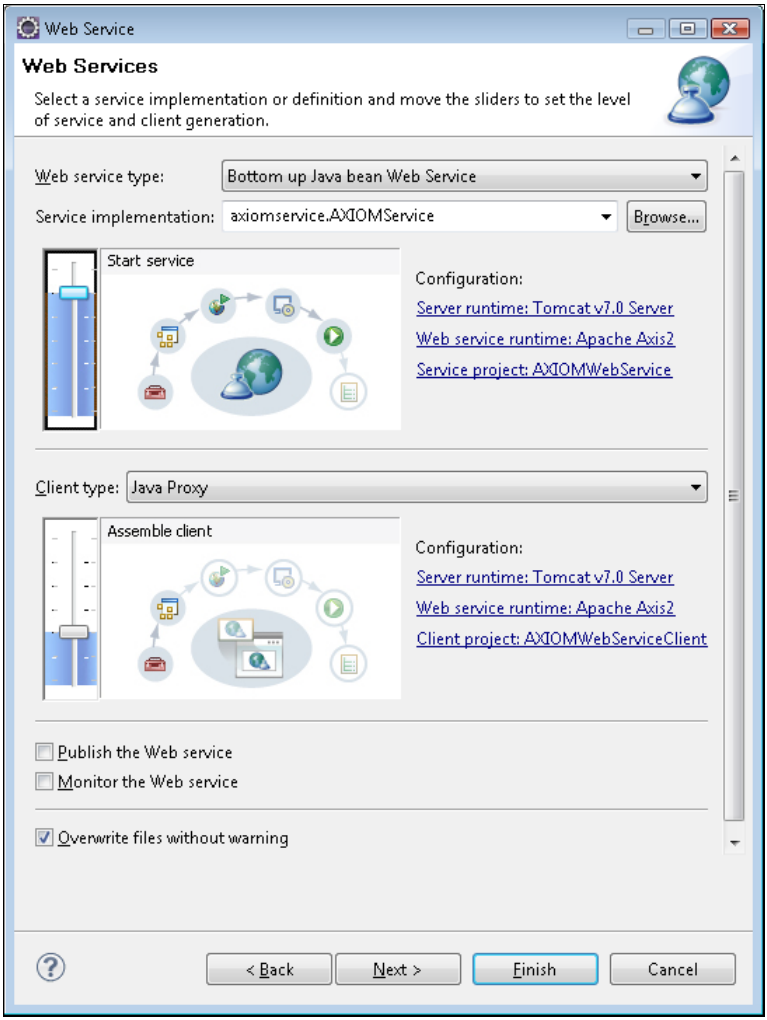


Рис. 6.5. Окно создания Web-сервиса и клиента Web-сервиса

1. Изменим дескриптор services.xml Web-сервиса AXIOMService следующим образом:

```
<service name="AXIOMService" >
<operation name="getHello">
<messageReceiver
```

```
  class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
        <actionMapping>urn:getHello</actionMapping>
</operation>
<parameter name="ServiceClass"
           locked="false">axiomservice.AXIOMService</parameter>
</service>
```

2. Подключим к Web-сервису AXIOMService Axis2-модуль SOAP-монитора для того, чтобы видеть входящие и исходящие сообщения Web-сервиса.
3. В окне **Project Explorer** среды Eclipse откроем узел **WebContent | WEB-INF | conf** проекта AXIOMWebService и в окне редактора добавим в дескриптор axis2.xml активацию `<module ref="soapmonitor"/>` Axis2-модуля SOAP-монитора на глобальном уровне.
4. Добавим в дескриптор web.xml узла **WebContent | WEB-INF** проекта AXIOMWebService определение сервлета `org.apache.axis2.soapmonitor.servlet.SOAPMonitorService`, отвечающего за активацию апплета SOAP-монитора:

```
<servlet>
  <servlet-name>SOAPMonitorService</servlet-name>
  <servlet-class>
    org.apache.axis2.soapmonitor.servlet.SOAPMonitorService
  </servlet-class>
  <init-param>
    <param-name>SOAPMonitorPort</param-name>
    <param-value>5001</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>SOAPMonitorService</servlet-name>
  <url-pattern>/SOAPMonitor</url-pattern>
</servlet-mapping>
```

5. Извлечем из Axis2-библиотеки axis2-soapmonitor-servlet.jar папку org\apache\axis2\soapmonitor\applet и добавим ее в узел **WebContent** проекта AXIOMWebService.
6. Теперь после запуска Web-приложения AXIOMWebService выбором пунктов **Run as | Run on Server | Finish** адрес **http://localhost:8080/AXIOMWebService/SOAPMonitor** дает доступ к SOAP-монитору Web-сервиса AXIOMService.
7. Для создания клиента Web-сервиса AXIOMService в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **AXIOMWebServiceClient | Java Resources: src | axiomservice** и выберем пункты **New | Class**, введем имя класса `AXIOMClient` и нажмем кнопку **Finish**.
8. Изменим код класса `AXIOMClient`, как показано в листинге 6.2.

Листинг 6.2. Код класса `AXIOMClient`

```
package axiomservice;
public class AXIOMClient {
  public static void main(String[] args) {
    AXIOMServiceStub.GetHello getHello=new AXIOMServiceStub.GetHello();
    getHello.setRoot("User");
    AXIOMServiceStub.GetHelloResponse res=
      new AXIOMServiceStub.GetHelloResponse();
    try {
      AXIOMServiceStub stub=new AXIOMServiceStub();
      res=stub.getHello(getHello);
      System.out.println(res.get_return());
    } catch (Exception e) {
      System.out.println(e.getMessage());
}}}
```

В методе `main()` класса `AXIOMClient` создается экземпляр клиентской заглушки `AXIOMServiceStub`, при этом устанавливается значение параметра запроса Web-сервису, представленного полем JavaBean-класса `AXIOMServiceStub.GetHello`. После вызова Web-сервиса экземпляр JavaBean-класса `AXIOMServiceStub.GetHelloResponse`, представляющий XML-данные SOAP-сообщения Web-сервиса, дает возможность извлечь строку ответа Web-сервиса.

После запуска класса `AXIOMClient` как Java-приложения в консоль среды Eclipse будет выведена строка ответа Web-сервиса, а на странице SOAP-монитора будут отображены входящее и исходящее SOAP-сообщения Web-сервиса.

На компакт-диске

Проекты AXIOMWebService и AXIOMWebServiceClient находятся в папке Примеры\Глава6\AXIOM компакт-диска.

Client API

В *разд. "Пример создания JAX-WS Web-сервиса и JAX-WS-клиента" главы 3* рассматривалось создание клиента Axis2 Web-сервиса с использованием среды Eclipse. В том примере клиент Web-сервиса создавался для конечной точки, поддерживающей протокол SOAP1.2/HTTP обмена сообщениями, при этом среда Eclipse генерировала артефакты в виде классов `Axis2ExampleStub` и `Axis2ExampleCallbackHandler`.

Класс `Axis2ExampleStub` расширяет класс `org.apache.axis2.client.Stub`, представляющий клиентскую заглушку. Класс `Axis2ExampleStub` имеет конструктор, в котором инициализируются объекты `org.apache.axis2.description.AxisService` и `org.apache.axis2.client.ServiceClient`, представленные полями `_service` и `_serviceClient` класса `Stub` соответственно, а также устанавливается адрес Soap12-конечной точки Web-сервиса Axis2Example.

Объект `AxisService` представляет конфигурацию Web-сервиса, и в конструкторе класса `Axis2ExampleStub` для `AxisService`-объекта `_service` устанавливается имя вызываемого Web-сервиса и определяется вызываемая операция Web-сервиса. Объект `ServiceClient` связан с объектом `AxisService` и обеспечивает вызов Web-сервиса. В конструкторе класса `Axis2ExampleStub` для `ServiceClient`-объекта `_serviceClient` с помощью объекта `org.apache.axis2.client.Options`, предназначенного для конфигурирования взаимодействия с Web-сервисом и получаемого методом `getOptions()` класса `ServiceClient`, устанавливаются адрес конечной точки и версия протокола SOAP.

Метод `getHello()` класса `Axis2ExampleStub` обеспечивает вызов одноименной операции Web-сервиса. Этот метод возвращает экземпляр JavaBean-класса `axis2sexample.Axis2ExampleStub.GetHelloResponse`, представляющего XML-данные тела ответного SOAP-сообщения, а его параметром служит экземпляр JavaBean-класса `axis2sexample.Axis2ExampleStub.GetHello`, представляющий XML-данные тела SOAP-сообщения запроса Web-сервису.

В методе `getHello()` класса `Axis2ExampleStub` с помощью `ServiceClient`-объекта `_serviceClient` создается экземпляр класса `org.apache.axis2.client.OperationClient`, обеспечивающий конфигурирование взаимодействия с Web-сервисом на уровне его операций с помощью объекта `Options`. Далее создается экземпляр класса `org.apache.axis2.context.MessageContext`, определяющий конфигурацию сообщений, устанавливается MessageContext-конфигурация элемента Envelope SOAP-сообщения запроса, объект `MessageContext` добавляется в объект `OperationClient` и производится вызов Web-сервиса методом `execute()` класса `OperationClient`. Метод `getMessageContext()` класса `OperationClient` позволяет извлечь MessageContext-контекст ответного SOAP-сообщения, а метод `getEnvelope()` класса `MessageContext` — получить объект `org.apache.axiom.soap.SOAPEnvelope`, представляющий элемент Envelope SOAP-сообщения. Методы `getBody()` и `getFirstElement()` класса `SOAPEnvelope` дают возможность получить корневой элемент XML-данных тела SOAP-сообщения, а метод `fromOM()` класса `Axis2ExampleStub` обеспечивает преобразование XML-данных тела SOAP-сообщения в экземпляр JavaBean-класса и соответственно метод `getHello()` класса `Axis2ExampleStub` возвращает, таким образом, созданный `GetHelloResponse`-объект.

JavaBean-класс `GetHello` представляет XML-данные тела SOAP-запроса Web-сервису, и его метод `setName()` позволяет установить параметр операции Web-сервиса. Метод `get_return()` JavaBean-класса `GetHelloResponse`, представляющего XML-данные тела ответного SOAP-сообщения, дает возможность получить строку, возвращаемую методом Web-сервиса Axis2Example. На основе методов `getHello()` класса `Axis2ExampleStub`, `setName()` класса `GetHello` и `get_return()` класса `GetHelloResponse` строится метод `main()` класса `Axis2Client`, отвечающего за вызов Web-сервиса в клиентском Java-приложении.

Так как при создании клиента Web-сервиса Axis2Example указывалась опция **Generate a client with supports both synchronous asynchronous invocation**, класс `Axis2ExampleStub` обеспечивает, помимо синхронного вызова Web-сервиса, его асинхронный вызов с помощью метода `startgetHello()`.

Метод `startgetHello()` класса `Axis2ExampleStub` принимает в качестве аргументов экземпляр класса `GetHello` и экземпляр класса, расширяющего абстрактный класс `Axis2ExampleCallbackHandler`, который сгенерирован средой Eclipse для организации обработки `GetHelloResponse`-объекта. В методе `startgetHello()` класса `Axis2ExampleStub` также с помощью `ServiceClient`-объекта `_serviceClient` создается экземпляр класса `OperationClient`, экземпляр класса `MessageContext`, устанавливается MessageContext-конфигурация элемента Envelope SOAP-сообщения запроса, объект `MessageContext` добавляется в объект `OperationClient`. Отличие асинхронного метода `startgetHello()` от синхронного метода `getHello()` заключается в том, что в методе `startgetHello()` для `OperationClient`-объекта методом `setCallback()` определяется объект `org.apache.axis2.client.async.AxisCallback`, отвечающий за асинхронное получение ответного сообщения с помощью метода `onMessage()`. Метод `onMessage()` интерфейса `AxisCallback` принимает в качестве аргумента `MessageContext`-объект ответного сообщения, из которого возможно извлечение `SOAPEnvelope`-объекта с последующим преобразованием XML-данных тела SOAP-сообщения в экземпляр JavaBean-класса `GetHelloResponse`. Кроме того, в синхронном методе `getHello()` вызов Web-сервиса производится методом `execute(true)` `OperationClient`-объекта, а в асинхронном методе `startgetHello()` вызов Web-сервиса осуществляется методом `execute(true)` `OperationClient`-объекта. Аргумент метода `execute(boolean block)` класса `OperationClient` задает синхронность или асинхронность выполнения MEP-операции.

Для организации асинхронного вызова Web-сервиса Axis2Example в метод `main()` класса `Axis2Client` включим следующий код:

```
Axis2ExampleCallbackHandler handler=new Axis2ExampleCallbackHandler(){
  public void receiveResultgetHello
    (axis2sexample.Axis2ExampleStub.GetHelloResponse result) {
    System.out.println("Асинхронный вызов"+" "+result.get_return());
}};
stub.startgetHello(getHello, handler);
Thread.sleep(1000);
```

На компакт-диске

Пример синхронного и асинхронного клиента SOAP12-конечной точки Web-сервиса находится в папке Примеры\Глава6\SOAP12 компакт-диска как Eclipse-проект Axis2_WebServiceClientSoap12.

1. Для создания Soap11-клиента Axis2 Web-сервиса в меню **File** среды Eclipse выберем пункты **New | Dynamic Web Project** и введем имя проекта Axis2_WebServiceClientSoap11. Нажав кнопкой **New Runtime** выберем **Apache Tomcat v7.0**, в раскрывающемся списке **Dynamic web module version** выберем значение **2.5** и нажмем кнопку **Finish**.
2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClientSoap11** и выберем пункт **Properties**. В появившемся диалоговом окне выберем вариант **Project Facets**, укажем **Axis2 Web Services** и нажмем кнопку **OK**.

3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClientSoap11** и выберем пункты **New | Other | Web Services | Web Service Client**, нажмем кнопку **Next**, в строке **Service definition** введем `http://localhost:8080/Axis2_WebService/services/Axis2Example?wsdl` и нажмем кнопку **Next**. В раскрывающемся списке **Port Name** выберем конечную точку **Axis2ExampleHttpSoap11Endpoint** и нажмем кнопку **Finish**.

   В результате средой Eclipse в пакете `axis2sexample` будут сгенерированы классы `Axis2ExampleStub` и `Axis2ExampleCallbackHandler`. Класс `Axis2ExampleStub` проекта Axis2_WebServiceClientSoap11 отличается от класса `Axis2ExampleStub` проекта Axis2_WebServiceClientSoap12 адресом конечной точки Web-сервиса — **http://localhost:8080/Axis2_WebService/services/Axis2Example.Axis2ExampleHttpSoap11Endpoint/**.

4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClientSoap11 | Java Resources: src | axis2sexample** и выберем пункты **New | Class**, введем имя класса `Axis2Client` и нажмем кнопку **Finish**.

5. Возьмем код класса `Axis2Client` из проекта Axis2_WebServiceClientSoap12.

6. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2_WebServiceClientSoap11 | Java Resources: src | axis2sexample | Axis2Client** и выберем **Run As | Java Application**. В результате в окно **Console** среды Eclipse будет выведена ответная строка предварительно развернутого Web-сервиса Axis2Example.

При подключенном модуле SOAP-монитора к Web-сервису Axis2Example на странице браузера по адресу **http://localhost:8080/Axis2_WebService/SOAPMonitor** можно видеть, что в случае вызова Web-сервиса Soap11-клиентом и Soap12-клиентом SOAP-сообщения отличаются пространством имен SOAP-элементов в соответствии с протоколами SOAP 1.1 и SOAP 1.2.

НА КОМПАКТ-ДИСКЕ

Пример клиента SOAP11-конечной точки Web-сервиса находится в папке Примеры\Глава6\SOAP11 компакт-диска как Eclipse-проект Axis2_WebServiceClientSoap11.

Рассмотрим создание клиента Web-сервиса без использования сгенерированных средой Eclipse артефактов.

1. Для создания клиента Web-сервиса в перспективе Java среды Eclipse в меню **File** выберем пункты **New | Java Project**, введем имя проекта Axis2ServiceClient и нажмем кнопку **Finish**.

2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2ServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module 2.5**, **Axis2 Web Services** и нажмем кнопку **OK**.

3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2ServiceClient** и выберем пункты **New | Class**, введем имя класса

`Axis2Client` и имя пакета `axis2client`, укажем опцию **public static void main(String[] args)** и нажмем кнопку **Finish**.

4. Изменим код метода `main()` класса `Axis2Client`, как показано в листинге 6.3.

Листинг 6.3. Код метода `main()` класса `Axis2Client` проекта Axis2ServiceClient

```
public static void main(String[] args) throws Exception {
  ServiceClient sc = new ServiceClient();
  OMFactory factory = OMAbstractFactory.getOMFactory();
  OMNamespace ns1 = factory.createOMNamespace
    ("http://axis2sexample", "ns1");
  OMElement root = factory.createOMElement("getHello", ns1);
  OMElement name = factory.createOMElement("name", ns1);
  name.setText("User");
  root.addChild(name);
  System.out.println(root.toString());
  Options op = new Options();
  op.setTo(new org.apache.axis2.addressing.EndpointReference
      ("http://127.0.0.1:1234/Axis2_WebService/services/Axis2Example"));
  op.setAction("urn:getHello");
  sc.setOptions(op);
  sc.engageModule(org.apache.axis2.Constants.MODULE_ADDRESSING);
  //синхронный вызов
  OMElement res = sc.sendReceive(root);
  System.out.println("Синхронный вызов "+res.getFirstElement().getText());
  //асинхронный вызов
  org.apache.axis2.client.async.AxisCallback callback =
    new org.apache.axis2.client.async.AxisCallback() {
    public void onMessage
      (org.apache.axis2.context.MessageContext msgContext) {
      System.out.println("Асинхронный вызов " +
        msgContext.getEnvelope().getBody().getFirstElement().
        getFirstElement().getText());}
    public void onFault
      (org.apache.axis2.context.MessageContext msgContext) { }
    public void onError(Exception e) {}
    public void onComplete() {}};
  sc.sendReceiveNonBlocking(root, callback);
  Thread.sleep(1000);
  //In-Only MEP
  sc.fireAndForget(root);
  //Robust In-Only MEP
  sc.sendRobust(root);
}
```

В методе `main()` клиента Web-сервиса Axis2Example с помощью OM API сначала создаются XML-данные SOAP-сообщения запроса Web-сервису. Далее создается

объект `org.apache.axis2.client.Options`, определяющий конфигурацию предварительно созданного объекта `org.apache.axis2.client.ServiceClient`, который даст возможность вызова Web-сервиса. Методами `Options`-объекта устанавливается адрес конечной точки Web-сервиса и `Action`-заголовок запроса, обеспечивающий связывание запроса с операцией Web-сервиса. После конфигурирования `Options`-объекта, он связывается с `ServiceClient`-объектом методом `setOptions`.

В данном примере используется инструмент TCPMon для просмотра исходящих и входящих сообщений Web-сервиса. Для использования монитора TCPMon необходимо предварительно скачать и инсталлировать его дистрибутив (**http://ws.apache.org/commons/tcpmon/download.cgi**). Пакетный файл tcpmon.bat папки build каталога дистрибутива позволяет запустить TCPMon-монитор. После запуска TCPMon-монитора на вкладке **Admin** окна **TCPMon** в поле **Listen Port #** наберем номер порта `1234`, а в поле **Target Port #** наберем номер порта `8080` и нажмем кнопку **Add** (рис. 6.6). Теперь вкладка **Port 1234** окна **TCPMon** даст возможность просматривать поступившие сообщения порта 1234 и исходящие сообщения порта 8080. Таким образом, TCPMon-монитор будет служить посредником между клиентом и Web-сервисом.

Так как в данном примере используется инструмент TCPMon для просмотра сообщений, в методе `main()` класса `Axis2Client` адрес конечной точки вызываемого Web-сервиса указывает порт 1234, а не порт 8080.

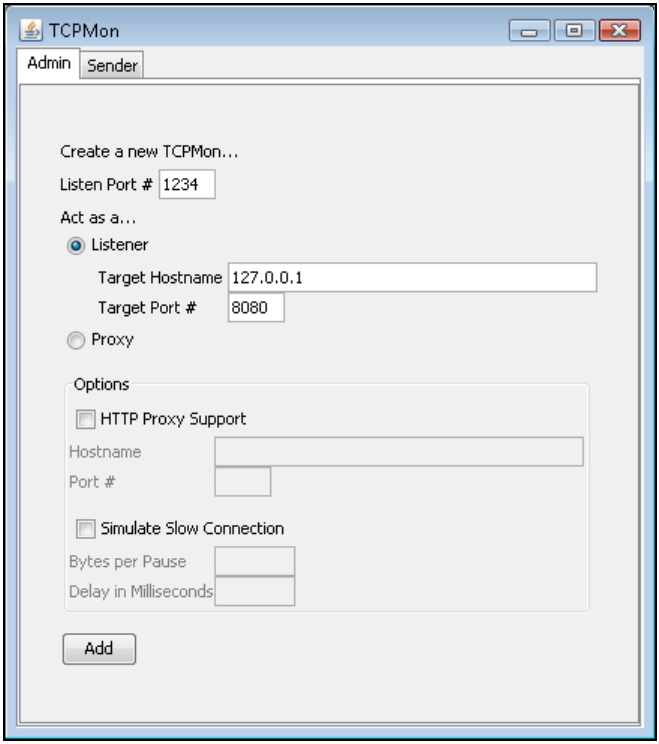


Рис. 6.6. Вкладка **Admin** окна администрирования монитора TCPMon

Для того чтобы клиентские SOAP-сообщения содержали заголовки WS-Addressing, на клиентской стороне необходимо подключить Axis2-модуль addressing.mar. Позволяет сделать это метод `engageModule()` класса `ServiceClient`. Также необходимо добавить файл модуля addressing.mar в путь клиентского приложения.

Чтобы добавить файл addressing.mar в путь приложения в окне **Project Explorer** среды Eclipse, щелкнем правой кнопкой мыши на узле **Axis2ServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Java Build Path**, перейдем на вкладку **Libraries** и, нажав кнопку **Add External JARs**, добавим файл addressing.mar папки modules платформы Axis2.

Платформа Axis2 на стороне Web-сервиса по умолчанию поддерживает WS-Addressing, однако среда выполнения Axis2 не будет добавлять WSA-заголовки в исходящее сообщение, если входящее сообщение Web-сервиса не будет содержать WSA-заголовков.

Добавляя или исключая строку `sc.engageModule(org.apache.axis2.Constants.MODULE_ADDRESSING);` метода `main()` класса `Axis2Client`, в окне TCPMon-монитора можно видеть, как добавляются или исключаются WSA-заголовки во входящем и исходящем SOAP-сообщениях Web-сервиса.

Метод `sendReceive()` класса `ServiceClient` обеспечивает синхронный вызов In-Out MEP-операции Web-сервиса. Метод `fireAndForget()` класса `ServiceClient` — вызов In-Only MEP-операции Web-сервиса, при этом в клиентское сообщение добавляется заголовок:

```
<wsa:ReplyTo>
<wsa:Address>http://www.w3.org/2005/08/addressing/none</wsa:Address>
</wsa:ReplyTo>
```

Метод `sendRobust()` класса `ServiceClient` обеспечивает вызов Robust In-Only MEP-операции Web-сервиса.

Для асинхронного вызова Web-сервиса можно использовать метод `public void sendReceiveNonBlocking(org.apache.axiom.om.OMElement elem, AxisCallback callback)` класса `ServiceClient`, где `AxisCallback`-объект обеспечивает метод `onMessage()`, вызываемый при поступлении ответного сообщения от Web-сервиса.

После запуска класса `Axis2Client` как Java-приложения в консоли среды Eclipse можно увидеть результат вызова Web-сервиса методами `sendReceive()` и `sendReceiveNonBlocking()`, а в окне TCPMon-монитора — обмен сообщениями при использовании методов `sendReceive()`, `sendReceiveNonBlocking()`, `fireAndForget()` и `sendRobust()`.

НА КОМПАКТ-ДИСКЕ

Проект Axis2ServiceClient находится в папке Примеры\Глава6\ServiceClient компакт-диска.

В рассмотренном примере исходящее и входящее сообщения клиента Web-сервиса передаются по HTTP-протоколу. Однако метод `setUseSeparateListener(true)` класса `org.apache.axis2.client.Options` позволяет разделить транспортные каналы для исходящего и входящего сообщений. После установки методом

`setTransportInProtocol(Constants.TRANSPORT_TCP)` класса `Options` протокол TCP для входящих сообщений и флага `true` для метода `setUseSeparateListener()` клиентские запросы будут передаваться по HTTP-протоколу, а ответы от Web-сервиса — по TCP-протоколу. При этом должна быть включена поддержка WS-Addressing.

1. Для создания клиента Web-сервиса на основе OperationClient API в перспективе Java среды Eclipse в меню **File** выберем пункты **New | Java Project**, введем имя проекта Axis2OperationClient и нажмем кнопку **Finish**.
2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2OperationClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module 2.5**, **Axis2 Web Services** и нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2OperationClient** и выберем пункты **New | Class**, введем имя класса `Axis2Client` и имя пакета `axis2client`, укажем опцию **public static void main(String[] args)** и нажмем кнопку **Finish**.
4. Изменим код метода `main()` класса `Axis2Client`, как показано в листинге 6.4.

Листинг 6.4. Код метода `main()` класса `Axis2Client` проекта Axis2OperationClient

```
public static void main(String[] args) throws Exception {
  ServiceClient sc = new ServiceClient();
  sc.engageModule(org.apache.axis2.Constants.MODULE_ADDRESSING);
  OperationClient opClient = sc.createClient(ServiceClient.ANON_OUT_IN_OP);
  org.apache.axis2.context.MessageContext outmc =
    new org.apache.axis2.context.MessageContext();
  Options op = outmc.getOptions();
  op.setTo(new
    org.apache.axis2.addressing.EndpointReference(
    "http://127.0.0.1:8080/Axis2_WebService/services/Axis2Example"));
  p.setAction("urn:getHello");
  SOAPFactory factory = OMAbstractFactory.getSOAP12Factory();
  SOAPEnvelope envelope = factory.getDefaultEnvelope();
  OMNamespace ns1 = factory.createOMNamespace("http://axis2sexample",
     "ns1");
  OMElement root = factory.createOMElement("getHello", ns1);
  OMElement name = factory.createOMElement("name", ns1);
  name.setText("User");
  root.addChild(name);
  envelope.getBody().addChild(root);
  outmc.setEnvelope(envelope);
  opClient.addMessageContext(outmc);
  opClient.execute(true);
  org.apache.axis2.context.MessageContext out =
    opClient.getMessageContext("Out");
```

```
System.out.println("OUT "+out.getEnvelope().toString());
org.apache.axis2.context.MessageContext inmc =
  opClient.getMessageContext("In");
SOAPEnvelope response = inmc.getEnvelope();
System.out.println("IN "+response);
System.out.println(response.getBody().getFirstElement().
  getFirstElement().getText());  }
```

В методе `main()` класса `Axis2Client` вызов Web-сервиса обеспечивает не объект `ServiceClient`, а объект `OperationClient`, создаваемый на основе `ServiceClient`-объекта для In-Out MEP-операции Web-сервиса.

В отличие от `ServiceClient`-объекта `OperationClient`-объект работает не на уровне XML-данных тела SOAP-сообщения, а на уровне контекста сообщения MessageContext, содержащего полностью SOAP-сообщение. Поэтому в данном случае для просмотра исходящих и входящих сообщений не нужен монитор, сообщения можно выводить в консоль.

В методе `main()` класса `Axis2Client` для конфигурирования контекста MessageContext исходящего SOAP-сообщения также используется `Options`-объект, а само исходящее SOAP-сообщение создается с помощью интерфейсов SOAP API и OM API, чтобы затем быть добавленным в исходящий MessageContext-контекст методом `setEnvelope()`. Исходящий MessageContext-контекст связывается с `OperationClient`-объектом методом `addMessageContext()`.

После вызова Web-сервиса исходящий и входящий MessageContext-контекст может быть получен методом `getMessageContext()` класса `OperationClient`, принимающим в качестве аргумента значение WSDL-атрибута `messageLabel` сообщения.

После запуска класса `Axis2Client` как Java-приложения в консоли среды Eclipse можно увидеть исходящее и входящее SOAP-сообщения клиента Web-сервиса и результат вызова Web-сервиса.

НА КОМПАКТ-ДИСКЕ

Проект Axis2OperationClient находится в папке Примеры\Глава6\OperationClient компакт-диска.

Применение ServiceClient API и OperationClient API подразумевает низкоуровневую обработку сообщений на основе Axiom API. Однако платформа Axis2 позволяет создавать и высокоуровневого клиента Web-сервиса без использования Axiom API. Обеспечивает создание такого клиента класс `org.apache.axis2.rpc.client.RPCServiceClient`, расширяющий класс `ServiceClient`.

НА КОМПАКТ-ДИСКЕ

Пример высокоуровневого клиента Web-сервиса находится в папке Примеры\Глава6\RPCClient компакт-диска как Eclipse-проект Axis2RPCClient.

Проект Axis2RPCClient содержит класс `axis2client.Axis2Client` с методом `main()`, код которого представлен в листинге 6.5.

Листинг 6.5. Код метода `main()` класса `Axis2Client` проекта Axis2RPCClient

```
public static void main(String[] args) throws Exception{
  org.apache.axis2.client.Options op =
    new org.apache.axis2.client.Options();
  op.setTo(new
    org.apache.axis2.addressing.EndpointReference(
    "http://localhost:8080/Axis2_WebService/services/Axis2Example"));
  op.setAction("urn:getHello");
  org.apache.axis2.rpc.client.RPCServiceClient client =
    new org.apache.axis2.rpc.client.RPCServiceClient();
  client.setOptions(op);
  javax.xml.namespace.QName getHello = new
  javax.xml.namespace.QName("http://axis2sexample", "getHello");
  Object[] arg = new Object[] {"User"};
  Object[] res  = client.invokeBlocking(getHello, arg,
    new Class[]{String.class});
  System.out.println(res[0]);  }
```

Из кода метода `main()` видно, что для создания клиента Web-сервиса на основе класса `RPCServiceClient` не требуется знание Axiom API.

Поддержка архитектуры REST

При развертывании Web-сервиса среда выполнения Axis2 автоматически создает набор конечных точек, поддерживающих протоколы SOAP 1.1, SOAP 1.2 и чистый HTTP. В зависимости от формата входящего сообщения Web-сервиса среда выполнения Axis2 автоматически производит соединение с подходящей конечной точкой, формирующей ответное сообщение, формат которого соответствует формату входящего сообщения.

Платформа Axis2 поддерживает чистый HTTP-протокол на основе реализации связывания WSDL 2.0 HTTP Binding, что дает возможность вызова Web-сервиса с помощью HTTP-запросов GET и POST.

Любой Web-сервис, развернутый на платформе Axis2, становится доступным для GET-запроса, состоящего из URL-адреса конечной точки, имени операции Web-сервиса и параметров вызываемой операции Web-сервиса, а также для POST-запроса, который может содержать чистые XML-данные.

Для включения поддержки REST на стороне клиента для `Options`-объекта необходимо установить соответствующее свойство:

```
Options options = new Options();
options.setProperty(Constants.Configuration.ENABLE_REST,
                    Constants.VALUE_TRUE);
```

Для RESTful Web-сервисов и клиентов RESTful Web-сервисов платформа Axis2 обеспечивает формат сообщений JSON. Для подключения поддержки JSON-

формата необходимо в конфигурационный файл axis2.xml включить следующие элементы:

- формат сообщений `application/json`:

```
<messageFormatters>
  <messageFormatter contentType="application/json"
    class="org.apache.axis2.json.JSONMessageFormatter"/>
  . . .
</messageFormatters>
<messageBuilders>
  <messageBuilder contentType="application/json"
    class="org.apache.axis2.json.JSONOMBuilder"/>
  . . .
</messageBuilders>
```

- формат сообщений `text/javascript`:

```
<messageFormatters>
  <messageFormatter contentType="text/javascript"
    class="org.apache.axis2.json.JSONBadgerfishMessageFormatter"/>
  . . .
  </messageFormatters>
<messageBuilders>
  <messageBuilder contentType="text/javascript"
    class="org.apache.axis2.json.JSONBadgerfishOMBuilder"/>
  . . .
</messageBuilders>
```

Также требуется включение в путь приложения библиотеки jettison.jar платформы Axis2, обеспечивающей сериализацию/десериализацию JSON-сообщений.

Рассмотрим пример RESTful Web-сервиса и клиента RESTful Web-сервиса платформы Axis2.

1. Для создания RESTful Web-сервиса откроем среду Eclipse, в меню **File** выберем пункты **New | Dynamic Web Project** и введем имя проекта Axis2REST-WebService. Нажав кнопку **New Runtime**, выберем **Apache Tomcat v7.0**, в раскрывающемся списке **Dynamic web module version** выберем значение **2.5** и нажмем кнопку **Finish**.
2. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Axis2 Preferences**. На вкладке **Axis2 Runtime**, нажав кнопку **Browse**, определим каталог предварительно инсталлированной среды выполнения Apache Axis2 (Binary Distribution, **http://axis.apache.org/axis2/java/core/download.cgi**).
3. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Server and Runtime**, в раскрывающемся списке **Server runtime** выберем **Tomcat v7.0 Server**, а в раскрывающемся списке **Web service runtime** — значение **Apache Axis2** и нажмем кнопку **OK**.

4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2RESTWebService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Axis2 Web Services** и нажмем кнопку **OK**.

   В результате среда выполнения Axis2 будет интегрирована в проект Axis2RESTWebService как Web-приложение сервера Tomcat.

5. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2RESTWebService** и выберем пункты **New | Class**, введем имя Web-сервиса RESTService и имя пакета `restservice` и нажмем кнопку **Finish**.
6. Дополним код класса **RESTService**, как показано в листинге 6.6.
7. Для развертывания Web-сервиса RESTService на платформе Axis2 поверх сервера Tomcat в окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2RESTWebService** и выберем пункты **New | Other | Web Services | Web Service**. Нажмем кнопку **Next**, в строке **Service implementation** введем `restservice.RESTService` и нажмем кнопку **Finish**.

В результате в папку WebContent\WEB-INF\services проекта Axis2RESTWebService будет добавлена папка RESTService с класс-файлом Web-сервиса и его конфигурационным файлом services.xml.

Листинг 6.6. Код класса `RESTService`

```
package restservice;
import org.apache.axiom.om.*;

public class RESTService {
  public OMElement getHello(OMElement root) throws Exception {
    org.apache.axis2.context.MessageContext mcIn =
      org.apache.axis2.context.MessageContext.
      getCurrentMessageContext().getOperationContext().
      getMessageContext(org.apache.axis2.wsdl.WSDLConstants.
        MESSAGE_LABEL_IN_VALUE);
  Object object =
    mcIn.getProperty(org.apache.axis2.Constants.
      Configuration.MESSAGE_TYPE);
  String messageType = (String) object;
  System.out.println(messageType);
  System.out.println(mcIn.getEnvelope().toString());
  OMElement element = root.getFirstElement();
  String name=element.getText();
  System.out.println("Name"+" "+name);
  if (!messageType.equals("application/json")) {
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace(
      "http://restservice", "ns1");
    OMNamespace s1 = factory.createOMNamespace(
      "http://www.w3.org/2001/XMLSchema-instance", "s1");
```

```
    OMNamespace s2 = factory.createOMNamespace(
      "http://www.w3.org/2001/XMLSchema", "s2");
    OMElement rootresponse = factory.createOMElement(
      "getHelloResponse", ns1);
    OMElement returnresponse = factory.createOMElement("return", ns1);
    returnresponse.declareNamespace(s1);
    returnresponse.declareNamespace(s2);
    OMAttribute type = factory.createOMAttribute("type",s1,"s2:string");
    returnresponse.addAttribute(type);
    returnresponse.setText("Hello"+" "+name);
    rootresponse.addChild(returnresponse);
    System.out.println(rootresponse.toString());
    return rootresponse;}
  if (messageType.equals("application/json")) {
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns = factory.createOMNamespace("", "");
    OMElement resJson = factory.createOMElement("getHelloResponse", ns);
    OMElement returnJson = factory.createOMElement("return", ns);
    returnJson.setText("Hello "+name);
    resJson.addChild(returnJson);
    System.out.println(resJson.toString());
    return resJson; }
  return null;
}}
```

Метод `getHello()` Web-сервиса RESTService использует низкоуровневую AXIOM-обработку входящих и исходящих сообщений. Поэтому необходимо изменить конфигурационный файл services.xml папки WebContent\WEB-INF\services\RESTService\META-INF проекта Axis2RESTWebService для установки Handler-обработчика сообщений `org.apache.axis2.receivers.RawXMLINOutMessageReceiver`. Так как конфигурационный файл axis2.xml папки WebContent\WEB-INF\conf проекта Axis2RESTWebService определяет Handler-обработчик `RawXMLINOutMessageReceiver` на глобальном уровне, файл services.xml примет следующий вид:

```
<service name="RESTService" >
  <parameter name="ServiceClass" locked="false">
   restservice.RESTService</parameter>
</service>
```

Метод `getHello()` обрабатывает различным способом два типа входящих сообщений. Первый тип сообщений — это входящие сообщения, HTTP-заголовок `Content-Type` которых имеет значения `text/xml`, `application/soap+xml` и `application/xml`. Второй тип входящих сообщений — это сообщения, HTTP-заголовок `Content-Type` которых имеет значение `application/json`.

Для включения поддержки JSON-формата добавим в папку WebContent\WEB-INF\lib проекта Axis2RESTWebService библиотечный файл jettison.jar платформы Axis2 и изменим файл axis2.xml:

```
<messageFormatters>
  <messageFormatter contentType="application/x-www-form-urlencoded"
    class="org.apache.axis2.transport.http.XFormURLEncodedFormatter"/>
  <messageFormatter contentType="multipart/form-data"
    class="org.apache.axis2.transport.http.MultipartFormDataFormatter"/>
  <messageFormatter contentType="application/xml"
    class="org.apache.axis2.transport.http.ApplicationXMLFormatter"/>
  <messageFormatter contentType="text/xml"
    class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>
  <messageFormatter contentType="application/soap+xml"
    class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>
  <messageFormatter contentType="application/json"
    class="org.apache.axis2.json.JSONMessageFormatter"/>
</messageFormatters>
<messageBuilders>
  <messageBuilder contentType="application/xml"
    class="org.apache.axis2.builder.ApplicationXMLBuilder"/>
  <messageBuilder contentType="application/x-www-form-urlencoded"
    class="org.apache.axis2.builder.XFormURLEncodedBuilder"/>
  <messageBuilder contentType="multipart/form-data"
    class="org.apache.axis2.builder.MultipartFormDataBuilder"/>
  <messageBuilder contentType="application/json"
class="org.apache.axis2.json.JSONOMBuilder"/>
</messageBuilders>
```

Из конфигурационного файла axis2.xml видно, что формат `text/xml` и `application/soap+xml` обрабатывает один и тот же SOAP-обработчик `SOAPMessageFormatter`.

В методе `getHello()` в качестве первого шага из входящего сообщения извлекается значение HTTP-заголовка `Content-Type` и передаваемый параметр. Затем, в зависимости от типа входящего сообщения, различным способом формируются XML-данные ответа Web-сервиса. Отличие заключается в том, что формат JSON не поддерживает пространства имен, и поэтому ответное сообщение для типа `application/json` формируется с учетом этого.

Наберем в строке браузера адрес **http://localhost:8080/Axis2RESTWebService/services/RESTService/getHello?name=User** и в ответ увидим сообщение Web-сервиса:

```
<ns1:getHelloResponse xmlns:ns1="http://restservice">
<ns1:return xmlns:s2="http://www.w3.org/2001/XMLSchema"
            xmlns:s1="http://www.w3.org/2001/XMLSchema-instance"
            s1:type="s2:string">Hello User</ns1:return>
</ns1:getHelloResponse>
```

В случае такого GET-запроса тип входящего сообщения будет `application/xml`, а сам GET-запрос будет конвертирован средой выполнения Axis2 в SOAP-сообщение:

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <axis2ns4:getHello xmlns:axis2ns4="http://restservice">
      <name>User</name>
    </axis2ns4:getHello>
  </soapenv:Body>
</soapenv:Envelope>
```

Ответное сообщение не является SOAP-сообщением, а будет представлять собой чистые XML-данные.

1. Для создания клиента Web-сервиса в перспективе Java среды Eclipse в меню **File** выберем пункты **New | Java Project**, введем имя проекта Axis2RESTClient и нажмем кнопку **Finish**.
2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2RESTClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module 2.5**, **Axis2 Web Services** и нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **Axis2RESTClient** и выберем пункты **New | Class**, введем имя класса `RESTClient` и имя пакета `restclient`, укажем опцию **public static void main(String[] args)** и нажмем кнопку **Finish**.
4. Изменим код класса `RESTClient`, как показано в листинге 6.7.

Листинг 6.7. Код класса `RESTClient`

```
package restclient;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class RESTClient {
  public static void main(String[] args) throws Exception {
    java.io.File configFile =
      new java.io.File("WebContent/WEB-INF/conf/axis2.xml");
    org.apache.axis2.context.ConfigurationContext configurationContext =
      org.apache.axis2.context.ConfigurationContextFactory.
        createConfigurationContextFromFileSystem(null,
          configFile.getAbsolutePath());
    ServiceClient sc = new ServiceClient(configurationContext, null);
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://restservice",
      "ns1");
    OMElement root = factory.createOMElement("getHello", ns1);
    OMElement name = factory.createOMElement("name", ns1);
    name.setText("User");
    root.addChild(name);
```

```
    Options op = new Options();
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
      "http://127.0.0.1:1234/Axis2RESTWebService/services/RESTService"));
    op.setProperty(org.apache.axis2.Constants.Configuration.ENABLE_REST,
      Boolean.TRUE);
    //Soap
    op.setProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE,
      "text/xml");
    op.setAction("urn:getHello");
    sc.setOptions(op);
    OMElement resSoap = sc.sendReceive(root);
    System.out.println("SOAP "+resSoap.toString());
    //XML
    op.setProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE,
      "application/xml");
    op.setAction("");
    sc.setOptions(op);
    OMElement resXml = sc.sendReceive(root);
    System.out.println("XML "+resXml.toString());
    //JSON
    op.setProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE,
      "application/json");
    sc.setOptions(op);
    OMNamespace ns = factory.createOMNamespace("", "");
    OMElement rootJson = factory.createOMElement("getHello", ns);
    OMElement nameJson = factory.createOMElement("name", ns);
    nameJson.setText("User");
    rootJson.addChild(nameJson);
    OMElement resJson = sc.sendReceive(rootJson);
    System.out.println("JSON "+resJson.toString());
}}
```

Для включения поддержки JSON-формата конфигурационный файл axis2.xml папки WebContent\WEB-INF\conf проекта Axis2RESTClient изменен аналогично файлу axis2.xml проекта Axis2RESTWebService и в путь приложения Axis2RESTClient добавлена библиотека jettison.jar платформы Axis2 с помощью выбора опций **Properties | Java Build Path | Libraries | Add External JARs**.

В методе `main()` класса `RESTClient` объект `ServiceClient` создается на основе измененного файла axis2.xml. Так как файл axis2.xml подключает на глобальном уровне Axis2-модуль addressing.mar, то и его необходимо добавить в путь приложения Axis2RESTClient.

Кроме того, в данном случае для просмотра сообщений, участвующих в обмене между клиентом и Web-сервисом, используется TCPMon-монитор, поэтому адрес конечной точки Web-сервиса указывает порт 1234, а не 8080.

В методе `main()` класса `RESTClient` поддержка REST включается с помощью определения свойства `Options`-объекта:

```
op.setProperty(org.apache.axis2.Constants.Configuration.ENABLE_REST,
                Boolean.TRUE);
```

Интерфейс OM API используется для создания двух типов исходящих сообщений. Первый тип исходящих сообщений содержит XML-данные с объявлением пространства имен и предназначен для значений HTTP-заголовка `Content-Type`, равных `text/xml` и `application/xml`. XML-данные второго типа исходящих сообщений не содержат объявление пространства имен и предназначены для JSON-формата. Тип исходящих сообщений устанавливается определением соответствующего свойства `Options`-объекта.

После запуска класса `RESTClient` как Java-приложения в консоли среды Eclipse можно увидеть входящие сообщения клиента Web-сервиса, а в окне TCPMon-монитора входящие и исходящие сообщения Web-сервиса.

Видно, что входящее и исходящее сообщения Web-сервиса типа `text/xml` соответствуют SOAP-формату, сообщения типа `application/xml` представляют собой чистые XML-данные, а сообщения типа `application/json` имеют JSON-формат.

На компакт-диске

Проекты Axis2RESTClient и Axis2RESTWebService находятся в папке Примеры\Глава6\REST компакт-диска.

Связывание данных

Платформа Axis2 поддерживает несколько систем связывания данных — Axis Data Binding (ADB), XMLBeans, JibX и JaxBRI, обеспечивающих связь XML-данных Web-сервиса, определяемых XML-схемой его WSDL-описания, с Java-объектами.

Система связывания данных предназначена для генерации Java-классов из XML-схем с последующей маршализацией/демаршализацией XML-данных на основе сгенерированных Java-классов.

Платформа Axis2 позволяет как Web-сервису, так и клиенту Web-сервиса работать без использования какой-либо системы связывания данных. При этом Web-сервис и клиент Web-сервиса обрабатывают XML-данные напрямую на основе модели AXIOM. На стороне клиента Web-сервиса это позволяет делать программный интерфейс ServiceClient API и OperationClient API. На стороне Web-сервиса за такую обработку отвечают обработчики `org.apache.axis2.receivers.RawXML*MessageReceiver`.

Java-классы, сгенерированные из WSDL-описания Web-сервиса, избавляют приложение от необходимости взаимодействовать напрямую с AXIOM-моделью. Приложение работает теперь с Java-структурами данных, а сгенерированный на основе системы связывания данных Java-код обеспечивает детали сериализации/десериализации Java-структур данных в XML-данные и из них.

Для генерации Java-кода на основе системы связывания данных платформа Axis2 предоставляет инструмент командной строки WSDL2Java, расположенный в папке bin каталога платформы Axis2. Для среды разработки Eclipse существует плагин Code Generator Wizard (**http://axis.apache.org/axis2/java/core/tools/index.html**),

обеспечивающий генерацию Java-кода из WSDL-документа и наоборот. После помещения плагина в папку dropins каталога среды Eclipse в окне **New** в узле **Other | Axis2 Wizards** среды Eclipse появится узел **Axis2 Code Generator** (рис. 6.7), позволяющий создавать артефакты как на стороне сервера, так и на стороне клиента.

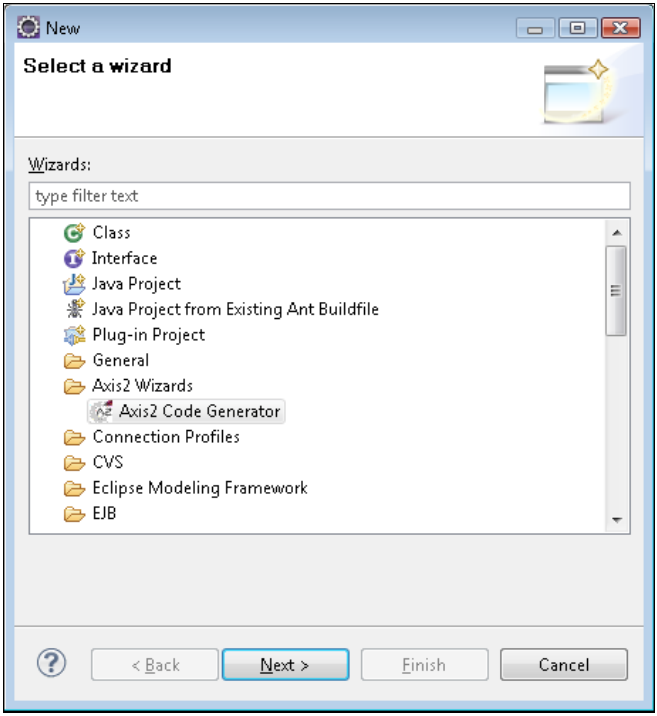


Рис. 6.7. Eclipse-плагин генерации Java-кода из WSDL-документа и наоборот

Axis2-инструмент WSDL2Java имеет следующие опции:

- `-uri <Location of WSDL>` — месторасположение WSDL-документа;
- `-o <output Location>` — месторасположение папки вывода;
- `-l <language>` — язык программирования артефактов;
- `-p <package name>` — имя пакета;
- `-a` — артефакты создаются для асинхронного клиента;
- `-s` — артефакты создаются для синхронного клиента;
- `-t` — генерация теста;
- `-ss` — генерация артефактов на стороне сервера;
- `-sd` — генерация файла server.xml на стороне сервера;
- `-d <databinding>` — используемая система связывания данных (`xmlbeans`, `adb`, `jibx`, `jaxbri`, `none`);
- `-g` — на стороне сервера генерируется и клиентский код;

- `-u` — генерируются отдельные классы связывания данных;
- `-sn <service name>` — имя сервиса;
- `-pn <port name>` — имя порта;
- `-ns2p` — связывает пространства имен с именами пакетов;
- `-ssi` — для артефакта на стороне сервера генерируется интерфейс;
- `-wv` — версия WSDL (2, 2.0, 1.1);
- `-S` — адрес для генерируемой папки src;
- `-R` — адрес для генерируемой папки resources;
- `-em` — внешний mapping-файл;
- `-f` — внутренние папки src и resources не создаются;
- `-uw` — параметры определяются без оберток;
- `-xsdconfig` — используется XMLBeans-файл .xsdconfig;
- `-ap` — код генерируется для всех портов;
- `-or` — перезаписывание существующих классов;
- `-b` — совместимость генерируемого кода с платформой Axis 1.*x*;
- `-sp` — исключение префиксов пространств имен;
- `--noBuildXML` — файл build.xml не генерируется;
- `--noWSDL` — WSDL-документ не генерируется;
- `--noMessageReceiver` — обработчик `MessageReceiver` не генерируется.

В случае применения инструмента WSDL2Java для генерации артефактов на стороне клиента создается класс клиентской заглушки, расширяющий класс `org.apache.axis2.client.Stub` и использующий Java-классы системы связывания данных. При использовании инструмента WSDL2Java для генерации артефактов на стороне сервера в папке вывода могут быть созданы следующие файлы, классы и интерфейсы:

- ant-файл build.xml;
- папка resources с WSDL-описанием Web-сервиса и его конфигурационным файлом services.xml;
- папка src с пакетом, содержащим:
  - классы системы связывания данных;
  - Skeleton-класс, представляющий основу класса реализации Web-сервиса и использующий классы системы связывания данных;
  - интерфейс Skeleton-класса;
  - обработчик `MessageReceiver`, использующий классы системы связывания данных;
  - extensionmapper-класс, связывающий XML-типы данных с Java-типами данных.

ADB (Axis2 Databinding)

Система связывания данных ADB является внутренней системой платформы Axis2, обеспечивающей компиляцию XML-схем и генерацию JavaBean-классов и построенной на модели XML-документа AXIOM. Система ADB определена платформой Axis2 системой по умолчанию для генерации Java-классов из XML-схем данных и затем для маршализации/демаршализации XML-данных на их основе.

Архитектура системы ADB состоит из следующих компонентов.

- Класс `org.apache.axis2.schema.SchemaCompiler` осуществляет разбор набора XML-схем на основе объектной Java-модели XMLSchema (**http://ws.apache.org/commons/XmlSchema/**) и затем генерирует Java-классы с помощью класса, реализующего интерфейс `org.apache.axis2.schema.writer.BeanWriter`. Класс `SchemaCompiler` также использует класс, реализующий интерфейс `org.apache.axis2.schema.typemap.TypeMap`, который отвечает за связывание QName-имен с именами классов.
- Класс `org.apache.axis2.schema.writer.JavaBeanWriter` является реализацией интерфейса `BeanWriter` по умолчанию и отвечает за создание JavaBean-классов.
- Класс `org.apache.axis2.schema.typemap.JavaTypeMap` является реализацией интерфейса `TypeMap` по умолчанию и осуществляет связывание типов данных.
- Файл schema-compile.properties устанавливает конфигурацию системы ADB. Указывает реализацию интерфейсов `BeanWriter` и `TypeMap` и XSL-файл, содержащий XSLT-шаблон, используемый `BeanWriter`-классом для генерации Java-классов определенного формата. По умолчанию система ADB содержит два шаблона — ADBBeanTemplate.xsl и PlainBeanTemplate.xsl. Шаблон ADBBeanTemplate.xsl определяет генерацию Java-классов, реализующих интерфейс `org.apache.axis2.databinding.ADBBean`, а шаблон PlainBeanTemplate.xsl — генерацию простых JavaBean-классов.
- Файл ADBBeanTemplate.xsl используется по умолчанию `BeanWriter`-классом в качестве шаблона для генерации `ADBBean`-классов, реализующих интерфейс `ADBBean`, который обеспечивает маршализацию/демаршализацию XML-данных на основе модели AXIOM.

Система ADB содержит класс `org.apache.axis2.schema.XSD2Java`, метод `main()` которого обеспечивает генерацию Java-кода из XML-схемы. В качестве аргументов метод `main()` класса `XSD2Java` принимает полное имя файла XML-схемы и полное имя исходящей папки файловой системы. Однако применить все возможности системы ADB для генерации Java-кода из XML-схем позволяет программный интерфейс ADB API, главным классом которого является класс `org.apache.axis2.schema.SchemaCompiler`.

Конструктор класса `SchemaCompiler` принимает в качестве аргумента объект `org.apache.axis2.schema.CompilerOptions`, обеспечивающий конфигурацию ADB-компилятора. Класс `CompilerOptions` своими методами позволяет установить папку, содержащую сгенерированный Java-код, имя Java-пакета и пространство имен

с ним связанное, определить режим, при котором генерируется один класс, содержащий все остальные классы (Wrapped Mode), режим генерации классов для всех элементов и именованных комплексных типов XML-схемы, режим генерации классов. Кроме того, метод `setHelperMode(boolean helperMode)` класса `CompilerOptions` позволяет установить режим компиляции, при котором генерируются простые JavaBean-классы, а все методы сериализации/десериализации выносятся в отдельные Helper-классы.

Несмотря на все удобство применения системы связывания ADB, она имеет ограничения в распознавании таких структур XML-схемы, как элементы `<extension>` и `<restriction>`. Поэтому платформа Axis2 поддерживает другие системы связывания, такие как XMLBeans, JibX и JaxBRI. В частности, среда Eclipse в окне, открываемом опциями **Window | Preferences | Web Services | Axis2 Preferences**, дает возможность сделать выбор между системой ADB и системой XMLBeans для генерации артефактов на стороне сервера и клиента Web-сервиса.

XMLBeans

Система связывания XMLBeans является полнофункциональной платформой для обработки XML-схем, не имеющей ограничений системы ADB, но более сложной по сравнению с ADB.

Платформа XMLBeans уже обсуждалась в *разд. "Связывание данных XMLBeans" главы 5*.

Платформа Axis2 обеспечивает поддержку системы XMLBeans с помощью пакета `org.apache.axis2.xmlbeans`, содержащего классы `CodeGenerationUtility` и `XSDConfig`. Для генерации Java-классов системы связывания данных формата XMLBeans используется XSLT-шаблон XmlbeansDatabindingTemplate.xsl.

JiBX

Система связывания данных JiBX (**http://jibx.sourceforge.net/**) обеспечивает следующий набор возможностей:

- генерацию XML-схемы из Java-кода;
- генерацию Java-кода из XML-схемы;
- связывание существующих Java-классов с XML-схемой.

Основным преимуществом системы JiBX является возможность работать с существующими Java-классами, представляющими данные, что позволяет создавать и разворачивать Web-сервисы из уже существующего Java-кода.

Архитектура системы JiBX состоит из основных компонентов, таких как:

- документ определения связывания;
- компилятор связывания;
- среда выполнения связывания.

Документ определения связывания устанавливает конфигурацию маршализации/демаршализации Java-объектов в XML-данные и из них. Документ JiBX-связывания используется компилятором связывания для обработки класс-файлов, представляющих данные, при которой производится компилирование документа JiBX-связывания в байт-код, добавляемый в обрабатываемые класс-файлы. Добавляемый байт-код реализует процессы маршализации/демаршализации Java-объектов и используется средой выполнения связывания для конвертации Java-объектов в XML-данные и наоборот. Компилятор связывания представлен классом `org.jibx.binding.Compile` библиотеки jibx-bind.jar, а среда выполнения связывания — библиотекой jibx-run.jar.

JiBX-компилятор связывания может быть запущен как Java-приложение инструментом командной строки `java` или с помощью Eclipse-плагина, который можно установить, используя меню **Help | Install New SoftWare** среды Eclipse и адрес **http://jibx.sourceforge.net/eclipse/**.

После модификации Java класс-файлов данных JiBX-компилятором маршализация/демаршализация данных осуществляется с помощью программного интерфейса JiBX API на основе JiBX-среды выполнения.

Статический метод `getFactory()` (с аргументом Java-класса связывания) класса `org.jibx.runtime.BindingDirectory` позволяет создать объект `org.jibx.runtime.IBindingFactory`, который предоставляет методы `createMarshallingContext()` и `createUnmarshallingContext()` для создания контекста маршализации и демаршализации `org.jibx.runtime.IMarshallingContext` и `org.jibx.runtime.IUnmarshallingContext`. Объекты `IMarshallingContext` и `IUnmarshallingContext` имеют методы `marshalDocument()` и `unmarshalDocument()`, обеспечивающие маршализацию/демаршализацию данных.

Компилятор связывания, среда выполнения связывания, а также другие библиотеки и инструменты системы JiBX находятся в папке lib JiBX-дистрибутива (**http://sourceforge.net/projects/jibx/files/**).

Корневым элементом документа JiBX-связывания является элемент `<binding>`, имеющий следующие атрибуты:

- `add-constructors` — если `true`, тогда компилятор связывания добавляет конструктор по умолчанию в класс данных;
- `direction` — направление связывания, возможные значения: `input` (только демаршализация), `output` (только маршализация), `both` (по умолчанию маршализация/демаршализация);
- `force-classes` — если `true`, тогда классы маршализации/демаршализации генерируются для нерасширяемого абстрактного связывания типов данных верхнего уровня (по умолчанию `false`);
- `forwards` — если `true` (по умолчанию), тогда при демаршализации допускаются опережающие ссылки на идентификаторы. Если `true`, тогда идентификаторы должны быть определены перед ссылками на них;
- `name` — имя связывания;

- `package` — имя Java-пакета создаваемого класса-фабрики;
- `track-source` — если `true`, тогда компилятор добавляет реализацию интерфейса `org.jibx.runtime.ITrackSource`, по умолчанию `false`;
- `trim-whitespace` — если `true`, тогда при десериализации удаляются пробелы;
- `value-style` — стиль связывания `"element"` или `"attribute"`.

Элемент `<include>` документа JiBX-связывания позволяет включить другой документ определения связывания в существующий документ. Элемент `<include>` имеет атрибуты:

- `path` — адрес включаемого документа;
- `precompiled` — если `true`, тогда включаемый документ уже компилировался и используемые им классы включены в `classpath`.

Элемент `<format>` документа JiBX-связывания определяет формат нестандартной конвертации простых Java-типов в XML-текст и наоборот. При демаршализации текстовое содержимое элемента или атрибута передается десериализатору, который определен элементом `<format>` и возвращает значение подходящего Java-типа. При маршализации значение определенного Java-типа передается сериализатору, который определен элементом `<format>` и возвращает текстовое содержимое элемента или атрибута. Элемент `<format>` имеет следующие атрибуты:

- `label` — идентификатор формата. Если идентификатор отсутствует, тогда формат становится форматом по умолчанию для данного Java-типа;
- `type` — полное имя обрабатываемого Java-типа;
- `default` — значение по умолчанию для конвертации;
- `enum-value-method` — метод, возвращающий XML-текст для Java-перечисления. По умолчанию используется метод `toString()` класса перечисления для маршализации и метод `valueOf()` — для демаршализации;
- `deserializer` — полное имя статического метода десериализации;
- `serializer` — полное имя статического метода сериализации;
- `whitespace` — способ обработки пробелов при десериализации. Возможные значения: `preserve`, `replace`, `collapse` и `trim`.

Элемент `<namespace>` документа JiBX-связывания определяет пространство имен, используемое в XML-документах, с помощью следующих атрибутов:

- `uri` — URI пространства имен;
- `default` — использование пространства имен по умолчанию. Возможные значения: `none` (использование при определении), `elements` (только для элементов), `attributes` (только для атрибутов), `all` (для элементов и атрибутов);
- `prefix` — префикс пространства имен.

Элемент `<mapping>` документа JiBX-связывания определяет конвертацию Java-объектов в XML-данные и наоборот.

Элемент `<mapping>` имеет следующие атрибуты:

- `class` — полное имя Java-класса;
- `abstract` — если `true`, тогда связывания является абстрактным, т. е. соответствует именованному комплексному типу XML-схемы;
- `extends` — полное имя Java-класса базового связывания, которое расширяет данное связывание;
- `type-name` — имя абстрактного связывания для ссылок атрибутом `map-as` элемента `<structure>`;
- `value-style` — стиль связывания для простых значений. Возможные значения: `element` и `attribute`;
- `name` — локальное имя элемента или атрибута для связывания;
- `ns` — URI пространства имен элемента или атрибута;
- `create-type` — Java-тип при демаршализации;
- `factory` — метод класса-фабрики для создания Java-объектов при демаршализации;
- `marshaller` — класс сериализации, реализующий интерфейс `org.jibx.runtime.Marshaller`;
- `nillable` — если `true`, тогда разрешается `xsi:nil="true"`, по умолчанию `false`;
- `post-set` — метод класса, вызываемый после демаршализации;
- `pre-get` — метод класса, вызываемый перед маршализацией;
- `pre-set` — метод класса, вызываемый перед демаршализацией;
- `unmarshaller` — класс десериализации, реализующий интерфейс `org.jibx.runtime.Unmarshaller`;
- `allow-repeats` — если `true`, тогда разрешаются повторяющиеся элементы, по умолчанию `false`;
- `choice` — если `true`, тогда связывание представляет выбор, если `false` (по умолчанию), тогда связывание представляет набор;
- `flexible` — если `true`, тогда неизвестные элементы игнорируются, по умолчанию `false`;
- `label` — идентификатор связывания;
- `ordered` — если `true` (по умолчанию), тогда связывание представляет упорядоченный список;
- `using` — ссылка на используемые элементы `<mapping>`, `<structure>`, `<collection>`.

Элемент `<value>` документа JiBX-связывания связывает поля Java-класса с XML-элементами или XML-атрибутами с помощью следующих атрибутов:

- `constant` — постоянное значение элемента или атрибута;
- `format` — имя формата для конвертации;

- `ident` — возможные значения: `none` (по умолчанию значение не является идентификатором), `def` (значение — идентификатор), `ref` (значение — ссылка);
- `style` — XML-тип, значения: `attribute`, `element`, `text`, `cdata`;
- `name` — локальное имя элемента или атрибута для связывания;
- `ns` — URI пространства имен элемента или атрибута;
- `field` — имя поля Java-класса;
- `flag-method` — имя вызываемого метода, которому в качестве аргумента передается флаг `true`, если элемент существует, или `false`, если элемент отсутствует;
- `get-method` — метод, возвращающий значение;
- `set-method` — метод, устанавливающий значение;
- `test-method` — метод, возвращающий `true`, если свойство класса существует;
- `type` — Java-тип значения;
- `usage` — обязательность свойства. Возможные значения: `required` и `optional`;
- `default` — значение по умолчанию для конвертации;
- `enum-value-method` — метод, возвращающий XML-текст для Java-перечисления. По умолчанию используется метод `toString()` класса перечисления для маршализации и метод `valueOf()` — для демаршализации;
- `deserializer` — полное имя статического метода десериализации;
- `serializer` — полное имя статического метода сериализации;
- `whitespace` — способ обработки пробелов при десериализации. Возможные значения: `preserve`, `replace`, `collapse` и `trim`.

Элемент `<structure>` документа JiBX-связывания определяет для связывания структуру, представляющую Java-объект, XML-элемент или их вместе. Элемент `<structure>` имеет следующие атрибуты:

- `map-as` — переопределение Java-типа;
- `value-style` — стиль связывания для простых значений. Возможные значения: `element` и `attribute`;
- `name` — локальное имя элемента или атрибута для связывания;
- `ns` — URI пространства имен элемента или атрибута;
- `create-type` — Java-тип при демаршализации;
- `factory` — метод класса-фабрики для создания Java-объектов при демаршализации;
- `marshaller` — класс сериализации, реализующий интерфейс `org.jibx.runtime.Marshaller`;
- `nillable` — если `true` , тогда разрешается `xsi:nil="true"`, по умолчанию `false`;
- `post-set` — метод класса, вызываемый после демаршализации;

- `pre-get` — метод класса, вызываемый перед маршализацией;
- `pre-set` — метод класса, вызываемый перед демаршализацией;
- `unmarshaller` — класс десериализации, реализующий интерфейс `org.jibx.runtime.Unmarshaller`;
- `field` — имя поля Java-класса;
- `flag-method` — имя вызываемого метода, которому в качестве аргумента передается флаг `true`, если элемент существует, или `false`, если элемент отсутствует;
- `get-method` — метод, возвращающий значение;
- `set-method` — метод, устанавливающий значение;
- `test-method` — метод, возвращающий `true`, если свойство класса существует;
- `type` — Java-тип значения;
- `usage` — обязательность свойства. Возможные значения: `required` и `optional`;
- `allow-repeats` — если `true`, тогда разрешаются повторяющиеся элементы, по умолчанию `false`;
- `choice` — если `true`, тогда связывание представляет выбор, если `false` (по умолчанию), тогда связывание представляет набор;
- `flexible` — если `true`, тогда неизвестные элементы игнорируются, по умолчанию `false`;
- `label` — идентификатор элемента;
- `ordered` — если `true` (по умолчанию), тогда связывание представляет упорядоченный список;
- `using` — ссылка на используемые элементы.

Элемент `<collection>` документа JiBX-связывания определяет связывание для Java-коллекции с помощью следующих атрибутов:

- `load-method` — метод, возвращающий значение из коллекции;
- `size-method` — метод, возвращающий размер коллекции;
- `store-method` — метод, сохраняющий значение в коллекции;
- `add-method` — метод, добавляющий значение в коллекцию;
- `iter-method` — метод, возвращающий объект `java.lang.Iterator` или `java.lang.Enumeration` коллекции;
- `item-type` — имя Java-типа коллекции;
- `value-style` — стиль связывания для простых значений. Возможные значения: `element` и `attribute`;
- `name` — локальное имя элемента или атрибута для связывания;
- `ns` — URI пространства имен элемента или атрибута;
- `create-type` — Java-тип при демаршализации;

- `factory` — метод класса-фабрики для создания Java-объектов при демаршализации;
- `marshaller` — класс сериализации, реализующий интерфейс `org.jibx.runtime.Marshaller`;
- `nillable` — если `true`, тогда разрешается `xsi:nil="true"`, по умолчанию `false`;
- `post-set` — метод класса, вызываемый после демаршализации;
- `pre-get` — метод класса, вызываемый перед маршализацией;
- `pre-set` — метод класса, вызываемый перед демаршализацией;
- `unmarshaller` — класс десериализации, реализующий интерфейс `org.jibx.runtime.Unmarshaller`;
- `field` — имя поля Java-класса;
- `flag-method` — имя вызываемого метода, которому в качестве аргумента передается флаг `true`, если элемент существует, или `false`, если элемент отсутствует;
- `get-method` — метод, возвращающий значение;
- `set-method` — метод, устанавливающий значение;
- `test-method` — метод, возвращающий `true`, если свойство класса существует;
- `type` — Java-тип значения;
- `usage` — обязательность свойства. Возможные значения: `required` и `optional`;
- `allow-repeats` — если `true`, тогда разрешаются повторяющиеся элементы, по умолчанию `false`;
- `choice` — если `true`, тогда связывание представляет выбор, если `false` (по умолчанию), тогда связывание представляет набор;
- `flexible` — если `true`, тогда неизвестные элементы игнорируются, по умолчанию `false`;
- `label` — идентификатор элемента;
- `ordered` — если `true` (по умолчанию), тогда связывание представляет упорядоченный список;
- `using` — ссылка на используемые элементы.

Система JiBX содержит набор инструментов для создания документа JiBX-связывания, Java-кода, XML-схемы и WSDL-документа.

Инструмент BindGen предназначен для генерации документа JiBX-связывания и XML-схемы из существующего Java-кода, представляющего данные.

Инструмент BindGen находится в библиотеке jibx-tools.jar как класс-файл `org.jibx.binding.generator.BindGen` с методом `main()`, принимающим в качестве аргумента список классов, которые должны быть включены в документ JiBX-связывания. Инструмент BindGen может быть запущен как Java-приложение инструментом командной строки `java` с включенными в путь приложения библиотеками jibx-tools.jar, jibx-bind.jar, jibx-schema.jar и jibx-run.jar.

Инструмент BindGen также имеет следующие параметры командной строки:

- `-a` — генерируется абстрактное связывание, соответствующее комплексному типу XML-схемы;
- `-b` *`name`* — имя генерируемого документа JiBX-связывания (по умолчанию binding.xml);
- `-c` *`path`* — расположение конфигурационного BindGen-файла, переопределяющего генерацию по умолчанию;
- `-m` — генерируется обычное не абстрактное связывание, соответствующее комплексному типу плюс элементу этого типа XML-схемы;
- `-n uri=`*`name`*`,...` — определяет имена файлов для пространств имен;
- `-o` — генерация только документа JiBX-связывания без XML-схемы;
- `-p` *`path`*`,...` — расположение обрабатываемых Java-классов;
- `-s` *`path`*`,...` — расположение файлов Java-классов с исходным кодом (по умолчанию не используются);
- `-t` *`path`* — путь исходящей каталога;
- `-v` — вывод информации при работе инструмента;
- `-w` — очистка исходящей каталога перед выводом.

Конфигурационный BindGen-файл имеет следующий вид:

```
<custom [атрибуты]>
  <package [атрибуты]>
   <class [атрибуты]>
      <value [атрибуты]/>
    ...
    </class>
    ...
  </package>
...
</custom>
```

где элементы `<custom>`, `<package>`, `<class>` и `<value>` переопределяют генерацию по умолчанию на разных уровнях с помощью своих атрибутов.

Общие атрибуты для элементов `<custom>`, `<package>`, `<class>` и `<value>`:

- `force-mapping` — если `true`, тогда инструмент BindGen генерирует связывание и схему для каждого класса, по умолчанию `false`;
- `force-names` — если `true` (по умолчанию), тогда для ссылок определяются оберточные элементы;
- `javadoc-formatter` — имя класса, реализующего интерфейс `org.jibx.custom.classes.IDocumentFormatter` и отвечающего за конвертацию JavaDoc-текста кода в документацию XML-схемы;
- `map-abstract` — если `true`, тогда генерируется абстрактное связывание;

- `namespace` — базовый URI-идентификатор пространства имен;
- `namespace-style` — способ создания пространств имен. Возможные значения: `none`, `package`, `fixed`;
- `name-style` — способ создания XML-имен. Возможные значения: `camel-case` (по умолчанию), `upper-camel-case`, `hyphenated`, `dotted`, `underscored`;
- `property-access` — если `true`, тогда используются значения, доступные с помощью методов класса `get()`/`set()`; если `true` (по умолчанию) — используются поля класса;
- `require` — Java-типы, обязательные к обработке. Возможные значения: `none`, `primitives` (по умолчанию), `objects`, `all`;
- `strip-prefixes` — удаляемые префиксы из Java-имен;
- `strip-suffixes` — удаляемые суффиксы из Java-имен;
- `use-javadocs` — если `true` (по умолчанию), тогда JavaDocs-комментарии экспортируются в XML-схему;
- `value-style` — тип представления простых значений. Возможные значения: `attribute` (по умолчанию) и `element`;
- `wrap-collections` — если `true`, тогда для коллекций определяется оберточный элемент, по умолчанию `false`.

Индивидуальные атрибуты элемента `<custom>`:

- `add-constructors` — если `true`, тогда в связывании определяется добавление конструктора по умолчанию;
- `direction` — для связывания определяется тип конвертации. Возможные значения: `input` (демаршализация), `output` (маршализация), `both`;
- `force-classes` — если `true`, тогда в связывании определяется генерация классов маршализации/демаршализации для нерасширяемого абстрактного связывания верхнего уровня;
- `track-source` — если `true`, тогда для связывания определяется реализация интерфейса `org.jibx.runtime.ITrackSource`.

Элемент `<package>` имеет индивидуальный атрибут `name`, указывающий имя пакета.

Элемент `<class>` имеет следующие индивидуальные атрибуты:

- `create-type` — Java-тип экземпляров класса;
- `deserializer` — статический метод класса десериализации;
- `element-name` — имя элемента для данного класса;
- `enum-value-method` — имя метода класса перечисления, возвращающего XML-текстовое значение;
- `excludes` — имена значений, исключаемых из XML-представления;
- `factory` — метод класса-фабрики для создания экземпляра класса;

- `form` — тип связывания. Возможные значения: `default`, `abstract-mapping`, `concrete-mapping`, `simple`;
- `includes` — имена значений, включаемых в XML-представление;
- `name` — имя класса;
- `optionals` — имена значений, необязательных для XML-представления;
- `requireds` — имена значений, обязательных для XML-представления;
- `serializer` — статический метод класса сериализации;
- `type-name` — имя комплексного элемента при генерации абстрактного связывания;
- `use-super` — если `true` (по умолчанию), тогда значения суперкласса включаются в XML-представление.

Индивидуальные атрибуты элемента `<value>`:

- `actual-type` — Java-тип значения;
- `attribute` — имя XML-атрибута для представления значения;
- `create-type` — Java-тип для создания экземпляра значения;
- `element` — имя XML-элемента для представления значения;
- `factory` — статический метод класса-фабрики для создания экземпляра значения;
- `field` — связывает значение с полем класса;
- `get-method` — метод, возвращающий значение;
- `item-name` — имя экземпляров коллекции;
- `item-type` — тип экземпляров коллекции;
- `property-name` — связывает значение со свойством класса;
- `required` — если `true`, тогда значение является обязательным XML-компонентом;
- `set-method` — метод, устанавливающий значение.

Атрибуты глобального уровня могут передаваться в командную строку инструмента BindGen без создания конфигурационного BindGen-файла.

Инструмент SchemaGen предназначен для генерации XML-схемы из существующих — документа JiBX-связывания и Java-кода, представляющего данные. Инструмент SchemaGen представлен классом `org.jibx.schema.generator.SchemaGen` библиотеки jibx-tools.jar.

Инструмент Jibx2Wsdl обеспечивает генерацию документа JiBX-связывания, WSDL-документа и XML-схемы из существующего Java-кода, позволяя представить существующий код как Web-сервис. Инструмент Jibx2Wsdl работает как Java-приложение `org.jibx.ws.wsdl.tools.Jibx2Wsdl` библиотеки jibx-tools.jar и может запускаться Java-инструментом командной строки `java`.

Инструмент Jibx2Wsdl имеет следующие параметры командной строки:

- `-b` *`name`* — имя документа JiBX-связывания;
- `-c` *`path`* — расположение конфигурационного Jibx2Wsdl-файла;
- `-d` — используется WSDL-стиль document/literal без оберток, при этом каждый метод Web-сервиса может иметь только один параметр;
- `-n uri=`*`name`*`,...` — связывает пространства имен с именами файлов;
- `-p` *`path`*`,...` — путь используемых Java-файлов;
- `-s` *`path`*`,...` — путь Java-файлов с исходным кодом;
- `-t` *`path`* — исходящий каталог;
- `-u` *`binding`*`,...;`*`schema`*`,...` — существующие документы JiBX-связывания и XML-схемы для генерации WSDL-документа;
- `-v` — вывод подробной информации при работе инструмента;
- `-w` — очистка исходящей каталога перед выводом;
- `-x` *`class`*`,...` — имена дополнительных используемых классов.

Конфигурационный Jibx2Wsdl-файл имеет те же элементы, что и конфигурационный BindGen-файл с дополнительными элементами `<wsdl>`, `<service>`, `<operation>`, `<parameter>` и `<return>`. Jibx2Wsdl-элементы имеют те же общие атрибуты, что и BindGen-элементы, а также свои индивидуальные атрибуты.

Индивидуальные атрибуты элемента `<wsdl>`:

- `service-base` — базовый адрес конечной точки Web-сервиса (по умолчанию `http://localhost:8080/axis2/services`);
- `set-actions` — если `true` (по умолчанию), тогда WSDL-документ будет содержать атрибуты `soapAction`;
- `use-nillable` — если `true`, тогда разрешается использование `nillable="true"` в XML-схеме, по умолчанию `true` (используется `minOccurs="0"`);
- `wrapped` — если `true` (по умолчанию), тогда применяется WSDL-стиль "wrapped" document/literal;
- `wsdl-namespace` — целевое пространство имен WSDL-документа.

Индивидуальные атрибуты элемента `<service>`:

- `service-base` — базовый адрес конечной точки Web-сервиса (по умолчанию `http://localhost:8080/axis2/services`);
- `set-actions` — если `true` (по умолчанию), тогда WSDL-документ будет содержать атрибуты `soapAction`;
- `use-nillable` — если `true`, тогда разрешается использование `nillable="true"` в XML-схеме, по умолчанию `true` (используется `minOccurs="0"`);
- `wrapped` — если `true` (по умолчанию), тогда применяется WSDL-стиль "wrapped" document/literal;

- `wsdl-namespace` — целевое пространство имен WSDL-документа;
- `binding-name` — имя элемента `<wsdl:binding>`;
- `class` — имя класса сервиса;
- `excludes` — имена исключаемых Java-методов;
- `includes` — имена включаемых Java-методов;
- `port-name` — имя элемента `<wsdl:port>`;
- `port-type-name` — имя элемента `<wsdl:portType>`;
- `service-address` — адрес конечной точки;
- `service-name` — имя сервиса.

Индивидуальные атрибуты элемента `<operation>`:

- `method-name` — имя метода сервиса;
- `operation-name` — имя WSDL-операции;
- `optionals` — список необязательных параметров;
- `request-message` — имя входящего сообщения;
- `request-wrapper` — имя оберточного элемента для входящего сообщения;
- `requireds` — список обязательных параметров;
- `response-message` — имя исходящего сообщения;
- `response-wrapper` — имя оберточного элемента для исходящего сообщения;
- `soap-action` — URI-идентификатор значения атрибута `soapAction`.

Элементы `<parameter>` и `<return>` являются дочерними элементами элемента `<operation>`, имеют те же атрибуты, что и BindGen-элемент `<value>`, а также атрибуты `element` (имя соответствующего XML-элемента) и `name` (имя параметра).

Атрибуты глобального уровня также могут передаваться в командную строку инструмента Jibx2Wsdl без создания конфигурационного Jibx2Wsdl-файла.

Инструмент CodeGen отвечает за генерацию Java-кода и документа JiBX-связывания из существующей XML-схемы. Инструмент CodeGen работает как Java-приложение `org.jibx.schema.codegen.CodeGen` библиотеки jibx-tools.jar и может запускаться Java-инструментом командной строки `java`.

Инструмент CodeGen имеет следующие параметры командной строки:

- `-b` *`name`* — имя документа JiBX-связывания;
- `-c` *`path`* — расположение конфигурационного CodeGen-файла;
- `-i` *`path1`*`,`*`path2`*`,...` — существующие JiBX-связывания;
- `-n` *`package`* — пакет по умолчанию без пространства имен;
- `-p` *`package`* — пакет по умолчанию;

- `-s` *`path`* — каталог XML-схем;
- `-t` *`path`* — исходящий каталог;
- `-u` *`uri`* — пространство имен для XML-схемы без пространства имен;
- `-v` — вывод подробной информации при работе инструмента;
- `-w` — очистка исходящей каталога перед выводом.

Корневым элементом конфигурационного CodeGen-файла является элемент `<schema-set>`, содержащий элементы `<schema>`. Элемент `<schema>` определяет генерацию из определенной схемы с помощью дочерних элементов `<attribute>`, `<complexType>`, `<element>` и др., соответствующих компонентам XML-схемы.

Элементы `<schema-set>` и `<schema>` могут иметь следующие общие атрибуты:

- `binding-file-name` — имя генерируемого документа JiBX-связывания;
- `binding-per-schema` — если `true` (по умолчанию), тогда для каждой XML-схемы генерируется свой документ JiBX-связывания;
- `delete-annotations` — если `true` (по умолчанию), тогда из документации удаляются аннотации;
- `enumeration-type` — тип Java-класса, генерируемого для перечислений; возможные значения: `java5` (по умолчанию) и `simple`;
- `generate-all` — если `true` (по умолчанию), тогда обрабатываются все компоненты XML-схемы, включая неиспользуемые;
- `import-docs` — если `true` (по умолчанию), тогда аннотации `xs:documentation` конвертируются в Javadoc-документацию;
- `line-width` — максимальная длина строки в генерируемом Java-коде;
- `package` — имя Java-пакета;
- `prefer-inline` — если `true`, тогда, где возможно, генерируется inline-код, по умолчанию `false`;
- `prefix` — префикс пространства имен;
- `repeated-type` — представление повторяющихся элементов XML-схемы в Java-коде. Возможные значения: `array`, `list` и `typed`;
- `show-schema` — если `true` (по умолчанию), тогда включаемые фрагменты схемы соответствуют генерируемому коду документации;
- `structure-optional` — если `true` (по умолчанию), тогда необязательные ссылки становятся необязательными в связывании;
- `use-inner` — если `true` (по умолчанию), тогда используются внутренние классы для вложенных структур.

Индивидуальные атрибуты элемента `<schema-set>`:

- `names` — список шаблонов имен схем, включаемых в набор;
- `namespaces` — список пространств имен, включаемых в набор.

Индивидуальные атрибуты элемента `<schema>`:

- `excludes` — список компонентов XML-схемы, исключаемых из обработки;
- `includes` — список компонентов XML-схемы, включаемых в обработку;
- `name` — имя обрабатываемой схемы;
- `namespace` — пространство имен обрабатываемой схемы.

Общие атрибуты дочерних элементов CodeGen-файла:

- `any-handling` — определяет представление `xs:any` в Java-коде. Возможные значения: `discard`, `dom`, `mapped`;
- `choice-exposed` — если `true`, тогда `xs:choice` представляется методом `stateXXX()`, если `false` (по умолчанию) — методом `ifXXX()`;
- `choice-handling` — определяет реализацию `xs:choice`. Возможные значения: `stateless`, `checkset`, `checkboth`, `overset`, `overboth`;
- `enforced-facets` и `ignored-facets` — определяют реализацию `xs:simpleType`;
- `union-exposed` и `union-handling` — определяют реализацию `xs:union`;
- `type-substitutions` — определяет замену типов.

Атрибуты элементов, определяющих обработку компонентов XML-схемы:

- `path` — Xpath-путь компонента XML-схемы;
- `position` — номер позиции компонента XML-схемы;
- `class-name` — имя Java-класса, представляющего компоненты `all`, `attribute`, `attributeGroup`, `choice`, `complexType`, `element`, `group`, `sequence`, `simpleType`;
- `exclude` — если `true`, тогда исключает компоненты `all`, `any`, `anyAttribute`, `attribute`, `attributeGroup`, `choice`, `complexType`, `element`, `group`, `sequence`, `simpleType` из обработки, по умолчанию `false`;
- `ignore` — если `true`, тогда компонент `attribute` или `element` игнорируется при демаршализации, по умолчанию `false`;
- `inline` — кодировка компонентов, возможные значения: `default`, `block` и `prefer`;
- `name` — имя компонентов `attribute`, `attributeGroup`, `complexType`, `element`, `group`, `simpleType`;
- `type` — замена типа компонентов `attribute`, `complexType`, `element`, `simpleType`;
- `value-name` — имя Java-свойства, представляющего компоненты `all`, `attribute`, `attributeGroup`, `choice`, `element`, `group`, `sequence`.

Атрибуты глобального уровня могут передаваться в командную строку инструмента CodeGen без создания конфигурационного CodeGen-файла.

Элементы `<schema-set>` и `<schema>` CodeGen-файла также могут содержать дочерние элементы расширений `<name-converter>`, `<class-decorator>` и `<schema-type>`.

Элемент `<name-converter>` определяет конвертацию XML-имен в Java-имена с помощью следующих атрибутов:

- `field-prefix` — префикс, добавляемый к именам полей;
- `field-suffix` — суффикс, добавляемый к именам полей;
- `static-prefix` — префикс, добавляемый к именам статических полей;
- `static-suffix` — суффикс, добавляемый к именам статических полей;
- `strip-prefixes` — список префиксов, удаляемых из XML-имен при конвертации;
- `strip-suffixes` — список суффиксов, удаляемых из XML-имен при конвертации;
- `class` — имя класса, реализующего интерфейс `org.jibx.schema.codegen.extend.NameConverter`.

Элемент `<class-decorator>` расширяет генерацию Java-кода, указывая с помощью атрибута `class` имя класса, реализующего интерфейс `org.jibx.schema.codegen.extend.ClassDecorator`.

Элемент `<schema-type>` определяет генерацию Java-кода для XML-типов данных с помощью следующих атрибутов:

- `check-method` — имя статического метода класса для проверки соответствия текстовой строки типу данных;
- `deserializer` — имя статического метода класса для конвертации текстовой строки в Java-тип;
- `format-name` — имя JiBX-формата, используемого для конвертации XML-типа в Java-тип;
- `java-class` — имя класса Java-типа;
- `serializer` — имя статического метода класса для конвертации Java-типа в текстовую строку;
- `type-name` — имя XML-типа.

Платформа Axis2 обеспечивает поддержку не полностью всей системы JiBX. Папка lib каталога платформы Axis2 содержит только библиотеку jibx-bind.jar JiBX-компилятора и библиотеку jibx-run.jar JiBX-среды выполнения связывания. Axis2-инструмент WSDL2Java с опцией `-d jibx` использует Axis2-библиотеку axis2-jibx.jar для генерации артефактов на стороне клиента и сервера на основе уже существующих Java-классов, представляющих данные, и документа JiBX-связывания. Расположение документа JiBX-связывания для инструмента WSDL2Java указывается с помощью параметра командной строки `-Ebindingfile {file}`.

Таким образом, для использования системы связывания JiBX совместно с платформой Axis2 необходимо произвести следующие действия:

1. Создать Java-классы, представляющие данные, и документ JiBX-связывания с помощью инструмента CodeGen JiBX-дистрибутива на основе XML-схемы WSDL-описания Web-сервиса.
2. Сгенерировать артефакты инструментом WSDL2Java на основе созданных Java-классов данных и документа JiBX-связывания.

3. Скомпилировать Java-классы данных JiBX-компилятором, добавляющим байт-код реализации процессов маршализации/демаршализации JiBX-среды выполнения связывания.

JAXB

Платформа Axis2 предоставляет возможность использовать технологию JAXB отдельно от стандарта JAX-WS как альтернативную систему связывания для генерации артефактов на стороне сервера и клиента с применением инструмента WSDL2Java. Поддержка системы JAXB осуществляется пакетом `org.apache.axis2.jaxbri`, работа которого включается опцией `-jaxbri` инструмента WSDL2Java.

В случае использования инструмента WSDL2Java для генерации Java-кода из WSDL-описания Web-сервиса на основе системы связывания данных JAXB создаются JAXB-классы, промаркированные JAXB-аннотациями, при этом инструмент WSDL2Java вызывает JAXB-компилятор XJC, включенный в JAXB-реализацию, который собственно и генерирует JAXB-классы из XML-схемы WSDL-документа.

Недостатком такой компиляции является невозможность применения богатого набора опций самого XJC-компилятора, т. к. компиляция определяется опциями Axis2-инструмента WSDL2Java. Кроме того, работа инструмента WSDL2Java совместно с системой JAXB осуществляется более корректно для раздельных WSDL-документа и XML-схем.

Поддержка MTOM

Платформа Axis2 обеспечивает поддержку механизма MTOM оптимизации передачи данных на уровне модели документа AXIOM.

Объекты `org.apache.axiom.om.OMText`, представляющие символьные данные XML-документа, могут содержать бинарные данные в форме объекта `javax.activation.DataHandler`. Переключение данных объекта `OMText` между бинарной формой и XML-представлением осуществляется методом `void setOptimize(boolean value)` интерфейса `OMText`.

При создании объекта `OMText` метод `createOMText()` интерфейса `org.apache.axiom.om.OMFactory` также позволяет установить бинарную или XML-форму представления данных своим параметром — флагом `boolean optimize`.

Для включения поддержки MTOM платформой Axis2 на стороне клиента необходимо установить соответствующее свойство объекта `org.apache.axis2.client.Options`:

```
Options options = new Options();
options.setProperty(Constants.Configuration.ENABLE_MTOM,
                    Constants.VALUE_TRUE);
```

Входящие оптимизированные сообщения Web-сервиса автоматически десериализуются средой выполнения Axis2 должным образом. Для оптимизации исходящих

сообщений на стороне сервера требуется модифицировать конфигурационный файл axis2.xml, изменив следующий элемент:

```
<parameter name="enableMTOM">true</parameter>
```

Доступ к бинарным данным объекта `OMText` также осуществляется через объект `DataHandler`, получить который можно методом `getDataHandler()` интерфейса `OMText`.

Рассмотрим пример создания Web-сервиса и клиента Web-сервиса, обменивающихся между собой MTOM-оптимизированными сообщениями.

1. Для создания Web-сервиса откроем среду Eclipse, в меню **File** выберем пункты **New | Dynamic Web Project** и введем имя проекта MTOMWebService. Нажав кнопку **New Runtime**, выберем **Apache Tomcat v7.0**, в раскрывающемся списке **Dynamic web module version** выберем значение **2.5** и нажмем кнопку **Finish**.
2. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Axis2 Preferences**. На вкладке **Axis2 Runtime**, нажав кнопку **Browse**, определим каталог предварительно установленной среды выполнения Apache Axis2 (Binary Distribution, **http://axis.apache.org/axis2/java/core/download.cgi**).
3. В меню **Window** среды Eclipse выберем пункты **Preferences | Web Services | Server and Runtime**. В раскрывающемся списке **Server runtime** выберем **Tomcat v7.0 Server**, в раскрывающемся списке **Web service runtime** — значение **Apache Axis2** и нажмем кнопку **OK**.
4. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **MTOMWebService** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Axis2 Web Services** и нажмем кнопку **OK**.

   В результате среда выполнения Axis2 будет интегрирована в проект MTOMWebService как Web-приложение сервера Tomcat.
5. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **MTOMWebService** и выберем пункты **New | Class**, введем имя Web-сервиса MTOMService и имя пакета `mtomservice` и нажмем кнопку **Finish**.
6. Дополним код класса `MTOMService`, как показано в листинге 6.8.

Листинг 6.8. Код класса `MTOMService`

```
package mtomservice;
import org.apache.axiom.om.*;

public class MTOMService  {
  public OMElement getImage(OMElement root) throws Exception {
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://mtomservice",
      "ns1");
    OMElement rootresponse = factory.createOMElement("getImageResponse", ns1);
    OMElement image = factory.createOMElement("image", ns1);
    OMElement hello = factory.createOMElement("hello", ns1);
```

```
    OMText requestData = (OMText) (root.getFirstOMChild());
    OMText responseData=factory.createOMText(
      "Hello "+requestData.getText());
    responseData.setOptimize(false);
    hello.addChild(responseData);
    java.net.URL url=this.getClass().getResource("resources/image.jpg");
    javax.activation.DataHandler dataHandler =
      new javax.activation.DataHandler(url);
    OMText data = factory.createOMText(dataHandler, true);
    image.addChild(data);
    rootresponse.addChild(hello);
    rootresponse.addChild(image);
    return rootresponse;
}}
```

Web-сервис MTOMService обрабатывает входящие и исходящие сообщения на уровне модели AXIOM.

Метод `getImage()` Web-сервиса MTOMService формирует исходящее сообщение, включающее в себя два элемента: `hello` и `image`. Элемент `hello` содержит текстовую строку, созданную на основе данных входящего сообщения. XML-форма представления данной текстовой строки устанавливается методом `setOptimize(true)` соответствующего `OMText`-объекта. Элемент `image` содержит ссылку `<xop:Include>` на вложение SOAP-сообщения с бинарными данными передаваемого изображения. Бинарная форма представления данных изображения устанавливается флагом `true` метода `createOMText()` при создании на основе `DataHandler`-объекта соответствующего `OMText`-объекта.

В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **MTOMWebService** и выберем пункты **New | Other | Web Services | Web Service**, нажмем кнопку **Next**, в строке **Service implementation** введем `mtomservice.MTOMService` и нажмем кнопку **Finish**.

В результате среда Eclipse добавит в папку services среды выполнения Axis2 класс-файл Web-сервиса MTOMService и его сгенерированный дескриптор развертывания services.xml.

Дескриптор services.xml Web-сервиса MTOMService указывает в качестве Axis2 Handler-обработчика класс `org.apache.axis2.rpc.receivers.RPCMessageReceiver`. Так как Web-сервис MTOMService обрабатывает сообщения на уровне модели AXIOM, то для него должен быть определен Handler-обработчик `org.apache.axis2.receivers.RawXMLINOutMessageReceiver`, оперирующий объектами `OMElement`. Конфигурационный файл axis2.xml устанавливает на глобальном уровне для MEP-шаблона In-Out-обработчик `RawXMLINOutMessageReceiver`, поэтому дескриптор services.xml Web-сервиса MTOMService примет следующий вид:

```
<service name="MTOMService" >
<parameter name="ServiceClass"
          locked="false">mtomservice.MTOMService</parameter>
</service>
```

Для включения поддержки механизма MTOM изменим конфигурационный файл axis2.xml узла **WebContent\WEB-INF\conf** проекта MTOMWebService:

```
<parameter name="enableMTOM">true</parameter>
```

А затем заново развернем Web-приложение MTOMWebService на сервере Tomcat.

1. Для создания клиента Web-сервиса MTOMService в перспективе Java среды Eclipse в меню **File** выберем пункты **New | Java Project**, введем имя проекта MTOMWebServiceClient и нажмем кнопку **Finish**.
2. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **MTOMWebServiceClient** и выберем пункт **Properties**. В появившемся диалоговом окне выберем **Project Facets**, укажем **Dynamic Web Module 2.5**, **Axis2 Web Services** и нажмем кнопку **OK**.
3. В окне **Project Explorer** щелкнем правой кнопкой мыши на узле **MTOMWebServiceClient** и выберем пункты **New | Class**, введем имя класса `MTOMClient` и имя пакета `mtomclient`, укажем опцию **public static void main(String[] args)** и нажмем кнопку **Finish**.
4. Изменим код класса `MTOMClient`, как показано в листинге 6.9.

Листинг 6.9. Код класса `MTOMClient`

```
package mtomclient;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class MTOMClient {
  public static void main(String[] args) throws Exception {
    ServiceClient sc = new ServiceClient();
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://mtomservice",
      "ns1");
    OMElement root = factory.createOMElement("getImage", ns1);
    OMText data = factory.createOMText("User");
    data.setOptimize(true);
    root.addChild(data);
    Options op = new Options();
    op.setProperty(org.apache.axis2.Constants.Configuration.ENABLE_MTOM,
      org.apache.axis2.Constants.VALUE_TRUE);
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
      "http://127.0.0.1:1234/MTOMWebService/services/MTOMService"));
    op.setAction("urn:getImage");
    sc.setOptions(op);
    OMElement res = sc.sendReceive(root);
    System.out.println(res);
    OMText binaryNode = (OMText) (res.getFirstChildWithName(new
      javax.xml.namespace.QName("http://mtomservice","image"))).
        getFirstOMChild();
```

```
    javax.activation.DataHandler dh =
     (javax.activation.DataHandler) binaryNode.getDataHandler();
    dh.writeTo(System.out);
}}
```

В методе `main()` класса `MTOMClient` клиент Web-сервиса создается на основе программного интерфейса ServiceClient API. С помощью OM API формируется SOAP-запрос Web-сервису, содержащий в теле элемент `getImage` с текстовыми данными, передаваемыми в бинарном формате как вложение SOAP-сообщения. Бинарная форма представления текстовых данных устанавливается методом `setOptimize(true)` соответствующего `OMText`-объекта.

Поддержка механизма MTOM для клиента Web-сервиса включается определением свойства `org.apache.axis2.Constants.Configuration.ENABLE_MTOM` объекта `Options`.

Доступ к бинарным данным ответного сообщения Web-сервиса осуществляется через объект `DataHandler`, извлекаемый из объекта `OMText`, представляющего содержимое элемента `image` тела SOAP-сообщения.

В данном примере используется инструмент TCPMon для просмотра входящих и исходящих сообщений Web-сервиса, поэтому в методе `main()` класса `MTOMClient` адрес конечной точки вызываемого Web-сервиса указывает порт 1234, а не порт 8080.

После запуска класса `MTOMClient` как Java-приложения выбором в окне **Project Explorer** среды Eclipse пунктов **Run As | Java Application** на вкладке **Port 1234** окна **TCPMon** можно увидеть входящее и исходящее сообщения Web-сервиса (рис. 6.8).

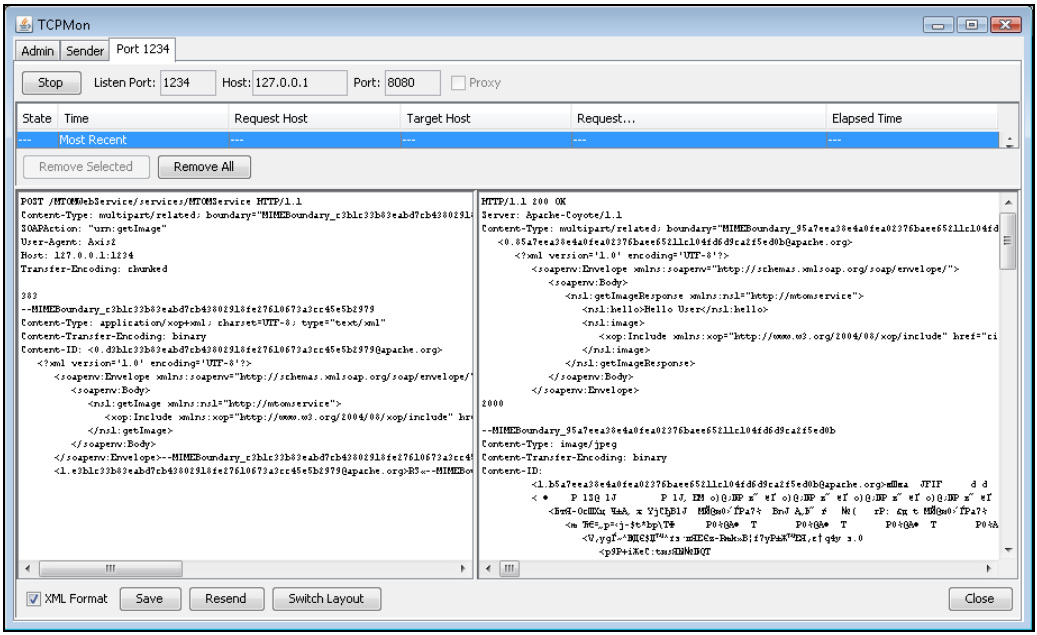


Рис. 6.8. Входящее и исходящее MTOM-оптимизированные сообщения Web-сервиса

Меняя флаги `optimize` с `true` на `false` можно видеть, как данные передаются в XML-формате, а не вложением с бинарными данными (рис. 6.9).

На компакт-диске

Проекты MTOMWebService и MTOMWebServiceClient находятся в папке Примеры\Глава6\MTOM компакт-диска.

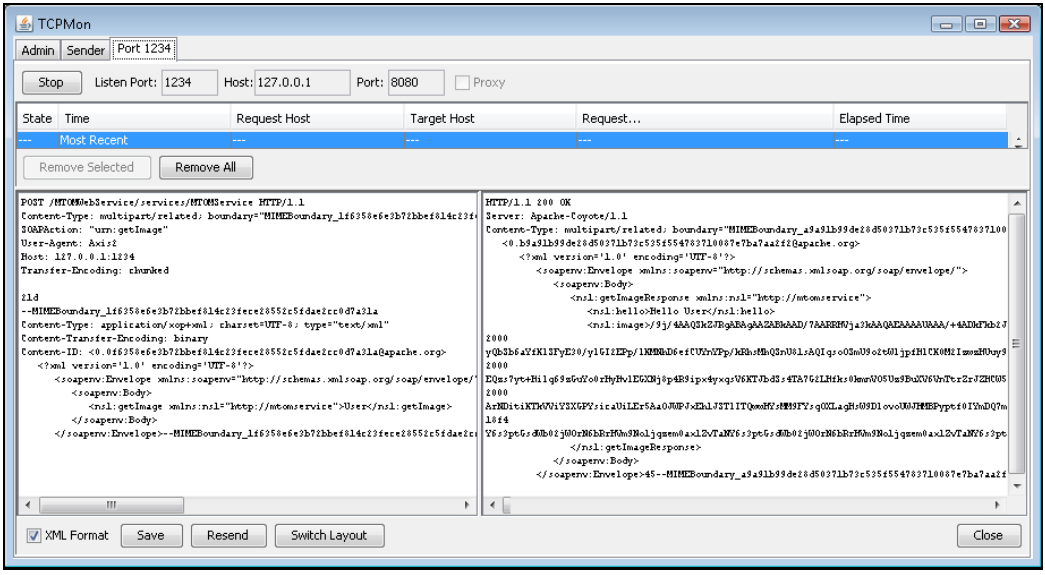


Рис. 6.9. Входящее и исходящее неоптимизированные сообщения Web-сервиса

Поддержка протокола HTTPS

При создании в среде Eclipse Axis2 Web-сервиса среда выполнения Axis2 интегрируется в проект в качестве Web-приложения. При этом в дескрипторе развертывания Web-приложения web.xml в качестве получателя HTTP-запросов фигурирует сервлет `org.apache.axis2.transport.http.AxisServlet`, помимо обработки HTTP-запросов отвечающий за загрузку конфигурационного файла axis2.xml, Axis2-модулей и развертывание Web-сервисов в каталоге services с обеспечением доступа к их WSDL-документам по запросу `?wsdl` конечной точки.

Реализацию интерфейса `org.apache.axis2.transport.TransportListener` для сервлета AxisServlet представляет класс `org.apache.axis2.transport.http.AxisServletListener`, поэтому для обработки HTTP-запросов средой выполнения Axis2, развернутой в Servlet-контейнере, необходимо в конфигурационном файле axis2.xml определить в качестве получателя сообщений не класс `org.apache.axis2.transport.http.SimpleHTTPServer`, а класс `AxisServletListener`.

Платформа Axis2 обеспечивает поддержку протокола SSL с помощью сервлета AxisServlet и обработчика `AxisServletListener`, генерируя при развертывании Web-сервиса WSDL-документ, содержащий конечную точку протокола HTTPS.

Для обработки входящих сообщений по протоколу HTTP и HTTPS средой выполнения Axis2, развернутой в Servlet-контейнере, необходимо в конфигурационный файл axis2.xml добавить элементы:

```
<transportReceiver name="http"
     class="org.apache.axis2.transport.http.AxisServletListener">
  <parameter name="port">8080</parameter>
</transportReceiver>

<transportReceiver name="https"
     class="org.apache.axis2.transport.http.AxisServletListener">
  <parameter name="port">8443</parameter>
</transportReceiver>
```

Теперь при развертывании Web-сервиса WSDL-документ будет содержать конечные точки:

```
<soap:address
 location="https://localhost:8443/[...]HttpsSoap11Endpoint/"/>
  <soap12:address
   location="https://localhost:8443/[...]HttpsSoap12Endpoint/"/>
<http:address location="https://localhost:8443/[...]HttpsEndpoint/"/>
```

Отправку сообщений по HTTPS-протоколу обеспечивает элемент конфигурационного файла axis2.xml:

```
<transportSender name="https"
     class="org.apache.axis2.transport.http.CommonsHTTPTransportSender">
  <parameter name="PROTOCOL">HTTP/1.1</parameter>
  <parameter name="Transfer-Encoding">chunked</parameter>
</transportSender>
```

Если среда выполнения Axis2 разворачивается поверх сервера Tomcat, тогда для взаимодействия по протоколу HTTPS требуется установить соответствующую конфигурацию сервера Tomcat.

Для поддержки сервером Tomcat протокола HTTPS необходимо в качестве первого шага создать хранилище ключей с парой "публичный ключ — приватный ключ". Для этого можно воспользоваться Java-инструментом командной строки `keytool` или инструментом с графическим интерфейсом KeyTool IUI (h**ttp://code.google.com/p/keytool-iui/**). После создания хранилища ключей требуется раздокументировать и изменить следующий элемент конфигурационного файла server.xml папки conf каталога сервера Tomcat:

```
<Connector protocol="HTTP/1.1" port="8443" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           keystoreFile="C:/keys/keystore.jks"
           keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS" />
```

После изменения файла server.xml дистрибутива сервера Tomcat в среде Eclipse нужно удалить и заново создать локальный сервер Tomcat или изменить файл server.xml узла **Servers | Tomcat Server** окна **Project Explorer** среды Eclipse.

На компакт-диске

Проект примера Axis2 Web-сервиса и клиента, взаимодействующих по HTTPS-протоколу, находится в папке Примеры\Глава6\HTTPS компакт-диска как Eclipse-проект HTTPSAxis2.

Проект HTTPSAxis2 содержит пакет `client` с классом `Client`, представляющим клиента Web-сервиса (листинг 6.10).

Листинг 6.10. Код класса `Client` проекта HTTPSAxis2

```
package client;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class Client {
  public static void main(String[] args)throws Exception  {
    System.setProperty("javax.net.ssl.trustStore",
                       "C:/keys/truststore.jks");
    System.setProperty("javax.net.ssl.trustStorePassword", "changeit");
    ServiceClient sc = new ServiceClient();
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://service", "ns1");
    OMElement root = factory.createOMElement("getHello", ns1);
    OMElement name = factory.createOMElement("name", ns1);
    name.setText("User");
    root.addChild(name);
    System.out.println(root.toString());
    Options op = new Options();
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
"https://localhost:8443/HTTPSAxis2/services/Service.ServiceHttpsSoap12Endpoint/
"));
    op.setAction("urn:getHello");
    sc.setOptions(op);
    OMElement res = sc.sendReceive(root);
    System.out.println(res.getFirstElement().getText());
}}
```

В методе `main()` класса `Client` в качестве конечной точки Web-сервиса указана конечная точка протокола HTTPS, а для поддержки протокола HTTPS определены JSSE системные свойства `javax.net.ssl.trustStore` и `javax.net.ssl.trustStore Password`.

Доверительное хранилище ключей на стороне клиента должно содержать доверенный сертификат, сгенерированный на основе хранилища ключей сервера.

HttpClient и аутентификация

Платформа Axis2 интегрирована с программным интерфейсом HttpClient API проекта Apache HttpComponents (**http://hc.apache.org/index.html**). Пакет `org.apache.axis2.transport.http` платформы Axis2 основывается на интерфейсе HttpClient API в части реализации отправки сообщений по протоколу HTTP. Конфигурационный файл axis2.xml своим элементом `<transportSender>` по умолчанию определяет в качестве класса обработки транспортного протокола исходящих сообщений класс `org.apache.axis2.transport.http.CommonsHTTPTransportSender`, использующий библиотеку commons-httpclient.jar программного интерфейса HttpClient API. Кроме того, платформа Axis2 этой библиотекой обеспечивает применение программного интерфейса HttpClient API для создания клиента Web-сервиса.

Программный интерфейс HttpClient API проекта Apache HttpComponents является реализацией HTTP-стандартов и упрощает создание клиентских приложений, использующих HTTP-протокол. Библиотека HttpClient обеспечивает на стороне клиента аутентификацию, управление HTTP-состоянием и HTTP-соединением, реализуя HTTP-методы GET, POST, PUT, DELETE, HEAD, OPTIONS и TRACE, поддерживая протоколы HTTP и HTTPS, различные схемы аутентификации, такие как Basic, Digest и NTLM, и другие возможности.

Программный интерфейс HttpClient API может быть использован в коде клиентского приложения платформы Axis2 различными способами.

Косвенным образом интерфейс HttpClient API применяется при определении свойств объекта `Options`, представленных классом `org.apache.axis2.transport.http.HTTPConstants`:

```
options.setProperty(HTTPConstants.[свойство], [значение свойства]);
```

Это позволяет установить такие свойства HTTP-протокола, как `SO_TIMEOUT`, `CONNECTION_TIMEOUT`, `CHUNKED`, `PROXY`, `AUTHENTICATE`, `REUSE_HTTP_CLIENT`, `CACHED_HTTP_CLIENT` и многие другие.

Прямым способом программный интерфейс HttpClient API может быть использован для создания объекта `org.apache.commons.httpclient.HttpClient`, своим методом `executeMethod()` обеспечивающего исполнение HTTP-запроса Web-сервису.

Рассмотрим пример базовой аутентификации клиента Web-сервиса платформы Axis2, обеспеченной интерфейсом HttpClient API.

1. Для создания защищенного Axis2 Web-сервиса создадим в среде Eclipse динамический Web-проект Axis2Authentication с интегрированной средой выполнения Axis2, разворачиваемой в сервере Tomcat.
2. Для ограничения доступа к Web-сервису изменим дескриптор web.xml, расположенный в папке WebContent\WEB-INF проекта, добавив следующие элементы:

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>default</realm-name>
</login-config>
```

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Service</web-resource-name>
    <url-pattern>/services/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>tomcat</role-name>
  </auth-constraint>
</security-constraint>
```

Теперь для определенного URL-шаблона запроса доступ будет ограничен указанной ролью сервера.

3. Для определения логина/пароля указанной роли сервера Tomcat необходимо разкомментировать содержимое конфигурационного файла tomcat-users.xml:

```
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
<user username="role1" password="tomcat" roles="role1"/>
```

4. После создания в проекте Axis2Authentication Web-сервиса service.Service с методом `getHello()` с помощью выбора пунктов **New | Other | Web Services | Web Service** и развертыванием Web-сервиса, используя пункты **Run As | Run on Server**, доступ к ссылке **Services** на странице по адресу **http://localhost:8080/Axis2Authentication/** окажется ограниченным и будет требовать введения логина/пароля.
5. Для создания клиента Web-сервиса, устанавливающего свою аутентификацию с помощью определения свойств объекта `Options`, создадим в проекте Axis2Authentication класс `client.Client` с методом `main()` (листинг 6.11).

Листинг 6.11. Код класса `Client` проекта Axis2Authentication

```
package client;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class Client {
  public static void main(String[] args)throws Exception  {
    ServiceClient sc = new ServiceClient();
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://service", "ns1");
    OMElement root = factory.createOMElement("getHello", ns1);
    OMElement name = factory.createOMElement("name", ns1);
    name.setText("User");
    root.addChild(name);
    System.out.println(root.toString());
    Options op = new Options();
```

```
    org.apache.axis2.transport.http.
      HttpTransportProperties.Authenticator authenticator =
        new org.apache.axis2.transport.http.
          HttpTransportProperties.Authenticator();
    authenticator.setUsername("tomcat");
    authenticator.setPassword("tomcat");
    op.setProperty(org.apache.axis2.transport.http.
      HTTPConstants.AUTHENTICATE, authenticator);
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
      "http://localhost:8080/Axis2Authentication/services/Service"));
    op.setAction("urn:getHello");
    sc.setOptions(op);
    OMElement res = sc.sendReceive(root);
    System.out.println(res.getFirstElement().getText());
  }}
```

В методе `main()` класса `Client` логин/пароль авторизированного пользователя сервера устанавливается определением свойства `HTTPConstants.AUTHENTICATE` `Options`-объекта.

После запуска класса `Client` как Java-приложения выбором пунктов **Run As | Java Application** в консоли среды Eclipse можно увидеть успешный результат вызова Web-сервиса.

Для создания клиента Web-сервиса, устанавливающего свою аутентификацию с помощью определения свойств объекта `HttpClient`, создадим в проекте Axis2Authentication класс `client.HTTPClient` с методом `main()` (листинг 6.12).

Листинг 6.12. Код класса `HTTPClient` проекта Axis2Authentication

```
package client;
import org.apache.commons.httpclient.*;
import org.apache.commons.httpclient.auth.*;
public class HTTPClient {
  public static void main(String[] args) throws Exception{
    HttpClient client = new HttpClient();
    UsernamePasswordCredentials creds =
      new UsernamePasswordCredentials("tomcat","tomcat");
    client.getState().setCredentials(AuthScope.ANY, creds);
    org.apache.commons.httpclient.methods.GetMethod method =
      new org.apache.commons.httpclient.methods.GetMethod(
"http://localhost:8080/Axis2Authentication/services/Service/getHello?name=User");
    method.setDoAuthentication(true);
    client.executeMethod(method);
    java.io.InputStream is = method.getResponseBodyAsStream();
    StringBuffer out = new StringBuffer();
    byte[] b = new byte[2048];
```

```
    for (int n; (n = is.read(b)) != -1;) {
      out.append(new String(b, 0, n));
    }
    System.out.println(out.toString());
    method.releaseConnection();
  }}
```

Код метода `main()` класса `HTTPClient` полностью основан на прямом применении программного интерфейса HttpClient API. В методе `main()` класса `HTTPClient` логин/пароль авторизированного пользователя сервера устанавливается с помощью объектов `UsernamePasswordCredentials` и `HttpState`, а вызов Web-сервиса осуществляется с использованием объекта `GetMethod`, созданного на основе REST-запроса Axis2 конечной точки Web-сервиса.

На компакт-диске

Проект Axis2Authentication находится в папке Примеры\Глава6 компакт-диска.

Транспортные протоколы проекта Axis2

Платформа Axis2 поддерживает большой набор транспортных протоколов, таких как HTTP/HTTPS, Local, TCP, JMS, E-Mail, SMS и др.

Стандартный дистрибутив платформы Axis2 своим транспортным модулем обеспечивает поддержку протоколов HTTP/HTTPS и Local. Поддержка других протоколов осуществляется отдельным проектом Axis2 Transports (**http://ws.apache.org/commons/transport/**).

Включение конкретного транспортного протокола, связанного с определенным портом, реализуется с помощью элементов `<transportReceiver>` и `<transportSender>` конфигурационного файла axis2.xml и внесением в путь приложения соответствующих библиотек.

Поддержка протокола HTTP обеспечивается слушателями `org.apache.axis2.transport.http.SimpleHTTPServer` и `org.apache.axis2.transport.http.AxisServletListener` и обработчиком `org.apache.axis2.transport.http.CommonsHTTPTransportSender`, включенными в платформу Axis2 по умолчанию.

Локальный транспорт включается с помощью слушателя `org.apache.axis2.transport.local.LocalTransportReceiver` и обработчика `org.apache.axis2.transport.local.`**`LocalTransportSender`**, также включенных в платформу Axis2 по умолчанию.

Использование других транспортных протоколов, таких как TCP, JMS, E-Mail, SMS, UDP, XMPP, Axis2 Asynchronous Transport, требует подключения соответствующих библиотек. Общий дистрибутив, содержащий весь набор транспортных адаптеров проекта Axis2 Transports, доступен для скачивания по адресу **http://ws.apache.org/commons/transport/download.cgi**.

TCP

Протокол TCP поддерживается слушателем `org.apache.axis2.transport.tcp.TCPServer` и обработчиком `org.apache.axis2.transport.tcp.TCPTransportSender`.

Так как протокол TCP подразумевает передачу SOAP-сообщений без каких-либо заголовков уровня приложения, то для обмена сообщениями между клиентом и Web-сервисом требуется подключение адресации WS-Addressing.

Адрес конечной точки Web-сервиса в случае использования TCP-протокола указывает в качестве схемы "tcp://...".

> ***На компакт-диске***
>
> Пример использования протокола TCP находится в папке Примеры\Глава6 компакт-диска как Eclipse-проект Axis2TCP.

Проект Axis2TCP содержит интегрированную среду выполнения Axis2, разворачиваемую на сервере Tomcat, и два класса — `service.Service` и `client.Client`, представляющих Web-сервис и клиента Web-сервиса соответственно.

Для подключения поддержки протокола TCP на стороне Web-сервиса изменим конфигурационный файл axis2.xml папки Axis2TCP\WebContent\WEB-INF\conf, раскомментировав следующие элементы:

```
<transportReceiver name="tcp"
      class="org.apache.axis2.transport.tcp.TCPServer">
  <parameter name="port">6060</parameter>
</transportReceiver>
<transportSender name="tcp"
     class="org.apache.axis2.transport.tcp.TCPTransportSender"/>
```

Кроме того, необходимо добавить библиотеку axis2-transport-all.jar (**http://ws.apache.org/commons/transport/download.cgi**) в папку WebContent\WEB-INF\lib проекта Axis2TCP.

Теперь после создания Web-сервиса выбором пунктов **New | Other | Web Services | Web Service** и развертывания Web-сервиса с помощью пунктов **Run As | Run on Server** WSDL-документ Web-сервиса будет содержать адреса конечных точек:

```
<soap:address location="tcp://localhost:6060/axis2/services/Service.
ServiceTcpSoap11Endpoint"/>
<soap12:address location="tcp://localhost:6060/axis2/services/Service.
ServiceTcpSoap12Endpoint"/>
```

Так как конфигурационный файл axis2.xml по умолчанию содержит элемент `<module ref="addressing"/>`, то модуль WS-Addressing подключается на стороне Web-сервиса автоматически.

В методе `main()` класса `Client` (листинг 6.13) клиент Web-сервиса создается на основе конфигурационного файла axis2.xml, в котором указано подключение модуля WS-Addressing, слушателя `org.apache.axis2.transport.tcp.TCPServer` и обработчика `org.apache.axis2.transport.tcp.TCPTransportSender`.

Для загрузки модуля WS-Addressing на стороне клиента необходимо в узел **Libraries** проекта Axis2TCP добавить ссылку на библиотеку addressing.mar.

После запуска класса `Client` как Java-приложения в консоли среды Eclipse можно увидеть ответное сообщение от Web-сервиса, передаваемое по TCP-протоколу.

Листинг 6.13. Код класса `Client` проекта Axis2TCP

```
package client;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class Client {
  public static void main(String[] args)throws Exception  {
    org.apache.axis2.context.ConfigurationContext configContext =
      org.apache.axis2.context.ConfigurationContextFactory.
        createConfigurationContextFromFileSystem(
          "C:/Axis2TCP/WebContent/WEB-INF/conf");
    ServiceClient sc = new ServiceClient(configContext,null);
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://service", "ns1");
    OMElement root = factory.createOMElement("getHello", ns1);
    OMElement name = factory.createOMElement("name", ns1);
    name.setText("User");
    root.addChild(name);
    System.out.println(root.toString());
    Options op = new Options();
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
  "tcp://localhost:6060/axis2/services/Service.ServiceTcpSoap12Endpoint"));
    op.setAction("urn:getHello");
    sc.setOptions(op);
    OMElement res = sc.sendReceive(root);
    System.out.println(res);
}}
```

JMS

Для использования JMS-транспорта при обмене сообщениями между клиентом и Web-сервисом требуется участие в качестве посредника JMS-поставщика. Наиболее популярной системой сообщений, применяемой совместно с платформой Axis2, является система Apache ActiveMQ, уже рассмотренная в *разд. "Apache Camel, JMS и Apache ActiveMQ" главы 5*. Далее обсудим использование именно этой системы для реализации JMS-транспорта платформы Axis2.

Включение поддержки Axis2 JMS-транспорта осуществляется с помощью определения элементов `<transportReceiver name="jms" class="org.apache.axis2.transport.jms.JMSListener">` и `<transportSender name="jms" class="org.apache.axis2.transport.jms.JMSSender">` в конфигурационном файле axis2.xml, а также включением в папку lib среды выполнения Axis2 библиотеки axis2-transport-all.jar проекта

Axis2 Transports (или отдельной библиотеки JMS-транспорта) и библиотек activemq-core.jar, geronimo-j2ee-management_spec.jar, geronimo-jms_spec.jar и geronimo-jta_spec.jar проекта Apache ActiveMQ.

Элемент `<transportReceiver>` в данном случае должен содержать дочерние элементы `<parameter>`, устанавливающие конфигурацию объекта `ConnectionFactory`. Проектом Axis2 Transports определены следующие параметры конфигурации `ConnectionFactory`:

- `java.naming.factory.initial` (обязательный) — класс-фабрика JMS-контекста (`org.apache.activemq.jndi.ActiveMQInitialContextFactory`);
- `java.naming.provider.url` (обязательный) — URL-адрес JMS-поставщика (**tcp://localhost:61616**, определяемый в конфигурационном файле activemq.xml);
- `transport.jms.ConnectionFactoryJNDIName` (обязательный) — JNDI-имя `ConnectionFactory` (по умолчанию создаются `TopicConnectionFactory` или `QueueConnectionFactory`);
- `java.naming.security.principal` — JNDI-логин;
- `java.naming.security.credentials` — JNDI-пароль;
- `transport.Transactionality` — режим транзакций. Возможные значения: `none` (по умолчанию), `local` или `jta`;
- `transport.UserTxnJNDIName` — JNDI-имя, используемое для транзакции;
- `transport.CacheUserTxn` — если `true` (по умолчанию), тогда используется кэширование при транзакции;
- `transport.jms.SessionTransacted` — если `true` (по умолчанию), тогда используется транзакция JMS-сессии;
- `transport.jms.SessionAcknowledgement` — режим подтверждения, возможные значения: `AUTO_ACKNOWLEDGE` (по умолчанию), `CLIENT_ACKNOWLEDGE`, `DUPS_OK_ACKNOWLEDGE`, `SESSION_TRANSACTED`;
- `transport.jms.ConnectionFactoryType` — тип `ConnectionFactory`; возможные значения: `queue` (по умолчанию) или `topic`;
- `transport.jms.JMSSpecVersion` — версия JMS, возможные значения: `1.1` (по умолчанию) или `1.0.2b`;
- `transport.jms.UserName` — логин;
- `transport.jms.Password` — пароль;
- `transport.jms.DefaultReplyDestination` — JNDI-имя объекта `Destination`;
- `transport.jms.DefaultReplyDestinationType` — тип объекта `Destination`;
- `transport.jms.MessageSelector` — реализация `MessageSelector`;
- `transport.jms.SubscriptionDurable` — возможные значения: `true` или `true` (по умолчанию);
- `transport.jms.DurableSubscriberName` — имя Durable-подписчика;

- `transport.jms.PubSubNoLocal` — если `true`, тогда для публикации и получения используется один и тот же канал, по умолчанию `false`;
- `transport.jms.CacheLevel` — уровень кэширования, возможные значения: `none`, `connection`, `session`, `consumer`, `producer`, `auto` (по умолчанию);
- `transport.jms.ReceiveTimeout` — время ожидания;
- `transport.jms.ConcurrentConsumers` — количество параллельных потоков;
- `transport.jms.MaxConcurrentConsumers` — максимальное количество параллельных потоков (по умолчанию 1);
- `transport.jms.IdleTaskLimit` — максимальное количество простоев (по умолчанию 10);
- `transport.jms.MaxMessagesPerTask` — максимальное количество сообщений в потоке (по умолчанию неограниченно);
- `transport.jms.InitialReconnectDuration` — максимальное время соединения (по умолчанию 1000 мс);
- `transport.jms.ReconnectProgressFactor` — коэффициент времени пересоединения (по умолчанию 2);
- `transport.jms.MaxReconnectDuration` — максимальное время пересоединения (по умолчанию 3 600 000 мс).

Некоторые глобальные параметры (`transport.jms.ConnectionFactory`, `transport.jms.Destination`, `transport.jms.DestinationType`, `transport.jms.ReplyDestination`, `transport.jms.ContentType`, `Wrapper`) конфигурации `ConnectionFactory` могут быть переопределены в конфигурационном файле services.xml:

```
<service name="...">
  <transports>
    ...
    <transport>jms</transport>
  </transports>
  ...
  <parameter name=". . ." locked="true">. . .</parameter>
  ...
</service>
```

Параметр `transport.jms.ContentType` определяет тип содержимого получаемых сообщений с помощью дочернего элемента `<rules>`:

```
<parameter name="transport.jms.ContentType">
  <rules>
  ...
  </rules>
</parameter>
```

Элемент `<rules>` может содержать следующие дочерние элементы:

- `<jmsProperty>` — свойство сообщения, определяющее его тип (по умолчанию `Content-Type`);

- `<bytesMessage>` — тип сообщения (по умолчанию `application/octet-stream`);
- `<textMessage>` — тип сообщения (по умолчанию `text/plain`);
- `<default>` — тип сообщения по умолчанию.

Параметр `Wrapper` указывает оберточный элемент для сообщений.

При использовании JMS-транспорта адрес конечной точки Web-сервиса имеет определенную схему:

```
"jms:/" jms-dest ["?" param  *(["&" param])]
   param = param-name "=" param-value
```

> ***На компакт-диске***
>
> Пример использования JMS-транспорта находится в папке Примеры\Глава6 компакт-диска как Eclipse-проект Axis2JMS.

Проект Axis2JMS содержит интегрированную среду выполнения Axis2, разворачиваемую на сервере Tomcat, и два класса — `service.Service` и `client.Client`, представляющих Web-сервис и клиента Web-сервиса соответственно.

Для подключения поддержки JMS-транспорта изменим конфигурационный файл axis2.xml папки Axis2JMS\WebContent\WEB-INF\conf, добавив следующие элементы:

```
<transportReceiver name="jms"
    class="org.apache.axis2.transport.jms.JMSListener">
  <parameter name="myTopicConnectionFactory" locked="false">
    <parameter name="java.naming.factory.initial" locked="false">
     org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url" locked="false">
     tcp://localhost:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName"
               locked="false">TopicConnectionFactory</parameter>
  </parameter>
  <parameter name="myQueueConnectionFactory" locked="false">
    <parameter name="java.naming.factory.initial" locked="false">
     org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url" locked="false">
     tcp://localhost:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName"
               locked="false">QueueConnectionFactory</parameter>
  </parameter>
  <parameter name="default" locked="false">
    <parameter name="java.naming.factory.initial" locked="false">
     org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
    <parameter name="java.naming.provider.url" locked="false">
     tcp://0.0.0.0:61616</parameter>
    <parameter name="transport.jms.ConnectionFactoryJNDIName"
               locked="false">QueueConnectionFactory</parameter>
```

```
  </parameter>
</transportReceiver>
<transportSender name="jms"
                 class="org.apache.axis2.transport.jms.JMSSender"/>
```

Кроме того, необходимо добавить библиотеки axis2-transport-all.jar, activemq-core.jar, geronimo-j2ee-management_spec.jar, geronimo-jms_spec.jar и geronimo-jta_spec.jar в папку WebContent\WEB-INF\lib проекта Axis2JMS.

Запустим JMS-поставщика, используя пакетный файл activemq.bat папки bin каталога ActiveMQ.

Теперь после создания Web-сервиса выбором пунктов **New | Other | Web Services | Web Service** и развертывания Web-сервиса с помощью пунктов **Run As | Run on Server** WSDL-документ Web-сервиса будет содержать адреса конечных точек:

```
<soap:address location="jms:/Service?transport.jms.DestinationType=
queue&transport.jms.ContentTypeProperty=Content-Type&java.naming.
provider.url=tcp://0.0.0.0:61616&java.naming.factory.initial=
org.apache.activemq.jndi.ActiveMQInitialContextFactory&transport.
jms.ConnectionFactoryJNDIName=QueueConnectionFactory" />
<soap12:address location="jms:/Service?transport.jms.DestinationType=
queue&transport.jms.ContentTypeProperty=Content-
Type&java.naming.provider.url=tcp://0.0.0.0:61616&java.naming.factory.initial=
org.apache.activemq.jndi.ActiveMQInitialContextFactory&transport.jms.
ConnectionFactoryJNDIName=QueueConnectionFactory" />
```

Листинг 6.14. Код класса `Client` проекта Axis2JMS

```
package client;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class Client {
  public static void main(String[] args)throws Exception  {
    org.apache.axis2.context.ConfigurationContext configContext =
      org.apache.axis2.context.ConfigurationContextFactory.
        createConfigurationContextFromFileSystem(
          "C:/Axis2JMS/WebContent/WEB-INF/conf");
    ServiceClient sc = new ServiceClient(configContext,null);
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://service", "ns1");
    OMElement root = factory.createOMElement("getHello", ns1);
    OMElement name = factory.createOMElement("name", ns1);
    name.setText("User");
    root.addChild(name);
    System.out.println(root.toString());
    Options op = new Options();
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
      "jms:/Service?transport.jms.DestinationType=queue&transport.
jms.ContentTypeProperty=Content-Type&java.naming.provider.url=tcp://
```

```
0.0.0.0:61616&java.naming.factory.initial=org.apache.activemq.jndi.
ActiveMQInitialContextFactory&transport.jms.ConnectionFactoryJNDIName=
QueueConnectionFactory"));
    op.setAction("urn:getHello");
    sc.setOptions(op);
    OMElement res = sc.sendReceive(root);
    System.out.println(res);
}}
```

В методе `main()` класса `Client` (см. листинг 6.14) клиент Web-сервиса создается на основе конфигурационного файла axis2.xml, в котором указано подключение модуля WS-Addressing, слушателя `org.apache.axis2.transport.jms.JMSListener` и обработчика `org.apache.axis2.transport.jms.JMSSender`.

Для загрузки модуля WS-Addressing на стороне клиента необходимо в узел **Libraries** проекта Axis2JMS добавить ссылку на библиотеку addressing.mar.

После запуска класса `Client` как Java-приложения в консоли среды Eclipse можно увидеть ответное сообщение от Web-сервиса, передаваемое по JMS-транспорту.

WS-ReliableMessaging

Платформа Axis2 обеспечивает поддержку спецификации WS-ReliableMessaging с помощью модуля Apache Sandesha2 (**http://axis.apache.org/axis2/java/sandesha/**).

Для скачивания модуль Apache Sandesha2 доступен по адресу **http://axis.apache.org/axis2/java/sandesha/download.cgi**. Для установки модуля Apache Sandesha2 необходимо распаковать скачанный дистрибутив и поместить файл модуля sandesha2.mar в папку modules среды выполнения Axis2, а библиотечные файлы дистрибутива — в папку lib среды выполнения Axis2.

Для активации модуля Apache Sandesha2 на стороне Web-сервиса можно включить элемент `<module ref="sandesha2"/>` в конфигурационный файл services.xml. Для активации модуля Apache Sandesha2 на стороне клиента Web-сервиса необходимо обеспечить выполнение Engage-операции для модулей sandesha2.mar и addressing.mar с помощью конфигурационного файла axis2.xml или программным способом и включить файлы модулей и библиотечные файлы в путь приложения.

Модуль Apache Sandesha2 реализует режим доставки сообщений In-order Exactly-once, где опция сохранения порядка сообщений In-order может быть выключена. Функционирование Sandesha2-модуля обеспечивается его специальными обработчиками, действующими в фазе RMPhase входящего и исходящего процессов обработки сообщений. При этом гарантия доставки сообщений реализуется с помощью хранения сообщений в памяти.

Модуль Apache Sandesha2 предоставляет программный интерфейс Sandesha2 Client API, позволяющий программным способом контролировать работу Sandesha2-модуля на стороне клиента. Модель программирования модуля Sandesha2 состоит из двух частей — управления работой Sandesha2-модуля можно с помощью опре-

деления Sandesha2-свойств объекта `Options` и использования класса `org.apache.sandesha2.client.SandeshaClient`.

Sandesha2-свойства представлены классом `org.apache.sandesha2.client.SandeshaClientConstants`, обеспечивающим определение свойств создаваемой последовательности сообщений. Класс `SandeshaClient` дает возможность управлять последовательностью, программным способом создавая последовательность сообщений, закрывая ее, получать статусную информацию и др.

Файл module.xml Sandesha2-модуля содержит политику, устанавливающую конфигурацию модуля:

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
  xmlns:sandesha2="http://ws.apache.org/sandesha2/policy"
  wsu:Id="RMPolicy">
<sandesha2:RMAssertion>
<wsp:Policy>
<sandesha2:AcknowledgementInterval> 3000</sandesha2:AcknowledgementInterval>
<sandesha2:RetransmissionInterval>6000</sandesha2:RetransmissionInterval>
<!-- '-1' conveys that there is no limit to the max. number of
retransmissions.-->
<sandesha2:MaximumRetransmissionCount>
10</sandesha2:MaximumRetransmissionCount>
<sandesha2:ExponentialBackoff>false</sandesha2:ExponentialBackoff>
<sandesha2:InactivityTimeout>60</sandesha2:InactivityTimeout>
<sandesha2:InactivityTimeoutMeasure>
seconds</sandesha2:InactivityTimeoutMeasure>
<!-- Once a sequence has been marked as deleted, or timed out, this is the
length of time that the sequence will remain before all sequence state is
totally removed -->
<sandesha2:SequenceRemovalTimeout>600</sandesha2:SequenceRemovalTimeout>
<sandesha2:SequenceRemovalTimeoutMeasure>
seconds</sandesha2:SequenceRemovalTimeoutMeasure>
<sandesha2:InvokeInOrder>true</sandesha2:InvokeInOrder>
<!-- These will not be overriden by service level policies -->
<sandesha2:MessageTypesToDrop>none</sandesha2:MessageTypesToDrop>
<!-- This will not be overriden by service level policies -->
<sandesha2:StorageManagers>
<sandesha2:InMemoryStorageManager>
org.apache.sandesha2.storage.inmemory.InMemoryStorageManager
</sandesha2:InMemoryStorageManager>
<sandesha2:PermanentStorageManager>
org.apache.sandesha2.storage.inmemory.InMemoryStorageManager
</sandesha2:PermanentStorageManager>
</sandesha2:StorageManagers>
<!-- This will not be overriden by service level policies -->
<sandesha2:SecurityManager>
org.apache.sandesha2.security.dummy.DummySecurityManager
</sandesha2:SecurityManager>
```

```
<sandesha2:ContextManager> org.apache.sandesha2.context.DummyContextManager
</sandesha2:ContextManager>
<sandesha2:EPRDecorator>
org.apache.sandesha2.addressing.EPRDecoratorImpl</sandesha2:EPRDecorator>
<sandesha2:MakeConnection>
<sandesha2:Enabled>true</sandesha2:Enabled>
<sandesha2:UseRMAnonURI>true</sandesha2:UseRMAnonURI>
</sandesha2:MakeConnection>
<!-- <sandesha2:UseMessageSerialization>
true</sandesha2:UseMessageSerialization> -->
<sandesha2:EnforceRM>false</sandesha2:EnforceRM>
</wsp:Policy>
</sandesha2:RMAssertion>
</wsp:Policy>
```

Большинство свойств, представленных элементами пространства имен `http://ws.apache.org/sandesha2/policy`, может быть переопределено в конфигурационном файле services.xml Web-сервиса (исключение составляет свойство `InvokeInOrder`, которое не может быть переопределено для отдельного сервиса).

Рассмотрим пример надежной и гарантированной доставки сообщений в системе, состоящей из клиента и Web-сервиса платформы Axis2, использующей модуль Apache Sandesha2.

> ***На компакт-диске***
>
> Пример использования WS-ReliableMessaging находится в папке Примеры\Глава6 компакт-диска как Eclipse-проект Axis2RM.

Проект Axis2RM содержит интегрированную среду выполнения Axis2, разворачиваемую на сервере Tomcat, и два класса — `service.Service` и `client.Client`, представляющих Web-сервис и клиента Web-сервиса соответственно.

1. Для подключения поддержки WS-ReliableMessaging изменим конфигурационный файл axis2.xml папки Axis2RM\WebContent\WEB-INF\conf, добавив элемент `<module ref="sandesha2"/>`.
2. Кроме того, необходимо добавить библиотеки дистрибутива Apache Sandesha2 в папку WebContent\WEB-INF\lib проекта Axis2RM.
3. Теперь после создания Web-сервиса выбором пунктов **New | Other | Web Services | Web Service** и развертывания Web-сервиса с помощью пунктов **Run As | Run on Server**, Sandesha2-модуль добавит Web-сервису операции `Sandesha2OperationOutIn`, `Sandesha2OperationDuplicateInOut`, `Sandesha2OperationInOnly`, `Sandesha2OperationOutOnly`, `Sandesha2OperationDuplicateInOnly` и `Sandesha2OperationInOut`, обеспечивающие RM-операции с помощью обработчика org.`apache.sandesha2.msgreceivers.RMMessageReceiver`.

Так как конфигурационный файл axis2.xml по умолчанию содержит элемент `<module ref="addressing"/>`, то модуль WS-Addressing подключается на стороне Web-сервиса автоматически.

Листинг 6.15. Код класса `Client` проекта Axis2RM

```
package client;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;
public class Client {
  public static void main(String[] args)throws Exception  {
    ServiceClient sc = new ServiceClient();
    OMFactory factory = OMAbstractFactory.getOMFactory();
    OMNamespace ns1 = factory.createOMNamespace("http://service", "ns1");
    OMElement root = factory.createOMElement("getHello", ns1);
    OMElement name = factory.createOMElement("name", ns1);
    name.setText("User");
    root.addChild(name);
    System.out.println(root.toString());
    Options op = new Options();
    op.setTo(new org.apache.axis2.addressing.EndpointReference(
      "http://localhost:1234/Axis2RM/services/Service"));
    op.setAction("urn:getHello");
    sc.setOptions(op);
    sc.engageModule(org.apache.axis2.Constants.MODULE_ADDRESSING);
    sc.engageModule("sandesha2");
    op.setProperty(org.apache.sandesha2.client.SandeshaClientConstants.
        LAST_MESSAGE, "true");
    OMElement res = sc.sendReceive(root);
    System.out.println(res);
}}
```

В методе `main()` класса `Client` (листинг 6.15) `ServiceClient`-объект активирует модули addressing и sandesha2 методом `engageModule()` и перед вызовом Web-сервиса указывает с помощью свойства `SandeshaClientConstants.LAST_MESSAGE`, что данное сообщение является последним в создаваемой последовательности.

В данном примере используется инструмент TCPMon для просмотра входящих и исходящих сообщений Web-сервиса, поэтому в методе `main()` класса `Client` адрес конечной точки вызываемого Web-сервиса указывает порт 1234, а не порт 8080.

Для загрузки модулей addressing и sandesha2 на стороне клиента необходимо в узел **Libraries** проекта Axis2RM добавить ссылки на файлы addressing.mar и sandesha2.mar.

После запуска класса `Client` как Java-приложения выбором в окне **Project Explorer** среды Eclipse пунктов **Run As | Java Application** на вкладке **Port 1234** окна **TCPMon** можно увидеть обмен сообщениями между клиентом и Web-сервисом для создания последовательности сообщений, а также клиентский запрос и ответ от Web-сервиса.

WS-Security

Платформа Axis2 обеспечивает безопасность Web-сервисов с помощью модуля Apache Rampart (**http://axis.apache.org/axis2/java/rampart/**).

Модуль Apache Rampart доступен для скачивания по адресу **http://axis.apache.org/axis2/java/rampart/download.html**. Для установки модуля Apache Rampart необходимо распаковать скачанный дистрибутив и поместить файлы модулей rahas.mar и rampart.mar в папку modules среды выполнения Axis2, а библиотечные файлы дистрибутива — в папку lib среды выполнения Axis2.

Для активации модуля Apache Rampart на стороне Web-сервиса можно включить элементы `<module ref="rampart"/>` и `<module ref="rahas"/>` в конфигурационный файл services.xml. Так как Rampart-модуль обеспечивает безопасность Web-сервиса, то дескриптор services.xml может, помимо элементов активации модуля, содержать соответствующую политику Web-сервиса, представленную элементом `<wsp:Policy>`.

Для активации модуля Apache Rampart на стороне клиента Web-сервиса необходимо обеспечить выполнение Engage-операции для модулей rampart.mar и addressing.mar с помощью конфигурационного файла axis2.xml или программным способом и включить файлы модулей и библиотечные файлы в путь приложения.

Для работы Rampart-модуля необходимо определить его конфигурацию как на стороне клиента, так и на стороне Web-сервиса с помощью элемента `RampartConfig` или параметров `InflowSecurity` и `OutflowSecurity`.

Модуль Apache Rampart реализует спецификации WS-Security, WS-SecureConversation, WS-SecurityPolicy, WS-Trust и SAML 2.0 на основе проекта Apache WSS4J (**http://ws.apache.org/wss4j/**). Файл rampart.mar модуля обеспечивает поддержку WS-Security и WS-SecureConversation, а файл rahas.mar — функции STS-сервиса.

Функционирование Rampart-модуля обеспечивается его специальными обработчиками `org.apache.rampart.handler.RampartReciever` и `org.apache.rampart.handler.RampartSender`, действующими в фазе Security входящего и исходящего процессов обработки сообщений.

Как уже было сказано, политика безопасности Axis2 Web-сервиса определяется блоком `<wsp:Policy>` дескриптора services.xml. При этом конфигурация самого Rampart-модуля устанавливается дочерним элементом `<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">` элемента `<wsp:Policy>`. Конфигурационный файл module.xml Rampart-модуля, в отличие от Sandesha2-модуля, не содержит предустановленной конфигурации модуля, в данном случае представленной элементом `<ramp:RampartConfig>` (рис. 6.10).

Элемент `<RampartConfig>` может иметь следующие дочерние элементы:

- `<user>` — имя пользователя UsernameToken-маркера;
- `<userCertAlias>` — сертификат пользователя;

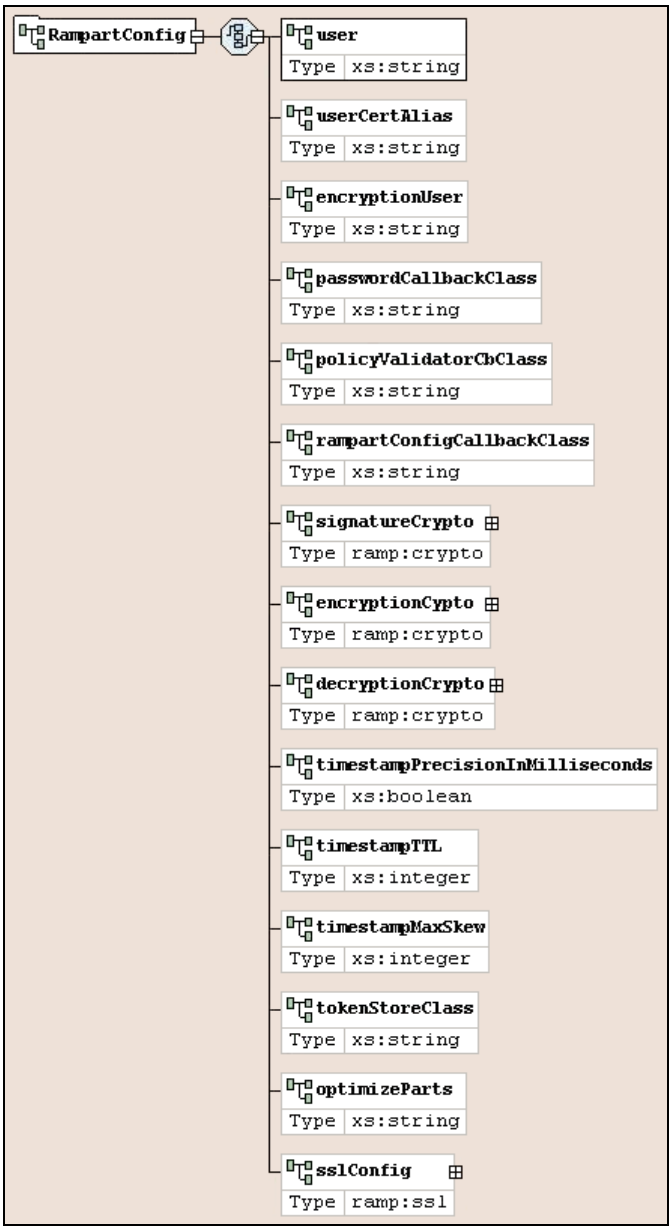


Рис. 6.10. Общая схема элемента `<RampartConfig>`

- `<encryptionUser>` — имя пользователя для шифрования;
- `<passwordCallbackClass>` — класс, обеспечивающий пароль UsernameToken-маркера;
- `<policyValidatorCbClass>` — класс Validator-обработчика;
- `<signatureCrypto>` — класс реализации цифровой подписи, хранилище ключей и пароль хранилища ключей, указываемых с помощью дочернего элемента:

```
<signatureCrypto>
<crypto provider="org.apache.ws.security.components.crypto.Merlin"
        cryptoKey="org.apache.ws.security.crypto.merlin.file"
        cacheRefreshInterval="300000 >
<property name="org.apache.ws.security.crypto.merlin.keystore.type">
  JKS</property>
<property name="org.apache.ws.security.crypto.merlin.file">[].jks
</property>
<property name="org.apache.ws.security.crypto.merlin.keystore.password">
[]</property>
</crypto>
<signatureCrypto>
```

- `<encryptionCypto>` — класс реализации шифрования, хранилище ключей и пароль хранилища ключей, указываемых с помощью дочерних элементов аналогично элементу `signatureCrypto`;
- `<decryptionCrypto>` — класс реализации дешифрования, хранилище ключей и пароль хранилища ключей, указываемых с помощью дочерних элементов аналогично элементу `encryptionCypto`;
- `<timestampTTL>` — значение Timestamp, по умолчанию 300 секунд;
- `<timestampMaxSkew>` — максимальная разница между временем получения сообщения и Timestamp-временем;
- `<timestampPrecisionInMilliseconds>` — если `true`, тогда Timestamp-время содержит миллисекунды;
- `<optimizeParts>` — MTOM-оптимизированные части сообщения;
- `<tokenStoreClass>` — класс, обеспечивающий хранение маркеров защиты;
- `<sslConfig>` — SSL-конфигурация, определяемая с помощью таких свойств, как `javax.net.ssl.trustStore` и `javax.net.ssl.trustStorePassword`.

Rampart-модуль с помощью файла rahas.mar обеспечивает STS-сервис, конфигурация которого определяется в файле services.xml Web-сервиса посредством параметра `saml-issuer-config`:

```
<parameter name="saml-issuer-config">
  <saml-issuer-config>
  ...
  </saml-issuer-config>
</parameter>
```

Наиболее важные дочерние элементы элемента `<saml-issuer-config>`:

- `<issuerName>` — имя STS-сервиса;
- `<issuerKeyAlias>` — сертификат STS-сервиса;
- `<issuerKeyPassword>` — пароль сертификата STS-сервиса;
- `<cryptoProperties>` — класс реализации шифрования, тип хранилища ключей, хранилище ключей и его пароль;

- `<timeToLive>` — время действия SAML-утверждения в секундах;
- `<trusted-services>` — конечные точки Web-сервисов, для которых запрашиваются SAML-утверждения.

Как альтернатива элементу `<wsp:Policy>` с дочерним элементом `<ramp:RampartConfig>` в конфигурационных файлах services.xml на стороне Web-сервиса и axis2.xml на стороне клиента Web-сервиса для определения конфигурации Rampart-модуля могут использоваться параметры `<parameter name="InflowSecurity">` и `<parameter name="OutflowSecurity">`.

> ***На компакт-диске***
>
> Примеры использования модуля Apache Rampart находятся в папке Примеры\Глава6\Rampart компакт-диска.

Eclipse-проект Axis2UsernameToken папки Примеры\Глава6\Rampart демонстрирует аутентификацию клиента с помощью UsernameToken-маркера, передаваемого по HTTPS-протоколу, с использованием параметров `InflowSecurity` и `OutflowSecurity` для конфигурирования Rampart-модуля.

Проект Axis2UsernameToken содержит интегрированную среду выполнения Axis2, разворачиваемую на сервере Tomcat, и два класса — `service.Service` и `client.Client`, представляющих Web-сервис и клиента Web-сервиса соответственно.

Для подключения поддержки протокола HTTPS на стороне Web-сервиса необходимо:

1. В конфигурационный файл axis2.xml добавить элементы:

```
<transportReceiver name="http"
   class="org.apache.axis2.transport.http.AxisServletListener">
  <parameter name="port">8080</parameter>
</transportReceiver>
<transportReceiver name="https"
   class="org.apache.axis2.transport.http.AxisServletListener">
  <parameter name="port">8443</parameter>
</transportReceiver>
```

2. Создать хранилище ключей с парой "публичный ключ — приватный ключ".
3. Установить соответствующую конфигурацию сервера Tomcat, изменив файл server.xml узла **Servers | Tomcat Server** в окне **Project Explorer** среды Eclipse:

```
<Connector protocol="HTTP/1.1" port="8443" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           keystoreFile="C:/keys/keystore.jks"
           keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS" />
```

На стороне клиента требуется добавить следующий код:

```
System.setProperty("javax.net.ssl.trustStore","C:/keys/truststore.jks");
System.setProperty("javax.net.ssl.trustStorePassword","changeit");
```

```
Options op = new Options();
op.setTo(new org.apache.axis2.addressing.EndpointReference(
"https://localhost:8443/Axis2UsernameToken/services/
Service.ServiceHttpsEndpoint/"));
```

Для подключения Rampart-модуля на стороне Web-сервиса необходимо:

1. В папку Axis2UsernameToken\WebContent\WEB-INF\modules добавить файл rampart.mar.
2. В папку Axis2UsernameToken\WebContent\WEB-INF\lib добавить библиотечные файлы Rampart-модуля.
3. В папку Axis2UsernameToken\WebContent\WEB-INF\lib добавить весь набор библиотечных файлов папки lib дистрибутива платформы Axis2.
4. В файл services.xml папки Axis2UsernameToken\WebContent\WEB-INF\services\Service\META-INF добавить:

```
<module ref="rampart" />
  <parameter name="InflowSecurity">
  <action>
    <items>UsernameToken</items>
    <passwordCallbackClass>service.PWCBHandler</passwordCallbackClass>
  </action>
</parameter>
```

5. Создать класс `service.PWCBHandler`, проверяющий пароль UsernameToken-маркера.

Для подключения Rampart-модуля на стороне клиента Web-сервиса требуется:

1. В узел **Libraries** проекта Axis2UsernameToken добавить ссылки на файлы addressing.mar и rampart.mar.
2. Создать клиентский файл axis2.xml и включить в него следующие элементы:

```
<module ref="rampart" />
<module ref="addressing"/>
<parameter name="OutflowSecurity">
  <action>
    <items>UsernameToken</items>
    <user>User</user>
    <passwordCallbackClass>client.PWCBHandler</passwordCallbackClass>
    <passwordType>PasswordText</passwordType>
  </action>
</parameter>
```

3. Создать класс `client.PWCBHandler`, устанавливающий пароль для UsernameToken-маркера.
4. В клиентском коде создать `ServiceClient`-объект на основе клиентского файла axis2.xml.

В результате при запуске класса `Client` как Java-приложения в клиентский запрос Web-сервису будет включен маркер защиты UsernameToken.

Eclipse-проект Axis2UsernameTokenPolicy папки Примеры\Глава6\Rampart демонстрирует аутентификацию клиента с помощью UsernameToken-маркера, передаваемого по HTTPS-протоколу, с использованием элемента `<wsp:Policy>` с дочерним элементом `<ramp:RampartConfig>` для конфигурирования Rampart-модуля.

Проект Axis2UsernameTokenPolicy содержит интегрированную среду выполнения Axis2, разворачиваемую на сервере Tomcat, и два класса — `service.Service` и `client.Client`, представляющих Web-сервис и клиента Web-сервиса соответственно.

Отличие проекта Axis2UsernameTokenPolicy от проекта Axis2UsernameToken состоит в том, что в первом проекте работа Rampart-модуля конфигурируется на стороне Web-сервиса файлом services.xml с помощью элемента `<wsp:Policy>`, который содержит элементы политики, определяющие использование HTTPS-протокола и UsernameToken-маркера для аутентификации клиента, и его дочернего элемента `<ramp:RampartConfig>`, устанавливающего конфигурацию самого Rampart-модуля. После развертывания Web-сервиса его WSDL-документ содержит политику, включенную в файл services.xml.

На стороне клиента Web-сервиса проекта Axis2UsernameTokenPolicy работа Rampart-модуля конфигурируется с помощью файла политики, содержащего элемент `<wsp:Policy>`, дочерние элементы политики которого определяют включение UsernameToken-маркера в клиентский запрос, а дочерний элемент `<ramp:RampartConfig>` устанавливает содержимое UsernameToken-маркера. В коде клиента Web-сервиса внешний файл политики конвертируется в объект `org.apache.neethi.Policy`, который устанавливается как свойство объекта `Options` клиента.

На компакт-диске

Eclipse-проект Axis2SecureConversation папки Примеры\Глава6\Rampart демонстрирует создание защищенной сессии между клиентом и Web-сервисом согласно спецификации WS-SecureConversation.

Проект Axis2SecureConversation содержит интегрированную среду выполнения Axis2, разворачиваемую на сервере Tomcat, и два класса — `service.Service` и `client.Client`, представляющих Web-сервис и клиента Web-сервиса соответственно.

Защищенная сессия создается с участием STS-сервиса, который предоставляется модулем rahas.mar. Для подключения и активации модулей rampart.mar и rahas.mar на стороне Web-сервиса необходимо:

1. В папку Axis2SecureConversation\WebContent\WEB-INF\modules добавить файлы rampart.mar и rahas.mar.
2. В папку Axis2SecureConversation\WebContent\WEB-INF\lib добавить библиотечные файлы Rampart-модуля.
3. В папку Axis2SecureConversation\WebContent\WEB-INF\lib добавить весь набор библиотечных файлов папки lib дистрибутива платформы Axis2.

4. В файл services.xml папки Axis2SecureConversation\WebContent\WEB-INF\services\Service\META-INF добавить элементы `<module ref="rampart"/>` и `<module ref="rahas"/>`, элемент `<wsp:Policy>`, дочерние элементы политики которого определяют шифрование и подпись сообщений с помощью производных ключей SCT-маркера, получаемого у STS-сервиса, а дочерний элемент `<ramp:RampartConfig>` которого определяет конфигурацию модуля rampart.mar, элемент `<parameter name="sct-issuer-config">`, устанавливающий конфигурацию модуля rahas.mar.
5. Создать класс `handler.PWCBHandler`, выдающий пароль для сертификата Web-сервиса.

Элемент `<ramp:RampartConfig>` Web-сервиса указывает хранилище ключей, содержащее пару "публичный ключ — приватный ключ" Web-сервиса, а также доверенные сертификаты клиента и STS-сервиса с их публичными ключами. Кроме того, элемент `<ramp:RampartConfig>` указывает класс `handler.PWCBHandler` для получения пароля доступа к паре "публичный — приватный ключ" Web-сервиса.

Элемент `<parameter name="sct-issuer-config">` Web-сервиса указывает хранилище ключей, содержащее пару "публичный ключ — приватный ключ" STS-сервиса, а также доверенные сертификаты клиента и Web-сервиса с их публичными ключами.

Для подключения Rampart-модуля на стороне клиента Web-сервиса требуется:

1. В узел **Libraries** проекта Axis2SecureConversation добавить ссылки на файлы addressing.mar и rampart.mar.
2. Создать файл policy.xml, определяющий своим элементом `<wsp:Policy>` требование получения SCT-маркера с последующим шифрованием и подписью сообщений его производными ключами и определяющий дочерним элементом `<ramp:RampartConfig>` элемента `<wsp:Policy>` конфигурацию модуля rampart.mar на стороне клиента.
3. В коде класса `Client` обеспечить конвертацию файла policy.xml в объект `org.apache.neethi.Policy` с установкой его в виде свойства объекта `Options` клиента.
4. В коде класса `Client` активировать модули addressing и rampart методом `engageModule()`.

Элемент `<ramp:RampartConfig>` клиента Web-сервиса указывает хранилище ключей, содержащее пару "публичный ключ — приватный ключ" клиента, а также доверенные сертификаты Web-сервиса и STS-сервиса с их публичными ключами. Кроме того, элемент `<ramp:RampartConfig>` указывает класс `handler.PWCBHandler` для получения пароля доступа к паре "публичный ключ — приватный ключ" клиента.

При развертывании Web-сервиса Rampart-модуль добавляет ему операцию `RequestSecurityToken` получения SCT-маркера. После запуска класса `Client` как Java-приложения выбором в окне **Project Explorer** среды Eclipse пунктов **Run As | Java Application** на вкладке **Port 1234** окна **TCPMon** можно увидеть обмен сообщениями между клиентом и Web-сервисом для получения SCT-маркера защиты, а также зашифрованные и подписанные клиентский запрос и ответ от Web-сервиса.

ПРИЛОЖЕНИЕ

Описание электронного архива

В этом приложении дано описание электронного архива к книге, выложенного на FTP-сервер издательства по адресу: **ftp://85.249.45.166/9785977507783.zip**. Ссылка доступна и со страницы книги на сайте **www.bhv.ru**.

Материалы этого архива следует использовать вместо упоминаемых в книге материалов прилагаемого компакт-диска.

В архиве находятся следующие папки:

- папка Приложения содержит приложения, ссылки на которые имеются в тексте книги;
- папка Примеры содержит рассмотренные в книге проекты, перечисленные в табл. П1.

Таблица П1. *Содержимое папки Примеры*

Папка	Вложенная папка	Описание
\Глава 3		
Axis2Eclipse	Axis2_WebService	Пример Apache Axis2 Web-сервиса, созданного с использованием среды разработки Eclipse IDE for Java EE Developers
	Axis2_WebServiceClient	Пример клиента Web-сервиса платформы Apache Axis2, созданного с использованием среды разработки Eclipse IDE for Java EE Developers
Axis2NetBeans	Axis2_WebService	Пример Apache Axis2 Web-сервиса, созданного с использованием среды разработки NetBeans
CXF_WebService		Пример Apache CXF Web-сервиса, созданного с использованием среды разработки Eclipse IDE for Java EE Developers
CXF_WebServiceClient		Пример клиента Web-сервиса платформы Apache CXF, созданного с использованием среды разработки Eclipse IDE for Java EE Developers

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
JAXBExample		Пример использования технологии JAXB, созданный с помощью среды разработки NetBeans
JAXMService		Пример Web-сервиса и клиента Web-сервиса на основе технологий JAXM и SAAJ, созданных с использованием среды разработки Eclipse IDE for Java EE Developers
JAXPExample		Пример использования технологии JAXP, созданный с помощью среды разработки Eclipse IDE for Java EE Developers
JAX_RPC_Example		Пример JAX-RPC Web-сервиса, созданного с использованием среды разработки Eclipse IDE for Java EE Developers
JAX_RPC_ExampleClient		Пример клиента Web-сервиса технологии JAX-RPC, созданного с использованием среды разработки Eclipse IDE for Java EE Developers
JAX_RS_Eclipse	JAX_RS_Example	Пример JAX-RS Web-сервиса, созданного с использованием среды разработки Eclipse IDE for Java EE Developers
JAX_RS_NetBeans	JAX_RS_Example	Пример JAX-RS Web-сервиса, созданного с использованием среды разработки NetBeans
JAX_WS_NetBeans	Async\ JAX_WS_SEClient	Пример Proxy-клиента технологии JAX-WS, асинхронно взаимодействующего с Web-сервисом и созданного с использованием среды разработки NetBeans
	Handler\ JAX_WS_SEClient	Пример клиента Web-сервиса технологии JAX-WS, использующего обработчик сообщений и созданного с помощью среды разработки NetBeans
	Handler\ JAX_WS_WebService	Пример JAX-WS Web-сервиса, использующего обработчик сообщений и созданного с помощью среды разработки NetBeans
	Holder\ JAX_WS_WebService	Пример JAX-WS Web-сервиса, использующего параметры типа `OUT` и `INOUT` и созданного с помощью среды разработки NetBeans
	Holder\ JAX_WS_SEClient	Пример клиента Web-сервиса технологии JAX-WS, использующего параметры типа `OUT` и `INOUT` и созданного с помощью среды разработки NetBeans
	JAX_WS_ DispatchClient	Пример Dispatch-клиента Web-сервиса технологии JAX-WS, созданного с использованием среды разработки NetBeans

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
	JAX_WS_ProviderService	Пример Provider-сервиса технологии JAX-WS, созданного с использованием среды разработки NetBeans
	JAX_WS_SEClient	Пример JAX-WS-клиента платформы Java SE, созданного с использованием среды разработки NetBeans
	JAX_WS_WebClient	Пример клиента Web-сервиса технологии JAX-WS, созданного с использованием среды разработки NetBeans
	JAX_WS_WebService	Пример JAX-WS Web-сервиса, созданного с использованием среды разработки NetBeans
	Security\ JAX_WS_WebService	Пример JAX-WS Web-сервиса, использующего базовую аутентификацию-авторизацию и созданного с помощью среды разработки NetBeans
	Security\JAX_WS_SEClient	Пример клиента Web-сервиса технологии JAX-WS, использующего базовую аутентификацию-авторизацию и созданного с помощью среды разработки NetBeans
\Глава4		
Addressing	Metro_WebService	Metro Web-сервиса, использующего технологию WS-Addressing
MakeConnection	Metro_WebService	Пример Metro Web-сервиса, использующего технологию WS-MakeConnection
	Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего технологию WS-MakeConnection
MTOM	Metro_WebService	Пример Metro Web-сервиса, использующего технологию MTOM
RM	Metro_WebService	Пример Metro Web-сервиса, использующего технологию WS-ReliableMessaging
Security	AsymmetricBinding\ Metro_WebService	Пример Metro Web-сервиса, использующего безопасность совместных сертификатов
	AsymmetricBinding\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего безопасность совместных сертификатов
	Endorsing\ Metro_WebService	Пример Metro Web-сервиса, использующего одобрение сертификата
	Endorsing\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего одобрение сертификата

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
	PasswordValidator\ Metro_WebService	Пример Metro Web-сервиса, использующего проверку подлинности имени пользователя с помощью производного ключа
	PasswordValidator\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего проверку подлинности имени пользователя с помощью производного ключа
	SamlHOKHandler\ Metro_WebService	Пример Metro Web-сервиса, использующего SAML-утверждение
	SamlHOKHandler\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего SAML-утверждение
	SAMLSSL\ Metro_WebService	Пример Metro Web-сервиса, использующего проверку подлинности SAML по SSL
	SAMLSSL\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего проверку подлинности SAML по SSL
	SamlSVHandler\ Metro_WebService	Пример Metro Web-сервиса, использующего подтверждение подлинности отправителя SAML-сертификатом
	SamlSVHandler\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего подтверждение подлинности отправителя SAML-сертификатом
	SecureConversation\ Metro_WebService	Пример Metro Web-сервиса, использующего технологию WS-SecureConversation
	SecureConversation\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего технологию WS-SecureConversation
	SSL\Metro_WebService	Пример Metro Web-сервиса, использующего протокол SSL
	SSL\Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего протокол SSL
	SSL\Metro_SEClient	Пример JavaSE-клиента Web-сервиса платформы Metro, использующего протокол SSL
	STS\Metro_WebService	Пример Metro Web-сервиса, использующего STS-маркер
	STS\Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего STS-маркер

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
	STS\STSProject	Пример STS-сервиса
	STSCertificate\ Metro_WebService	Пример Metro Web-сервиса, использующего STS-маркер с сертификатом службы
	STSCertificate\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего STS-маркер с сертификатом службы
	STSCertificate\STSProject	Пример STS-сервиса
	STSEndorsing\ Metro_WebService	Пример Metro Web-сервиса, использующего STS-маркер одобрения
	STSEndorsing\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего STS-маркер одобрения
	STSEndorsing\STSProject	Пример STS-сервиса
	STSSupporting\ Metro_WebService	Пример Metro Web-сервиса, использующего STS-маркер поддержки
	STSSupporting\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего STS-маркер поддержки
	STSSupporting\ STSProject	Пример STS-сервиса
	SymmetricBinding\ Metro_WebService	Пример Metro Web-сервиса, использующего проверку подлинности имени пользователя с помощью симметричного ключа
	SymmetricBinding\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего проверку подлинности имени пользователя с помощью симметричного ключа
	UsernameTokenSSL\ Metro_WebService	Пример Metro Web-сервиса, использующего проверку подлинности сообщения по протоколу SSL
	UsernameTokenSSL\ Metro_WebClient	Пример клиента Web-сервиса платформы Metro, использующего проверку подлинности сообщения по протоколу SSL
\Глава5		
Aegis	SimpleWebService	Пример CXF Web-сервиса, использующего систему Aegis совместно с программным интерфейсом Simple Frontend
	SimpleWebServiceClient	Пример клиента Web-сервиса платформы CXF, использующего систему Aegis совместно с программным интерфейсом Simple Frontend

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
CamelTransport	CXFCamelTransport_1	Пример CXF Web-сервиса и CXF-клиента, использующих протокол Camel transport for CXF
	CXFCamelTransport_2	Пример CXF Web-сервиса, использующего маршрут `<route>`, и CXF-клиента, применяющих протокол Camel transport for CXF
	CXFCamelTransport_3	Пример CXF-клиента, использующего объявление `<jaxws:client>`, и CXF Web-сервиса, применяющих протокол Camel transport for CXF
	CXFCamelTransport_4	Пример CXF Web-сервиса, использующего для подключения протокола Camel transport for CXF элемент `<camel:destination>`, и CXF-клиента, применяющих протокол Camel transport for CXF
CXFCamelComponent		Пример CXF Web-сервиса и CXF-клиента, использующих CXF Camel-компонент
CXFJavaScript		Пример CXF JavaScript-клиента Web-сервиса
CXFLocalTransport		Пример CXF Web-сервиса и CXF-клиента, использующих локальный транспорт
CXFMTOM		Пример CXF Web-сервиса и CXF-клиента, использующих механизм MTOM оптимизации обмена сообщениями
HTTP	CXFHTTPClient	Пример CXF-клиента, использующего протокол xformat
	CXFHTTPService	Пример CXF Web-сервиса, использующего протокол xformat
HTTPS	CXFHTTPSClient	Пример CXF-клиента, использующего протокол HTTP/SSL
	CXFHTTPSService	Пример CXF Web-сервиса, использующего протокол HTTP/SSL
Interceptors	CXFClient	Пример CXF-клиента, использующего обработчики Interceptor
	CXFService	Пример CXF Web-сервиса, использующего обработчики Interceptor
JAX-RS	CXFJAXRSExample	Пример JAX-RS Web-сервиса и его клиента на платформе CXF
JMSService	CXFJMSCamel_1	Пример CXF Web-сервиса и CXF-клиента, взаимодействующих через Camel-компонент `JmsComponent` ActiveMQ-брокера по Camel-транспорту

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
	CXFJMSCamel_2	Пример CXF Web-сервиса и CXF-клиента, взаимодействующих через Camel-компонент `ActiveMQComponent` ActiveMQ-брокера по Camel-транспорту
	CXFJMSTransport_1	Пример CXF Web-сервиса и CXF-клиента, использующих ActiveMQ-брокера через JMS-транспорт
	CXFJMSTransport_2	Пример CXF Web-сервиса и CXF-клиента, взаимодействующих друг с другом через ActiveMQ-брокера
	CXFJMSTransport_3	Пример CXF Web-сервиса и CXF-клиента, использующих технологию SOAP over JMS
SimpleCXFClient		Пример динамического Simple Frontend клиента
SimpleWebService		Пример Simple Frontend Web-сервиса
SimpleWebServiceClient		Пример Simple Frontend клиента
Soap11	CXFClient	Пример CXF-клиента, использующего протокол SOAP 1.1/HTTP
	CXFSoap11Service	Пример CXF Web-сервиса, использующего протокол SOAP 1.1/HTTP
Soap12	CXFClient	Пример CXF-клиента, использующего протокол SOAP 1.2/HTTP
	CXFSoap11Service	Пример CXF Web-сервиса, использующего протокол SOAP 1.2/HTTP
WS	CXFAddressing_1	Пример CXF Web-сервиса и CXF-клиента, использующих элемент `<jaxws:features>` для подключения WS-Addressing
	CXFAddressing_2	Пример CXF Web-сервиса и CXF-клиента, использующих политику конечной точки для подключения WS-Addressing
	CXFRM_1	Пример CXF Web-сервиса и CXF-клиента, использующих элемент `<p:policies>` для подключения WS-ReliableMessaging
	CXFRM_2	Пример CXF Web-сервиса и CXF-клиента, использующих элементы `<wsa:addressing/>` и `<wsrm-mgr:reliableMessaging/>` для подключения WS-ReliableMessaging
	CXFSecureConversation	Пример CXF Web-сервиса и CXF-клиента, использующих технологии WS-SecurityPolicy и WS-SecureConversation

Таблица П1 *(продолжение)*

Папка	Вложенная папка	Описание
	CXFSecurity	Пример CXF Web-сервиса и CXF-клиента, использующих WSS4J-реализацию
	CXFSecurityPolicy	Пример CXF Web-сервиса и CXF-клиента, использующих технологию WS-SecurityPolicy
\Глава6		
AXIOM	AXIOMWebService	Пример Axis2 Web-сервиса, использующего систему AXIOM
	AXIOMWebServiceClient	Пример Axis2-клиента, использующего систему AXIOM
Axis2Authentication		Пример базовой аутентификации клиента Web-сервиса платформы Axis2
Axis2JMS		Пример использования JMS-транспорта платформой Axis2
Axis2RM		Пример использования технологии WS-ReliableMessaging платформой Axis2
Axis2TCP		Пример использования протокола TCP платформой Axis2
Deployment	Axis2_WebService	Пример развертывания Axis2 Web-сервиса различными способами
HTTPS	HTTPSAxis2	Пример Axis2 Web-сервиса и Axis2-клиента, взаимодействующих по HTTPS-протоколу
	keys	Содержит хранилища ключей, используемые в примере
MTOM	MTOMWebService	Пример Axis2 Web-сервиса, использующего технологию MTOM
	MTOMWebServiceClient	Пример Axis2-клиента, использующего технологию MTOM
OperationClient	Axis2_WebService	Пример Axis2 Web-сервиса, взаимодействующего с OperationClient-клиентом
	Axis2OperationClient	Пример Axis2-клиента, созданного на основе OperationClient API
Rampart	Axis2SecureConversation	Пример создания защищенной сессии между Axis2-клиентом и Axis2 Web-сервисом согласно спецификации WS-SecureConversation
	Axis2UsernameToken	Пример аутентификации клиента с помощью UsernameToken-маркера, передаваемого по HTTPS-протоколу, с использованием параметров `InflowSecurity` и `OutflowSecurity` для конфигурирования Rampart-модуля

Таблица П1 *(окончание)*

Папка	Вложенная папка	Описание
	Axis2UsernameTokenPolicy	Пример аутентификации клиента с помощью UsernameToken-маркера, передаваемого по HTTPS-протоколу, с использованием элемента `<wsp:Policy>` с дочерним элементом `<ramp:RampartConfig>` для конфигурирования Rampart-модуля
REST	Axis2RESTClient	Пример клиента RESTful Web-сервиса платформы Axis2
	Axis2RESTWebService	Пример RESTful Web-сервиса платформы Axis2
RPCClient	Axis2_WebService	Пример Axis2 Web-сервиса, взаимодействующего с RPCServiceClient-клиентом
	Axis2RPCClient	Пример Axis2-клиента, созданного на основе RPCServiceClient API
ServiceClient	Axis2_WebService	Пример Axis2 Web-сервиса, взаимодействующего с ServiceClient-клиентом
	Axis2ServiceClient	Пример Axis2-клиента, созданного на основе ServiceClient API
SOAP11	Axis2_WebService	Пример Axis2 Web-сервиса, использующего протокол SOAP 1.1
	Axis2_WebServiceClientSoap11	Пример Axis2-клиента, использующего протокол SOAP 1.1
SOAP12	Axis2_WebService	Пример Axis2 Web-сервиса, использующего протокол SOAP 1.2
	Axis2_WebServiceClientSoap12	Пример Axis2-клиента, использующего протокол SOAP 1.2

Список литературы

1. Bill Burke. RESTful Java with Jax-RS (Animal Guide). — O'Reilly Media, 2009.
2. Bruce Snyder, Dejan Bosanac, Rob Davies. ActiveMQ in Action. — Manning Publications Co., 2011.
3. Claus Ibsen, Jonathan Anstey. Camel in Action. — Manning Publications Co., 2011.
4. David Chappell, Tyler Jewell. Java Web Services. — O'Reilly Media, 2002.
5. Deepal Jayasinghe, Afkham Azeez. Apache Axis2 Web Services. — Packt Publishing, 2011.
6. Eben Hewitt. Java SOA Cookbook. — O'Reilly Media, 2009.
7. Gregor Hohpe, Bobby Woolf. Enterprise Integration Patterns. — Addison-Wesley Professional, 2003.
8. James Bean. SOA and Web Services Interface Design. — Morgan Kaufmann, 2009.
9. Kent Ka Iok Tong. Developing Web Services with Apache CXF and Axis2. — TipTec Development, 2010.
10. Kim Topley. Java Web Services in Nutshell. — O'Reilly Media, 2003.
11. Leonard Richardson, Sam Ruby. RESTful Web Services. — O'Reilly Media, 2007.
12. Mark D. Hansen. SOA Using Java Web Services. — Pearson Education, Inc., 2007.
13. Martin Kalin. Java Web Services: Up and Running. — O'Reilly Media, 2009.
14. Matjaz B. Juric, Benny Mathew, Poornachandra Sarang. Business Process Execution Language for Web Services. — Packt Publishing Ltd, 2006.
15. Naveen Balan, Rajeev Hathi. Apache CXF Web Service Development. — Packt Publishing Ltd, 2009.
16. Richard Monson-Haefel. J2EE Web Services. — Addison-Wesley Professional, 2003.
17. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson. Web Services Platform Architecture. — Prentice Hall, 2005.
18. Subbu Allamaraju. RESTful Web Services Cookbook. — Yahoo Press, 2010.

Предметный указатель

A

Aegis 400
Apache CXF 388
Axis2 464
◊ модули 473
◊ связывание данных 500
◊ транспортные протоколы 530

B

BindGen, инструмент 510

C

Choreography Description Language (CDL) 16
CodeGen, инструмент 515

D

DISCO 67
disco.exe, утилита 68
Document Type Definition (DTD) 20, 26

E

ebXML 64

F

Fast Infoset 384

J

JAXB 229
JAXM 211
JAXP 222
JAXR, библиотека 68
JAX-RS 294
JAX-WS 262
Jibx2Wsdl, инструмент 513
JSON 298

M

Message Oriented Model (MOM) 14
Metro 312
Multipurpose Internet Mail Extensions (MIME) 47

P

Policy Model 17

R

RELAX NG 27
Representational State Transfer (REST) 294
Resource Oriented Model (ROM) 16

S

SchemaGen, инструмент 230, 513
Secure Sockets Layer (SSL), протокол 351
Service Description Language (SDL) 15
Service Oriented Architecture (SOA) 17
Service Oriented Model (SOM) 15
SOAP 41

U

UDDI 62

W

Web Application Description Language (WADL) 299
Web Services Description Language (WSDL) 50
Web-сервис 16
◊ документ-ориентированный 60
◊ метод-ориентированный 60
◊ оркестровка 71
◊ ресурс-ориентированный 60
◊ хореография 71
WS-BPEL 71
WS-CDL 71, 86
wscompile, инструмент 249
wsdeploy, инструмент 249, 257
wsgen, инструмент 267
wsimport, инструмент 268

X

xjc, инструмент 230
XML 19
◊ документ 19
▫ без правил 26
▫ действительный 21
▫ логическая структура 20
▫ правильный 21
▫ с учетом правил 26
▫ физическая структура 21
◊ инструкция 20
◊ комментарий 20
◊ процессор 26
◊ ссылка 20
◊ сущность 21
◊ схема 26
◊ элемент:
▫ атрибут 20, 23
▫ дочерний 20
▫ корневой 20
XML Infoset 22
XML Namespaces 21
XML Schema 27
XMLBeans 403
XML-процессор 19

А

Агент 13, 16

И

Информационная единица 22
Информационное пространство, синтетическое 26

М

Маркер защиты 118
Маршрут 423

П

Политика 98
Поставщик сервиса 13
Потребитель сервиса 13
Пространство имен 21
◊ по умолчанию 21

Р

Разметка 20

С

Сериализация 42

Ф

Федерация доменов 160

Х

Хореография Web-сервисов 16

ТИМУР МАШНИН

bhv®

Web-сервисы Java

ОСНОВЫ ТЕХНОЛОГИИ WEB-СЕРВИСОВ В СПЕЦИФИКАЦИЯХ ПЕРВОГО И ВТОРОГО УРОВНЯ

СТАНДАРТЫ ТЕХНОЛОГИИ WEB-СЕРВИСОВ ПЛАТФОРМЫ JAVA

JAVA-СТЕКИ WEB-СЕРВИСОВ: Metro, CXF И Axis2

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Дополнительные материалы

+

Материалы на www.bhv.ru

JAXB API

Класс `JAXBContext` пакета `javax.xml.bind` имеет следующие методы.

- `public static JAXBContext newInstance(String contextPath)`

  `public static JAXBContext newInstance(String contextPath, ClassLoader classLoader)`

  `public static JAXBContext newInstance(String contextPath, ClassLoader classLoader, Map<String,?> properties)`

  `public static JAXBContext newInstance(Class... classesToBeBound)`

  `public static JAXBContext newInstance(Class[] classesToBeBound, Map<String,?> properties)`

  Создают объекты `JAXBContext`.

- `public abstract Unmarshaller createUnmarshaller()`

  Создает объект `javax.xml.bind.Unmarshaller`, обеспечивающий процесс десериализации XML-данных из различного рода источников в дерево Java-объектов. Интерфейс `Unmarshaller` имеет следующие методы.

  - `Object unmarshal(java.io.File f)`

    `Object unmarshal(java.io.InputStream is)`

    `Object unmarshal(java.io.Reader reader)`

    `Object unmarshal(java.net.URL url)`

    `Object unmarshal(org.xml.sax.InputSource source)`

    `Object unmarshal(org.w3c.dom.Node node)`

    `Object unmarshal(javax.xml.transform.Source source)`

    `Object unmarshal(javax.xml.stream.XMLStreamReader reader)`

    `Object unmarshal(javax.xml.stream.XMLEventReader reader)`

    Десериализуют XML-данные из различных источников.

  - `<T> JAXBElement<T> unmarshal(org.w3c.dom.Node node, Class<T> declaredType)`

    `<T> JAXBElement<T> unmarshal(javax.xml.transform.Source source, Class<T> declaredType)`

    `<T> JAXBElement<T> unmarshal(javax.xml.stream.XMLStreamReader reader, Class<T> declaredType)`

    `<T> JAXBElement<T> unmarshal(XMLEventReader reader, Class<T> declaredType)`

    Десериализуют XML-данные из различных источников с параметром `declaredType`, который позволяет осуществлять десериализацию, даже если нет связи "XML-в-Java" в объекте `JAXBContext` для корневого XML-элемента.

Параметр `declaredType` также переопределяет связь "XML-в-Java" в объекте `JAXBContext` для корневого XML-элемента.

- `UnmarshallerHandler getUnmarshallerHandler()`

  Возвращает объект `javax.xml.bind.UnmarshallerHandler`, реализующий десериализацию как объект `org.xml.sax.ContentHandler`. Интерфейс `UnmarshallerHandler` расширяет интерфейс `ContentHandler` и имеет дополнительно метод `Object getResult()`, возвращающий результат демаршализации.

- `void setEventHandler(ValidationEventHandler handler)`

  `ValidationEventHandler getEventHandler()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.bind.ValidationEventHandler`, получающий уведомления об ошибках корректности XML-данных в процессе демаршализации. Интерфейс `ValidationEventHandler` имеет метод `boolean handleEvent(ValidationEvent event)`, вызываемый реализацией при возникновении ошибки. Интерфейс `javax.xml.bind.ValidationEvent` представляет ошибки, фатальные ошибки и предупреждения корректности XML-данных и имеет методы:

  - `int getSeverity()` (возвращает 1 — ошибка, 2 — фатальная ошибка, 3 — предупреждение);
  - `String getMessage()`;
  - `Throwable getLinkedException()`;
  - `ValidationEventLocator getLocator()`.

  Интерфейс `javax.xml.bind.ValidationEventLocator` представляет расположение ошибки и имеет методы `URL getURL()`, `int getOffset()`, `int getLineNumber()`, `int getColumnNumber()`, `Object getObject()`, `Node getNode()`.

- `void setProperty(String name, Object value)`

  `Object getProperty(String name)`

  Устанавливает и возвращает свойства реализации.

- `void setSchema(Schema schema)`

  `Schema getSchema()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.validation.Schema`, используемый для проверки корректности XML-данных. Если объект `Schema` является нулевым, тогда функция проверки отключена.

- `void setAdapter(XmlAdapter adapter)`

  или

  `<A extends XmlAdapter> void setAdapter(Class<A> type, A adapter)`

  `<A extends XmlAdapter> A getAdapter(Class<A> type)`

  Первый и второй методы устанавливают, третий метод возвращает объект `javax.xml.bind.annotation.adapters.XmlAdapter<ValueType,BoundType>`, связывающий Java-тип с XML-представлением. Класс `XmlAdapter` имеет методы

`public abstract BoundType unmarshal(ValueType v)`

`public abstract ValueType marshal(BoundType v)`

конвертирующие XML-тип данных в Java-тип данных и наоборот. Пакет `javax.xml.bind.annotation.adapters` содержит классы `CollapsedStringAdapter`, `HexBinaryAdapter`, `NormalizedStringAdapter`, расширяющие класс `XmlAdapter` и адаптирующие XML-типы `xsd:token`, `xsd:hexBinary`, `xsd:normalizedString` к Java-типу `String`.

- `void setAttachmentUnmarshaller(AttachmentUnmarshaller au)`

  `AttachmentUnmarshaller getAttachmentUnmarshaller()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.bind.attachment.AttachmentUnmarshaller`, обеспечивающий десериализацию XML-документа, содержащего оптимизированные бинарные данные. Класс `AttachmentUnmarshaller` имеет методы

  `public abstract DataHandler getAttachmentAsDataHandler(String cid)`

  `public abstract byte[] getAttachmentAsByteArray(String cid)`

  возвращающие вложения по ссылке, а также метод `public boolean isXOPPackage()`, возвращающий `true`, если обрабатывается XOP-документ.

- `void setListener(Unmarshaller.Listener listener)`

  `Unmarshaller.Listener getListener()`

  Регистрируется и возвращается объект `javax.xml.bind.Unmarshaller.Listener`, отвечающий за получение уведомлений в процессе десериализации. Класс `Unmarshaller.Listener` имеет методы

  `public void beforeUnmarshal(Object target, Object parent)`

  `public void afterUnmarshal(Object target, Object parent)`

  оперирующие экземплярами JAXB-классов, связанных с XML-представлением.

❒ `public abstract Marshaller createMarshaller()`

Создает объект `javax.xml.bind.Marshaller`, обеспечивающий сериализацию дерева Java-объектов в XML-данные. Интерфейс `Marshaller` имеет следующие методы.

- `void marshal(Object jaxbElement, javax.xml.transform.Result result)`

  `void marshal(Object jaxbElement, java.io.OutputStream os)`

  `void marshal(Object jaxbElement, java.io.Writer writer)`

  `void marshal(Object jaxbElement, org.xml.sax.ContentHandler handler)`

  `void marshal(Object jaxbElement, org.w3c.dom.Node node)`

  ```
  void marshal(Object jaxbElement,
    javax.xml.stream.XMLStreamWriter writer)
  void marshal(Object jaxbElement,
    javax.xml.stream.XMLEventWriter writer)
  ```

  Сериализуют дерево Java-объектов в XML-данные.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `Node getNode(Object contentTree)`

  Возвращает DOM-дерево дерева Java-объектов.

- `void setProperty(String name, Object value)`

  `Object getProperty(String name)`

  Первый метод устанавливает, второй метод возвращает свойства реализации, такие как `jaxb.encoding`, `jaxb.formatted.output`, `jaxb.schemaLocation`, `jaxb.noNamespaceSchemaLocation`, `jaxb.fragment`.

- `void setEventHandler(ValidationEventHandler handler)`

  `ValidationEventHandler getEventHandler()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.bind.ValidationEventHandler`, получающий уведомления об ошибках корректности XML-данных в процессе маршализации.

- `void setAdapter(XmlAdapter adapter)`

  или

  `<A extends XmlAdapter> void setAdapter(Class<A> type, A adapter)`

  `<A extends XmlAdapter> A getAdapter(Class<A> type)`

  Первый и второй методы устанавливают, третий метод возвращает объект `javax.xml.bind.annotation.adapters.XmlAdapter<ValueType,BoundType>`, адаптирующий Java-тип к XML-представлению.

- `void setAttachmentMarshaller(AttachmentMarshaller am)`

  `AttachmentMarshaller getAttachmentMarshaller()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.bind.attachment.AttachmentMarshaller`, обеспечивающий сериализацию XML-документа, содержащего оптимизированные бинарные данные. Класс `AttachmentMarshaller` имеет методы:

  ```
  public abstract String addMtomAttachment(DataHandler data,
    String elementNamespace, String elementLocalName)
  public abstract String addMtomAttachment(byte[] data, int offset,
    int length, String mimeType, String elementNamespace,
    String elementLocalName)
  public boolean isXOPPackage()
  public abstract String addSwaRefAttachment(DataHandler data)
  ```

- `void setSchema(Schema schema)`

  `Schema getSchema()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.validation.Schema`, используемый для проверки корректности XML-данных. Если объект `Schema` является нулевым, тогда функция проверки отключена.

- `void setListener(Marshaller.Listener listener)`

  `Marshaller.Listener getListener()`

  Регистрируется и возвращается объект `javax.xml.bind.Marshaller.Listener`, отвечающий за получение уведомлений в процессе сериализации.

Класс `Marshaller.Listener` имеет методы:

```
public void beforeMarshal(Object source)
public void afterMarshal(Object source)
```

- `public <T> Binder<T> createBinder(Class<T> domType)`
  `public Binder<Node> createBinder()`

  Создает объект `javax.xml.bind.Binder<XmlNode>`, обеспечивающий синхронизацию между деревом узлов XML infoset и деревом JAXB-объектов. Класс `Binder` имеет следующие методы:

  - `public abstract Object unmarshal(XmlNode xmlNode)`
    `public abstract <T> JAXBElement<T> unmarshal(XmlNode xmlNode, Class<T> declaredType)`

    Десериализуют дерево узлов XML infoset в дерево JAXB-объектов так же, как `Unmarshaller.unmarshal`, за исключением того, что класс `Binder` позволяет в дальнейшем синхронизовать изменения в JAXB-дереве с изменениями в XML-дереве путем их обновления.

  - `public abstract void marshal(Object jaxbObject, XmlNode xmlNode)`

    Сериализует дерево JAXB-объектов в новый XML-документ так же, как `Marshaller.marshal`, за исключением того, что класс `Binder` позволяет в дальнейшем синхронизовать изменения в JAXB-дереве с изменениями в XML-дереве путем их обновления.

  - `public abstract XmlNode getXMLNode(Object jaxbObject)`
    `public abstract Object getJAXBNode(XmlNode xmlNode)`

    Возвращают XML-элемент, связанный с указанным JAXB-объектом, и наоборот.

  - `public abstract XmlNode updateXML(Object jaxbObject)`
    `public abstract XmlNode updateXML(Object jaxbObject, XmlNode xmlNode)`
    `public abstract Object updateJAXB(XmlNode xmlNode)`

    Обновляют дерево узлов XML infoset и дерево JAXB-объектов, синхронизируя изменения.

  - `void setSchema(Schema schema)`
    `Schema getSchema()`

    Первый метод устанавливает, второй метод возвращает объект `javax.xml.validation.Schema`, используемый для проверки корректности XML-данных. Если объект `Schema` является нулевым, тогда функция проверки отключена.

  - `void setEventHandler(ValidationEventHandler handler)`
    `ValidationEventHandler getEventHandler()`

    Первый метод устанавливает, второй метод возвращает объект `javax.xml.bind.ValidationEventHandler`, получающий уведомления об ошибках корректности XML-данных.

- `void setProperty(String name, Object value)`

  `Object getProperty(String name)`

  Первый метод устанавливает, второй метод возвращает свойства реализации.

❒ `public JAXBIntrospector createJAXBIntrospector()`

Создает объект `javax.xml.bind.JAXBIntrospector`, обеспечивающий доступ к XML-данным, связанным с определенным JAXB-объектом с помощью методов:

```
public abstract boolean isElement(Object JAXBElement)
public abstract QName getElementName(Object jaxbElement)
public static Object getValue(Object jaxbElement)
```

❒ `public void generateSchema(SchemaOutputResolver outputResolver)`

Генерирует XML-схему. Класс `javax.xml.bind.SchemaOutputResolver` позволяет определить размещение генерируемой XML-схемы с помощью метода

```
public abstract javax.xml.transform.Result createOutput
  (String namespaceUri, String suggestedFileName).
```

Перед осуществлением маршализации или демаршализации, при вызове приложением метода `JAXBContext.newInstance()`, JAXB-реализация вызывает статический метод

```
public static void setDatatypeConverter
  (DatatypeConverterInterface converter)
```

класса `javax.xml.bind.DatatypeConverter`, устанавливая механизм конвертации по умолчанию Java-типов в XML-типы данных и наоборот. Этот механизм может быть переопределен в объявлениях связей элементом `<jaxb:javaType>`. Параметром данного метода служит объект `DatatypeConverterInterface`, представляющий экземпляр класса JAXB-реализации интерфейса `javax.xml.bind.DatatypeConverterInterface`.

Класс `javax.xml.bind.DatatypeConverter`, упоминавшийся в описании элемента `<jaxb:javaType>` объявлений связей, имеет следующие методы:

❒ `public static String parseString(String lexicalXSDString)`

`public static String printString(String val)`

Конвертирует данные типа `xsd:string` в значения `String` и наоборот.

❒ `public static BigInteger parseInteger(String lexicalXSDInteger)`

`public static String printInteger(BigInteger val)`

Конвертирует данные типа `xsd:integer` в значения `BigInteger` и наоборот.

❒ `public static int parseInt(String lexicalXSDInt)`

`public static String printInt(int val)`

Конвертирует данные типа `xsd:int` в значения `int` и наоборот.

❒ `public static long parseLong(String lexicalXSDLong)`

`public static String printLong(long val)`

Конвертирует данные типа `xsd:long` в значения `long` и наоборот.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public static short parseShort(String lexicalXSDShort)`
  `public static String printShort(short val)`

  Конвертирует данные типа `xsd:short` в значения `short` и наоборот.
- `public static BigDecimal parseDecimal(String lexicalXSDDecimal)`
  `public static String printDecimal(BigDecimal val)`

  Конвертирует данные типа `xsd:decimal` в значения `BigDecimal` и наоборот.
- `public static float parseFloat(String lexicalXSDFloat)`
  `public static String printFloat(float val)`

  Конвертирует данные типа `xsd:float` в значения `float` и наоборот.
- `public static double parseDouble(String lexicalXSDDouble)`
  `public static String printDouble(double val)`

  Конвертирует данные типа `xsd:double` в значения `double` и наоборот.
- `public static boolean parseBoolean(String lexicalXSDBoolean)`
  `public static String printBoolean(boolean val)`

  Конвертирует данные типа `xsd:boolean` в значения `boolean` и наоборот.
- `public static byte parseByte(String lexicalXSDByte)`
  `public static String printByte(byte val)`

  Конвертирует данные типа `xsd:byte` в значения `byte` и наоборот.
- `public static QName parseQName(String lexicalXSDQName, NamespaceContext nsc)`
  `public static String printQName(QName val, NamespaceContext nsc)`

  Конвертирует данные типа `xsd:Qname` в значения `Qname` и наоборот.
- `public static Calendar parseDateTime(String lexicalXSDDateTime)`
  `public static String printDateTime(Calendar val)`

  Конвертирует данные типа `xsd:datetime` в значения `Calendar` и наоборот.
- `public static byte[] parseBase64Binary(String lexicalXSDBase64Binary)`
  `public static String printBase64Binary(byte[] val)`

  Конвертирует данные типа `xsd:base64Binary` в значения массива байтов и наоборот.
- `public static byte[] parseHexBinary(String lexicalXSDHexBinary)`
  `public static String printHexBinary(byte[] val)`

  Конвертирует данные типа `xsd:hexBinary` в значения массива байтов и наоборот.
- `public static long parseUnsignedInt(String lexicalXSDUnsignedInt)`
  `public static String printUnsignedInt(long val)`

  Конвертирует данные типа `xsd:unsignedInt` в значения `long` и наоборот.
- `public static int parseUnsignedShort(String lexicalXSDUnsignedShort)`
  `public static String printUnsignedShort(int val)`

  Конвертирует данные типа `xsd:unsignedShort` в значения `int` и наоборот.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public static Calendar parseTime(String lexicalXSDTime)`
  `public static String printTime(Calendar val)`

  Конвертирует данные типа `xsd:time` в значения `Calendar` и наоборот.
- `public static Calendar parseDate(String lexicalXSDDate)`
  `public static String printDate(Calendar val)`

  Конвертирует данные типа `xsd:Date` в значения `Calendar` и наоборот.
- `public static String parseAnySimpleType(String lexicalXSDAnySimpleType)`
  `public static String printAnySimpleType(String val)`

  Конвертирует данные типа `simple type` в значения `String` и наоборот.

В некоторых случаях XML-представление связывается не с JAXB-классами, а с классом `javax.xml.bind.JAXBElement<T>`. Это происходит тогда, когда XML-компоненты не могут быть просто представлены объектной моделью Java, например, в случае использования атрибута `substitutionGroup`. Класс `JAXBElement` представляет XML-элемент и имеет следующие поля, конструкторы и методы.

- `protected final QName name` — имя XML-элемента.
- `protected final Class<T> declaredType` — Java-тип для связи с XML-типом.
- `protected final Class scope` — область действия XML-элемента. Для глобального XML-элемента — `JAXBElement.GlobalScope`, для локального элемента — Java-класс, представляющий объявление сложного типа `complex type`, содержащее объявление XML-элемента.
- `protected T value` — значение XML-элемента.
- `protected boolean nil` — `true`, если элемент имеет атрибут `xsi:nil="true"`.
- `public JAXBElement(QName name, Class<T> declaredType, Class scope, T value)`
  `public JAXBElement(QName name, Class<T> declaredType, T value)`

  Создают объект `JAXBElement`, представляющий XML-элемент.
- `public Class<T> getDeclaredType()`

  Возвращает Java-тип, связанный с XML-типом данных элемента.
- `public QName getName()`

  Возвращает имя элемента.
- `public void setValue(T t)`
  `public T getValue()`

  Первый метод устанавливает, второй метод возвращает содержимое элемента и значения его атрибутов.
- `public Class getScope()`

  Возвращает область действия элемента.
- `public boolean isNil()`
  `public void setNil(boolean value)`

  Первый метод возвращает, второй метод устанавливает `true`, если элемент имеет атрибут `xsi:nil="true"`.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public boolean isGlobalScope()`

  Возвращает `True`, если элемент является глобальным.

- `public boolean isTypeSubstituted()`

  Возвращает `true`, если элемент имеет значение другого типа, нежели было указано при его объявлении.

Пакеты `javax.xml.bind.annotation` и `javax.xml.bind.annotation.adapters` определяют аннотации программных элементов, обеспечивающие их связь с XML-схемой. JAXB-аннотации классифицируются относительно областей их применения: аннотации пакета, классов, перечислений, JavaBean-свойств, JavaBean-полей, параметров.

Аннотация `@XmlAccessorOrder` применяется к пакету и к классу верхнего уровня и используется для определения порядка, в котором располагаются XML-элементы, связанные с JavaBean-свойствами (или полями). Аннотация `@XmlAccessorType(XmlAccessOrder.UNDEFINED)` является аннотацией по умолчанию, и порядок обработки JavaBean-свойств (или полей) зависит от реализации. Аннотация `@XmlAccessorOrder(AccessorOrder.ALPHABETICAL)` определяет порядок расположения по алфавиту.

Аннотация `@XmlAccessorType` применяется к пакету и к классу верхнего уровня и используется для установки сериализации JavaBean-свойств (или полей). По умолчанию JAXB-реализация сериализует все публичные поля и свойства (аннотация `@XmlAccessorType(XmlAccessType.PUBLIC_MEMBER)`). С помощью аннотаций `@XmlAccessorType(XmlAccessType.FIELD)`, `@XmlAccessorType(XmlAccessType.PROPERTY)`, `@XmlAccessorType(XmlAccessType.NONE)` можно контролировать сериализацию только полей, только свойств или отменять сериализацию.

Аннотация `@XmlSchema` применяется к пакету и используется для связывания имени пакета с XML-пространством имен. Аннотация `@XmlSchema(namespace = "...")` используется для переопределения значения атрибута `targetNamespace` в элементе `<xsd:schema>`, аннотация `@XmlSchema(xmlns = {@XmlNs(prefix = "...", namespaceURI="...")})` служит для переопределения объявления префикса и URI пространства имен в элементе `<xsd:schema>` XML-схемы. Аннотация `@XmlSchema(elementFormDefault=[XmlNsForm.UNQUALIFIED|XmlNsForm.QUALIFIED|XmlNsForm.UNSET])` используется для определения значения атрибута `elementFormDefault` в элементе `<xsd:schema>`.

Аннотация `@XmlSchemaType(name="..." type=... namespace="...")` применяется к пакету и к JavaBean-свойствам (или полям) и используется для связывания Java-типа с XML-типом данных. Элемент `name` указывает XML-тип, а дополнительные элементы `type` и `namespace` — Java-класс, представляющий Java-тип, и пространство имен соответствующего элемента XML-схемы (по умолчанию `"http://www.w3.org/2001/XMLSchema"`).

Аннотация `@XmlSchemaTypes` является контейнером для аннотаций `@XmlSchemaType`: `@XmlSchemaTypes({@XmlSchemaType(...), @XmlSchemaType(...)})`.

Аннотация `@XmlJavaTypeAdapter(value=... type=...)` применяется к пакету, классу, перечислениям, к JavaBean-свойствам (или полям), параметрам и используется для указания Java-класса, расширяющего класс `javax.xml.bind.annotation.adapters.XmlAdapter<ValueType,BoundType>` и обеспечивающего адаптацию Java-типа к XML-типу при маршализации. Обязательный элемент `value` указывает Java-класс, расширяющий `XmlAdapter`, а дополнительный элемент `type` указывает адаптируемый Java-тип.

Аннотация `@XmlJavaTypeAdapters` служит контейнером для аннотаций `@XmlJavaTypeAdapter`: `@XmlJavaTypeAdapters({@XmlJavaTypeAdapter(...), @XmlJavaTypeAdapter(...)})`.

Аннотация `@XmlInlineBinaryData` применяется к классу и к JavaBean-свойствам (или полям) и запрещает вызов методов `AttachmentMarshaller.addMtomAttachment()` и `AttachmentMarshaller.addMtomAttachment()` для JAXB-свойств.

Аннотация `@XmlRootElement(name="..." namespace="...")` применяется к классам и перечислениям и связывает их с XML-элементами. Дополнительные элементы `name` и `namespace` указывают локальное имя и пространство имен XML-элемента.

Аннотация `@XmlType(name="..." propOrder={...} namespace="..." factoryClass=... factoryMethod=...)` применяется к классам и перечислениям и связывает их с объявлениями типа данных в XML-схеме. Дополнительные элементы: `name` — имя объявления `<xsd:complexType>` или `<xsd:simpleType>`; `propOrder` — порядок расположения XML-элементов; `namespace` — URI целевого пространства имен; `factoryClass` — класс-фабрика, отвечающий за создание JAXB-объектов; `factoryMethod` — метод-фабрика, отвечающий за создание JAXB-объектов.

Аннотация `@XmlEnum(value=...)` применяется к Java-перечислениям и используется для связывания их с XML-перечислениями `<xsd:enumeration>`. Дополнительный элемент `value` указывает Java-тип, связанный с простым XML-типом перечисления (по умолчанию `java.lang.String.class`). Аннотация `@XmlEnumValue(value=...)` применяется к константам перечисления и уточняет значения атрибутов `value` элементов `<xsd:enumeration>`.

Аннотация `@XmlElement(name="..." nillable=... required=... namespace="..." defaultValue=... type=...)` применяется к JavaBean-свойствам (или полям) и связывает их с XML-элементами `<xsd:element>`. Дополнительные элементы: `name` — имя XML-элемента; `nillable` — значение атрибута `xsi:nil` элемента XML-схемы; `required` — если `false` (по умолчанию), тогда `minOccurs="0"` и `maxOccurs="1"` или `maxOccurs="unbounded"`; `namespace` — URI целевого пространства имен элемента XML-схемы; `defaultValue` — значение атрибута `default` элемента XML-схемы; `type` — Java-класс, представляющий тип данных элемента.

Аннотация `@XmlElements` служит контейнером для аннотаций `@XmlElement`: `@XmlElements({ @XmlElement(...),@XmlElement(...) })`.

Аннотация `@XmlElementRef(type=... namespace="..." name="...")`, также как и аннотация `@XmlElement`, применяется к JavaBean-свойствам (или полям) и связывает их с XML-элементами, однако аннотация `@XmlElement` связывает статически имя

XML-элемента с именем Java-свойства, а аннотация `@XmlElementRef` позволяет получать имя XML-элемента из экземпляра типа Java-свойства во время выполнения кода, т. е. динамически. Аннотация `@XmlElementRef` связывает Java-свойство с атрибутом `ref` элемента `<xsd:element>`. Дополнительные элементы: `type` — Java-класс, представляющий тип данных элемента; `namespace` и `name` — пространство имен и имя XML-элемента для Java-свойства.

Аннотация `@XmlElementRefs` служит контейнером для аннотаций `@XmlElementRef` и связывает Java-свойство с элементом `<xsd:choice>`.

Аннотация `@XmlElementWrapper(name="..." nillable=... namespace="...")` применяется к JavaBean-свойствам (или полям), представляющим коллекции, и отвечает за генерацию XML-элемента обертки для коллекции элементов. Дополнительные элементы: `name` — имя элемента-обертки, `namespace` — пространство имен элемента-обертки, `nillable` — значение атрибута `xsi:nil`.

Аннотация `@XmlAnyElement(lax=... value=...)` применяется к JavaBean-свойствам (или полям) и связывает их с элементами `<xsd:any>`. Дополнительные элементы: `lax` — если `true`, тогда значение атрибута `processContents="lax"`; `value` указывает Java-класс, реализующий интерфейс `javax.xml.bind.annotation.DomHandler <ElementT,ResultT extends Result>` и отвечающий за конвертацию XML-данных в DOM-данные и наоборот (по умолчанию `javax.xml.bind.annotation.W3CDomHandler.class`).

Аннотация `@XmlAttribute(name="..." required=... namespace="...")` применяется к JavaBean-свойствам (или полям) и связывает их с элементами `<xsd:attribute>`. Дополнительные элементы: `name` — имя атрибута; `required` — если `true`, тогда значение атрибута `use="required"`; `namespace` — пространство имен атрибута.

Аннотация `@XmlAnyAttribute` применяется к JavaBean-свойствам (или полям) типа `java.util.Map` и связывает их с элементом `<xsd:anyAttribute/>`.

Аннотация `@XmlTransient` применяется к JavaBean-свойствам (или полям) и предотвращает их связывание с XML-представлением.

Аннотация `@XmlValue` применяется к JavaBean-свойствам (или полям) и связывает их с элементами `<xsd:simpleType>` или элементом `<xsd:simpleContent>` элемента `<xsd:complexType>`.

Аннотация `@XmlID` применяется к JavaBean-свойствам (или полям) типа `java.lang.String` и связывает их с элементами `<xsd:attribute name="..." type="xsd:ID"/>`.

Аннотация `@XmlIDREF` применяется к JavaBean-свойствам (или полям) и связывает их с элементами `<xs:element name="..." type="xs:IDREF"/>`.

Аннотация `@XmlList` применяется к JavaBean-свойствам (или полям) и параметрам типа коллекции и связывает их с XML-списками простого типа.

Аннотация `@XmlMixed` применяется к JavaBean-свойствам (или полям) и связывает их с содержимым элемента `<xsd:complexType name="..." mixed="true">`.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Аннотация `@XmlMimeType(value=["image/*", "text/*"...])` применяется к JavaBean-свойствам (или полям) и параметрам типа `java.awt.Image` или `javax.xml.transform.Source` и связывает их с элементами

```
<xsd:element name="..." xmime:expectedContentTypes="value"
             type="xs:base64Binary"
             xmlns:xmime="http://www.w3.org/2005/05/xmlmime"/>
```

Аннотация `@XmlAttachmentRef` применяется к JavaBean-свойствам (или полям) и параметрам и связывает их со ссылками на MIME-вложения — `<xsd:element name="..." type="ref:swaRef"/>`.

Аннотация

```
@XmlElementDecl(name="..." scope=... namespace="..."
substitutionHeadNamespace="..." substitutionHeadName="..." defaultValue="...")
```

маркирует метод-фабрику, связывая его с элементом `<xsd:element>`, при этом класс-фабрика маркируется аннотацией `@XmlRegistry`. Обязательный элемент `name` указывает имя XML-элемента, а дополнительные элементы: `scope` — область действия элемента (по умолчанию `javax.xml.bind.annotation.XmlElementDecl.GLOBAL.class`), `namespace` — пространство имен элемента, `substitutionHeadNamespace` — пространство имен элемента, указанного в атрибуте `substitutionGroup`, `substitutionHeadName` — имя элемента, указанного в атрибуте `substitutionGroup`, `defaultValue` — значение по умолчанию элемента.

Пакет `javax.xml.bind.util` JAXB API определяет три класса: `JAXBResult`, `JAXBSource` и `ValidationEventCollector`.

Класс `JAXBResult` расширяет класс `javax.xml.transform.sax.SAXResult` и позволяет осуществлять демаршализацию XML-документа, полученного в результате XSLT-преобразования. Этот класс имеет следующие конструкторы и методы:

- `public JAXBResult(JAXBContext context)`
  `public JAXBResult(Unmarshaller _unmarshaller)`

  Создают объекты `JAXBResult`.
- `public Object getResult()`

  Производит демаршализацию.

Класс `JAXBSource` расширяет класс `javax.xml.transform.sax.SAXSource` и позволяет осуществлять маршализацию XML-документа для его XSLT-преобразования, используя конструкторы:

```
public JAXBSource(JAXBContext context, Object contentObject)
public JAXBSource(Marshaller marshaller, Object contentObject)
```

Класс `ValidationEventCollector` реализует интерфейс `javax.xml.bind.ValidationEventHandler` и собирает события в коллекции. Объекты `ValidationEventCollector` используются в качестве аргументов метода `setEventHandler()`. Класс `ValidationEventCollector` имеет следующие конструкторы и методы.

- `public ValidationEventCollector()`

  Создает объекты `ValidationEventCollector`.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public ValidationEvent[] getEvents()`

  Возвращает все ошибки и предупреждения после проверки XML-данных.
- `public void reset()`

  Возвращает объект `ValidationEventCollector` в исходное состояние.
- `public boolean hasEvents()`

  Возвращает `true`, если объект `ValidationEventCollector` не пустой.
- `public boolean handleEvent(ValidationEvent event)`

  Обрабатывает уведомление об ошибке.

JAXM API и SAAJ API

Пакет *javax.xml.messaging*

- **Интерфейс `OnewayListener`.** Интерфейс для Web-сервиса — получателя асинхронных клиентских сообщений, ответы на которые посылаются отдельными операциями.

  Интерфейс имеет метод

  `public void onMessage(javax.xml.soap.SOAPMessage message)`

  вызываемый контейнером Java EE сервера при получении сообщения от поставщика сообщений. Далее сообщение может обрабатываться, используя объект `SOAPMessage`.

  Клиентское приложение, работающее в контейнере Java EE сервера, также может реализовывать данный интерфейс для получения сообщений от поставщика сообщений.

- **Интерфейс `ReqRespListener`.** Интерфейс для Web-сервиса — получателя синхронных клиентских сообщений. Отправка запроса для такого типа сообщений и получение ответа выполняются в одной операции, поэтому Web-сервис, реализующий данный интерфейс, при получении клиентского сообщения сразу должен на него ответить.

  Интерфейс имеет метод

  `public javax.xml.soap.SOAPMessageonMessage(javax.xml.soap.SOAPMessage message)`

  вызываемый контейнером Java EE сервера при получении сообщения и возвращающий ответное сообщение.

  Клиентское приложение, работающее в контейнере Java EE сервера, также может реализовывать данный интерфейс для получения сообщений от поставщика сообщений.

- **Класс `ProviderConnectionFactory`.** Класс-фабрика, используемый клиентским приложением для создания объекта `ProviderConnection`, обеспечивающего соединение с поставщиком сообщений Messaging Provider.

  Класс имеет два метода.

  - `public static ProviderConnectionFactory newInstance()`

    Создает объект фабрики, связанный с поставщиком сообщений, определенным по умолчанию. Если же требуется установить сообщение с именованным поставщиком сообщений, тогда необходимо использовать пакет `javax.naming` следующим образом:

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3


```
Context ctx = new InitialContext();
ProviderConnectionFactory pcf =
  (ProviderConnectionFactory)ctx.lookup("[JNDI-имя]");
```

- `public abstract ProviderConnection createConnection()`

  Создает объект `ProviderConnection`, представляющий соединение с поставщиком сообщений.

❒ **Интерфейс `ProviderConnection`.** Интерфейс для клиентского приложения, устанавливающего соединение с поставщиком сообщений Messaging Provider.

Интерфейс имеет следующие методы:

- `public ProviderMetaData getMetaData()`

  Возвращает объект `ProviderMetaData`, содержащий информацию о поставщике сообщений.

- `public void close()`

  Закрывает соединение с поставщиком сообщений и освобождает все связанные ресурсы.

- `public javax.xml.soap.MessageFactory`
  `createMessageFactory(java.lang.String profile)`

  Создает объект `MessageFactory`, используемый для создания сообщений и соответствующий конкретному JAXM-профилю, определенному в параметре метода. JAXM-профиль — это расширение, позволяющее, например, использовать ebXML-протокол поверх SOAP-протокола.

- `public void send(javax.xml.soap.SOAPMessage message)`

  Отправляет сообщение через поставщика сообщений. Для отправки синхронного сообщения от точки к точке необходимо применять метод `call()` класса `javax.xml.soap.SOAPConnection`. В этом случае сообщение отправляется без использования поставщика сообщений.

❒ **Интерфейс `ProviderMetaData`.** Обеспечивает информацию о поставщике сообщений с помощью следующих методов:

- `public java.lang.String getName()`

  Возвращает имя поставщика сообщений.

- `public int getMajorVersion()`

  Возвращает максимальную версию, которой соответствует поставщик сообщений.

- `public int getMinorVersion()`

  Возвращает минимальную версию, которой соответствует поставщик сообщений.

- `public java.lang.String[] getSupportedProfiles()`

  Возвращает перечень поддерживаемых поставщиком сообщений JAXM-профилей.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- **Класс `Endpoint`.** Поставщик сообщений создает объекты `Endpoint`, представляющие по умолчанию URI-адреса конечных точек для доставки сообщения, исходя из заголовка SOAP-сообщения. Если же отправитель посылает синхронное сообщение без использования поставщика сообщений, он должен сам создать объект `Endpoint`, в частности объект его подкласса `URLEndpoint`, как параметр метода `call()` класса `javax.xml.soap.SOAPConnection`.

  Класс имеет конструктор `public Endpoint(java.lang.String uri)` и метод `public java.lang.String toString()`, возвращающий URI-адрес.

- **Класс `URLEndpoint`.** Класс представляет URL-адрес конечной точки для доставки сообщения и имеет конструктор `public URLEndpoint(java.lang.String url)` и метод `public java.lang.String getURL()`.

- **Класс `JAXMServlet`.** Класс, который расширяется классом Web-сервиса — получателем JAXM-сообщений. Расширяет `javax.servlet.http.HttpServlet` и дополнительно имеет следующие методы:

  - `public void init(javax.servlet.ServletConfig servletConfig)`

    Отвечает за инициализацию сервлета.

  - `public void setMessageFactory(javax.xml.soap.MessageFactory msgFactory)`

    Определяет объект `MessageFactory` для получения сообщений.

  - `protected static javax.xml.soap.MimeHeaders getHeaders(javax.servlet.http.HttpServletRequest req)`

    Возвращает заголовок запроса.

  - `protected static void putHeaders(javax.xml.soap.MimeHeaders headers, javax.servlet.http.HttpServletResponse res)`

    Определяет заголовок ответа.

  - `public void doPost(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse resp)`

    Принимает запрос и создает ответ.

- **Исключение `JAXMException`.** Представляет ошибку при отправке сообщения, установке соединения с поставщиком сообщений и др.

Как видно из приведенного списка, класс запрашиваемого Web-сервиса расширяет класс `JAXMServlet` и реализует интерфейс `OnewayListener` или `ReqRespListener`. Использование метода `onMessage()` интерфейса избавляет от необходимости переопределения метода `doPost()`.

Класс клиентского приложения, посылающего сообщение Web-сервису с помощью поставщика сообщений Messaging Provider, использует класс `ProviderConnectionFactory` и интерфейс `ProviderConnection` для установки связи с поставщиком сообщений Messaging Provider и отправки сообщения.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Пакет *javax.xml.soap*

- **Интерфейс `Detail`.** Представляет дочерние элементы `Detail` элемента `Fault` SOAP-сообщения; расширяет `SOAPFaultElement`.

  Интерфейс имеет следующие методы:

  - `DetailEntry addDetailEntry(Name name)`

    Добавляет новый элемент `Detail` с указанным именем.
  - `DetailEntry addDetailEntry(QName qname)`

    Добавляет новый элемент `Detail` с указанным QName-именем.
  - `Iterator getDetailEntries()`

    Возвращает итератор элементов `Detail`.
- **Интерфейс `DetailEntry`.** Представляет дочерний элемент `Detail` элемента `Fault` SOAP-сообщения, идентифицированный полным именем; расширяет `SOAPElement`.
- **Интерфейс `Name`.** Представляет XML-имя и имеет следующие методы:
  - `String getLocalName()`

    Возвращает локальное имя XML-элемента.
  - `String getQualifiedName()`

    Возвращает локальное имя с префиксом пространства имен.
  - `String getPrefix()`

    Возвращает префикс пространства имен.
  - `String getURI()`

    Возвращает пространство имен.
- **Интерфейс `Node`.** Представляет узел XML-документа и имеет следующие методы:
  - `String getValue()`

    Возвращает значение узла.
  - `void setValue(String value)`

    Устанавливает значение узла.
  - `void setParentElement(SOAPElement parent)`

    Устанавливает родительский узел для данного узла.
  - `SOAPElement getParentElement()`

    Возвращает родительский узел для данного узла.
  - `void detachNode()`

    Удаляет данный узел.
  - `void recycleNode()`

    Вызывается после метода `detachNode()` и дает возможность использования удаленного узла повторно в дальнейшем.

- **Интерфейс `SOAPBody`.** Представляет тело SOAP-сообщения, расширяет `SOAPElement` и имеет следующие методы:
  - `SOAPFault addFault()`

    Добавляет элемент `Fault` в тело SOAP-сообщения.
  - `SOAPFault addFault(Name faultCode, String faultString, Locale locale)`

    Добавляет элемент `Fault` в тело SOAP-сообщения с дочерним элементом, имеющим указанное имя кода ошибки, и дочерним элементом причины ошибки. Параметр `Locale` указывает язык текста причины ошибки.
  - `SOAPFault addFault(QName faultCode, String faultString, Locale locale)`

    Добавляет элемент `Fault` в тело SOAP-сообщения с дочерним элементом, имеющим указанное QName-имя кода ошибки, и дочерним элементом причины ошибки. Параметр `Locale` указывает язык текста причины ошибки.
  - `SOAPFault addFault(Name faultCode, String faultString)`

    Добавляет элемент `Fault` в тело SOAP-сообщения с дочерним элементом, имеющим указанное имя кода ошибки, и дочерним элементом причины ошибки.
  - `SOAPFault addFault(QName faultCode, String faultString)`

    Добавляет элемент `Fault` в тело SOAP-сообщения с дочерним элементом, имеющим указанное QName-имя кода ошибки, и дочерним элементом причины ошибки.
  - `boolean hasFault()`

    Указывает, существует ли элемент `Fault` в теле SOAP-сообщения.
  - `SOAPFault getFault()`

    Возвращает элемент `Fault` тела SOAP-сообщения.
  - `SOAPBodyElement addBodyElement(Name name)`

    Добавляет новый элемент с указанным именем в тело SOAP-сообщения.
  - `SOAPBodyElement addBodyElement(QName qname)`

    Добавляет новый элемент с указанным QName-именем в тело SOAP-сообщения.
  - `SOAPBodyElement addDocument(Document document)`

    Добавляет корневой элемент в тело SOAP-сообщения.
  - `Document extractContentAsDocument()`

    Возвращает объект, представляющий содержимое тела SOAP-сообщения.
- **Интерфейс `SOAPBodyElement`.** Представляет элемент тела SOAP-сообщения, расширяет `SOAPElement`.
- **Интерфейс `SOAPConstants`.** Определяет константы, относящиеся к SOAP-протоколу.

- **Интерфейс `SOAPElement`.** Базовый интерфейс для элементов SOAP-сообщения.

  Интерфейс имеет следующие методы:

  - `SOAPElement addChildElement(Name name)`

    Добавляет дочерний элемент с указанным именем.
  - `SOAPElement addChildElement(QName qname)`

    Добавляет дочерний элемент с указанным QName-именем.
  - `SOAPElement addChildElement(String localName)`

    Добавляет дочерний элемент с указанным локальным именем.
  - `SOAPElement addChildElement(String localName, String prefix)`

    Добавляет дочерний элемент с указанным локальным именем и префиксом.
  - `SOAPElement addChildElement(String localName, String prefix, String uri)`

    Добавляет дочерний элемент с указанным локальным именем, префиксом и пространством имен.
  - `SOAPElement addChildElement(SOAPElement element)`

    Добавляет определенный дочерний элемент.
  - `void removeContents()`

    Удаляет содержимое элемента.
  - `SOAPElement addTextNode(String text)`

    Добавляет текстовое значение элемента.
  - `SOAPElement addAttribute(Name name, String value)`

    Добавляет атрибут с указанным именем и значением в элемент.
  - `SOAPElement addAttribute(QName qname, String value)`

    Добавляет атрибут с указанным QName-именем и значением в элемент.
  - `SOAPElement addNamespaceDeclaration(String prefix, String uri)`

    Добавляет объявление пространства имен в элемент.
  - `String getAttributeValue(Name name)`

    Возвращает значение атрибута с указанным именем.
  - `String getAttributeValue(QName qname)`

    Возвращает значение атрибута с указанным QName-именем.
  - `Iterator getAllAttributes()`

    Возвращает итератор имен атрибутов элемента.
  - `Iterator getAllAttributesAsQNames()`

    Возвращает итератор QName-имен атрибутов элемента.
  - `String getNamespaceURI(String prefix)`

    Возвращает пространство имен элемента для указанного префикса.

- `Iterator getNamespacePrefixes()`

  Возвращает итератор префиксов пространств имен элемента.

- `Iterator getVisibleNamespacePrefixes()`

  Возвращает итератор префиксов пространств имен видимой области элемента.

- `QName createQName(String localName, String prefix)`

  Создает QName-имя элемента.

- `Name getElementName()`

  Возвращает имя элемента.

- `QName getElementQName()`

  Возвращает QName-имя элемента.

- `SOAPElement setElementQName(QName newName)`

  Изменяет имя элемента.

- `boolean removeAttribute(Name name)`

  Удаляет атрибут с указанным именем.

- `boolean removeAttribute(QName qname)`

  Удаляет атрибут с указанным QName-именем.

- `boolean removeNamespaceDeclaration(String prefix)`

  Удаляет объявление пространства имен элемента.

- `Iterator getChildElements()`

  Возвращает итератор дочерних элементов данного элемента.

- `Iterator getChildElements(Name name)`

  Возвращает итератор дочерних элементов с указанным именем.

- `Iterator getChildElements(QName qname)`

  Возвращает итератор дочерних элементов с указанным QName-именем.

- `void setEncodingStyle(String encodingStyle)`

  Устанавливает стиль кодировки для данного элемента.

- `String getEncodingStyle()`

  Возвращает стиль кодировки элемента.

❒ **Интерфейс `SOAPEnvelope`.** Представляет элемент `Envelope` SOAP-сообщения, расширяет `SOAPElement` и имеет следующие методы:

- `Name createName(String localName, String prefix, String uri)`

  Создает имя элемента, состоящее из локального имени, префикса и пространства имен.

- `Name createName(String localName)`

  Создает имя элемента из локального имени.

- `SOAPHeader getHeader()`

  Возвращает заголовок SOAP-сообщения.
- `SOAPBody getBody()`

  Возвращает тело SOAP-сообщения.
- `SOAPHeader addHeader()`

  Добавляет заголовок SOAP-сообщения.
- `SOAPBody addBody()`

  Добавляет тело SOAP-сообщения.

❒ **Интерфейс `SOAPFault`.** Представляет элемент `Fault` SOAP-сообщения, расширяет `SOAPBodyElement` и имеет следующие методы:

- `void setFaultCode(Name faultCodeQName)`

  Устанавливает элемент `Fault` с именованным кодом ошибки.
- `void setFaultCode(QName faultCodeQName)`

  Устанавливает элемент `Fault` с QName-именованным кодом ошибки.
- `void setFaultCode(String faultCode)`

  Устанавливает элемент `Fault` с именованным строкой кодом ошибки.
- `Name getFaultCodeAsName()`

  Возвращает имя кода ошибки.
- `QName getFaultCodeAsQName()`

  Возвращает QName-имя кода ошибки.
- `Iterator getFaultSubcodes()`

  Возвращает итератор подкодов ошибки.
- `void removeAllFaultSubcodes()`

  Удаляет подкоды ошибки.
- `void appendFaultSubcode(QName subcode)`

  Добавляет подкод ошибки.
- `String getFaultCode()`

  Возвращает код ошибки.
- `void setFaultActor(String faultActor)`

  Устанавливает адресат, вызвавший ошибку.
- `String getFaultActor()`

  Возвращает идентификатор адресата, вызвавшего ошибку.
- `void setFaultString(String faultString)`

  Устанавливает текст причины ошибки.
- `void setFaultString(String faultString, Locale locale)`

  Устанавливает текст причины ошибки и его язык.

- `String getFaultString()`

  Возвращает текст причины ошибки.

- `Locale getFaultStringLocale()`

  Возвращает язык текста причины ошибки.

- `boolean hasDetail()`

  Возвращает `true`, если есть элемент `Detail`.

- `Detail getDetail()`

  Возвращает объект `Detail`.

- `Detail addDetail()`

  Добавляет объект `Detail`.

- `Iterator getFaultReasonLocales()`

  Возвращает итератор языков элементов `Reason/Text`.

- `Iterator getFaultReasonTexts()`

  Возвращает итератор элементов `Reason/Text`.

- `String getFaultReasonText(Locale locale)`

  Возвращает значение элемента `Reason/Text` с указанным языком.

- `void addFaultReasonText(String text, Locale locale)`

  Добавляет значение элемента `Reason/Text` с указанным языком.

- `String getFaultNode()`

  Возвращает значение элемента `Node`.

- `void setFaultNode(String uri)`

  Устанавливает значение элемента `Node`.

- `String getFaultRole()`

  Возвращает значение элемента `Role`.

- `void setFaultRole(String uri)`

  Устанавливает значение элемента `Role`.

❒ **Интерфейс `SOAPFaultElement`.** Представляет дочерние элементы элемента `Fault`, расширяет `SOAPElement`.

❒ **Интерфейс `SOAPHeader`.** Представляет заголовок SOAP-сообщения, расширяет `SOAPElement` и имеет следующие методы:

- `SOAPHeaderElement addHeaderElement(Name name)`

  Добавляет новый блок заголовка с указанным именем.

- `SOAPHeaderElement ddHeaderElement(QName qname)`

  Добавляет новый блок заголовка с указанным QName-именем.

- `Iterator examineMustUnderstandHeaderElements(String actor)`

  Возвращает итератор блоков заголовка для указанного адресата и с атрибутом `MustUnderstand="true"`.

- `Iterator examineHeaderElements(String actor)`

  Возвращает итератор блоков заголовка для указанного адресата.

- `Iterator extractHeaderElements(String actor)`

  Возвращает итератор блоков заголовка для указанного адресата и удаляет их.

- `SOAPHeaderElement addNotUnderstoodHeaderElement(QName name)`

  Добавляет блок заголовка `NotUnderstood` в сообщении об ошибке.

- `SOAPHeaderElement addUpgradeHeaderElement(Iterator supportedSOAPURIs)`

  Добавляет блок заголовка `Upgrade` в сообщении об ошибке с указанным списком URI поддерживаемых SOAP-спецификаций.

- `SOAPHeaderElement addUpgradeHeaderElement(String[] supportedSoapUris)`

  Добавляет блок заголовка `Upgrade` в сообщении об ошибке с указанным массивом URI поддерживаемых SOAP-спецификаций.

- `SOAPHeaderElement addUpgradeHeaderElement(String supportedSoapUri)`

  Добавляет блок заголовка `Upgrade` в сообщении об ошибке с указанным URI поддерживаемой SOAP-спецификации.

- `Iterator examineAllHeaderElements()`

  Возвращает итератор блоков заголовков.

- `Iterator extractAllHeaderElements()`

  Возвращает итератор блоков заголовков и удаляет их.

❒ **Интерфейс `SOAPHeaderElement`.** Представляет блок заголовка, расширяет `SOAPElement` и имеет следующие методы:

- `void setActor(String actorURI)`

  Устанавливает адресата блока заголовка, по умолчанию `SOAPConstants.URI_SOAP_ACTOR_NEXT`.

- `void setRole(String uri)`

  Устанавливает значение атрибута `Role`.

- `String getActor()`

  Возвращает значение атрибута `actor`.

- `String getRole()`

  Возвращает значение атрибута `Role`.

- `void setMustUnderstand(boolean mustUnderstand)`

  Устанавливает значение атрибута `mustUnderstand`.

- `boolean getMustUnderstand()`

  Возвращает значение атрибута `mustUnderstand`.

- `void setRelay(boolean relay)`

  Устанавливает значение атрибута `relay`.

- `boolean getRelay()`

  Возвращает значение атрибута `relay`.

❒ **Интерфейс `Text`.** Представляет элемент `Text`, имеет метод `boolean isComment()`, возвращающий `true`, если содержимое элемента — комментарии.

❒ **Класс `AttachmentPart`.** Представляет вложение SOAP-сообщения, состоящее из MIME-заголовков и самого вложения.

Класс имеет следующие методы:

- `public abstract int getSize()`

  Возвращает размер вложения в байтах.

- `public abstract void clearContent()`

  Удаляет содержимое самого вложения без MIME-заголовков.

- `public abstract Object getContent()`

  Возвращает содержимое вложения как объект.

- `public abstract InputStream getRawContent()`

  Возвращает содержимое вложения как поток.

- `public abstract byte[] getRawContentBytes()`

  Возвращает содержимое вложения как массив.

- `public abstract InputStream getBase64Content()`

  Возвращает содержимое вложения как поток Base64-символов.

- `public abstract void setContent(Object object, String contentType)`

  Устанавливает содержимое вложения из объекта с указанным заголовком `Content-Type`.

- `public abstract void setRawContent(InputStream content, String contentType)`

  Устанавливает содержимое вложения из потока с указанным заголовком `Content-Type`.

- `public abstract void setRawContentBytes(byte[] content, int offset, int len, String contentType)`

  Устанавливает содержимое вложения из массива с указанным заголовком `Content-Type`.

- `public abstract void setBase64Content(InputStream content, String contentType)`

  Устанавливает содержимое вложения из Base64-потока с указанным заголовком `Content-Type`.

- `public abstract DataHandler getDataHandler()`

  Возвращает объект класса `javax.activation.DataHandler`, предназначенный для управления данными.

- `public abstract void setDataHandler(DataHandler dataHandler)`

  Устанавливает объект `DataHandler` для вложения.

- `public String getContentId()`

  Возвращает значение MIME-заголовка `Content-ID`.

- `public String getContentLocation()`

  Возвращает значение MIME-заголовка `Content-Location`.

- `public String getContentType()`

  Возвращает значение MIME-заголовка `Content-Type`.

- `public void setContentId(String contentId)`

  Устанавливает значение MIME-заголовка `Content-ID`.

- `public void setContentLocation(String contentLocation)`

  Устанавливает значение MIME-заголовка `Content-Location`.

- `public void setContentType(String contentType)`

  Устанавливает значение MIME-заголовка `Content-Type`.

- `public abstract void removeMimeHeader(String header)`

  Удаляет MIME-заголовки с указанным именем.

- `public abstract void removeAllMimeHeaders()`

  Удаляет все MIME-заголовки.

- `public abstract String[] getMimeHeader(String name)`

  Возвращает значения MIME-заголовков с указанным именем.

- `public abstract void setMimeHeader(String name, String value)`

  Изменяет значение MIME-заголовка с указанным именем.

- `public abstract void addMimeHeader(String name, String value)`

  Добавляет MIME-заголовок с указанным именем и значением.

- `public abstract Iterator getAllMimeHeaders()`

  Возвращает итератор MIME-заголовков.

- `public abstract Iterator getMatchingMimeHeaders(String[] names)`

  Возвращает итератор MIME-заголовков с фильтром по именам.

- `public abstract Iterator getNonMatchingMimeHeaders(String[] names)`

  Возвращает итератор MIME-заголовков с фильтром по несовпадению имен.

❒ **Класс `MessageFactory`.** Класс-фабрика для создания объектов `SOAPMessage`, представляющих SOAP-сообщения.

Класс имеет следующие методы:

- `public static MessageFactory newInstance()`

  Создает новый объект `MessageFactory`, соответствующий реализации протокола SOAP 1.1 по умолчанию.

- `public static MessageFactory newInstance(String protocol)`

  Создает новый объект `MessageFactory`, соответствующий указанной реализации протокола, например `SOAPConstants.SOAP_1_2_PROTOCOL`.

- `public abstract SOAPMessage createMessage()`

  Создает новое SOAP-сообщение с заголовком и телом по умолчанию.

- `public abstract SOAPMessage createMessage(MimeHeaders headers, InputStream in)`

  Создает новое SOAP-сообщение из потока.

❒ **Класс `MimeHeader`.** Представляет MIME-заголовок и имеет следующие методы:

- `public String getName()`

  Возвращает имя заголовка.

- `public String getValue()`

  Возвращает значение заголовка.

❒ **Класс `MimeHeaders`.** Является контейнером для объектов `MimeHeader` и имеет следующие методы:

- `public String[] getHeader(String name)`

  Возвращает все значения заголовка с указанным именем.

- `public void setHeader(String name, String value)`

  Изменяет значение заголовка с указанным именем.

- `public void addHeader(String name, String value)`

  Добавляет заголовок в контейнер.

- `public void removeHeader(String name)`

  Удаляет заголовки с указанным именем из контейнера.

- `public void removeAllHeaders()`

  Удаляет все заголовки из контейнера.

- `public Iterator getAllHeaders()`

  Возвращает итератор заголовков в контейнере.

- `public Iterator getMatchingHeaders(String[] names)`

  Возвращает итератор заголовков в контейнере с фильтром по именам.

- `public Iterator getNonMatchingHeaders(String[] names)`

  Возвращает итератор заголовков в контейнере с фильтром по несовпадению имен.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- **Класс `SAAJMetaFactory`.** Класс реализации спецификации SAAJ.
- **Класс `SAAJResult`.** Представляет результаты преобразования JAXP или JAXB в виде дерева.

  Класс имеет метод `public Node getResult()`, возвращающий результирующее дерево.
- **Класс `SOAPConnection`.** Представляет соединение клиента с адресатом и имеет следующие методы:
  - `public abstract SOAPMessage call(SOAPMessage request, Object to)`

    Посылает синхронный запрос конечной точке, определенной вторым параметром типа `java.lang.String`, `java.net.URL` или `javax.xml.messaging.URLEndpoint`.
  - `public SOAPMessage get(Object to)`

    Получает сообщение от конечной точки.
  - `public abstract void close()`

    Закрывает соединение.
- **Класс `SOAPConnectionFactory`.** Класс-фабрика для создания объектов `SOAPConnection`.

  Класс имеет следующие методы:
  - `public static SOAPConnectionFactory newInstance()`

    Создает объект `SOAPConnectionFactory`.
  - `public abstract SOAPConnection createConnection()`

    Создает соединение `SOAPConnection`.
- **Класс `SOAPFactory`.** Класс-фабрика для создания XML-фрагментов.

  Класс имеет следующие методы:
  - `public SOAPElement createElement(Element domElement)`

    Создает SOAP-элемент из DOM-элемента.
  - `public abstract SOAPElement createElement(Name name)`

    Создает SOAP-элемент с указанным именем.
  - `public SOAPElement createElement(QName qname)`

    Создает SOAP-элемент с указанным QName-именем.
  - `public abstract SOAPElement createElement(String localName)`

    Создает SOAP-элемент с указанным локальным именем.
  - `public abstract SOAPElement createElement(String localName, String prefix, String uri)`

    Создает SOAP-элемент с указанным локальным именем, префиксом и пространством имен.
  - `public abstract Detail createDetail()`

    Создает новый объект `Detail`.

- `public abstract SOAPFault createFault(String reasonText, QName faultCode)`

  Создает новый объект `Fault` с кодом ошибки и ее причинами.

- `public abstract SOAPFault createFault()`

  Создает новый объект `Fault` по умолчанию.

- `public abstract Name createName(String localName, String prefix, String uri)`

  Создает имя из локального имени, префикса и пространства имен.

- `public abstract Name createName(String localName)`

  Формирует имя из локального имени.

- `public static SOAPFactory newInstance()`

  Создает новый объект `SOAPFactory` для протокола SOAP 1.1.

- `public static SOAPFactory newInstance(String protocol)`

  Создает новый объект `SOAPFactory` для указанного протокола.

❒ **Класс `SOAPMessage`.** Представляет SOAP-сообщение и имеет следующие методы:

- `public abstract void setContentDescription(String description)`

  Устанавливает описание содержимого сообщения.

- `public abstract String getContentDescription()`

  Возвращает описание содержимого сообщения.

- `public abstract SOAPPart getSOAPPart()`

  Возвращает основную часть сообщения, состоящую из MIME-заголовков, `Envelope`-, `Body`- и `Header`-элементов.

- `public SOAPBody getSOAPBody()`

  Возвращает тело сообщения.

- `public SOAPHeader getSOAPHeader()`

  Возвращает заголовок сообщения.

- `public abstract void removeAllAttachments()`

  Удаляет все вложения сообщения.

- `public abstract int countAttachments()`

  Возвращает количество вложений.

- `public abstract Iterator getAttachments()`

  Возвращает итератор вложений.

- `public abstract Iterator getAttachments(MimeHeaders headers)`

  Возвращает итератор вложений с фильтрацией по заголовкам.

- `public abstract void removeAttachments(MimeHeaders headers)`

  Удаляет вложения с фильтрацией по заголовкам.

- `public abstract AttachmentPart getAttachment(SOAPElement element)`

  Возвращает вложение по ссылке.
- `public abstract void addAttachmentPart(AttachmentPart AttachmentPart)`

  Добавляет вложение.
- `public abstract AttachmentPart createAttachmentPart()`

  Создает новое пустое вложение.
- `public AttachmentPart createAttachmentPart (javax.activation.DataHandler dataHandler)`

  Создает новое вложение.
- `public abstract MimeHeaders getMimeHeaders()`

  Возвращает MIME-заголовки.
- `public AttachmentPart createAttachmentPart(Object content, String contentType)`

  Создает новое вложение с содержимым.
- `public abstract void saveChanges()`

  Обновляет объект `SOAPMessage`.
- `public abstract boolean saveRequired()`

  Если возвращает `true`, тогда требуется вызов метода `saveChanges()`.
- `public abstract void writeTo(OutputStream out)`

  Создает исходящий поток сообщения.
- `public void setProperty(String property, Object value)`

  Устанавливает свойства объекта `SOAPMessage`, такие как `WRITE_XML_DECLARATION` и `CHARACTER_SET_ENCODING`.
- `public Object getProperty(String property)`

  Возвращает значение свойства объекта `SOAPMessage`.

❒ **Класс `SOAPPart`.** Представляет часть SOAP-сообщения, состоящую из MIME-заголовков, `Envelope`-, `Body`- и `Header`-элементов.

Класс имеет следующие методы:

- `public abstract SOAPEnvelope getEnvelope()`

  Возвращает конверт сообщения.
- `public String getContentId()`

  Возвращает MIME-заголовок `Content-Id`.
- `public String getContentLocation()`

  Возвращает MIME-заголовок `Content-Location`.
- `public void setContentId(String contentId)`

  Устанавливает значение MIME-заголовка `Content-Id`.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public void setContentLocation(String contentLocation)`

  Устанавливает значение MIME-заголовка `Content-Location`.
- `public abstract void removeMimeHeader(String header)`

  Удаляет MIME-заголовки с данным именем.
- `public abstract void removeAllMimeHeaders()`

  Удаляет все MIME-заголовки.
- `public abstract String[] getMimeHeader(String name)`

  Возвращает значения MIME-заголовка.
- `public abstract void setMimeHeader(String name, String value)`

  Устанавливает MIME-заголовок.
- `public abstract void addMimeHeader(String name, String value)`

  Добавляет MIME-заголовок.
- `public abstract Iterator getAllMimeHeaders()`

  Возвращает итератор MIME-заголовков.
- `public abstract Iterator getMatchingMimeHeaders(String[] names)`

  Возвращает итератор MIME-заголовков с фильтрацией по именам.
- `public abstract Iterator getNonMatchingMimeHeaders(String[] names)`

  Возвращает итератор MIME-заголовков с фильтрацией по несоответствию именам.
- `public abstract void setContent(javax.xml.transform.Source source)`

  Устанавливает содержимое конверта сообщения.
- `public abstract Source getContent()`

  Возвращает содержимое конверта сообщения.

❒ **Исключение `SOAPException`.** Представляет ошибки, связанные с установлением соединения, отправкой сообщения и др.

Как видно из приведенного списка, пакет `javax.xml.soap` дает возможность создания и детальной обработки SOAP-сообщений. Кроме того, пакет `javax.xml.soap` с помощью класса `SOAPConnection` позволяет установить соединение с конечной точкой и отправить/получить от нее синхронное сообщение.

JAXP API

Пакет *javax.xml.parsers*

Пакет `javax.xml.parsers` содержит классы `DocumentBuilder` и `SAXParser`, представляющие DOM- и SAX-анализаторы, а также классы-фабрики `DocumentBuilderFactory` и `SAXParserFactory`, предназначенные для создания экземпляров DOM- и SAX-анализаторов.

Класс `DocumentBuilder` позволяет получить объект `org.w3c.dom.Document`, представляющий дерево XML-документа, из XML-документа. Класс `DocumentBuilder` имеет перечисленные далее методы.

- `public void reset()`

  Устанавливает DOM-анализатор в исходное состояние.

- `public Document parse(InputStream is)`

  Создает объект `Document` из входящего потока.

- `public Document parse(InputStream is, String systemId)`

  Создает объект `Document` из входящего потока с учетом базового URI.

- `public Document parse(String uri)`

  Создает объект `Document` из XML-документа, находящегося по указанному URI-адресу.

- `public Document parse(File f)`

  Создает объект `Document` из XML-документа, находящегося в указанном файле.

- `public abstract Document parse(InputSource is)`

  Создает объект `Document` из объекта `org.xml.sax.InputSource`, содержащего информацию о XML-документе.

- `public abstract boolean isNamespaceAware()`

  Возвращает `true`, если DOM-анализатор распознает пространства имен.

- `public abstract boolean isValidating()`

  Возвращает `true`, если DOM-анализатор осуществляет проверку XML-документа относительно его схемы.

- `public abstract void setEntityResolver(EntityResolver er)`

  Устанавливает объект `org.xml.sax.EntityResolver` для распознавания внешних сущностей, представленных в анализируемом XML-документе.

- `public abstract void setErrorHandler(ErrorHandler eh)`

  Устанавливает объект `org.xml.sax.ErrorHandler` для управления ошибками разбора XML-документа.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public abstract Document newDocument()`

  Создает новый экземпляр объекта `Document`.

- `public abstract DOMImplementation getDOMImplementation()`

  Создает экземпляр объекта `org.w3c.dom.DOMImplementation`, представляющего DOM-реализацию.

- `public Schema getSchema()`

  Возвращает объект `javax.xml.validation.Schema`, представляющий схему XML-документа.

- `public boolean isXIncludeAware()`

  Возвращает `true`, если DOM-анализатор поддерживает механизм XML Inclusions включений в XML-документы текстовых файлов или других XML-документов или их частей.

Класс `SAXParser` осуществляет разбор XML-документа с помощью интерфейса `org.xml.sax.XMLReader` и имеет следующие методы:

- `public void reset()`

  Устанавливает SAX-анализатор в исходное состояние.

- `public void parse(InputStream is, DefaultHandler dh)`

  Осуществляет анализ XML-документа из входящего потока, используя обработчик событий `org.xml.sax.helpers.DefaultHandler`.

- `public void parse(InputStream is, DefaultHandler dh, String systemId)`

  Осуществляет анализ XML-документа из входящего потока, используя обработчик событий `org.xml.sax.helpers.DefaultHandler` с учетом базового URI.

- `public void parse(String uфri, DefaultHandler dh)`

  Осуществляет разбор XML-документа, находящегося по указанному URI-адресу.

- `public void parse(File f, DefaultHandler dh)`

  Осуществляет разбор XML-документа, находящегося в указанном файле.

- `public void parse(InputSource is, DefaultHandler dh)`

  Осуществляет разбор XML-документа, информация о котором содержится в объекте `org.xml.sax.InputSource`.

- `public abstract XMLReader getXMLReader()`

  Возвращает объект `org.xml.sax.XMLReader`, представляющий SAX-анализатор.

- `public abstract boolean isNamespaceAware()`

  Возвращает `true`, если SAX-анализатор распознает пространства имен.

- `public abstract boolean isValidating()`

  Возвращает `true`, если SAX-анализатор осуществляет проверку XML-документа относительно его схемы.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public abstract void setProperty(String name, Object value)`

  Устанавливает свойства SAX-анализатора.

- `public abstract Object getProperty(String name)`

  Возвращает свойства SAX-анализатора.

- `public Schema getSchema()`

  Возвращает объект `javax.xml.validation.Schema`, представляющий схему XML-документа.

- `public boolean isXIncludeAware()`

  Возвращает `true`, если SAX-анализатор поддерживает механизм XML Inclusions включений в XML-документы текстовых файлов или других XML-документов или их частей.

StAX

Пакет `javax.xml.stream` позволяет осуществлять разбор XML-документов с помощью двух интерфейсов — `XMLStreamReader` и `XMLEventReader`, отличающихся тем, что первый интерфейс разбирает XML-поток как курсор (Cursor API), а второй интерфейс — как итератор (Event Iterator API), преобразующий XML-поток в поток объектов событий.

Интерфейс `XMLStreamReader` имеет следующие методы:

- `Object getProperty(String name)`

  Возвращает значения свойств StAX-анализатора.

- `int next()`

  Передвигает курсор на следующую позицию, возвращая код типа события.

- `void require(int type, String namespaceURI, String localName)`

  Тестирует текущее событие на соответствие типу, пространству имен и локальному имени. Возможные типы событий — это `START_ELEMENT`, `ATTRIBUTE`, `NAMESPACE`, `END_ELEMENT`, `CHARACTERS`, `CDATA`, `COMMENT`, `SPACE`, `START_DOCUMENT`, `END_DOCUMENT`, `PROCESSING_INSTRUCTION`, `ENTITY_REFERENCE` и `DTD`.

- `String getElementText()`

  Возвращает содержимое текстового элемента.

- `int nextTag()`

  Передвигает курсор на следующий начальный или конечный XML-тег, минуя пробелы, комментарии и инструкции по обработке, возвращая код типа события.

- `boolean hasNext()`

  Передвигает курсор на следующую позицию, возвращая `true`, если далее есть другие события.

- `void close()`

  Освобождает ресурсы, связанные со StAX-анализатором.

- `String getNamespaceURI(String prefix)`

  Возвращает пространство имен для данного префикса.

- `boolean isStartElement()`

  Возвращает `true`, если курсор находится на начальном теге элемента.

- `boolean isEndElement()`

  Возвращает `true`, если курсор находится на закрывающем теге элемента.

- `boolean isCharacters()`

  Возвращает `true`, если курсор находится на символьных данных.

- `boolean isWhiteSpace()`

  Возвращает `true`, если курсор находится на пробеле.

- `String getAttributeValue(String namespaceURI, String localName)`

  Возвращает значение атрибута.

- `int getAttributeCount()`

  Возвращает количество атрибутов элемента.

- `QName getAttributeName(int index)`

  Возвращает QName-имя атрибута с данным номером.

- `String getAttributeNamespace(int index)`

  Возвращает пространство имен атрибута с данным номером.

- `String getAttributeLocalName(int index)`

  Возвращает локальное имя атрибута с данным номером.

- `String getAttributePrefix(int index)`

  Возвращает префикс атрибута с данным номером.

- `String getAttributeType(int index)`

  Возвращает XML-тип атрибута с данным номером.

- `String getAttributeValue(int index)`

  Возвращает значение атрибута с данным номером.

- `boolean isAttributeSpecified(int index)`

  Возвращает `true`, если данный атрибут создан по умолчанию.

- `int getNamespaceCount()`

  Возвращает количество продекларированных пространств имен элемента.

- `String getNamespacePrefix(int index)`

  Возвращает префикс пространства имен с данным номером.

- `String getNamespaceURI(int index)`

  Возвращает URI пространства имен с данным номером.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `NamespaceContext getNamespaceContext()`

  Возвращает для данной позиции объект `javax.xml.namespace.NamespaceContext`, содержащий информацию о префиксах и URI пространства имен.

- `int getEventType()`

  Возвращает код типа события для данной позиции.

- `String getText()`

  Возвращает текстовое значение текущего события.

- `char[] getTextCharacters()`

  Возвращает символы текущего события.

- `int getTextCharacters(int sourceStart, char[] target, int targetStart, int length)`

  Копирует текст в символьный массив и возвращает количество скопированных символов.

- `int getTextStart()`

  Возвращает индекс первого символа текста в массиве.

- `int getTextLength()`

  Возвращает длину текста.

- `String getEncoding()`

  Возвращает кодировку текста.

- `boolean hasText()`

  Возвращает `true`, если текущее событие содержит текст (для событий `CHARACTERS`, `DTD`, `ENTITY_REFERENCE`, `COMMENT`, `SPACE`).

- `Location getLocation()`

  Возвращает объект `javax.xml.stream.Location`, содержащий информацию о местонахождении события. Интерфейс `Location` имеет методы:

  - `int getLineNumber()` — возвращает номер строки окончания события;
  - `int getColumnNumber()` — возвращает номер столбца окончания события;
  - `int getCharacterOffset()` — возвращает позицию события в потоке;
  - `String getPublicId()` — возвращает публичный идентификатор документа;
  - `String getSystemId()` — возвращает системный идентификатор документа.

- `QName getName()`

  Возвращает QName-имя элемента.

- `String getLocalName()`

  Возвращает локальное имя события (для событий `START_ELEMENT`, `END_ELEMENT`, `ENTITY_REFERENCE`).

- `boolean hasName()`

  Возвращает `true`, если события `START_ELEMENT` или `END_ELEMENT` имеют имя.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `String getNamespaceURI()`

  Для событий `START_ELEMENT` или `END_ELEMENT` возвращает пространство имен по умолчанию.

- `String getPrefix()`

  Возвращает префикс текущего события.

- `String getVersion()`

  Возвращает продекларированную XML-версию.

- `boolean isStandalone()`

  Возвращает `true`, если документ автономен.

- `boolean standaloneSet()`

  Возвращает `true`, если автономность документа продекларирована.

- `String getCharacterEncodingScheme()`

  Возвращает объявленную кодировку документа.

- `String getPITarget()`

  Возвращает значение атрибута `target` инструкции по обработке.

- `String getPIData()`

  Возвращает инструкцию по обработке.

Интерфейс `XMLEventReader` имеет следующие методы:

- `XMLEvent nextEvent()`

  Возвращает объект `javax.xml.stream.events.XMLEvent`, представляющий следующее событие в XML-потоке.

- `boolean hasNext()`

  Возвращает `true`, если существуют еще события.

- `XMLEvent peek()`

  Возвращает следующее событие без перемещения итератора.

- `String getElementText()`

  Возвращает содержимое текстового элемента.

- `XMLEvent nextTag()`

  Возвращает следующее событие `START_ELEMENT` или `END_ELEMENT`.

- `Object getProperty(String name)`

  Возвращает значения свойств StAX-анализатора.

- `void close()`

  Освобождает ресурсы, задействованные StAX-анализатором.

- `<E> next()`

  Метод, унаследованный от `java.util.Iterator`, возвращает следующий элемент итерации.

- `void remove()`

  Метод, унаследованный от `java.util.Iterator`, удаляет из коллекции последний элемент, возвращенный итератором.

Интерфейсы `XMLStreamWriter` и `XMLEventWriter` пакета `javax.xml.stream` позволяют записывать XML-документы.

Интерфейс `XMLStreamWriter` имеет следующие методы:

- `void writeStartElement(String localName)`

  Записывает начальный тег элемента с локальным именем.

- `void writeStartElement(String namespaceURI, String localName)`

  Записывает начальный тег элемента с локальным именем и пространством имен.

- `void writeStartElement(String prefix, String locфalName, String namespaceURI)`

  Записывает начальный тег элемента с локальным именем, префиксом и пространством имен.

- `void writeEmptyElement(String localName)`

  `void writeEmptyElement(String namespaceURI, String localName)`

  `void writeEmptyElement(String prefix, String localName, String namespaceURI)`

  Записывают пустой элемент.

- `void writeEndElement()`

  Записывает закрывающий тег элемента.

- `void writeEndDocument()`

  Закрывает все начальные теги.

- `void close()`

  Освобождает все ресурсы, связанные с записью документа.

- `void flush()`

  Выводит на запись все кэшированные данные.

- `void writeAttribute(String localName, String value)`

  `void writeAttribute(String prefix, String namespaceURI, String localName, String value)`

  `void writeAttribute(String namespaceURI, String localName, String value)`

  Записывают атрибут элемента.

- `void writeNamespace(String prefix, String namespaceURI)`

  `void writeDefaultNamespace(String namespaceURI)`

  Декларируют пространство имен.

- `void writeComment(String data)`

  Записывает комментарии.

- `void writeProcessingInstruction(String target)`
  `void writeProcessingInstruction(String target, String data)`
  Записывают инструкции по обработке.
- `void writeCData(String data)`
  Записывает раздел символьных данных `CDATA`.
- `void writeDTD(String dtd)`
  Записывает раздел DTD-описание.
- `void writeEntityRef(String name)`
  Записывает ссылку.
- `void writeStartDocument()`
  `void writeStartDocument(String version)`
  `void writeStartDocument(String encoding, String version)`
  Записывают начальную XML-декларацию.
- `void writeCharacters(String text)`
  `void writeCharacters(char[] text, int start, int len)`
  Записывают текст.
- `String getPrefix(String uri)`
  Возвращает префикс.
- `void setPrefix(String prefix, String uri)`
  Устанавливает префикс для URI.
- `void setDefaultNamespace(String uri)`
  Устанавливает URI как пространство имен по умолчанию.
- `void setNamespaceContext(NamespaceContext context)`
  Устанавливает пространство имен и префикс для записи.
- `NamespaceContext getNamespaceContext()`
  Возвращает текущий контекст пространства имен.
- `Object getProperty(String name)`
  Возвращает свойства StAX-реализации.

Интерфейс `XMLEventWriter` имеет следующие методы:

- `void flush()`
  Выводит на запись все кэшированные данные.
- `void close()`
  Освобождает все ресурсы, связанные с записью.
- `void add(XMLEvent event)`
  Выводит объект события на запись.
- `void add(XMLEventReader reader)`
  Выводит весь поток объектов событий на запись.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `String getPrefix(String uri)`

  Возвращает префикс.

- `void setPrefix(String prefix, String uri)`

  Устанавливает префикс.

- `void setDefaultNamespace(String uri)`

  Устанавливает URI как пространство имен по умолчалнию.

- `void setNamespaceContext(NamespaceContext context)`

  Устанавливает контекст пространства имен для записи.

- `NamespaceContext getNamespaceContext()`

  Возвращает текущий контекст пространства имен.

Объекты `XMLEventReader` и `XMLStreamReader` создаются с помощью класса-фабрики `javax.xml.stream.XMLInputFactory`, причем класс `XMLInputFactory` позволяет создавать входящие XML-потоки с применением фильтров. Объекты `XMLEventWriter` и `XMLStreamWriter` создаются с помощью класса-фабрики `javax.xml.stream.XMLOutputFactory`.

Класс `XMLInputFactory` имеет ряд свойств.

- `public static final String IS_NAMESPACE_AWARE`

  Устанавливает поддержку StAX-анализатором пространств имен.

- `public static final String IS_VALIDATING`

  Устанавливает поддержку проверки документа относительно его схемы.

- `public static final String IS_COALESCING`

  Устанавливает требование к StAX-анализатору объединять смежные секции `CDATA`.

- `public static final String IS_REPLACING_ENTITY_REFERENCES`

  Устанавливает требование к StAX-анализатору заменять внутренние ссылки данными.

- `public static final String IS_SUPPORTING_EXTERNAL_ENTITIES`

  Устанавливает поддержку внешних ссылок.

- `public static final String SUPPORT_DTD`

  Устанавливает поддержку DTD-схемы документа.

- `public static final String REPORTER`

  Устанавливает реализацию интерфейса `javax.xml.stream.XMLReporter`. Интерфейс `XMLReporter` предназначен для различного рода предупреждений с помощью метода `void report(String message, String errorType, Object relatedInformation, Location location)`, создающего сообщение о некритичной ошибке.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public static final String RESOLVER`

  Устанавливает реализацию интерфейса `javax.xml.stream.XMLResolver`. Интерфейс `XMLResolver` отвечает за поиск ресурсов при анализе XML-документа, используя метод `Object resolveEntity(String publicID, String systemID, String baseURI, String namespace)`.

- `public static final String ALLOCATOR`

  Устанавливает реализацию интерфейса `javax.xml.stream.util.XMLEventAllocator`. Интерфейс `XMLEventAllocator` является мостом между `XMLStreamReader` и `XMLEventReader`, предоставляя методы `XMLEvent allocate(XMLStreamReader reader)` — отображает курсор на объект события и `void allocate(XMLStreamReader reader, XMLEventConsumer consumer)` — отображает курсор на объекты событий и передает их потребителю событий. Интерфейс `javax.xml.stream.util.XMLEventConsumer` маркирует объект для приема событий, используя метод `void add(XMLEvent event)`.

Класс `XMLInputFactory` также имеет набор перечисленных далее методов.

- `public static XMLInputFactory newInstance()`

  `public static XMLInputFactory newInstance(String factoryId, ClassLoader classLoader)`

  Создают экземпляр фабрики.

- `public abstract XMLStreamReader createXMLStreamReader(java.io.Reader reader)`

  `public abstract XMLStreamReader createXMLStreamReader(javax.xml.transform.Source source)`

  `public abstract XMLStreamReader createXMLStreamReader(java.io.InputStream stream)`

  `public abstract XMLStreamReader createXMLStreamReader(java.io.InputStream stream, String encoding)`

  `public abstract XMLStreamReader createXMLStreamReader(String systemId, java.io.InputStream stream)`

  `public abstract XMLStreamReader createXMLStreamReader(String systemId, java.io.Reader reader)`

  Создают новый объект `XMLStreamReader` из различных источников.

- `public abstract XMLEventReader createXMLEventReader(Reader reader)`

  `public abstract XMLEventReader createXMLEventReader(String systemId, Reader reader)`

  `public abstract XMLEventReader createXMLEventReader(XMLStreamReader reader)`

  `public abstract XMLEventReader createXMLEventReader(Source source)`

  `public abstract XMLEventReader createXMLEventReader(InputStream stream)`

  `public abstract XMLEventReader createXMLEventReader(InputStream stream, String encoding)`

  `public abstract XMLEventReader createXMLEventReader(String systemId, InputStream stream)`

  Создают объект `XMLEventReader` из различных источников.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public abstract XMLStreamReader createFilteredReader(XMLStreamReader reader, StreamFilter filter)`

  Добавляет к объекту `XMLStreamReader` фильтр. Интерфейс `javax.xml.stream.StreamFilter` имеет фильтрующий метод `boolean accept(XMLStreamReader reader)`, возвращающий `true`, если фильтр принимает событие.

- `public abstract XMLEventReader createFilteredReader(XMLEventReader reader, EventFilter filter)`

  Добавляет к объекту `XMLEventReader` фильтр. Интерфейс `javax.xml.stream.EventFilter` имеет фильтрующий метод `boolean accept(XMLEvent event)`, возвращающий `true`, если фильтр принимает событие.

- `public abstract XMLResolver getXMLResolver()`

  Возвращает объект `XMLResolver`, отвечающий за поиск ресурсов.

- `public abstract void setXMLResolver(XMLResolver resolver)`

  Устанавливает объект `XMLResolver` для фабрики.

- `public abstract XMLReporter getXMLReporter()`

  `public abstract void setXMLReporter(XMLReporter reporter)`

  Первый метод возвращает, а второй метод устанавливает объект `XMLReporter`, отвечающий за сообщения о некритичных ошибках.

- `public abstract void setProperty(String name, Object value)`

  `public abstract Object getProperty(String name)`

  Первый метод устанавливает, второй метод возвращает свойства StAX-реализации.

- `public abstract boolean isPropertySupported(String name)`

  Проверяет свойства, поддерживаемые фабрикой.

- `public abstract void setEventAllocator(XMLEventAllocator allocator)`

  `public abstract XMLEventAllocator getEventAllocator()`

  Первый метод устанавливает, второй метод возвращает объект `XMLEventAllocator` для фабрики.

Из рассмотрения набора методов фабрики `XMLInputFactory` видно, что метод `createFilteredReader()` позволяет осуществлять фильтрацию входящего потока событий.

Более сложное управление входящим потоком позволяют выполнять классы `EventReaderDelegate` и `StreamReaderDelegate` пакета `javax.xml.stream.util`, предоставляя свои методы для переопределения.

Класс `EventReaderDelegate` реализует интерфейс `XMLEventReader` и имеет следующие методы:

- `public void setParent(XMLEventReader reader)`

  `public XMLEventReader getParent()`

  Первый метод устанавливает, второй метод возвращает исходный объект `XMLEventReader`, который нагружается фильтрацией.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- Методы

  `public XMLEvent nextEvent()`

  `public Object next()`

  `public boolean hasNext()`

  `public XMLEvent peek()`

  `public void close()`

  `public String getElementText()`

  `public XMLEvent nextTag()`

  `public Object getProperty(String name)`

  `public void remove()`

  интерфейса `XMLEventReader`, перегружаемые для управления входящим потоком.

Класс `StreamReaderDelegate` реализует интерфейс `XMLStreamReader` и имеет следующие методы.

- `public void setParent(XMLStreamReader reader)`

  `public XMLStreamReader getParent()`

  Первый метод устанавливает, второй метод возвращает исходный объект `XMLStreamReader`, нагружаемый фильтрацией.

- Методы

  `public int next()`

  `public int nextTag()`

  `public String getElementText()`

  `public void require(int type, String namespaceURI, String localName)`

  `public boolean hasNext(), public void close()`

  `public String getNamespaceURI(String prefix)`

  `public NamespaceContext getNamespaceContext()`

  `public boolean isStartElement()`

  `public boolean isEndElement()`

  `public boolean isCharacters()`

  `public boolean isWhiteSpace()`

  `public String getAttributeValue(String namespaceUri, String localName)`

  `public int getAttributeCount()`

  `public QName getAttributeName(int index)`

  `public String getAttributePrefix(int index)`

  `public String getAttributeNamespace(int index)`

  `public String getAttributeLocalName(int index)`

  `public String getAttributeType(int index)`

  `public String getAttributeValue(int index)`

  `public boolean isAttributeSpecified(int index)`

  `public int getNamespaceCount()`

  `public String getNamespacePrefix(int index)`

```
public String getNamespaceURI(int index)
public int getEventType()
public String getText()
public int getTextCharacters(int sourceStart, char[] target,
  int targetStart, int length)
public char[] getTextCharacters()
public int getTextStart()
public int getTextLength()
public String getEncoding()
public boolean hasText()
public Location getLocation()
public QName getName()
public String getLocalName()
public boolean hasName()
public String getNamespaceURI()
public String getPrefix()
public String getVersion()
public boolean isStandalone()
public boolean standaloneSet()
public String getCharacterEncodingScheme()
public String getPITarget()
public String getPIData()
public Object getProperty(String name)
```

интерфейса `XMLStreamReader`, перегружаемые для управления входящим потоком.

Класс-фабрика `XMLOutputFactory`, предназначенный для создания объектов `XMLEventWriters` и `XMLStreamWriters`, имеет свойство `public static final String IS_REPAIRING_NAMESPACES`, устанавливающее при записи упорядочение и исправление деклараций пространств имен, а также перечисленные далее методы.

- `public static XMLOutputFactory newInstance()`

  `public static XMLInputFactory newInstance(String factoryId, ClassLoader classLoader)`

  Создают новый экземпляр фабрики.

- `public abstract XMLStreamWriter createXMLStreamWriter(java.io.Writer stream)`

  `public abstract XMLStreamWriter createXMLStreamWriter (java.io.OutputStream stream)`

  `public abstract XMLStreamWriter createXMLStreamWriter (java.io.OutputStream stream, String encoding)`

  `public abstract XMLStreamWriter createXMLStreamWriter (javax.xml.transform.Result result)`

  Создают исходящий поток `XMLStreamWriter` из различных источников.

- `public abstract XMLEventWriter createXMLEventWriter(Result result)`

  `public abstract XMLEventWriter createXMLEventWriter(OutputStream stream)`

  `public abstract XMLEventWriter createXMLEventWriter(OutputStream stream, String encoding)`

  `public abstract XMLEventWriter createXMLEventWriter(Writer stream)`

  Создают исходящий поток `XMLEventWriter` из различных источников.

- `public abstract void setProperty(String name, Object value)`

  `public abstract Object getProperty(String name)`

  Первый метод устанавливает, второй метод возвращает свойства StAX-реализации.

- `public abstract boolean isPropertySupported(String name)`

  Проверяет свойства, поддерживаемые фабрикой.

Интерфейс `XMLEvent` пакета `javax.xml.stream.events` представляет объекты событий, связывающие XML-документ с Java-приложением. В отличие от потока `XMLStreamReader`, поток `XMLEventReader` создает объекты `XMLEvent`, которые могут быть кэшированы в результате разбора XML-документа.

Интерфейс `XMLEvent` имеет следующие методы:

- `int getEventType()`

  Возвращает код события.

- `Location getLocation()`

  Возвращает объект `Location`, содержащий информацию о расположении события.

- `boolean isStartElement()`

  Возвращает `true`, если событие представляет начальный тег элемента.

- `boolean isAttribute()`

  Возвращает `true`, если событие представляет атрибут элемента.

- `boolean isNamespace()`

  Возвращает `true`, если событие представляет декларацию пространства имен.

- `boolean isEndElement()`

  Возвращает `true`, если событие представляет закрывающий тег элемента.

- `boolean isEntityReference()`

  Возвращает `true`, если событие представляет ссылку на данные.

- `boolean isProcessingInstruction()`

  Возвращает `true`, если событие представляет инструкцию по обработке.

- `boolean isCharacters()`

  Возвращает `true`, если событие представляет символьные данные.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `boolean isStartDocument()`

  Возвращает `true`, если событие представляет начало документа.

- `boolean isEndDocument()`

  Возвращает `true`, если событие представляет конец документа.

- `StartElement asStartElement()`

  Возвращает открывающий тег элемента или исключение.

- `EndElement asEndElement()`

  Возвращает закрывающий тег элемента или исключение.

- `Characters asCharacters()`

  Возвращает символьные данные или исключение.

- `QName getSchemaType()`

  Возвращает тип события.

- `void writeAsEncodedUnicode(Writer writer)`

  Преобразует объект `XMLEvent` в объект `java.io.Writer`.

Интерфейс `XMLEvent` является базовым для набора интерфейсов, отображенных в табл. 1.

Таблица 1. *Интерфейсы пакета* `javax.xml.stream.events`*, расширяющие* `XMLEvent`

Интерфейс	Описание
`Attribute`	Представляет атрибут элемента, дополнительно имеет следующие методы: • `QName getName()` — возвращает QName-имя атрибута; • `String getValue()` — возвращает значение атрибута; • `String getDTDType()` — возвращает тип атрибута, по умолчанию — `CDATA`; • `boolean isSpecified()` — возвращает `true`, если атрибут определен в начальном теге элемента, а не по умолчанию в схеме документа
`Characters`	Представляет символьные данные и дополнительно имеет следующие методы: • `String getData()` — возвращает символьные данные; • `boolean isWhiteSpace()` — возвращает `true`, если символьные данные — пробелы; • `boolean isCData()` — возвращает `true`, если данные — это секция `CDATA`; • `boolean isIgnorableWhiteSpace()` — возвращает `true`, если событие — пробел
`Comment`	Представляет комментарии и имеет дополнительно метод `String getText()` — возвращает текст комментария

***Таблица 1** (продолжение)*

Интерфейс	Описание
`DTD`	Представляет DTD-описание документа и имеет дополнительно следующие методы: • `String getDocumentTypeDeclaration()` — возвращает DTD-схему документа одной строкой; • `Object getProcessedDTD()` — возвращает структурированную DTD-схему документа; • `List getNotations()` — возвращает список DTD-объявлений `<!NOTATION>`; • `List getEntities()` — возвращает список DTD-объявлений `<!ENTITY>`
`EndDocument`	Представляет окончание документа
`EndElement`	Представляет закрывающий тег элемента и имеет дополнительно следующие методы: • `QName getName()` — возвращает QName-имя элемента; • `Iterator getNamespaces()` — возвращает итератор пространств имен элемента
`EntityDeclaration`	Представляет объявления неанализируемых сущностей и имеет дополнительно следующие методы: • `String getPublicId()` — возвращает публичный идентификатор данных; • `String getSystemId()` — возвращает системный идентификатор данных; • `String getName()` — возвращает имя сущности; • `String getNotationName()` — возвращает имя связанной с сущностью нотации; • `String getReplacementText()` — возвращает заменяющий сущность текст; • `String getBaseURI()` — возвращает базовый URI-адрес сущности
`EntityReference`	Представляет ссылки на сущности и имеет дополнительно следующие методы: • `EntityDeclaration getDeclaration()` — возвращает объявление сущности; • `String getName()` — возвращает имя сущности
`Namespace`	Представляет декларацию пространства имен и имеет дополнительно следующие методы: • `String getPrefix()` — возвращает префикс пространства имен; • `String getNamespaceURI()` — возвращает URI-идентификатор пространства имен; • `boolean isDefaultNamespaceDeclaration()` — возвращает `true`, если пространство имен является пространством имен по умолчанию

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

***Таблица 1** (окончание)*

Интерфейс	Описание
`NotationDeclaration`	Представляет объявление формата данных `<!NOTATION>` и имеет дополнительно следующие методы: • `String getName()` — возвращает имя нотации; • `String getPublicId()` — возвращает публичный идентификатор нотации; • `String getSystemId()` — возвращает системный идентификатор нотации
`ProcessingInstruction`	Представляет инструкции по обработке данных и имеет дополнительно следующие методы: • `String getTarget()` — возвращает идентификатор приложения для инструкции (сразу после `<?`); • `String getData()` — возвращает саму инструкцию
`StartDocument`	Представляет начало документа и имеет дополнительно следующие методы: • `String getSystemId()` — возвращает системный идентификатор документа; • `String getCharacterEncodingScheme()` — возвращает кодировку документа; • `boolean encodingSet()` — возвращает `true`, если есть декларация кодировки; • `boolean standaloneSet()` — возвращает `true`, если установлено значение атрибута `standalone`; • `String getVersion()` — возвращает XML-версию
`StartElement`	Представляет начальный тег элемента и имеет дополнительно следующие методы: • `QName getName()` — возвращает QName-имя элемента; • `Iterator getAttributes()` — возвращает итератор атрибутов элемента; • `Iterator getNamespaces()` — возвращает итератор деклараций пространств имен элемента; • `Attribute getAttributeByName(QName name)` — возвращает объект `Attribute`, представляющий атрибут по его имени; • `NamespaceContext getNamespaceContext()` — возвращает контекст пространств имен элемента; • `String getNamespaceURI(String prefix)` — возвращает URI пространства имен по его префиксу

За создание объектов `XMLEvents` отвечает класс-фабрика `XMLEventFactory` пакета `javax.xml.stream`, который имеет следующие методы:

- `public static XMLEventFactory newInstance()`

  `public static XMLEventFactory newInstance(String` *`factoryId`*`, ClassLoader` *`classLoader`*`)`

  Создают новый экземпляр фабрики.

- `public abstract void setLocation(Location location)`

  Устанавливает местонахождение создаваемого события.

- `public abstract Attribute createAttribute(String prefix, String namespaceURI, String localName, String value)`

  `public abstract Attribute createAttribute(String localName, String value)`

  `public abstract Attribute createAttribute(QName name, String value)`

  Создают новый атрибут.

- `public abstract Namespace createNamespace(String namespaceURI)`

  Создает декларацию пространства имен по умолчанию.

- `public abstract Namespace createNamespace(String prefix, String namespaceUri)`

  Создает декларацию пространства имен.

- `public abstract StartElement createStartElement(QName name, Iterator attributes, Iterator namespaces)`

  `public abstract StartElement createStartElement(String prefix, String namespaceUri, String localName)`

  `public abstract StartElement createStartElement(String prefix, String namespaceUri, String localName, Iterator attributes, Iterator namespaces)`

  `public abstract StartElement createStartElement(String prefix, String namespaceUri, String localName, Iterator attributes, Iterator namespaces, NamespaceContext context)`

  Создают открывающий тег элемента.

- `public abstract EndElement createEndElement(QName name, Iterator namespaces)`

  `public abstract EndElement createEndElement(String prefix, String namespaceUri, String localName)`

  `public abstract EndElement createEndElement(String prefix, String namespaceUri, String localName, Iterator namespaces)`

  Создают закрывающий тег элемента.

- `public abstract Characters createCharacters(String content)`

  Создает событие, представляющее символьные данные.

- `public abstract Characters createCData(String content)`

  Создает секцию `CDATA`.

- `public abstract Characters createSpace(String content)`

  Создает пробел.

- `public abstract Characters createIgnorableSpace(String content)`

  Создает пробелы, используемые для структурирования XML-документа.

- `public abstract StartDocument createStartDocument()`

  `public abstract StartDocument createStartDocument(String encoding, String version, boolean standalone)`

  `public abstract StartDocument createStartDocument(String encoding, String version)`

  `public abstract StartDocument createStartDocument(String encoding)`

  Создают начало документа.

- `public abstract EndDocument createEndDocument()`

  Создает конец документа.

- `public abstract EntityReference createEntityReference(String name, EntityDeclaration declaration)`

  Создает ссылку на сущность.

- `public abstract Comment createComment(String text)`

  Создает комментарии.

- `public abstract ProcessingInstruction createProcessingInstruction (String target, String data)`

  Создает инструкцию по обработке.

- `public abstract DTD createDTD(String dtd)`

  Создает DTD-описание документа.

DOM

Пакеты `org.w3c.dom`, `org.w3c.dom.bootstrap`, `org.w3c.dom.events` и `org.w3c.dom.ls` обеспечивают поддержку модели Document Object Model (DOM) для платформы Java SE, позволяя управлять содержимым и структурой XML-документов.

> ***Примечание***
>
> Модель DOM описывается набором классификаций, разделенным на три уровня. Первый уровень обеспечивает базовые возможности модели, такие как получение и управление деревом узлов документа. Второй уровень обеспечивает поддержку пространства имен и событий. Третий уровень включает в себя расширения DOM. Платформа Java SE реализует все три уровня DOM-модели.

Так как в модели DOM документ представляется в виде дерева узлов, основным интерфейсом DOM API является интерфейс `Node` пакета `org.w3c.dom`, представляющий узел дерева документа.

Интерфейс `Node` имеет перечисленные далее поля:

- `static final short ELEMENT_NODE` — узел представляет элемент;
- `static final short ATTRIBUTE_NODE` — узел представляет атрибут;
- `static final short TEXT_NODE` — узел представляет текстовый объект;
- `static final short CDATA_SECTION_NODE` — узел представляет секцию `CDATA`;

- `static final short ENTITY_REFERENCE_NODE` — узел представляет ссылку на сущность;
- `static final short ENTITY_NODE` — узел представляет сущность;
- `static final short PROCESSING_INSTRUCTION_NODE` — узел представляет инструкцию по обработке;
- `static final short COMMENT_NODE` — узел представляет комментарии;
- `static final short DOCUMENT_NODE` — узел представляет корневой элемент документа;
- `static final short DOCUMENT_TYPE_NODE` — узел представляет DTD-описание;
- `static final short DOCUMENT_FRAGMENT_NODE` — узел представляет фрагмент документа;
- `static final short NOTATION_NODE` — узел представляет нотацию.

Следующие поля позволяют сравнивать расположение определенного узла с текущим узлом в иерархии DOM:

- `static final short DOCUMENT_POSITION_DISCONNECTED` — два узла не имеют общего узла-контейнера;
- `static final short DOCUMENT_POSITION_PRECEDING` — определенный узел предшествует текущему узлу;
- `static final short DOCUMENT_POSITION_FOLLOWING` — определенный узел следует за текущим узлом;
- `static final short DOCUMENT_POSITION_CONTAINS` — определенный узел содержит текущий узел;
- `static final short DOCUMENT_POSITION_CONTAINED_BY` — текущий узел содержит определенный узел;
- `static final short DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC` — расположение зависит от реализации.

Интерфейс `Node` имеет перечисленные далее методы.

- `String getNodeName()`

  Возвращает имя узла.

- `String getNodeValue()`

  `void setNodeValue(String nodeValue)`

  Первый метод возвращает, второй метод устанавливает значение узла.

- `short getNodeType()`

  Возвращает код, соответствующий типу узла.

- `Node getParentNode()`

  Возвращает узел, родительский для данного узла.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `NodeList getChildNodes()`

  Возвращает список дочерних узлов данного узла. Интерфейс `org.w3c.dom.NodeList` представляет коллекцию DOM-узлов и имеет методы:

  - `Node item(int index)` — возвращает узел по его индексу в коллекции;
  - `int getLength()` — возвращает количество узлов в коллекции.

- `Node getFirstChild()`

  `Node getLastChild()`

  Первый метод возвращает первый дочерний узел данного узла, второй метод — последний дочерний узел данного узла.

- `Node getPreviousSibling()`

  `Node getNextSibling()`

  Первый метод возвращает предшествующий узел данного узла, второй метод — последующий узел для данного узла.

- `NamedNodeMap getAttributes()`

  Возвращает объект `org.w3c.dom.NamedNodeMap`, представляющий коллекцию атрибутов данного узла. Интерфейс `NamedNodeMap` представляет коллекцию узлов, характеризующихся именами, и имеет следующие методы:

  - `Node getNamedItem(String name)` — возвращает узел по его имени;
  - `Node setNamedItem(Node arg)` — добавляет узел в коллекцию, используя его имя;
  - `Node removeNamedItem(String name)` — удаляет узел из коллекции, используя его имя;
  - `Node item(int index)` — возвращает узел по его индексу в коллекции;
  - `int getLength()` — возвращает количество узлов в коллекции;
  - `Node getNamedItemNS(String namespaceURI, String localName)` — возвращает узел по его локальному имени и пространству имен;
  - `Node setNamedItemNS(Node arg)` — добавляет узел в коллекцию, используя его локальное имя и пространство имен;
  - `Node removeNamedItemNS(String namespaceURI, String localName)` — удаляет узел из коллекции, используя его локальное имя и пространство имен.

- `Document getOwnerDocument()`

  Возвращает объект `org.w3c.dom.Document`, связанный с данным узлом. Интерфейс `Document` также является ключевым в DOM API, т. к. представляет полный HTML- или XML-документ. Интерфейс `Document` расширяет интерфейс `Node` и дополнительно предоставляет следующие методы:

  - `DocumentType getDoctype()` — возвращает DTD-описание документа;
  - `DOMImplementation getImplementation()` — возвращает объект `org.w3c.dom.DOMImplementation`, обеспечивающий взаимодействие с конкретной DOM-реализацией.

Интерфейс `DOMImplementation` предоставляет методы:

- `boolean hasFeature(String feature, String version)` — тестирует свойства и версию реализации DOM;
- `DocumentType createDocumentType(String qualifiedName, String publicId, String systemId)` — создает пустой DTD-узел;
- `Document createDocument(String namespaceURI, String qualifiedName, DocumentType doctype)` — создает новый узел `Document`;
- `Object getFeature(String feature, String version)` — возвращает объект, реализующий определенное свойство.

Объект `DOMImplementation` создается с помощью класса-фабрики `org.w3c.dom.bootstrap.DOMImplementationRegistry`.

Класс `DOMImplementationRegistry` имеет методы:

- `public static DOMImplementationRegistry newInstance()` — создает объект фабрики;
- `public DOMImplementation getDOMImplementation(String features)` — возвращает объект `DOMImplementation`;
- `public DOMImplementationList getDOMImplementationList(String features)` — возвращает список DOM-реализаций, поддерживающих определенное свойство;
- `public void addSource(DOMImplementationSource s)` — регистрирует определенную DOM-реализацию;

- `Element getDocumentElement()` — возвращает корневой элемент документа;
- `Element createElement(String tagName)`
  `Element createElementNS(String namespaceURI, String qualifiedName)`

  Создают элемент документа. Интерфейс `org.w3c.dom.Element` расширяет интерфейс `Node` и представляет элемент HTML- или XML-документа, дополнительно обеспечивая методы:

  - `String getTagName()` — возвращает имя элемента;
  - `String getAttribute(String name)` — возвращает атрибут элемента по его имени;
  - `void setAttribute(String name, String value)` или `void setAttributeNS(String namespaceURI, String qualifiedName, String value)` — добавляют новый атрибут;
  - `void removeAttribute(String name)` или `void removeAttributeNS(String namespaceURI, String localName)` — удаляют атрибут;
  - `Attr getAttributeNode(String name)` или `Attr getAttributeNodeNS(String namespaceURI, String localName)` — возвращает узел атрибута;
  - `Attr setAttributeNode(Attr newAttr)` или `Attr setAttributeNodeNS(Attr newAttr)` — добавляют узел атрибута;

  - `Attr removeAttributeNode(Attr oldAttr)` — удаляет узел атрибута;
  - `NodeList getElementsByTagName(String name)` или `NodeList getElementsByTagNameNS(String namespaceURI, String localName)` — возвращают список дочерних элементов;
  - `String getAttributeNS(String namespaceURI, String localName)` — возвращает значение атрибута;
  - `boolean hasAttribute(String name)` или `boolean hasAttributeNS(String namespaceURI, String localName)` — тестирует наличие атрибута;
  - `TypeInfo getSchemaTypeInfo()` — возвращает описание элемента;
  - `void setIdAttribute(String name, boolean isId)`, `void setIdAttributeNS(String namespaceURI, String localName, boolean isId)` или `void setIdAttributeNode(Attr idAttr, boolean isId)` — устанавливают атрибут-идентификатор элемента;
- `DocumentFragment createDocumentFragment()` — создает фрагмент документа;
- `Text createTextNode(String data)` — создает текстовый узел;
- `Comment createComment(String data)` — создает узел комментариев;
- `CDATASection createCDATASection(String data)` — создает узел секции `CDATA`;
- `ProcessingInstruction createProcessingInstruction(String target, String data)` — создает узел инструкции по обработке;
- `Attr createAttribute(String name)` или `Attr createAttributeNS(String namespaceURI, String qualifiedName)` — создают узел атрибута;
- `EntityReference createEntityReference(String name)` — создает узел ссылки на сущность;
- `NodeList getElementsByTagName(String tagname)` или `NodeList getElementsByTagNameNS(String namespaceURI, String localName)` — возвращают список элементов документа;
- `Node importNode(Node importedNode, boolean deep)` — осуществляет импорт узла из другого документа с поддеревом или без него;
- `Element getElementById(String elementId)` — возвращает элемент документа по его идентификатору;
- `String getInputEncoding()` — возвращает кодировку документа, используемую при разборе документа;
- `String getXmlEncoding()` — возвращает значение атрибута `encoding` элемента `<?xml ... ?>`;
- `boolean getXmlStandalone()` — определяет, является ли документ автономным;
- `void setXmlStandalone(boolean xmlStandalone)` — устанавливает значение атрибута `standalone`;

- `String getXmlVersion()` и `void setXmlVersion(String xmlVersion)` — возвращает и устанавливает значение атрибута `version`;
- `boolean getStrictErrorChecking()` и `void setStrictErrorChecking(boolean strictErrorChecking)` — первый метод определяет, второй метод устанавливает строгую проверку ошибок;
- `String getDocumentURI()` и `void setDocumentURI(String documentURI)` — первый метод возвращает, второй метод устанавливает URI-адрес документа;
- `Node adoptNode(Node source)` — адаптирует узел при его импорте в документ;
- `DOMConfiguration getDomConfig()` — возвращает объект `org.w3c.dom.DOMConfiguration`, представляющий конфигурацию документа с параметрами. Интерфейс `DOMConfiguration` предоставляет методы `void setParameter(String name, Object value)` и `Object getParameter(String name)` для установки и возврата параметров конфигурации, таких как `canonical-form`, `cdata-sections`, `check-character-normalization`, `comments`, `datatype-normalization`, `element-content-whitespace`, `entities`, `error-handler`, `infoset`, `namespaces`, `namespace-declarations`, `normalize-characters`, `schema-location`, `schema-type`, `split-cdata-sections`, `validate`, `validate-if-schema`, `well-formed`, а также методы:
  - `boolean canSetParameter(String name, Object value)` — тестирует поддержку параметра;
  - `DOMStringList getParameterNames()` — список параметров конфигурации;
- `void normalizeDocument()` — осуществляет нормализацию документа;
- `Node renameNode(Node n, String namespaceURI, String qualifiedName)` — переименовывает узел элемента или атрибута.

❒ `Node insertBefore(Node newChild, Node refChild)`
`Node replaceChild(Node newChild, Node oldChild)`
`Node removeChild(Node oldChild)`
`Node appendChild(Node newChild)`
`Node cloneNode(boolean deep)`

Вставляет, заменяет, удаляет и дублирует узлы.

❒ `boolean hasChildNodes()`

Тестирует, имеет ли узел дочерние узлы.

❒ `void normalize()`

Осуществляет нормализацию узла.

❒ `boolean isSupported(String feature, String version)`

Тестирует DOM-реализацию на поддержку свойств.

❒ `String getNamespaceURI()`
`String getLocalName()`
`String getPrefix()`
`void setPrefix(String prefix)`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Первый метод возвращает пространство имен, второй метод — локальное имя, третий метод возвращает префикс узла, четвертый метод устанавливает префикс узла.

- `boolean hasAttributes()`

  Тестирует узел на наличие атрибутов.

- `String getBaseURI()`

  Возвращает базовый URI узла.

- `short compareDocumentPosition(Node other)`

  Сравнивает взаимное расположение двух узлов.

- `String getTextContent()`
  `void setTextContent(String textContent)`

  Первый метод возвращает, второй метод устанавливает текстовое содержимое узла и его дочерних узлов.

- `boolean isSameNode(Node other)`
  `boolean isEqualNode(Node arg)`

  Перый метод тестирует на схожесть двух узлов, второй метод — на полное соответствие двух узлов.

- `String lookupPrefix(String namespaceURI)`
  `String lookupNamespaceURI(String prefix)`

  Перый метод возвращает префикс пространства имен, второй метод — URI пространства имен.

- `boolean isDefaultNamespace(String namespaceURI)`

  Тестирует пространство имен на значение по умолчанию.

- `Object getFeature(String feature, String version)`

  Возвращает объект, реализующий определенное свойство.

- `Object setUserData(String key, Object data, UserDataHandler handler)`
  `Object getUserData(String key)`

  Устанавливает ключ для объекта и возвращает объект по ключу.

Интерфейс `Node` является базовым для таких интерфейсов, как `Document`, `DocumentFragment`, `Attr`, `CharacterData`, `DocumentType`, `Element`, `Entity`, `EntityReference`, `Notation` и `ProcessingInstruction`.

Интерфейс `Attr`, представляющий атрибут элемента, дополнительно имеет перечисленные далее методы.

- `String getName()`

  Возвращает имя атрибута.

- `boolean getSpecified()`

  Возвращает `true`, если значение атрибута явно определено в документе.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `String getValue()`

  `void setValue(String value)`

  Первый метод возвращает, второй метод устанавливает значение атрибута.

- `Element getOwnerElement()`

  Возвращает объект `Element`, представляющий элемент атрибута.

- `TypeInfo getSchemaTypeInfo()`

  Возвращает описание атрибута в схеме документа.

- `boolean isId()`

  Возвращает `true`, если атрибут является идентификатором.

Интерфейс `CharacterData`, представляющий символьные данные, дополнительно имеет перечисленные далее методы.

- `String getData()`

  `void setData(String data)`

  Первый метод возвращает, второй метод устанавливает символьные данные объекта `CharacterData`.

- `int getLength()`

  Возвращает 16-битовую длину символьных данных.

- `String substringData(int offset, int count)`

  Возвращает часть символьных данных.

- `void appendData(String arg)`

  `void insertData(int offset, String arg)`

  `void deleteData(int offset, int count)`

  `void replaceData(int offset, int count, String arg)`

  Добавляет, вставляет, удаляет и заменяет символьные данные.

Интерфейс `Comment` расширяет интерфейс `CharacterData`, представляя комментарии документа.

Интерфейс `DocumentType` представляет DTD-описание документа и имеет дополнительно к методам интерфейса `Node` перечисленные далее методы.

- `String getName()`

  Возвращает имя DTD-описания.

- `NamedNodeMap getEntities()`

  Возвращает объект `NamedNodeMap`, содержащий список именованных сущностей, продекларированных в DTD-описании документа.

- `NamedNodeMap getNotations()`

  Возвращает объект `NamedNodeMap`, содержащий список именованных нотаций, продекларированных в DTD-описании документа.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `String getPublicId()`

  `String getSystemId()`

  Первый метод возвращает публичный идентификатор DTD-описания, второй метод — системный идентификатор DTD-описания.

- `String getInternalSubset()`

  Возвращает DTD-описание документа одной строкой.

Интерфейс `Entity` представляет как анализируемую, так и не анализируемую сущность и дополнительно к методам интерфейса `Node` имеет перечисленные далее методы.

- `String getPublicId()`

  `String getSystemId()`

  Первый метод возвращает публичный идентификатор сущности, второй метод — и системный идентификатор сущности.

- `String getNotationName()`

  Для неанализируемой сущности возвращает имя нотации.

- `String getInputEncoding()`

  Для внешней сущности возвращает кодировку разбора.

- `String getXmlEncoding()`

  Для внешней сущности возвращает продекларированную кодировку.

- `String getXmlVersion()`

  Для внешней сущности возвращает XML-версию.

Интерфейс `EntityReference` представляет ссылку на сущность и не имеет дополнительных методов к методам интерфейса `Node`.

Интерфейс `Notation` представляет нотации DTD-описания и дополнительно к методам интерфейса `Node` имеет методы `String getPublicId()` и `String getSystemId()`.

Интерфейс `ProcessingInstruction` представляет инструкцию по обработке и расширяет интерфейс `Node` с дополнительными методами, перечисленными далее.

- `String getTarget()`

  Возвращает первую лексему открывающего тега инструкции по обработке.

- `String getData()`

  `void setData(String data)`

  Первый метод возвращает, второй метод устанавливает саму инструкцию по обработке.

Интерфейс `Text` расширяет интерфейс `CharacterData`, представляя текстовое содержимое элемента или атрибута с дополнительными методами.

- `Text splitText(int offset)`

  Из одного узла создает два узла, возвращая новый узел.

- `boolean isElementContentWhitespace()`

  Возвращает `true`, если узел содержит игнорируемые пробелы.

- `String getWholeText()`

  `Text replaceWholeText(String content)`

  Первый метод возвращает, второй метод заменяет текстовое содержимое.

Для организации таких структур, как хэш-таблица, DOM-узел может хранить ключи объектов, создаваемых с помощью метода `Node.setUserData()`. При модификации такого DOM-узла необходимо управление общей структурой данных, включающей связанные с ключами объекты. Интерфейс `org.w3c.dom.UserDataHandler` как раз и предназначен для вызова при клонировании, импорте или переименовании DOM-узла. При клонировании, импорте, удалении, переименовании или адаптации узла вызывается метод `void handle(short operation, String key, Object data, Node src, Node dst)` интерфейса `UserDataHandler`.

Интерфейс `org.w3c.dom.TypeInfo` представляет описание типа данных элемента или атрибута, данное в схеме документа. Интерфейс `TypeInfo` позволяет извлекать имя и пространство имен типа данных с помощью методов `String getTypeName()` и `String getTypeNamespace()`. Метод `boolean isDerivedFrom(String typeNamespaceArg, String typeNameArg, int derivationMethod)` интерфейса `TypeInfo` возвращает `true`, если текущий тип данных происходит из указанного в параметрах метода типа данных. Третий параметр метода — это способ присхождения типа данных (расширение — 2, список — 8, ограничение — 1, объединение — 4).

Пакет `org.w3c.dom.events` позволяет регистрировать DOM-узлы в качестве слушателей различных событий, таких как событий интерфейса пользователя — щелчки кнопки мыши или нажатия клавиш, или событий, связанных с изменениями структуры документа.

Регистрация DOM-узлов как слушателей событий производится с помощью метода `void addEventListener(String type, EventListener listener, boolean useCapture)` интерфейса `org.w3c.dom.events.EventTarget`, который реализуется всеми DOM-узлами, поддерживающими модель DOM Event Model. Первый параметр `type` метода определяет тип события, для которого производится регистрация; второй параметр `listener` определяет объект `EventListener`, содержащий метод, вызываемый при возникновении определенного события; третий параметр `useCapture` указывает способ доставки события внутри дерева документа. Если параметр `useCapture` установлен в значение `true`, тогда событие доставляется первому узлу, зарегистрированному для получения данного события. Затем осуществляется поиск его дочерних элементов, также зарегистрированных для получения данного события. Таким образом, событие доставляется сверху вниз по дереву — capture phase. Если же параметр `useCapture` установлен в значение `false`, тогда событие сначала доставляется последнему зарегистрированному дочернему элементу, а затем поднимается вверх по дереву и доставляется всем родительским элементам, зарегистрированным для получения данного события — bubble phase.

Интерфейс `org.w3c.dom.events.EventListener` реализуется классом пользовательского приложения, которое обрабатывает документ, для переопределения метода

`void handleEvent(Event evt)` интерфейса, вызываемого для зарегистрированного DOM-узла при получении определенного события.

Интерфейс `EventTarget` также имеет методы `void removeEventListener(String type, EventListener listener, boolean useCapture)` — удаляет регистрацию DOM-узла, и `boolean dispatchEvent(Event evt)` — симулирует создание события, возвращает `true`, если действия по умолчанию, связанные с данным событием, аннулированы.

Интерфейс `org.w3c.dom.events.Event` является базовым интерфейсом для интерфейсов, представляющих события модели DOM Event Model. Интерфейс `Event` имеет перечисленные далее методы.

- `String getType()`

  Возвращает имя события.

- `EventTarget getTarget()`

  Возвращает узел, который зарегистрирован для получения данного события.

- `EventTarget getCurrentTarget()`

  Возвращает узел, который в данный момент обрабатывает получение события.

- `short getEventPhase()`

  Указывает, какая фаза доставки события в данный момент выполняется — capture phase, target phase или bubble phase.

- `boolean getBubbles()`

  Указывает, может ли данное событие доставляться реверсивным способом.

- `boolean getCancelable()`

  Возвращает `true`, если действия по умолчанию, связанные с данным событием, могут быть аннулированы. Например, действие, связанное с активацией гиперссылки, может быть аннулировано вызовом метода `void preventDefault()`.

- `void preventDefault()`

  Аннулирует действия по умолчанию, связанные с данным событием.

- `long getTimeStamp()`

  Возвращает время, для которого событие создано.

- `void stopPropagation()`

  Останавливает дальнейшую передачу события.

- `void initEvent(String eventTypeArg, boolean canBubbleArg, boolean cancelableArg)`

  Производит инициализацию события.

Интерфейс `org.w3c.dom.events.UIEvent` расширяет интерфейс `Event` и представляет события, связанные с интерфейсом пользователя. Интерфейс `UIEvent` дополнительно имеет следующие методы:

- `org.w3c.dom.views.AbstractView getView()`

  Возвращает представление документа.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `int getDetail()`

  Указывает дополнительную информацию о событии, например количество щелчков кнопки мыши.

- `void initUIEvent(String typeArg, boolean canBubbleArg, boolean cancelableArg, org.w3c.dom.views.AbstractView viewArg, int detailArg)`

  Инициализирует событие.

Интерфейс `org.w3c.dom.events.MutationEvent` расширяет интерфейс `Event` и представляет события, связанные с изменением документа. Интерфейс `MutationEvent` дополнительно имеет следующие методы:

- `Node getRelatedNode()`

  Указывает измененный узел документа.

- `String getPrevValue()`

  Указывает предыдущее значение атрибута или символьных данных.

- `String getNewValue()`

  Возвращает новое значение атрибута или символьных данных.

- `String getAttrName()`

  Возвращает имя измененного атрибута.

- `short getAttrChange()`

  Указывает тип изменения атрибута: 1 — модификация, 2 — добавление, 3 — удаление.

- `void initMutationEvent(String typeArg, boolean canBubbleArg, boolean cancelableArg, Node relatedNodeArg, String prevValueArg, String newValueArg, String attrNameArg, short attrChangeArg)`

  Инициализирует событие.

Интерфейс `org.w3c.dom.events.DocumentEvent` реализуется объектом `Document` и позволяет создать событие определенного типа с помощью метода `Event createEvent(String eventType)`. После создания события его необходимо инициализировать, используя метод `initEvent`.

Пакет `org.w3c.dom.ls` обеспечивает поддержку спецификации DOM Level 3 Load and Save — набор интерфейсов для загрузки и сохранения объектов документа.

Интерфейс `org.w3c.dom.ls.LSParser` предназначен для разбора XML-документа и создания соответствующей DOM-структуры. Интерфейс `LSParser` предлагает следующие методы:

- `DOMConfiguration getDomConfig()`

  Возвращает конфигурацию, используемую при разборе документа.

- `LSParserFilter getFilter()`

  `void setFilter(LSParserFilter filter)`

  Первый метод возвращает, второй метод устанавливает объект `LSParserFilter`, используемый для управления процессом разбора документа. Интерфейс

`org.w3c.dom.ls.LSParserFilter` имеет следующие методы, вызываемые при разборе документа:

- `short startElement(Element elementArg)` — вызывается при разборе начального тега элемента, возвращает: 1 — элемент допускается для включения в DOM-дерево, 2 — элемент отвергается, 3 — элемент пропускается, 4 — обработка документа прерывается;
- `short acceptNode(Node nodeArg)` — вызывается после разбора каждого узла, также возвращает: 1 — `FILTER_ACCEPT`, 2 — `FILTER_REJECT`, 3 — `FILTER_SKIP`, 4 — `FILTER_INTERRUPT`;
- `int getWhatToShow()` — возвращает код узла, который доступен для метода `LSParserFilter.acceptNode()`.

❒ `boolean getAsync()`

Возвращает `true`, если процесс разбора документа является асинхронным.

❒ `boolean getBusy()`

Возвращает `true`, если парсер занят загрузкой документа.

❒ `Document parse(LSInput input)`

Осуществляет разбор документа из ресурса, представленного объектом `LSInput`. Интерфейс `org.w3c.dom.ls.LSInput` представляет входящий источник данных и имеет следующие методы:

- `Reader getCharacterStream()` и `void setCharacterStream(Reader characterStream)` — возвращает и устанавливает символьный поток данных ресурса;
- `InputStream getByteStream()` и `void setByteStream(InputStream byteStream)` — возвращает и устанавливает байтовый поток данных ресурса;
- `String getStringData()` и `void setStringData(String stringData)` — возвращает и устанавливает строковые данные ресурса;
- `String getSystemId()` и `void setSystemId(String systemId)` — возвращает и устанавливает системный идентификатор ресурса;
- `String getPublicId()` и `void setPublicId(String publicId)` — возвращает и устанавливает публичный идентификатор ресурса;
- `String getBaseURI()` и `void setBaseURI(String baseURI)` — возвращает и устанавливает базовый URI ресурса;
- `String getEncoding()` и `void setEncoding(String encoding)` — возвращает и устанавливает XML-кодировку документа;
- `boolean getCertifiedText()` и `void setCertifiedText(boolean certifiedText)` — возвращает `true` и устанавливает, что ресурс является Unicode-нормализованным текстом.

❒ `Document parseURI(String uri)`

Осуществляет разбор документа, находящегося по URI-адресу.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `Node parseWithContext(LSInput input, Node contextArg, short action)`

  Осуществляет разбор XML-фрагмента и вставляет его в существующее дерево.

- `void abort()`

  Прерывает загрузку документа парсером.

Интерфейс `org.w3c.dom.ls.LSSerializer` предназначен для сериализации DOM-дерева в XML-документ. Интерфейс `LSSerializer` предлагает следующие методы:

- `DOMConfiguration getDomConfig()`

  Возвращает конфигурацию, используемую при сериализации документа.

- `String getNewLine()`

  `void setNewLine(String newLine)`

  Первый метод возвращает, второй метод устанавливает символы конца строки документа.

- `LSSerializerFilter getFilter()`

  `void setFilter(LSSerializerFilter filter)`

  Первый метод возвращает, второй метод устанавливает фильтр сериализации документа. Интерфейс `org.w3c.dom.ls.LSSerializerFilter` имеет следующие методы:

  - `short acceptNode(Node nodeArg)` — вызывается при сериализации каждого узла, возвращает: 1 — `FILTER_ACCEPT`, 2 — `FILTER_REJECT`, 3 — `FILTER_SKIP`, 4 — `FILTER_INTERRUPT`;
  - `int getWhatToShow()` — возвращает код узла, который доступен для фильтрации.

- `boolean write(Node nodeArg, LSOutput destination)`

  Осуществляет сериализацию узла в объект `LSOutput`, возвращает `true` в случае успешной сериализации. Интерфейс `org.w3c.dom.ls.LSOutput` представляет исходящие данные и имеет следующие методы:

  - `Writer getCharacterStream()` и `void setCharacterStream(Writer characterStream)` — возвращает и устанавливает исходящий символьный поток;
  - `OutputStream getByteStream()` и `void setByteStream(OutputStream byteStream)` — возвращает и устанавливает исходящий байтовый поток;
  - `String getSystemId()` и `void setSystemId(String systemId)` — возвращает и устанавливает системный идентификатор исходящего ресурса;
  - `String getEncoding()` и `void setEncoding(String encoding)` — возвращает и устанавливает кодировку исходящего ресурса.

- `boolean writeToURI(Node nodeArg, String uri)`

  Осуществляет сериализацию узла в ресурс с указанным системным URI-идентификатором, возвращает `true` в случае успешной сериализации.

- `String writeToString(Node nodeArg)`

  Осуществляет сериализацию в строку.

Интерфейс `org.w3c.dom.ls.LSResourceResolver` реализуется классом пользовательского приложения для распознавания парсером ссылок на внешние ресурсы. Реализация интерфейса `LSResourceResolver` регистрируется как параметр "resource-resolver" объекта `DOMConfiguration` и при открытии внешнего ресурса парсер вызывает метод `LSInput resolveResource(String type, String namespaceURI, String publicId, String systemId, String baseURI)` интерфейса `LSResourceResolver`.

Пакет `org.w3c.dom.ls` также содержит интерфейс `LSLoadEvent`, расширяющий интерфейс `org.w3c.dom.events.Event` и представляющий события, связанные с окончанием загрузки документа для разбора. Интерфейс `LSLoadEvent` дополнительно имеет методы `Document getNewDocument()` — возвращает загруженный документ, и `LSInput getInput()` — возвращает объект источника данных для разбора.

Интерфейс `org.w3c.dom.ls.LSProgressEvent` расширяет интерфейс `org.w3c.dom.events.Event` и представляет события, связанные с процессом разбора документа. Интерфейс `LSProgressEvent` дополнительно предоставляет следующие методы:

- `LSInput getInput()`

  Возвращает объект источника данных разбора.

- `int getPosition()`

  Возвращает текущую позицию разбора источника данных.

- `int getTotalSize()`

  Возвращает размер разбираемого документа.

Новые объекты `LSInput`, `LSOutput`, `LSParser` и `LSSerializer` создаются с помощью класса-фабрики `org.w3c.dom.ls.DOMImplementationLS`, объект которого, как правило, получается методом `DOMImplementation.getFeature()`. Для создания объектов интерфейс `DOMImplementationLS` предоставляет методы:

- `LSParser createLSParser(short mode, String schemaType)`, где параметр `mode` принимает значения 1 — синхронный парсер и 2 — асинхронный парсер;
- `LSSerializer createLSSerializer();`
- `LSInput createLSInput();`
- `LSOutput createLSOutput()`.

SAX

Пакеты `org.xml.sax`, `org.xml.sax.ext` и `org.xml.sax.helpers` обеспечивают поддержку SAX-анализа — быстрого и эффективного механизма доступа к XML-документам.

Основным интерфейсом пакета `org.xml.sax` является интерфейс `XMLReader`, позволяющий считывать XML-документ, используя обратные вызовы методов.

Интерфейс `XMLReader` реализуется SAX-парсером и имеет следующие методы:

- `boolean getFeature(String name)`
  `void setFeature(String name, boolean value)`
  `Object getProperty(String name)`
  `void setProperty(String name, Object value)`

  Первая пара методов проверяет и устанавливает режимы SAX-парсера, вторая пара методов проверяет и устанавливает и свойства SAX-парсера.

- `EntityResolver getEntityResolver()`
  `void setEntityResolver(EntityResolver resolver)`

  Первый метод возвращает, второй метод регистрирует объект `EntityResolver`, предназначенный для распознавания сущностей. Интерфейс `org.xml.sax.EntityResolver` может реализовываться классом пользовательского приложения для управления процессом разрешения внешних сущностей с помощью вызываемого парсером метода `InputSource resolveEntity(String publicId, String systemId)`. Возвращаемый методом объект `org.xml.sax.InputSource` представляет входящий источник XML-сущности. Класс `InputSource` имеет следующие конструкторы и методы:

  - `public InputSource()`, `public InputSource(String systemId)`, `public InputSource(InputStream byteStream)`, `public InputSource(Reader characterStream)` — создает объект `InputSource` из различных ресурсов;
  - `public void setPublicId(String publicId)` и `public String getPublicId()`, `public void setSystemId(String systemId)` и `public String getSystemId()` — устанавливает и возвращает публичный и системный идентификаторы входящего источника данных;
  - `public void setByteStream(InputStream byteStream)` и `public InputStream getByteStream()`, `public void setCharacterStream(Reader characterStream)` и `public Reader getCharacterStream()` — регистрирует и возвращает байтовый и символьный потоки для входящего источника данных;
  - `public void setEncoding(String encoding)` и `public String getEncoding()` — устанавливает и возвращает XML-кодировку данных.

- `void setDTDHandler(DTDHandler handler)`
  `DTDHandler getDTDHandler()`

  Первый метод регистрирует, второй метод возвращает объект `org.xml.sax.DTDHandler`, получающий уведомления о DTD-событиях в процессе разбора документа. Интерфейс `DTDHandler` реализуется классом пользовательского приложения для переопределения методов `void notationDecl(String name, String publicId, String systemId)` и `void unparsedEntityDecl(String name, String publicId, String systemId, String notationName)`, вызываемых парсером при разборе DTD-описания.

- `void setContentHandler(ContentHandler handler)`
  `ContentHandler getContentHandler()`

Первый метод регистрирует, второй метод возвращает объект `org.xml.sax.ContentHandler`, получающий уведомления о событиях, связанных с содержимым XML-документа в процессе его разбора. Интерфейс `ContentHandler` реализуется классом пользовательского приложения для переопределения следующих методов:

- `void setDocumentLocator(Locator locator)` — получает объект `org.xml.sax.Locator`, представляющий связь SAX-события с документом. Интерфейс `Locator` имеет методы `String getPublicId()`, `String getSystemId()`, `int getLineNumber()` и `int getColumnNumber()`;
- `void startDocument()` и `void endDocument()` — вызываются при появлении начала и конца документа соответственно;
- `void startPrefixMapping(String prefix, String uri)` и `void endPrefixMapping (String prefix)` — первый метод вызывается в начале области действия префикса определенного пространства имен, второй метод — в конце;
- `void startElement(String uri, String localName, String qName, Attributes atts)` и `void endElement(String uri, String localName, String qName)` — первый метод вызывается при разборе начала элемента, второй метод — конца. Интерфейс `org.xml.sax.Attributes` представляет список атрибутов и имеет методы, обеспечивающие информацию об атрибутах в списке;
- `void characters(char[] ch, int start, int length)` — вызывается при разборе символьных данных;
- `void ignorableWhitespace(char[] ch, int start, int length)` — вызывается при разборе игнорируемых пробелов;
- `void processingInstruction(String target, String data)` — вызывается при разборе инструкции по обработке;
- `void skippedEntity(String name)` — вызывается при встрече парсером сущности, которую он пропускает, например, если парсер не разбирает ссылки на внешние сущности.

❒ `void setErrorHandler(ErrorHandler handler)`
`ErrorHandler getErrorHandler()`

Первый метод регистрирует объект `org.xml.sax.ErrorHandler`, обеспечивающий управление ошибками разбора, второй метод возвращает указанный объект. Для обработки ошибок интерфейс `ErrorHandler` предоставляет методы `void warning(SAXParseException exception)`, `void error(SAXParseException exception)` и `void fatalError(SAXParseException exception)`.

❒ `void parse(InputSource input)`
`void parse(String systemId)`

Запускает разбор парсером входящего источника данных.

Интерфейс `org.xml.sax.XMLFilter` расширяет интерфейс `XMLReader` и позволяет фильтровать поток событий, получаемый от парсера, с помощью методов, унасле-

дованных от интерфейса `XMLReader`, а также дополнительно имеет методы `void setParent(XMLReader parent)` и `XMLReader getParent()` для установки и возврата исходного парсера.

Пакет `org.xml.sax.ext` обеспечивает SAX-расширения и содержит классы и интерфейсы, представленные в табл. 2 и 3.

Таблица 2. *Интерфейсы пакета* `org.xml.sax.ext`

Интерфейс	Описание
`Attributes2`	Расширяет интерфейс `org.xml.sax.Attributes`. Объект `Attributes2` используется как аргумент метода `ContentHandler.startElement()`, если SAX-реализация поддерживает данное расширение, при этом режим парсера `http://xml.org/sax/features/use-attributes2` установлен в значение `true`. Интерфейс `Attributes2` дополнительно имеет методы: • `boolean isDeclared(int index)`, `boolean isDeclared(String qName)`, `boolean isDeclared(String uri, String localName)` — возвращают `true`, если атрибут объявлен в DTD-описании; • `boolean isSpecified(int index)`, `boolean isSpecified(String uri, String localName)`, `boolean isSpecified(String qName)` — возвращают `true`, если значение атрибута определенно непосредственно в документе, а не по умолчанию
`DeclHandler`	Используется вместе с `LexicalHandler` для получения событий между `LexicalHandler.startDTD` и `LexicalHandler.endDTD`. Объект `DeclHandler` регистрируется для парсера с помощью метода `setProperty()` со свойством `http://xml.org/sax/properties/declaration-handler`. Интерфейс `DeclHandler` имеет методы: • `void elementDecl(String name, String model)` — вызывается при разборе декларации типа элемента; • `void attributeDecl(String eName, String aName, String type, String mode, String value)` — вызывается при разборе декларации типа атрибута; • `void internalEntityDecl(String name, String value)` — вызывается при разборе декларации внутренней сущности; • `void externalEntityDecl(String name, String publicId, String systemId)` — вызывается при разборе декларации внешней сущности
`LexicalHandler`	Обеспечивает дополнительно информацию о комментариях и секциях `CDATA` документа. Объект `LexicalHandler` регистрируется для парсера с помощью метода `setProperty()` со свойством `http://xml.org/sax/properties/lexical-handler`. Интерфейс `LexicalHandler` имеет следующие методы: • `void startDTD(String name, String publicId, String systemId)` и `void endDTD()` — первый метод вызывается в начале разбора DTD-описания, второй метод — в конце; • `void startEntity(String name)` и `void endEntity(String name)` — первый метод вызывается в начале объявления сущности, второй метод — в конце; • `void startCDATA()` и `void endCDATA()` — первый метод вызывается в начале секции `CDATA`, второй метод — в конце; • `void comment(char[] ch, int start, int length)` — вызывается при разборе комментариев

***Таблица 2** (окончание)*

Интерфейс	Описание
`sEntityResolver2`	Расширяет интерфейс `org.xml.sax.EntityResolver`. Объект `EntityResolver2` регистрируется для парсера с помощью метода `XMLReader.setEntityResolver()`, в случае, если SAX-реализация поддерживает данное расширение, при этом режим парсера `http://xml.org/sax/features/use-entity-resolver2` устанавливается в значение `true`. Интерфейс `EntityResolver2` дополнительно имеет методы: • `InputSource getExternalSubset(String name, String baseURI)` — обеспечивает источник данных с внешним DTD-описанием; • `InputSource resolveEntity(String name, String publicId, String baseURI, String systemId)` — обеспечивает разрешение ссылок на внешние сущности
`Locator2`	Расширяет интерфейс `org.xml.sax.Locator`. Объект `Locator2` регистрируется для парсера с помощью метода `ContentHandler.setDocumentLocator()`, в случае, если SAX-реализация поддерживает данное расширение, при этом режим парсера `http://xml.org/sax/features/use-locator2` устанавливается в значение `true`. Интерфейс `Locator2` дополнительно имеет методы: • `String getXMLVersion()` — возвращает XML-версию сущности; • `String getEncoding()` — возвращает кодировку сущности

***Таблица 3.** Классы пакета `org.xml.sax.ext`*

Класс	Описание
`Attributes2Impl`	Расширяет класс `org.xml.sax.helpers.AttributesImpl`, реализация по умолчанию интерфейса `Attributes2`. Класс `Attributes2Impl` имеет конструкторы `public Attributes2Impl()` и `public Attributes2Impl(Attributes atts)` и дополнительно методы `public void setDeclared(int index, boolean value)` и `public void setSpecified(int index, boolean value)` — устанавливают значение флага декларации и определения значения атрибута
`DefaultHandler2`	Расширяет класс `org.xml.sax.helpers.DefaultHandler`, реализует интерфейсы `LexicalHandler`, `DeclHandler`, `EntityResolver2`. Класс `DefaultHandler2` имеет конструктор `public DefaultHandler2()`
`Locator2Impl`	Расширяет класс `org.xml.sax.helpers.LocatorImpl`, реализует интерфейс `Locator2`. Класс `Locator2Impl` имеет конструкторы `public Locator2Impl()` и `public Locator2Impl(Locator locator)` и дополнительно методы `public void setXMLVersion(String version)` и `public void setEncoding(String encoding)` — устанавливает XML-версию и кодировку сущности

Пакет `org.xml.sax.helpers` обеспечивает загрузку приложений, использующих SAX-анализ, и содержит классы, представленные в табл. 4.

Таблица 4. *Классы пакета* `org.xml.sax.helpers`

Класс	Описание
`AttributesImpl`	Реализует интерфейс `Attributes`. Класс `AttributesImpl` имеет конструкторы `public AttributesImpl()` и `public AttributesImpl(Attributes atts)` и дополнительно методы: • `public void clear()` — очищает список атрибутов; • `public void addAttribute(String uri, String localName, String qName, String type, String value)` — добавляет атрибут в список; • `public void setAttribute(int index, String uri, String localName, String qName, String type, String value)` — определяет атрибут в списке; • `public void removeAttribute(int index)` — удаляет атрибут из списка; • `public void setURI(int index, String uri)`, `public void setLocalName(int index, String localName)`, `public void setQName(int index, String qName)`, `public void setType(int index, String type)`, `public void setValue(int index, String value)` — устанавливают параметры атрибута в списке
`DefaultHandler`	Реализует интерфейсы `EntityResolver`, `DTDHandler`, `ContentHandler`, `ErrorHandler`. Класс `DefaultHandler` имеет конструктор `public DefaultHandler()`

`LocatorImpl`	Реализует интерфейс `Locator`, имеет конструкторы `public LocatorImpl()` и `public LocatorImpl(Locator locator)`, а также дополнительно методы:
	`public void setPublicId(String publicId)`
	`public void setSystemId(String systemId)`
	`public void setLineNumber(int lineNumber)`
	`public void setColumnNumber(int columnNumber)`

`NamespaceSupport`	Обеспечивает поддержку пространств имен с помощью следующих методов: • `public void reset()` — очищает объект `NamespaceSupport`; • `public void pushContext()` — начинает новый контекст пространства имен; • `public void popContext()` — возвращает к предыдущему контексту пространства имен; • `public boolean declarePrefix(String prefix, String uri)` — декларирует префикс пространства имен; • `public String[] processName(String qName, String[] parts, boolean isAttribute)` — возвращает URI пространства имен, локальное имя и QName-имя; • `public String getURI(String prefix)`, `public Enumeration getPrefixes()`, `public String getPrefix(String uri)`, `public Enumeration getPrefixes(String uri)`, `public Enumeration getDeclaredPrefixes()` — возвращают URI и префиксы контекста пространства имен; • `public void setNamespaceDeclUris(boolean value)` и `public boolean isNamespaceDeclUris()` — первый метод устанавливает, второй метод возвращает связь декларации пространства имен с атрибутом `xmlns`

Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

***Таблица 4** (окончание)*

Класс	Описание
`ParserAdapter`	Реализует интерфейсы `XMLReader`, `DocumentHandler`, имеет конструкторы `public ParserAdapter()` и `public ParserAdapter(Parser parser)`, с помощью которых адаптирует неиспользуемый интерфейс `org.xml.sax.Parser` как парсер XMLReader
`XMLFilterImpl`	Реализует интерфейсы `XMLFilter`, `EntityResolver`, `DTDHandler`, `ContentHandler`, `ErrorHandler`, имеет конструкторы `public XMLFilterImpl()` и `public XMLFilterImpl(XMLReader parent)`
`XMLReaderAdapter`	Реализует интерфейсы `Parser`, `ContentHandler`, имеет конструкторы `public XMLReaderAdapter()` и `public XMLReaderAdapter(XMLReader xmlReader)`, с помощью которых адаптирует интерфейс `XMLReader` как парсер Parser
`XMLReaderFactory`	Класс-фабрика для создания объектов `XMLReader` с помощью метода `public static XMLReader createXMLReader()` или `public static XMLReader createXMLReader(String className)`

TrAX

Технология XSLT (eXtensible Stylesheet Language Transformations) позволяет преобразование исходного дерева первоначального XML-документа в конечное дерево, представляющее документ, который может иметь формат XML, HTML, XHTML или TXT. При этом к исходному документу применяются таблицы XSLT-стилей — шаблоны, определяющие правила преобразования исходного дерева с помощью языка XPath.

Пакеты `javax.xml.transform`, `javax.xml.transform.sax`, `javax.xml.transform.dom` и `javax.xml.transform.stream` являются частью спецификации JAXP и составляют интерфейс Transformation API for XML (TrAX), обеспечивающий поддержку технологии XSLT для платформы Java.

Пакет `javax.xml.transform` используется для обработки таблиц XSLT-стилей и преобразования исходного документа в конечный документ.

Основным классом пакета `javax.xml.transform` является класс `Transformer`, обеспечивающий преобразование исходного дерева документа в конечное дерево. Класс `Transformer` предоставляет следующие методы:

- `public void reset()`

  Возвращает объект `Transformer` в исходную конфигурацию, обеспечивая его повторное использование.

- `public abstract void transform(Source xmlSource, Result outputTarget)`

  Осуществляет преобразование исходного документа в конечный документ. Интерфейс `javax.xml.transform.Source` представляет исходный документ и имеет методы `void setSystemId(String systemId)` и `String getSystemId()` — установка и получение системного идентификатора исходного документа. Интерфейс `javax.xml.transform.Result` представляет конечный документ и имеет методы

`void setSystemId(String systemId)` и `String getSystemId()` — установка и получение системного идентификатора конечного документа.

- `public abstract void setParameter(String name, Object value)`
  `public abstract Object getParameter(String name)`
  `public abstract void clearParameters()`

  Первый метод устанавливает, второй метод возвращает, третий метод очищает параметры, определенные в таблице XSLT-стилей преобразования.

- `public abstract void setURIResolver(URIResolver resolver)`
  `public abstract URIResolver getURIResolver()`

  Первый метод устанавливает, второй метод возвращает объект `URIResolver`, используемый при преобразовании для разрешения URI-адресов в документе. Интерфейс `javax.xml.transform.URIResolver` имеет метод `Source resolve(String href, String base)`, вызываемый процессором для разрешения URI-ссылок в объекты `Source`.

- `public abstract void setOutputProperties(java.util.Properties oformat)`
  `public abstract java.util.Properties getOutputProperties()`

  Первый метод устанавливает, второй метод возвращает такие свойства преобразования, как:

  - `javax.xml.transform.OutputKeys.METHOD` — значения `"xml"`, `"html"` или `"text"` означают преобразование в документ формата XML, HTML или TXT соответственно;
  - `javax.xml.transform.OutputKeys.VERSION` — версия формата конечного документа, например, для формата XML это может быть значение 1.0;
  - `javax.xml.transform.OutputKeys.ENCODING` — кодировка конечного документа, например, UTF-8 или UTF-16;
  - `javax.xml.transform.OutputKeys.OMIT_XML_DECLARATION` — значения `"yes"` или `"no"` определяют, должен ли XSLT-процессор выдавать в конечном документе XML-декларацию `<?xml ...?>`;
  - `javax.xml.transform.OutputKeys.STANDALONE` — значения `"yes"` или `"no"` определяют, должен ли XSLT-процессор выдавать в конечном документе объявление `standalone="yes"`;
  - `javax.xml.transform.OutputKeys.DOCTYPE_PUBLIC` — публичный идентификатор DTD-описания конечного документа;
  - `javax.xml.transform.OutputKeys.DOCTYPE_SYSTEM` — системный идентификатор DTD-описания конечного документа;
  - `javax.xml.transform.OutputKeys.CDATA_SECTION_ELEMENTS` — перечень элементов с секциями `CDATA`;
  - `javax.xml.transform.OutputKeys.INDENT` — значения `"yes"` или `"no"` определяют, может ли XSLT-процессор структурировать конечное дерево путем добавления дополнительных пробелов;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `javax.xml.transform.OutputKeys.MEDIA_TYPE` — MIME-тип данных конечного дерева.

❒ `public abstract void setErrorListener(ErrorListener listener)`

`public abstract ErrorListener getErrorListener()`

Первый метод устанавливает, второй метод возвращает объект `javax.xml.transform.ErrorListener`, обеспечивающий обработку ошибок XSLT-преобразования. Интерфейс `ErrorListener` имеет методы `void warning(TransformerException exception)`, `void error(TransformerException exception)` и `void fatalError(TransformerException exception)`, получающие уведомления об ошибках и предупреждениях.

Объект `Transformer` создается с помощью класса-фабрики `javax.xml.transform.TransformerFactory`, имеющего следующие методы:

❒ `public static TransformerFactory newInstance()`

`public static TransformerFactory newInstance(String factoryClassName, ClassLoader classLoader)`

Создают новый экземпляр класса `TransformerFactory`.

❒ `public abstract Transformer newTransformer(Source source)`

Создает объект `Transformer`, используемый для преобразования исходного документа.

❒ `public abstract Transformer newTransformer()`

Создает объект `Transformer`, используемый для копирования исходного документа.

❒ `public abstract Templates newTemplates(Source source)`

Создает объект `javax.xml.transform.Templates`, представляющий таблицы XSLT-стилей. Интерфейс `Templates` имеет методы `Transformer newTransformer()` — создает новый объект `Transformer` для данного XSLT-шаблона, и `Properties getOutputProperties()` — возвращает свойства преобразования.

❒ `public abstract Source getAssociatedStylesheet(Source source, String media, String title, String charset)`

Возвращает каскадные таблицы стилей, связанные с исходным документом.

❒ `public abstract void setURIResolver(URIResolver resolver)`

`public abstract URIResolver getURIResolver()`

Устанавливает и возвращает объект `URIResolver`, используемый при преобразовании для разрешения URI-адресов в документе.

❒ `public abstract void setFeature(String name, boolean value)`

`public abstract boolean getFeature(String name)`

`public abstract void setAttribute(String name, Object value)`

`public abstract Object getAttribute(String name)`

Первая пара методов устанавливает и возвращает свойства преобразования, вторая пара методов — атрибуты преобразования.

- `public abstract void setErrorListener(ErrorListener listener)`
  `public abstract ErrorListener getErrorListener()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.transform.ErrorListener`, обеспечивающий обработку ошибок XSLT-преобразования.

Пакет `javax.xml.transform.dom` позволяет оперировать DOM-деревьями в качестве исходных и конечных деревьев XSLT-преобразования. Пакет `javax.xml.transform.dom` содержит классы `DOMSource` и `DOMResult`, реализующие интерфейсы `Source` и `Result` пакета `javax.xml.transform` соответственно.

Класс `DOMSource` представляет исходное DOM-дерево XSLT-преобразования и имеет следующие конструкторы и методы:

- `public DOMSource()`
  `public DOMSource(Node n)`
  `public DOMSource(Node node, String systemID)`

  Создает новый источник преобразования.

- `public void setNode(Node node)`
  `public Node getNode()`

  Первый метод устанавливает, второй метод возвращает DOM-узел, представляющий исходное дерево.

- `public void setSystemId(String systemID)`
  `public String getSystemId()`

  Первый метод устанавливает, второй метод возвращает базовый идентификатор для разрешения URL-адресов.

Класс `DOMResult` представляет конечное DOM-дерево XSLT-преобразования и имеет следующие конструкторы и методы:

- `public DOMResult()`
  `public DOMResult(Node node)`
  `public DOMResult(Node node, String systemId)`
  `public DOMResult(Node node, Node nextSibling)`
  `public DOMResult(Node node, Node nextSibling, String systemId)`

  Создают новый объект `DOMResult`.

- `public void setNode(Node node)`
  `public Node getNode()`

  Первый метод устанавливает, второй метод возвращает DOM-узел, содержащий конечное дерево преобразования.

- `public void setNextSibling(Node nextSibling)`
  `public Node getNextSibling()`

  Первый метод устанавливает, второй метод возвращает DOM-узел, являющийся дочерним для конечного дерева.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public void setSystemId(String systemId)`
  `public String getSystemId()`

  Первый метод устанавливает, второй метод возвращает системный идентификатор конечного дерева.

Пакет `javax.xml.transform.sax` позволяет использовать SAX-потоки событий при XSLT-преобразованиях. Пакет `javax.xml.transform.sax` содержит классы `SAXResult`, `SAXSource` и `SAXTransformerFactory`, а также интерфейсы `TemplatesHandler` и `TransformerHandler`.

Класс `SAXResult` реализует интерфейс `javax.xml.transform.Result` и имеет следующие конструкторы и методы:

- `public SAXResult()`
  `public SAXResult(ContentHandler handler)`

  Создают конечный объект XSLT-преобразования.

- `public void setHandler(ContentHandler handler)`
  `public ContentHandler getHandler()`

  Первый метод устанавливает, второй метод возвращает объект `ContentHandler`, позволяющий анализировать конечный документ.

- `public void setLexicalHandler(LexicalHandler handler)`
  `public LexicalHandler getLexicalHandler()`

  Первый метод устанавливает, второй метод возвращает объект `org.xml.sax.ext.LexicalHandler`, обеспечивающий лексический анализ конечного документа.

- `public void setSystemId(String systemId)`
  `public String getSystemId()`

  Первый метод устанавливает, второй метод возвращает системный идентификатор конечного документа.

Класс `SAXSource` реализует интерфейс `javax.xml.transform.Source` и имеет следующие конструкторы и методы:

- `public SAXSource()`
  `public SAXSource(XMLReader reader, InputSource inputSource)`
  `public SAXSource(InputSource inputSource)`

  Создают исходный объект XSLT-преобразования.

- `public void setXMLReader(XMLReader reader)`
  `public XMLReader getXMLReader()`

  Первый метод устанавливает, второй метод возвращает объект `org.xml.sax.XMLReader`, представляющий SAX-парсер исходного документа.

- `public void setInputSource(InputSource inputSource)`
  `public InputSource getInputSource()`

Первый метод устанавливает, второй метод возвращает объект `org.xml.sax.InputSource`, представляющий входящий поток XML-данных.

- `public void setSystemId(String systemId)`

  `public String getSystemId()`

  Первый метод устанавливает, второй метод возвращает системный идентификатор исходных данных.

- `public static InputSource sourceToInputSource(Source source)`

  Преобразует объект `Source` в объект `InputSource`.

Интерфейс `TemplatesHandler` расширяет интерфейс `org.xml.sax.ContentHandler` и предназначен для SAX-анализа шаблона XSLT-преобразования. Интерфейс `TemplatesHandler` дополнительно имеет методы `Templates getTemplates()`, `void setSystemId(String systemID)` и `String getSystemId()`.

Интерфейс `TransformerHandler` расширяет интерфейсы `org.xml.sax.ContentHandler`, `org.xml.sax.ext.LexicalHandler` и `org.xml.sax.DTDHandler` и обеспечивает XSLT-преобразование SAX-потока событий. Интерфейс `TransformerHandler` дополнительно имеет методы `void setResult(Result result)`, `void setSystemId(String systemID)`, `String getSystemId()` и `Transformer getTransformer()`.

Класс-фабрика `SAXTransformerFactory` расширяет класс-фабрику `javax.xml.transform.TransformerFactory` и позволяет создавать объекты `TemplatesHandler`, `TransformerHandler` и `XMLFilter` с помощью методов:

```
public abstract TransformerHandler newTransformerHandler(Source src)
public abstract TransformerHandler newTransformerHandler(Templates
  templates)
public abstract TransformerHandler newTransformerHandler()
public abstract TemplatesHandler newTemplatesHandler()
public abstract XMLFilter newXMLFilter(Source src)
public abstract XMLFilter newXMLFilter(Templates templates)
```

Пакет `javax.xml.transform.stax` позволяет использовать StAX-потоки при XSLT-преобразованиях, предоставляя классы `StAXResult` и `StAXSource`.

Класс `StAXResult` реализует интерфейс `javax.xml.transform.Result` и имеет следующие конструкторы и методы:

- `public StAXResult(XMLEventWriter xmlEventWriter)`
- `public StAXResult(XMLStreamWriter xmlStreamWriter)`

  Создают новый экземпляр класса StAXResult.

- `public XMLEventWriter getXMLEventWriter()`

  `public XMLStreamWriter getXMLStreamWriter()`

  Возвращают объекты `javax.xml.stream.XMLEventWriter` и `javax.xml.stream.XMLStreamWriter`, используемые для записи XML-документа.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public void setSystemId(String systemId)`

  `public String getSystemId()`

  Устанавливает и возвращает системный идентификатор конечных данных.

Класс `StAXSource` реализует интерфейс `javax.xml.transform.Source` и имеет следующие конструкторы и методы:

- `public StAXSource(XMLEventReader xmlEventReader)`

  `public StAXSource(XMLStreamReader xmlStreamReader)`

  Создают новый экземпляр класса `StAXSource`.

- `public XMLEventReader getXMLEventReader()`

  `public XMLStreamReader getXMLStreamReader()`

  Возвращают объекты `javax.xml.stream.XMLEventReader` и `javax.xml.stream.XMLStreamReader`, используемые для анализа исходного документа.

- `public void setSystemId(String systemId)`

  `public String getSystemId()`

  Первый метод устанавливает, второй метод возвращает системный идентификатор исходных данных.

Пакет `javax.xml.transform.stream` дает возможность использовать различного рода потоки при XSLT-преобразованиях, предоставляя для этого классы `StreamResult` и `StreamSource`.

Класс `StreamResult`, представляющий результат XSLT-преобразования, реализует интерфейс `javax.xml.transform.Result` и позволяет создавать и обрабатывать объекты из символьного или байтового потока, из URL-адреса или файла, используя следующие конструкторы и методы:

- `public StreamResult()`

  `public StreamResult(OutputStream outputStream)`

  `public StreamResult(Writer writer)`

  `public StreamResult(String systemId)`

  `public StreamResult(File f)`

  Создают объект `StreamResult`.

- `public void setOutputStream(OutputStream outputStream)`

  `public OutputStream getOutputStream()`

  Первый метод устанавливает, второй метод возвращает байтовый поток, в который записывается результат XSLT-преобразования.

- `public void setWriter(Writer writer)`

  `public Writer getWriter()`

  Первый метод устанавливает, второй метод возвращает символьный поток, в который записывается результат XSLT-преобразования.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public void setSystemId(String systemId)`
  `public void setSystemId(File f)`
  `public String getSystemId()`

  Первые два метода устанавливают, третий метод возвращает системный идентификатор результата XSLT-преобразования.

Класс `StreamSource`, представляющий источник XSLT-преобразования, реализует интерфейс `javax.xml.transform.Source` и дает возможность работать с различного рода потоками с помощью следующих конструкторов и методов:

- `public StreamSource()`
  `public StreamSource(InputStream inputStream)`
  `public StreamSource(InputStream inputStream, String systemId)`
  `public StreamSource(Reader reader)`
  `public StreamSource(Reader reader, String systemId)`
  `public StreamSource(String systemId)`
  `public StreamSource(File f)`

  Создают объект `StreamSource`.

- `public void setInputStream(InputStream inputStream)`
  `public InputStream getInputStream()`

  Первый метод устанавливает, второй метод возвращает входящий байтовый поток.

- `public void setReader(Reader reader)`
  `public Reader getReader()`

  Первый метод устанавливает, второй метод возвращает входящий символьный поток.

- `public void setPublicId(String publicId)`
  `public String getPublicId()`
  `public void setSystemId(String systemId)`
  `public String getSystemId()`
  `public void setSystemId(File f)`

  Методы устанавливают или возвращают публичный и системный идентификаторы исходных данных XSLT-преобразования.

Таким образом, простой код, обеспечивающий XSLT-преобразование XML-данных в требуемый формат, может выглядеть следующим образом:

```
Source inputXML = new StreamSource(inputStreamXML);
Source inputXSL = new StreamSource(inputStreamXSL);
Result output = new StreamResult(outputStream);
TransformerFactory factory = TransformerFactory.newInstance();
Templates templates = factory.newTemplates(inputXSL);
Transformer transformer = templates.newTransformer();
transformer.transform(inputXML, output);
```

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

XPath API

Программный интерфейс XPath API, представленный пакетом `javax.xml.xpath`, обеспечивает запросы к XML-элементам с помощью XPath-выражений.

Основным интерфейсом пакета `javax.xml.xpath` является интерфейс `XPath`, позволяющий обрабатывать XPath-выражения с помощью следующих методов:

- `void reset()`

  Сбрасывает объект `XPath` в исходное состояние, тем самым обеспечивая его повторное использование.

- `void setXPathVariableResolver(XPathVariableResolver resolver)`

  `XPathVariableResolver getXPathVariableResolver()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.xpath.XPathVariableResolver`, обеспечивающий доступ к переменным XPath-выражения. Интерфейс `XPathVariableResolver` имеет метод `Object resolveVariable(QName variableName)`, возвращающий значение переменной с указанным именем.

- `void setXPathFunctionResolver(XPathFunctionResolver resolver)`

  `XPathFunctionResolver getXPathFunctionResolver()`

  Первый метод устанавливает, второй метод возвращает объект `javax.xml.xpath.XPathFunctionResolver`, обеспечивающий доступ к пользовательским XPath-функциям. Интерфейс `XPathFunctionResolver` имеет метод `XPathFunction resolveFunction(QName functionName, int arity)`, возвращающий объект `javax.xml.xpath.XPathFunction`, представляющий XPath-функцию с указанным именем и определенным количеством параметров. Интерфейс `XPathFunction` имеет метод `Object evaluate(List args)`, возвращающий результат вычисления функции с определенным списком аргументов.

- `void setNamespaceContext(NamespaceContext nsContext)`

  `NamespaceContext getNamespaceContext()`

  Первый метод устанавливает, второй метод возвращает используемый контекст пространства имен.

- `XPathExpression compile(String expression)`

  Компилирует XPath-выражение в объект `javax.xml.xpath.XpathExpression`. Интерфейс `XPathExpression` обеспечивает обработку XPath-выражений с помощью следующих методов:

  - `String evaluate(Object item)` или `Object evaluate(Object item, QName returnType)` — вычисляет XPath-выражение в контексте указанного объекта, например узла, и возвращает результат вычисления определенного типа или строку. Результат вычисления XPath-выражения может быть следующих типов: `XPathConstants.STRING`, `XPathConstants.NUMBER`, `XPathConstants.BOOLEAN`, `XPathConstants.NODE` и `XPathConstants.NODESET`;

- `String evaluate(InputSource source)` или `Object evaluate(InputSource source, QName returnType)` — вычисляет XPath-выражение в контексте входящих данных `org.xml.sax.InputSource` и возвращает результат вычисления определенного типа или строку.

❒ `Object evaluate(String expression, Object item, QName returnType)`

`String evaluate(String expression, Object item)`

`Object evaluate(String expression, InputSource source, QName returnType)`

`String evaluate(String expression, InputSource source)`

Вычисляют XPath-выражение в определенном контексте.

Объект `XPath` создается с помощью класса-фабрики `javax.xml.xpath.XPathFactory`, имеющим перечисленные далее методы.

❒ `public static final XPathFactory newInstance()`

`public static final XPathFactory newInstance(String uri)`

`public static XPathFactory newInstance(String uri, String factoryClassName, ClassLoader classLoader)`

Создают новый экземпляр `XPathFactory`.

❒ `public abstract boolean isObjectModelSupported(String objectModel)`

Возвращает `true`, если XPath-реализация поддерживает указанную модель, например `http://java.sun.com/jaxp/xpath/dom`.

❒ `public abstract void setFeature(String name, boolean value)`

`public abstract boolean getFeature(String name)`

Первый метод устанавливает, второй метод возвращает свойства XPath-реализации.

❒ `public abstract void setXPathVariableResolver(XPathVariableResolver resolver)`

`public abstract void setXPathFunctionResolver(XPathFunctionResolver resolver)`

Устанавливает объекты `XpathVariableResolver` и `XpathFunctionResolver` для XPath-реализации.

❒ `public abstract XPath newXPath()`

Создает объект XPath.

Пакет *javax.xml.validation*

Пакет `javax.xml.validation` обеспечивает программный интерфейс для проверки XML-документов относительно их описания.

Для проверки XML-документов относительно описания их структуры пакет `javax.xml.validation` предоставляет два класса: `Validator` и `ValidatorHandler`.

Класс `Validator` позволяет работать с объектами `javax.xml.transform.Source`, представляющими проверяемые XML-документы, и имеет следующие методы:

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public abstract void reset()`

  Сбрасывает объект `Validator` в его исходное состояние, тем самым обеспечивая его повторное использование.

- `public abstract void validate(Source source, Result result)`

  `public void validate(Source source)`

  Осуществляют проверку определенного XML-документа или XML-элемента.

- `public abstract void setErrorHandler(ErrorHandler errorHandler)`

  `public abstract ErrorHandler getErrorHandler()`

  Первый метод устанавливает, второй метод возвращает объект `org.xml.sax.ErrorHandler`, получающий уведомления об ошибках в процессе проверки XML-документа.

- `public abstract void setResourceResolver(LSResourceResolver resourceResolver)`

  `public abstract LSResourceResolver getResourceResolver()`

  Первый метод устанавливает, второй метод возвращает объект `org.w3c.dom.ls.LSResourceResolver`, используемый для разрешения ссылок на внешние ресурсы.

- `public void setFeature(String name, boolean value)`

  `public boolean getFeature(String name)`

  `public void setProperty(String name, Object object)`

  `public Object getProperty(String name)`

  Первая пара методов устанавливает и возвращает режимы реализации, вторая пара методов — свойства реализации.

Класс `ValidatorHandler` реализует интерфейс `org.xml.sax.ContentHandler` и обеспечивает проверку SAX-потоков, проверяя, соответствуют ли SAX-события ограничениям описания структуры XML-документа. Класс `ValidatorHandler` дополнительно к методам интерфейса `ContentHandler` имеет следующие методы:

- `public abstract void setContentHandler(ContentHandler receiver)`

  `public abstract ContentHandler getContentHandler()`

  Первый метод устанавливает, второй метод возвращает объект `ContentHandler`, получающий результат проверки.

- `public abstract void setErrorHandler(ErrorHandler errorHandler)`

  `public abstract ErrorHandler getErrorHandler()`

  Первый метод устанавливает, второй метод возвращает объект `org.xml.sax.ErrorHandler`, получающий уведомления об ошибках в процессе проверки.

- `public abstract void setResourceResolver(LSResourceResolver resourceResolver)`

  `public abstract LSResourceResolver getResourceResolver()`

  Первый метод устанавливает, второй метод возвращает объект `org.w3c.dom.ls.LSResourceResolver`, используемый для разрешения ссылок на внешние ресурсы.

- `public abstract TypeInfoProvider getTypeInfoProvider()`

  Возвращает объект `javax.xml.validation.TypeInfoProvider`, обеспечивающий доступ к информации о типах данных атрибута или элемента. Класс `TypeInfoProvider` имеет методы:

  - `public abstract TypeInfo getElementTypeInfo()` — возвращает объект `org.w3c.dom.TypeInfo`, представляющий тип данных текущего элемента;
  - `public abstract TypeInfo getAttributeTypeInfo(int index)` — возвращает тип данных определенного атрибута текущего элемента;
  - `public abstract boolean isIdAttribute(int index)` — возвращает `true`, если атрибут является идентификатором;
  - `public abstract boolean isSpecified(int index)` — возвращает `false`, если атрибут был добавлен валидатором.

- `public void setFeature(String name, boolean value)`
  `public boolean getFeature(String name)`
  `public void setProperty(String name, Object object)`
  `public Object getProperty(String name)`

  Первая пара методов устанавливает и возвращает режимы реализации, вторая пара методов — свойства реализации.

Объекты `Validator` и `ValidatorHandler` создаются с помощью методов `public abstract Validator newValidator()` и `public abstract ValidatorHandler newValidatorHandler()` класса `javax.xml.validation.Schema`, представляющего описание структуры XML-документа или его схему, созданную с использованием таких языков, как Document Type Definition (DTD), W3C XML Schema (WXS), RELAX NG (RNG), Schematron и др.

Для создания объектов `Schema` пакет `javax.xml.validation` предоставляет класс-фабрику `SchemaFactory`, который имеет следующие методы:

- `public static final SchemaFactory newInstance(String schemaLanguage)`
  `public static SchemaFactory newInstance(String schemaLanguage, String factoryClassName, ClassLoader classLoader)`

  Создает объект `SchemaFactory` реализации с поддержкой указанного языка, например, `XMLConstants.W3C_XML_SCHEMA_NS_URI` или `XMLConstants.RELAXNG_NS_URI`.

- `public abstract boolean isSchemaLanguageSupported(String schemaLanguage)`

  Возвращает `true`, если объект `SchemaFactory` поддерживает указанный язык описания структуры XML-документа.

- `public boolean getFeature(String name)`
  `public void setFeature(String name, boolean value)`
  `public void setProperty(String name, Object object)`
  `public Object getProperty(String name)`

  Первая пара методов устанавливает и возвращает режимы реализации, вторая пара методов — свойства реализации.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public abstract void setErrorHandler(ErrorHandler errorHandler)`

  `public abstract ErrorHandler getErrorHandler()`

  Первый метод устанавливает, второй метод возвращает объект `org.xml.sax.ErrorHandler`, получающий уведомления об ошибках в процессе компиляции схемы XML-документа.

- `public abstract void setResourceResolver(LSResourceResolver resourceResolver)`

  `public abstract LSResourceResolver getResourceResolver()`

  Первый метод устанавливает, второй метод возвращает объект `org.w3c.dom.ls.LSResourceResolver`, используемый для разрешения ссылок на внешние ресурсы в процессе компиляции схемы XML-документа.

- `public Schema newSchema(Source schema)`

  `public Schema newSchema(File schema)`

  `public Schema newSchema(URL schema)`

  `public abstract Schema newSchema(Source[] schemas)`

  `public abstract Schema newSchema()`

  Подготавливают схему XML-документа к использованию для проверки XML-документов путем создания объекта `Schema`.

JAX-RPC API

Пакет *javax.xml.rpc*

Пакет `javax.xml.rpc` содержит интерфейсы `Call`, `Service`, `Stub` и классы `NamespaceConstants`, `ParameterMode` и `ServiceFactory`, а также исключения `JAXRPCException` и `ServiceException`.

Интерфейс `Stub` реализуется `Stub`-классом, генерируемым JAX-RPC-реализацией на стороне клиента. Интерфейс `Stub` имеет следующие свойства и методы:

- `static final java.lang.String USERNAME_PROPERTY` — имя пользователя для аутентификации;
- `static final java.lang.String PASSWORD_PROPERTY` — пароль для аутентификации;

  Примечание

  Свойства `USERNAME_PROPERTY` и `PASSWORD_PROPERTY` интерфейса `Stub` позволяют реализовывать программную аутентификацию клиента для конечной точки Web-сервиса.

- `static final java.lang.String ENDPOINT_ADDRESS_PROPERTY` — адрес конечной точки Web-сервиса;
- `static final java.lang.String SESSION_MAINTAIN_PROPERTY` — если `true`, тогда клиент участвует в HTTP-сессии, инициированной конечной точкой Web-сервиса;
- `void _setProperty(java.lang.String name, java.lang.Object value)`, `java.lang.Object _getProperty(java.lang.String name)`, `java.util.Iterator _getPropertyNames()` — первый метод устанавливает, второй метод возвращает свойства `Stub`-объекта.

Класс `ServiceFactory` используется клиентским приложением для создания Service-объекта, являющегося фабрикой объектов статической или динамической заглушки, а также `Call`-объектов. Реализация `ServiceFactory` обеспечивается средой выполнения JAX-RPC. Класс `ServiceFactory` имеет следующие поля и методы:

- `public static final java.lang.String SERVICEFACTORY_PROPERTY` — имя реализации класса `ServiceFactory`;
- `public static ServiceFactory newInstance()` — возвращает `ServiceFactory`-объект;
- `public abstract Service createService(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)`

  `public abstract Service createService(javax.xml.namespace.QName serviceName)`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

`public abstract Service loadService(java.lang.Class serviceInterface)`

`public abstract Service loadService(java.net.URL wsdlDocumentLocation, java.lang.Class serviceInterface, java.util.Properties properties)`

`public abstract Service loadService(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName, java.util.Properties properties)`

Возвращают `Service`-объект.

Интерфейс `Service` реализуется генерируемым с помощью JAX-RPC-инструмента `Service`-классом или `Service`-объектом, полученным методом класса `ServiceFactory`. `Service`-объект служит фабрикой для создания `Stub`-объектов, `Proxy`-объектов и `Call`-объектов, используя следующие методы.

- `java.rmi.Remote getPort(javax.xml.namespace.QName portName, java.lang.Class serviceEndpointInterface)`

  `java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)`

  Возвращают `Stub`-объект или `Proxy`-объект.

- `Call[] getCalls(javax.xml.namespace.QName portName)`

  `Call createCall(javax.xml.namespace.QName portName)`

  `Call createCall(javax.xml.namespace.QName portName, javax.xml.namespace.QName operationName)`

  `Call createCall(javax.xml.namespace.QName portName, java.lang.String operationName)`

  `Call createCall()`

  Возвращают `Call`-объекты.

- `javax.xml.namespace.QName getServiceName()`

  Возвращает имя Web-сервиса.

- `java.util.Iterator getPorts()`

  Возвращает итератор для списка портов Web-сервиса.

- `java.net.URL getWSDLDocumentLocation()`

  Возвращает адрес WSDL-документа.

- `TypeMappingRegistry getTypeMappingRegistry()`

  Возвращает объект `javax.xml.rpc.encoding.TypeMappingRegistry`, служащий реестром объектов `javax.xml.rpc.encoding.TypeMapping`, которые обеспечивают связывание Java-типов с XML-типами.

- `HandlerRegistry getHandlerRegistry()`

  Возвращает объект `javax.xml.rpc.handler.HandlerRegistry`, служащий реестром обработчиков SOAP-сообщений.

Интерфейс `Call` обеспечивает динамический вызов конечной точки Web-сервиса без генерации заглушки с помощью следующих свойств и методов.

- `static final java.lang.String USERNAME_PROPERTY` — имя клиента для аутентификации.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `static final java.lang.String PASSWORD_PROPERTY` — пароль для аутентификации.

  Примечание

  Свойства `USERNAME_PROPERTY` и `PASSWORD_PROPERTY` интерфейса `Call` позволяют реализовывать программную аутентификацию клиента для конечной точки Web-сервиса.

- `static final java.lang.String OPERATION_STYLE_PROPERTY` — стиль `"rpc"` или `"document"` вызываемой операции.
- `static final java.lang.String SOAPACTION_USE_PROPERTY` — если `true`, тогда заголовок `SOAPAction` используется.
- `static final java.lang.String SOAPACTION_URI_PROPERTY` — URI-идентификатор для `SOAPAction`-заголовка.
- `static final java.lang.String ENCODINGSTYLE_URI_PROPERTY` — URI стиля кодировки, по умолчанию `http://schemas.xmlsoap.org/soap/encoding/`.
- `static final java.lang.String SESSION_MAINTAIN_PROPERTY` — если `true`, тогда клиент участвует в сессии, инициированной конечной точкой.
- `boolean isParameterAndReturnSpecRequired(javax.xml.namespace.QName` *`operationName`*`)`

  Если `true`, тогда вызывались методы `addParameter()` и `setReturnType()` для указанной операции.

- `void addParameter(java.lang.String` *`paramName`*`, javax.xml.namespace.QName` *`xmlType`*`, ParameterMode` *`parameterMode`*`)`

  `void addParameter(java.lang.String` *`paramName`*`, javax.xml.namespace.QName` *`xmlType`*`, java.lang.Class` *`javaType`*`, ParameterMode` *`parameterMode`*`)`

  Устанавливают параметры для вызываемой операции. Объект `javax.xml.rpc.ParameterMode` указывает тип параметра `IN`, `OUT` или `INOUT`.

- `javax.xml.namespace.QName getParameterTypeByName(java.lang.String` *`paramName`*`)`

  Возвращает XML-тип параметра.

- `void setReturnType(javax.xml.namespace.QName` *`xmlType`*`)`

  `void setReturnType(javax.xml.namespace.QName` *`xmlType`*`, java.lang.Class` *`javaType`*`)`

  `javax.xml.namespace.QName getReturnType()`

  Устанавливает и выдает тип возвращаемого операцией значения.

- `void removeAllParameters()`

  `setReturnType(null)`

  Очищают объект `Call`.

- `javax.xml.namespace.QName getOperationName()`

  `void setOperationName(javax.xml.namespace.QName` *`operationName`*`)`

  Первый метод возвращает, второй метод устанавливает имя операции.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `javax.xml.namespace.QName getPortTypeName()`

  `void setPortTypeName(javax.xml.namespace.QName portType)`

  Первый метод возвращает, второй метод устанавливает имя порта.

- `void setTargetEndpointAddress(java.lang.String address)`

  `java.lang.String getTargetEndpointAddress()`

  Первый метод устанавливает, второй метод возвращает адрес конечной точки Web-сервиса.

- `void setProperty(java.lang.String name, java.lang.Object value)`

  `java.lang.Object getProperty(java.lang.String name)`

  `void removeProperty(java.lang.String name)`

  Первый метод устанавливает, второй метод возвращает, а третий метод удаляет свойства `Call`-объекта.

- `java.util.Iterator getPropertyNames()`

  Возвращает итератор свойств `Call`-объекта.

- `java.lang.Object invoke(java.lang.Object[] inputParams)`

  `java.lang.Object invoke(javax.xml.namespace.QName operationName, java.lang.Object[] inputParams)`

  Вызывают операцию в синхронном режиме "запрос — ответ".

- `void invokeOneWay(java.lang.Object[] inputParams)`

  Вызывает операцию в синхронном режиме "запрос без ответа", ожидает HTTP-код 200 или HTTP-код ошибки.

- `java.util.Map getOutputParams()`

  `java.util.List getOutputValues()`

  Возвращают исходящие параметры.

Пакет *javax.xml.rpc.encoding*

Пакет `javax.xml.rpc.encoding` обеспечивает связывание Java-типов с XML-типами для JAX-RPC-реализации. Пакет `javax.xml.rpc.encoding` содержит класс `XMLType` и интерфейсы `DeserializationContext`, `Deserializer`, `DeserializerFactory`, `SerializationContext`, `Serializer`, `SerializerFactory`, `TypeMapping` и `TypeMappingRegistry`.

Интерфейс `DeserializationContext` реализуется средой выполнения JAX-RPC для создания контекста десериализации.

Интерфейс `Deserializer` реализуется классами-десериализаторами, конвертирующими XML-представления в Java-объекты. Интерфейс `Deserializer` имеет метод `java.lang.String getMechanismType()`, возвращающий идентификатор типа XML-обработки конвертируемых XML-данных, например, `http://java.sun.com/jax-rpc-ri/1.0/streaming/`.

Объекты `Deserializer` создаются с помощью интерфейса `DeserializerFactory`, имеющего методы `Deserializer getDeserializerAs(java.lang.String` *`mechanismType`*`)` и `java.util.Iterator getSupportedMechanismTypes()`.

Интерфейс `SerializationContext` реализуется средой выполнения JAX-RPC для создания контекста сериализации.

Интерфейс `Serializer` реализуется классами-сериализаторами, конвертирующими Java-объекты в XML-представление. Интерфейс `Serializer` имеет метод `java.lang.String getMechanismType()`, возвращающий идентификатор типа результирующих XML-данных, которые могут быть, например, SAX-потоком или DOM-деревом.

Объекты `Serializer` создаются с помощью интерфейса `SerializerFactory`, имеющего методы `Serializer getSerializerAs(java.lang.String` *`mechanismType`*`)` и `java.util.Iterator getSupportedMechanismTypes()`.

Объекты `DeserializerFactory` и `SerializerFactory` содержатся в объекте `TypeMapping` в виде наборов {Java-тип, `SerializerFactory`, `DeserializerFactory`, XML-тип}. Интерфейс `TypeMapping` имеет перечисленные далее методы.

- `java.lang.String[] getSupportedEncodings()`

  `void setSupportedEncodings(java.lang.String[]` *`encodingStyleURIs`*`)`

  возвращает и устанавливает поддерживаемые стили кодировки.

- `boolean isRegistered(java.lang.Class` *`javaType`*`,`
  `javax.xml.namespace.QName` *`xmlType`*`)`

  Возвращает `true`, если связь между Java-типом и XML-типом зарегистрирована.

- `void register(java.lang.Class` *`javaType`*`,`
  `javax.xml.namespace.QName` *`xmlType`*`, SerializerFactory` *`sf`*`,`
  `DeserializerFactory` *`dsf`*`)`

  Регистрирует связь между Java-типом и XML-типом с указанными фабриками для сериализаторов и десериализаторов.

- `SerializerFactory getSerializer(java.lang.Class` *`javaType`*`,`
  `javax.xml.namespace.QName` *`xmlType`*`)`

  `DeserializerFactory getDeserializer(java.lang.Class` *`javaType`*`,`
  `javax.xml.namespace.QName` *`xmlType`*`)`

  Первый метод возвращает фабрику сериализатора для определенной связи, второй метод — фабрику десериализатора.

- `void removeSerializer(java.lang.Class` *`javaType`*`,`
  `javax.xml.namespace.QName` *`xmlType`*`)`

  `void removeDeserializer(java.lang.Class` *`javaType`*`,`
  `javax.xml.namespace.QName` *`xmlType`*`)`

  Первый метод удаляет фабрику сериализатора для определенной связи, второй метод — фабрику десериализатора.

Объекты `TypeMapping` содержатся в реестре `TypeMappingRegistry`, получаемом методом `getTypeMappingRegistry()` интерфейса `Service`. Интерфейс `TypeMappingRegistry` имеет следующие методы для управления реестром связей.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `TypeMapping register(java.lang.String encodingStyleURI, TypeMapping mapping)`

  Регистрирует объект `TypeMapping` с определенным стилем кодировки.

- `void registerDefault(TypeMapping mapping)`

  `TypeMapping getDefaultTypeMapping()`

  Первый метод регистрирует, второй метод возвращает объект `TypeMapping` для всех кодировок по умолчанию.

- `java.lang.String[] getRegisteredEncodingStyleURIs()`

  Возвращает список зарегистрированных стилей кодировок.

- `TypeMapping getTypeMapping(java.lang.String encodingStyleURI)`

  Возвращает зарегистрированный объект `TypeMapping` для определенной кодировки.

- `TypeMapping createTypeMapping()`

  Создает новый пустой объект `TypeMapping`.

- `TypeMapping unregisterTypeMapping(java.lang.String encodingStyleURI)`

  Аннулирует регистрацию объекта `TypeMapping` для определенного стиля кодировки.

- `boolean removeTypeMapping(TypeMapping mapping)`

  Удаляет объект `TypeMapping` из реестра.

- `void clear()`

  Очищает реестр.

Класс `XMLType` определяет константы, представляющие XML- и SOAP-типы данных.

Пакеты *javax.xml.rpc.handler* и *javax.xml.rpc.handler.soap*

Пакеты `javax.xml.rpc.handler` и `javax.xml.rpc.handler.soap` позволяют создавать обработчики SOAP-сообщений как на стороне клиента, так и на стороне сервера. При этом обработчики группируются в цепочки, которые будут служить фильтрами для входящих и исходящих SOAP-сообщений. Цепочки обработчиков связываются с `Service`-объектом с помощью конфигурационного файла config.xml или программным образом, используя реестр обработчиков `HandlerRegistry`, или с конечной точкой Web-сервиса с помощью конфигурационных файлов config.xml и jaxrpc-ri.xml.

Каждый класс обработчика SOAP-сообщений должен реализовывать интерфейс `javax.xml.rpc.handler.Handler`, предоставляющий следующие методы.

- `boolean handleRequest(MessageContext context)`

  `boolean handleResponse(MessageContext context)`

  `boolean handleFault(MessageContext context)`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Обрабатывают запросы, ответы и SOAP-ошибки, используя объекты `javax.xml.rpc.handler.MessageContext`, представляющие контекст сообщения. Интерфейс `MessageContext` имеет методы:

- `void setProperty(java.lang.String name, java.lang.Object value)`
  `java.lang.Object getProperty(java.lang.String name)`
  `void removeProperty(java.lang.String name)`

  устанавливающие, возвращающие и удаляющие используемые свойства контекста;

- `boolean containsProperty(java.lang.String name)`
  `java.util.Iterator getPropertyNames()`

  проверяющие наличие свойства и возвращающие итератор свойств.

Сама же обработка SOAP-сообщений базируется на пакете `javax.xml.soap`. Для получения SOAP-сообщения используется интерфейс `javax.xml.rpc.handler.soap.SOAPMessageContext`, расширяющий интерфейс `MessageContext` и имеющий методы `javax.xml.soap.SOAPMessage getMessage()`, `void setMessage(javax.xml.soap.SOAPMessage message)` и `java.lang.String[] getRoles()`.

- `void init(HandlerInfo config)`

  Обеспечивает инициализацию обработчика, используя объект `javax.xml.rpc.handler.HandlerInfo`, представляющий информацию об обработчике. Класс `HandlerInfo` имеет следующие конструкторы и методы:

  - `public HandlerInfo()`, `public HandlerInfo(java.lang.Class handlerClass, java.util.Map config, javax.xml.namespace.QName[] headers)` — создают объекты `HandlerInfo`;
  - `public void setHandlerClass(java.lang.Class handlerClass)` и `public java.lang.Class getHandlerClass()` — первый метод устанавливает, второй метод возвращает класс обработчика;
  - `public void setHandlerConfig(java.util.Map config)` и `public java.util.Map getHandlerConfig()` — первый метод устанавливает, второй метод возвращает конфигурацию обработчика;
  - `public void setHeaders(javax.xml.namespace.QName[] headers)` и `public javax.xml.namespace.QName[] getHeaders()` — первый метод устанавливает, второй метод возвращает подлежащие обработке блоки SOAP-заголовка.

- `void destroy()`

  Позволяет очистить все ресурсы по завершении работы обработчика.

- `javax.xml.namespace.QName[] getHeaders()`

  Возвращает подлежащие обработке блоки SOAP-заголовка.

Как правило, пользовательский класс обработчика не реализует напрямую интерфейс `Handler`, а расширяет абстрактный класс `javax.xml.rpc.handler.GenericHandler`, который реализует интерфейс `Handler` и имеет методы:

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3


```
public boolean handleRequest(MessageContext context)
public boolean handleResponse(MessageContext context)
public boolean handleFault(MessageContext context)
public void init(HandlerInfo config)
public void destroy()
public abstract javax.xml.namespace.QName[] getHeaders()
```

Интерфейс `javax.xml.rpc.handler.HandlerRegistry` обеспечивает программную конфигурацию цепочек обработчиков на стороне клиента. Объект `HandlerRegistry` получается методом `javax.xml.rpc.Service.getHandlerRegistry()`, после чего его методы `java.util.List getHandlerChain(javax.xml.namespace.QName portName)` и `void setHandlerChain(javax.xml.namespace.QName portName, java.util.List chain)` позволяют возвратить и установить списки обработчиков в цепочке, состоящие из объектов `javax.xml.rpc.handler.HandlerInfo`.

При выполнении цепочки обработчиков управляются объектами `javax.xml.rpc.handler.HandlerChain` среды выполнения JAX-RPC, причем за создание класса реализации интерфейса `HandlerChain` отвечает сама JAX-RPC реализация.

Пакет *javax.xml.rpc.holders*

Пакет `javax.xml.rpc.holders` обеспечивает симуляцию способа pass-by-reference передачи аргументов при вызове удаленного метода Web-сервиса.

Существуют три вида JAX-RPC-параметров:

- `ParameterMode.IN` — исходящие параметры;
- `ParameterMode.OUT` — возвращаемые параметры;
- `ParameterMode.INOUT` — исходящие параметры, значения которых изменяются в результате запроса.

Значения параметров `IN` передаются способом pass-by-copy, определенным в языке Java. Значения же параметров `OUT` и `INOUT` должны передаваться способом pass-by-reference, которого нет в языке Java. Поэтому объекты классов, реализующих интерфейс `java.xml.rpc.holders.Holder`, служат обертками для значений параметров `OUT` и `INOUT` при выполнении запроса клиентом.

Пакет `javax.xml.rpc.holders` содержит классы-обертки, связывающие простые XML-типы данных с простыми Java-типами. Для сложных XML-типов JAX-RPC-реализация генерирует на стороне клиента `Holder`-классы с помощью инструмента wscompile из WSDL-описания Web-сервиса. Возможно также создание пользовательских `Holder`-классов, при этом они также должны реализовывать интерфейс `java.xml.rpc.holders.Holder` и своей структурой соответствовать классам пакета `javax.xml.rpc.holders`.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Пакет *javax.xml.rpc.server*

Пакет `javax.xml.rpc.server` обеспечивает управление жизненным циклом и контекстом конечной точки для JAX-RPC Web-сервиса, развернутого в Servlet-контейнере.

Интерфейс `javax.xml.rpc.server.ServiceLifecycle` реализуется классом реализации SEI-интерфейса и предоставляет следующие методы.

- `void init(javax.xml.rpc.server.ServletEndpointContext` *`context`*`)`

  Вызывается контейнером после создания экземпляра конечной точки для ее инициализации.

- `void destroy()`

  Обеспечивает освобождение всех ресурсов, связанных с работой конечной точки Web-сервиса.

Интерфейс `javax.xml.rpc.server.ServletEndpointContext` представляет контекст конечной точки и имеет следующие методы.

- `MessageContext getMessageContext()`

  Возвращает контекст SOAP-сообщения.

- `java.security.Principal getUserPrincipal()`

  Возвращает имя аутентифицированного пользователя, вызывающего конечную точку.

- `javax.servlet.http.HTTPSession getHttpSession()`

  Возвращает объект, представляющий текущую HTTP-сессию.

- `javax.servlet.ServletContext getServletContext()`

  Возвращает контекст сервлета, связанного с конечной точкой.

- `boolean isUserInRole(java.lang.String` *`role`*`)`

  Возвращает `true`, если аутентифицированный пользователь соответствует указанной роли.

JAX-RS API

Пакет *javax.ws.rs*

Аннотация `@Path(value=[URI-шаблон])` определяет URI-шаблон, обеспечивающий запрос к классу `RESTful` Web-сервиса (классу ресурса) или методу класса ресурса.

Аннотация `@ApplicationPath(value=[базовый URI-шаблон])` применяется к классу, расширяющему класс `javax.ws.rs.core.Application`, который обеспечивает развертывание JAX-RS-приложения на платформе Java EE. Аннотация `@ApplicationPath` указывает базовый URI-адрес для всех URI-адресов классов ресурсов JAX-RS-приложения.

Аннотации `@GET`, `@PUT`, `@POST`, `@DELETE`, `@HEAD` маркируют Java-методы класса ресурса, которые отвечают на соответствующие HTTP-методы запроса.

Аннотация `@MatrixParam(value=[имя параметра matrix])` маркирует параметры метода класса ресурса или его поля/свойства и извлекает значение параметра `matrix` из строки URI-запроса.

Аннотация `@QueryParam(value=[имя параметра query])` маркирует параметры метода класса ресурса или его поля/свойства и извлекает значение параметра `query` из строки URI-запроса.

Аннотация `@PathParam(value=[имя параметра])` маркирует параметры метода класса ресурса или его поля/свойства и извлекает значение указанного параметра из строки URI-запроса.

Аннотация `@CookieParam(value=[имя HTTP cookie])` маркирует параметры метода класса ресурса или его поля/свойства и извлекает значение HTTP cookie.

Аннотация `@HeaderParam(value=[имя HTTP-заголовка])` маркирует параметры метода класса ресурса или его поля/свойства и извлекает значение указанного HTTP-заголовка.

Аннотация `@FormParam(value=[имя параметра])` маркирует параметры метода класса ресурса или его поля/свойства и извлекает значение указанного параметра из тела URI-запроса.

Аннотация `@DefaultValue(value=[значение по умолчанию])` используется вместе с аннотациями `@PathParam`, `@QueryParam`, `@MatrixParam`, `@CookieParam`, `@FormParam` и `@HeaderParam` и определяет значения по умолчанию для параметров запроса на случай, если они не определены в строке запроса.

Аннотация `@Produces(value=[список MIME-типов])` ограничивает MIME-типы представления ресурса, отсылаемого клиенту в ответ на запрос, по умолчанию — любой тип.

Аннотация `@Consumes(value=[`*`список MIME-типов`*`])` ограничивает MIME-типы представления ресурса, посылаемого клиентом вместе с запросом, по умолчанию — любой тип.

Аннотация `@Encoded` используется вместе с аннотациями `@QueryParam`, `@MatrixParam`, `@PathParam`, `@FormParam` и отключает автоматическое декодирование значений параметров запроса. При формировании клиентом строки запроса параметры запроса кодируются в соответствии с URL-кодировкой. При извлечении параметров из строки запроса JAX-RS-реализация автоматически их декодирует. Если же есть необходимость работать с кодированными значениями параметров, тогда используется аннотация `@Encoded`.

Аннотация `@HttpMethod(value=[`*`имя HTTP-метода`*`])` позволяет создавать новые аннотации типа `@GET`, `@POST`, `@PUT`, `@DELETE` и `@HEAD`, связывая определенный HTTP-метод с аннотацией.

Аннотация `@OPTIONS` маркирует Java-метод класса ресурса, который отвечает на HTTP-метод `OPTIONS` запроса, предоставляя информацию об опциях соединения.

Класс `WebApplicationException` представляет исключение, автоматически конвертируемое JAX-RS-реализацией в HTTP-ответ клиенту. Класс `WebApplicationException` имеет следующие конструкторы:

```
public WebApplicationException()
public WebApplicationException(javax.ws.rs.core.Response response)
public WebApplicationException(int HTTP-status)
public WebApplicationException(javax.ws.rs.core.Response.Status status)
public WebApplicationException(java.lang.Throwable cause)
public WebApplicationException(java.lang.Throwable cause,
  javax.ws.rs.core.Response response)
public WebApplicationException(java.lang.Throwable cause, int HTTP-status)
public WebApplicationException(java.lang.Throwable cause,
  javax.ws.rs.core.Response.Status status)
```

Пакет *javax.ws.rs.core*

Аннотация `@Context` позволяет, дополнительно к аннотациям `@HeaderParam`, `@CookieParam`, `@MatrixParam`, `@QueryParam`, `@FormParam` или `@PathParam` пакета `javax.ws.rs`, вводить информацию контекста клиентского запроса или самого JAX-RS-приложения. Аннотация `@Context` дает возможность вводить такие объекты, как `Application`, `UriInfo`, `Request`, `HttpHeaders`, `SecurityContext`, `Providers`, `ServletConfig`, `ServletContext`, `HttpServletRequest` и `HttpServletResponse`.

Объект `HttpHeaders`, введенный с помощью аннотации `@Context`, реализует интерфейс `HttpHeaders`, обеспечивающий информацию о HTTP-заголовках запроса. Интерфейс `HttpHeaders` имеет следующие поля:

- `static final java.lang.String ACCEPT` — заголовок `Accept` указывает MIME-типы, допустимые для ответного сообщения;

- `static final java.lang.String ACCEPT_CHARSET` — заголовок `Accept-Charset` указывает допустимую символьную кодировку для ответного сообщения;
- `static final java.lang.String ACCEPT_ENCODING` — заголовок `Accept-Encoding` указывает допустимый стиль кодирования содержимого ответа;
- `static final java.lang.String ACCEPT_LANGUAGE` — заголовок `Accept-Language` указывает допустимые языки ответа;
- `static final java.lang.String AUTHORIZATION` — заголовок `Authorization` содержит аутентификационную информацию клиента;
- `static final java.lang.String CACHE_CONTROL` — заголовок `Cache-Control` содержит директивы относительно кэширования;
- `static final java.lang.String CONTENT_ENCODING` — заголовок `Content-Encoding` указывает стиль кодирования содержимого запроса;
- `static final java.lang.String CONTENT_LANGUAGE` — заголовок `Content-Language` указывает локализацию запроса;
- `static final java.lang.String CONTENT_LENGTH` — заголовок `Content-Length` указывает размер содержимого запроса;
- `static final java.lang.String CONTENT_LOCATION` — заголовок `Content-Location` содержит ссылку на ресурс в запросе;
- `static final java.lang.String CONTENT_TYPE` — заголовок `Content-Type` указывает MIME-тип запроса;
- `static final java.lang.String DATE` — заголовок `Date` указывает дату и время запроса;
- `static final java.lang.String ETAG` — заголовок `Etag` указывает идентификатор ресурса для кэширования ответа;
- `static final java.lang.String EXPIRES` — заголовок `Expires` указывает время действия ответа при кэшировании;
- `static final java.lang.String HOST` — заголовок `Host` указывает хост и порт запрашиваемого ресурса;
- `static final java.lang.String IF_MATCH` — заголовок `If-Match` содержит идентификатор кэшированного ресурса для указания того, что метод PUT не может быть применен к ресурсу, если его кэш уже не является его представлением;
- `static final java.lang.String IF_MODIFIED_SINCE` — заголовок `If-Modified-Since` указывает время, начиная с которого запрашиваемый ресурс не модифицировался; если ресурс модифицировался, тогда ответ содержит новое представление ресурса;
- `static final java.lang.String IF_NONE_MATCH` — заголовок `If-None-Match` содержит идентификатор кэшированного ресурса для его сравнения с идентификатором текущей версии ресурса; если идентификаторы не совпадают, тогда ответ содержит новое представление ресурса;

- `static final java.lang.String IF_UNMODIFIED_SINCE` — заголовок `If-Unmodified-Since` указывает время, начиная с которого запрашиваемый ресурс не модифицировался; если ресурс не модифицировался, тогда посылается ответ;
- `static final java.lang.String LAST_MODIFIED` — заголовок `Last-Modified` указывает время последней модификации ресурса;
- `static final java.lang.String LOCATION` — заголовок `Location` содержит адрес для перенаправления запроса;
- `static final java.lang.String USER_AGENT` — заголовок `User-Agent` содержит идентификатор клиентского приложения;
- `static final java.lang.String VARY` — заголовок `Vary` указывает кэшу названия заголовков запроса, при изменении которых ответ может содержать другое представление ресурса;
- `static final java.lang.String WWW_AUTHENTICATE` — заголовок `WWW-Authenticate` указывает требуемую схему аутентификации для доступа к ресурсу;
- `static final java.lang.String COOKIE` — заголовок `Cookie` содержит cookie запроса;
- `static final java.lang.String SET_COOKIE` — заголовок `Set-Cookie` устанавливает cookie.

Далее перечислены методы интерфейса `HttpHeaders`, обеспечивающие HTTP-заголовки запроса и ответа.

- `java.util.List<java.lang.String> getRequestHeader(java.lang.String name)`

  Возвращает заголовок запроса по его имени.

- `MultivaluedMap<java.lang.String,java.lang.String> getRequestHeaders()`

  Возвращает заголовки запроса как объект `javax.ws.rs.core.MultivaluedMap<K,V>`, расширяющий интерфейс `java.util.Map<K,java.util.List<V>>` методами `void putSingle(K key, V value)`, `void add(K key, V value)`, `V getFirst(K key)`.

- `java.util.List<MediaType> getAcceptableMediaTypes()`

  Возвращает список MIME-типов, допустимых для ответа.

- `java.util.List<java.util.Locale> getAcceptableLanguages()`

  Возвращает список языков, допустимых для ответа.

- `MediaType getMediaType()`

  Возвращает MIME-тип запроса в виде объекта `javax.ws.rs.core.MediaType`.

- `java.util.Locale getLanguage()`

  Возвращает язык запроса.

- `java.util.Map<java.lang.String,Cookie> getCookies()`

  Возвращает cookies запроса.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Класс `MediaType` представляет MIME-тип и имеет следующие поля:

- `public static final java.lang.String MEDIA_TYPE_WILDCARD` — MIME-тип `*`;
- `public static final java.lang.String WILDCARD`, `public static final MediaType WILDCARD_TYPE` — MIME-тип `*/*`;
- `public static final java.lang.String APPLICATION_XML`, `public static final MediaType APPLICATION_XML_TYPE` — MIME-тип `application/xml`;
- `public static final java.lang.String APPLICATION_ATOM_XML`, `public static final MediaType APPLICATION_ATOM_XML_TYPE` — MIME-тип `application/atom+xml`;
- `public static final java.lang.String APPLICATION_XHTML_XML`, `public static final MediaType APPLICATION_XHTML_XML_TYPE` — MIME-тип `application/xhtml+xml`;
- `public static final java.lang.String APPLICATION_SVG_XML`, `public static final MediaType APPLICATION_SVG_XML_TYPE` — MIME-тип `application/svg+xml`;
- `public static final java.lang.String APPLICATION_JSON`, `public static final MediaType APPLICATION_JSON_TYPE` — MIME-тип `application/json`;
- `public static final java.lang.String APPLICATION_FORM_URLENCODED`, `public static final MediaType APPLICATION_FORM_URLENCODED_TYPE` — MIME-тип `application/x-www-form-urlencoded`;
- `public static final java.lang.String MULTIPART_FORM_DATA`, `public static final MediaType MULTIPART_FORM_DATA_TYPE` — MIME-тип `multipart/form-data`;
- `public static final java.lang.String APPLICATION_OCTET_STREAM`, `public static final MediaType APPLICATION_OCTET_STREAM_TYPE` — MIME-тип `application/octet-stream`;
- `public static final java.lang.String TEXT_PLAIN`, `public static final MediaType TEXT_PLAIN_TYPE` — MIME-тип `text/plain`;
- `public static final java.lang.String TEXT_XML`, `public static final MediaType TEXT_XML_TYPE` — MIME-тип `text/xml`;
- `public static final java.lang.String TEXT_HTML`, `public static final MediaType TEXT_HTML_TYPE` — MIME-тип `text/html`.

Далее приведены конструкторы и методы класса `MediaType`.

- `public MediaType(java.lang.String type, java.lang.String subtype, java.util.Map<java.lang.String,java.lang.String> parameters)`

  `public MediaType(java.lang.String type, java.lang.String subtype)`

  `public MediaType()`

  Создают объект `MediaType`.

- `public static MediaType valueOf(java.lang.String type)`

  Создает объект `MediaType` из MIME-типа.

- `public java.lang.String getType()`

  Возвращает MIME-тип.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public boolean isWildcardType()`

  Если `true`, тогда MIME-тип `*`.

- `public java.lang.String getSubtype()`

  Возвращает подтип.

- `public boolean isWildcardSubtype()`

  Если `true`, тогда подтип `*`.

- `public java.util.Map<java.lang.String,java.lang.String> getParameters()`

  Возвращает параметры MIME-типа.

- `public boolean isCompatible(MediaType other)`

  Возвращает `true`, если данный MIME-тип совместим с указанным.

- `public boolean equals(java.lang.Object obj)`

  Возвращает `true`, если два MIME-типа одинаковы.

- `public int hashCode()`

  Генерирует хэш из MIME-типа.

- `public java.lang.String toString()`

  Конвертирует MIME-тип в строку для HTTP-заголовка.

Класс `Application` обеспечивает развертывание JAX-RS приложения с помощью своего класса реализации. Класс, расширяющий класс `Application`, переопределяет его методы, перечисленные далее.

- `public java.util.Set<java.lang.Class<?>> getClasses()`

  Возвращает набор классов, предназначенных для развертывания.

- `public java.util.Set<java.lang.Object> getSingletons()`

  Возвращает набор экземпляров классов для развертывания.

Метод `getClasses()` контролирует классы приложения, а метод `getSingletons()` дает возможность контролировать создание объектов при развертывании приложения.

Объект `Request`, введенный с помощью аннотации `@Context`, реализует интерфейс `Request`, помогающий обрабатывать запрос клиента. Интерфейс `Request` имеет перечисленные далее методы.

- `java.lang.String getMethod()`

  Возвращает HTTP-метод запроса.

- `Variant selectVariant(java.util.List<Variant> variants)`

  Возвращает объект `javax.ws.rs.core.Variant`, как наиболее подходящий вариант представления ресурса для запроса. Наиболее подходящий вариант представления ресурса выбирается из списка вариантов, который создается с помощью класса `javax.ws.rs.core.Variant.VariantListBuilder`, имеющего следующие методы:

- `public static Variant.VariantListBuilder newInstance()` — создает объект `Variant.VariantListBuilder`;
- `public abstract Variant.VariantListBuilder languages(java.util.Locale... languages)` — устанавливает допустимые языки для представлений ресурса;
- `public abstract Variant.VariantListBuilder encodings(java.lang.String... encodings)` — устанавливает допустимые стили кодирования для представлений ресурса;
- `public abstract Variant.VariantListBuilder mediaTypes(MediaType... mediaTypes)` — устанавливает допустимые MIME-типы для представлений ресурса;
- `public abstract java.util.List<Variant> build()` — создает набор вариантов после установки языков, стилей кодирования и MIME-типов;
- `public abstract Variant.VariantListBuilder add()` — позволяет разграничить и дифференцировать создаваемые наборы вариантов, например, `Variant.VariantBuilder.newInstance. languages().encodings().mediaTypes().add().languages().encodings().mediaTypes().build()`.

❒ `Response.ResponseBuilder evaluatePreconditions(EntityTag eTag)`

`Response.ResponseBuilder evaluatePreconditions(java.util.Date lastModified)`

`Response.ResponseBuilder evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)`

Сравнивают `ETag` текущего состояния ресурса и дату его модификации с заголовком `If-Modified-Since` или/и `If-None-Match` запроса и при совпадении возвращают `null`. Если же условия не совпадают, тогда возвращается объект `javax.ws.rs.core.Response.ResponseBuilder` с новыми заголовками `ETag` или/и `Last-Modified`.

❒ `Response.ResponseBuilder evaluatePreconditions()`

Обеспечивает проверку заголовков запроса `If-Match: *` и `If-None-Match: *`.

Класс `Variant`, представляющий вариант представления ресурса, имеет следующие конструкторы и методы.

❒ `public Variant(MediaType mediaType, java.util.Locale language, java.lang.String encoding)`

Создает объект `Variant`.

❒ `public java.util.Locale getLanguage()`

Возвращает язык варианта.

❒ `public MediaType getMediaType()`

Возвращает MIME-тип варианта.

❒ `public java.lang.String getEncoding()`

Возвращает стили кодирования варианта.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public static Variant.VariantListBuilder mediaTypes(MediaType... mediaTypes)`

  `public static Variant.VariantListBuilder languages(java.util.Locale... languages)`

  `public static Variant.VariantListBuilder encodings(java.lang.String... encodings)`

  Создают объекты `Variant.VariantListBuilder`, обеспечивающие создание списка вариантов.

- `public int hashCode()`

  Создает хэш варианта.

- `public boolean equals(java.lang.Object obj)`

  Сравнивает два варианта.

- `public java.lang.String toString()`

  Конвертирует вариант в строку.

Класс `EntityTag` представляет HTTP-заголовок `ETag` и имеет следующие конструкторы и методы.

- `public EntityTag(java.lang.String valueofETag)`

  `public EntityTag(java.lang.String valueofETag, boolean weak)`

  Создают объект `EntityTag`, при этом если `weak` равно `false`, тогда текущие версии ресурса должны быть идентичны byte-for-byte.

- `public static EntityTag valueOf(java.lang.String valueofETag)`

  Создает объект `EntityTag` из строки.

- `public boolean isWeak()`

  Если `true`, тогда используется `WETag`.

- `public java.lang.String getValue()`

  Возвращает значение заголовка `ETag`.

- `public boolean equals(java.lang.Object obj)`

  Сравнивает два заголовка `ETag`.

- `public int hashCode()`

  Создает хэш заголовка.

- `public java.lang.String toString()`

  Конвертирует заголовок в строку.

Объект `SecurityContext`, введенный с помощью аннотации `@Context`, реализует интерфейс `SecurityContext`, обеспечивающий аутентификационную информацию запроса. Интерфейс `SecurityContext` имеет следующие поля и методы:

- `static final java.lang.String BASIC_AUTH` — базовая аутентификация клиента с помощью логина/пароля;
- `static final java.lang.String CLIENT_CERT_AUTH` — аутентификация с помощью сертификата;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `static final java.lang.String DIGEST_AUTH` — аутентификация с помощью зашифрованного пароля;
- `static final java.lang.String FORM_AUTH` — аутентификация с помощью пользовательского интерфейса;
- `java.security.Principal getUserPrincipal()` — возвращает имя клиента;
- `boolean isUserInRole(java.lang.String role)` — возвращает `true`, если клиент соответствует роли;
- `boolean isSecure()` — возвращает `true`, если запрос сделан по HTTPS-каналу;
- `java.lang.String getAuthenticationScheme()` — возвращает тип аутентификации клиента `BASIC_AUTH`, `FORM_AUTH`, `CLIENT_CERT_AUTH`, `DIGEST_AUTH`.

Объект `UriInfo`, введенный с помощью аннотации `@Context`, реализует интерфейс `UriInfo`, обеспечивающий информацию о URI запроса. Интерфейс `UriInfo` имеет следующие методы.

- `java.lang.String getPath()`
  `java.lang.String getPath(boolean decode)`

  Возвращают URI-путь запроса относительно базового URI. Если значение `decode` равно `false`, тогда строка URI не декодируется.
- `java.util.List<PathSegment> getPathSegments()`
  `java.util.List<PathSegment> getPathSegments(boolean decode)`

  Возвращают URI-путь запроса относительно базового URI в виде списка объектов `javax.ws.rs.core.PathSegment`, представляющих сегменты URI-пути и связанных с ними параметров. Интерфейс `PathSegment` имеет методы `java.lang.String getPath()` и `MultivaluedMap<java.lang.String,java.lang.String> getMatrixParameters()`.
- `java.net.URI getRequestUri()`

  Возвращает абсолютный URI-путь запроса.
- `UriBuilder getRequestUriBuilder()`

  Возвращает абсолютный URI-путь запроса в виде объекта `javax.ws.rs.core.UriBuilder`.
- `java.net.URI getAbsolutePath()`

  Возвращает абсолютный URI-путь запроса без параметров.
- `UriBuilder getAbsolutePathBuilder()`

  Возвращает абсолютный URI-путь запроса без параметров в виде объекта `javax.ws.rs.core.UriBuilder`.
- `java.net.URI getBaseUri()`

  Возвращает базовый URI-путь приложения.
- `UriBuilder getBaseUriBuilder()`

  Возвращает базовый URI-путь приложения в виде объекта `javax.ws.rs.core.UriBuilder`.

- `MultivaluedMap<java.lang.String,java.lang.String> getPathParameters()`

  `MultivaluedMap<java.lang.String,java.lang.String> getPathParameters(boolean decode)`

  `MultivaluedMap<java.lang.String,java.lang.String> getQueryParameters()`

  `MultivaluedMap<java.lang.String,java.lang.String> getQueryParameters(boolean decode)`

  Возвращают набор параметров запроса.

- `java.util.List<java.lang.String> getMatchedURIs()`

  `java.util.List<java.lang.String> getMatchedURIs(boolean decode)`

  Возвращают список относительных путей ресурсов приложения.

- `java.util.List<java.lang.Object> getMatchedResources()`

  Возвращает список ресурсов.

Класс `UriBuilder` позволяет создавать URI с помощью следующих методов.

- `public static UriBuilder fromUri(java.net.URI uri)`

  `public static UriBuilder fromUri(java.lang.String uri)`

  `public static UriBuilder fromPath(java.lang.String path)`

  `public static UriBuilder fromResource(java.lang.Class<?> resource)`

  Создают объекты `UriBuilder` из адреса, пути и ресурса.

- `public abstract UriBuilder clone()`

  Создает копию объекта `UriBuilder`.

- `public abstract UriBuilder uri(java.net.URI uri)`

  Изменяет объект `UriBuilder`.

- `public abstract UriBuilder scheme(java.lang.String scheme)`

  Устанавливает URI-схему, например, http, file или др.

- `public abstract UriBuilder schemeSpecificPart(java.lang.String ssp)`

  Устанавливает часть URI, следующую за URI-схемой.

- `public abstract UriBuilder userInfo(java.lang.String ui)`

  Устанавливает часть URI, содержащую информацию о клиенте.

- `public abstract UriBuilder host(java.lang.String host)`

  Устанавливает URI-хост.

- `public abstract UriBuilder port(int port)`

  Устанавливает URI-порт.

- `public abstract UriBuilder replacePath(java.lang.String path)`

  Устанавливает URI-путь.

- `public abstract UriBuilder path(java.lang.String path)`

  `public abstract UriBuilder path(java.lang.Class resource)`

```
public abstract UriBuilder path(java.lang.Class resource,
  java.lang.String method)
public abstract UriBuilder path(java.lang.reflect.Method method)
```

Добавляют путь, определенный аннотацией `@Path`, к существующему.

- `public abstract UriBuilder segment(java.lang.String... segments)`

  Добавляет сегмент к пути.

- `public abstract UriBuilder replaceMatrix(java.lang.String matrix)`

  Устанавливает `matrix`-параметры.

- `public abstract UriBuilder matrixParam(java.lang.String name, java.lang.Object... values)`

  Добавляет `matrix`-параметры к существующим.

- `public abstract UriBuilder replaceMatrixParam(java.lang.String name, java.lang.Object... values)`

  Заменяет значения существующих `matrix`-параметров.

- `public abstract UriBuilder replaceQuery(java.lang.String query)`

  Устанавливает `query`-параметры.

- `public abstract UriBuilder queryParam(java.lang.String name, java.lang.Object... values)`

  Добавляет `query`-параметры к существующим.

- `public abstract UriBuilder replaceQueryParam(java.lang.String name, java.lang.Object... values)`

  Заменяет значения существующих `query`-параметров.

- `public abstract UriBuilder fragment(java.lang.String fragment)`

  Устанавливает URI-фрагмент.

- `public abstract java.net.URI buildFromMap (java.util.Map<java.lang.String,? extends java.lang.Object> values)`

  `public abstract java.net.URI buildFromEncodedMap (java.util.Map<java.lang.String,? extends java.lang.Object> values)`

  `public abstract java.net.URI build(java.lang.Object... values)`

  `public abstract java.net.URI buildFromEncoded (java.lang.Object... values)`

  Создают URI с параметрами.

Класс `Response` представляет ответ, возвращаемый методом ресурса, и имеет следующие методы.

- `public abstract java.lang.Object getEntity()`

  Возвращает тело ответа.

- `public abstract int getStatus()`

  Возвращает HTTP-код ответа.

- `public abstract MultivaluedMap<java.lang.String,java.lang.Object> getMetadata()`

  Возвращает HTTP-заголовки, добавляемые к телу ответа.

- `public static Response.ResponseBuilder fromResponse(Response response)`

  `public static Response.ResponseBuilder status(Response.StatusType status)`

  `public static Response.ResponseBuilder status(Response.Status status)`

  `public static Response.ResponseBuilder status(int status)`

  `public static Response.ResponseBuilder ok()`

  `public static Response.ResponseBuilder ok(java.lang.Object entity)`

  `public static Response.ResponseBuilder ok(java.lang.Object entity, MediaType type)`

  `public static Response.ResponseBuilder ok(java.lang.Object entity, java.lang.String type)`

  `public static Response.ResponseBuilder ok(java.lang.Object entity, Variant variant)`

  `public static Response.ResponseBuilder serverError()`

  `public static Response.ResponseBuilder created(java.net.URI location)`

  `public static Response.ResponseBuilder noContent()`

  `public static Response.ResponseBuilder notModified()`

  `public static Response.ResponseBuilder notModified(EntityTag tag)`

  `public static Response.ResponseBuilder notModified(java.lang.String tag)`

  `public static Response.ResponseBuilder seeOther(java.net.URI location)`

  `public static Response.ResponseBuilder temporaryRedirect(java.net.URI location)`

  `public static Response.ResponseBuilder notAcceptable(java.util.List<Variant> variants)`

  Создают новый объект `Response.ResponseBuilder` из копии ответа с различным HTTP-кодом, MIME-типом и вариантами представления, с телом ответа и URI ресурса, а также для пустого ответа.

Класс `Response.ResponseBuilder` позволяет создавать объекты `Response`, содержащие различные метаданные, которые добавляются к телу ответа, и имеет следующие методы.

- `public abstract Response build()`

  Создает новый объект `Response`.

- `public abstract Response.ResponseBuilder clone()`

  Создает копию объекта `Response.ResponseBuilder`.

- `public abstract Response.ResponseBuilder status(int status)`

  `public Response.ResponseBuilder status(Response.StatusType status)`

  `public Response.ResponseBuilder status(Response.Status status)`

  Устанавливают HTTP-код ответа.

- `public abstract Response.ResponseBuilder entity(java.lang.Object entity)`

  Устанавливает тело ответа.

- `public abstract Response.ResponseBuilder type(MediaType type)`

  `public abstract Response.ResponseBuilder type(java.lang.String type)`

  Устанавливают MIME-тип ответа.

- `public abstract Response.ResponseBuilder variant(Variant variant)`

  `public abstract Response.ResponseBuilder variants(java.util.List<Variant> variants)`

  Устанавливают варианты представления.

- `public abstract Response.ResponseBuilder language(java.lang.String language)`

  `public abstract Response.ResponseBuilder language(java.util.Locale language)`

  Устанавливают язык ответа.

- `public abstract Response.ResponseBuilder location(java.net.URI location)`

  `public abstract Response.ResponseBuilder contentLocation(java.net.URI location)`

  Устанавливают URI ответа.

- `public abstract Response.ResponseBuilder tag(EntityTag tag)`

  `public abstract Response.ResponseBuilder tag(java.lang.String tag)`

  Устанавливают заголовок `ETag` ответа.

- `public abstract Response.ResponseBuilder lastModified(java.util.Date lastModified)`

  Устанавливает заголовок `Last-Modified` ответа.

- `public abstract Response.ResponseBuilder cacheControl(CacheControl cacheControl)`

  Устанавливает заголовок `Cache-Control` ответа.

- `public abstract Response.ResponseBuilder expires(java.util.Date expires)`

  Устанавливает заголовок `Expires` ответа.

- `public abstract Response.ResponseBuilder header(java.lang.String name, java.lang.Object value)`

  Устанавливает заголовки ответа.

- `public abstract Response.ResponseBuilder cookie(NewCookie... cookies)`

  Устанавливает cookies ответа.

Реализация интерфейса `Response.StatusType` позволяет расширить стандартные HTTP-коды статуса ответа. Интерфейс `Response.StatusType` обеспечивает информацию о статусе ответа с помощью перечисленных далее методов.

- `int getStatusCode()`

  Возвращает код статуса.

- `java.lang.String getReasonPhrase()`

  Возвращает описание статуса.

- `Response.Status.Family getFamily()`

  Возвращает объект перечисления `javax.ws.rs.core.Response.Status.Family` кода статуса. Перечисление `Response.Status.Family` имеет константы `INFORMATIONAL`, `SUCCESSFUL`, `REDIRECTION`, `CLIENT_ERROR`, `SERVER_ERROR`, `OTHER`.

Перечисление `Response.Status` HTTP-кодов имеет следующие константы:

- `OK` — 200 OK;
- `CREATED` — 201 Created;
- `ACCEPTED` — 202 Accepted;
- `NO_CONTENT` — 204 No Content;
- `MOVED_PERMANENTLY` — 301 Moved Permanently;
- `SEE_OTHER` — 303 See Other;
- `NOT_MODIFIED` — 304 Not Modified;
- `TEMPORARY_REDIRECT` — 307 Temporary Redirect;
- `BAD_REQUEST` — 400 Bad Request;
- `UNAUTHORIZED` — 401 Unauthorized;
- `FORBIDDEN` — 403 Forbidden;
- `NOT_FOUND` — 404 Not Found;
- `NOT_ACCEPTABLE` — 406 Not Acceptable;
- `CONFLICT` — 409 Conflict;
- `GONE` — 410 Gone;
- `PRECONDITION_FAILED` — 412 Precondition Failed;
- `UNSUPPORTED_MEDIA_TYPE` — 415 Unsupported Media Type;
- `INTERNAL_SERVER_ERROR` — 500 Internal Server Error;
- `SERVICE_UNAVAILABLE` — 503 Service Unavailable.

Класс `GenericEntity<T>` представляет тело ответа общего Java-типа и имеет следующие конструкторы и методы.

- `public GenericEntity(T entity, java.lang.reflect.Type genericType)`

  Создает новый объект `GenericEntity`.

- `public final java.lang.Class<?> getRawType()`

  Возвращает Java-тип тела ответа.

- `public final java.lang.reflect.Type getType()`

  Возвращает общий Java-тип тела ответа.

- `public final T getEntity()`

  Возвращает тело ответа.

Метод ресурса может также возвращать объект, реализующий интерфейс `StreamingOutput`, который имеет метод `void write(java.io.OutputStream output)` для создания тела ответа.

Класс `CacheControl` обеспечивает HTTP-заголовок `Cache-Control` и имеет следующие конструкторы и методы.

- `public CacheControl()`

  Создает новый объект `CacheControl`.

- `public static CacheControl valueOf(java.lang.String value)`

  Создает новый объект `CacheControl` из строки.

- `public boolean isMustRevalidate()`

  Если `true`, тогда заголовок содержит директиву `must-revalidate`.

- `public void setMustRevalidate(boolean mustRevalidate)`

  Устанавливает для заголовка директиву `must-revalidate`.

- `public boolean isProxyRevalidate()`

  Если `true`, тогда заголовок содержит директиву `proxy-revalidate`.

- `public void setProxyRevalidate(boolean proxyRevalidate)`

  Устанавливает для заголовка директиву `proxy-revalidate`.

- `public int getMaxAge()`
  `public void setMaxAge(int maxAge)`

  Первый метод возвращает, второй метод устанавливает директиву `max-age`.

- `public int getSMaxAge()`
  `public void setSMaxAge(int sMaxAge)`

  Первый метод возвращает, второй метод устанавливает директиву `s-maxage`.

- `public java.util.List<java.lang.String> getNoCacheFields()`
  `public void setNoCache(boolean noCache)`

  Первый метод возвращает, второй метод устанавливает директиву `no-cache`.

- `public boolean isNoCache()`

  Если `true`, тогда заголовок содержит директиву `no-cache`.

- `public boolean isPrivate()`

  Если `true`, тогда заголовок содержит директиву `private`.

- `public java.util.List<java.lang.String> getPrivateFields()`
  `public void setPrivate(boolean _private)`

  Первый метод возвращает, второй метод устанавливает директиву `private`.

- `public boolean isNoTransform()`

  Если `true`, тогда заголовок содержит директиву `no-transform`.

- `public void setNoTransform(boolean noTransform)`

  Устанавливает директиву `no-transform`.

- `public boolean isNoStore()`

  Если `true`, тогда заголовок содержит директиву `no-store`.

- `public void setNoStore(boolean noStore)`

  Устанавливает директиву `no-store`.

- `public java.util.Map<java.lang.String, java.lang.String> getCacheExtension()`

  Возвращает расширения директив.

- `public java.lang.String toString()`

  Конвертирует объект в строку.

- `public int hashCode()`

  Создает хэш объекта.

- `public boolean equals(java.lang.Object obj)`

  Сравнивает два объекта.

Класс `Cookie` представляет HTTP-запроса cookie и имеет следующие поля, конструкторы и методы.

- `public static final int DEFAULT_VERSION` — версия, по умолчанию 1 (RFC 2109).

- `public Cookie(java.lang.String name, java.lang.String value, java.lang.String path, java.lang.String domain, int version)`

  `public Cookie(java.lang.String name, java.lang.String value, java.lang.String path, java.lang.String domain)`

  `public Cookie(java.lang.String name, java.lang.String value)`

  Создают новый объект `Cookie`.

- `public static Cookie valueOf(java.lang.String value)`

  Создает объект `Cookie` из строки.

- `public java.lang.String getName()`

  Возвращает имя cookie.

- `public java.lang.String getValue()`

  Возвращает значение cookie.

- `public int getVersion()`

  Возвращает версию cookie.

- `public java.lang.String getDomain()`

  Возвращает домен cookie.

- `public java.lang.String getPath()`

  Возвращает путь cookie.

- `public java.lang.String toString()`

  Конвертирует объект в строку.

- `public int hashCode()`

  Создает хэш объекта.

- `public boolean equals(java.lang.Object obj)`

  Сравнивает два объекта.

Класс `NewCookie` расширяет класс `Cookie`, представляет HTTP-ответа cookie и имеет дополнительно следующие поля, конструкторы и методы:

- `public static final int DEFAULT_MAX_AGE` — время действия cookie.
- `public NewCookie(java.lang.String name, java.lang.String value)`

  `public NewCookie(java.lang.String name, java.lang.String value, java.lang.String path, java.lang.String domain, java.lang.String comment, int maxAge, boolean secure)`

  `public NewCookie(java.lang.String name, java.lang.String value, java.lang.String path, java.lang.String domain, int version, java.lang.String comment, int maxAge, boolean secure)`

  `public NewCookie(Cookie cookie)`

  `public NewCookie(Cookie cookie, java.lang.String comment, int maxAge, boolean secure)`

  Создают новый объект `NewCookie`.
- `public static NewCookie valueOf(java.lang.String value)`

  Создает новый объект `NewCookie` из строки.
- `public java.lang.String getComment()`

  Возвращает комментарии cookie.
- `public int getMaxAge()`

  Возвращает время действия cookie.
- `public boolean isSecure()`

  Если `true`, тогда cookie должны передаваться по защищенному каналу.
- `public Cookie toCookie()`

  Конвертирует в объект `Cookie` для сравнения.
- `public java.lang.String toString()`

  Конвертирует объект в строку.
- `public int hashCode()`

  Создает хэш объекта.
- `public boolean equals(java.lang.Object obj)`

  Сравнивает два объекта.

Пакет *javax.ws.rs.ext*

Пакет `javax.ws.rs.ext` обеспечивает связывание между представлениями ресурса и Java-типами и позволяет создать собственные классы, расширяющие среду выполнения JAX-RS.

Классы поставщиков, обеспечивающие связывание с Java-типами, маркируются аннотацией `@Provider` и реализуют один из интерфейсов `MessageBodyReader`, `MessageBodyWriter`, `ContextResolver` и `ExceptionMapper` в зависимости от их предназначения.

Классы поставщиков, реализующие интерфейс `MessageBodyReader` или `MessageBodyWriter`, называются *Entity Providers*, т. к. обеспечивают чтение и запись тела сообщения.

Интерфейс `MessageBodyReader<T>` демаршализует тело запроса в Java-тип и имеет следующие методы:

- `boolean isReadable(java.lang.Class<?> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType)`

  Вызывается средой выполнения JAX-RS для проверки, способен ли данный `MessageBodyReader`-класс осуществить демаршализацию тела запроса. Проверка соответствия передаваемого в качестве аргумента Java-типа может быть осуществлена с помощью различных методов сравнения, в том числе метода `type.isAnnotationPresent()`, проверяющего наличие определенных аннотаций в классе Java-типа.

- `TreadFrom(java.lang.Class<T> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType, MultivaluedMap<java.lang.String,java.lang.String> httpHeaders, java.io.InputStream entityStream)`

  Создает Java-объект из входящего потока тела запроса.

Интерфейс `MessageBodyWriter<T>` маршализует Java-тип в исходящий поток ответа и имеет следующие методы:

- `boolean isWriteable(java.lang.Class<?> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType)`

  Вызывается средой выполнения JAX-RS для проверки того, способен ли данный `MessageBodyWriter`-класс осуществить маршализацию Java-объекта.

- `long getSize(T t, java.lang.Class<?> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType)`

  Вызывается перед маршализацией для установки длины сериализованного объекта. Если длина не определяется, метод возвращает –1.

- `void writeTo(T t, java.lang.Class<?> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType, MultivaluedMap<java.lang.String,java.lang.Object> httpHeaders, java.io.OutputStream entityStream)`

  Записывает HTTP-ответ.

Для ограничения MIME-типов в классах Entity Providers применяются аннотации `@Consumes` и `@Produces`.

JAX-RS-реализация обеспечивает по умолчанию классы Entity Providers для маршализации/демаршализации в следующие Java-типы:

- `byte[]` — MIME-тип `*/*`;
- `java.lang.String` — MIME-тип `*/*`;
- `java.io.InputStream` — MIME-тип `*/*`;
- `java.io.Reader` — MIME-тип `*/*`;
- `java.io.File` — MIME-тип `*/*`;
- `javax.activation.DataSource` — MIME-тип `*/*`;
- `javax.xml.transform.Source` — XML-тип `text/xml`, `application/xml` и `application/*+xml`;
- `javax.xml.bind.JAXBElement` и JAXB-классы приложения — XML-тип `text/xml`, `application/xml` и `application/*+xml`;
- `MultivaluedMap<String,String>` — `application/x-www-form-urlencoded`;
- `StreamingOutput` — MIME-тип `*/*`.

Таким образом, объекты вышеперечисленных типов могут без дополнительного кодирования возвращаться и потребляться методами `@GET`, `@PUT` и `@POST` ресурса. Если же методу ресурса необходимо возвращать объект другого Java-типа, тогда метод ресурса возвращает объект `javax.ws.rs.core.Response`, который создается с помощью класса `javax.ws.rs.core.Response.ResponseBuilder` и класса `javax.ws.rs.core.GenericEntity<T>`. Или же используется `MessageBodyWriter`-класс, позволяющий возвращать пользовательский Java-тип напрямую или как часть объекта `javax.ws.rs.core.Response`. Также, если методу ресурса `@PUT` необходимо использовать в качестве параметра нестандартный Java-объект, необходимо создать `MessageBodyReader`-класс, обеспечивающий демаршализацию запроса в Java-объект.

Из перечня стандартных для маршализации/демаршализации Java-типов видно, что методы ресурса могут возвращать и потреблять XML-данные, представленные JAXB-классами приложения. JAXB-классы приложения создаются с помощью JAXB-аннотаций `@XmlRootElement`, `@XmlElement` и т. д., а среда выполнения JAX-RS использует стандартную JAXBContext-реализацию для маршализации/демаршализации их экземпляров, отвечая за создание и инициализацию объекта `JAXBContext`. Если же приложению требуется самому конфигурировать объект `JAXBContext`, необходимо создать класс поставщика, маркированный аннотацией `@Provider` и реализующий интерфейс `ContextResolver<T>`, который имеет единственный метод `T getContext(java.lang.Class<?> type)`, вызываемый средой выполнения JAX-RS для получения объекта `JAXBContext`. Ограничить MIME-типы `ContextResolver`-класса можно с помощью аннотации `@Produces`.

Выполнение методов ресурса или поставщика может вызывать исключительную ситуацию. При этом среда выполнения JAX-RS связывает ошибку с подходящим HTTP-ответом. Приложение может само определять такого рода связывания с помощью класса поставщика, маркированного аннотацией `@Provider` и реализующего

интерфейс `ExceptionMapper<E extends java.lang.Throwable>`, который имеет единственный метод `Response toResponse(E exception)`, связывающий Java-исключение с объектом `Response`.

Аннотация `@Context` может вводить, помимо объектов `Application`, `UriInfo`, `Request`, `HttpHeaders` и `SecurityContext`, объект `Providers`, реализующий интерфейс `Providers`, который имеет следующие методы.

- `<T> MessageBodyReader<T> getMessageBodyReader(java.lang.Class<T> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType)`

  Возвращает объект `MessageBodyReader`, соответствующий указанным критериям.

- `<T> MessageBodyWriter<T> getMessageBodyWriter(java.lang.Class<T> type, java.lang.reflect.Type genericType, java.lang.annotation.Annotation[] annotations, MediaType mediaType)`

  Возвращает объект `MessageBodyWriter`, соответствующий указанным критериям.

- `<T extends java.lang.Throwable> ExceptionMapper<T> getExceptionMapper(java.lang.Class<T> type)`

  Возвращает объект `ExceptionMapper` для указанного типа исключений.

- `<T> ContextResolver<T> getContextResolver(java.lang.Class<T> contextType, MediaType mediaType)`

  Возвращает объект `ContextResolver`, соответствующий указанным критериям.

Таким образом, с помощью введенного объекта `Providers` можно получать объекты поставщиков, например, для создания классов поставщиков сложного типа или, скажем, для использования объекта `JAXBContext`, полученного с помощью `ContextResolver`, в `MessageBodyReader`- или `MessageBodyWriter`-классе.

Пакет `javax.ws.rs.ext` также содержит классы `RuntimeDelegate` и `RuntimeDelegate.HeaderDelegate<T>`, являющиеся внутренними для JAX-RS и используемые средой выполнения JAX-RS для создания таких объектов, как `Response.ResponseBuilder`, `UriBuilder`, `Variant.VariantListBuilder`, и для маршализации/демаршализации HTTP-заголовков.

Метод `public abstract <T> T createEndpoint(Application application, java.lang.Class<T> endpointType)` класса `RuntimeDelegate` может быть использован для развертывания JAX-RS-приложения на платформе Java SE.

JAX-WS API

Пакеты *javax.jws* и *javax.jws.soap*

Аннотация `@WebService(element=value)` маркирует SEI-интерфейс или Java-класс, его реализующий (SEI-класс). Аннотация `@WebService` имеет следующие необязательные элементы, определяющие связывание "Java → WSDL":

- `public abstract java.lang.String name` — имя WSDL-элемента `<wsdl:portType>`;
- `public abstract java.lang.String targetNamespace` — целевое пространство имен для элементов `<wsdl:portType>` и/или `<wsdl:service>`;
- `public abstract java.lang.String serviceName` — имя элемента `<wsdl:service>` при маркировке SEI-класса;
- `<public abstract java.lang.String portName>` — имя элемента `<wsdl:port>` при маркировке SEI-класса;
- `<public abstract java.lang.String wsdlLocation>` — URL-адрес существующего WSDL-документа, которому должен соответствовать SEI-класс;
- `public abstract java.lang.String endpointInterface` — полное имя SEI-интерфейса при маркировке SEI-класса. Данный элемент позволяет определить отдельно SEI-интерфейс и SEI-класс.

Таким образом, модель программирования Web-сервиса JAX-WS отличается от модели JAX-RPC. В модели JAX-RPC всегда нужно кодировать SEI-интерфейс и SEI-класс, а в модели JAX-WS можно создать только SEI-класс.

Аннотация `@WebMethod(element=value)` маркирует бизнес-метод (метод, представляющий операцию Web-сервиса). Аннотация `@WebMethod` имеет следующие необязательные элементы:

- `public abstract java.lang.String operationName` — имя элемента `<wsdl:operation>`;
- `public abstract java.lang.String action` — значение атрибута `soapAction` элемента `<soap:operation>` для элемента `<wsdl:binding>`;
- `public abstract boolean exclude` — используется для маркировки метода SEI-класса, который не является бизнес-методом.

Аннотация `@Oneway` используется вместе с аннотацией `@WebMethod` для маркировки метода, имеющего только входящее сообщение и не имеющего исходящего сообщения.

Аннотация `@SOAPBinding(element=value)` пакета `javax.jws.soap` определяет параметры связывания конечной точки Web-сервиса с SOAP-протоколом с помощью следующих необязательных элементов.

- `public abstract SOAPBinding.Style style`

  Определяет стиль кодировки SOAP-сообщений. Перечисление `javax.jws.soap.SOAPBinding.Style` имеет константы `DOCUMENT` (по умолчанию) и `RPC`.

- `public abstract SOAPBinding.Use use`

  Определяет формат SOAP-сообщений. Перечисление `javax.jws.soap.SOAPBinding.Use` имеет константы `LITERAL` (по умолчанию) и `ENCODED`.

- `public abstract SOAPBinding.ParameterStyle parameterStyle`

  Определяет содержание параметров SOAP-сообщения в элементах-обертках. Перечисление `javax.jws.soap.SOAPBinding.ParameterStyle` имеет константы `BARE` и `WRAPPED` (по умолчанию).

Аннотация `@WebParam(element=value)` маркирует параметры бизнес-метода, уточняя их связывание. Аннотация `@WebParam` имеет следующие необязательные элементы:

- `public abstract java.lang.String name` — имя элемента `<wsdl:part>` (в случае RPC-стиля) или имя XML-элемента;
- `public abstract java.lang.String partName` — имя элемента `<wsdl:part>` (в случае RPC-стиля);
- `public abstract java.lang.String targetNamespace` — XML-пространство имен элемента параметра;
- `public abstract WebParam.Mode mode` — тип параметра: `IN`, `OUT` или `INOUT`;
- `public abstract boolean header` — если `true`, тогда параметр определен в SOAP-заголовке.

Аннотация `@WebResult(element=value)` маркирует возвращаемое значение бизнес-метода, уточняя его связывание. Аннотация `@WebResult` имеет следующие необязательные элементы:

- `public abstract java.lang.String name` — имя элемента `<wsdl:part>` (в случае RPC-стиля) или имя XML-элемента;
- `public abstract java.lang.String partName` — имя элемента `<wsdl:part>` (в случае RPC-стиля);
- `public abstract java.lang.String targetNamespace` — XML-пространство имен элемента результата;
- `public abstract boolean header` — если `true`, тогда результат определен в SOAP-заголовке.

Аннотация `@HandlerChain(element=value)` связывает реализацию Web-сервиса с файлом описания цепочки обработчиков SOAP-сообщений с помощью обязательного элемента `public abstract java.lang.String file`, указывающего расположение Handler-файла.

Конфигурационный Handler-файл

Конфигурационный Handler-файл имеет корневой элемент `<handler-chains>` (пространство имен `http://java.sun.com/xml/ns/javaee`), содержащий один или несколько элементов `<handler-chain>`.

Элемент `<handler-chain>` (рис. 1) имеет следующие дочерние элементы:

- `<service-name-pattern>` — QName-имя WSDL-элемента `<service>`, с которым связывается данная цепочка обработчиков;
- `<port-name-pattern>` — QName-имя WSDL-элемента `<port>`, с которым связывается данная цепочка обработчиков;
- `<protocol-binding>` — URI протокола сообщений;

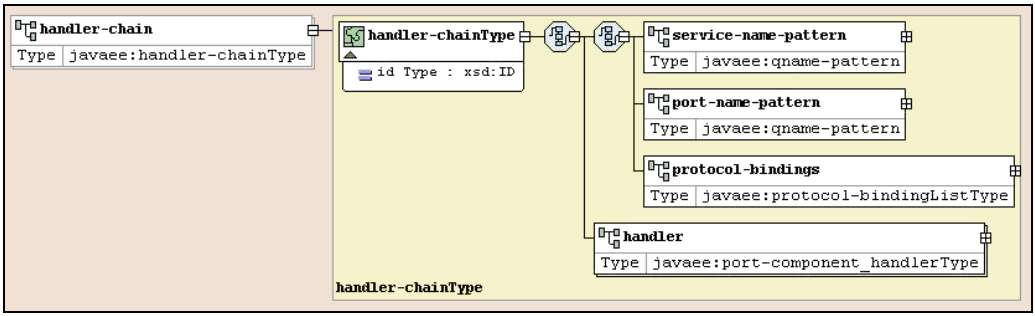


Рис. 1. Схема элемента `<handler-chain>`

- `<handler>` — обработчик сообщений, описываемый с помощью дочерних элементов:
  - обязательный элемент `<handler-name>` — имя обработчика;
  - обязательный элемент `<handler-class>` — полное имя Java-класса обработчика;
  - дополнительный элемент `<init-param>` определяет параметры инициализации обработчика с помощью дочерних элементов `<param-name>` и `<param-value>`;
  - дополнительный элемент `<soap-header>` — QName-имя обрабатываемого SOAP-заголовка;
  - дополнительный элемент `<soap-role>` — роль для обработчика.

Пакет *javax.xml.ws*

Аннотация `@WebServiceRef(element=value)` используется на стороне клиента Web-сервиса для ввода `Service`-объекта. Аннотация `@WebServiceRef` имеет следующие необязательные элементы:

- `public abstract java.lang.String name` — JNDI-имя `Service`-класса;
- `public abstract java.lang.Class<?> type` — Java-тип `Service`-класса, по умолчанию `java.lang.Object.class`;

- public abstract java.lang.String mappedName — глобальное JNDI-имя для Service-класса;
- public abstract java.lang.Class<? extends Service> value — имя класса javax.xml.ws.Service или его расширения;
- public abstract java.lang.String wsdlLocation — URL-адрес WSDL-документа;
- public abstract java.lang.String lookup — переносимое JNDI-имя Service-класса.

Аннотация @WebServiceRefs служит контейнером для аннотаций @WebServiceRef.

Аннотация @WebServiceProvider(*element=value*) маркирует класс, представляющий Web-сервис и реализующий интерфейс javax.xml.ws.Provider<T>. Аннотация @WebServiceProvider является альтернативой аннотации @WebService и позволяет создавать Web-сервисы, ориентированные на сообщения, которые основаны на модели RESTFul и обрабатывают XML-сообщения программным образом.

Использование аннотации @WebService скрывает от разработчика маршализацию/демаршализацию SOAP-сообщений. Аннотация же @WebServiceProvider дает возможность полного контроля над обработкой входящих сообщений и формированием исходящих сообщений.

Интерфейс Provider<T> имеет один метод T invoke(T *request*), вызываемый при получении сообщения. При этом поддерживаются три типа объектов, представляющих XML-данные:

- javax.xml.transform.Source;
- javax.activation.DataSource;
- javax.xml.soap.SOAPMessage.

В переопределенном методе invoke() используется соответствующий низкоуровневый интерфейс XML API для обработки входящих объектов вышеуказанных типов и формирования ответных сообщений.

Аннотация @WebServiceProvider имеет следующие необязательные элементы:

- public abstract java.lang.String wsdlLocation — адрес WSDL-документа;
- public abstract java.lang.String serviceName — имя WSDL-элемента <service>;
- public abstract java.lang.String targetNamespace — целевое пространство имен WSDL-элемента <service>;
- public abstract java.lang.String portName — имя WSDL-элемента <port>.

Вместе с аннотацией @WebServiceProvider используется аннотация @ServiceMode(*element=value*), указывающая, обрабатывает ли Provider-класс все сообщение или только его содержимое без заголовков. Для этого аннотация @ServiceMode использует элемент public abstract Service.Mode value со значениями javax.xml.ws.Service.Mode.PAYLOAD (по умолчанию обрабатывается только со-

держимое сообщения без заголовков) или `javax.xml.ws.Service.Mode.MESSAGE` (сообщение обрабатывается целиком).

Архитектура REST (Representational State Transfer) является альтернативой модели RPC (Remote Procedure Call) построения распределенных систем. RESTful (соответствующая архитектуре REST) распределенная система — это система распределенных информационных ресурсов, каждый из которых характеризуется своим URI-идентификатором и доступен по HTTP-протоколу.

RESTful Web-сервисы — это Web-сервисы, которые обрабатывают HTTP-запросы GET, PUT, POST и DELETE документов, представляющих состояния информационных ресурсов.

Для того чтобы реализовать в методе `invoke()` `Provider`-класса обработку HTTP-запросов GET, PUT, POST и DELETE, необходимо знать, какой HTTP-метод был применен при передаче запроса. Для решения этой задачи можно воспользоваться интерфейсом `javax.xml.ws.WebServiceContext`, обеспечивающим доступ к контексту сообщения. Интерфейс `WebServiceContext` имеет следующие методы.

- `MessageContext getMessageContext()`

  Возвращает объект `javax.xml.ws.handler.MessageContext`, представляющий контекст сообщения. Интерфейс `MessageContext` расширяет `java.util.Map<java.lang.String,java.lang.Object>`, обеспечивая набор свойств сообщения, и дополнительно имеет следующие поля и методы:

  - `static final java.lang.String MESSAGE_OUTBOUND_PROPERTY` — если `true`, тогда сообщение исходящее, если `false` — тогда входящее;
  - `static final java.lang.String INBOUND_MESSAGE_ATTACHMENTS` — набор типа `java.util.Map` вложений {MIME `Content-ID`, `DataHandler`} для входящего сообщения;
  - `static final java.lang.String OUTBOUND_MESSAGE_ATTACHMENTS` — набор типа `java.util.Map` вложений {MIME `Content-ID`, `DataHandler`} для исходящего сообщения;
  - `static final java.lang.String WSDL_DESCRIPTION` — источник типа `org.xml.sax.InputSource` WSDL-документа;
  - `static final java.lang.String WSDL_SERVICE` — имя WSDL-элемента `<service>`;
  - `static final java.lang.String WSDL_PORT` — имя WSDL-элемента `<port>`;
  - `static final java.lang.String WSDL_INTERFACE` — имя WSDL-элемента `<portType>`;
  - `static final java.lang.String WSDL_OPERATION` — имя WSDL-элемента `<operation>`;
  - `static final java.lang.String HTTP_RESPONSE_CODE` — ответный HTTP-код;
  - `static final java.lang.String HTTP_REQUEST_HEADERS` — набор типа `java.util.Map<java.lang.String>` или `java.util.List<java.lang.String>` HTTP-заголовков запроса;

- `static final java.lang.String HTTP_RESPONSE_HEADERS` — набор типа `java.util.Map<java.lang.String>` или `java.util.List<java.lang.String>` HTTP-заголовков ответа;
- `static final java.lang.String HTTP_REQUEST_METHOD` — HTTP-метод запроса;

Примечание

Таким образом, метод `getMessageContext()` объекта `WebServiceContext` дает доступ к HTTP-методу запроса, используя свойство `HTTP_REQUEST_METHOD` объекта `MessageContext`.

- `static final java.lang.String SERVLET_REQUEST` — объект `javax.servlet.http.HttpServletRequest`;
- `static final java.lang.String SERVLET_RESPONSE` — объект `javax.servlet.http.HttpServletResponse`;
- `static final java.lang.String SERVLET_CONTEXT` — объект `javax.servlet.ServletContext`;
- `static final java.lang.String QUERY_STRING` — строка HTTP-запроса.
- `static final java.lang.String PATH_INFO` — HTTP-свойство пути ресурса `PathInfo`;
- `static final java.lang.String REFERENCE_PARAMETERS` — набор SOAP-заголовков, имеющих атрибут `wsa:IsReferenceParameter="true"`;
- `void setScope(java.lang.String name, MessageContext.Scope scope)` и `MessageContext.Scope getScope(java.lang.String name)` — первый метод устанавливает, второй метод возвращает область действия свойства сообщения. Перечисление `javax.xml.ws.handler.MessageContext.Scope` имеет константы `APPLICATION` (свойство доступно обработчику, клиенту, конечной точке) и `HANDLER` (свойство доступно только обработчику).

❒ `java.security.Principal getUserPrincipal()`

Возвращает объект `java.security.Principal`, представляющий пользователя запроса.

❒ `boolean isUserInRole(java.lang.String role)`

Возвращает `true`, если аутентифицированный пользователь соответствует указанной роли.

❒ `EndpointReference getEndpointReference(org.w3c.dom.Element... referenceParameters)`

`<T extends EndpointReference> T getEndpointReference(java.lang.Class<T> clazz, org.w3c.dom.Element... referenceParameters)`

Возвращает объект `javax.xml.ws.EndpointReference`, представляющий ссылку на конечную точку Web-сервиса. Класс `EndpointReference` имеет методы:

`public static EndpointReference readFrom(javax.xml.transform.Source eprInfoset)`

`public abstract void writeTo(javax.xml.transform.Result result)`

```
public <T> T getPort(java.lang.Class<T> serviceEndpointInterface,
  WebServiceFeature... features)
public java.lang.String toString()
```

Объект `WebServiceContext` вводится в класс Web-сервиса с помощью аннотации `@Resources`. Имея объект `WebServiceContext`, можно получить объект `MessageContext` и, соответственно, HTTP-метод запроса. Дальше уже можно реализовать методы, обрабатывающие GET-, PUT-, POST- и DELETE-запросы.

Аннотация `@BindingType(element=value)` позволяет определить для класса реализации Web-сервиса связывание конечной точки с определенным протоколом передачи сообщений. Дополнительный элемент `public abstract java.lang.String value` имеет следующие возможные значения:

- `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING` — протокол SOAP 1.1 через HTTP (по умолчанию);
- `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING` — протокол SOAP 1.2 через HTTP;
- `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING` — протокол SOAP 1.1 через HTTP с применением MTOM;
- `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_MTOM_BINDING` — протокол SOAP 1.2 через HTTP с применением MTOM;
- `javax.xml.ws.http.HTTPBinding.HTTP_BINDING` — используется для маркировки `Provider`-класса для связывания конечной точки с XML/HTTP-протоколом передачи сообщений.

Аннотация `@Action(element=value)` маркирует бизнес-метод Web-сервиса и определяет значение атрибута `wsam:Action` WSDL-элементов `<input>`, `<output>` и `<fault>` элемента `<operation>`. Аннотация `@Action` имеет следующие дополнительные элементы:

- `public abstract java.lang.String input` — значение атрибута `wsam:Action` WSDL-элемента `<input>`;
- `public abstract java.lang.String output` — значение атрибута `wsam:Action` WSDL-элемента `<output>`;
- `public abstract FaultAction[] fault` — значение атрибута `wsam:Action` WSDL-элементов `<fault>`. Для того чтобы определить значение атрибута `wsam:Action` для каждого WSDL-элемента `<fault>`, необходимо воспользоваться аннотацией `@FaultAction` — `@Action(fault={@FaultAction(className=[Exception].class, value="...")...}).`

Аннотация `@RequestWrapper(element=value)` маркирует бизнес-метод Web-сервиса и определяет имя SOAP-элемента, содержащего параметры запроса в теле сообщения, а также имя генерируемого реализацией JAXB-класса, отвечающего за маршализацию/демаршализацию данных. Аннотация `@RequestWrapper` имеет следующие дополнительные элементы:

- public abstract java.lang.String localName — локальное имя элемента-обертки параметров;
- public abstract java.lang.String targetNamespace — пространство имен элемента-обертки параметров;
- public abstract java.lang.String partName — имя элемента <wsdl:part> параметров;
- public abstract java.lang.String className — имя JAXB-класса.

Аннотация @ResponseWrapper(*element=value*) маркирует бизнес-метод Web-сервиса и определяет имя SOAP-элемента, содержащего параметры ответа в теле сообщения, а также имя генерируемого реализацией JAXB-класса, отвечающего за маршализацию/демаршализацию данных. Аннотация @ResponseWrapper имеет следующие дополнительные элементы:

- public abstract java.lang.String localName — локальное имя элемента-обертки параметров;
- public abstract java.lang.String targetNamespace — пространство имен элемента-обертки параметров;
- public abstract java.lang.String partName — имя элемента <wsdl:part> параметров;
- public abstract java.lang.String className — имя JAXB-класса.

Спецификация Java API for XML-Based Web Services (JAX-WS) Version 2.1 вводит понятие свойств (features) для контроля над свойствами JAX-WS-реализации. Аннотации javax.xml.ws.@RespectBinding, javax.xml.ws.soap.@Addressing и javax.xml.ws.soap.@MTOM, так же как и классы javax.xml.ws.RespectBindingFeature, javax.xml.ws.soap.AddressingFeature, javax.xml.ws.soap.MTOMFeature, помогают осуществлять контроль над применением WSDL-документа, WS-Addressing и MTOM во время выполнения. Данные аннотации используются вместе с аннотациями @WebService, @WebServiceProvider и @WebServiceRef.

Аннотация @RespectBinding(enabled=true/false) в случае значения элемента enabled=true заставляет JAX-WS-реализацию проверять во время выполнения элемент <wsdl:binding> на предмет соответствия связывания конечной точки.

Класс RespectBindingFeature имеет конструктор public RespectBindingFeature (boolean enabled), позволяющий включать или выключать данное свойство.

Аннотация @WebServiceClient(*element=value*) маркирует генерируемый JAX-WS-реализацией на стороне клиента Service-класс, расширяющий класс javax.xml.ws.Service. Аннотация @WebServiceClient имеет следующие дополнительные элементы:

- public abstract java.lang.String name — локальное имя элемента <wsdl:service>;
- public abstract java.lang.String targetNamespace — пространство имен элемента <wsdl:service>;
- public abstract java.lang.String wsdlLocation — URL-адрес WSDL-документа.

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Аннотация `@WebEndpoint(name=[`*локальное имя конечной точки*`])` маркирует методы `getPortName()` генерируемого `Service`-класса.

Класс `javax.xml.ws.Service` (в отличие от JAX-RPC это класс, а не интерфейс) служит для JAX-WS-клиента фабрикой динамических объектов заглушки — Proxy-объектов, а также объектов `javax.xml.ws.Dispatch<T>`, обеспечивающих динамический вызов операций SEI-интерфейса.

Отличие JAX-WS от JAX-RPC состоит в том, что в модели программирования JAX-WS на стороне клиента отсутствует Stub-объект — есть только Proxy-объект, а вместо Call-объекта используется Dispatch-объект.

Класс `Service` предоставляет для JAX-WS-клиента следующие методы:

- `public <T> T getPort(javax.xml.namespace.QName` *`portName`*`, java.lang.Class<T>` *`serviceEndpointInterface`*`)`

  `public <T> T getPort(javax.xml.namespace.QName` *`portName`*`, java.lang.Class<T>` *`serviceEndpointInterface`*`, WebServiceFeature... features)`

  `public <T> T getPort(java.lang.Class<T>` *`serviceEndpointInterface`*`)`

  `public <T> T getPort(java.lang.Class<T>` *`serviceEndpointInterface`*`, WebServiceFeature...` *`features`*`)`

  `public <T> T getPort(EndpointReference` *`endpointReference`*`, java.lang.Class<T>` *`serviceEndpointInterface`*`, WebServiceFeature...` *`features`*`)`

  Возвращают Proxy-объект для SEI-интерфейса Web-сервиса. При этом JAX-WS-реализация использует Dynamic Proxy Class API — класс `java.lang.reflect.Proxy`. Класс `javax.xml.ws.WebServiceFeature` позволяет программным образом установить требуемые свойства JAX-WS-реализации. Спецификация JAX-WS предопределяет некоторые свойства, представленные классами `javax.xml.ws.soap.AddressingFeature`, `javax.xml.ws.soap.MTOMFeature` и `javax.xml.ws.RespectBindingFeature`, расширяющими класс `WebServiceFeature`. Альтернативный способ установки свойств JAX-WS-реализации — это применение аннотаций `javax.xml.ws.@RespectBinding`, `javax.xml.ws.soap.@Addressing` и `javax.xml.ws.soap.@MTOM`.

- `public void addPort(javax.xml.namespace.QName` *`portName`*`, java.lang.String` *`bindingId`*`, java.lang.String` *`endpointAddress`*`)`

  Для использования Dispatch-объекта создает порт Web-сервиса с указанными параметрами: `portName` — QName-имя конечной точки, `endpointAddress` — адрес конечной точки, `bindingId` (связывание конечной точки — `SOAPBinding.SOAP11HTTP_BINDING`, `SOAPBinding.SOAP12HTTP_BINDING`, `HTTPBinding.HTTP_BINDING` и др.).

- `public <T> Dispatch<T> createDispatch(javax.xml.namespace.QName` *`portName`*`, java.lang.Class<T>` *`type`*`, Service.Mode` *`mode`*`)`

  `public <T> Dispatch<T> createDispatch(javax.xml.namespace.QName` *`portName`*`, java.lang.Class<T>` *`type`*`, Service.Mode` *`mode`*`, WebServiceFeature...` *`features`*`)`

```
public <T> Dispatch<T> createDispatch(EndpointReference
  endpointReference, java.lang.Class<T> type, Service.Mode mode,
  WebServiceFeature... features)
public Dispatch<java.lang.Object>
  createDispatch(javax.xml.namespace.QName portName,
  JAXBContext context, Service.Mode mode)
public Dispatch<java.lang.Object>
  createDispatch(javax.xml.namespace.QName portName,
  JAXBContext context, Service.Mode mode, WebServiceFeature... features)
public Dispatch<java.lang.Object> createDispatch(EndpointReference
  endpointReference, JAXBContext context, Service.Mode mode,
  WebServiceFeature... features)
```

Создают Dispatch-объект с указанными параметрами, где: *portName* — QName-имя конечной точки; *type* — Java-тип объекта, представляющего сообщения — `javax.xml.transform.Source`, `javax.xml.soap.SOAPMessage` или `javax.activation.DataSource`; *mode* — режим обработки сообщений (`Service.Mode.MESSAGE` или `Service.Mode.PAYLOAD`); *context* — объект JAXBContext, используемый для маршализации/демаршализации сообщений.

- `public javax.xml.namespace.QName getServiceName()`

  Возвращает QName-имя Web-сервиса.

- `public java.util.Iterator<javax.xml.namespace.QName> getPorts()`

  Возвращает итератор QName-имен конечных точек Web-сервиса.

- `public java.net.URL getWSDLDocumentLocation()`

  Возвращает URL-адрес WSDL-документа.

- `public HandlerResolver getHandlerResolver()`

  `public void setHandlerResolver(HandlerResolver handlerResolver)`

  Первый метод возвращает, второй метод устанавливает объект `javax.xml.ws.handler.HandlerResolver`, содержащий информацию о цепочке обработчиков сообщений для определенной конечной точки.

- `public java.util.concurrent.Executor getExecutor()`

  `public void setExecutor(java.util.concurrent.Executor executor)`

  Первый метод возвращает, второй метод устанавливает объект `java.util.concurrent.Executor`, используемый `Service`-объектом для выполнения асинхронных обратных вызовов.

- `public static Service create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)`

  `public static Service create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName, WebServiceFeature... features)`

  `public static Service create(javax.xml.namespace.QName serviceName)`

  `public static Service create(javax.xml.namespace.QName serviceName, WebServiceFeature... features)`

  Динамически создают `Service`-объект. В JAX-WS отсутствует, в отличие от JAX-RPC, класс-фабрика для создания `Service`-объектов, его заменяет статиче-

ский метод `create()`. Использование динамического `Service`-объекта делает ненужным генерацию артефактов на стороне клиента.

Интерфейс `javax.xml.ws.Dispatch<T>` позволяет осуществлять синхронный и асинхронный обмен сообщениями с конечной точкой Web-сервиса, при этом сообщения могут быть представлены объектами Java-типа `javax.xml.transform.Source`, `javax.xml.soap.SOAPMessage` или `javax.activation.DataSource`. В случае использования интерфейса `Dispatch` клиентское приложение полностью отвечает за создание исходящих сообщений. Можно сказать, что интерфейс `Dispatch` представляет низкоуровневый API обработки сообщений на стороне клиента, также как интерфейс `Provider` на стороне сервера. Интерфейс `Dispatch` имеет следующие методы.

- `T invoke(T msg)` — синхронный вызов конечной точки.
- `Response<T> invokeAsync(T msg)` или `java.util.concurrent.Future<?> invokeAsync (T msg, AsyncHandler<T> handler)` — асинхронный вызов конечной точки. При использовании первого метода клиенту необходимо в цикле проверять, когда конечная точка Web-сервиса возвратит ответ на запрос и объект `Response<T>` будет сформирован. Интерфейс `javax.xml.ws.Response<T>` расширяет интерфейс `java.util.concurrent.Future<T>` и дополнительно предоставляет метод `java.util.Map<java.lang.String, java.lang.Object> getContext()`, возвращающий контекст ответного сообщения. Применение метода `Future<?> invokeAsync(T msg, AsyncHandler<T> handler)` избавляет клиента от необходимости в цикле проверять наличие ответа от сервера. Вместо этого клиент использует слушателя `AsyncHandler<T>` — объект пользовательского класса, реализующего интерфейс `javax.xml.ws.AsyncHandler<T>`, который имеет один метод `void handleResponse(Response<T> res)`, вызываемый реализацией при получении ответа от сервера. Интерфейс `java.util.concurrent.Future<T>` представляет результат асинхронной операции и имеет методы:
  - `boolean cancel(boolean mayInterruptIfRunning)` — прерывает выполнение задачи;
  - `boolean isCancelled()` — возвращает `true`, если задача была прервана;
  - `boolean isDone()` — возвращает `true`, если задача была выполнена;
  - `V get()` или `V get(long timeout, TimeUnit unit)` — возвращает результат выполнения задачи.
- `void invokeOneWay(T msg)`

  Вызывает конечную точку в синхронном режиме "запрос без ответа", ожидает HTTP-код 200 или HTTP-код ошибки.

В JAX-RPC интерфейсы `Stub` и `Call` сами имеют набор свойств, таких как `USERNAME_PROPERTY`, `PASSWORD_PROPERTY` и др. В JAX-WS за конфигурацию объектов `Proxy` и `Dispatch` отвечает интерфейс `javax.xml.ws.BindingProvider`, который они реализуют. Интерфейс `BindingProvider` имеет следующие свойства и методы.

- `static final java.lang.String USERNAME_PROPERTY` — имя пользователя для аутентификации.

- `static final java.lang.String PASSWORD_PROPERTY` — пароль для аутентификации.
- `static final java.lang.String ENDPOINT_ADDRESS_PROPERTY` — адрес конечной точки.
- `static final java.lang.String SESSION_MAINTAIN_PROPERTY` — если `true`, тогда клиент участвует в сессии, назначенной конечной точкой.
- `static final java.lang.String SOAPACTION_USE_PROPERTY` — если `true`, тогда заголовок `SOAPAction` используется.
- `static final java.lang.String SOAPACTION_URI_PROPERTY` — URI-идентификатор для `SOAPAction`-заголовка.
- `java.util.Map<java.lang.String,java.lang.Object> getRequestContext()`

  Возвращает контекст запроса для его последующей инициализации с помощью метода `V put(K key, V value)` интерфейса `Map`. Контекст запроса может содержать различного рода свойства, в том числе и свойства интерфейса `BindingProvider`.
- `java.util.Map<java.lang.String,java.lang.Object> getResponseContext()`

  Возвращает контекст ответа на синхронный запрос. Контекст ответа на асинхронный запрос возвращается через интерфейс `Response`.
- `Binding getBinding()`

  Возвращает объект `javax.xml.ws.Binding`, представляющий связывание с протоколами передачи сообщений. Интерфейс `Binding` является базовым для интерфейсов `javax.xml.ws.http.HTTPBinding` и `javax.xml.ws.soap.SOAPBinding` и имеет следующие методы:

  - `java.util.List<Handler> getHandlerChain()` и `void setHandlerChain(java.util.List<Handler> chain)` — первый метод возвращает, второй метод устанавливает цепочку обработчиков сообщений для данного связывания;
  - `java.lang.String getBindingID()` — возвращает URI-идентификатор связывания. Для `SOAPBinding` — это `http://schemas.xmlsoap.org/wsdl/soap/http` (`SOAP11HTTP_BINDING`), `http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true` (`SOAP11HTTP_MTOM_BINDING`), `http://www.w3.org/2003/05/soap/bindings/HTTP/` (`SOAP12HTTP_BINDING`), `http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true` (`SOAP12HTTP_MTOM_BINDING`), для `HTTPBinding` — `http://www.w3.org/2004/08/wsdl/http`.
- `EndpointReference getEndpointReference()`

  `<T extends EndpointReference> T getEndpointReference(java.lang.Class<T> clazz)`

  Возвращает объект `javax.xml.ws.EndpointReference`, содержащий информацию о конечной точке согласно спецификации WS-Addressing.

Аннотация `@WebFault(element=value)` используется для маркировки специфических классов исключений на стороне сервера, определяя их связывание с WSDL-

элементом `<wsdl:fault>`, а также маркирует генерируемые JAX-WS-реализацией из WSDL-документа классы исключений на стороне клиента. Аннотация `@WebFault` имеет следующие дополнительные элементы:

- `public abstract java.lang.String name` — локальное имя элемента, определенное как значение атрибута `element` WSDL-элемента `<wsdl:part>` элемента `<wsdl:message>`, на который ссылается элемент `<wsdl:fault>`;
- `public abstract java.lang.String targetNamespace` — пространство имен элемента;
- `public abstract java.lang.String faultBean` — имя Java-класса, представляющего Java-тип данных элемента;
- `public abstract java.lang.String messageName` — имя WSDL-элемента `<wsdl:message>`, на который ссылается элемент `<wsdl:fault>`, по умолчанию имя класса исключения.

Класс `javax.xml.ws.Holder<T>` используется для представления параметров `WebParam.Mode.OUT` и `WebParam.Mode.INOUT` аналогично JAX-RPC. Класс `Holder` имеет поле `public T value` и конструкторы `public Holder()` и `public Holder(T value)`.

Класс `javax.xml.ws.Endpoint` позволяет разворачивать Web-сервис на платформе Java SE 6. Платформа Java SE 6 включает в себя облегченный вариант Web-сервера приложений, дающий возможность публикации конечной точки Web-сервиса динамически во время выполнения. Для запуска Web-сервиса на платформе Java SE 6 создается главный Java-класс, содержащий метод `main()`, который использует Endpoint API.

Класс `Endpoint` имеет следующие поля, конструкторы и методы.

- `public static final java.lang.String WSDL_SERVICE` — QName-имя WSDL-элемента `<service>`.
- `public static final java.lang.String WSDL_PORT` — QName-имя WSDL-элемента `<port>`.
- `public Endpoint()`

  Создает объект `Endpoint`.
- `public static Endpoint create(java.lang.Object implementor)`

  `public static Endpoint create(java.lang.Object implementor, WebServiceFeature... features)`

  `public static Endpoint create(java.lang.String bindingId, java.lang.Object implementor)`

  `public static Endpoint create(java.lang.String bindingId, java.lang.Object implementor, WebServiceFeature... features)`

  Создает объект `Endpoint` с определенным протоколом обмена сообщениями, свойствами среды выполнения и для указанного класса реализации Web-сервиса, промаркированного аннотацией `@WebService` или `@WebServiceProvider`.

- `public abstract Binding getBinding()`

  Возвращает объект `javax.xml.ws.Binding`, представляющий связывание с протоколами передачи сообщений.

- `public abstract java.lang.Object getImplementor()`

  Возвращает объект класса реализации Web-сервиса.

- `public abstract void publish(java.lang.String address)`

  `public static Endpoint publish(java.lang.String address, java.lang.Object implementor)`

  `public static Endpoint publish(java.lang.String address, java.lang.Object implementor, WebServiceFeature... features)`

  `public abstract void publish(java.lang.Object serverContext)`

  `public void publish(HttpContext serverContext)`

  Создают и публикуют конечную точку Web-сервиса по указанному адресу. Использование класса `javax.xml.ws.spi.http.HttpContext` обеспечивает развертывание Web-сервиса в любом HTTP-контейнере, не только JAX-WS RI, поддерживающем пакет `javax.xml.ws.spi.http`. Для этого создается специфический для конкретного контейнера `HttpContext`-объект и передается в метод `Endpoint.publish()`. При этом среда выполнения JAX-WS вызывает метод `setHandler(HttpHandler handler)` класса `HttpContext` для установки обработчика HTTP-запросов для данного контекста. Во время выполнения обработчики `javax.xml.ws.spi.http.HttpHandler` оперируют объектами `javax.xml.ws.spi.http.HttpExchange`, представляющими HTTP-запросы.

- `public abstract void stop()`

  Останавливает публикацию конечной точки.

- `public abstract boolean isPublished()`

  Возвращает `true`, если конечная точка опубликована.

- `public abstract java.util.List<javax.xml.transform.Source> getMetadata()`

  `public abstract void setMetadata (java.util.List<javax.xml.transform.Source> metadata)`

  Первый метод возвращает, второй метод устанавливает список документов для Web-сервиса (описание WSDL и схема XML Schema документов).

- `public abstract java.util.concurrent.Executor getExecutor()`

  `public abstract void setExecutor(java.util.concurrent.Executor executor)`

  Возвращает и устанавливает объект `Executor`, отвечающий за адресацию запросов объекту класса реализации Web-сервиса.

- `public abstract java.util.Map<java.lang.String,java.lang.Object> getProperties()`

  `public abstract void setProperties (java.util.Map<java.lang.String,java.lang.Object> properties)`

  Первый метод возвращает, второй метод устанавливает свойства конечной точки.

- `public abstract EndpointReference getEndpointReference(org.w3c.dom.Element... referenceParameters)`

  `public abstract <T extends EndpointReference> T getEndpointReference(java.lang.Class<T> clazz, org.w3c.dom.Element... referenceParameters)`

  Создают объект `EndpointReference`, содержащий информацию о конечной точке.

- `public void setEndpointContext(EndpointContext ctxt)`

  Используется средой выполнения JAX-WS при развертывании Web-сервиса, содержащего несколько конечных точек.

Спецификация JAX-WS определяет разрешение `WebServicePermission ("publishEndpoint")`, проверяемое `SecurityManager`, при публикации конечной точки. Разрешение `javax.xml.ws.WebServicePermission` расширяет `java.security.BasicPermission` и позволяет администратору Web-сервера ограничить возможность приложений публиковать конечные точки.

Пакеты *javax.xml.ws.handler* и *javax.xml.ws.handler.soap*

Пакеты `javax.xml.ws.handler` и `javax.xml.ws.handler.soap` позволяют создавать обработчиков входящих и исходящих SOAP-сообщений как на стороне клиента, так и на стороне сервера. На стороне сервера цепочки обработчиков устанавливаются с помощью аннотации `@javax.jws.HandlerChain`, на стороне клиента цепочки обработчиков конфигурируются посредством интерфейса `javax.xml.ws.handler.HandlerResolver`.

Объект `HandlerResolver` используется в качестве аргумента метода `Service.setHandlerResolver(HandlerResolver handlerResolver)` и создается как объект клиентского класса, реализующего интерфейс `HandlerResolver`. Интерфейс `HandlerResolver` имеет единственный метод `java.util.List<Handler> getHandlerChain(PortInfo portInfo)`, вызываемый средой выполнения для получения списка обработчиков. Таким образом, основной задачей переопределения метода `getHandlerChain()` является создание списка обработчиков сообщений на стороне клиента. Объект `javax.xml.ws.handler.PortInfo` является внутренним для JAX-WS-реализации и предоставляет информацию о порте Web-сервиса, для которого создается цепочка обработчиков, с помощью перечисленных далее методов.

- `javax.xml.namespace.QName getServiceName()`

  Возвращает имя WSDL-элемента `<service>`.

- `javax.xml.namespace.QName getPortName()`

  Возвращает имя WSDL-элемента `<port>`.

- `java.lang.String getBindingID()`

  Возвращает идентификатор связывания с протоколом передачи сообщений.

Интерфейс `javax.xml.ws.handler.Handler<C extends MessageContext>` является базовым интерфейсом для реализации классами JAX-WS обработчиков сообщений.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Интерфейс `Handler` имеет следующие методы.

- `boolean handleMessage(C context)`

  Вызывается средой выполнения для обработки входящих и исходящих сообщений. Возвращает `false`, если дальнейшая обработка сообщения в цепочке заблокирована.

- `boolean handleFault(C context)`

  Вызывается средой выполнения для обработки сообщений об ошибках.

- `void close(MessageContext context)`

  Вызывается после окончания обработки для освобождения ресурсов.

Интерфейс `javax.xml.ws.handler.MessageContext` представляет контекст обрабатываемых сообщений и имеет следующие поля:

- `static final java.lang.String MESSAGE_OUTBOUND_PROPERTY` — если `true`, тогда сообщение исходящее;
- `static final java.lang.String INBOUND_MESSAGE_ATTACHMENTS` — набор типа `java.util.Map` вложений входящего сообщения {MIME `Content-ID`, `DataHandler`};
- `static final java.lang.String OUTBOUND_MESSAGE_ATTACHMENTS` — набор типа `java.util.Map` вложений исходящего сообщения {MIME `Content-ID`, `DataHandler`};
- `static final java.lang.String WSDL_DESCRIPTION` — входящий источник типа `org.xml.sax.InputSource` для WSDL-документа;
- `static final java.lang.String WSDL_SERVICE` — QName-имя WSDL-элемента `<service>`;
- `static final java.lang.String WSDL_PORT` — QName-имя WSDL-элемента `<port>`;
- `static final java.lang.String WSDL_INTERFACE` — QName-имя WSDL-элемента `<interface>` или `<portType>`;
- `static final java.lang.String WSDL_OPERATION` — QName-имя WSDL-элемента `<operation>`;
- `static final java.lang.String HTTP_RESPONSE_CODE` — HTTP-код ответа;
- `static final java.lang.String HTTP_REQUEST_HEADERS` — HTTP-заголовки запроса;
- `static final java.lang.String HTTP_RESPONSE_HEADERS` — HTTP-заголовки ответа;
- `static final java.lang.String HTTP_REQUEST_METHOD` — HTTP-метод запроса;
- `static final java.lang.String SERVLET_REQUEST` — объект запроса `javax.servlet.http.HttpServletRequest`;
- `static final java.lang.String SERVLET_RESPONSE` — объект ответа `javax.servlet.http.HttpServletResponse`;
- `static final java.lang.String SERVLET_CONTEXT` — объект `javax.servlet.ServletContext`;

- `static final java.lang.String QUERY_STRING` — строка запроса;
- `static final java.lang.String PATH_INFO` — HTTP-свойство `PathInfo`;
- `static final java.lang.String REFERENCE_PARAMETERS` — SOAP-заголовки, имеющие атрибут `wsa:IsReferenceParameter="true"`.

Интерфейс `MessageContext` имеет следующие методы:

```
void setScope(java.lang.String name, MessageContext.Scope scope)
MessageContext.Scope getScope(java.lang.String name)
```

Первый метод устанавливает, второй метод возвращает область действия свойства. Перечисление `javax.xml.ws.handler.MessageContext.Scope` имеет константы `APPLICATION` и `HANDLER`.

Интерфейс `javax.xml.ws.handler.LogicalHandler<C extends LogicalMessageContext>` расширяет интерфейс `Handler<C>` и позволяет создавать обработчиков, имеющих доступ к свойствам контекста сообщения и к его содержимому за исключением заголовков, специфичных для протокола передачи сообщений.

Интерфейс `javax.xml.ws.handler.LogicalMessageContext` расширяет интерфейс `MessageContext` и имеет дополнительно метод `LogicalMessage getMessage()`, возвращающий объект `javax.xml.ws.LogicalMessage`, который представляет содержимое сообщения без специфичных для протокола заголовков. Интерфейс `LogicalMessage` имеет методы `javax.xml.transform.Source getPayload()`, `void setPayload(javax.xml.transform.Source payload)`, `java.lang.Object getPayload(JAXBContext context)`, `void setPayload(java.lang.Object payload, JAXBContext context)`, возвращающие и устанавливающие логическое содержимое сообщения.

Интерфейс `javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext>` расширяет интерфейс `Handler<T>` и позволяет создавать обработчиков заголовков сообщения. Интерфейс `SOAPHandler` дополнительно имеет метод `java.util.Set<javax.xml.namespace.QName> getHeaders()`, возвращающий QName-имена обрабатываемых заголовков сообщения.

Интерфейс `javax.xml.ws.handler.soap.SOAPMessageContext` расширяет интерфейс `MessageContext` и дополнительно имеет методы `javax.xml.soap.SOAPMessage getMessage()`, `void setMessage(javax.xml.soap.SOAPMessage message)`, `java.lang.Object[] getHeaders(javax.xml.namespace.QName header, JAXBContext context, boolean allRoles)`, `java.util.Set<java.lang.String> getRoles()`.

Пакеты *javax.xml.ws.http* и *javax.xml.ws.soap*

Пакет `javax.xml.ws.http` содержит исключение `HTTPException` и интерфейс `HTTPBinding` с полем `HTTP_BINDING`, представляющим XML/HTTP-связывание. Константа `HTTP_BINDING` используется аннотацией `@BindingType`, а также методами `Service.addPort()` и `Endpoint.create()`. Интерфейс `HTTPBinding` также наследует методы `getBindingID()`, `getHandlerChain()`, `setHandlerChain()` от интерфейса `javax.xml.ws.Binding`, который он расширяет. Объект `Binding` может быть получен

методом `getBinding()` интерфейса `javax.xml.ws.BindingProvider`, который реализуют объекты `Proxy` и `Dispatch` на стороне клиента, а также методом `Endpoint.getBinding()`.

Пакет `javax.xml.ws.soap` содержит интерфейс `SOAPBinding`, классы `AddressingFeature` и `MTOMFeature`, перечисление `AddressingFeature.Responses`, аннотации `@Addressing` и `@MTOM`, исключение `SOAPFaultException`.

Интерфейс `SOAPBinding` расширяет интерфейс `javax.xml.ws.Binding` и имеет константы `SOAP11HTTP_BINDING`, `SOAP12HTTP_BINDING`, `SOAP11HTTP_MTOM_BINDING` и `SOAP12HTTP_MTOM_BINDING`, используемые аннотацией `@BindingType`, а также методами `Service.addPort()` и `Endpoint.create()`. Интерфейс `SOAPBinding` дополнительно имеет, помимо методов `getBindingID()`, `getHandlerChain()`, `setHandlerChain()`, наследуемых от интерфейса `javax.xml.ws.Binding`, перечисленные далее методы.

- `java.util.Set<java.lang.String> getRoles()`
  `void setRoles(java.util.Set<java.lang.String> roles)`

  Возвращает и устанавливает роли для получателя SOAP-сообщений (например, `http://www.w3.org/2003/05/soap-envelope/role/next`).

- `boolean isMTOMEnabled()`
  `void setMTOMEnabled(boolean flag)`

  Возвращает и устанавливает использование MTOM.

- `javax.xml.soap.SOAPFactory getSOAPFactory()`

  Возвращает SAAJ-объект `SOAPFactory`, используемый для создания элементов SOAP-сообщений.

- `javax.xml.soap.MessageFactory getMessageFactory()`

  Возвращает SAAJ-объект `MessageFactory`, используемый для создания SOAP-сообщений.

Классы `AddressingFeature`, `MTOMFeature` и аннотации `@Addressing`, `@MTOM`, так же как и класс `javax.xml.ws.RespectBindingFeature` и аннотация `@RespectBinding`, позволяют устанавливать свойства JAX-WS-реализации. Классы `AddressingFeature`, `MTOMFeature` и `RespectBindingFeature` расширяют класс `javax.xml.ws.WebServiceFeature`, объекты которого используются методами `Service.getPort`, `Service.createDispatch`, `Endpoint.create` и `Endpoint.publish`. Аннотации `@Addressing`, `@MTOM` и `@RespectBinding` могут использоваться вместе с аннотациями `@WebService`, `@WebServiceProvider` и `@WebServiceRef`.

Класс `AddressingFeature` определяет использование спецификации WS-Addressing в протоколе SOAP 1.1/HTTP или SOAP 1.2/HTTP передачи сообщений. Класс `AddressingFeature` имеет следующие конструкторы, позволяющие контролировать данное свойство.

- `public AddressingFeature()`

  Делает возможным использование WS-Addressing.

- `public AddressingFeature(boolean enabled)`

  Если `false`, тогда использование WS-Addressing невозможно.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public AddressingFeature(boolean enabled, boolean required)`

  Если `required=true`, тогда использование WS-Addressing обязательно.

- `public AddressingFeature(boolean enabled, boolean required, AddressingFeature.Responses responses)`

  Перечисление `javax.xml.ws.soap.AddressingFeature.Responses` имеет константы `ANONYMOUS`, `NON_ANONYMOUS` и `ALL`, конфигурирующие ответы конечной точки.

Аннотация `@Addressing(element=value)` имеет следующие дополнительные элементы:

- `public abstract boolean enabled` — включает или выключает свойство;
- `public abstract boolean required` — делает обязательным использование свойства;
- `public abstract AddressingFeature.Responses responses` — конфигурирует ответы конечной точки.

Класс `javax.xml.ws.soap.MTOMFeature` определяет использование MTOM — метода эффективной передачи бинарных данных. Класс `MTOMFeature` имеет следующие конструкторы.

- `public MTOMFeature()`

  Делает возможным использование MTOM.

- `public MTOMFeature(boolean enabled)`

  Если `false`, тогда использование MTOM невозможно.

- `public MTOMFeature(int threshold)`

  Указывается размер в байтах бинарных данных, начиная с которого используется MTOM.

- `public MTOMFeature(boolean enabled, int threshold)`

  Включает или выключает данное свойство с указанием порога его применения.

Аннотация `@MTOM(element=value)` имеет следующие дополнительные элементы:

- `public abstract boolean enabled` — включает или выключает свойство;
- `public abstract int threshold` — указывает порог применения свойства.

Пакет *javax.xml.ws.spi*

Пакет `javax.xml.ws.spi` используется внутри JAX-WS API и обеспечивает связь с JAX-WS-реализацией.

Аннотация `javax.xml.ws.spi.@WebServiceFeatureAnnotation` используется для идентификации таких аннотаций, как `@Addressing`, `@MTOM` и `@RespectBinding`.

Класс `javax.xml.ws.spi.ServiceDelegate` является классом JAX-WS-реализации, которому класс `javax.xml.ws.Service` делегирует все методы за исключением метода `create()`.

Класс `javax.xml.ws.spi.Provider` используется классами `Service` и `Endpoint` в качестве класса-фабрики для создания объектов `Endpoint` и `ServiceDelegate`.

Класс `javax.xml.ws.spi.Invoker` используется JAX-WS-реализацией для вызова конечной точки Web-сервиса.

Пакет *javax.xml.ws.spi.http*

Пакет `javax.xml.ws.spi.http` обеспечивает развертывание JAX-WS Web-сервисов в HTTP-контейнере.

Процесс развертывания Web-сервиса в HTTP-контейнере, имеющем среду выполнения JAX-WS, состоит из следующих шагов:

1. Для каждой конечной точки Web-сервиса контейнер создает объект `javax.xml.ws.spi.Invoker`.
2. Используя метод `javax.xml.ws.spi.Provider.createEndpoint`, контейнер создает конечные точки и конфигурирует их.
3. Контейнер создает объект `javax.xml.ws.EndpointContext` для набора конечных точек.
4. Для каждой конечной точки контейнер создает объект `javax.xml.ws.spi.http.HttpContext`.
5. Каждая конечная точка публикуется с помощью метода `Endpoint.publish(HttpContext)`. При этом создаются объекты `javax.xml.ws.spi.http.HttpHandler`, обрабатывающие HTTP-запросы и устанавливающиеся с помощью метода `HttpContext.setHandler(HttpHandler)`.

При запросе Web-сервиса контейнер создает объект `javax.xml.ws.spi.http.HttpExchange`, представляющий HTTP-запрос и ответ. Далее объект `HttpExchange` передается в метод `HttpHandler.handle(HttpExchange)`, использующий объект `Invoker` для вызова Web-сервиса, по завершении которого HTTP-ответ записывается в объект `HttpExchange`.

Клиентское приложение может использовать классы `javax.xml.ws.Endpoint` и `javax.xml.ws.spi.http.HttpContext` для развертывания Web-сервиса в специфическом HTTP-контейнере, реализация которого предоставляет API для создания объектов `HttpContext`, используемых методом `Endpoint.publish(HttpContext)`.

Пакет *javax.xml.ws.wsaddressing*

Пакет `javax.xml.ws.wsaddressing` содержит классы `W3CEndpointReference` и `W3CEndpointReferenceBuilder`.

Класс `W3CEndpointReference` расширяет класс `javax.xml.ws.EndpointReference` и представляет удаленную ссылку на конечную точку Web-сервиса, соответствующую спецификации W3C WS-Addressing 1.0 — Core Recommendation. Объекты

`W3CEndpointReference` возвращаются методами `javax.xml.ws.BindingProvider.getEndpointReference()`, `javax.xml.ws.Endpoint.getEndpointReference()`, `javax.xml.ws.WebServiceContext.getEndpointReference` в случае использования протокола передачи сообщения SOAP1.1/HTTP или SOAP1.2/HTTP.

SEI-интерфейс Web-сервиса, предоставляющий метод, который возвращает ссылку на конечную точку, соответствующую спецификации W3C WS-Addressing, должен использовать объекты `W3CEndpointReference`.

JAXB по умолчанию связывает элемент `<wsa:EndpointReference>` с классом `W3CEndpointReference`.

Если SEI-интерфейс Web-сервиса должен содержать методы, возвращающие различные ссылки на конечную точку, необходимо использовать класс `W3CEndpointReferenceBuilder`, который имеет следующие конструкторы и методы.

- `public W3CEndpointReferenceBuilder()`

  Создает объекты `W3CEndpointReferenceBuilder`.

- `public W3CEndpointReferenceBuilder address(java.lang.String address)`

  Устанавливает элемент `<wsa:Address>`.

- `public W3CEndpointReferenceBuilder interfaceName(javax.xml.namespace.QName interfaceName)`

  Устанавливает элемент `<wsam:InterfaceName>` элемента `<wsa:Metadata>`.

- `public W3CEndpointReferenceBuilder serviceName (javax.xml.namespace.QName serviceName)`

  Устанавливает элемент `<wsam:ServiceName>` элемента `<wsa:Metadata>`.

- `public W3CEndpointReferenceBuilder endpointName(javax.xml.namespace.QName endpointName)`

  Устанавливает `EndpointName` элемента `<wsam:ServiceName>`.

- `public W3CEndpointReferenceBuilder wsdlDocumentLocation (java.lang.String wsdlDocumentLocation)`

  Устанавливает `<wsdli:wsdlLocation>` элемента `<wsa:Metadata>`.

- `public W3CEndpointReferenceBuilder referenceParameter(org.w3c.dom.Element referenceParameter)`

  Устанавливает элемент `<wsa:ReferenceParameters>`.

- `public W3CEndpointReferenceBuilder metadata(org.w3c.dom.Element metadataElement)`

  Устанавливает элемент `<wsa:Metadata>`.

- `public W3CEndpointReferenceBuilder element(org.w3c.dom.Element element)`

  Добавляет элементы расширения в элемент `<wsa:EndpointReference>`.

- `public W3CEndpointReferenceBuilder attribute (javax.xml.namespace.QName name, java.lang.String value)`

  Добавляет атрибуты расширения в элемент `<wsa:EndpointReference>`.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `public W3CEndpointReference build()`

  Создает объект `W3CendpointReference`.

Пакет *javax.annotation*

В JAX-WS аннотация `@Resource` используется для ввода объекта `WebServiceContext` в классе реализации конечной точки Web-сервиса.

Аннотации `@PostConstruct` и `@PreDestroy`, как правило, служат для маркировки методов класса реализации конечной точки Web-сервиса или Handler-класса обработчика сообщений. Данные методы отвечают за инициализацию конфигурации экземпляра класса с настройкой доступа к внешним ресурсам и за освобождение всех задействованных ресурсов соответственно. Контейнер вызывает метод, промаркированный аннотацией `@PostConstruct`, после создания экземпляра класса до начала его работы, а метод, промаркированный аннотацией `@PreDestroy`, перед уничтожением экземпляра класса. В этом смысле методы класса реализации конечной точки JAX-WS Web-сервиса с аннотациями `@PostConstruct` и `@PreDestroy` являются аналогом методов `init()` и `destroy()` интерфейса `javax.xml.rpc.server.ServiceLifecycle` JAX-RPC SEI-класса.

SAML

Security Assertion Markup Language (SAML) — это XML-стандарт обмена информацией системы безопасности между Web-сторонами, решающий задачу единого входа (Single Sign On).

Технология единого входа — это система, включающая в себя клиента, центр аутентификации и сервисы. Клиент проходит один раз аутентификацию в центральном сервисе аутентификации и тем самым получает автоматический доступ ко всем сервисам, которые поддерживают данный центр аутентификации и к которым центр аутентификации разрешил доступ клиенту.

В стандарте SAML технология SSO реализуется путем включения выдаваемых центральным сервисом аутентификации, который представляет собой Web-приложение, утверждений (assertions) в сообщения, участвующие в обмене между клиентом, использующим HTTP-агента, например Web-браузер, и остальными сервисами, представляющими собой Web-приложения, обеспечивающие для клиента необходимые ему Web-ресурсы. Утверждения при этом содержат информацию, подтверждающую и описывающую факт аутентификации данного клиента в центре аутентификации, которому доверяют все остальные сервисы, участвующие во взаимодействии с клиентом. На основании информации, содержащейся в утверждениях, запрашиваемые сервисы аутентифицируют клиента и выдают ему определенные права, разрешающие доступ к определенным локальным ресурсам.

SAML 1.1

Спецификация SAML 1.1 определяет следующие понятия.

- Assertions — утверждения. Каждое утверждение представлено структурой XML-элемента `<Assertion>`. Элемент `<Assertion>` может содержать информацию о том, что Web-сторона прошла аутентификацию определенным способом (например, с помощью пароля) в конкретное время, информацию о самой Web-стороне и права, предоставленные Web-стороне.
- Protocol — протокол запроса и получения утверждений.
- Bindings — механизмы передачи запроса и ответа SAML-протокола с помощью известных протоколов, например, с помощью SOAP/HTTP.
- Profiles — механизмы реализации SAML. Определены два профиля SSO — Browser/Artifact Profile и Browser/POST Profile.

Профиль Browser/Artifact Profile основывается на модели, в которой клиент с помощью HTTP-агента связывается с сайтом центра аутентификации и проходит аутентификацию/авторизацию для доступа к запрашиваемому Web-ресурсу. После

успешного прохождения аутентификации/авторизации клиента центр аутентификации генерирует HTTP-переменную, при этом ее значение включает в себя идентификатор центра аутентификации и ссылку на утверждение, содержащее подтверждение и описание факта аутентификации клиента. Данная HTTP-переменная включается в клиентский HTTP-запрос к ресурсу, к которому HTTP-запрос и перенаправляется. Получив клиентский HTTP-запрос, сайт ресурса, используя значение HTTP-переменной, посылает запрос центру аутентификации, который в ответ посылает сообщение, содержащее утверждение. Если утверждение является корректным, тогда клиент получает доступ к ресурсу. В другом сценарии клиент может в качестве первого шага связаться с сайтом ресурса. Сайт ресурса перенаправит клиентский запрос сайту центра аутентификации, и далее будут проделаны все вышеописанные действия.

Профиль Browser/POST Profile реализует сценарий, в котором клиент связывается с сайтом центра аутентификации и проходит аутентификацию/авторизацию. Центр аутентификации генерирует утверждение и посылает его в сообщении клиенту. После получения утверждения от центра аутентификации клиент посылает POST-запрос, содержащий утверждение, сайту ресурса, который, после проверки утверждения, обеспечивает клиенту доступ к ресурсу.

Утверждение представляет собой XML-данные, корневым элементом которых является элемент `<Assertion>` (пространство имен `urn:oasis:names:tc:SAML:1.0:assertion`). Элемент `<Assertion>` содержит следующие атрибуты и вложенные элементы:

- обязательный атрибут `MajorVersion` — высший порядок версии спецификации (значение — 1);
- обязательный атрибут `MinorVersion` — низший порядок версии спецификации (значение — 1). Таким образом, при значениях `MajorVersion=1` и `MinorVersion=1` версия спецификации SAML будет 1.1;
- обязательный атрибут `AssertionID` — идентификатор утверждения;
- обязательный атрибут `Issuer` — идентификатор стороны, генерирующей утверждение;
- обязательный атрибут `IssueInstant` — дата и время создания утверждения;
- дополнительный элемент `<Conditions>` — условия действительности утверждения;
- дополнительный элемент `<Advice>` — дополнительная информация об утверждении, например дополнительные утверждения, представленные элементами `<Assertion>` или ссылками на утверждения, представленные элементами `<AssertionIDReference>`;
- дополнительный элемент `<ds:Signature>` (пространство имен `xmlns:ds="http://www.w3.org/2000/09/xmldsig#"`) — цифровая подпись XML Signature.

Элемент `<Assertion>` содержит какие-либо из элементов:

- элемент `<Statement>` содержит элементы расширения спецификации SAML, описывающие утверждение;

- элемент `<SubjectStatement>` содержит элементы расширения спецификации SAML, описывающие утверждение, включая элемент `<Subject>` с информацией о клиенте;
- элемент `<AuthenticationStatement>` описывает утверждение того, что клиент прошел аутентификацию;
- элемент `<AuthorizationDecisionStatement>` описывает утверждение того, что клиент получил определенные права доступа к запрашиваемым ресурсам;
- элемент `<AttributeStatement>` описывает утверждение того, что клиент ассоциируется с определенными атрибутами (имя/значение), влияющими на права доступа, которые сервис ресурса выдает клиенту.

Элемент `<Conditions>` описывает условия действительности утверждения с помощью следующих атрибутов и вложенных элементов:

- дополнительный атрибут `NotBefore` указывает время, начиная с которого действует утверждение;
- дополнительный атрибут `NotOnOrAfter` указывает время, до которого действует утверждение;
- необязательный элемент `<Condition>` содержит элементы расширения спецификации SAML, описывающие дополнительные условия;
- необязательный элемент `<AudienceRestrictionCondition>` указывает стороны, для которых предназначено данное утверждение с помощью вложенных элементов `<Audience>`, содержащих URI-идентификаторы сторон;
- необязательный элемент `<DoNotCacheCondition>` — наличие элемента указывает, что данное утверждение не является повторно используемым.

Элемент `<Subject>` описывает клиента, которому выдано данное утверждение, используя вложенные элементы `<NameIdentifier>` и `<SubjectConfirmation>`.

Элемент `<NameIdentifier>` содержит имя клиента и имеет следующие атрибуты:

- дополнительный атрибут `NameQualifier` — сторона, определяющая имя клиента;
- дополнительный атрибут `Format` — URI-идентификатор формата имени клиента (по умолчанию `urn:oasis:names:tc:SAML:1.0:nameid-format:unspecified`). Спецификация предопределяет следующие форматы имени клиента:
  - `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`;
  - `urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName`;
  - `urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName`.

Элемент `<SubjectConfirmation>` содержит информацию, дающую возможность сервису ресурса подтвердить аутентификацию клиента с помощью вложенных элементов:

- обязательный элемент `<ConfirmationMethod>` содержит URI-идентификатор протокола, используемого для подтверждения аутентификации клиента. Спецификация предопределяет следующие URI-идентификаторы:

- urn:oasis:names:tc:SAML:1.0:cm:holder-of-key — аутентификация производится с помощью ключа, информация о котором содержится во вложенном элементе <ds:KeyInfo>. Это может быть, например, публичный ключ цифровой подписи, содержащейся в элементе <SubjectConfirmationData>;
- urn:oasis:names:tc:SAML:1.0:cm:sender-vouches — информация, помогающая аутентифицировать клиента;
- urn:oasis:names:tc:SAML:1.0:cm:artifact — клиент является стороной, передающей сервису ресурса ссылку, которую сервис ресурса использует для получения утверждения у центра аутентификации (профиль Browser/Artifact Profile);
- urn:oasis:names:tc:SAML:1.0:cm:bearer — клиент является стороной, передающей сервису ресурса утверждение (профиль Browser/POST Profile);

❒ необязательный элемент <SubjectConfirmationData> содержит данные, используемые протоколом для подтверждения аутентификации клиента;

❒ необязательный элемент <ds:KeyInfo> содержит информацию о ключе клиента.

Элемент <AuthenticationStatement> описывает утверждение аутентификации клиента, используя атрибуты и вложенные элементы:

❒ обязательный атрибут AuthenticationMethod — URI-идентификатор типа аутентификации. Спецификация предопределяет следующие типы аутентификации:
- urn:oasis:names:tc:SAML:1.0:am:password — аутентификация с помощью пароля;
- urn:ietf:rfc:1510 — аутентификация с помощью протокола Kerberos;
- urn:ietf:rfc:2945 — аутентификация по протоколу Secure Remote Password Protocol (SRP);
- urn:oasis:names:tc:SAML:1.0:am:HardwareToken — аутентификация с помощью аппаратного маркера защиты;
- urn:ietf:rfc:2246 — аутентификация по протоколу SSL или TLS;
- urn:oasis:names:tc:SAML:1.0:am:X509-PKI — аутентификация с помощью X.509 PKI-ключа;
- urn:oasis:names:tc:SAML:1.0:am:PGP — аутентификация с использованием ключа PGP (Pretty Good Privacy);
- urn:oasis:names:tc:SAML:1.0:am:SPKI — аутентификация с помощью SPKI PKI-ключа;
- urn:oasis:names:tc:SAML:1.0:am:XKMS — аутентификация с помощью ключа XKMS (XML Key Management Specification);
- urn:ietf:rfc:3075 — аутентификация с использованием цифровой подписи;
- urn:oasis:names:tc:SAML:1.0:am:unspecified — неопределенная аутентификация;

❒ обязательный атрибут AuthenticationInstant — время, когда клиент прошел аутентификацию;

- необязательный элемент `<SubjectLocality>` — локализация клиента (DNS-имя и IP-адрес), указываемая с помощью атрибутов `IPAddress` и `DNSAddress`;
- необязательный элемент `<AuthorityBinding>` — информация о центре аутентификации, указываемая атрибутами `AuthorityKind` (тип SAML-запроса центру аутентификации), `Location` (URI-адрес центра аутентификации), `Binding` (URI-идентификатор реализации SAML-протокола для связи с центром аутентификации);
- элемент `<Subject>` — сведения о клиенте.

Элемент `<AttributeStatement>` описывает ассоциацию клиента с атрибутами с помощью вложенных элементов `<Attribute>` и `<Subject>`. Элемент `<Attribute>` имеет обязательные атрибуты `AttributeNamespace`, `AttributeName` и вложенный элемент `<AttributeValue>`.

Элемент `<AuthorizationDecisionStatement>` содержит информацию о правах клиента, используя следующие атрибуты и вложенные элементы:

- обязательный атрибут `Resource` — URI-идентификатор ресурса, относительно которого назначаются права клиенту;
- обязательный атрибут `Decision` — права клиента, возможные значения: `Permit`, `Deny`, `Indeterminate`;
- обязательный элемент `<Action>` — действия, разрешенные для ресурсов, указываемые с помощью дополнительного атрибута `Namespace` (пространство имен действия; пространство имен `urn:oasis:names:tc:SAML:1.0:action:rwedc` соответствует действиям `Read`, `Write`, `Execute`, `Delete`, `Control`; пространство имен `urn:oasis:names:tc:SAML:1.0:action:rwedc-negation`, по умолчанию, соответствует действиям `Read`, `Write`, `Execute`, `Delete`, `Control`, `~Read`, `~Write`, `~Execute`, `~Delete` и `~Control`; пространство имен `urn:oasis:names:tc:SAML:1.0:action:ghpp` соответствует действиям `GET`, `HEAD`, `PUT`, `POST`; пространство имен `urn:oasis:names:tc:SAML:1.0:action:unix` соответствует действиям `extended`, `user`, `group`, `world`, представленным кодом из 4-х цифр) и обязательного значения, определяющего разрешенное действие;
- необязательный элемент `<Evidence>` — утверждения, которые подлежат авторизации. Этот элемент включает в себя элементы `<AssertionIDReference>` или `<Assertion>`;
- элемент `<Subject>` — сведения о клиенте.

SAML-запросы к центру аутентификации и SAML-ответы, содержащие утверждения, включаются в тело SOAP-сообщения с помощью элементов `<Request>` и `<Response>`.

Элемент `<Request>` имеет следующие атрибуты и вложенные элементы:

- обязательный атрибут `RequestID` — идентификатор запроса;
- обязательный атрибут `MajorVersion` — высший порядок версии спецификации SAML (значение — 1);

- обязательный атрибут `MinorVersion` — низший порядок версии спецификации SAML (значение — 1). Таким образом, при значениях `MajorVersion=1` и `MinorVersion=1` версия спецификации SAML будет 1.1;
- обязательный атрибут `IssueInstant` — время создания запроса;
- необязательный элемент `<RespondWith>` — тип ожидаемого ответа в виде имени SAML-утверждения (например, `saml:AttributeStatement`);
- необязательный элемент `<ds:Signature>` — цифровая подпись запроса.

А также какие-либо из элементов:

- `<SubjectQuery>` — запрос информации о клиенте;
- `<AuthenticationQuery>` — запрос информации об аутентификации клиента. Имеет дополнительный атрибут `AuthenticationMethod` (фильтрация по методу аутентификации);
- `<AttributeQuery>` — запрос информации об атрибутах клиента. Имеет дополнительный атрибут `Resource` (фильтрация по ресурсам) и необязательный вложенный элемент `<AttributeDesignator>` (фильтрация по имени атрибута);
- `<AuthorizationDecisionQuery>` — запрос информации о правах клиента. Имеет дополнительный атрибут `Resource` и необязательные элементы `<Action>` и `<Evidence>`;
- `<AssertionIDReference>` — запрос по ссылке на утверждение;
- `<AssertionArtifact>` — запрос утверждения сервисом ресурса у центра аутентификации в профиле Browser/Artifact Profile.

Элемент `<Response>` имеет следующие атрибуты и вложенные элементы:

- обязательный атрибут `ResponseID` — идентификатор ответа;
- дополнительный атрибут `InResponseTo` — ссылка на идентификатор запроса;
- обязательный атрибут `MajorVersion` — высший порядок версии спецификации SAML (значение — 1);
- обязательный атрибут `MinorVersion` — низший порядок версии спецификации SAML (значение — 1). Таким образом, при значениях `MajorVersion=1` и `MinorVersion=1` версия спецификации SAML будет 1.1;
- обязательный атрибут `IssueInstant` — время создания ответа;
- дополнительный атрибут `Recipient` — получатель ответа;
- необязательный элемент `<ds:Signature>` — цифровая подпись ответа;
- обязательный элемент `<Status>` — статус выполнения соответствующего запроса. Вложенные элементы: `<StatusCode>` (обязательный элемент, указывает код статуса), `<StatusMessage>` (дополнительное сообщение) и `<StatusDetail>` (дополнительные детали). Элемент `<StatusCode>` может иметь вложенные элементы `<StatusCode>` (второстепенный код) и имеет обязательный атрибут `Value` с возможными значениями `samlp:Success`, `samlp:VersionMismatch` (ошибка версии),

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

samlp:Requester (ошибка на стороне запроса) и samlp:Responder (ошибка на стороне ответа), возможные значения второстепенного кода — samlp:RequestVersionTooHigh, samlp:RequestVersionTooLow, samlp:RequestVersionDeprecated, samlp:TooManyResponses (ошибочное количество элементов), samlp:RequestDenied (запрет ответа), samlp:ResourceNotRecognized (неподдерживаемый ресурс). Пространство имен значений кода статуса — urn:oasis:names:tc:SAML:1.0:protocol;

- необязательный элемент <Assertion> — возвращаемое утверждение.

SAML 2.0

Спецификация SAML 2.0, дополнительно к сценарию, описывающему SSO Web-взаимодействие между клиентом, центром аутентификации и сервисом ресурса, определяет сценарий, в котором клиент проходит аутентификацию на одном сервисе ресурса, затем связывается с другим сервисом ресурса, который является партнером первого сервиса. Так как клиент уже прошел на первом сервисе аутентификацию, то второй сервис запрашивает у клиента разрешение на совместное использование его входа с первым сервисом. Далее клиент снова соединяется с первым сервисом, генерирующим псевдоним для клиента, ассоциированный с первоначальными данными клиента. После этого клиент передает второму сервису SAML-утверждение, содержащее информацию о псевдониме клиента. Теперь клиент может проходить аутентификацию на втором сервисе, используя свои первоначальные данные для первого сервиса, т. к. они ассоциированы с псевдонимом, позволяющим доступ ко второму сервису.

Таким образом, спецификация SAML 2.0 обеспечивает поддержку технологии интегрированной идентичности (federated identity) с определением понятия псевдонима — скрытого идентификатора, представляющего клиента для партнерского сервиса ресурсов.

Спецификация SAML 2.0, по сравнению со спецификацией SAML 1.1, определяет более широкий набор профилей, описывающих реализацию SAML, включающий в себя:

- профили технологии единого входа SSO:
  - Web Browser SSO Profile;
  - Enhanced Client or Proxy (ECP) Profile;
  - Identity Provider Discovery Profile;
  - Single Logout Profile;
  - Name Identifier Management Profile;
- Artifact Resolution Profile;
- Assertion Query/Request Profile;
- Name Identifier Mapping Profile;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- SAML Attribute Profiles — профили атрибутов:
  - Basic Attribute Profile;
  - X.500/LDAP Attribute Profile;
  - UUID Attribute Profile;
  - DCE PAC Attribute Profile;
  - XACML Attribute Profile.

Профиль Web Browser SSO Profile описывает сценарий, в котором клиент делает HTTP-запрос к защищенному Web-ресурсу. Сервис защищенного Web-ресурса получает адрес центра аутентификации и перенаправляет клиентский запрос, содержащий теперь уже SAML-элемент `<AuthnRequest>`, к центру аутентификации с использованием одного из протоколов: HTTP Redirect, HTTP POST или HTTP Artifact. После того как клиент проходит аутентификацию, центр аутентификации посылает сообщение, содержащее SAML-элемент `<Response>` приложению клиента, которое перенаправляет его сервису ресурса с помощью протокола HTTP POST или HTTP Artifact (протокол HTTP Redirect не используется, т. к. длина URL-строки при этом будет слишком большой). После получения сообщения сервис ресурса назначает права клиенту.

Профиль Enhanced Client or Proxy (ECP) Profile описывает сценарий, в котором клиент также делает HTTP-запрос к защищенному Web-ресурсу. Далее сервис защищенного ресурса создает сообщение, содержащее SAML-элемент `<AuthnRequest>`, и посылает его клиенту. Клиент, в свою очередь, отправляет данное сообщение центру аутентификации с помощью протокола SOAP. Центр аутентификации производит аутентификацию клиента и посылает ему сообщение, содержащее SAML-элемент `<Response>`, которое клиентское приложение передает сервису ресурса, назначающего клиенту права. Таким образом, в данном профиле клиент сам знает адрес центра аутентификации. При этом обмен сообщениями между клиентом и сервисом ресурса осуществляется с использованием протокола Reverse SOAP (PAOS).

Протокол Reverse SOAP (PAOS) отличается от протокола SOAP тем, что SOAP-запрос в протоколе PAOS передается в HTTP-ответе, а SOAP-ответ — в HTTP-запросе.

ECP-клиент делает HTTP-запрос сервису ресурса. Сервис ресурса посылает HTTP-ответ, содержащий SOAP-запрос `<AuthnRequest>`, обратно ECP-клиенту. В результате аутентификации клиента центром аутентификации, ECP-клиент снова посылает HTTP-запрос, содержащий SOAP-ответ `<Response>`, сервису ресурса. Поэтому протокол PAOS и называется "обратным" SOAP.

Профиль Identity Provider Discovery Profile используется сервисом ресурса для поиска подходящего центра для аутентификации клиента в случае, если система включает в себя несколько центров аутентификации. URI-идентификатор центра аутентификации сохраняется в виде значения cookie с именем "_saml_idp" при аутентификации клиента. Сервис ресурса, в случае необходимости обращения к

центру аутентификации, запрашивает сервис, отвечающий за управление и хранение cookie, и получает адрес центра, который произвел аутентификацию нужного клиента.

Профиль Single Logout Profile применяется в случае, если при взаимодействии клиента, центра аутентификации и сервиса ресурса центр аутентификации отвечает за создание сессии с клиентом и является инициатором создания сессии сервиса ресурса с клиентом. В данном профиле клиенту или администратору дается возможность инициировать окончание сессии. При этом центру аутентификации отправляется сообщение, содержащее SAML-элемент `<LogoutRequest>`, в ответ на которое центр аутентификации посылает сообщение, содержащее SAML-элемент `<LogoutResponse>`.

Профиль Name Identifier Management Profile описывает обмен сохраняемыми идентификаторами клиента (псевдонимами) центра аутентификации с сервисом ресурса. Данный обмен включает в себя уведомления центром аутентификации сервиса ресурса об изменениях псевдонима, возможность создания сервисом ресурса нового псевдонима для клиента, уведомления сервисом ресурса об окончании действия псевдонима. Управление псевдонимами в данном профиле осуществляется путем обмена сообщениями, содержащими SAML-элементы `<ManageNameIDRequest>` и `<ManageNameIDResponse>`.

Профиль Artifact Resolution Profile описывает обмен артефактами, содержащими ссылки на SAML-утверждения. Отправка артефакта выполняется с помощью сообщения с SAML-элементом `<ArtifactResolve>`. После обработки данного сообщения в ответ посылается сообщение с SAML-элементом `<ArtifactResponse>`, включающим в себя информацию о статусе обработки артефакта.

Профиль Assertion Query/Request Profile описывает запрос утверждений по ссылке или информации, содержащейся в утверждениях, с помощью SAML-элементов `<AssertionIDRequest>`, `<SubjectQuery>`, `<AuthnQuery>`, `<AttributeQuery>`, `<AuthzDecisionQuery>`. Сообщение, посылаемое в ответ, содержит SAML-элемент `<Response>`.

Профиль Name Identifier Mapping Profile предназначен для ассоциации клиента с несколькими идентификаторами. Запрос альтернативного идентификатора для клиента осуществляется центру аутентификации с помощью сообщения, содержащего SAML-элемент `<NameIDMappingRequest>`. В ответ центр аутентификации посылает сообщение с SAML-элементом `<NameIDMappingResponse>`.

Профили атрибутов описывают форматы XML-представления атрибутов, ассоциированных с клиентом.

Профиль Basic Attribute Profile имеет следующий формат:

```
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
  Name="urn:. . .">
  <saml:AttributeValue xsi:type="xs:string">. . .</saml:AttributeValue>
</saml:Attribute>
```

Профиль X.500/LDAP Attribute Profile имеет следующий формат:

```
<saml:Attribute
  xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500"
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:. . ." FriendlyName=". . .">
  <saml:AttributeValue xsi:type="xs:string"
    x500:Encoding="LDAP">. . .</saml:AttributeValue>
</saml:Attribute>
```

Профиль UUID (Universally Unique Identifiers) Attribute Profile имеет следующий формат:

```
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:uuid:. . ."
  FriendlyName=". . .">
  <saml:AttributeValue xsi:type="xs:integer">. . .</saml:AttributeValue>
</saml:Attribute>
```

Профиль DCE PAC Attribute Profile имеет следующий формат:

```
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:oasis:names:tc:SAML:2.0:profiles:attribute:DCE:realm">
  <saml:AttributeValue xsi:type="dce:DCEValueType"
    dce:FriendlyName=". . .">
    urn:uuid:. . .
  </saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:oasis:names:tc:SAML:2.0:profiles:attribute:DCE:principal">
  <saml:AttributeValue xsi:type="dce:DCEValueType"
     dce:FriendlyName=". . .">
     urn:uuid:. . .
  </saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:oasis:names:tc:SAML:2.0:profiles:attribute:DCE:primary-group">
  <saml:AttributeValue xsi:type="dce:DCEValueType"
    dce:FriendlyName=". . .">
    urn:uuid:. . .
  </saml:AttributeValue>
</saml:Attribute>
```

Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3


```
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:oasis:names:tc:SAML:2.0:profiles:attribute:DCE:groups">
  <saml:AttributeValue xsi:type="dce:DCEValueType"
    dce:FriendlyName=". . .">
    urn:uuid:. . .
  </saml:AttributeValue>
  <saml:AttributeValue xsi:type="dce:DCEValueType"
    dce:FriendlyName=". . .">
    urn:uuid:. . .
  </saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:oasis:names:tc:SAML:2.0:profiles:attribute:DCE:foreign-
groups">
  <saml:AttributeValue xsi:type="dce:DCEValueType"
   dce:FriendlyName=". . ."
   dce:Realm="urn:uuid:. . . ">
   urn:uuid:. . .
  </saml:AttributeValue>
</saml:Attribute>
```

Профиль XACML (eXtensible Access Control Markup Language) Attribute Profile имеет следующий формат:

```
<saml:Attribute
  xmlns:xacmlprof="urn:oasis:names:tc:SAML:2.0:profiles:attribute:XACML"
  xmlns:ldapprof="urn:oasis:names:tc:SAML:2.0:profiles:attribute:LDAP"
  xacmlprof:DataType="http://www.w3.org/2001/XMLSchema#string"
  ldapprof:Encoding="LDAP"
  NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
  Name="urn:. . ." FriendlyName=". . .">
  <saml:AttributeValue xsi:type="xs:string">. . .</saml:AttributeValue>
</saml:Attribute>
```

Корневым элементом XML-данных, представляющих утверждение, так же как и в спецификации SAML 1.1, является элемент `<Assertion>` (пространство имен `urn:oasis:names:tc:SAML:2.0:assertion`), имеющий следующие атрибуты и вложенные элементы:

- обязательный атрибут `Version` — версия спецификации (2.0). В SAML 1.1 — это атрибуты `MajorVersion` и `MinorVersion`;
- обязательный атрибут `ID` — идентификатор утверждения. В SAML 1.1 — это атрибут `AssertionID`;

- обязательный атрибут `IssueInstant` — дата и время создания утверждения;
- обязательный элемент `<Issuer>` — имя стороны, создавшей утверждение. В SAML 1.1 — это обязательный атрибут `Issuer`. Элемент `<Issuer>` имеет дополнительные атрибуты `NameQualifier` (сторона, определяющая имя), `SPNameQualifier` (имя сервиса, определяющего имя), `Format` (формат имени), `SPProvidedID` (идентификатор сервиса, создавшего утверждение). Атрибут `Format` имеет следующие возможные значения:
  - `urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified` — интерпретация зависит от конкретной реализации;
  - `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress` — формат адреса электронной почты;
  - `urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName` — формат элемента `<ds:X509SubjectName>` спецификации XML Signature;
  - `urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName` — формат "DomainName\UserName";
  - `urn:oasis:names:tc:SAML:2.0:nameid-format:Kerberos` — формат `name[/instance]@REALM`;
  - `urn:oasis:names:tc:SAML:2.0:nameid-format:entity` — формат URI, включающий значения атрибутов `NameQualifier`, `SPNameQualifier`, `SPProvidedID`, которые должны при этом отсутствовать, по умолчанию;
  - `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` — формат сохраняемого псевдонима;
  - `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` — формат несохраняемого псевдонима;
- дополнительный элемент `<Conditions>` — условия действительности утверждения;
- дополнительный элемент `<Advice>` — дополнительная информация об утверждении, например дополнительные утверждения, представленные элементами `<Assertion>` или ссылками на утверждения, представленные элементами `<AssertionIDReference>`. В отличие от SAML 1.1 дополнительные утверждения могут быть также представлены элементами `<AssertionURIRef>` и `<EncryptedAssertion>`. Элемент `<AssertionIDReference>` содержит значение атрибута `ID`, а элемент `<AssertionURIRef>` — строку HTTP-запроса с параметром `ID` — значением атрибута `ID`. Элемент `<EncryptedAssertion>` содержит зашифрованное утверждение в соответствии со спецификацией XML Encryption с вложенными элементами `<xenc:EncryptedData>` и `<xenc:EncryptedKey>` (пространство имен `http://www.w3.org/2001/04/xmlenc#`);
- дополнительный элемент `<ds:Signature>` (пространство имен `xmlns:ds="http://www.w3.org/2000/09/xmldsig#"`) — цифровая подпись XML Signature;
- дополнительный элемент `<Subject>` — описание клиента утверждения.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

А также какие-либо из элементов:

- элемент `<Statement>` — элементы расширения спецификации SAML, описывающие утверждение. Элемент `<SubjectStatement>` в SAML 2.0 отсутствует;
- элемент `<AuthenticationStatement>` описывает утверждение того, что клиент прошел аутентификацию;
- элемент `<AuthorizationDecisionStatement>` описывает утверждение того, что клиент получил определенные права доступа к запрашиваемым ресурсам;
- элемент `<AttributeStatement>` описывает утверждение того, что клиент ассоциируется с определенными атрибутами (имя/значение), влияющими на права доступа, которые сервис ресурса выдает клиенту.

Элемент `<Conditions>` описывает условия действительности утверждения с помощью следующих атрибутов и вложенных элементов:

- дополнительный атрибут `NotBefore` — время, начиная с которого действует утверждение;
- дополнительный атрибут `NotOnOrAfter` — время, до которого действует утверждение;
- необязательный элемент `<Condition>` — элементы расширения спецификации SAML, описывающие дополнительные условия;
- необязательный элемент `<AudienceRestriction>` (в SAML 1.1 это элемент `<AudienceRestrictionCondition>`) — стороны, для которых предназначено данное утверждение, описываемые с помощью вложенных элементов `<Audience>`, содержащих URI-идентификаторы сторон;
- необязательный элемент `<OneTimeUse>` (в SAML 1.1 это элемент `<DoNotCacheCondition>`) указывает, что утверждение не является повторно используемым;
- необязательный элемент `<ProxyRestriction>` (в SAML 1.1 этого элемента нет) накладывает ограничения на получающую утверждение сторону, которая сама является стороной, формирующей новое утверждение. Элемент `<ProxyRestriction>` имеет дополнительный атрибут `Count` — максимальное количество создаваемых производных утверждений и дополнительные элементы `<Audience>`, указывающие стороны, для которых предназначаются производные утверждения.

Элемент `<Subject>` описывает клиента, которому выдано данное утверждение, используя вложенные элементы `<BaseID>`, `<NameID>` или `<EncryptedID>` (в SAML 1.1 это `<NameIdentifier>`) и `<SubjectConfirmation>`.

Элемент `<BaseID>` является базовым для элементов, определяющих идентификацию. Элемент `<BaseID>` имеет дополнительные атрибуты `NameQualifier` — сторона, определяющая имя клиента, и `SPNameQualifier` — имя сервиса, определяющего имя клиента.

Элемент `<NameID>` содержит имя клиента и имеет следующие атрибуты:

- дополнительный атрибут `NameQualifier` — сторона, определяющая имя клиента;
- дополнительный атрибут `SPNameQualifier` — имя сервиса;
- дополнительный атрибут `Format` — URI-идентификатор формата имени клиента (см. элемент `<Issuer>`), по умолчанию `urn:oasis:names:tc:SAML:1.0:nameid-format:unspecified`;
- дополнительный атрибут `SPProvidedID` — идентификатор сервиса.

Элемент `<EncryptedID>` описывает зашифрованный идентификатор с помощью вложенных элементов `<xenc:EncryptedData>` и `<xenc:EncryptedKey>`.

Элемент `<SubjectConfirmation>` содержит информацию, дающую возможность сервису ресурса подтвердить аутентификацию клиента с помощью вложенных элементов:

- обязательный атрибут `Method` (в SAML 1.1 это обязательный элемент `<ConfirmationMethod>`) содержит URI-идентификатор протокола, используемого для подтверждения аутентификации клиента. Спецификация предопределяет следующие URI-идентификаторы:
  - `urn:oasis:names:tc:SAML:2.0:cm:holder-of-key` — аутентификация производится с помощью ключа, информация о котором содержится во вложенном элементе `<ds:KeyInfo>`. Это может быть, например, публичный ключ цифровой подписи, содержащейся в элементе `<SubjectConfirmationData>`;
  - `urn:oasis:names:tc:SAML:2.0:cm:sender-vouches` — информация, помогающая аутентифицировать клиента;
  - `urn:oasis:names:tc:SAML:2.0:cm:bearer` — клиент является стороной, передающей утверждение;

  Примечание

  URI-идентификатор `urn:oasis:names:tc:SAML:1.0:cm:artifact` в SAML 2.0 отсутствует.

- необязательные элементы `<BaseID>`, `<NameID>` или `<EncryptedID>`;
- необязательный элемент `<SubjectConfirmationData>` содержит данные, используемые протоколом подтверждения аутентификации клиента. Элемент `<SubjectConfirmationData>` имеет следующие дополнительные атрибуты и вложенные элементы:
  - `NotBefore` — время, до которого подтверждение не производится;
  - `NotOnOrAfter` — время, после которого подтверждение не производится;
  - `Recipient` — URI-идентификатор получателя утверждения;
  - `InResponseTo` — ID-сообщения, в ответ на которое представляется утверждение;
  - `Address` — IP-адрес, представляющий утверждение;
  - `<ds:KeyInfo>` — информация о ключе.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<AuthnStatement>` (в SAML 1.1 это элемент `<AuthenticationStatement>`) описывает утверждение аутентификации клиента, используя следующие атрибуты и вложенные элементы:

> ***Примечание***
> Обязательный атрибут `AuthenticationMethod` и необязательный элемент `<AuthorityBinding>` в SAML 2.0 отсутствуют. Элемент `<Subject>` включен непосредственно в элемент `<Assertion>`.

- обязательный атрибут `AuthenticationInstant` — время, когда клиент прошел аутентификацию;
- необязательный элемент `<SubjectLocality>` — локализация клиента (DNS-имя и IP-адрес), указываемая с помощью атрибутов `Address` и `DNSName` (в SAML 1.1 `IPAddress` и `DNSAddress`);
- дополнительный атрибут `SessionIndex` (только в SAML 2.0) — индекс сессии между клиентом и центром аутентификации;
- дополнительный атрибут `SessionNotOnOrAfter` (только в SAML 2.0) — время окончания сессии;
- обязательный элемент `<AuthnContext>` (только в SAML 2.0) — контекст аутентификации.

Контекст аутентификации — это дополнительная информация, которая может потребоваться сервису ресурса. Например, это может быть информация о механизме идентификации и аутентификации клиента, хранении и защите прав клиента и т. д. Информация контекста аутентификации содержится в элементе `<AuthnContextDecl>`. Кроме того, информация контекста аутентификации может быть сгруппирована по классам контекста аутентификации, URI-идентификаторы которых указываются с помощью элемента `<AuthnContextClassRef>`.

Таким образом, элемент `<AuthnContext>` может иметь вложенные элементы `<AuthnContextClassRef>`, `<AuthnContextDecl>` или `<AuthnContextDeclRef>` (содержит URI-ссылку на элемент `<AuthnContextDecl>`), а также элементы `<AuthenticatingAuthority>`, которые указывают идентификаторы центров аутентификации клиента.

Элемент `<AuthnContextDecl>` может содержать следующие элементы:

- `<Identification>` — характеристики идентификации клиента;
- `<TechnicalProtection>` — характеристики защиты ключа аутентификации;
- `<OperationalProtection>` — характеристики защиты процесса аутентификации;
- `<AuthnMethod>` — метод аутентификации;
- `<GoverningAgreements>` — характеристики обязательств и гарантий аутентификации;
- `<Extension>` — расширение спецификации.

Спецификация предопределяет следующие URI-идентификаторы классов контекста аутентификации:

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocol` — клиент проходит аутентификацию, используя IP-адрес;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:InternetProtocolPassword` — клиент проходит аутентификацию, используя IP-адрес в дополнение к логину-паролю;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:Kerberos` — клиент проходит аутентификацию, используя Kerberos-билеты;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:MobileOneFactorUnregistered` — аутентификацию проходит мобильное устройство, а не пользователь;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorUnregistered` — аутентификацию не проходит ни пользователь, ни мобильное устройство;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:MobileOneFactorContract` — аутентификация с использованием цифровой подписи мобильного устройства;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:MobileTwoFactorContract` — аутентификация с использованием цифровой подписи мобильного устройства и идентификатора пользователя;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:Password` — клиент проходит аутентификацию с помощью пароля по HTTP-протоколу;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport` — клиент проходит аутентификацию с помощью пароля по защищенному протоколу;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession` — аутентификация клиента производится с помощью предыдущей сессии;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:X509` — клиент проходит аутентификацию, используя цифровую подпись и X.509-ключ;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:PGP` — клиент проходит аутентификацию, используя цифровую подпись и PGP-ключ;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:SPKI` — клиент проходит аутентификацию, используя цифровую подпись и SPKI-ключ;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:XMLDSig` — клиент проходит аутентификацию, используя цифровую подпись согласно спецификации XML Signature;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:Smartcard` — клиент проходит аутентификацию, используя смарт-карту;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:SmartcardPKI` — клиент проходит аутентификацию, используя смарт-карту и PIN-код;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:SoftwarePKI` — клиент проходит аутентификацию, используя X.509-сертификат;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:Telephony` — клиент проходит аутентификацию, используя телефонный номер, передаваемый по ADSL-протоколу;
- `urn:oasis:names:tc:SAML:2.0:ac:classes:NomadTelephony` — клиент проходит аутентификацию, используя телефонную карту;

- urn:oasis:names:tc:SAML:2.0:ac:classes:PersonalTelephony — клиент проходит аутентификацию, используя телефонный номер и пользовательский идентификатор, передаваемые по ADSL-протоколу;
- urn:oasis:names:tc:SAML:2.0:ac:classes:AuthenticatedTelephony — клиент проходит аутентификацию, используя телефонный номер, пользовательский идентификатор и пароль;
- urn:oasis:names:tc:SAML:2.0:ac:classes:SecureRemotePassword — аутентификация производится с помощью протокола Secure Remote Password Protocol (SRP);
- urn:oasis:names:tc:SAML:2.0:ac:classes:TLSClient — клиент проходит аутентификацию, используя сертификат и протокол SSL/TLS;
- urn:oasis:names:tc:SAML:2.0:ac:classes:TimeSyncToken — аутентификация производится с помощью синхронизированных по времени маркеров защиты;
- urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified — неопределенный способ аутентификации.

Элемент <AttributeStatement> описывает ассоциацию клиента с атрибутами с помощью вложенных элементов <Attribute> или <EncryptedAttribute>. Элемент <Attribute> имеет обязательный атрибут Name, необязательные атрибуты NameFormat и FriendlyName (в SAML 1.1 атрибуты AttributeNamespace и AttributeName) и вложенный элемент <AttributeValue> (значение атрибута).

Элемент <EncryptedAttribute> (только в SAML 2.0) содержит элементы <xenc:EncryptedData> и <xenc:EncryptedKey>.

Атрибут NameFormat указывает URI-идентификатор формата имени атрибута. Спецификация предопределяет следующие URI-идентификаторы:

- urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified — интерпретация имени атрибута зависит от конкретной реализации, по умолчанию;
- urn:oasis:names:tc:SAML:2.0:attrname-format:uri — формат URI;
- urn:oasis:names:tc:SAML:2.0:attrname-format:basic — тип xs:Name (пространство имен http://www.w3.org/2001/XMLSchema).

Атрибут FriendlyName указывает упрощенное имя атрибута.

Элемент <AuthzDecisionStatement> (в SAML 1.1 это элемент <Authorization DecisionStatement>) содержит информацию о правах клиента, используя следующие атрибуты и вложенные элементы:

- обязательный атрибут Resource — URI-идентификатор ресурса, относительно которого назначаются права клиенту;
- обязательный атрибут Decision — права клиента, возможные значения: Permit, Deny, Indeterminate;
- обязательный элемент <Action> — действия, разрешенные для ресурсов, указываемые с помощью дополнительного атрибута Namespace (пространство имен действия):

- urn:oasis:names:tc:SAML:1.0:action:rwedc — соответствует действиям Read, Write, Execute, Delete, Control;
- urn:oasis:names:tc:SAML:1.0:action:rwedc-negation (по умолчанию) — соответствует действиям Read, Write, Execute, Delete, Control, ~Read, ~Write, ~Execute, ~Delete и ~Control;
- urn:oasis:names:tc:SAML:1.0:action:ghpp — соответствует действиям GET, HEAD, PUT, POST;
- urn:oasis:names:tc:SAML:1.0:action:unix — соответствует действиям extended, user, group, world, представленным кодом из 4-х цифр;

❒ необязательный элемент <Evidence> — утверждения, которые подлежат авторизации. Элемент <Evidence> может включать в себя элементы <AssertionIDReference>, <AssertionURIRef>, <Assertion>, <EncryptedAssertion>.

SAML-запросы к центру аутентификации и SAML-ответы, содержащие утверждения, включаются в тело SOAP-сообщения с помощью элементов, представляющих запросы, и элементов, представляющих ответы.

Элементы, представляющие запросы, имеют следующие атрибуты и вложенные элементы:

❒ обязательный атрибут ID (в SAML 1.1 это атрибут RequestID) — идентификатор запроса;

❒ обязательный атрибут Version (в SAML 1.1 это атрибуты MajorVersion и MinorVersion) — версия спецификации SAML (значение — 2.0);

❒ обязательный атрибут IssueInstant — время создания запроса;

❒ необязательный атрибут Destination (SAML 2.0) — URI-адрес SAML-запроса;

❒ необязательный атрибут Consent (SAML 2.0) — по умолчанию urn:oasis:names:tc:SAML:2.0:consent:unspecified, указывает способ получения согласия пользователя на отправку запроса. Возможные значения:

- urn:oasis:names:tc:SAML:2.0:consent:unspecified — неопределенный способ получения согласия;
- urn:oasis:names:tc:SAML:2.0:consent:obtained — согласие получается запрашивающей стороной;
- urn:oasis:names:tc:SAML:2.0:consent:prior — согласие получается запрашивающей стороной предварительно;
- urn:oasis:names:tc:SAML:2.0:consent:current-implicit — неявное согласие в контексте общего согласия;
- urn:oasis:names:tc:SAML:2.0:consent:current-explicit — явный запрос согласия;
- urn:oasis:names:tc:SAML:2.0:consent:unavailable — запрашивающая сторона не получает согласие;
- urn:oasis:names:tc:SAML:2.0:consent:inapplicable — согласие не требуется;

Примечание

Необязательный элемент `<RespondWith>` в SAML 2.0 отсутствует.

- необязательный элемент `<ds:Signature>` — цифровая подпись запроса;
- необязательный элемент `<saml:Issuer>` (SAML 2.0) — сторона, генерирующая запрос;
- необязательный элемент `<Extensions>` — расширение спецификации.

Запрос может быть представлен каким-либо из следующих элементов, которые могут иметь дополнительно к вышеперечисленным атрибуты и вложенные элементы:

- `<SubjectQuery>` — запрос информации о клиенте, содержит вложенный элемент `<saml:Subject>`;
- `<AuthnQuery>` (в SAML 1.1 `<AuthenticationQuery>`) — запрос информации об аутентификации клиента. Имеет дополнительный атрибут `SessionIndex` — фильтрация по сессии, и дополнительный вложенный элемент `<RequestedAuthnContext>` — фильтрация по контексту аутентификации (в SAML 1.1 `AuthenticationMethod`);
- `<RequestedAuthnContext>` (SAML 2.0) — запрос контекста аутентификации. Имеет вложенные элементы `<saml:AuthnContextClassRef>` или `<saml:AuthnContextDeclRef>` и дополнительный атрибут `Comparison` — метод оценки требуемого контекста. Возможные значения атрибута `Comparison`: `"exact"` (по умолчанию, точно совпадает с указанным), `"minimum"` (возвращаемый контекст по меньшей мере сильнее, чем указанный), `"better"` (контекст сильнее, чем указанный) и `"maximum"` (возвращаемый контекст максимально строже, чем указанный);
- `<AttributeQuery>` — запрос информации об атрибутах клиента. Вложенные элементы `<saml:Attribute>` указывают атрибуты, чьи значения должны быть возвращены (в SAML 1.1 это дополнительный атрибут `Resource` — фильтрация по ресурсам, и необязательный вложенный элемент `<AttributeDesignator>` — фильтрация по имени атрибута);
- `<AuthzDecisionQuery>` (в SAML 1.1 `<AuthorizationDecisionQuery>`) — запрос информации о правах клиента. Имеет дополнительный атрибут `Resource` и необязательные элементы `< saml:Action>` и `< saml:Evidence>`;
- `<AssertionIDRequest>` (в SAML 1.1 `<AssertionIDReference>`) — запрос по ссылке на утверждение. Имеет вложенный элемент `<saml:AssertionIDRef>`;

Примечание

Элемент `<AssertionArtifact>` в SAML 2.0 отсутствует.

- `<AuthnRequest>` — запрос клиентом утверждений у центра аутентификации. Дополнительные вложенные элементы и атрибуты:
  - элемент `<saml:Subject>` описывает клиента, для которого должны быть созданы утверждения;
  - элемент `<NameIDPolicy>` накладывает ограничения на имя клиента. Дополнительный атрибут `Format` указывает URI-идентификатор формата имени, воз-

можные значения: unspecified (по умолчанию), emailAddress, X509SubjectName, WindowsDomainQualifiedName, kerberos, entity, persistent, transient и encrypted (пространство имен urn:oasis:names:tc:SAML:2.0:nameid-format). Дополнительные атрибуты: SPNameQualifier (имя сервиса, определяющего имя клиента) и AllowCreate (true/false, разрешает ли отвечающей стороне создавать новый идентификатор для клиента);

- элемент <saml:Conditions> — предполагаемые условия для возвращаемых утверждений;
- элемент <RequestedAuthnContext> — требования к контексту аутентификации;
- элемент <Scoping> определяет отвечающие стороны, которым доверяет запрашивающая сторона. Имеет дополнительный атрибут ProxyCount (количество перенаправлений между центром, получившим запрос, и центром, совершившим аутентификацию) и дополнительные элементы <IDPList> (список доверенных центров аутентификации) и <RequesterID> (список идентификаторов запрашивающей стороны). Элемент <IDPList> содержит элементы <IDPEntry> (информация о центре аутентификации) и дополнительный элемент <GetComplete> (URI-ссылка на дополнительную информацию). Элемент <IDPEntry> имеет обязательный атрибут ProviderID (идентификатор центра аутентификации) и дополнительные атрибуты Name (имя центра аутентификации) и Loc (URI-адрес конечной точки);
- атрибут ForceAuthn — если true, тогда центр аутентификации производит аутентификацию клиента, даже если он уже ее перед этим прошел, по умолчанию — false;
- атрибут IsPassive — если true, тогда центр аутентификации сам не взаимодействует с интерфейсом пользователя, по умолчанию — false;
- атрибут AssertionConsumerServiceIndex — идентификатор, по которому должен быть направлен ответ;
- атрибут AssertionConsumerServiceURL — URI-адрес, по которому должен быть направлен ответ;
- атрибут ProtocolBinding — URI-идентификатор протокола, с помощью которого должен быть направлен ответ. Возможные значения (пространство имен urn:oasis:names:tc:SAML:2.0:bindings): SOAP, PAOS (Reverse SOAP в профиле ECP SSO), HTTP-Redirect (сообщение передается по HTTP-протоколу, закодированное в URL-строке), URL-Encoding:DEFLATE (сообщение кодируется в URL-строке, используя механизм сжатия DEFLATE), HTTP-POST (сообщение передается методом HTTP POST), HTTP-Artifact (передача ссылки на утверждение), URI (передача утверждения по идентификатору);
- атрибут AttributeConsumingServiceIndex — идентификатор информации, описывающей SAML-атрибуты, ожидаемые в ответе;
- атрибут ProviderName — короткое имя запрашивающей стороны;

- `<ArtifactResolve>` — отправка артефакта, содержащего ссылку на утверждение. Вложенный элемент `<Artifact>` содержит значение, включающее в себя ссылку на утверждение;
- `<ManageNameIDRequest>` — сообщение об изменении идентификатора или о прекращении действия идентификатора. Обязательный вложенный элемент `<saml:NameID>` или `<saml:EncryptedID>` — текущий идентификатор, и `<NewID>` или `<NewEncryptedID>` (новый идентификатор) или `<Terminate>` (указатель о прекращении действия идентификатора);
- `<LogoutRequest>` — запрос на завершение сессии. Дополнительные атрибуты: `NotOnOrAfter` (время, до которого действует запрос) и `Reason` (причина завершения сессии, например: `urn:oasis:names:tc:SAML:2.0:logout:user` — клиент хочет завершить сессию; `urn:oasis:names:tc:SAML:2.0:logout:admin` — администратор хочет завершить сессию), а также дополнительные вложенные элементы `<saml:BaseID>` или `<saml:NameID>` или `<saml:EncryptedID>` (идентификатор клиента) и `<SessionIndex>` (идентификатор сессии);
- `<NameIDMappingRequest>` — запрос альтернативного идентификатора клиента. Вложенные элементы: `<saml:BaseID>` либо `<saml:NameID>` или `<saml:EncryptedID>` (идентификатор клиента) и `<NameIDPolicy>` (формат требуемого идентификатора).

Элементы, представляющие ответы, имеют следующие атрибуты и вложенные элементы:

- обязательный атрибут `ID` (в SAML 1.1 ResponseID) — идентификатор ответа;
- дополнительный атрибут `InResponseTo` — ссылка на идентификатор запроса;
- обязательный атрибут `Version` (в SAML 1.1 это атрибуты `MajorVersion` и `MinorVersion`) — версия спецификации SAML (значение — 2.0);
- обязательный атрибут `IssueInstant` — время создания ответа;
- дополнительный атрибут `Destination` (в SAML 1.1 Recipient) — URI получателя ответа;
- необязательный элемент `<ds:Signature>` — цифровая подпись ответа;
- необязательный атрибут `Consent` (SAML 2.0) — по умолчанию `urn:oasis:names:tc:SAML:2.0:consent:unspecified`, указывает способ получения согласия пользователя на отправку ответа;
- необязательный элемент `<saml:Issuer>` (SAML 2.0) — сторона, генерирующая ответ;
- необязательный элемент `<Extensions>` — расширение спецификации;
- обязательный элемент `<Status>` — статус выполнения соответствующего запроса. Вложенные элементы: `<StatusCode>` (обязательный элемент указывает код статуса), `<StatusMessage>` (дополнительное сообщение) и `<StatusDetail>` (дополнительные детали). Элемент `<StatusCode>` может иметь вложенные элементы `<StatusCode>` (второстепенный код) и имеет обязательный атрибут `Value` с возможными значениями:

- urn:oasis:names:tc:SAML:2.0:status:Success — запрос обработан успешно;
- urn:oasis:names:tc:SAML:2.0:status:Requester — ошибка на стороне запроса;
- urn:oasis:names:tc:SAML:2.0:status:Responder — ошибка на стороне ответа;
- urn:oasis:names:tc:SAML:2.0:status:VersionMismatch — ошибка версии.

Значения второстепенного кода:

- urn:oasis:names:tc:SAML:2.0:status:AuthnFailed — ошибка аутентификации клиента;
- urn:oasis:names:tc:SAML:2.0:status:InvalidAttrNameOrValue — ошибка содержимого атрибута;
- urn:oasis:names:tc:SAML:2.0:status:InvalidNameIDPolicy — политика идентификатора не поддерживается;
- urn:oasis:names:tc:SAML:2.0:status:NoAuthnContext — ошибка запроса контекста аутентификации;
- urn:oasis:names:tc:SAML:2.0:status:NoAvailableIDP — ошибка списка доверенных центров аутентификации (элемент <IDPList>);
- urn:oasis:names:tc:SAML:2.0:status:NoPassive — пассивная аутентификация невозможна;
- urn:oasis:names:tc:SAML:2.0:status:NoSupportedIDP — неподдерживаемый список доверенных центров аутентификации (элемент <IDPList>);
- urn:oasis:names:tc:SAML:2.0:status:PartialLogout — невозможность полного завершения сессии;
- urn:oasis:names:tc:SAML:2.0:status:ProxyCountExceeded — невозможность перенаправить запрос для аутентификации;
- urn:oasis:names:tc:SAML:2.0:status:RequestDenied — невозможность ответа на запрос;
- urn:oasis:names:tc:SAML:2.0:status:RequestUnsupported — неподдерживаемый запрос;
- urn:oasis:names:tc:SAML:2.0:status:RequestVersionDeprecated — неподдерживаемая версия протокола отвечающей стороной;
- urn:oasis:names:tc:SAML:2.0:status:RequestVersionTooHigh — версия протокола отвечающей стороны требует обновления;
- urn:oasis:names:tc:SAML:2.0:status:RequestVersionTooLow — версия протокола запрашивающей стороны требует обновления;
- urn:oasis:names:tc:SAML:2.0:status:ResourceNotRecognized — ошибка ресурса;
- urn:oasis:names:tc:SAML:2.0:status:TooManyResponses — ошибка структуры сообщения запроса;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `urn:oasis:names:tc:SAML:2.0:status:UnknownAttrProfile` — неподдерживаемый профиль атрибутов;
- `urn:oasis:names:tc:SAML:2.0:status:UnknownPrincipal` — неизвестный пользователь;
- `urn:oasis:names:tc:SAML:2.0:status:UnsupportedBinding` — неподдерживаемый протокол.

Ответ может быть представлен каким-либо из следующих элементов, которые могут иметь дополнительно к вышеперечисленным атрибуты и вложенные элементы:

- `<Response>` имеет необязательный вложенный элемент `<Assertion>` или `<saml:EncryptedAssertion>`, который содержит возвращаемое утверждение;
- `<ArtifactResponse>` — ответ на запрос `<ArtifactResolve>`;
- `<ManageNameIDResponse>` — ответ на запрос `<ManageNameIDRequest>`;
- `<LogoutResponse>` — ответ на запрос `<LogoutRequest>`;
- `<NameIDMappingResponse>` — ответ на запрос `<NameIDMappingRequest>`. Вложенные элементы `<saml:NameID>` или `<saml:EncryptedID>`.

Спецификация SAML 2.0 определяет способ дополнительного описания участников SAML-взаимодействия в виде XML-документов, содержащих SAML-метаданные взаимодействующих сторон. SAML-метаданные представляют собой конфигурационные данные о том, какие протоколы и профили, системы безопасности поддерживает конкретный участник SAML-взаимодействия. Применение SAML-метаданных упрощает развертывание и работу SAML-системы. При этом SAML-метаданные могут быть опубликованы различными способами, например с помощью идентификатора участника, содержащего URL-адрес документа SAML-метаданных.

Корневым элементом XML-документа SAML-метаданных является элемент `<EntitiesDescriptor>` или `<EntitiesDescriptor>` (пространство имен `urn:oasis:names:tc:SAML:2.0:metadata`), имеющий следующие атрибуты и вложенные элементы:

- дополнительный атрибут `ID` — идентификатор документа;
- дополнительный атрибут `validUntil` — время окончания действия метаданных;
- дополнительный атрибут `cacheDuration` — максимальное время кэширования метаданных;
- дополнительный атрибут `Name` — имя группы участников, для которых создаются метаданные;
- дополнительный элемент `<ds:Signature>` — цифровая подпись документа;
- дополнительный элемент `<Extensions>` содержит элементы расширения спецификации;
- обязательные элементы `<EntitiesDescriptor>` или `<EntityDescriptor>` содержат метаданные участников группы.

Элемент `<EntityDescriptor>` представляет метаданные одного участника с помощью следующих атрибутов и вложенных элементов:

- обязательный атрибут `entityID` — идентификатор участника;
- дополнительный атрибут `ID` — идентификатор участника в пределах группы;
- дополнительный атрибут `validUntil` — время окончания действия метаданных;
- дополнительный атрибут `cacheDuration` — максимальное время кэширования метаданных;
- дополнительный элемент `<ds:Signature>` — цифровая подпись метаданных;
- дополнительный элемент `<Extensions>` — элементы расширения спецификации;
- обязательный элемент `<RoleDescriptor>`, или `<IDPSSODescriptor>`, или `<SPSSODescriptor>`, или `<AuthnAuthorityDescriptor>`, или `<AttributeAuthorityDescriptor>`, или `<PDPDescriptor>`, или `<AffiliationDescriptor>`;
- дополнительный элемент `<Organization>` — идентификатор организации, к которой принадлежит участник. Вложенные элементы: `<Extensions>`, `<OrganizationName>`, `<OrganizationDisplayName>` и `<OrganizationURL>`;
- дополнительный элемент `<ContactPerson>` — контактная информация. Имеет обязательный атрибут `contactType` — тип контакта, возможные значения: `technical`, `support`, `administrative`, `billing` и `other`. Дополнительные вложенные элементы: `<Extensions>`, `<Company>`, `<GivenName>`, `<SurName>`, `<EmailAddress>` и `<TelephoneNumber>`;
- дополнительный элемент `<AdditionalMetadataLocation>` — местонахождение дополнительных метаданных.

Элемент `<RoleDescriptor>` содержит следующие атрибуты и элементы:

- дополнительный элемент `ID` — идентификатор документа;
- дополнительный атрибут `validUntil` — время окончания действия метаданных, содержащихся в элементе;
- дополнительный атрибут `cacheDuration` — максимальное время кэширования метаданных, содержащихся в элементе;
- обязательный атрибут `protocolSupportEnumeration` — перечень URI-идентификаторов протоколов, поддерживаемых участником;
- дополнительный атрибут `errorURL` — адрес дополнительной поддержки;
- дополнительный элемент `<ds:Signature>` — цифровая подпись метаданных, содержащихся в элементе;
- дополнительный элемент `<Extensions>` — элементы расширения спецификации;
- дополнительный элемент `<KeyDescriptor>` — информация о криптографических ключах участника. Имеет дополнительный атрибут `use` — использование ключа, возможные значения: `encryption` и `signing`, обязательный вложенный элемент

`<ds:KeyInfo>` (спецификация XMLSig) и дополнительный вложенный элемент `<EncryptionMethod>` (спецификация XMLEnc);

- дополнительный элемент `<Organization>` — идентификатор организации, к которой принадлежит участник. Вложенные элементы: `<Extensions>`, `<OrganizationName>`, `<OrganizationDisplayName>` и `<OrganizationURL>`;
- дополнительный элемент `<ContactPerson>` — контактная информация. Имеет обязательный атрибут `contactType` — тип контакта, возможные значения: `technical`, `support`, `administrative`, `billing` и `other`. Дополнительные вложенные элементы: `<Extensions>`, `<Company>`, `<GivenName>`, `<SurName>`, `<EmailAddress>` и `<TelephoneNumber>`.

Элемент `<IDPSSODescriptor>` содержит информацию о центре аутентификации клиента и имеет атрибуты и элементы элемента `<RoleDescriptor>`, а также дополнительно следующие атрибуты и элементы:

- дополнительный элемент `<ArtifactResolutionService>` — информация о конечной точке участника, поддерживающего профиль Artifact Resolution. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки), `index` (целочисленный идентификатор конечной точки) и дополнительный атрибут `isDefault` (`true`/`false`, указывает использование данной конечной точки по умолчанию);
- дополнительный элемент `<SingleLogoutService>` — информация о конечной точке участника, поддерживающего профиль Single Logout. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<ManageNameIDService>` — информация о конечной точке участника, поддерживающего профиль Name Identifier Management. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<NameIDFormat>` — URI-идентификатор формата псевдонима, поддерживаемый участником (пространство имен `urn:oasis:names:tc:SAML:1.1:nameid-format`). Возможные значения: `unspecified`, `emailAddress`, `X509SubjectName`, `WindowsDomainQualifiedName`, `kerberos`, `entity`, `persistent`, `transient`;
- дополнительный атрибут `WantAuthnRequestsSigned` — требование запросов `<samlp:AuthnRequest>`, получаемых участником, быть снабженными цифровой подписью, по умолчанию `false`;
- обязательный элемент `<SingleSignOnService>` — информация о конечной точке участника, поддерживающего профиль Authentication Request. Имеет обязатель-

ные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями) и `Location` (URI-адрес конечной точки);

- дополнительный элемент `<NameIDMappingService>` — информация о конечной точке участника, поддерживающего профиль Name Identifier Mapping. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями) и `Location` (URI-адрес конечной точки);
- дополнительный элемент `<AssertionIDRequestService>` — информация о конечной точке участника, поддерживающего профиль Assertion Request. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<AttributeProfile>` — URI-идентификатор профиля атрибутов, поддерживаемый участником;
- дополнительный элемент `<saml:Attribute>` — идентификатор атрибута, поддерживаемый участником.

Элемент `<SPSSODescriptor>` содержит информацию о сервисе ресурса и имеет атрибуты и элементы элемента `<RoleDescriptor>`, а также дополнительно следующие атрибуты и элементы:

- дополнительный элемент `<ArtifactResolutionService>` — информация о конечной точке участника, поддерживающего профиль Artifact Resolution. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки), `index` (целочисленный идентификатор конечной точки) и дополнительный атрибут `isDefault` (`true`/`false`, указывает использование данной конечной точки по умолчанию);
- дополнительный элемент `<SingleLogoutService>` — информация о конечной точке участника, поддерживающего профиль Single Logout. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<ManageNameIDService>` — информация о конечной точке участника, поддерживающего профиль Name Identifier Management. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<NameIDFormat>` — URI-идентификатор формата псевдонима, поддерживаемый участником (пространство имен `urn:oasis:names:tc:SAML:1.1:nameid-format`), возможные значения: `unspecified`, `emailAddress`, `X509SubjectName`, `WindowsDomainQualifiedName`, `kerberos`, `entity`, `persistent`, `transient`;

- дополнительный атрибут `AuthnRequestsSigned` указывает, что запрос `<samlp:AuthnRequest>`, посылаемый участником, будет подписан цифровой подписью, по умолчанию `false`;
- дополнительный атрибут `WantAssertionsSigned` указывает, что ответ `<saml:Assertion>`, получаемый участником, должен быть подписан цифровой подписью, по умолчанию `false`;
- обязательный элемент `<AssertionConsumerService>` — информация о конечной точке участника, поддерживающего профиль Authentication Request. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки), дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения), `index` (целочисленный идентификатор конечной точки) и дополнительный атрибут `isDefault` (`true`/`false`, указывает использование данной конечной точки по умолчанию);
- дополнительный элемент `<AttributeConsumingService>` — информация о сервисе, использующем SAML-атрибуты. Имеет обязательный атрибут `index` — целочисленный идентификатор элемента, и дополнительный атрибут `isDefault`, указывающий, что данный сервис используется по умолчанию, по умолчанию значение атрибута — `false`. Обязательные вложенные элементы: `<ServiceName>` — имя сервиса и `<RequestedAttribute>` (дополнительный атрибут `isRequired`, принимающий значения `true`/`false`, указывает необходимость атрибута) — атрибуты, необходимые сервису, а также дополнительный вложенный элемент `<ServiceDescription>` — описание сервиса.

Элемент `<AuthnAuthorityDescriptor>` содержит информацию об участнике, отвечающем на запросы `<samlp:AuthnQuery>`, и имеет атрибуты и элементы элемента `<RoleDescriptor>`, а также дополнительно следующие элементы:

- обязательный элемент `<AuthnQueryService>` — информация о конечной точке участника, поддерживающего запросы `<samlp:AuthnQuery>`. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<AssertionIDRequestService>` — информация о конечной точке участника, поддерживающего запросы `<samlp:AssertionIDRequest>`. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<NameIDFormat>` — URI-идентификатор формата псевдонима, поддерживаемый участником (пространство имен `urn:oasis:names:tc:SAML:1.1:nameid-format`), возможные значения: `unspecified`, `emailAddress`, `X509SubjectName`, `WindowsDomainQualifiedName`, `kerberos`, `entity`, `persistent`, `transient`.

Элемент `<PDPDescriptor>` содержит информацию об участнике, отвечающем на запросы `<samlp:AuthzDecisionQuery>`, и имеет атрибуты и элементы элемента `<RoleDescriptor>`, а также дополнительно следующие элементы:

- обязательный элемент `<AuthzService>` — информация о конечной точке участника, поддерживающего запросы `<samlp:AuthzDecisionQuery>`. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<AssertionIDRequestService>` — информация о конечной точке участника, поддерживающего запросы `<samlp:AssertionIDRequest>`. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<NameIDFormat>` — URI-идентификатор формата псевдонима, поддерживаемый участником (пространство имен `urn:oasis:names:tc:SAML:1.1:nameid-format`), возможные значения: `unspecified`, `emailAddress`, `X509SubjectName`, `WindowsDomainQualifiedName`, `kerberos`, `entity`, `persistent`, `transient`.

Элемент `<AttributeAuthorityDescriptor>` содержит информацию об участнике, отвечающем на запросы `<samlp:AttributeQuery>`, и имеет атрибуты и элементы элемента `<RoleDescriptor>`, а также дополнительно следующие элементы:

- обязательный элемент `<AttributeService>` — информация о конечной точке участника, поддерживающего запросы `<samlp:AttributeQuery>`. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<AssertionIDRequestService>` — информация о конечной точке участника, поддерживающего запросы `<samlp:AssertionIDRequest>`. Имеет обязательные атрибуты `Binding` (URI-идентификатор протокола обмена SAML-сообщениями), `Location` (URI-адрес конечной точки) и дополнительный атрибут `ResponseLocation` (URI-адрес, по которому должны отправляться ответные сообщения);
- дополнительный элемент `<NameIDFormat>` — URI-идентификатор формата псевдонима, поддерживаемый участником (пространство имен `urn:oasis:names:tc:SAML:1.1:nameid-format`), возможные значения: `unspecified`, `emailAddress`, `X509SubjectName`, `WindowsDomainQualifiedName`, `kerberos`, `entity`, `persistent`, `transient`;
- дополнительный элемент `<AttributeProfile>` указывает URI-идентификатор профиля атрибутов, поддерживаемого участником;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- дополнительный элемент `<saml:Attribute>` — идентификатор атрибута, поддерживаемый участником.

Элемент `<AffiliationDescriptor>` описывает объединение участников с помощью следующих атрибутов и элементов:

- обязательный атрибут `affiliationOwnerID` — идентификатор участника, отвечающего за объединение;
- дополнительный атрибут `ID` — идентификатор элемента;
- дополнительный атрибут `validUntil` — время окончания действия метаданных элемента;
- дополнительный атрибут `cacheDuration` — максимальное время кэширования метаданных элемента;
- дополнительный элемент `<ds:Signature>` — цифровая подпись метаданных элемента;
- дополнительный элемент `<Extensions>` — элементы расширения спецификации;
- обязательные элементы `<AffiliateMember>` — идентификаторы членов объединения;
- дополнительный элемент `<KeyDescriptor>` — информация о криптографических ключах объединения.

WSDL 1.1

В табл. 1 перечислены пространства имен, используемые в документе WSDL 1.1.

Таблица 1. *Пространства имен WSDL 1.1*

Префикс	Пространство имен
wsdl	http://schemas.xmlsoap.org/wsdl/
soap	http://schemas.xmlsoap.org/wsdl/soap/
http	http://schemas.xmlsoap.org/wsdl/http/
mime	http://schemas.xmlsoap.org/wsdl/mime/
soapenc	http://schemas.xmlsoap.org/soap/encoding/
soapenv	http://schemas.xmlsoap.org/soap/envelope/
xsi	http://www.w3.org/2000/10/XMLSchema-instance
xsd	http://www.w3.org/2000/10/XMLSchema

Корневым элементом документа WSDL 1.1 является элемент `<wsdl:definitions>` (рис. 1). Этот элемент имеет необязательные атрибуты `name` (имя документа) и `targetNamespace` (целевое пространство имен WSDL-описания).

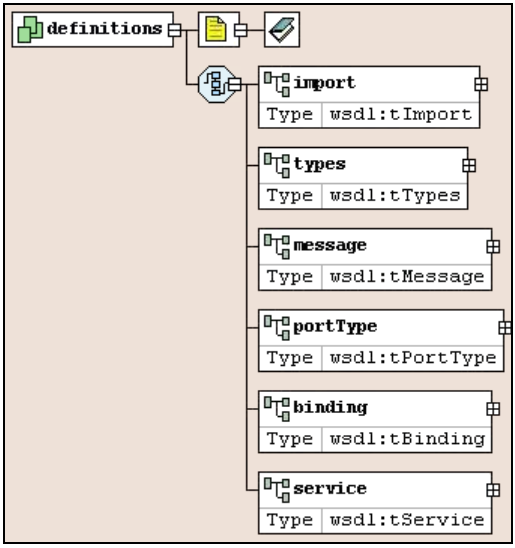


Рис. 1. Общая схема элемента `<wsdl:definitions>`

Элемент `<wsdl:import>` позволяет связать пространство имен с адресом документа с помощью атрибутов `namespace` (URI-идентификатор пространства имен, объявленный посредством атрибута `xmlns` элемента `<wsdl:definitions>`) и `location` (URI-адрес XML-схемы).

Элемент `<wsdl:document>` служит контейнером для документации (комментариев) элементов WSDL-описания Web-сервиса.

Элемент `<wsdl:types>` описывает типы данных, используемые в обмене сообщений Web-сервиса с помощью XML-схем, представленных вложенными элементами `<xsd:schema>`.

Элемент `<wsdl:message>` описывает сообщения, причем сообщения состоят из одной или нескольких частей, содержимое которых определяется XML-схемами, определенными в элементе `<wsdl:types>`. Элемент `<wsdl:message>` имеет атрибут `name`, указывающий идентификатор сообщения, а вложенные элементы `<part>` описывают неделимые части сообщения, связывая типы данных с идентификаторами частей сообщений, с помощью атрибутов:

- `name` — идентификатор части сообщения;
- `element` — ссылка на элемент `<element>` типа данных, задаваемая с помощью QName-имени элемента;
- `type` — ссылка на элемент `<simpleType>` или `<complexType>` типа данных, задаваемая с помощью QName-имени.

Элемент `<wsdl:portType>` описывает набор абстрактных операций Web-сервиса и связанные с ними сообщения и имеет атрибут `name` (идентификатор элемента) и вложенные элементы `<wsdl:operation>` (описывают операции).

Элемент `<wsdl:operation>` (рис. 2) имеет атрибут `name` (идентификатор операции) и вложенные элементы:

- `<wsdl:input>` — входящее сообщение, атрибуты `name` (идентификатор) и `message` (ссылка на элемент `<wsdl:message>` с помощью QName);
- `<wsdl:output>` — исходящее сообщение, атрибуты `name` (идентификатор) и `message` (ссылка на элемент `<wsdl:message>` с помощью QName);
- `<wsdl:fault>` — сообщение об ошибке, атрибуты `name` (идентификатор) и `message` (ссылка на элемент `<wsdl:message>` с помощью QName).

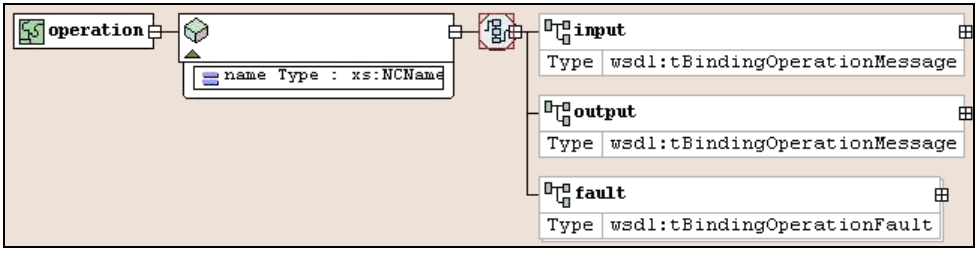


Рис. 2. Общая схема элемента `<wsdl:operation>`

Элемент <wsdl:binding> (рис. 3) описывает детали формата сообщений и протокола для операций порта, имеет атрибуты name (идентификатор связи) и type (имя элемента <wsdl:portType>). Детали формата сообщений и протокола их передачи описываются с помощью элементов WSDL-расширения, которые могут быть как дочерними элементами элемента <wsdl:binding>, так и дочерними элементами его вложенных элементов <wsdl:operation>, <wsdl:input>, <wsdl:output> и <wsdl:fault>.

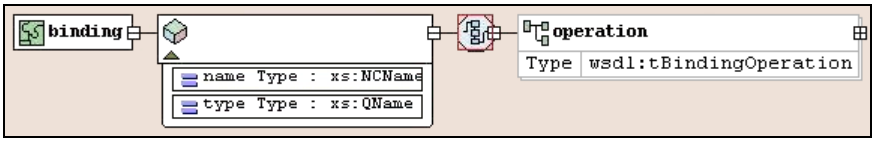


Рис. 3. Общая схема элемента <wsdl:binding>

Элементы WSDL-расширения

Протокол SOAP 1.1

- Элемент <soap:binding> имеет атрибуты:
  - transport, который указывает транспортный протокол для SOAP-сообщений (например, "http://schemas.xmlsoap.org/soap/http" — HTTP-протокол);
  - style, принимающий значение rpc (сообщение содержит параметры) и возвращает значения или document (по умолчанию, сообщение содержит документы).
- Элемент <soap:operation> содержит информацию для операции в целом, имеет атрибуты style (значение rpc или document) и soapAction (определяет заголовок операции).
- Элемент <soap:body> определяет композицию частей сообщения в элементе сообщения SOAP Body, имеет атрибуты:
  - parts — часть сообщения;
  - use, принимающий значение literal (кодировка части сообщения определяется конкретной XML-схемой, на которую ссылаются атрибуты element или type элемента <part>) или encoded (кодировка части сообщения определяется правилами кодирования, указанными с помощью атрибута encodingStyle и применяемыми к абстрактному типу, на который ссылается атрибут type элемента <part>);
  - encodingStyle — список URI-идентификаторов кодировок, используемых в сообщении;
  - namespace — пространство имен тела сообщения.
- Элемент <soap:fault> описывает содержимое элемента сообщения SOAP Fault Details и содержит атрибуты name, use, encodingStyle и namespace. Элемент <soap:fault> является дочерним элементом элемента <wsdl:fault>.

- Элементы `<soap:header>` и `<soap:headerfault>` содержат информацию для заголовка SOAP-сообщения. Элементы имеют атрибуты `message`, `part`, `use`, `namespace` и `encodingStyle`. Элемент `<soap:headerfault>` может присутствовать в элементе `<soap:header>` для описания передачи информации об ошибки в заголовке SOAP-сообщения.
- Элемент `<soap:address>` связывает порт с URI-адресом с помощью атрибута `location`.

Протокол HTTP 1.1

- Элемент `<http:address>` указывает базовый URI порта.
- Элемент `<http:binding>` указывает, что связка использует HTTP-протокол, атрибут `verb` указывает HTTP-метод (например, GET).
- Элемент `<http:operation>` указывает относительный URI-адрес для операции, который в комбинации с `<http:address>` формирует полный адрес HTTP-запроса.
- Элемент `<http:urlEncoded/>` указывает, что части сообщения кодируются в HTTP-запросе, используя стандартную URI-кодировку (`name1=value&name2=value...`).
- Элемент `<http:urlReplacement/>` указывает, что части сообщения кодируются в HTTP-запросе согласно алгоритму замещения.

Формат MIME

- Элемент `<mime:content>` связывает содержимое сообщения с MIME-типами с помощью атрибутов `part` и `type`.
- Элемент `<mime:multipartRelated>` составляет набор частей в одно сообщение MIME-типа `multipart/related` с помощью вложенных элементов `<mime:part>`.
- Элемент `<mime:mimeXml>` используется для описания передаваемых XML-данных, не имеющих элемента SOAP Envelope, но определяемых XML-схемой. Атрибут `part` элемента `<mime:mimeXml>` ссылается на часть сообщения, которая описывается конкретной XML-схемой.

Элемент `<wsdl:service>` описывает реализацию Web-сервиса путем группировки его конечных точек, являющихся получателями-отправителями сообщений Web-сервиса. Элемент `<wsdl:service>` имеет атрибут `name` (идентификатор сервиса) и вложенные элементы `<wsdl:port>`, определяющие конечные точки Web-сервиса.

Элемент `<wsdl:port>` содержит атрибуты `name` (имя конечной точки) и `binding` (QName-имя соответствующего элемента `<wsdl:binding>`), а также вложенные элементы WSDL-расширения `<soap:address>`, указывающие URI-адрес конечной точки.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Основные отличия WSDL 1.1 от WSDL 2.0:

- в WSDL 2.0 элемент `<definitions>` заменен элементом `<description>`;
- в WSDL 2.0 элемент `<portType>` заменен элементом `<interface>`;
- в WSDL 2.0 элемент `<port>` заменен элементом `<endpoint>`;
- в WSDL 2.0 элемент `<message>` отсутствует;
- элемент `<interface>` имеет дополнительные атрибуты `extends` и `styleDefault`;
- элемент `<operation>` элемента `<interface>` имеет обязательные атрибуты `name` и `pattern` и дополнительные атрибуты `style`, `wsdlx:safe` и `wrpc:signature`;
- в WSDL 2.0 элемент `<fault>` является дочерним элементом элемента `<interface>`;
- в WSDL 2.0 определены элементы `<infault>` и `<outfault>`, являющиеся дочерними для элементов `<operation>`;
- элементы `<input>` и `<output>`, определенные в элементе `<interface>`, в WSDL 2.0 имеют необязательные атрибуты `messageLabel` и `element`;
- элемент `<endpoint>` в WSDL 2.0 имеет дополнительный атрибут `address`;
- элементы расширения WSDL 2.0 поддерживают протокол SOAP 1.2;
- в WSDL 2.0 определены элементы `<include>` и `<import>`;
- элемент `<types>` WSDL 2.0 может иметь атрибуты `wsdlx:interface` и `wsdlx:binding`;
- атрибут `targetNamespace` обязателен для элемента `<description>`;
- в WSDL 2.0 вложенный элемент `<operation>` элемента `<binding>` имеет обязательный атрибут `ref`.

XML-схема конфигурации ActiveMQ

Элемент `<broker>` является основным элементом для определения конфигурации ActiveMQ-брокера (рис. 1 и 2).

Элемент `<adminView>` содержит информацию о брокере для администратора.

Элемент `<brokerContext>` содержит информацию о контексте брокера.

Элемент `<consumerSystemUsage>` содержит дочерний элемент `<systemUsage>`, конфигурирующий системные свойства клиента брокера (рис. 3).

Элемент `<destinationFactory>` указывает фабрику для создания объектов `Destination`.

Элемент `<destinationInterceptors>` определяет использование виртуальной очереди доставки сообщения нескольким физическим очередям.

Элемент `<destinationPolicy>` определяет политику объектов `Destination`.

Элемент `<destinations>` устанавливает объекты `Destination`, которые должны быть созданы при запуске брокера.

Элемент `<ioExceptionHandler>` определяет обработчик ошибок ввода/вывода при сохранении сообщений.

Элемент `<jmsBridgeConnectors>` обеспечивает связь с другими JMS-поставщиками.

Элемент `<managementContext>` конфигурирует поддержку JMX (рис. 4).

Элемент `<messageAuthorizationPolicy>` определяет объект `org.apache.activemq.security.MessageAuthorizationPolicy`, обеспечивающий авторизацию сообщений.

Элемент `<networkConnectorURIs>` определяет список адресов для связи распределенных брокеров между собой.

Элемент `<networkConnectors>` обеспечивает связь данного брокера с другими распределенными брокерами.

Элемент `<persistenceAdapter>` устанавливает набор адаптеров для связи с базами данных хранения сообщений (рис. 5—10).

Элемент `<persistenceFactory>` обеспечивает фабрики для адаптеров хранения.

Элемент `<persistenceTaskRunnerFactory>` обеспечивает управление пулом потоков сохранения.

Элемент `<plugins>` определяет набор используемых брокером плагинов.

Элемент `<producerSystemUsage>` содержит дочерний элемент `<systemUsage>`, конфигурирующий системные свойства брокера (см. рис. 3).

Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

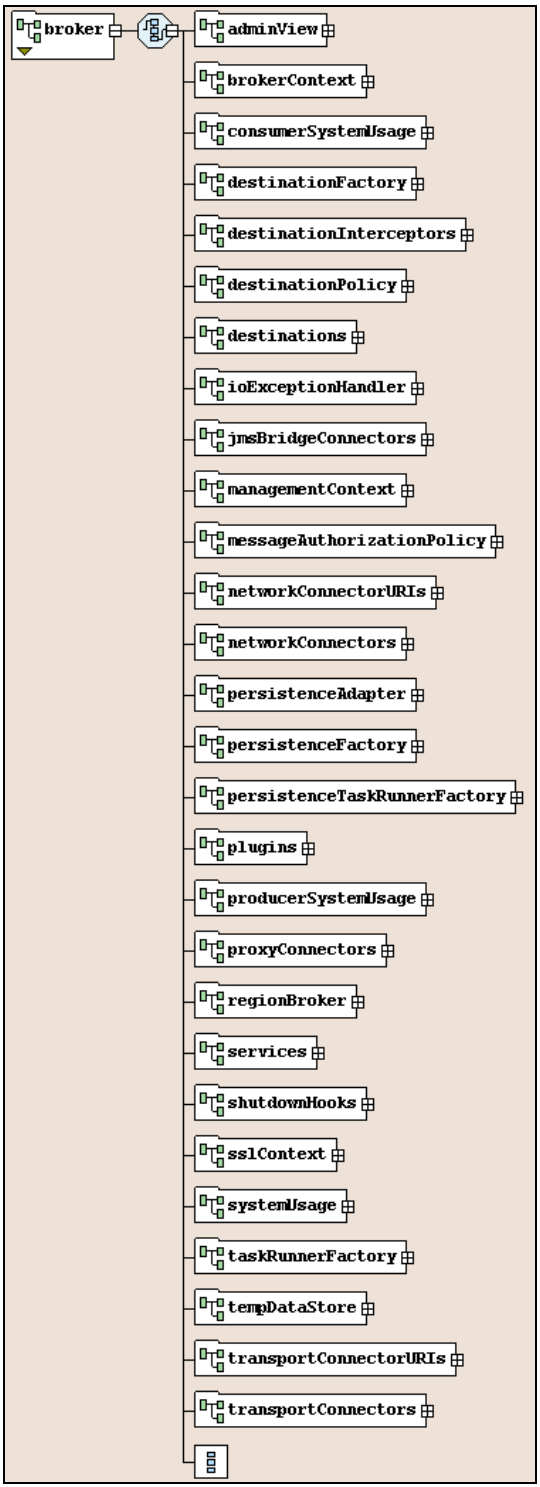


Рис. 1. Общая схема элемента `<broker>`

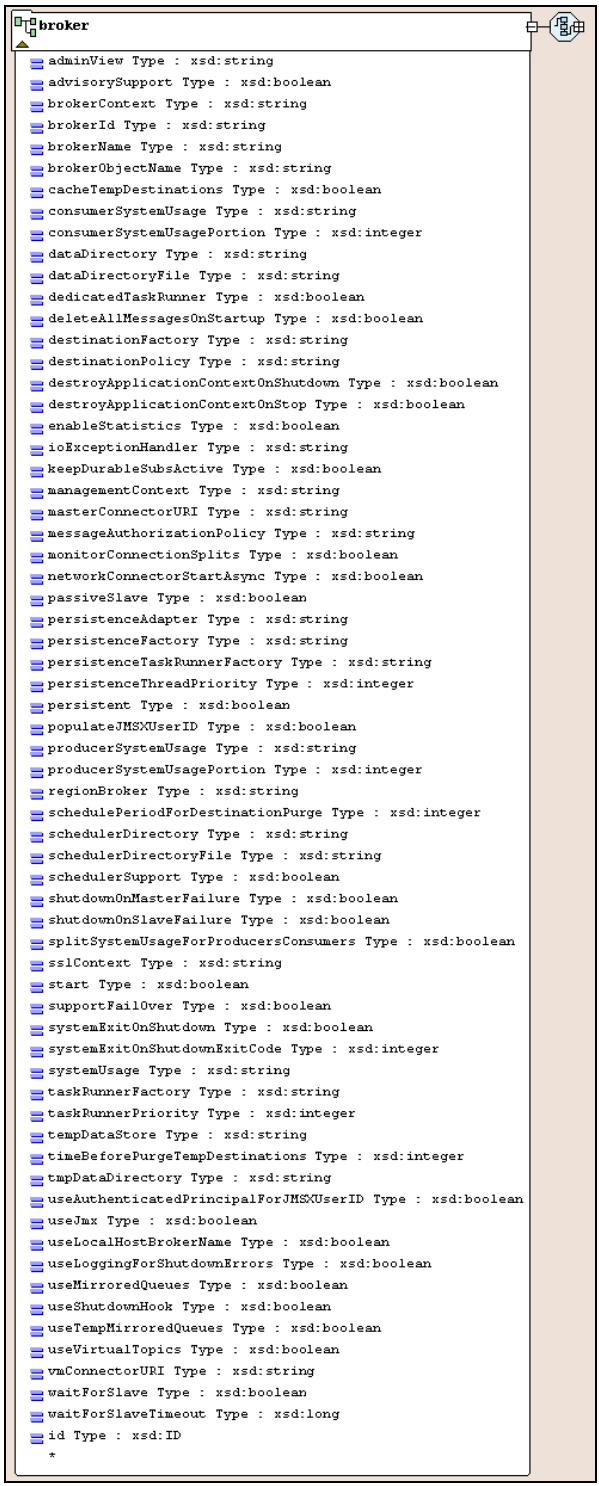


Рис. 2. Атрибуты элемента `<broker>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

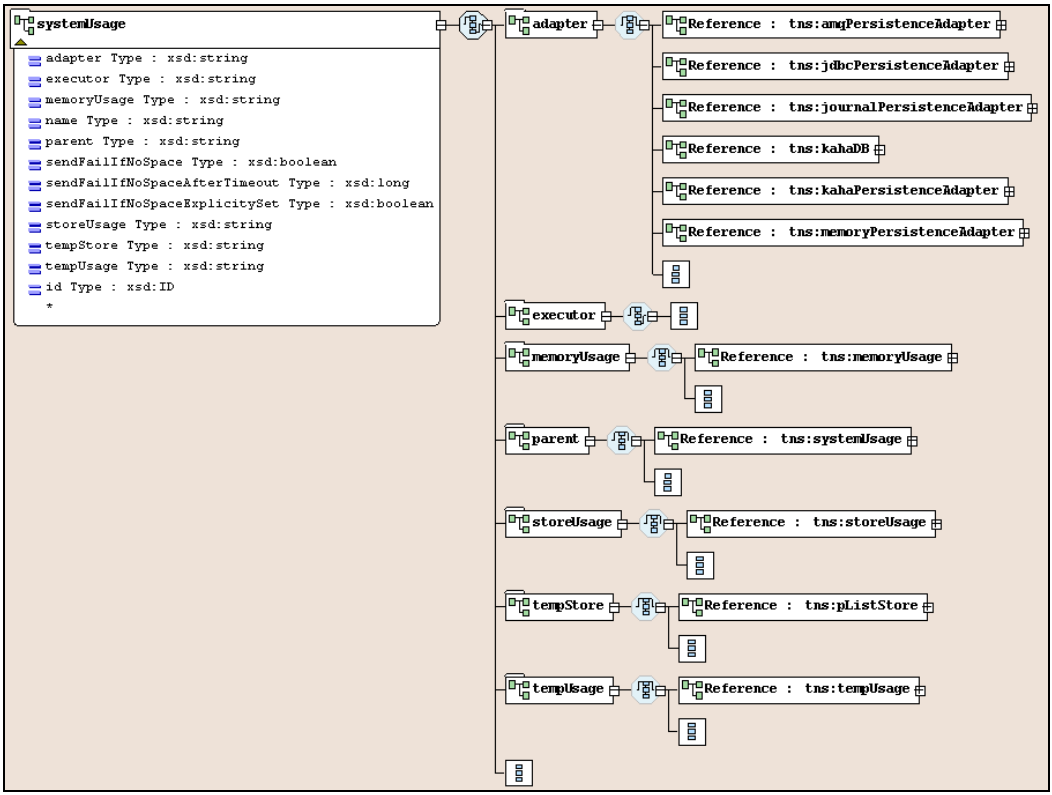


Рис. 3. Общая схема элемента `<systemUsage>`

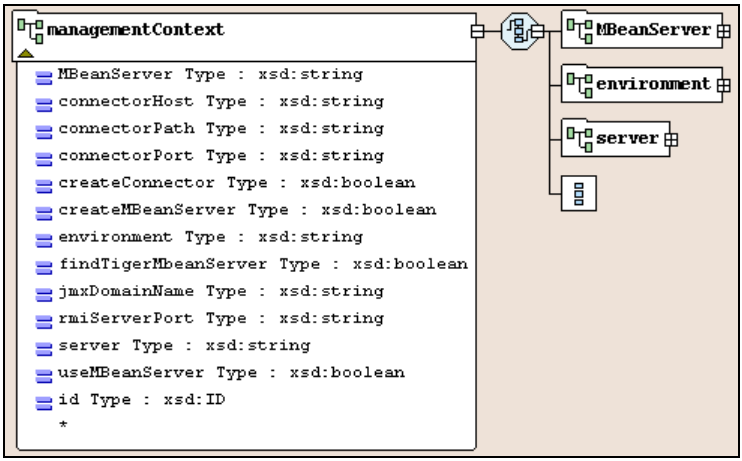


Рис. 4. Общая схема элемента `<managementContext>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<proxyConnectors>` определяет связь данного брокера с другими распределенными брокерами.

Элемент `<regionBroker>` представляет блок обработки сообщений (см. рис. 49).

Элемент `<services>` устанавливает сервисы для данного брокера (см. рис. 50).

Элемент `<shutdownHooks>` определяет выполнение кода по завершению работы брокера.

Элемент `<sslContext>` определяет конфигурацию использования протокола SSL (рис. 52).

Элемент `<systemUsage>` определяет системные свойства (см. рис. 3).

Элемент `<taskRunnerFactory>` определяет объект `org.apache.activemq.thread.TaskRunnerFactory`.

Элемент `<tempDataStore>` определяет конфигурацию объекта `org.apache.activemq.store.kahadb.plist.PListStore` с помощью дочернего элемента `<pListStore>`.

Элемент `<transportConnectorURIs>` указывает список адресов для соединения с брокером.

Элемент `<transportConnectors>` конфигурирует транспортный протокол для связи с брокером.

Дочерние элементы элемента *<systemUsage>*

Элемент `<adapter>` содержит информацию об адаптерах баз данных, используемых для хранения сообщений.

Элемент `<executor>` содержит информацию о пуле потоков.

Элемент `<memoryUsage>` конфигурирует использование памяти.

Элемент `<parent>` указывает родительский объект для данного объекта.

Элемент `<storeUsage>` определяет хранение сообщений.

Элемент `<tempStore>` определяет параметры хранения базы KahaDB.

Элемент `<tempUsage>` конфигурирует временное хранение сообщений.

Дочерние элементы элемента *<adapter>*

Элемент `<amqPersistenceAdapter>` конфигурирует адаптер для встроенной базы данных по умолчанию AMQ Message Store (рис. 5).

Элемент `<jdbcPersistenceAdapter>` конфигурирует JDBC-адаптер для хранения сообщений (рис. 6).

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

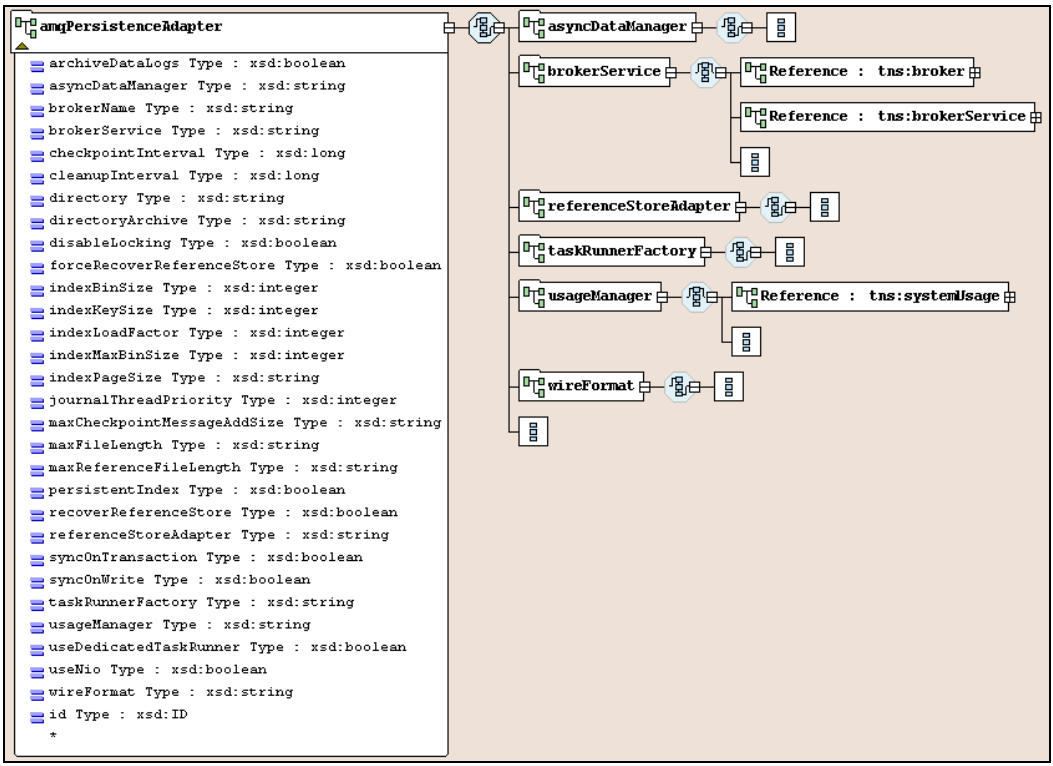


Рис. 5. Общая схема элемента `<amqPersistenceAdapter>`

Элемент `<journalPersistenceAdapter>` конфигурирует адаптер хранения сообщений в log-файле (рис. 7).

Элемент `<kahaDB>` конфигурирует локальную файловую базу данных хранения сообщений (рис. 8).

Элемент `<kahaPersistenceAdapter>` конфигурирует адаптер для базы данных KahaDB (рис. 9).

Элемент `<memoryPersistenceAdapter>` конфигурирует адаптер для хранения сообщений в памяти (рис. 10).

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

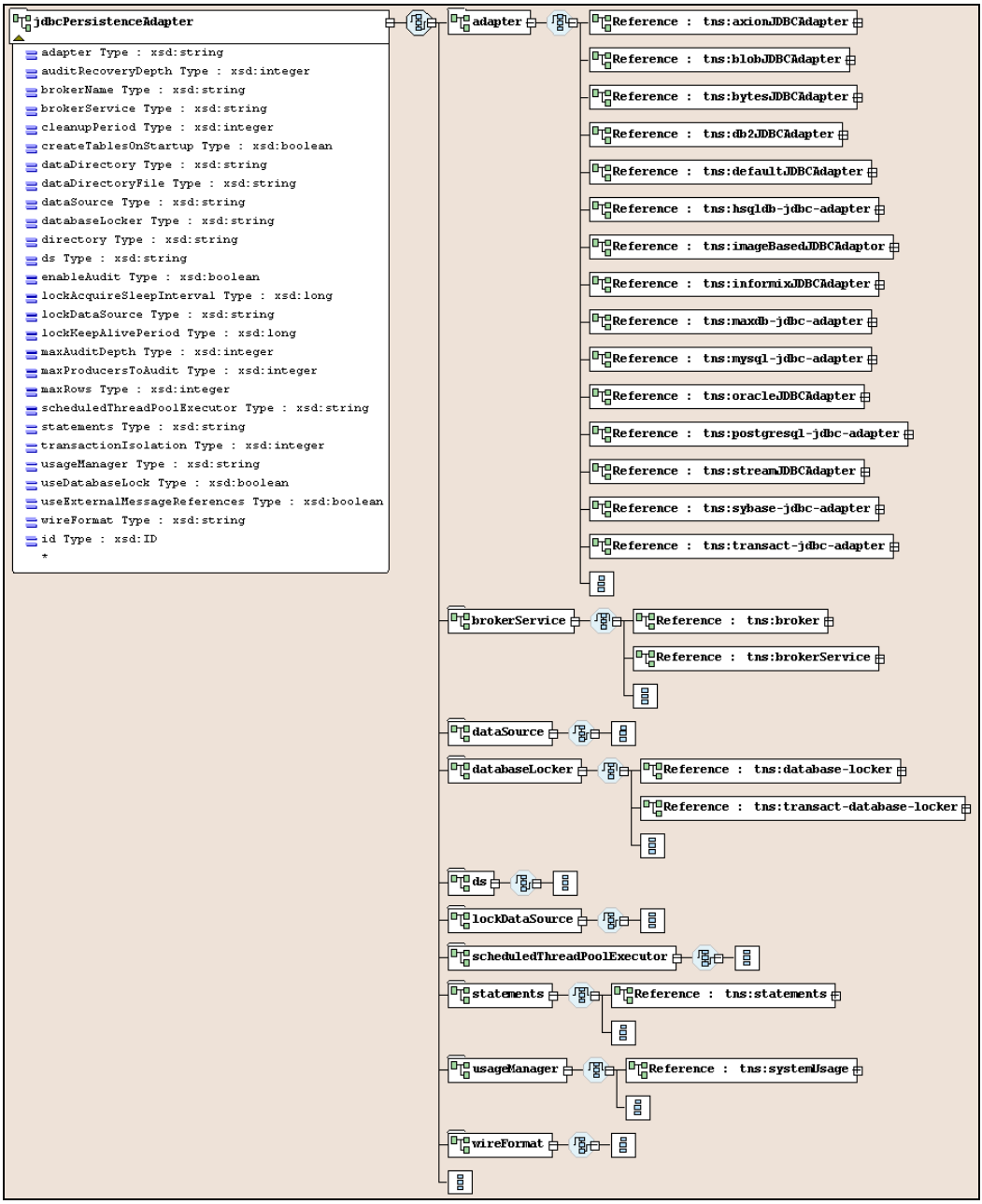


Рис. 6. Общая схема элемента `<jdbcPersistenceAdapter>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

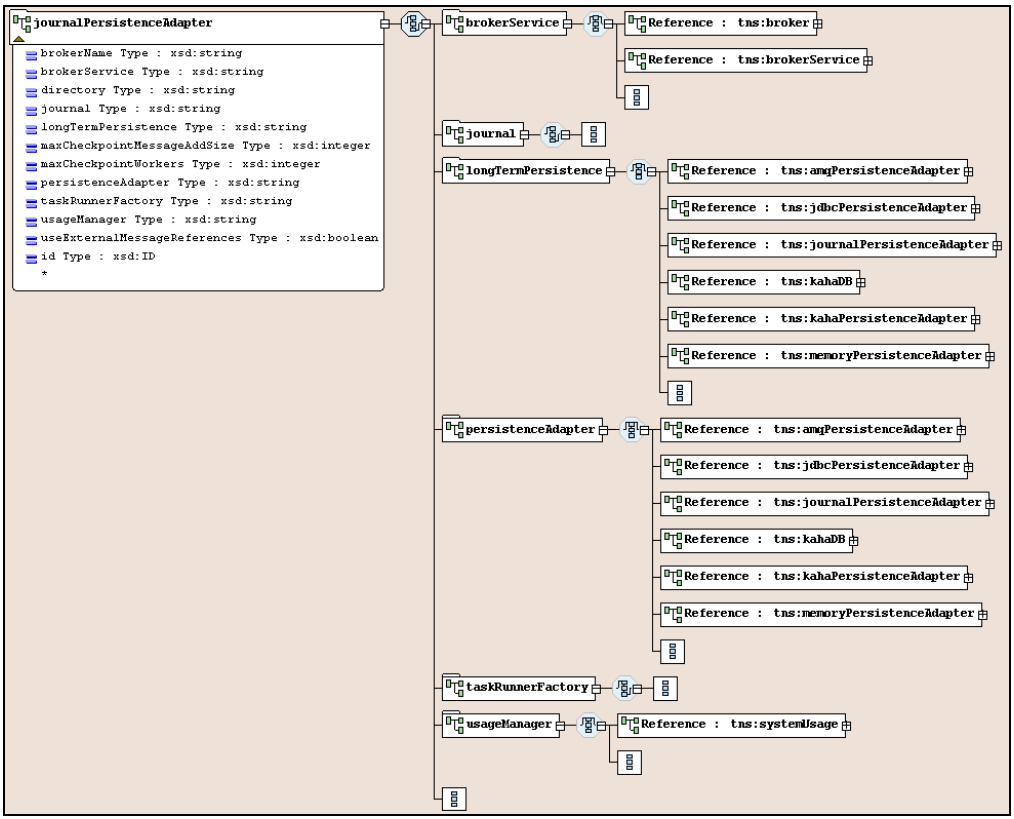


Рис. 7. Общая схема элемента `<journalPersistenceAdapter>`

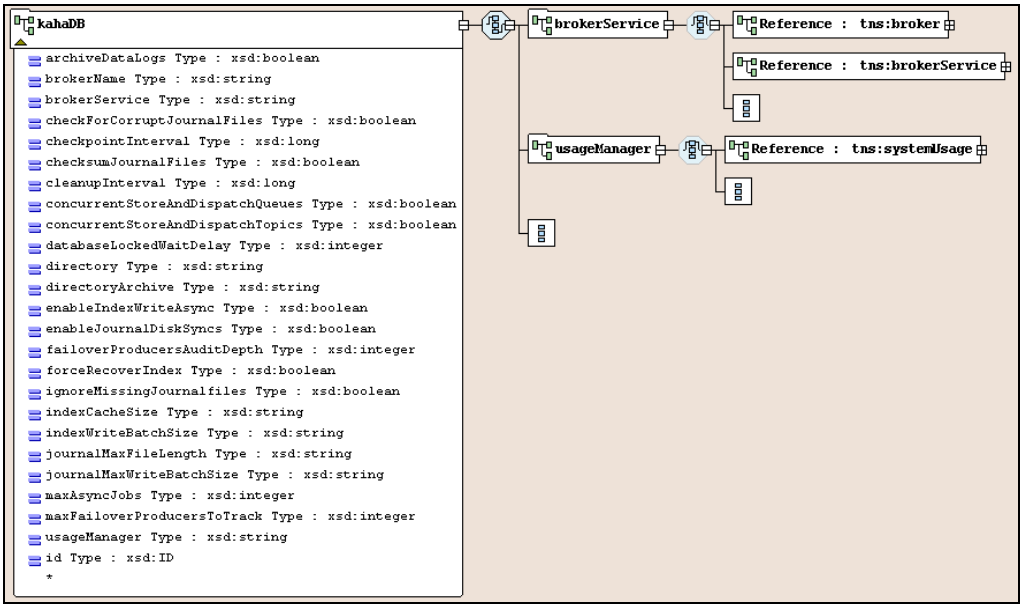


Рис. 8. Общая схема элемента `<kahaDB>`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

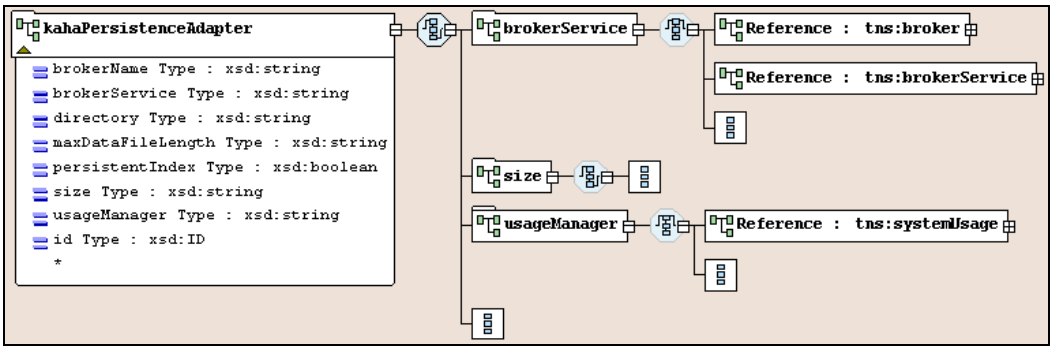


Рис. 9. Общая схема элемента `<kahaPersistenceAdapter>`

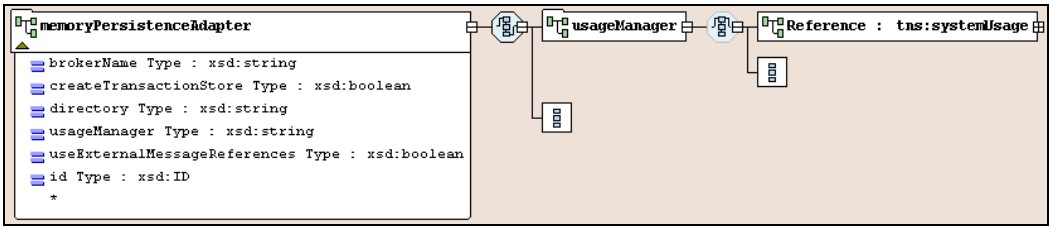


Рис. 10. Общая схема элемента `<memoryPersistenceAdapter>`

Дочерние элементы элемента *<memoryUsage>*

Элемент `<memoryUsage>` содержит следующие вложенные элементы (рис. 11):

- элемент `<executor>` содержит информацию о пуле потоков;
- элемент `<limiter>` ограничивает использование памяти;
- элемент `<parent>` указывает родительский объект для данного объекта.

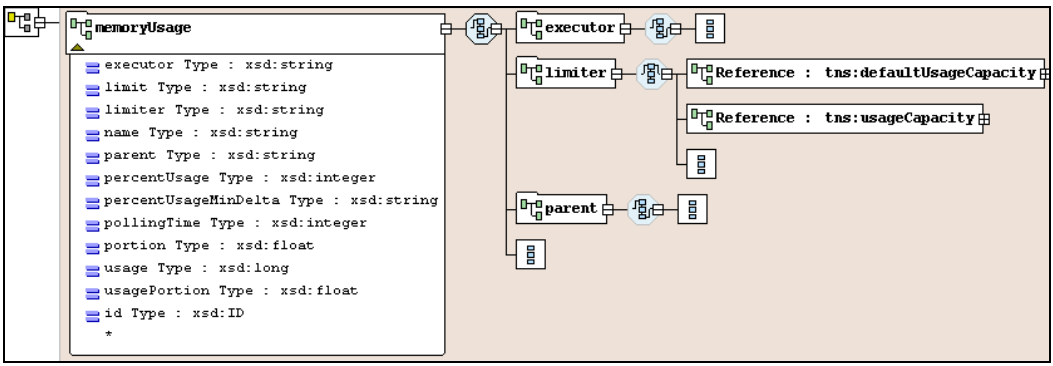


Рис. 11. Общая схема элемента `<memoryUsage>`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*storeUsage*>

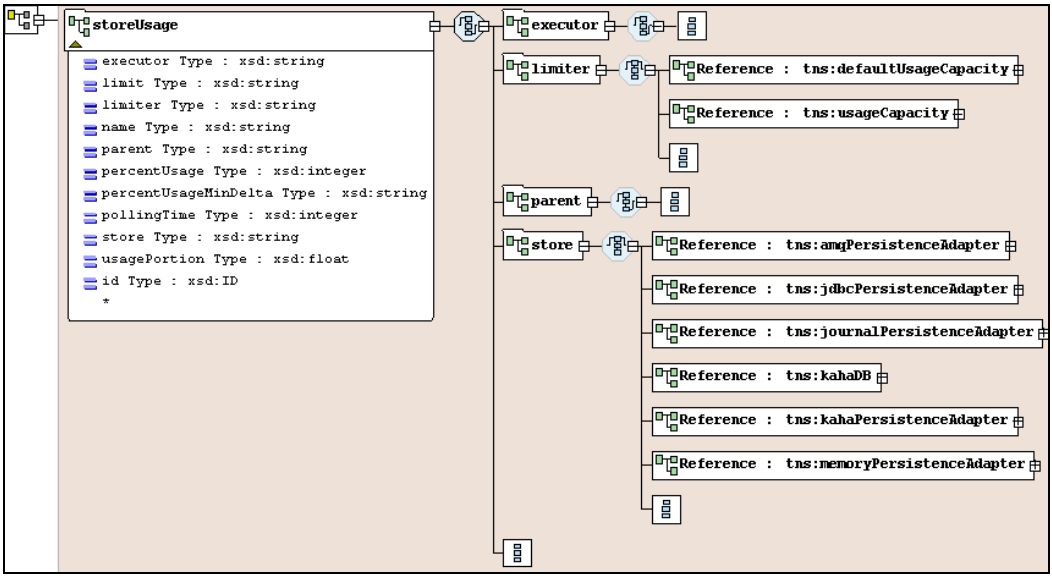


Рис. 12. Общая схема элемента `<storeUsage>`

Дочерние элементы элемента <*tempUsage*>

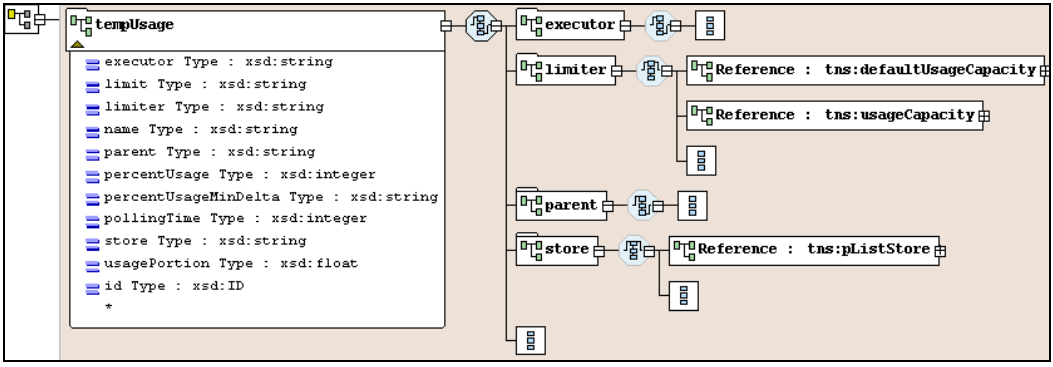


Рис. 13. Общая схема элемента `<tempUsage>`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*destinationInterceptors*>

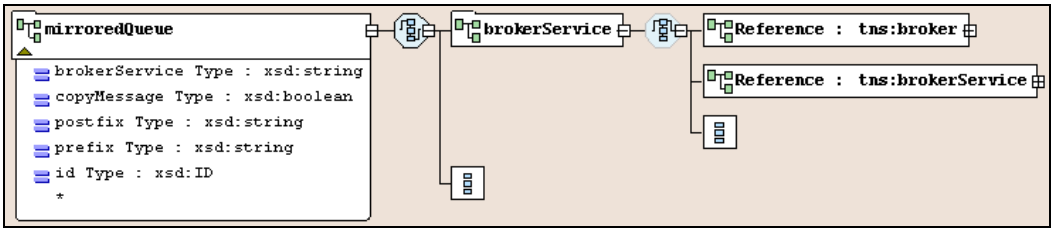


Рис. 14. Общая схема элемента `<mirroredQueue>`

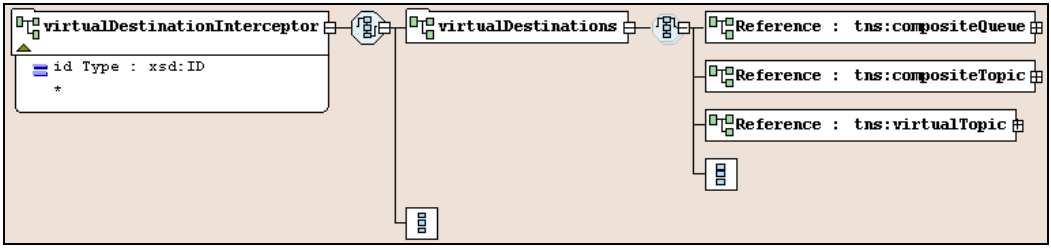


Рис. 15. Общая схема элемента `<virtualDestinationInterceptor>`

Дочерние элементы элемента <*destinationPolicy*>

Элемент `<policyMap>` определяет конфигурацию политик объектов `Destination` (рис. 16).

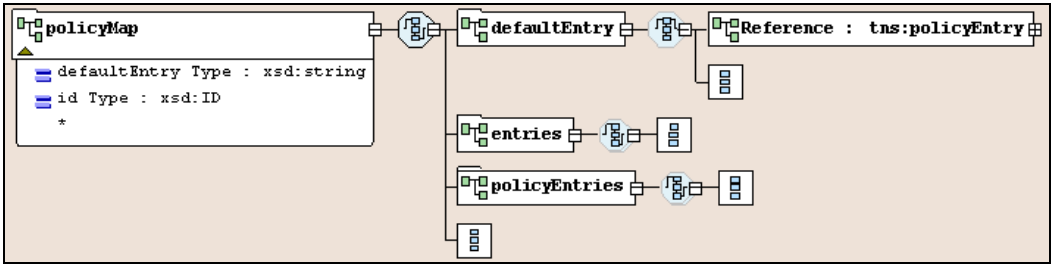


Рис. 16. Общая схема элемента `<policyMap>`

Элемент `<policyEntry>` представляет политику для определенных объектов `Destination` (рис. 17 и 18).

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

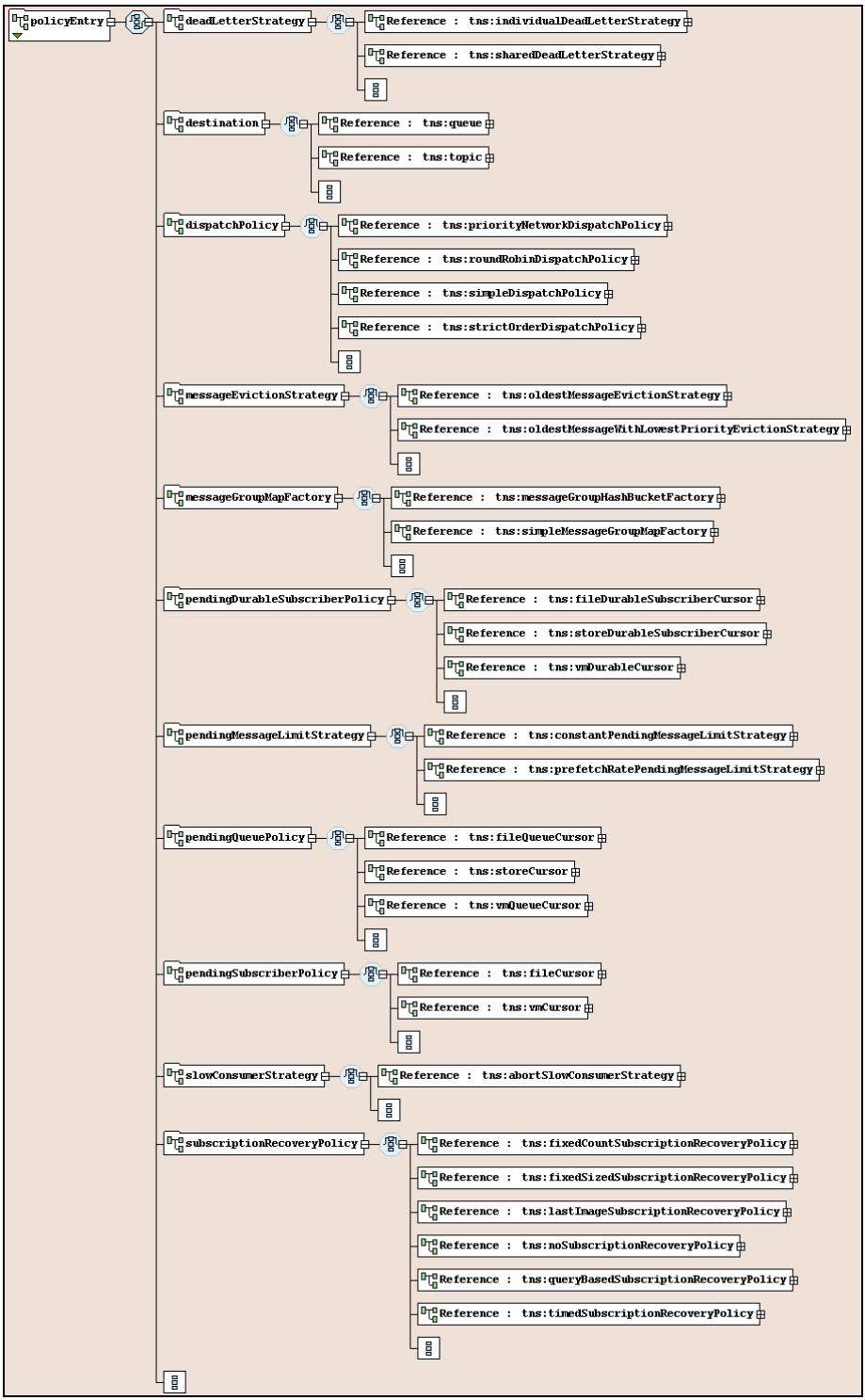


Рис. 17. Общая схема элемента `<policyEntry>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

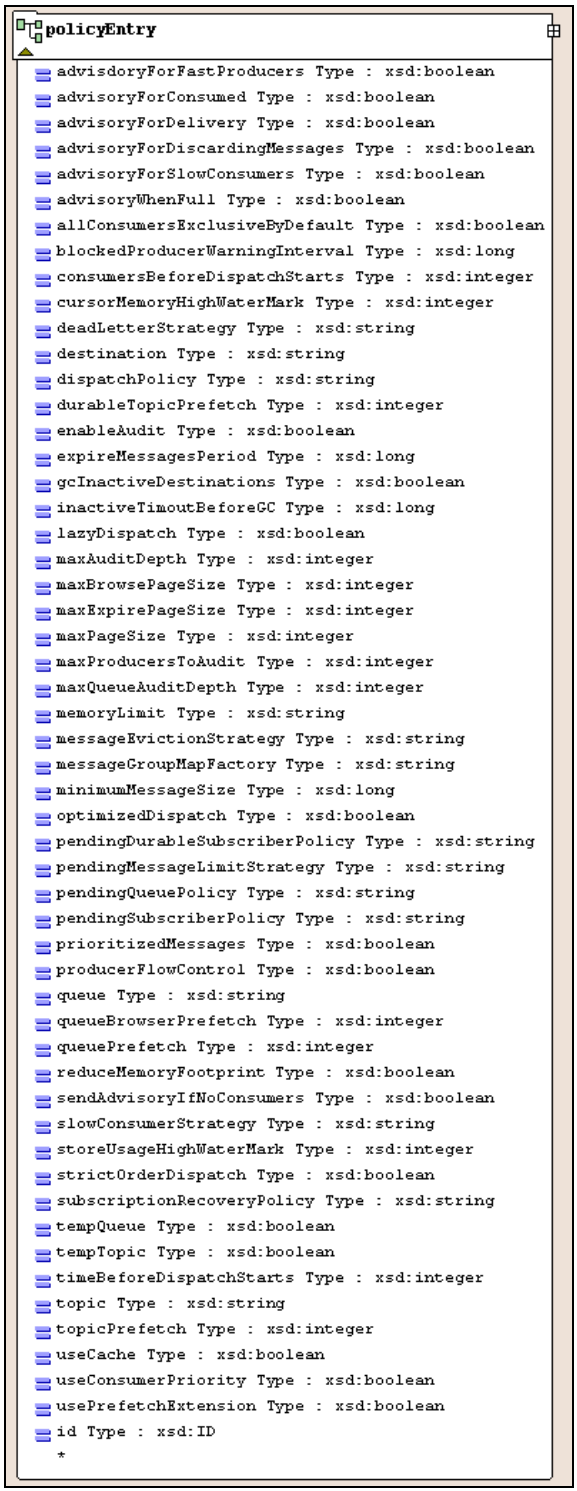


Рис. 18. Атрибуты элемента `<policyEntry>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента *<destinations>*

Элемент `<queue>` определяет конфигурацию объектов ActiveMQ Queue (рис. 19).

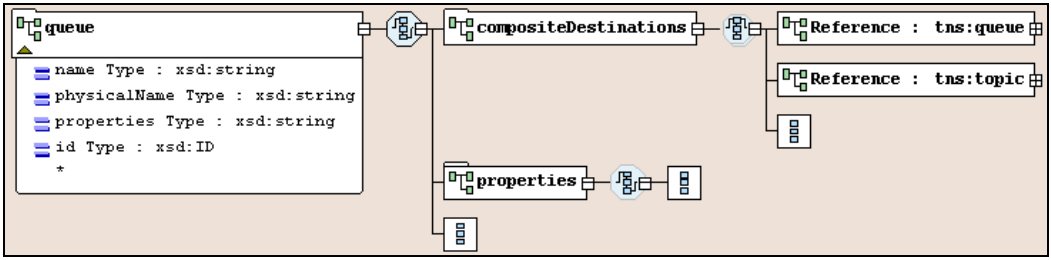


Рис. 19. Общая схема элемента `<queue>`

Элемент `<compositeDestinations>` определяет коллекцию объектов `Destination`, которым доставляется сообщение.

Элемент `<topic>` определяет конфигурацию объектов ActiveMQ Topic (рис. 20).

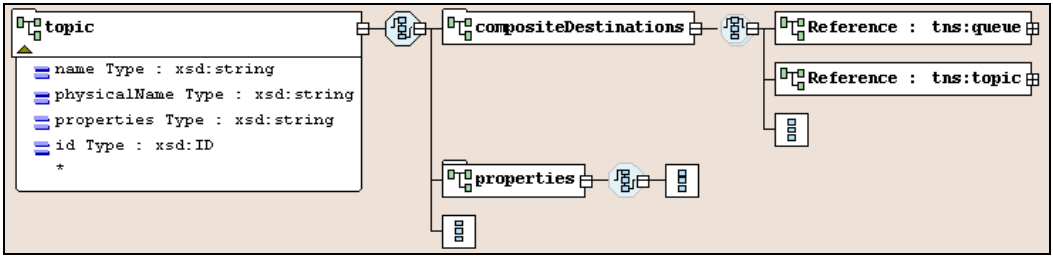


Рис 20. Общая схема элемента `<topic>`

Дочерние элементы элемента *<IOExceptionHandler>*

Элемент `<defaultIOExceptionHandler>` переопределяет объект `org.apache.activemq.util.DefaultIOExceptionHandler` (рис. 21).

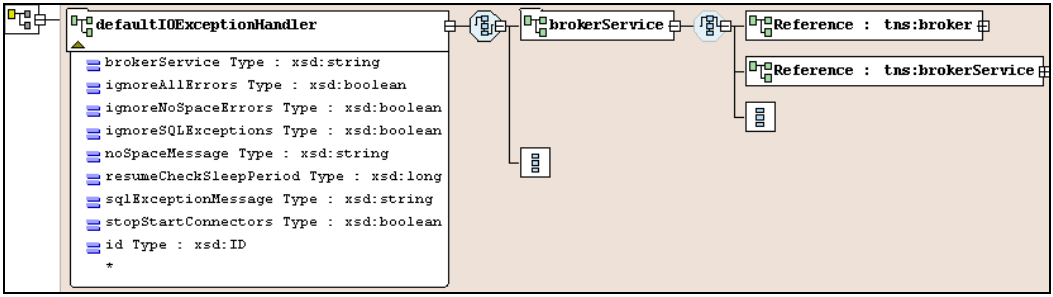


Рис. 21. Общая схема элемента `<defaultIOExceptionHandler>`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*jmsBridgeConnectors*>

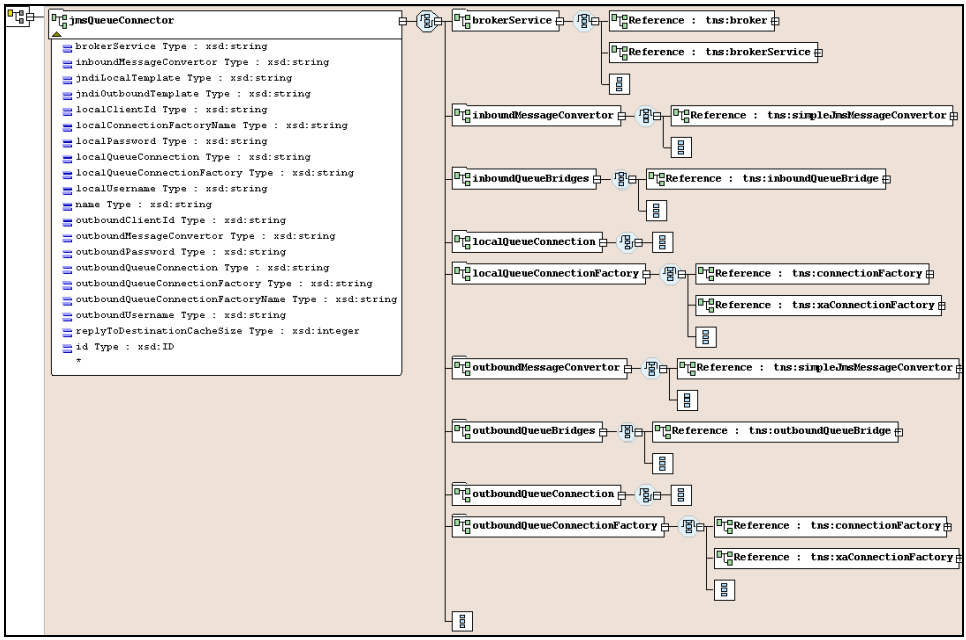


Рис. 22. Общая схема элемента `<jmsQueueConnector>`

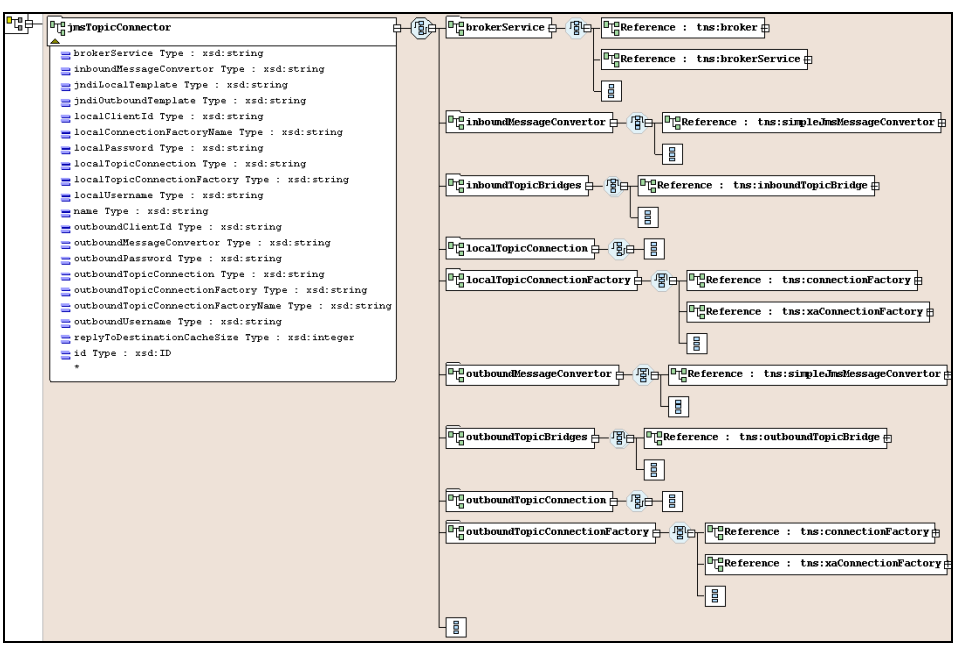


Рис. 23. Общая схема элемента `<jmsTopicConnector>`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*networkConnectors*>

Элемент `<ldapNetworkConnector>` обеспечивает динамическую связь с распределенными брокерами (рис. 24).

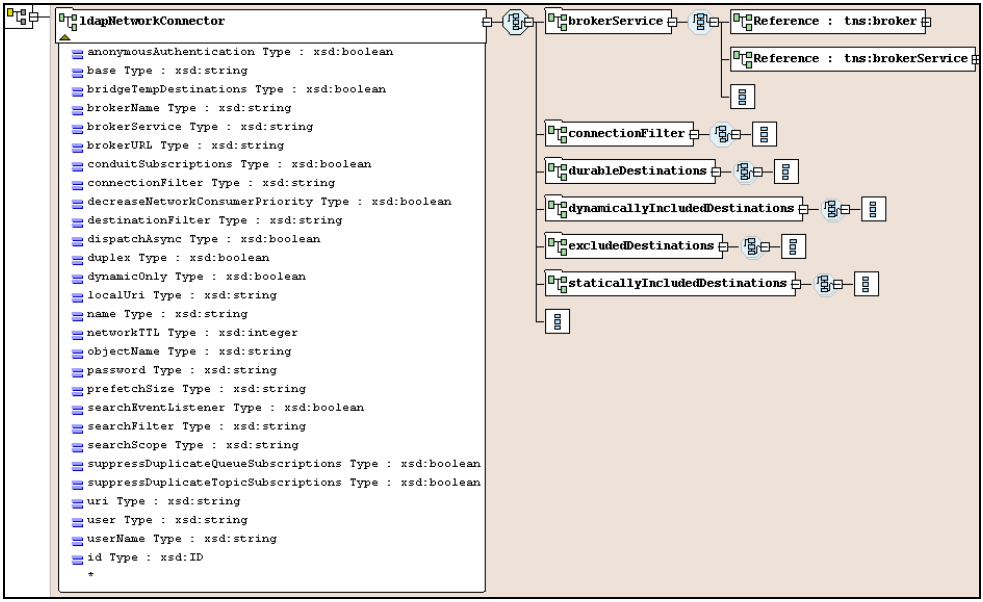


Рис. 24. Общая схема элемента `<ldapNetworkConnector>`

Элемент `<multicastNetworkConnector>` обеспечивает многоадресную связь с распределенными брокерами (рис. 25).

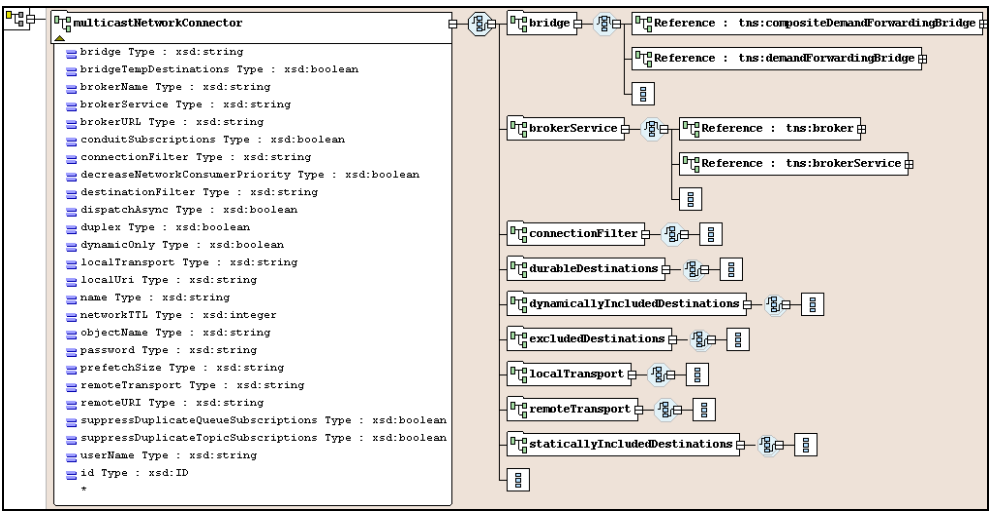


Рис. 25. Общая схема элемента `<multicastNetworkConnector>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<networkConnector>` обеспечивает связь с распределенными брокерами с применением объекта `org.apache.activemq.transport.discovery.DiscoveryAgent` (рис. 26).

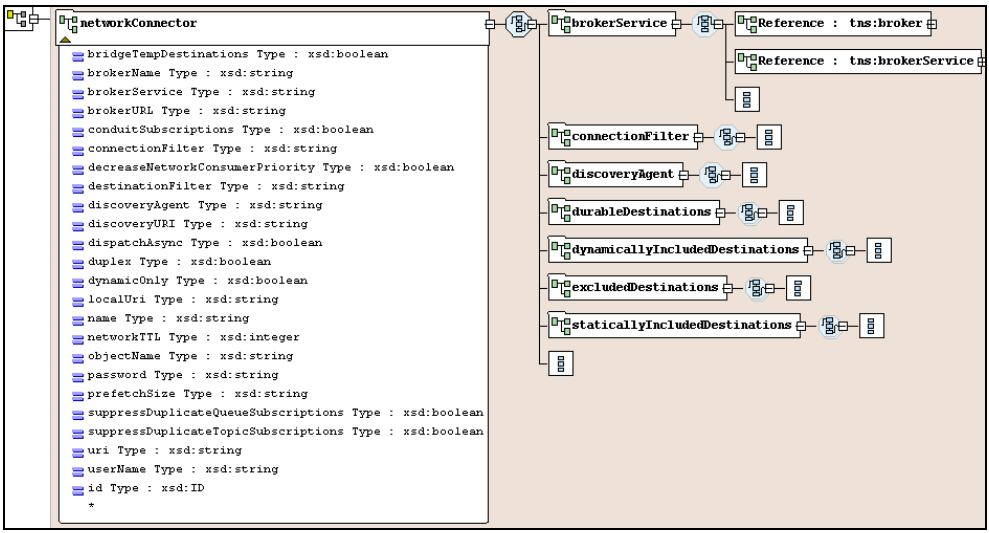


Рис. 26. Общая схема элемента `<networkConnector>`

Дочерние элементы элемента *<persistenceFactory>*

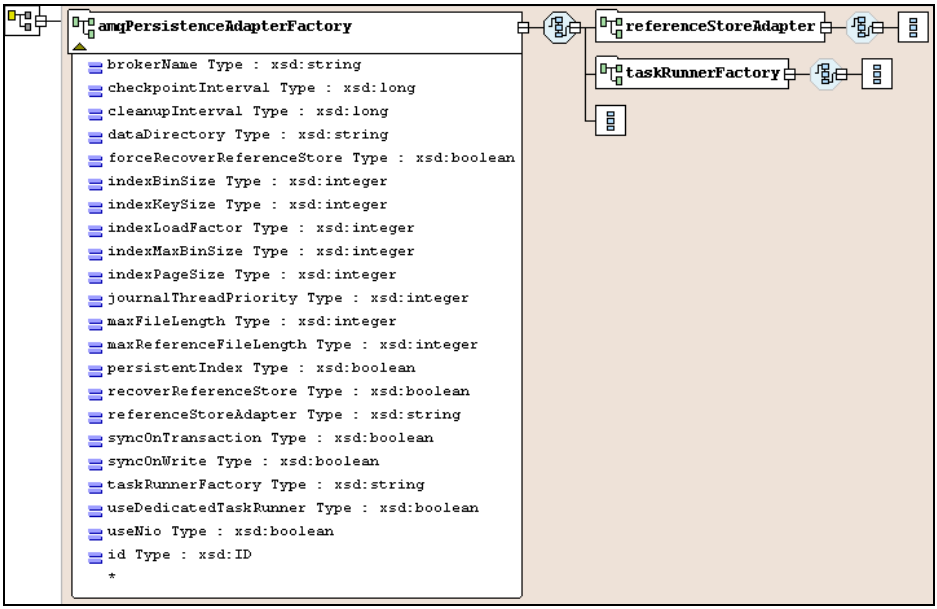


Рис. 27. Общая схема элемента `<amqPersistenceAdapterFactory>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

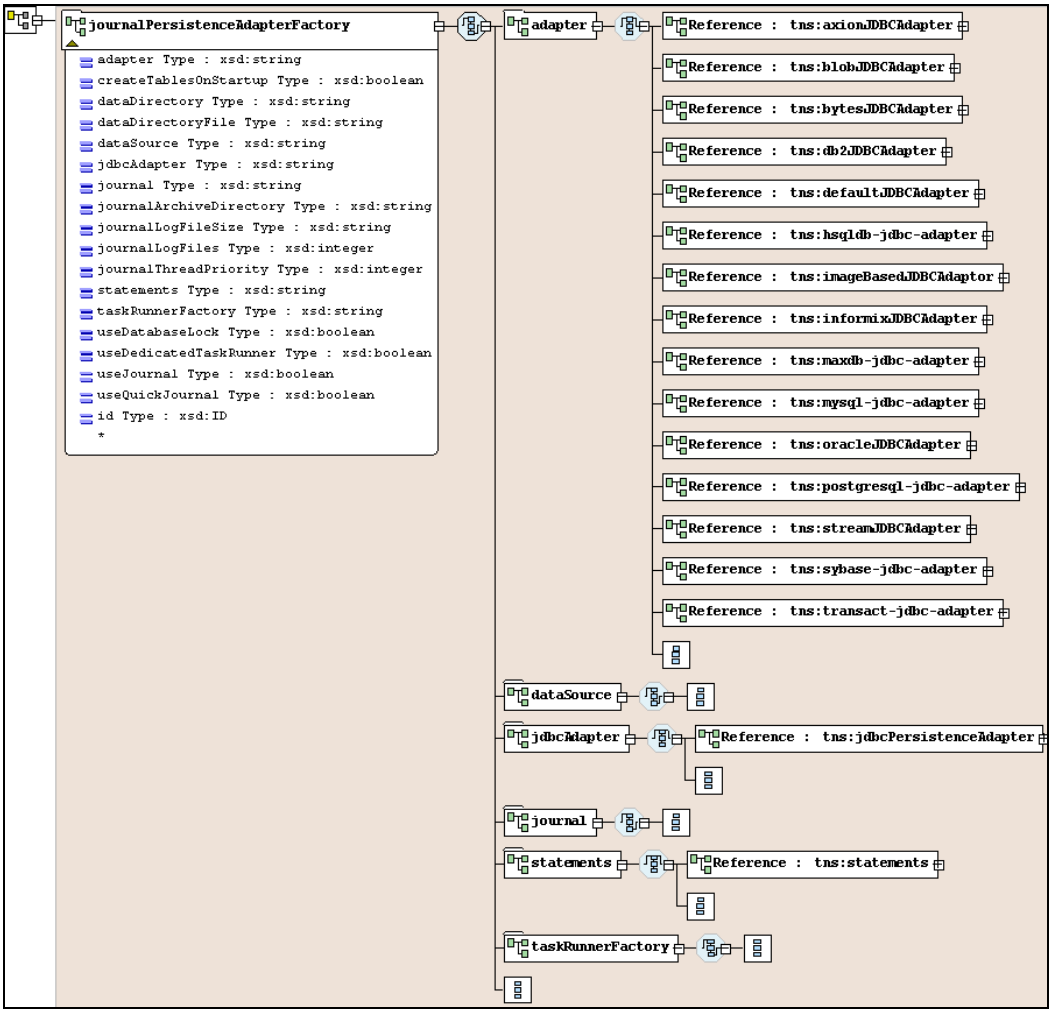


Рис. 28. Общая схема элемента `<journalPersistenceAdapterFactory>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

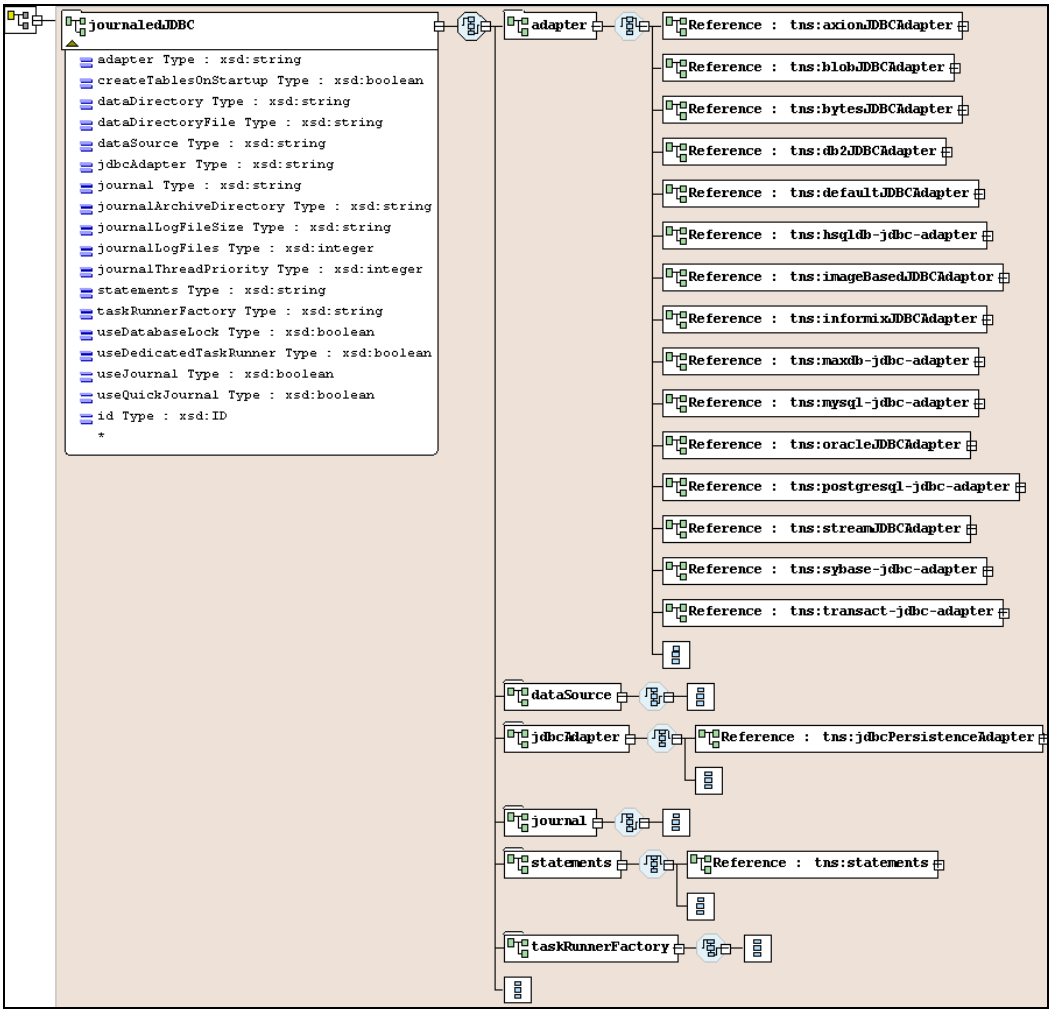


Рис. 29. Общая схема элемента `<journaledJDBC>`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*plugins*>

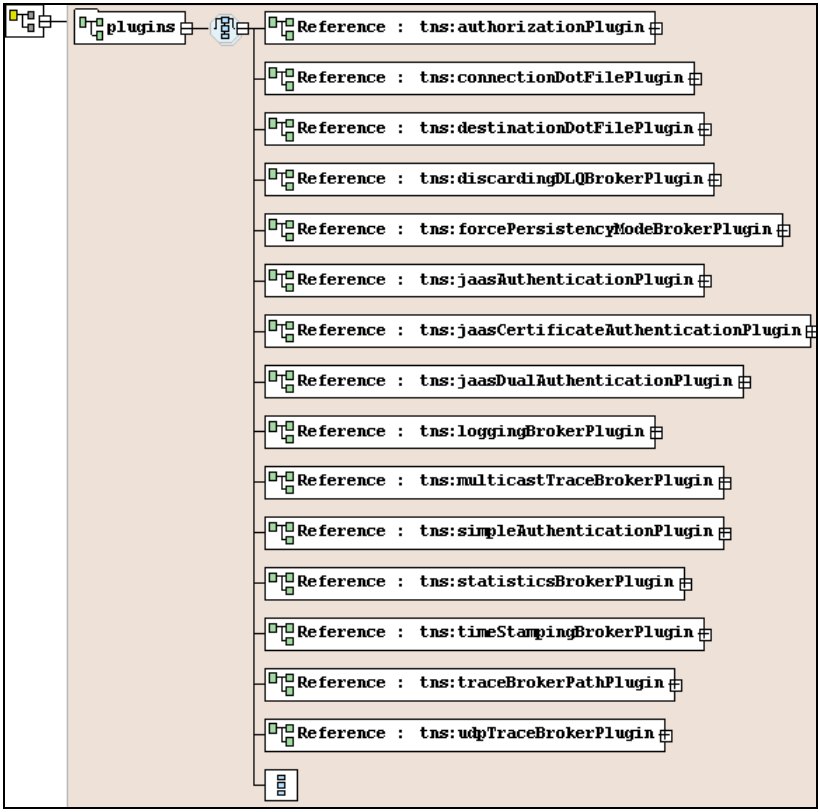


Рис. 30. Общая схема элемента `<plugins>`

Элемент `<authorizationPlugin>` определяет конфигурацию механизма авторизации (рис. 31).

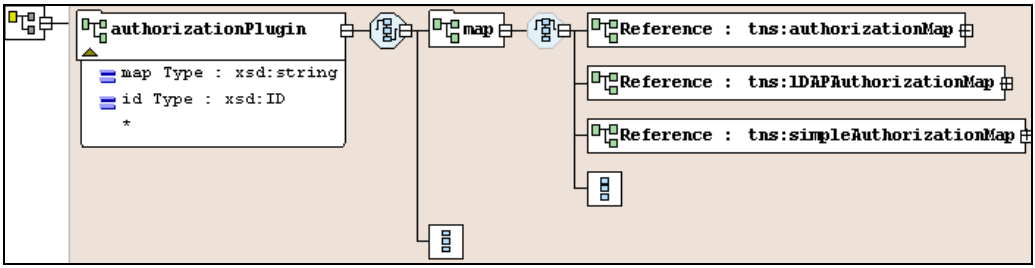


Рис. 31. Общая схема `<authorizationPlugin>`

Элемент `<connectionDotFilePlugin>` представляет модуль, создающий DOT-файл для текущих соединений (рис. 32).

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<destinationDotFilePlugin>` представляет модуль, создающий DOT-файл для текущих объектов `Destination` (рис. 33).

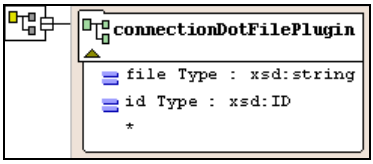


Рис. 32. Общая схема элемента `<connectionDotFilePlugin>`

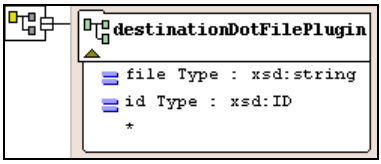


Рис. 33. Общая схема элемента `<destinationDotFilePlugin>`

Элемент `<discardingDLQBrokerPlugin>` представляет модуль отбрасывания сообщений, предназначенных для Dead Letter Queue (рис. 34).

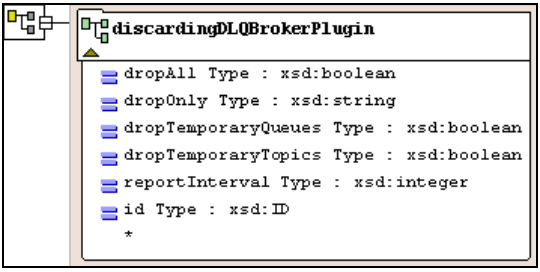


Рис. 34. Общая схема элемента `<discardingDLQBrokerPlugin>`

Элемент `<forcePersistencyModeBrokerPlugin>` определяет сохранение или нет всех входящих сообщений (рис. 35).

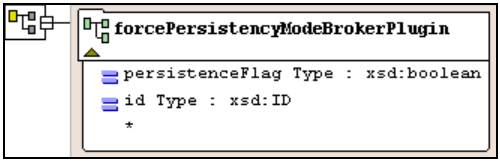


Рис. 35. Общая схема элемента `<forcePersistencyModeBrokerPlugin>`

Элемент `<jaasAuthenticationPlugin>` представляет модуль аутентификации JAAS (рис. 36).

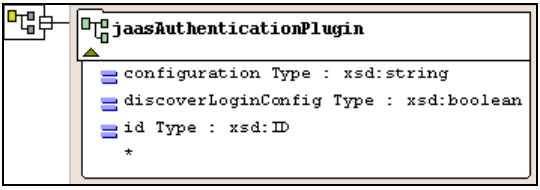


Рис. 36. Общая схема элемента `<jaasAuthenticationPlugin>`

Элемент `<jaasCertificateAuthenticationPlugin>` представляет модуль аутентификации JAAS/SSL (рис. 37).

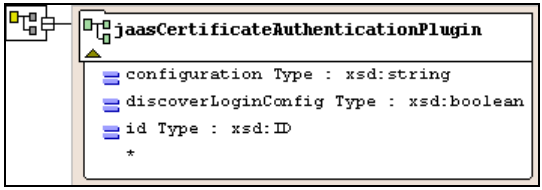


Рис. 37. Общая схема элемента `<jaasCertificateAuthenticationPlugin>`

Элемент `<jaasDualAuthenticationPlugin>` представляет двойной модуль аутентификации JAAS и JAAS/SSL (рис. 38).

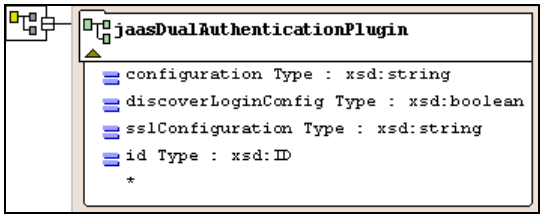


Рис. 38. Общая схема элемента `<jaasDualAuthenticationPlugin>`

Элемент `<loggingBrokerPlugin>` представляет модуль, отключающий/включающий регистрацию сообщений (рис. 39).

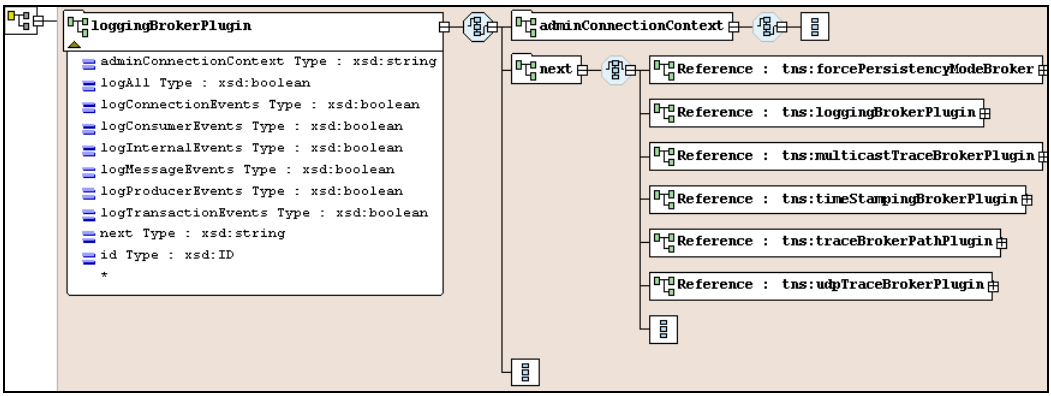


Рис. 39. Общая схема элемента `<loggingBrokerPlugin>`

Элемент `<multicastTraceBrokerPlugin>` представляет модуль регистрации для Multicast-сокета (рис. 40).

Элемент `<simpleAuthenticationPlugin>` представляет модуль упрощенной аутентификации (рис. 41).

Элемент `<statisticsBrokerPlugin>` представляет модуль получения статистики MapMessage от брокера (рис. 42).

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

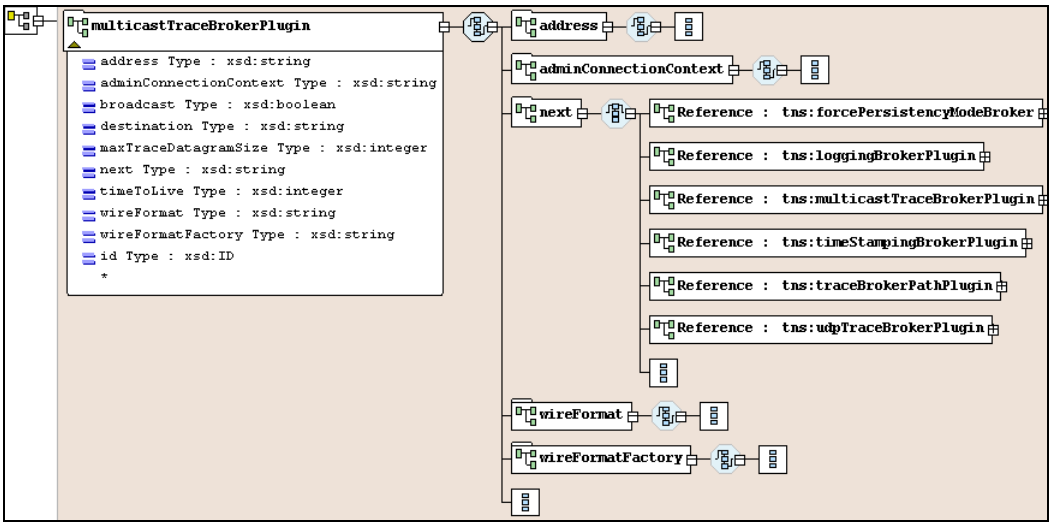


Рис. 40. Общая схема элемента `<multicastTraceBrokerPlugin>`

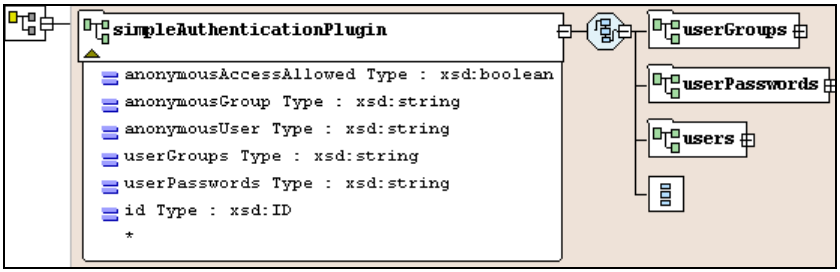


Рис. 41. Общая схема элемента `<simpleAuthenticationPlugin>`

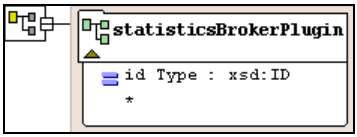


Рис. 42. Общая схема элемента `<statisticsBrokerPlugin>`

Элемент `<timeStampingBrokerPlugin>` представляет модуль установки времени действия сообщений (рис. 43).

Элемент `<traceBrokerPathPlugin>` представляет модуль включения имени брокера в свойство сообщения для регистрации в системе распределенных брокеров (рис. 44).

Элемент `<udpTraceBrokerPlugin>` представляет модуль регистрации для UDP-сокета (рис. 45).

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

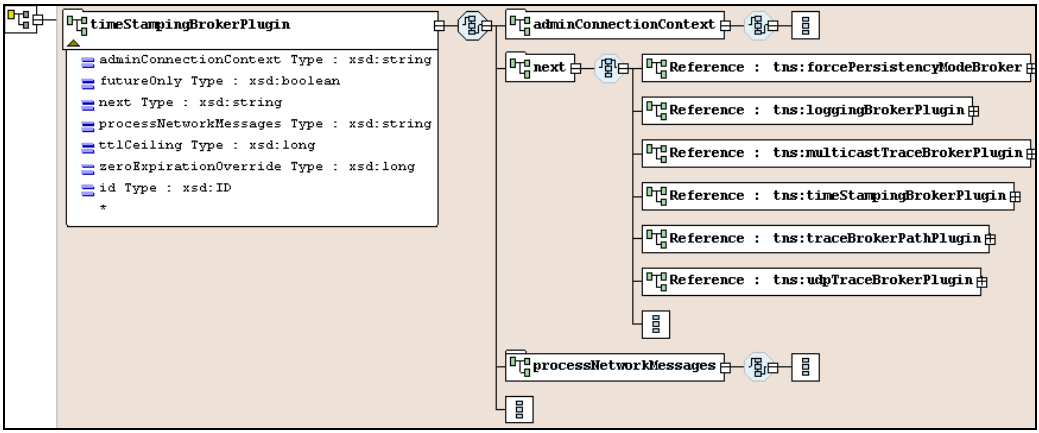


Рис. 43. Общая схема элемента `<timeStampingBrokerPlugin>`

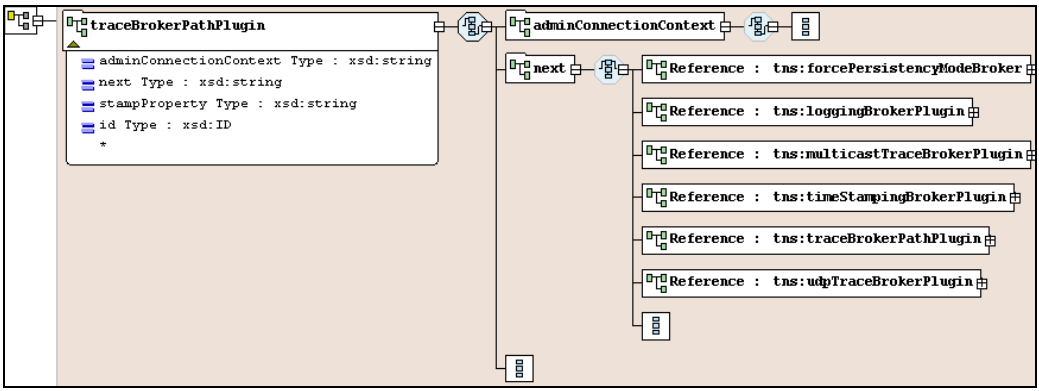


Рис. 44. Общая схема элемента `<traceBrokerPathPlugin>`

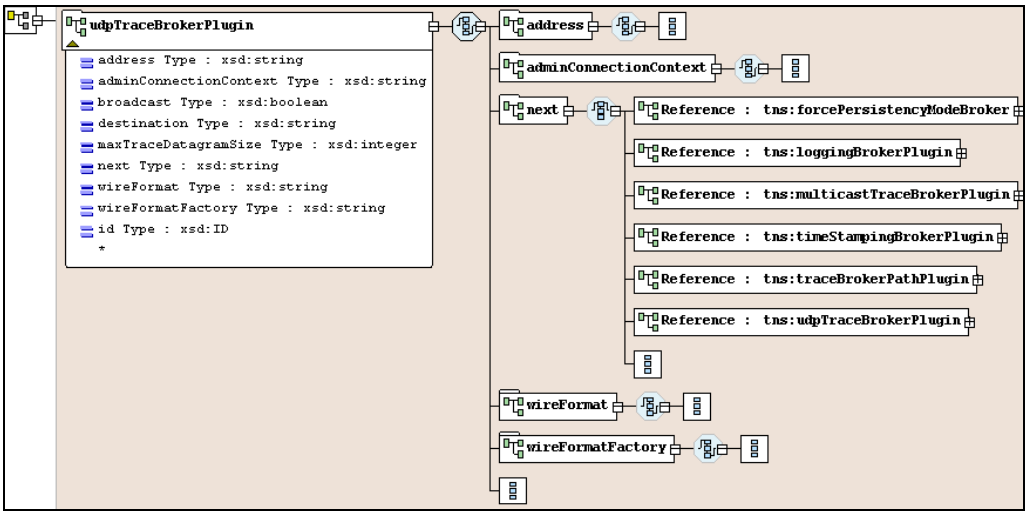


Рис. 45. Общая схема элемента `<udpTraceBrokerPlugin>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*authorizationPlugin*>

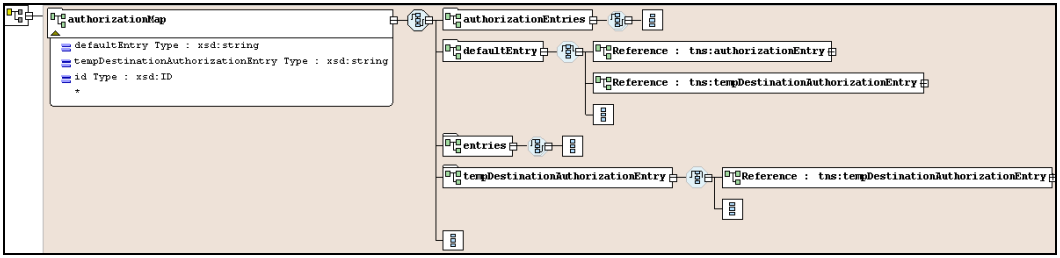


Рис. 46. Общая схема элемента `<authorizationMap>`

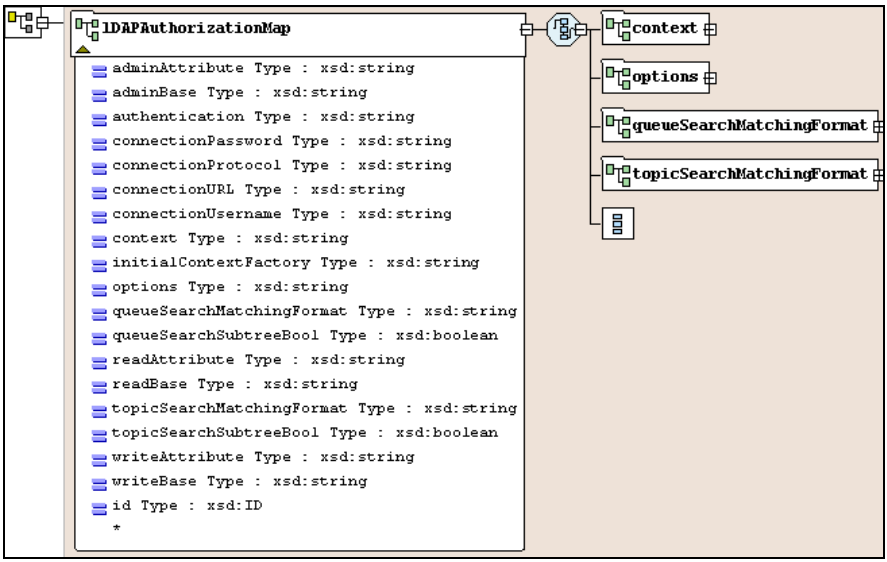


Рис. 47. Общая схема элемента `<lDAPAuthorizationMap>`

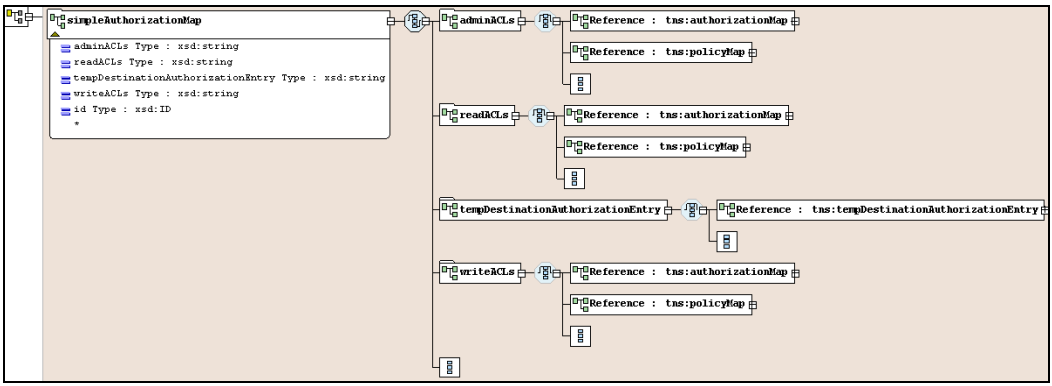


Рис. 48. Общая схема элемента `<simpleAuthorizationMap>`

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Дочерние элементы элемента <*regionBroker*>

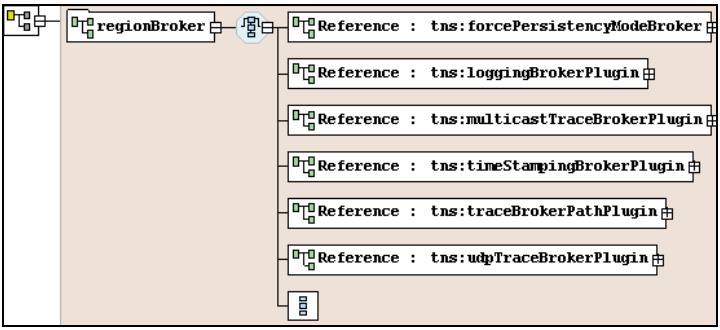


Рис. 49. Общая схема элемента `<regionBroker>`

Дочерние элементы элемента <*services*>

Рис. 50. Общая схема элемента `<services>`

bhv Машнин Т. С. Web-сервисы Java. – Спб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<brokerService>` определяет конфигурацию объекта `org.apache.activemq.broker.BrokerService`, обеспечивающего жизненный цикл брокера (рис. 51).

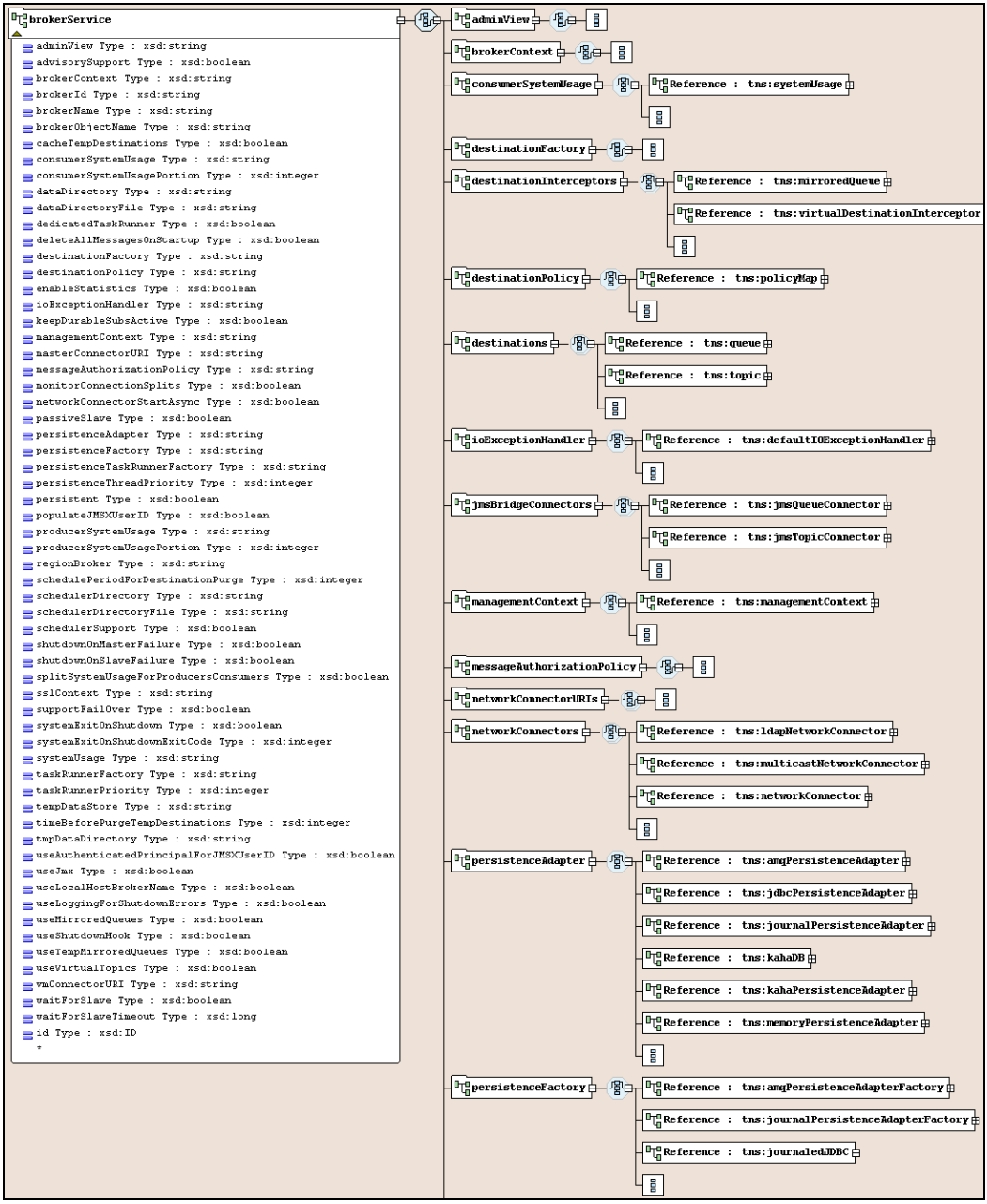


Рис. 51. Общая схема элемента `<brokerService>`

bhv Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

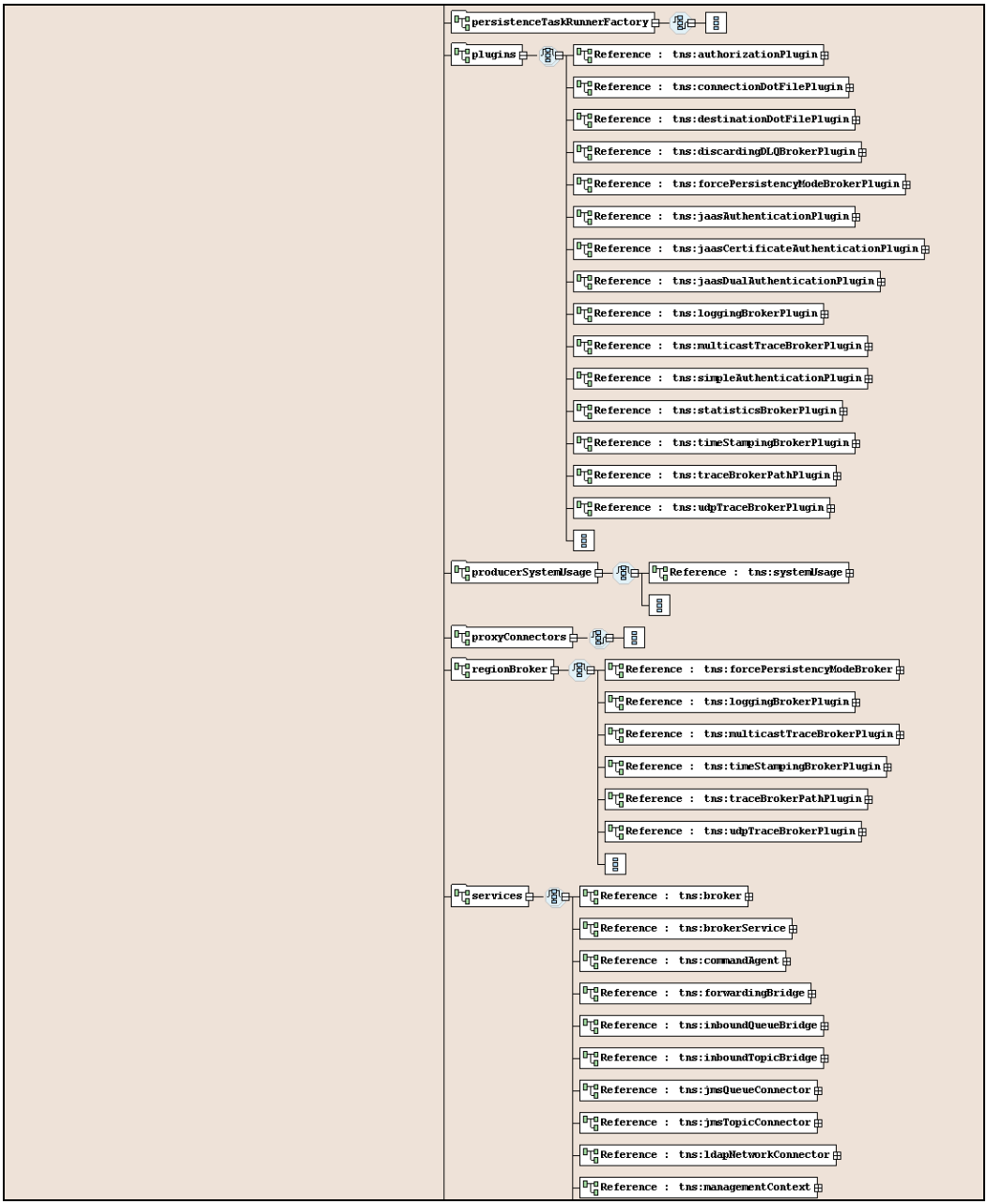


Рис. 51 *(продолжение)*

bhv Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

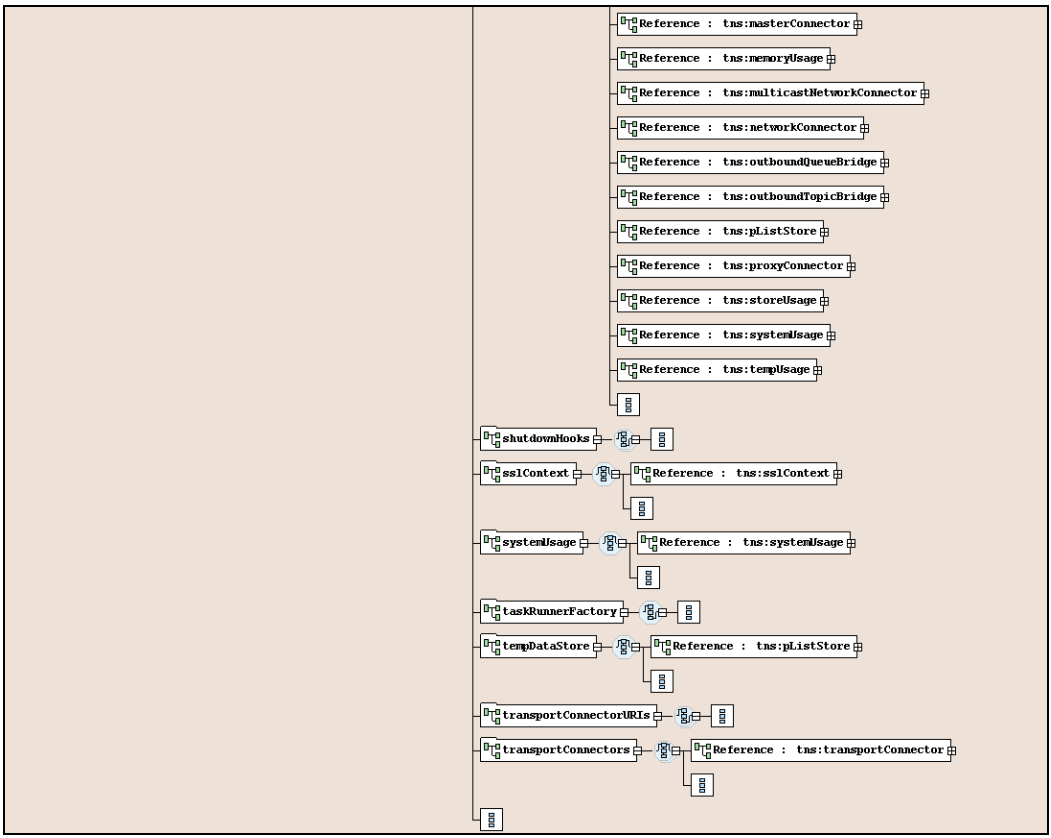


Рис. 51 *(окончание)*

Дочерние элементы элемента <*sslContext*>

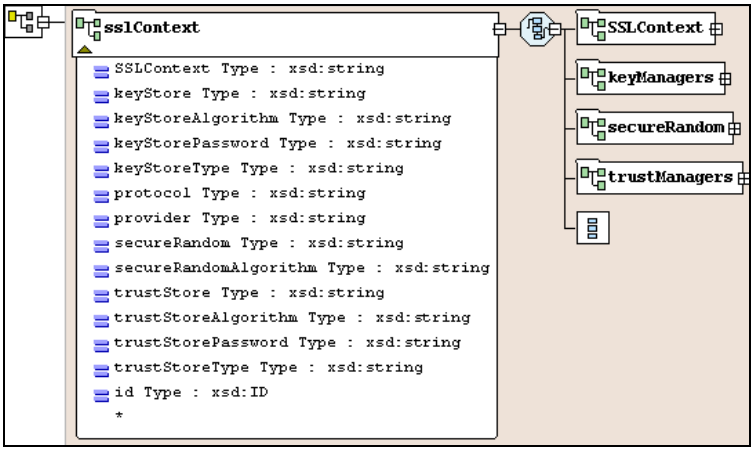


Рис. 52. Общая схема элемента `<sslContext>`

Протокол SOAP 1.1

Корневой элемент SOAP 1.1 сообщения `<SOAP-ENV:Envelope>` (пространство имен `xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"`) может содержать дочерние элементы `<SOAP-ENV:Header>` и должен включать в себя элемент `<SOAP-ENV:Body>`. Кроме того, возможно наличие дополнительных элементов, имеющих свои пространства имен и следующих сразу после элемента `<SOAP-ENV:Body>`.

Все элементы SOAP 1.1 сообщения могут иметь атрибут `encodingStyle`, содержащий перечень URI-идентификаторов правил кодирования SOAP-сообщения. Спецификация предопределяет кодировку с URI-идентификатором `"http://schemas.xmlsoap.org/soap/encoding/"`.

Элемент `<SOAP-ENV:Header>` содержит дочерние элементы SOAP-расширения, относящиеся к системе безопасности, надежности, поддержке транзакций и т. д. Каждый дочерний элемент элемента `<SOAP-ENV:Header>` может иметь атрибуты:

- `actor` — URI-адрес получателя элемента. Если данный атрибут отсутствует, тогда получатель сообщения является конечным получателем элемента заголовка. Значение атрибута `"http://schemas.xmlsoap.org/soap/actor/next"` указывает, что элемент заголовка обрабатывается первым SOAP-приложением;
- `mustUnderstand` — возможные значения: 0 (по умолчанию) или 1 (указывает обязательность обработки).

Дочерние элементы элемента `<SOAP-ENV:Header>` могут также иметь атрибут `root` (пространство имен `http://schemas.xmlsoap.org/soap/encoding/`). Значение атрибута `root="0"` указывает, что элемент сериализуется как независимый элемент.

Элемент `<SOAP-ENV:Body>` является контейнером для дочерних элементов, содержащих пересылаемую информацию. Дочерние элементы элемента `<SOAP-ENV:Body>` могут иметь атрибут `root` (возможное значение — 0 или 1).

Элемент `<SOAP-ENV:Body>` также может содержать элемент `<SOAP-ENV:Fault>`, который используется для передачи информации об ошибках или о статусе.

Элемент `<SOAP-ENV:Fault>` может содержать следующие элементы:

- `<faultcode>` — содержит код ошибки. Спецификация предопределяет следующие ошибки:
  - `VersionMismatch` — неправильное пространство имен SOAP;
  - `MustUnderstand` — при значении атрибута `mustUnderstand="1"` элемент заголовка не может быть обработан;
  - `Client` — сообщение неверно оформлено или несет некорректную информацию;
  - `Server` — сообщение не может быть обработано получателем;

- `<faultstring>` — текстовая информация об ошибке;
- `<faultactor>` — источник ошибки;
- `<detail>` — дополнительная информация об ошибке.

Передача SOAP-сообщения по протоколу HTTP подразумевает определение MIME-типа как `text/xml`.

При передаче SOAP-сообщения по протоколу HTTP-методом POST спецификация определяет HTTP-заголовок `SOAPAction`, указывающий URI-адрес получателя запроса.

При использовании SOAP-сообщений для удаленного вызова процедур RPC вызов метода и ответ представляются вложенным элементом элемента `<SOAP-ENV:Body>`. При этом вложенный элемент имеет имя, идентичное имени вызываемого метода, а его дочерние элементы представляют входящие и выходящие параметры метода и имеют соответственно имена, идентичные именам параметров метода. В случае ответа на вызов метода, первый дочерний элемент представляет возвращаемый результат, а далее следуют дочерние элементы, представляющие входящие и выходящие параметры метода согласно его сигнатуре. Имя элемента, несущего ответ на вызов метода, составлено из имени вызванного метода и "Response". Кроме того, при возникновении ошибки вызова метода информация об ошибке содержится в элементе `<SOAP-ENV:Fault>`.

Спецификация SOAP Messages with Attachments определяет механизм передачи SOAP-сообщений с вложениями, которые могут нести данные в двоичном формате.

Для передачи SOAP-сообщения с вложениями, оно упаковывается внутри MIME-сообщения `multipart/related`, которое состоит из MIME-заголовков, первичного SOAP-сообщения и вложений:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
        start="ID-start"
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: ID-start

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
. . .
<Named-element href="ID-attachment"/>
. . .
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: ID-attachment
...binary TIFF image...
--MIME_boundary—
```

Основные отличия протокола SOAP 1.1 от SOAP 1.2:

- SOAP 1.1 основывается на XML 1.0, а SOAP 1.2 — на XML Infoset;
- SOAP 1.2 не разрешает наличие дополнительных дочерних элементов после элемента `<SOAP-ENV:Body>`;
- в SOAP 1.2 атрибут `encodingStyle` может использоваться только в дочерних элементах SOAP Body, Header и fault Detail;
- SOAP 1.2 определяет новые заголовки `NotUnderstood` и `Upgrade`;
- в SOAP 1.2 атрибут `actor` переименован в `role`, и к роли Next добавлены две предопределенные роли — None и Ultimate Receiver;
- в SOAP 1.2 добавлены стандартные ошибки `DataEncodingUnknown`, `ProcedureNotPresent` и `BadArguments`. Ошибка `Client` переименована в `Sender`, а `Server` — в `Receiver`;
- в SOAP 1.2 изменена структура элемента `<Fault>`;
- в SOAP 1.2 отсутствует атрибут `root`;
- в SOAP 1.2 возвращаемый RPC-результат представлен элементом `<rpc:result>`;
- в SOAP 1.2 при передаче SOAP-сообщений по HTTP-протоколу MIME-тип `text/xml` изменен на `application/soap+xml`, а заголовок `SOAPAction` заменен дополнительным параметром `action`;
- в SOAP 1.2 добавлена поддержка HTTP-метода GET;
- в SOAP 1.2 атрибут `mustUnderstand` имеет значение `true` или `false`, а не 1 или 0;
- для заголовков в SOAP 1.2 определен новый атрибут `relay`;
- в SOAP 1.2 определена передача SOAP-сообщений по протоколу SMTP.

Спецификация XML Encryption

Спецификация XML Encryption описывает механизм шифрования данных и представления их в XML-документе.

Процесс шифрования может быть применен ко всему XML-документу, к элементам XML-документа и к содержимому элементов XML-документа. При этом данные шифруются с помощью одного из блочных симметричных криптографических алгоритмов Triple DES (3DES) или Advanced Encryption Standard (AES) с использованием симметричного ключа. Для передачи ключа шифрования от шифрующей стороны дешифрующей стороне используется его асимметричное шифрование алгоритмом RSA с применением публичного ключа или протокол передачи ключей Diffie-Hellman key exchange (DH). Дешифрование данных производится с помощью общего симметричного ключа после его расшифровки.

Алгоритм Triple DES — симметричный блочный алгоритм шифрования, созданный на основе алгоритма DES, имеющий длину ключа 168 бит и размер блока шифрования — 64 бита (эффективных 56 бит). Первая часть ключа шифрует первый блок, вторая часть ключа дешифрует второй блок, а третья часть ключа — шифрует третий блок данных.

Алгоритм AES представляет собой также симметричный блочный алгоритм шифрования, оперирующий 128-битным блоком данных. Существуют варианты AES-шифрования с ключами длиной 128, 192 и 256 бит. Кроме того, алгоритм AES может использоваться вместе с режимом GCM (Galois/Counter Mode), что эквивалентно одновременно шифрованию и созданию подписи HMAC, обеспечивая, таким образом, одновременно шифрование и аутентификацию данных.

Размер данных, подлежащих шифрованию, может быть не кратным размеру блока шифрования, поэтому перед шифрованием данные дополняются до блока шифрования последовательностью произвольных байтов с конечным байтом, значение которого представляет собой количество вставленных байтов.

Симметричный ключ шифрования может быть производным от оригинального ключа и получен из оригинального ключа с помощью алгоритма ConcatKDF или PBKDF2.

ConcatKDF-ключ составляется из хэш-значений, полученных хэшированием исходного ключа, счетчика итерации и параметров алгоритма. В качестве параметров алгоритма выступают:

- `AlgorithmID` — битовая строка, указывающая, как производный ключ должен анализироваться и для чего он предназначается, например, что биты 81—208 используются в качестве AES-ключа;
- `PartyUInfo` — информация о стороне, создающей производный ключ;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `PartyVInfo` — информация о стороне, использующей производный ключ;
- `SuppPubInfo` и `SuppPrivInfo` — дополнительная информация о производном ключе.

Алгоритм PBKDF2 принимает в качестве параметров исходный ключ и произвольное значение, которые затем хэширует определенное количество раз.

Для передачи расшифровывающей стороне симметричный ключ шифрования сам может зашифровываться алгоритмом RSA или RSA совместно с OAEP. Optimal Asymmetric Encryption Padding (OAEP) — алгоритм дополнительного шифрования путем дополнения исходных данных фиктивными данными.

Также передача симметричного ключа может осуществляться с помощью алгоритма Diffie-Hellman key agreement (DH), который может быть применен к оригинальному симметричному ключу или к производному ключу, или с применением алгоритма Elliptic Curve Diffie-Hellman (ECDH) Key Agreement, представляющего собой улучшенный вариант DH, основанный на алгебраической структуре эллиптических кривых. Суть алгоритма DH заключается в том, что обе стороны, применяющие симметричный ключ, выбирают по значению, которое используют как степень преобразования общего ключа, и передают друг другу полученные значения. Симметричный ключ получается следующим преобразованием с применением числа, которое было выбрано каждой из сторон.

Зашифрованные данные и зашифрованный симметричный ключ представляются в XML-документе с помощью элементов `<EncryptedData>` и `<EncryptedKey>` соответственно. При этом используются пространства имен `http://www.w3.org/2001/04/xmlenc#` (версия спецификации 1.0) и `http://www.w3.org/2009/xmlenc11#` (версия спецификации 1.1).

Элемент `<EncryptedData>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `Id` — идентификатор элемента;
- необязательный атрибут `Type` — тип зашифрованных данных. Возможные значения: `http://www.w3.org/2001/04/xmlenc#Element` (шифруется XML-элемент) и `http://www.w3.org/2001/04/xmlenc#Content` (шифруется содержимое XML-элемента);
- необязательный атрибут `MimeType` — MIME-тип зашифрованных данных;
- необязательный атрибут `Encoding` — кодировка данных (например, `http://www.w3.org/2000/09/xmldsig#base64`);
- необязательный элемент `<EncryptionMethod>` — метод шифрования;
- необязательный элемент `<ds:KeyInfo>` (пространство имен `http://www.w3.org/2000/09/xmldsig#`) — информация о ключе шифрования;
- обязательный элемент `<CipherData>` содержит зашифрованные данные;
- необязательный элемент `<EncryptionProperties>` — дополнительная информация о шифровании (дата шифрования, информация о программном обеспечении и т. д.).

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<EncryptionMethod>` описывает алгоритм шифрования данных. Обязательный атрибут `Algorithm` содержит в качестве значения URI-идентификатор алгоритма шифрования. Спецификация предопределяет следующие URI-идентификаторы для алгоритмов шифрования:

- `http://www.w3.org/2001/04/xmlenc#tripledes-cbc` — алгоритм Triple DES;
- `http://www.w3.org/2001/04/xmlenc#aes128-cbc`,
  `http://www.w3.org/2001/04/xmlenc#aes192-cbc`,
  `http://www.w3.org/2001/04/xmlenc#aes256-cbc` — алгоритм AES;
- `http://www.w3.org/2009/xmlenc11#aes128-gcm`,
  `http://www.w3.org/2009/xmlenc11#aes256-gcm` — алгоритм AES-GCM.

Элемент `<CipherData>` содержит последовательность зашифрованных данных в формате Base64 в элементе `<CipherValue>` или ссылку на ресурс, содержащий зашифрованные данные. Ссылка на внешний ресурс осуществляется с помощью вложенного элемента `<CipherReference>`.

Элемент `<CipherReference>` имеет обязательный атрибут `URI`, указывающий URI-адрес внешнего ресурса. Кроме того, элемент `<CipherReference>` может содержать вложенный элемент `<Transforms>`, описывающий преобразования данных, которые необходимо произвести для извлечения зашифрованных данных из внешнего ресурса. Элемент `<Transforms>` содержит вложенные элементы `<ds:Transform>` (пространство имен `http://www.w3.org/2000/09/xmldsig#`), структура которых описывается спецификацией XML Signature.

Элемент `<ds:KeyInfo>` (пространство имен `http://www.w3.org/2000/09/xmldsig#`), структура которого регламентируется спецификацией XML Signature, описывает ключ шифрования данных. Как правило, для элемента `<EncryptedData>` в элементе `<ds:KeyInfo>` используются вложенные элементы `<ds:KeyName>` и `<ds:RetrievalMethod>`. Элемент `<ds:RetrievalMethod>` с помощью атрибутов `URI` и `Type` (`http://www.w3.org/2001/04/xmlenc#EncryptedKey`) указывает ссылку на элемент `<EncryptedKey>`, описывающий ключ шифрования. Хотя дополнительно к спецификации XML Signature элемент `<ds:KeyInfo>` может содержать элементы `<EncryptedKey>` и `<AgreementMethod>`, непосредственно в элементе `<EncryptedData>` описывающие ключ шифрования. Элемент `<ds:RetrievalMethod>`, также используя атрибуты `URI` и `Type` (`http://www.w3.org/2009/xmlenc11#DerivedKey`), может ссылаться на элемент `<DerivedKey>`. Элемент `<DerivedKey>` может и напрямую содержаться в элементе `<ds:KeyInfo>`, описывая производный ключ.

Элемент `<EncryptedKey>` может представлять отдельное описание ключа шифрования или быть дочерним элементом элемента `<ds:KeyInfo>` и описывать ключ шифрования непосредственно в элементе `<EncryptedData>`. Элемент `<EncryptedKey>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `Id` — идентификатор элемента;
- необязательный атрибут `Recipient` — получатель ключа шифрования;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- необязательный элемент `<EncryptionMethod>` — метод шифрования симметричного ключа. Может содержать элементы `<KeySize>` (размер ключа) и `<OAEPparams>` (параметры шифрования алгоритма RSA-OAEP). Спецификация предопределяет следующие значения атрибута `Algorithm` элемента `<EncryptionMethod>`:
  - `http://www.w3.org/2001/04/xmlenc#rsa-1_5` — алгоритм RSA-v1.5;
  - `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p` — алгоритм RSA-OAEP;
- необязательный элемент `<ds:KeyInfo>` (пространство имен `http://www.w3.org/2000/09/xmldsig#`) — информация о ключе шифрования. Помимо стандартных элементов спецификации XML Signature может содержать элемент `<AgreementMethod>`;
- обязательный элемент `<CipherData>` содержит зашифрованный ключ. Содержит дочерний элемент `<CipherValue>` или `<CipherReference>`;
- необязательный элемент `<EncryptionProperties>` — дополнительная информация о шифровании ключа (дата шифрования, информация о программном обеспечении и т. д.);
- необязательный элемент `<ReferenceList>` содержит ссылки на данные, которые были зашифрованные этим ключом. Это могут быть элементы `<EncryptedData>` и `<EncryptedKey>` (зашифрованный ключ может быть повторно зашифрован). Дочерние элементы `<DataReference>` указывают ссылки на элементы `<EncryptedData>` с помощью атрибута `URI`, а дочерние элементы `<KeyReference>` указывают ссылки на элементы `<EncryptedKey>`, также используя атрибут `URI`;
- необязательный элемент `<CarriedKeyName>` содержит идентификатор зашифрованного ключа.

Элемент `<DerivedKey>` (пространство имен `http://www.w3.org/2009/xmlenc11#`) может представлять отдельное описание производного ключа шифрования или быть дочерним элементом элемента `<ds:KeyInfo>` и описывать производный ключ шифрования непосредственно в элементе `<EncryptedData>`. Элемент `<DerivedKey>` имеет следующие атрибуты и дочерние элементы:

- необязательный атрибут `Id` — идентификатор элемента;
- необязательный атрибут `Recipient` — получатель производного ключа шифрования;
- необязательный элемент `<KeyDerivationMethod>` (пространство имен `http://www.w3.org/2009/xmlenc11#`) описывает метод получения производного ключа из оригинального ключа. Спецификация предопределяет следующие значения атрибута `Algorithm` элемента `<KeyDerivationMethod>`:
  - `http://www.w3.org/2009/xmlenc11#ConcatKDF` — алгоритм ConcatKDF. В этом случае элемент `<KeyDerivationMethod>` содержит элемент `<ConcatKDFParams>` (параметры алгоритма) с вложенными элементами `<ds:DigestMethod>` (`http://www.w3.org/2000/09/xmldsig#sha1`, `http://www.w3.org/2001/04/xmlenc#sha256`,

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

http://www.w3.org/2001/04/xmlenc#sha384, http://www.w3.org/2001/04/xmlenc#sha512), <AlgorithmID>, <PartyUInfo>, <PartyVInfo>, <SuppPubInfo> и <SuppPrivInfo>;

- http://www.w3.org/2009/xmlenc11#pbkdf2 — алгоритм PBKDF2. В этом случае элемент <KeyDerivationMethod> содержит элемент <PBKDF2-params> (параметры алгоритма) с вложенными элементами <Salt> (дочерний элемент <Specified> или <OtherSource>), <IterationCount>, <KeyLength>, <PRF> (атрибут Algorithm);

- необязательный элемент <ReferenceList> содержит ссылки на данные, которые были зашифрованные этим ключом. Дочерние элементы <DataReference> указывают ссылки на элементы <EncryptedData> с помощью атрибута URI, а дочерние элементы <KeyReference> указывают ссылки на элементы <EncryptedKey>, также используя атрибут URI;
- необязательный элемент <DerivedKeyName> — идентификатор производного ключа;
- необязательный элемент <MasterKeyName> — идентификатор оригинального ключа.

Элемент <AgreementMethod> является дочерним элементом элемента <ds:KeyInfo> и описывает алгоритм передачи общего ключа шифрования на основе алгоритма Key Agreement. Спецификация предопределяет следующие значения атрибута Algorithm элемента <AgreementMethod>:

- http://www.w3.org/2009/xmlenc11#dh-es — алгоритм Diffie-Hellman Key Agreement with Explicit Key Derivation Functions. В этом случае элемент <ds:KeyValue> содержит элемент <DHKeyValue> с вложенными элементами <P>, <Q>, <Generator>, <Public>, <seed> и <pgenCounter>;
- http://www.w3.org/2001/04/xmlenc#dh — алгоритм Diffie-Hellman Key Agreement with Legacy Key Derivation Function. В этом случае также элемент <ds:KeyValue> содержит элемент <DHKeyValue> с вложенными элементами <P>, <Q>, <Generator>, <Public>, <seed> и <pgenCounter>;
- http://www.w3.org/2009/xmlenc11#ECDH-ES — алгоритм Elliptic Curve Diffie-Hellman (ECDH). В этом случае элемент <ds:KeyValue> содержит элемент <ECKeyValue>.

Элемент <AgreementMethod> может иметь следующие дочерние элементы:

- <KA-Nonce> содержит значение DH-параметра, гарантирующего уникальность генерируемого общего ключа;
- <KeyDerivationMethod> (пространство имен http://www.w3.org/2009/xmlenc11#) описывает алгоритм создания производного ключа;
- <OriginatorKeyInfo> содержит информацию от отправителя документа, необходимую для создания общего ключа. Структура элемента аналогична структуре элемента <ds:KeyInfo>;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `<RecipientKeyInfo>` содержит информацию от получателя документа, необходимую для создания общего ключа. Структура элемента аналогична структуре элемента `<ds:KeyInfo>`.

При шифровании данные сериализуются в формате UTF-8, после этого Encryption-процессор должен получить ключ шифрования и создать элементы `<ds:KeyInfo>`, `<EncryptedKey>` или `<DerivedKey>`. Далее данные шифруются и создается элемент `<EncryptedData>`. Дешифрование заключается в получении Decryption-процессором информации о ключе из элемента `<ds:KeyInfo>` и затем дешифровании данных элемента `<CipherData>`.

Спецификация XML Signatures

Спецификация XML Signatures определяет правила обработки и синтаксис цифровой подписи XML-документов.

Цифровая подпись XML-документа — это битовое значение в формате Base64, полученное с помощью определенного алгоритма преобразования с использованием закрытого ключа, которое было применено к битовому значению, возвращенному в результате применения хэш-функции к XML-документу. Таким образом, цифровая подпись обеспечивает целостность XML-документа — хэш-значение изменяется при изменении документа, и аутентификацию документа — при преобразовании используется закрытый ключ.

Хэширование XML-документа производится с помощью алгоритма SHA-1, а получение цифровой подписи — с помощью алгоритма DSA или RSA. В результате применения хэш-функции SHA-1 к XML-документу получается 160-битовая строка, которая затем преобразуется с применением алгоритма DSA или RSA.

Алгоритм RSA (Rivest, Shamir и Adleman) — это асимметричный алгоритм шифрования данных с использованием пары "закрытый ключ — открытый ключ". Алгоритм RSA может использоваться как для получения цифровой подписи, так и для шифрования данных. Так как в алгоритме RSA размер блока шифруемых данных ограничен размером ключа, то RSA-шифрование применяется к ключу симметричного шифрования, который шифруется с помощью алгоритма RSA с использованием открытого RSA-ключа. Затем зашифрованный симметричный ключ передается вместе с зашифрованными данными получателю, который расшифровывает симметричный ключ с помощью закрытого RSA-ключа и далее расшифровывает данные симметричным ключом.

При получении цифровой подписи отправитель хэширует передаваемые данные, зашифровывает алгоритмом RSA полученное хэш-значение с использованием закрытого RSA-ключа и посылает данные вместе с зашифрованным хэш-значением получателю. Получатель расшифровывает хэш-значение открытым RSA-ключом и сравнивает расшифрованное хэш-значение с хэш-значением, которое адресат переданных данных получает их хэшированием. Если два хэш-значения совпадают, тогда данные не были изменены в процессе передачи. Так обеспечивается целостность данных. Также, если зашифрованное хэш-значение расшифровывается открытым RSA-ключом, тогда хэш-значение зашифровывал именно тот отправитель, который обладает закрытым RSA-ключом. Таким образом производится аутентификация отправителя данных.

Алгоритм DSA (Digital Signature Algorithm) — это, так же как и RSA, асимметричный алгоритм шифрования данных с использованием пары "закрытый ключ — открытый ключ". Однако в отличие от RSA, алгоритм DSA применяется только к

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

хэш-значению, полученному хэшированием данных, поэтому DSA используется лишь для получения цифровой подписи. При использовании алгоритма DSA отправитель зашифровывает с помощью DSA хэш-значение данных закрытым DSA-ключом, а получатель расшифровывает его открытым DSA-ключом, чтобы затем сравнить два хэш-значения и проверить целостность данных.

Открытые ключи, используемые для расшифровки хэш-значения данных, могут быть заверены цифровым сертификатом, который подтверждает принадлежность открытого ключа его владельцу. В этом случае получателю данных передается не открытый ключ, а цифровой сертификат, содержащий информацию о владельце открытого ключа, открытый ключ и сведения о стороне, выдавшей сертификат.

Помимо цифровой подписи с данными может передаваться HMAC-значение, которое также подтверждает целостность данных и обеспечивает их аутентификацию. HMAC-значение (Hashed Message Authentication Code) получается хэшированием хэш-значения данных и общего (симметричного) для отправителя и получателя ключа. После получения таких данных адресат вычисляет свое HMAC-значение с использованием ключа и сравнивает его с переданным HMAC-значением. Если два значения совпадают, тогда целостность данных не была нарушена. Для защиты общего ключа может применяться механизм Kerberos, который основан на выдаче третьей стороной отправителю и получателю Kerberos-билетов, содержащих временный ключ. Kerberos-билет представляет собой данные, включающие в себя временный ключ и зашифрованные с помощью общего для третьей стороны и отправителя или получателя ключа.

Подпись XML-документов описывается с помощью набора XML-элементов, принадлежащих пространству имен `http://www.w3.org/2000/09/xmldsig#`.

Корневым элементом описания подписи является элемент `<Signature>`, имеющий дополнительный атрибут `Id` — идентификатор элемента, позволяющий использовать несколько подписей в XML-документе.

Элемент `<SignatureValue>` содержит значение цифровой подписи или HMAC-значение в формате Base64.

Элемент `<SignedInfo>` описывает, как была получена цифровая подпись и дополнительные данные, которые снабжаются цифровой подписью. Элемент `<SignedInfo>` имеет дополнительный атрибут `Id` (идентификатор элемента) и вложенные элементы `<CanonicalizationMethod>`, `<SignatureMethod>` и `<Reference>`.

Элемент `<CanonicalizationMethod>` — обязательный элемент, указывающий алгоритм канонизации, примененный к данным перед созданием цифровой подписи.

Одна и та же информация может быть представлена отличающимися друг от друга XML-документами, поэтому перед созданием цифровой подписи XML-документ необходимо привести к канонической форме для того, чтобы последующая проверка цифровой подписи была успешной, т. к. создание цифровой подписи и ее проверка должны применяться к одному и тому же XML-документу.

Спецификация XML Signatures указывает, что поддержка алгоритмов канонизации Canonical XML 1.0 и Canonical XML 1.1 должна быть обязательной, а поддержка

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

алгоритмов канонизации Canonical XML 1.0 with Comments и Canonical XML 1.1 with Comments — рекомендованной.

Алгоритм канонизации указывается с помощью значения атрибута `Algorithm` элемента `<CanonicalizationMethod>`:

- `http://www.w3.org/TR/2001/REC-xml-c14n-20010315` — Canonical XML 1.0;
- `http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments` — Canonical XML 1.0 with Comments;
- `http://www.w3.org/2006/12/xml-c14n11` — Canonical XML 1.1;
- `http://www.w3.org/2006/12/xml-c14n11#WithComments` — Canonical XML 1.1 with Comments.

Элемент `<SignatureMethod>` — обязательный вложенный элемент элемента `<SignedInfo>`, указывает алгоритм, который был применен для получения значения цифровой подписи, с помощью значения атрибута `Algorithm`:

- `http://www.w3.org/2000/09/xmldsig#dsa-sha1` — обязательный для поддержки алгоритм DSA;
- `http://www.w3.org/2000/09/xmldsig#rsa-sha1` — рекомендованный для поддержки алгоритм RSA-SHA1;
- `http://www.w3.org/2000/09/xmldsig#hmac-sha1` — обязательный для поддержки алгоритм HMAC.

В случае применения алгоритма HMAC для получения значения элемента `<SignatureValue>` элемент `<SignatureMethod>` может содержать вложенный элемент `<HMACOutputLength>`, значение которого определяет длину (усечение) HMAC-значения.

Дополнительный вложенный элемент `<Reference>` элемента `<SignedInfo>` описывает дополнительные данные, включаемые в цифровую подпись. Элемент `<Reference>` может иметь следующие атрибуты:

- `Id` — идентификатор элемента;
- `URI` — идентификатор объекта данных, включаемых в цифровую подпись;
- `Type` — тип объекта данных в виде URI-идентификатора.

Кроме того, элемент `<Reference>` имеет следующие дочерние элементы:

- `<Transforms>` — необязательный элемент, содержит последовательность элементов `<Transform>`, которые указывают преобразования, произведенные над объектом данных перед его хэшированием. Элемент `<Transform>` имеет атрибут `Algorithm`, определяющий алгоритм преобразования, и может содержать параметры преобразования. Спецификация XML Signatures рекомендует для поддержки следующие алгоритмы трансформации:
  - `http://www.w3.org/TR/2001/REC-xml-c14n-20010315` — канонизация Canonical XML 1.0;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments — канонизация Canonical XML 1.0 with Comments;
- http://www.w3.org/2006/12/xml-c14n11 — канонизация Canonical XML 1.1;
- http://www.w3.org/2006/12/xml-c14n11#WithComments — канонизация Canonical XML 1.1 with Comments;
- http://www.w3.org/2000/09/xmldsig#base64 — форматирование Base64;
- http://www.w3.org/TR/1999/REC-xpath-19991116 — XPath-фильтрация. Элемент <Transform> содержит элемент <XPath>, описывающий XPath-выражения, которые преобразовали объект данных;
- http://www.w3.org/2000/09/xmldsig#enveloped-signature удаляет элемент <Signature> из хэширования объекта данных, если он его содержал;
- http://www.w3.org/TR/1999/REC-xslt-19991116 — XSLT-преобразование. Элемент <Transform> содержит элемент <stylesheet>, описывающий таблицу стилей XSLT-преобразования;

- <DigestMethod> — обязательный элемент, указывает алгоритм хэширования объекта данных с помощью атрибута Algorithm. Спецификация XML Signatures требует поддержки алгоритма SHA1 со значением атрибута Algorithm= "http://www.w3.org/2000/09/xmldsig#sha1";
- <DigestValue> — обязательный элемент, содержит хэш-значение объекта данных в формате Base64.

Элемент <KeyInfo> является необязательным дочерним элементом элемента <Signature> и содержит информацию о публичном ключе цифровой подписи. Элемент <KeyInfo> служит контейнером для следующих элементов:

- <KeyName> — имя ключа;
- <KeyValue> — сам публичный ключ. Спецификация XML Signatures определяет вложенные элементы <DSAKeyValue> и <RSAKeyValue> для передачи публичных DSA- и RSA-ключей соответственно;
- <RetrievalMethod> организует ссылку на информацию о публичном ключе, которая может быть определена вне элемента <KeyInfo>. Элемент <RetrievalMethod> имеет атрибуты URI (идентификатор объекта, на который ссылаются) и Type (URI-идентификатор типа объекта ссылки), а также вложенный элемент <Transforms>, указывающий преобразования объекта ссылки;
- <X509Data> описывает X509-сертификат публичного ключа с помощью вложенных элементов <X509IssuerSerial>, <X509IssuerName>, <X509SerialNumber>, <X509SKI>, <X509SubjectName>, <X509Certificate>, <X509CRL>;
- <PGPData> описывает публичные ключи стандарта PGP с помощью вложенных элементов <PGPKeyID> и <PGPKeyPacket>;
- <SPKIData> описывает публичные ключи стандарта SPKI с помощью вложенных элементов <SPKISexp>;
- <MgmtData> содержит зашифрованные ключи или соглашения о ключах.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Элемент `<Object>` также является необязательным дочерним элементом элемента `<Signature>` и содержит дополнительные данные, на которые можно ссылаться, пользуясь значением атрибута `Id` элемента `<Object>`. Элемент `<Object>`, помимо атрибута `Id`, имеет атрибуты `MimeType` (тип данных) и `Encoding` (URI-идентификатор кодировки данных).

Дополнительный дочерний элемент `<Manifest>` элемента `<Object>` служит контейнером для элементов `<Reference>`, описывающих дополнительные данные, включаемые в цифровую подпись. В отличие от вложенных элементов `<Reference>` элемента `<SignedInfo>`, условия проверки хэш-значений элемента `<Manifest>` определяются получателем. То есть если один из элементов `<Reference>` элемента `<SignedInfo>` является ошибочным, тогда вся цифровая подпись неверна, а если какой-либо из элементов `<Reference>` элемента `<Manifest>` является ошибочным, верность цифровой подписи определяет получатель.

Дополнительный дочерний элемент `<SignatureProperties>` элемента `<Object>` содержит дополнительную информацию о цифровой подписи, например время действия цифровой подписи, роль владельца подписи и т. д.

Таким образом, создание цифровой подписи подразумевает следующие шаги:

1. Вычисляются значения элементов `<DigestValue>` элементов `<Reference>` элемента `<SignedInfo>`. Для этого используются алгоритмы элементов `<Transforms>`, затем алгоритмы элементов `<DigestMethod>`.
2. Вычисляется значение элемента `<SignatureValue>` с использованием элементов `<CanonicalizationMethod>`, `<SignatureMethod>` и `<Reference>` элемента `<SignedInfo>`, а также элементов `<Object>`.

Проверка цифровой подписи состоит из следующих действий:

1. Полученный документ приводится к канонической форме согласно алгоритму, указанному в элементе `<CanonicalizationMethod>`.
2. Вычисляются хэш-значения данных, указанных элементами `<Reference>` элемента `<SignedInfo>` с применением информации элементов `<Transforms>` и `<DigestMethod>`. Полученные значения сравниваются со значениями элементов `<DigestValue>`. Значения должны совпадать.
3. Документ обрабатывается согласно информации элементов `<SignedInfo>` и `<Object>`.
4. На основе информации элементов `<KeyInfo>` и `<SignatureMethod>` подтверждается значение элемента `<SignatureValue>`.

Язык XPath

Язык XPath (XML Path Language) — это язык выражений, обеспечивающий запросы к XML-документам.

Язык XPath используется различными XML-обработчиками для навигации по XML-документам и выбора их элементов согласно заданным критериям и интегрирован с различными технологиями, такими как технология XSLT (eXtensible Stylesheet Language Transformations), предназначенная для преобразования XML-документов, технология XPointer (XML Pointer Language), обеспечивающая адресацию в XML-документах, и др.

На сегодняшний день существуют две версии языка XPath — XPath 1.0 и XPath 2.0.

Язык XPath 2.0 создан на основе языка XQuery 1.0 и является дальнейшим развитием языка XPath 1.0, предоставляя более широкий набор операторов и функций, а также большие возможности по обработке типов данных.

Язык XPath 1.0

Язык XPath основывается на модели XML-документа, описывающей его как иерархическую структуру — XPath-дерево, состоящее из узлов семи типов:

- корневой узел;
- узлы элементов;
- текстовые узлы;
- узлы атрибутов;
- узлы пространств имен;
- узлы инструкций по обработке;
- узлы комментариев.

Язык XPath представляет собой выражения, которые после обработки XPath-процессором, в качестве результата возвращают узел (или набор узлов), отобранный согласно критериям, а также позволяют получить информацию о структуре и свойствах узлов. Чтобы решать эти задачи, язык XPath предлагает выражения различного типа.

Основной тип выражений языка XPath — это выражения пути, возвращающие узел или набор узлов, отобранный согласно критериям.

Выражения пути являются основой и могут содержать другие XPath-выражения, определяющие критерии выбора узлов и возвращающие информацию о структуре и свойствах узлов.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Синтаксис выражения пути базируется на последовательности, представляющей адрес узла или набора узлов в дереве. Соответственно такой адрес может быть абсолютным или относительным.

Выражение пути начинается с абсолютного адреса, состоящего из символа `/` и отправляющего XPath-процессор к корневому узлу. Далее следует относительный адрес, вычисляемый последовательно слева направо, соответственно шагам адресации, из которых относительный адрес состоит.

Каждый шаг адресации отделяется от предыдущего символом `/` и обеспечивает отбор узлов относительно предыдущего шага (контекста или уровня).

Синтаксис выражения пути бывает двух типов — полный и сокращенный.

Полный синтаксис шага адресации состоит из оси, определяющей взаимоотношения между отбираемыми на данном шаге узлами и текущим контекстом, условия отбора узлов, представленного типом и названием отбираемых узлов, утверждений, использующих выражения, которые уточняют отбор узлов на данном шаге.

Оси языка XPath представлены следующим набором:

- `ancestor::` — содержит всех предков текущего контекста;
- `ancestor-or-self::` — содержит текущий контекст и всех его предков;
- `attribute::` — содержит все атрибуты текущего контекста;
- `child::` — содержит множество потомков текущего контекста;
- `descendant::` — содержит полное множество потомков текущего уровня, включая "внуков" (т. е. потомков потомков) и т. д.;
- `descendant-or-self::` — содержит полное множество потомков и текущий контекст;
- `following::` — содержит все узлы документа после текущего контекста, за исключением потомков и узлов атрибутов и пространств имен;
- `following-sibling::` — содержит все узлы-братья после текущего контекста, за исключением узлов атрибутов и пространств имен;
- `namespace::` — содержит все узлы пространств имен текущего контекста;
- `parent::` — содержит предка на один уровень назад;
- `preceding::` — содержит все узлы документа перед текущим контекстом, за исключением предков и узлов атрибутов и пространств имен;
- `preceding-sibling::` — содержит все узлы-братья перед текущим контекстом, за исключением узлов атрибутов и пространств имен;
- `self::` — содержит текущий контекст.

Сокращенный синтаксис шага адресации позволяет сделать следующие замены:

- `attribute::` — можно заменить символом `@`;
- `child::` — можно опустить;
- `descendant-or-self::` — можно заменить символами `//`;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `parent::` — можно заменить двумя точками (`..`);
- `self::` — можно заменить точкой (`.`).

В XPath-выражении, после оси, следует условие отбора узлов, представленное типом и названием отбираемых узлов.

Условие отбора может иметь форму QName-имени узла, символа `*`, означающего выбор всех узлов для данной оси, а также функций `node()`, `text()`, `comment()` и `processing-instruction()`, означающих выбор любого узла, узлов текста, комментариев и инструкций по обработке соответственно.

После условия отбора в XPath-выражении следуют утверждения, которые уточняют или фильтруют отбор узлов на данном шаге. Утверждения ограничиваются квадратными скобками (`[]`) и могут состоять из:

- выражений пути;
- чисел, определяющих позицию выбираемого узла;
- выражений, содержащих операторы — арифметические `+`, `-`, `*`, `div` и `mod`, логические `and` и `or`, объединения `|`, сравнения `=`, `!=`, `<`, `>`, `<=`, `>=`;
- функций.

Язык XPath предлагает библиотеку стандартных функций, позволяющих получить информацию о контексте, свойствах узлов, а также работать со строками и числами.

Библиотека языка XPath 1.0 состоит из следующих функций:

- `last()` — возвращает число узлов контекста или размер контекста;
- `position()` — возвращает позицию узла в наборе узлов контекста;
- `count(`*`node-set`*`)` — возвращает число узлов в наборе узлов аргумента функции;
- `id(`*`object`*`)` — возвращает набор узлов, выбранных по их идентификатору;
- `local-name(`*`node-set`*`?)` — возвращает имя узла без пространства имен, первого в наборе узлов аргумента;
- `namespace-uri(`*`node-set`*`?)` — возвращает URI пространства имен узла, первого в наборе узлов аргумента;
- `name(`*`node-set`*`?)` — возвращает QName-имя узла, первого в наборе узлов аргумента;
- `string(`*`object`*`?)` — преобразовывает объект аргумента в строку;
- `concat(`*`string`*`, `*`string`*`, `*`string`*`*)` — объединяет строки аргументов;
- `starts-with(`*`string`*`, `*`string`*`)` — возвращает `true`, если вторая строка входит в начало первой, иначе возвращает `false`;
- `contains(`*`string`*`, `*`string`*`)` — возвращает `true`, если первая строка содержит вторую, иначе возвращает `false`;
- `substring-before(`*`string`*`, `*`string`*`)` — если найдена вторая строка в первой, возвращает подстроку первой строки до первого вхождения второй строки;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `substring-after(`*`string`*`, `*`string`*`)` — если найдена вторая строка в первой, возвращает подстроку первой строки после первого вхождения второй строки;
- `substring(`*`string`*`, `*`number`*`, `*`number`*`?)` — возвращает подстроку первого аргумента, начиная с позиции второго аргумента и с количеством символов третьего аргумента;
- `string-length(`*`string`*`?)` — возвращает длину строки аргумента;
- `normalize-space(`*`string`*`?)` — возвращает нормализованную строку аргумента с убранными пробелами, а так же замененными управляющими символами;
- `translate(`*`string`*`, `*`string`*`, `*`string`*`)` — возвращает строку, как результат замены символов строки первого аргумента, которые встречаются в строке второго аргумента, на соответствующие по позиции символам из строки второго аргумента символы из строки третьего аргумента;
- `boolean(`*`object`*`)` — преобразует объект аргумента в логическое значение;
- `not(`*`boolean`*`)` — возвращает `true`, если аргумент `false`;
- `true()` — возвращает `true`;
- `false()` — возвращает `false`;
- `lang(`*`string`*`)` — возвращает `true`, если язык, определяемый аргументом, совпадает с языком, определенным атрибутом `xml:lang` контекста;
- `number(`*`object`*`?)` — преобразует аргумент в число;
- `sum(`*`node-set`*`)` — возвращает число как сумму чисел, полученных в результате применения функции `number()` к каждому узлу набора аргумента;
- `floor(`*`number`*`)` — возвращает наибольшее целое число, не большее, чем аргумент;
- `ceiling(`*`number`*`)` — возвращает наименьшее целое число, не меньшее, чем аргумент;
- `round(`*`number`*`)` — возвращает число как результат округления аргумента.

Язык XPath 2.0

Язык XPath 2.0 опирается на новую, по сравнению с языком XPath 1.0, модель данных. Язык XPath 1.0 работает с четырьмя типами данных — наборы узлов, числа, строки, логические значения. В языке XPath 2.0 все данные — это последовательности.

Последовательность языка XPath 2.0 представляет собой упорядоченный набор объектов различных типов, которые могут быть атомарными значениями или узлами XML-документа. При этом последовательности не могут быть вложенными одна в другую. Любое выражение языка XPath 2.0 возвращает последовательность.

Узлы, входящие в последовательности, могут быть, так же как и в языке XPath 1.0, семи типов — узлы документов, узлы элементов, узлы атрибутов, узлы пространств имен, узлы инструкций по обработке, узлы комментариев и текстовые узлы.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

В языке XPath 2.0 корневой узел набора из семи типов узлов Xpath-дерева языка XPath 1.0 заменен узлом документа, т. к. язык XPath 2.0 позволяет, в отличие от языка XPath 1.0, для передачи результата от одного запроса другому генерировать деревья, корневые узлы которых могут отличаться от узла документа. Кроме того, узлы языка XPath 2.0 могут иметь тип данных (простой или сложный), определенный согласно схеме XML Schema.

XPath-дерево, корневым узлом которого является узел документа, называется *документом*, в противном случае XPath-дерево называется *фрагментом*.

Атомарные значения языка XPath 2.0 могут иметь один из 19 простых типов, поддерживаемых XML Schema — это `string`, `boolean`, `double`, `float`, `decimal`, `dateTime` и т. д., 5 дополнительных простых типов данных, а также типы данных, являющиеся производными ограничениями от этих простых типов.

Таким образом, язык XPath 2.0 значительно расширяет, по сравнению с языком XPath 1.0, возможности по обработке данных, обеспечивая полную поддержку типов данных XML Schema.

Каждый объект последовательности языка XPath 2.0 имеет значение и тип данных, который идентифицируется расширенным QName-именем (префикс + URI пространства имен + локальное имя) типа данных, поэтому можно сказать, что язык XPath 2.0 строго типизирован. Кроме того, каждый узел как объект последовательности, помимо значения и типа данных, имеет свой идентификатор. Это означает, что последовательность может содержать дубликаты, в отличие от наборов узлов языка XPath 1.0.

Пять вышеупомянутых дополнительных простых типов данных языка XPath 2.0 представлены следующим набором:

- `xs:untyped` — тип данных узла элемента, не имеющий описания в XML-схеме;
- `xs:untypedAtomic` — тип данных узла атрибута, не имеющий описания в XML-схеме;
- `xs:anyAtomicType` — абстрактный тип данных, являющийся базовым для всех типов данных атомарных значений;
- `xs:dayTimeDuration` — тип данных, представляющий последовательность из дня, часа, минут и секунд;
- `xs:yearMonthDuration` — тип данных, представляющий последовательность из года и месяца.

Язык XPath 2.0 поддерживает комментарии для документирования XPath-выражений. Такого рода комментарии представляют собой строки, ограниченные символами `(:` и `:)`, и не влияют на обработку XPath-выражений. Также язык XPath 2.0, в отличие от XPath 1.0, обеспечивает использование переменных в виде `$[`*имя_переменной*`]`, которым можно присваивать значения.

Самой важной конструкцией языка XPath 2.0, так же как и XPath 1.0, являются выражения пути, состоящие из шагов адресаций, разделенных оператором `/`, которые, в свою очередь, представляют собой последовательности из оси, условия отбора узлов и утверждений, уточняющих отбор узлов XML-документа.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

В языке XPath 2.0 присутствуют оси языка XPath 1.0, за исключением оси `namespace::`, использование данной оси заменено вызовом функций `in-scope-prefixes()` и `namespace-uri-for-prefix()`.

Сокращенный синтаксис шага адресации языка XPath 2.0 позволяет сделать следующие замены:

- `attribute::` — можно заменить символом `@`;
- `child::` — можно опустить;
- `descendant-or-self::` — можно заменить символами `//`;
- `parent::` — можно заменить двумя точками (`..`).

Точка (`.`) в языке XPath 2.0 является простым XPath-выражением (выражением контекста), а не сокращенным синтаксисом, как замена `self::` в языке XPath 1.0. Выражение контекста возвращает объект контекста, т. е. объект, который обрабатывается в данный момент. Соответственно объект контекста может быть узлом или атомарным значением, тогда как в языке XPath 1.0 есть только узел контекста.

В отличие от XPath 1.0, условие отбора узлов языка XPath 2.0 дополнено следующими функциями:

- `item()` — выбор любого объекта — узла или атомарного значения;
- `element()` — выбор любого узла элемента;
- `schema-element(`*имя элемента*`)` — выбор узла элемента с указанным именем и соответствующим типом данных, описанным в XML-схеме;
- `element(`*имя элемента*`)` — выбор узла элемента с указанным именем без привязки к типу данных;
- `element(`*имя элемента*`, `*тип данных*`)` — выбор узла элемента с указанным именем и типом данных;
- `element(*, `*тип данных*`)` — выбор любого узла элемента с указанным типом данных;
- `attribute()` — выбор любого узла атрибута;
- `attribute(`*имя атрибута*`)` — выбор узла атрибута с указанным именем без привязки к типу данных;
- `attribute(*, `*тип данных*`)` — выбор любого узла атрибута с указанным типом данных;
- `document-node()` — выбор любого узла документа;
- `document-node(element(`*имя элемента*`))` — выбор узла документа, содержащего узел элемента с указанным именем.

Таким образом, язык XPath 2.0, в отличие от XPath 1.0, позволяет осуществлять выбор узлов не только по их имени, но и по их типу данных.

Существенным отличием от языка XPath 1.0 также является возможность использования в языке XPath 2.0, в качестве шагов адресаций, любых XPath-выражений, возвращающих последовательность узлов XML-документа.

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Утверждения языка XPath 2.0, уточняющие отбор узлов в выражениях пути, представляют собой XPath-выражения, заключенные в квадратные скобки (`[]`).

Выражения пути языка XPath 2.0 возвращают упорядоченные последовательности узлов XML-документа, в которых исключены дубликаты.

XPath-выражения языка XPath 2.0 классифицируются как выражения пути, простые выражения, выражения последовательностей, арифметические выражения, выражения сравнения, логические выражения, выражения цикла, выражения условий, количественные выражения и выражения типа.

Простые выражения языка XPath 2.0 являются его основой и состоят из чисел, строк, переменных (`$QName`), XPath-выражений, заключенных в скобки (`()`), выражений контекста "`.`" (возвращает объект контекста) и вызовов функций.

Выражения последовательностей предназначены для обработки последовательностей языка XPath 2.0 и содержат операторы конструирования, фильтрации и объединения последовательностей объектов.

Для создания последовательностей применяется оператор "запятая" (`,`), связывающий объекты последовательности, а также оператор `to`, определяющий диапазон целых чисел (например, `1 to 10`).

Фильтрация последовательностей осуществляется с помощью утверждений — XPath-выражений, заключенных в квадратные скобки (`[]`). Утверждение, следующее за простым XPath-выражением, осуществляет фильтрацию последовательности, которую возвращает в качестве результата данное простое XPath-выражение.

Объединение последовательностей узлов возможно с использованием операторов:

- `union` и `|` — простое объединение последовательностей узлов;
- `intersect` — объединяет в последовательность, содержащую узлы, которые повторяются во всех объединяемых последовательностях;
- `except` — объединяет в последовательность, содержащую узлы, которые не повторяются во всех объединяемых последовательностях.

При объединении последовательностей происходит исключение дубликатов.

Арифметические выражения используются для выполнения вычислений с числами и датами и содержат операторы `+`, `-`, `*`, `div`, `idiv` (деление с округлением до целого числа) и `mod`, возвращающие последовательность, состоящую из одного объекта.

Выражения сравнения возвращают `true`/`false` и содержат:

- операторы сравнения двух величин:
  - `eq` — равно;
  - `ne` — не равно;
  - `lt` — меньше чем;
  - `le` — меньше чем или равно;
  - `gt` — больше чем;
  - `ge` — больше чем или равно;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- общие операторы сравнения двух последовательностей — `=`, `!=`, `<`, `<=`, `>`, `>=`;
- операторы сравнения двух узлов:
  - `is` — два объекта являются одним и тем же узлом;
  - `<<` — один узел перед другим в иерархии документа;
  - `>>` — один узел после другого в иерархии документа.

Логические выражения возвращают `true`/`false` и содержат операторы:

- `or` — возвращает `true`, если любой из объектов — `true`;
- `and` — возвращает `true`, если оба объекта — `true`.

Выражения цикла содержат конструкцию

`for` *`переменная_диапазона`* `in` *`XPath-выражение`* `return` *`XPath-выражение`*

XPath-выражение, следующее после `in`, возвращает последовательность, определяющую диапазон для переменной диапазона. XPath-выражение, следующее после `return`, содержит переменную диапазона и вычисляет объекты последовательности, которую в итоге возвращает выражение цикла.

Выражения условий содержат конструкцию

`if` *`XPath-выражение`* `then` *`XPath-выражение`* `else` *`XPath-выражение`*

Выражение условия означает следующее: если XPath-выражение, следующее после `if`, равно `true`, тогда выполняется XPath-выражение, следующее после `then`, иначе выполняется XPath-выражение, следующее после `else`.

Количественные выражения возвращают `true`/`false` и содержат конструкции

`some` *или* `every` *`переменная_диапазона`* `in` *`XPath-выражение`* `satisfies` *`XPath-выражение`*

XPath-выражение, следующее после `in`, определяет диапазон для переменной диапазона. XPath-выражение, следующее после `satisfies`, содержит переменную диапазона и возвращает `true`/`false` для каждого значения переменной в определенном диапазоне.

В случае применения ключевого слова `some` количественное выражение возвращает `true`, если XPath-выражение, следующее после `satisfies`, хотя бы один раз возвратит `true`.

В случае применения ключевого слова `every` количественное выражение возвращает `true`, если XPath-выражение, следующее после `satisfies`, для каждого значения переменной диапазона возвратит `true`.

Выражения типа состоят из следующих конструкций:

- *`XPath-выражение`* `instance of` *`тип`* — возвращает `true`, если объект, возвращаемый XPath-выражением, имеет указанный тип;
- *`XPath-выражение`* `cast as` *`атомарный тип данных`* — приводит тип объекта, возвращаемого XPath-выражением, к указанному атомарному типу данных;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- *XPath-выражение* `castable as` *атомарный тип данных* — возвращает `true`, если тип объекта, возвращаемый XPath-выражением, можно привести к указанному атомарному типу данных;
- *XPath-выражение* `treat as` *тип* — является утверждением, что тип объекта, возвращаемый XPath-выражением, соответствует указанному типу. Если утверждение верно, то выражение возвращает объект, если нет — возникает ошибка.

Язык XPath 2.0 содержит библиотеку стандартных функций для использования в перечисленных далее XPath-выражениях.

> ***Примечание***
>
> Префикс `xs` соответствует пространству имен `http://www.w3.org/2001/XMLSchema`. Префикс `fn` соответствует пространству имен `http://www.w3.org/2005/xpath-functions`. Префикс `err` соответствует пространству имен `http://www.w3.org/2005/xqt-errors`.

- Функции оценки:
  - `fn:node-name(`*имя узла*`)` — возвращает расширенное QName-имя узла;
  - `fn:nilled (`*имя узла*`)` — возвращает `true`, если узел элемента содержит атрибут `xsi:nil="true"`;
  - `fn:string(`*объект*`)` — конвертирует аргумент в строку. Если аргумент является узлом, тогда возвращает текстовое содержание узла;
  - `fn:data(`*объекты*`)` — конвертирует последовательность объектов в последовательность атомарных значений. Последовательность узлов конвертируется в последовательность связанных с ними значениями;
  - `fn:base-uri(`*имя узла*`)` — возвращает базовый URI-адрес узла;
  - `fn:document-uri(`*имя узла документа*`)` — возвращает URI-адрес документа.
- Функция ошибок `fn:error(`*сообщение об ошибке*`)` завершает вычисление XPath-выражения — используется в случае, когда приложение определяет ошибку.
- Функция анализа `fn:trace(`*сообщение*`)` — отладочная функция, выводит аргумент в диагностический поток (например, создается запись в файле регистрации).
- Функции-конструкторы для типов данных:
  - `xs:string($arg as xs:anyAtomicType?)`
  - `xs:boolean($arg as xs:anyAtomicType?)`
  - `xs:decimal($arg as xs:anyAtomicType?)`
  - `xs:float($arg as xs:anyAtomicType?)`
  - `xs:double($arg as xs:anyAtomicType?)`
  - `xs:duration($arg as xs:anyAtomicType?)`
  - `xs:dateTime($arg as xs:anyAtomicType?)`
  - `xs:time($arg as xs:anyAtomicType?)`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `xs:date($arg as xs:anyAtomicType?)`
- `xs:gYearMonth($arg as xs:anyAtomicType?)`
- `xs:gYear($arg as xs:anyAtomicType?)`
- `xs:gMonthDay($arg as xs:anyAtomicType?)`
- `xs:gDay($arg as xs:anyAtomicType?)`
- `xs:gMonth($arg as xs:anyAtomicType?)`
- `xs:hexBinary($arg as xs:anyAtomicType?)`
- `xs:base64Binary($arg as xs:anyAtomicType?)`
- `xs:anyURI($arg as xs:anyAtomicType?)`
- `xs:QName($arg as xs:anyAtomicType?)`
- `xs:normalizedString($arg as xs:anyAtomicType?)`
- `xs:token($arg as xs:anyAtomicType?)`
- `xs:language($arg as xs:anyAtomicType?)`
- `xs:NMTOKEN($arg as xs:anyAtomicType?)`
- `xs:Name($arg as xs:anyAtomicType?)`
- `xs:NCName($arg as xs:anyAtomicType?)`
- `xs:ID($arg as xs:anyAtomicType?)`
- `xs:IDREF($arg as xs:anyAtomicType?)`
- `xs:ENTITY($arg as xs:anyAtomicType?)`
- `xs:integer($arg as xs:anyAtomicType?)`
- `xs:nonPositiveInteger($arg as xs:anyAtomicType?)`
- `xs:negativeInteger($arg as xs:anyAtomicType?)`
- `xs:long($arg as xs:anyAtomicType?)`
- `xs:int($arg as xs:anyAtomicType?)`
- `xs:short($arg as xs:anyAtomicType?)`
- `xs:byte($arg as xs:anyAtomicType?)`
- `xs:nonNegativeInteger($arg as xs:anyAtomicType?)`
- `xs:unsignedLong($arg as xs:anyAtomicType?)`
- `xs:unsignedInt($arg as xs:anyAtomicType?)`
- `xs:unsignedShort($arg as xs:anyAtomicType?)`
- `xs:unsignedByte($arg as xs:anyAtomicType?)`
- `xs:positiveInteger($arg as xs:anyAtomicType?)`
- `xs:yearMonthDuration($arg as xs:anyAtomicType?)`
- `xs:dayTimeDuration($arg as xs:anyAtomicType?)`
- `xs:untypedAtomic($arg as xs:anyAtomicType?)`
- `fn:dateTime($arg1 as xs:date?, $arg2 as xs:time?)`

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- Функции обработки чисел:
  - `fn:abs(`*число*`)` — возвращает абсолютное значение числа;
  - `fn:ceiling(`*число*`)` — округляет число в большую сторону;
  - `fn:floor(`*число*`)` — округляет число в меньшую сторону;
  - `fn:round(`*число*`)` — округляет число;
  - `fn:round-half-to-even(`*число*`, `*точность*`)` — округляет число с указанной точностью.
- Функции обработки строк:
  - `fn:codepoints-to-string(`*числа*`)` — создает строку из последовательности Unicode-кодировки;
  - `fn:string-to-codepoints(`*строка*`)` — возвращает последовательность Unicode-кодировки, соответствующую символам строки;
  - `fn:compare(`*строка1*`, `*строка2*`, `*URI условия сопоставления*`)` — сравнивает две строки с учетом условия сопоставления символов, возвращает –1, 0 и +1, если первая строка меньше, равна или больше второй строки;
  - `fn:codepoint-equal(`*строка1*`, `*строка2*`)` — сравнивает две строки в соответствии с Unicode-кодировкой, возвращает `true`/`false`;
  - `fn:concat(`*атомарные значения*`)` — конвертирует аргументы в строки, которые соединяет в результирующую строку;
  - `fn:string-join(`*строки*`, `*строка-разделитель*`)` — соединяет строки со строкой-разделителем;
  - `fn:substring(`*строка*`, `*позиция*`, `*длина*`)` — возвращает подстроку строки, начиная с указанной позиции и указанной длины;
  - `fn:string-length(`*строка*`)` — возвращает длину строки;
  - `fn:normalize-space(`*строка*`)` — удаляет пробелы в начале и в конце строки, заменяет последовательности символов пробелов на пробел;
  - `fn:normalize-unicode(`*строка*`, `*форма нормализации Unicode*`)` — нормализует строку согласно указанному алгоритму нормализации (`NFC`, `NFD`, `NFKC`, `NFKD`, `FULLY-NORMALIZED`);
  - `fn:upper-case(`*строка*`)` — переводит символы строки в верхний регистр;
  - `fn:lower-case(`*строка*`)` — переводит символы строки в нижний регистр;
  - `fn:translate(`*строка*`, `*заменяемые символы*`, `*заменяющие символы*`)` — заменяет символы в строке;
  - `fn:encode-for-uri(`*URI-фрагмент*`)` — нормализует URI-фрагменты;
  - `fn:iri-to-uri(`*IRI*`)` — конвертирует IRI в URI;
  - `fn:escape-html-uri(`*строка с URI*`)` — нормализует URI-адрес в строке;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `fn:contains(`*`строка1, строка2, условие сопоставления`*`)` — возвращает `true`, если первая строка содержит вторую с учетом условия сопоставления;
- `fn:starts-with(`*`строка1, строка2, условие сопоставления`*`)` — возвращает `true`, если первая строка начинается второй с учетом сопоставления;
- `fn:ends-with(`*`строка1, строка2, условие сопоставления`*`)` — возвращает `true`, если первая строка заканчивается второй с учетом сопоставления;
- `fn:substring-before(`*`строка1, строка2, условие сопоставления`*`)` — возвращает подстроку первой строки, находящуюся перед второй строкой, которая является подстрокой первой строки, с учетом сопоставления;
- `fn:substring-after(`*`строка1, строка2, условие сопоставления`*`)` — возвращает подстроку первой строки, находящуюся после второй строки, которая является подстрокой первой строки, с учетом сопоставления;
- `fn:matches(`*`строка, регулярное выражение, способ сопоставления`*`)` — возвращает `true`, если строка согласуется с регулярным выражением в соответствии со способом сопоставления (`i`, `m`, `s`, `x`);
- `fn:replace(`*`строка, регулярное выражение, замещающая строка, способ сопоставления`*`)` — заменяет в строке подстроку, соответствующую регулярному выражению с учетом способа сопоставления, замещающей строкой;
- `fn:tokenize(`*`строка, регулярное выражение, способ сопоставления`*`)` — разделяет строку на последовательность подстрок согласно разделителю, соответствующему регулярному выражению.

❒ Функция обработки идентификаторов `fn:resolve-uri(`*`относительный URI, базовый URI`*`)` конвертирует относительный URI-адрес в абсолютный URI-адрес.

❒ Логические функции:

- `fn:true()` — возвращает `true`;
- `fn:false()` — возвращает `false`;
- `fn:not(`*`объект`*`)` — возвращает `true`, если объект — `false`, и наоборот.

❒ Функции обработки дат:

- `fn:years-from-duration(`*`период`*`)` — возвращает год;
- `fn:months-from-duration(`*`период`*`)` — возвращает месяц;
- `fn:days-from-duration(`*`период`*`)` — возвращает день;
- `fn:hours-from-duration(`*`период`*`)` — возвращает час;
- `fn:minutes-from-duration(`*`период`*`)` — возвращает минуты;
- `fn:seconds-from-duration(`*`период`*`)` — возвращает секунды;
- `fn:year-from-dateTime(`*`дата+время`*`)` — возвращает год;
- `fn:month-from-dateTime(`*`дата+время`*`)` — возвращает месяц;
- `fn:day-from-dateTime(`*`дата+время`*`)` — возвращает день;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `fn:hours-from-dateTime(`*дата+время*`)` — возвращает час;
- `fn:minutes-from-dateTime(`*дата+время*`)` — возвращает минуты;
- `fn:seconds-from-dateTime(`*дата+время*`)` — возвращает секунды;
- `fn:timezone-from-dateTime(`*дата+время*`)` — возвращает временную зону;
- `fn:year-from-date(`*дата*`)` — возвращает год;
- `fn:month-from-date(`*дата*`)` — возвращает месяц;
- `fn:day-from-date(`*дата*`)` — возвращает день;
- `fn:timezone-from-date(`*дата*`)` — возвращает временную зону;
- `fn:hours-from-time(`*время*`)` — возвращает час;
- `fn:minutes-from-time(`*время*`)` — возвращает минуты;
- `fn:seconds-from-time(`*время*`)` — возвращает секунды;
- `fn:timezone-from-time(`*время*`)` — возвращает временную зону;
- `fn:adjust-dateTime-to-timezone(`*дата+время*`, `*временная зона*`)` — модифицирует *дату+время* согласно временной зоне;
- `fn:adjust-date-to-timezone(`*дата*`, `*временная зона*`)` — модифицирует дату согласно временной зоне;
- `fn:adjust-time-to-timezone(`*время*`, `*временная зона*`)` — модифицирует время согласно временной зоне.

❒ Функции обработки имени:

- `fn:resolve-QName(`*QName*`, `*узел элемента*`)` — возвращает расширенное имя QName с учетом пространства имен узла элемента;
- `fn:QName(`*URI*`, `*QName*`)` — возвращает расширенное имя QName;
- `fn:prefix-from-QName(`*расширенное QName*`)` — возвращает префикс;
- `fn:local-name-from-QName(`*расширенное QName*`)` — возвращает локальное имя;
- `fn:namespace-uri-from-QName(`*расширенное QName*`)` — возвращает пространство имен;
- `fn:namespace-uri-for-prefix(`*префикс*`, `*узел элемента*`)` — возвращает пространство имен, соответствующее префиксу узла элемента;
- `fn:in-scope-prefixes(`*узел элемента*`)` — возвращает префикс узла элемента.

❒ Функции обработки узлов:

- `fn:name(`*узел*`)` — возвращает имя узла;
- `fn:local-name(`*узел*`)` — возвращает локальное имя узла;
- `fn:namespace-uri(`*узел*`)` — возвращает пространство имен узла;
- `fn:number(xs:anyAtomicType?)` — конвертирует аргумент в значение типа `xs:double`;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `fn:lang(`*язык*`, `*узел*`)` — возвращает `true`, если значение атрибута `xml:lang` соответствует аргументу;
- `fn:root(`*узел*`)` — возвращает корневой узел.

❒ Функции обработки последовательностей:

- `fn:boolean(`*последовательность*`)` — вычисляет `true`/`false` для последовательности;
- `fn:index-of(`*последовательность атомарных значений*`, `*атомарное значение*`, `*условие сопоставления*`)` — возвращает позиции вхождения атомарного значения в последовательность с учетом условия сопоставления;
- `fn:empty(`*последовательность*`)` — возвращает `true`, если аргумент — пустая последовательность;
- `fn:exists(`*последовательность*`)` — возвращает `false`, если аргумент — пустая последовательность;
- `fn:distinct-values(`*последовательность атомарных значений*`, `*условие сопоставления*`)` — устраняет дубликаты из последовательности;
- `fn:insert-before(`*последовательность1*`, `*позиция*`, `*последовательность2*`)` — вставляет `последовательность2` в `последовательность1`, начиная с указанной позиции;
- `fn:remove(`*последовательность*`, `*позиция*`)` — удаляет из последовательности объект, находящийся на указанной позиции;
- `fn:reverse(`*последовательность*`)` — меняет на противоположный порядок объектов в последовательности;
- `fn:subsequence(`*последовательность*`, `*позиция*`, `*длина*`)` — возвращает часть последовательности, начиная с указанной позиции и имеющей указанную длину;
- `fn:unordered(`*последовательность*`)` — меняет порядок объектов в последовательности. Изменение зависит от реализации XPath-процессора;
- `fn:zero-or-one(`*последовательность*`)` — возвращает последовательность, если она содержит не более одного объекта, иначе — возникает ошибка;
- `fn:one-or-more(`*последовательность*`)` — возвращает последовательность, если она содержит более одного объекта, иначе — возникает ошибка;
- `fn:exactly-one(`*последовательность*`)` — возвращает последовательность, если она содержит один объект, иначе — возникает ошибка;
- `fn:deep-equal(`*последовательность1*`, `*последовательность2*`, `*условие сопоставления*`)` — возвращает `true`, если объекты двух последовательностей равны друг другу;
- `fn:count(`*последовательность*`)` — возвращает количество объектов в последовательности;
- `fn:avg(`*последовательность атомарных значений*`)` — возвращает среднее последовательности значений;

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

- `fn:max(`*`последовательность атомарных значений, условие сопоставления`*`)` — возвращает максимальное значение в последовательности;
- `fn:min(`*`последовательность атомарных значений, условие сопоставления`*`)` — возвращает минимальное значение в последовательности;
- `fn:sum(`*`последовательность атомарных значений`*`)` — возвращает сумму значений последовательности;
- `fn:id(`*`ID, идентификатор документа`*`)` — возвращает последовательность узлов документа, имеющих указанное значение атрибута ID;
- `fn:idref(`*`IDREF, идентификатор документа`*`)` — возвращает последовательность узлов документа, имеющих указанное значение атрибута `IDREF`;
- `fn:doc(`*`URI-адрес`*`)` — находит документ по указанному URI-адресу и возвращает узел документа;
- `fn:doc-available(`*`URI-адрес`*`)` — возвращает `true`, если документ доступен по указанному URI-адресу;
- `fn:collection(`*`URI-адрес`*`)` — возвращает последовательность узлов, идентифицируемых указанным URI-адресом.

❒ Функции контекста:

- `fn:position()` — возвращает позицию контекста при обработке последовательности;
- `fn:last()` — возвращает количество объектов обрабатываемой последовательности — размер контекста;
- `fn:current-dateTime()` — возвращает системные дату и время;
- `fn:current-date()` — возвращает системную дату;
- `fn:current-time()` — возвращает системное время;
- `fn:implicit-timezone()` — возвращает системную временную зону;
- `fn:default-collation()` — возвращает условие сопоставления по умолчанию;
- `fn:static-base-uri()` — возвращает базовый URI-адрес документа.

Язык XSLT

Язык XSLT (Extensible Stylesheet Language Transformations) — это основанный на XML язык преобразования одного XML-документа во вновь созданный другой XML-документ.

Суть XSLT-преобразования состоит в применении XSLT-процессором таблицы XSLT-стилей к XML-документу, в результате чего генерируется новый XML-документ, HTML-документ или простой текст. Поэтому XSLT-синтаксис определяет структуру XML-документа, называемого таблицей XSLT-стилей, при этом при определении правил выборки данных используется язык XPath.

XSLT-документ использует пространство имен `http://www.w3.org/1999/XSL/Transform` и содержит элементы, описанные в табл. 1 и 2.

Таблица 1. *Элементы верхнего уровня XSLT-синтаксиса*

Элемент верхнего уровня	Атрибуты	Описание
`stylesheet` (его синоним — `transform`)	• `id` — идентификатор документа; • `extension-element-prefixes` — пространство имен элементов XSLT-расширения; • `exclude-result-prefixes` — пространство имен, которое необходимо исключить из результирующего документа; • `version` — обязательный атрибут, указывающий версию XSLT	Корневой элемент таблицы XSLT-стилей
`import`	`href` — обязательный атрибут, указывающий URI-адрес импортируемого XSLT-документа	Импортирует другой XSLT-документ, при этом исходный XSLT-документ имеет приоритет над импортируемым XSLT-документом
`include`	`href` — обязательный атрибут, указывающий URI-адрес включаемого XSLT-документа	Включает другой XSLT-документ
`strip-space`	`elements` — обязательный атрибут, содержит перечень элементов исходного XML-документа, в которых должны быть удалены пробелы	Определяет удаление пробелов в XML-документе
`preserve-space`	`elements` — обязательный атрибут, содержит перечень элементов исходного XML-документа, в которых должны быть сохранены пробелы	Определяет сохранение пробелов в XML-документе

Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Таблица 1 *(продолжение)*

Элемент верхнего уровня	Атрибуты	Описание
`output`	• `method` — указывает результирующий формат (`"xml" | "html" | "text"|` *имя*); • `version` — версия результирующего формата; • `encoding` — кодировка; • `omit-xml-declaration` — `"yes"` или `"no"`, если `"no"` (по умолчанию), тогда XML-объявления исходного документа включаются в результирующий документ; • `standalone` — `"yes"` или `"no"`, если `"no"` (по умолчанию), тогда объявление автономности не включается в результирующий документ; • `doctype-public` — устанавливает значение атрибута `PUBLIC` декларации `DOCTYPE` в результирующем документе; • `doctype-system` — устанавливает значение атрибута `SYSTEM` декларации `DOCTYPE` в результирующем документе; • `cdata-section-elements` — перечень элементов, чье содержимое должно быть выведено с помощью разделов `CDATA`; • `indent` — `"yes"` или `"no"`, определяет форматирование результирующего документа согласно его иерархической структуре; • `media-type` — MIME-тип результирующего документа	Определяет формат результирующего XML-документа
`key`	• `name` — обязательный атрибут, указывает имя ключа; • `match` — обязательный атрибут, указывает элементы XML-документа, к которым данный ключ применяется; • `use` — обязательный атрибут, содержит выражение значения ключа	Определяет ключ "имя — значение", используемый функцией `key()` в XPath-выражениях для эффективного поиска элементов XML-документа

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Таблица 1 *(продолжение)*

Элемент верхнего уровня	Атрибуты	Описание
decimal-format	• name — имя формата; • decimal-separator — символ разделения десятичной части числа от целой, по умолчанию "."; • grouping-separator — символ группировки тысяч, по умолчанию ","; • infinity — бесконечность, по умолчанию "Infinity"; • minus-sign — символ отрицательности числа, по умолчанию "-"; • NaN — неопределенное число, по умолчанию "NaN"; • percent — символ процентов, по умолчанию "%"; • per-mille — символ промилле, по умолчанию "‰"; • zero-digit — ноль, по умолчанию "0"; • digit — символ цифры, по умолчанию #; • pattern-separator — символ разделения положительных и отрицательных подшаблонов в шаблоне, по умолчанию ";"	Определяет символы, используемые при конвертации чисел в строку функцией format-number(). Указывает способ группировки цифр в числе, символ разделения десятичной части числа от целой и т. д.
namespace-alias	• stylesheet-prefix — исходный префикс; • result-prefix — результирующий префикс	Служит для замены префикса пространства имен
attribute-set	• name — обязательный атрибут, указывает имя набора; • use-attribute-sets — перечень имен атрибутов	Создает именованный набор атрибутов, вставляемых в результирующий XML-документ. Служит контейнером для элементов attribute (обязательный атрибут name — имя вставляемого атрибута, namespace — его пространство имен). Элементы attribute содержат значения вставляемых атрибутов
variable	• name — обязательный атрибут, указывает имя переменной; • select — содержит выражение, определяющее значение переменной	Объявляет переменные преобразования

Машнин Т. С. Web-сервисы Java. – СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

***Таблица 1** (окончание)*

Элемент верхнего уровня	Атрибуты	Описание
`param`	• `name` — обязательный атрибут, указывает имя параметра; • `select` — содержит выражение, определяющее значение параметра	Объявляет параметры преобразования
`template`	• `match` — имя элемента исходного XML-документа, к которому применяется шаблон; • `name` — имя шаблона; • `priority` — номер приоритета шаблона; • `mode` — режим преобразования	Определяет шаблон преобразования

***Таблица 2.** Элементы нижнего уровня XSLT-синтаксиса*

Элемент нижнего уровня	Атрибуты	Описание
`apply-imports`		Указывает, что необходимо применить шаблон из импортируемой таблицы XSLT-стилей
`apply-templates`	• `select` — указывает дочерний элемент текущего элемента, к которому применяется шаблон; • `mode` — режим преобразования	Указывает применение шаблона к элементу исходного XML-документа
`call-template`	`name` — имя вызываемого шаблона	Вызывает именованный шаблон
`choose`		Выбор условия, вложенные элементы `when` (обязательный атрибут `test` — условие) и `otherwise`
`comment`		Создает комментарии в результирующем XML-документе
`copy`	`use-attribute-sets` — перечень атрибутов элемента	В результирующем XML-документе создает копию текущего элемента исходного XML-документа без дочерних элементов и неуказанных атрибутов
`copy-of`	`select` — обязательный атрибут, указывает имя копируемого элемента	Создает копию, включая дочерние элементы и атрибуты

Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Таблица 2 *(продолжение)*

Элемент нижнего уровня	Атрибуты	Описание
element	• name — имя создаваемого элемента; • namespace — его пространство имен; • use-attribute-sets — перечень атрибутов элемента	В результирующем XML-документе создает элемент
fallback		Определяет альтернативную обработку элементов, которые не распознаются XSLT-процессором
for-each	select — обязательный атрибут, указывает имя элемента исходного XML-документа	Создает цикл обработки для указанного элемента исходного XML-документа
if	test — обязательный атрибут, содержит условие преобразования	Определяет условие преобразования
message	terminate — "yes" или "no", указывает необходимость прерывания преобразования	Выдает сообщение в процессе преобразования
number	• level — "single" или "multiple", или "any", указывает уровень исходного XML-документа; • count — указывает элементы уровня, которые необходимо учитывать; • from — указывает точку отсчета; • value — содержит выражение, которое будет преобразовано в число; • format — формат вывода чисел; • lang — код языка для чисел; • letter-value — "alphabetic" или "traditional", указывает нумерацию; • grouping-separator — группировка цифр; • grouping-size — размер группировки	Вставляет форматированное число в результирующий XML-документ
processing-instruction	name — обязательный атрибут, указывает имя инструкции по обработке	Создает элемент инструкции по обработке в исходящем XML-документе

Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

***Таблица 2** (окончание)*

Элемент нижнего уровня	Атрибуты	Описание
`sort`	• `select` — указывает элементы сортировки; • `lang` — код языка, используемый для сортировки; • `data-type` — `"text"` или `"number"`, или `"qname"`, указывает тип данных сортировки; • `order` — `"ascending"` или `"descending"`, указывает порядок сортировки; • `case-order` — `"upper-first"` или `"lower-first"`, указывает регистр сортировки	Указывает критерии сортировки для элементов `for-each` и `apply-templates`
`text`	`disable-output-escaping` — `"yes"` или `"no"`, указывает вывод специальных символов как есть	Создает текст в исходящем XML-документе
`value-of`	• `select` — обязательный атрибут, указывает имя элемента, значение которого извлекается; • `disable-output-escaping` — `"yes"` или `"no"`, указывает вывод специальных символов как есть	Извлекает значение элемента исходного XML-документа и вставляет его в виде текста в результирующий XML-документ
`with-param`	• `name` — имя параметра; • `select` — значение параметра	Передает параметр шаблону

В дополнение к XPath-функциям для работы с данными язык XSLT содержит ряд функций, описанных в табл. 3.

***Таблица 3.** XSLT-функции*

Функция	Параметры	Описание
`current()`		Возвращает набор узлов, содержащий данный узел
`document()`	• `object` (обязательный) — URI-адрес внешнего документа; • `node-set` — набор узлов, используемый для разрешения относительного URI-адреса внешнего документа	Обеспечивает доступ к внешним XML-документам

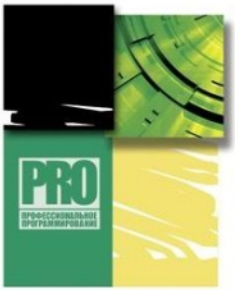
Машнин Т. С. Web-сервисы Java. — СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Таблица 3 *(окончание)*

Функция	Параметры	Описание
`element-available()`	`string` (обязательный) — тестируемый элемент	Указывает поддержку элемента XSLT-процессором
`format-number()`	• `number` (обязательный) — число; • `Format` (обязательный) — шаблон формата конвертации; • `decimalformat` — имя элемента `decimalformat`	Конвертирует число в строку
`function-available()`	`string` (обязательный) — тестируемая функция	Указывает поддержку функции XSLT-процессором
`generate-id()`	`node-set` — набор узлов	Генерирует идентификатор для узла
`key()`	• `string` (обязательный) — имя элемента `key`; • `object` (обязательный) — значение поиска	Возвращает набор узлов документа, используя элемент `key`
`system-property()`	`string` (обязательный) — имя свойства	Возвращает значение системного свойства
`unparsed-entity-uri()`	`string` (обязательный) — имя неразбираемой сущности	Возвращает URI-адрес не XML-ресурса

Машнин Т. С. Web-сервисы Java. –
СПб.: БХВ-Петербург, ISBN 978-5-9775-0778-3

Web-сервисы Java



Машнин Тимур Сергеевич, инженер-программист с многолетним опытом разработки программных комплексов и внедрения информационных систем. Автор книг «Современные Java-технологии на практике» и др.

В книге последовательно описана технология Web-сервисов Java, начиная с основ и заканчивая практической реализацией в Java-стеках.

Рассмотрены:

- спецификации первого уровня: XML, SOAP и WSDL;
- спецификации второго уровня: WS-* расширения первого уровня WS-Policy и WS-PolicyAttachment, WS-Security и WS-SecurityPolicy, WS-MetadataExchange, WS-ReliableMessaging и WS-ReliableMessaging Policy, WS-MakeConnection, WS-AtomicTransaction, WS-Coordination, WS-Trust и др.;
- Java-стандарты: JAXM, SAAJ, JAXP, JAXB, JAX-RPC, JAX-WS, JAX-RS;
- Java-стеки Web-сервисов: Metro, CXF и Axis2.

Материал книги сопровождается более 70 примерами с подробным анализом исходных кодов.

КАТЕГОРИЯ:
Java

Примеры проектов из книги и дополнительные материалы можно скачать по ссылке ftp://85.249.45.166/9785977507783.zip, а также на странице книги на сайте www.bhv.ru

ISBN 978-5-9775-0778-3

БХВ-ПЕТЕРБУРГ, 190005,
Санкт-Петербург, Измайловский пр., 29
E-mail: mail@bhv.ru Internet: www.bhv.ru
Тел.: (812) 251-42-44, тел./факс: (812) 320-01-79