Игорь Шапошников

bhv

Web-сервисы Microsoft .NET

- XML Web-сервисы
- Visual Studio .NET
- Протокол SOAP и язык WSDL
- Взаимодействие с базами данных
- Создание и использование Web-сервисов

МАСТЕР

Игорь Шапошников

Web-сервисы Microsoft .NET

Санкт-Петербург
«БХВ-Петербург»
2002

УДК 681.3.06
ББК 32.973
Ш24

Шапошников И. В.

Web-сервисы Microsoft .NET. — СПб.: БХВ-Петербург, 2002. — 336 с.: ил.

ISBN 5-94157-199-2

Книга посвящена одной из самых интересных частей новой технологии Microsoft .NET — разработке XML Web-сервисов. Рассматриваются основные приемы создания Web-сервисов, вопросы интеграции их с серверами баз данных на основе технологии ADO.NET, процедуры создания распределенных приложений на базе Web-сервисов, а также применение XML Web-сервисов на практике. Приводятся сведения о языках WSDL и SOAP, с помощью которых осуществляется разработка сервисов и клиентских приложений. Книга насыщена большим количеством авторских примеров действующих Web-сервисов с подробными комментариями.

Для программистов, разработчиков и администраторов Web-сайтов

УДК 681.3.06
ББК 32.973

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Елена Дудко*
Корректор	*Сергей Минин*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 17.07.02.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 27,09.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9-я линия, 12.

ISBN 5-94157-199-2

Содержание

Введение **1**

Глава 1. Основы Web-сервисов **3**

Microsoft .NET 3
Web-сервисы 5
Структура сервиса 7
Технические требования 8

Глава 2. Создание простейшего сервиса **12**

Сервис 12
Клиентская часть 17

Глава 3. Язык WSDL **21**

Краткий обзор 21
Структура элемента <*types*> 26
Типы данных WSDL 28
Структура элемента <*message*> 33
Структура элемента <*portType*> 35
Структура элемента <*binding*> 37
Структура элемента <*port*> 41
Структура элемента <*service*> 41

Глава 4. Язык SOAP **43**

Основы 43
Структура SOAP 45
Заголовки сообщений SOAP 47
Тело SOAP-сообщения 49
Типы данных SOAP 52

Глава 5. SOAP-клиенты **59**

Первый клиент 59
Клиент на основе ASP.NET 66
Прокси-классы 72

Глава 6. Работа с базами данных 76

ADO.NET 76
Постановка задачи 77
Сервис для просмотра данных 83
Использование SQL-запроса 100
Добавление данных 118
Изменение и удаление записей 133
Прямое редактирование набора данных 147
XML-документ в качестве источника данных 162

Глава 7. Основные приемы разработки Web-сервисов 170

Сохранение состояний 170
Идентификация пользователей 177
Конфигурационный файл Web-сервиса 186

Глава 8. B2B-решение 191

Постановка задачи 191
Структура базы данных 193
Структура сервиса 195
Разработка сервиса 196
Разработка сайта 223
Создание клиентского приложения 245

Глава 9. Поддержка пиринговых сетей 273

Структура пиринговых сетей 273
Обслуживающий Web-сервис 274
Пример клиентского приложения 287

Глава 10. Распределенные приложения 317

Новая модель 317
Структура распределенного приложения 319
Разработка ядра приложения 320

Послесловие 331

Даниленко Ольге

"Сентябрьских яблок
Тонкая кислинка
И губ твоих"

Введение

Эта книга посвящена одной из самых интересных частей новой технологии Microsoft .NET. Как вы уже поняли из наименования книги, речь пойдет о Web-сервисах, создаваемых при помощи этой технологии. Сама технология Microsoft .NET является несомненным новшеством, а Web-сервисы можно назвать ее изюминкой, так называемой режущей гранью (cutting edge). Более подробно о них будет рассказано в первой главе, и потом на протяжении всей книги мы будем узнавать о них все больше и больше, но уже сейчас их можно охарактеризовать вкратце. Сервисы — это приложения, которые функционируют на серверах с установленной поддержкой Microsoft .NET и осуществляют коммуникацию с клиентскими приложениями при помощи языка XML и протокола HTTP. За счет того, что используются платформенно-независимые язык XML и протокол HTTP, достигается возможность создавать клиентские приложения на любой компьютерной платформе.

Естественно, область применения подобных Web-сервисов достаточно широка, и они отлично вписываются в парадигму Сети, согласно которой доступ к документам и сервисам может осуществляться с любой компьютерной платформы. Но в отличие от статических HTML-документов, Web-сервисы позволяют гораздо эффективнее оперировать информацией и предоставляют пользователю достаточно высокую функциональность. Именно такая смесь межплатформенности (только в части клиентских приложений, естественно) и высокой функциональности позволяет создавать приложения, действующие через Сеть. Web-сервисы на базе Microsoft .NET позволяют разбивать приложение на серверную и клиентскую части, переходя к новой парадигме распространения программного обеспечения, когда пользователю передается не само приложение, а всего лишь доступ к нему.

Но перейдем к рассмотрению структуры книги. *Первая глава* посвящена постановке задачи. Мы узнаем, что же представляет собой концепция Microsoft .NET, и что такое Web-сервисы. Во *второй главе* мы рассмотрим создание простейшего Web-сервиса и средства доступа к нему с помощью двух

достаточно простых клиентских приложений. В *третьей главе* рассматривается язык WDSL, при помощи которого описывается конфигурация Web-сервисов. Это очень важный аспект разработки Web-сервисов, так как только на основе описания конфигурации сервиса клиентское приложение сможет получить доступ к функциям "незнакомого" сервиса. *Четвертая глава* посвящена рассмотрению языка SOAP, при помощи которого сервисы и клиенты обмениваются данными. В *пятой главе* мы разберем механизм разработки клиентских приложений, которые используют язык SOAP. Из *шестой главы* мы узнаем, как Web-сервисы могут интегрироваться с серверами баз данных. А *седьмая глава* посвящена изучению некоторых приемов разработки Web-сервисов.

Следующие три главы демонстрируют возможности применения Web-сервисов для решения различных задач. Так, в *восьмой главе* показано, как Web-сервис может функционировать в роли основы решения класса B2B. В *девятой главе* Web-сервис будет использоваться для координирования работы пиринговой сети. А *десятая глава* предназначена для рассмотрения процедуры создания на основе Web-сервисов распределенных приложений.

Итак, мы уже знаем, про что эта книга. Мы уже знаем, какова ее структура. Самое время перейти к работе.

Глава 1

Основы Web-сервисов

Microsoft .NET

Что же такое Microsoft .NET? Это новая технология Microsoft, которая предоставляет разработчику возможность достаточно легкого решения стандартных проблем, преследующих его в Интернете. Если же говорить более точно, то Microsoft .NET — это присоединяемая среда периода выполнения (runtime), функционирующая в операционных системах Windows 2000 и Windows XP. Основой этой технологии является .NET Framework, та самая среда, в которой и выполняются программы .NET. Эта среда выполнения является некоей надстройкой над операционной системой и предоставляет разработчику некоторые дополнительные возможности и удобства. Среди наиболее известных дополнительных возможностей — автоматический сборщик "мусора" и упрощенное развертывание как самой среды, так и приложений. Плюс ко всему эта среда предоставляет новый усовершенствованный способ доступа к базам данных — ADO.NET.

К среде, естественно, прилагается SDK.NET (Software Development Kit, средство для разработки программного обеспечения, не включающее в себя интегрированную среду разработки), позволяющее разрабатывать приложения, функционирующие в .NET Framework. Впрочем, гораздо удобнее разрабатывать эти приложения в оболочке Visual Studio .NET. Тем более, что при работе в этом средстве разработки программист может использовать один из трех языков — Visual Basic .NET, Visual C++ .NET и Visual C# .NET.

Объединение трех различных языков программирования под одной оболочкой стало возможным благодаря еще нескольким концепциям Microsoft .NET. Все эти концепции опираются на понятие *управляемого кода* (managed code). Приложения, написанные при помощи этого управляемого кода, исполняются не сразу в операционной системе, а в среде CLR (Common Language Runtime). Эта среда, как мы уже отмечали, является некоторой надстройкой над операционной системой и предоставляет приложениям несколько большие возможности, нежели сама Windows.

Однако, как мы помним, чтобы программы выполнялись в среде CLR, необходимо, чтобы они были написаны на стандартном языке MSIL (Microsoft Intermediate Language). Любое средство разработки приложений в среде Microsoft .NET обязано компилировать приложения именно в формат MSIL, а не только в исполняемый код операционной системы. Именно поэтому в рамках Visual Studio .NET объединено три языка.

Но необходимо понимать, что MSIL-код не может выполняться операционной системой напрямую. Его необходимо еще перевести в машинный код. Эту операцию назвали компиляцией "по требованию" (just-in-time). Соответственно, компилятор, переводящий MSIL-код в машинный код для специфичной операционной системы, называется *джиттером* (just-in-time compiler, JITter). Основное достоинство этой концепции состоит в том, что подобные компиляторы-джиттеры можно сделать практически для любой платформы. Таким образом, создав код приложения один раз в среде Microsoft .NET, его при помощи джиттеров можно безболезненно портировать под все платформы, для которых разработаны джиттеры.

Вам ничего эта модель не напоминает? Да, конечно, то же самое декларировала компания Sun, но только применительно к Java. Однако, несмотря на все усилия апологетов Java, она не получила столь широкого распространения, на которое они рассчитывали. И теперь межплатформенность заявлена в Microsoft .NET. На самом деле, на данный момент джиттеры созданы лишь для операционных систем семейства Windows, да и то не для всех. Кстати, Microsoft пока даже и не объявила о планах создания джиттеров для систем, не входящих в семейство Windows. Интересно, почему?

Так как все программы в концепции Microsoft .NET из исходных кодов компилируются в MSIL-код, то теоретически все языки .NET должны предоставлять разработчику одинаковый набор системных функций. Таким образом, унифицируется и технология разработки программ. Действительно, доселе различные языки программирования имели разные возможности, например, для доступа к COM-объектам, которые являются действительно хорошим средством расширяемости и модульности приложений. В языках .NET набор системных функций одинаков, поэтому доступ к тем же COM-объектам будет производиться одинаково как из кода Visual Basic, так и из кода, написанного на C++.

Естественно, подобная функциональность CLR реализуется при помощи дополнительного пространства имен System. Соответственно, разработчик получает несколько более высокую структурированность используемых системных функций и избавление от конфликтов имен. Идеология пространств имен, получившая широкое распространение еще со времен распространения XML, действительно позволяет избавиться от конфликтов имен функций, что дает возможность назначать им достаточно короткие и осмысленные имена.

К остальным достоинствам технологии Microsoft .NET можно отнести прозрачное взаимодействие с объектами COM, улучшенное управление версиями, разграничение доступа к системным функциям для программ (а не только для пользователей), автоматическую сборку "мусора" в куче памяти, и многое-многое другое. Уже просто знакомясь со списком нововведений Microsoft .NET, хочется начать работать в этой системе. Ну, так и не отказывайте себе в удовольствии.

Web-сервисы

Web-сервисы являются одной из самых интересных возможностей, которые предоставляет среда Microsoft .NET. Если говорить точнее, то сама идея сервисов в Сети не нова, но никогда их создание не было таким простым, как с использованием средства разработки Visual Studio .NET. Однако вернемся к самим Web-сервисам.

Давайте вспомним основной принцип работы WWW, когда документы создаются в едином формате HTML, а клиентские приложения-браузеры, функционирующие на самых различных компьютерных платформах, почти одинаково отображают эти документы по заранее оговоренным правилам. Однако это были всего лишь статические документы. Для придания им динамики требовалось использование различных средств программирования, действующих как на стороне клиента, так и на стороне сервера.

Если разработчик применял скриптовые языки, функционирующие на стороне клиента, то область их действия была ограничена отображаемым документом, а в качестве входных данных использовались либо действия пользователя, либо информация, хранящаяся на его машине. Связывалось это ограничение с тем, что скриптовые языки не получали какой-либо дополнительной информации от WWW-сервера, с которого был загружен документ, вместе с которым и были получены скрипты.

Если на сайте требовалось использовать несколько более серьезные способы управления контентом, приходилось применять исполняемые модули, функционирующие на самом сервере. Только таким образом можно было осуществлять доступ к базам данных, и на основе полученной информации формировать страницы, передаваемые затем удаленному пользователю. Только на сервере удавалось обеспечить поддержку сеансов работы и идентификацию пользователей, столь необходимые во всех приложениях электронной коммерции.

Естественно, технологий Web-программирования было немало. Для этой цели использовались и классические языки, такие как C++, и технологии, специально ориентированные на Web, такие как ASP. Однако все они в своем разнообразии имели одну общую черту — конечный вывод документов производился в формате HTML.

Если учитывать общую скорость прогресса в компьютерной индустрии, то можно признать, что долгожитель HTML достаточно долго использовался, и обещанное вытеснение его XML произойдет еще не скоро. Однако нельзя не видеть, что XML все сильнее и сильнее проникает в мир WWW. И основное ограничение заключается в том, что стандартные браузеры не в состоянии адекватно отображать содержимое XML-документов.

Итак, проблема локализована. Было бы неплохо заменить HTML более функциональным языком XML, облегчить взаимодействие между пользователем и сервером. Однако одна из основных проблем — стандартные браузеры, которые отображают только HTML и не в состоянии пересылать на WWW-сервер что-либо, кроме стандартных HTTP-пакетов. Следовательно, для получения большей функциональности необходимо отойти от стандартных браузеров.

Естественно, этот шаг повлечет за собой определенные проблемы, так как для каждого ресурса будет нужно отдельное клиентское приложение, но следует признать, что специализированные клиенты потребуются только для тех немногих ресурсов, которым необходима какая-либо особая функциональность. Для обычных Web-сайтов с небольшой функциональностью стандартных браузеров более чем достаточно.

Система Web-сервисов и обособленных клиентов нужна только для ресурсов с чрезвычайно сложной функциональностью. Первые ласточки среди таких ресурсов уже появляются. Это серверы различных пиринговых сетей, таких, как, например, небезызвестный Napster, или серверы распределенных вычислений (знаменитая программа CETI@Home является отличным примером). Однако написание подобных проектов являлось до недавнего времени достаточно нетривиальной и трудоемкой задачей. Заметили? Мы сказали: "До недавнего времени".

Технология Microsoft .NET позволяет писать сервисы достаточно легко. По сравнению с тем, что было раньше — чрезвычайно легко. Конечно, написание клиентских приложений для этих Web-сервисов будет несколько более сложной задачей, по сравнению с разработкой самих сервисов, но тоже не такой уж и трудной. Вы увидите.

Сервисы .NET обладают весьма высокой функциональностью и теоретически позволяют разбивать обычные приложения на две части — клиентскую и серверную. Если взять, например Microsoft Excel, то клиентская часть может включать в себя только средства отображения информации, подобно некоему специализированному табличному браузеру, а вся логика приложения может находиться в серверной части.

Конечно, для функционирования подобных приложений пользователю потребуется иметь весьма "толстый" канал связи с сервером, но в условиях локальной сети эта проблема может быть легко решена. А использование модели "разделенных приложений" позволяет достаточно легко изменить

политику лицензирования программного обеспечения и начать новый виток борьбы с пиратами (впрочем, судя по всему, эта борьба в ближайшее время не закончится).

В общем случае, Web-сервисы принимают и передают информацию на языке XML. Этот язык, естественно, является открытым, а не проприетарным стандартом. Также следует учитывать, что XML, как и его предшественник HTML, не зависит от платформы и операционной системы. Сочетание этих двух факторов позволяет разработчикам свободно создавать самые различные клиентские приложения, функционирующие на разных компьютерных платформах. То есть, одному сервису может соответствовать несколько клиентских приложений.

XML-документы, пересылающиеся от сервера к клиенту и обратно, передаются по протоколу HTTP, который пропускается практически всеми брандмауэрами, что также прибавляет привлекательности идее Web-сервисов.

Следует отметить и тот факт, что Web-сервисы являются самодокументируемыми. То есть любое клиентское приложение может получить информацию о структуре сервиса, о его функциональности и правилах вызова функций, поддерживаемых Web-сервисом.

Web-сервисы способны передавать и получать информацию тремя способами: с применением методов GET и POST стандартного протокола HTTP или при помощи языка SOAP, который является производным от XML. Первые два варианта разрешают использовать в качестве клиентского приложения стандартный браузер, но необходимо отдавать себе отчет, что это далеко не идеальный вариант. Во-первых, при помощи браузера весьма трудно организовать вызов всех функций достаточно сложного сервиса. Разработчику необходимо исследовать структуру сервиса заранее, перед тем, как создавать Web-страницы для доступа к сервису. Так что в этом случае мы теряем преимущества самодокументируемости. Во-вторых, следует помнить, что вывод информации все равно будет идти в "чистом" XML, который браузер не сможет адекватно отобразить. Поэтому для работы с Web-сервисами, обладающими достаточно серьезной функциональностью, следует все же ориентироваться на язык SOAP. Его структуру мы разберем в *главе 4*.

Однако пришло время перейти от общих слов к детальному рассмотрению концепции и структуры Web-сервисов.

Структура сервиса

Рассмотрим сначала структуру взаимодействия Web-сервиса .NET с клиентским приложением. При первом обращении клиентского приложения, которое "не знает" структуры сервиса, тот передает клиенту информацию о поддерживаемых им функциях и параметрах, необходимых для вызова этих функций. Передача этой информации о структуре называется "контрактом".

Подобная информация извлекается из описания сервиса, которое для каждого сервиса создается на языке WSDL. Естественно, языку WSDL и структуре описания сервиса мы уделим особое внимание в *главе 3*.

После того как клиент получает этот "контракт", он анализирует полученную информацию и на ее основе формирует запросы к сервису, используя вызовы его функций. А затем идет просто обмен запросами к сервису и ответами на них, которые формируются на языке XML. Естественно, чтобы Web-сервисы могли получать и отправлять информацию, для чего, как мы знаем, используется протокол HTTP, на серверной машине должен быть установлен WWW-сервер IIS. Но об этом будет сказано в *разд. "Технические требования"* данной главы.

Теперь перейдем к рассмотрению внутренней структуры сервиса. Web-сервисы действуют на платформе Microsoft .NET, следовательно, они вполне могут использовать все возможности этой платформы, такие, как доступ к базам данных, отслеживание своего состояния и прочие системные функции.

Следует обратить внимание, что до появления платформы Microsoft .NET разработка подобных Web-сервисов тоже была возможна, но требовала куда больших затрат сил и времени разработчика. Средство разработки Visual Studio .NET позволяет создавать Web-сервисы чрезвычайно легко. Дело в том, что при создании сервисов в этом средстве разработки среда берет на себя управление модулем, который ответственен за связь с клиентским приложением, форматирование отправляемых данных по правилам XML, документирование сервиса и многие другие обязательные, но рутинные вещи. Таким образом, разработчик фокусируется на программировании именно логики и функциональности сервиса, а рутину передает среде разработки. Подобная концепция весьма напоминает ситуацию, когда появились первые визуальные средства разработки приложений RAD (Rapid Application Development), которые избавили программистов от разработки стандартных интерфейсов, предоставив им визуальные средства проектирования. Теперь то же самое произошло и при создании Web-сервисов, что позволит разрабатывать их намного быстрее. Однако перед тем, как мы перейдем к созданию своего первого сервиса, следует уточнить, какие системные требования предъявляются к той машине, на которой будет функционировать Web-сервис.

Технические требования

Естественно, технические требования к серверу, на котором будут развернуты Web-сервисы, вытекают из основных нужд этих сервисов. Итак, сервер, под которым мы понимаем совокупность аппаратных компонентов и программного обеспечения, должен обеспечивать выполнение следующего набора базовых функций:

- Возможность получать входящие запросы по протоколу HTTP.
- Возможность осуществлять аутентификацию и авторизацию удаленных пользователей.
- Изолировать сервисы друг от друга, чтобы каждый имел свое собственное адресное пространство в оперативной памяти, и ошибка в одном сервисе не повлекла бы за собой сбои в остальных сервисах, выполняющихся одновременно с проблемным сервисом.
- Предоставление администратору средств для развертывания сервисов, а также для контроля и наблюдения за ними.
- Возможность управлять ресурсами, которые выделяются каждому сервису.

Естественно, некоторые сервисы могут предъявлять повышенные и нестандартные требования к машине, на которой они установлены, но базовый перечень требований мы привели. Остальное администратор сервера может делать самостоятельно.

Для программирования сервиса неплохо также иметь средство разработки, которое позволяло бы программисту достаточно прозрачно работать с языком XML, так как иначе ему придется писать свой парсер.

Теперь, когда список требований определен, настало время узнать, какая платформа им удовлетворяет. Так как мы собираемся при разработке сервисов опираться на технологию Microsoft .NET, естественно, нас будут интересовать операционные системы семейства Windows. Поэтому отметим, что перечисленные требования начали поддерживаться в этих операционных системах, начиная с Windows 2000. Само собой, кроме нее установленным критериям удовлетворяют Windows XP и Windows .NET. При этом следует помнить, что во всех этих системах необходимо устанавливать и активировать WWW-сервер IIS, который позволяет принимать и отправлять информацию по протоколу HTTP.

Естественно, все перечисленные системы имеют API, который позволяет приложениям использовать следующие возможности:

- доступ к COM-объектам, которые могут предоставлять функциональность для доступа к базам данных и элементам бизнес-логики;
- ADO, OLE DB и ODBC для доступа к базам данных;
- MSXML для анализа, разбора и создания сообщений на языке XML;
- Возможности технологии ASP для приема запросов по протоколу HTTP.

Мы здесь не учитываем потенциал .NET Framework, которая предоставляет еще больше возможностей, но смысл уже понятен. Для разработки Web-сервисов нам придется использовать именно систему Windows 2000 или старше. Конечно, для более комфортной работы следует установить сре-

ду .NET, а для разработки было бы неплохо использовать Visual Studio .NET. Впрочем, можно обойтись без среды разработки Visual Studio .NET и пользоваться для написания кода стандартным Блокнотом, но честное слово, с ней работается настолько легче, что мы искренне советуем обзавестись этим средством RAD.

В качестве взаимодействующих с Web-сервисами средств перечислим следующие продукты:

- Application Center 2000 для развертывания приложений и управления ими;
- BizTalk Server 2000 для связи Web-сервисов с уже готовыми приложениями, обменивающимися информацией на языке SOAP;
- Commerce Server 2000 для создания сервисов, работающих в сфере электронной коммерции;
- Internet Security and Acceleration Server 2000, позволяющий обезопасить работу в Сети при помощи встроенного брандмауэра и управляющий кэшированием данных на сервере;
- Mobile Information Server 2001, позволяющий предоставлять доступ к данным с мобильных телефонов и других устройств, воспринимающих WML;
- SQL Server 2000 для связи сервисов с базами данных.

Совершенно необязательно устанавливать на сервер все эти продукты. Каждый администратор сам выберет необходимые компоненты из этого списка или добавит еще некоторые средства. Чаще всего для работы Web-сервисов ограничиваются SQL-сервером SQL Server 2000 и Commerce Server 2000.

Конечно, подобные средства разработки и операционные системы выставляют некоторые требования к аппаратному обеспечению, на котором они установлены. В том случае, если используется весь вышеперечисленный набор, необходим будет действительно мощный компьютер с быстрым дисковым массивом, объемом оперативной памяти не менее гигабайта и процессором уровня Pentium 4. Но это крайний случай. Обычно требуется установить операционную систему, .NET Framework и SQL Server 2000. Для этих целей вполне можно обойтись достаточно средней по сегодняшним меркам платформой. А именно: 256 Мбайт оперативной памяти, обычный качественный жесткий диск и процессор с частотой около 700 МГц для обработки запросов к базам данных вполне подойдут для сервера, на котором будут базироваться искомые Web-сервисы. Впрочем, все зависит не только от внутренних задач, но и от популярности сервиса, а значит, и от количества одновременных подключений пользователей.

Конечно же, не стоит даже особого упоминания, что к любой конфигурации сервера необходима хорошая сетевая плата известного производителя на 100 Мбит. Сама связь с Сетью ни в коем случае не должна быть "узким местом" в производительности сервера.

Теперь, после того как мы рассмотрели основные требования к системе, пришла пора рассмотреть пример создания самого первого и самого простого Web-сервиса. Этакий "Hello, World", но с учетом нашей специфики.

Глава 2

Создание простейшего сервиса

Сервис

Для начала попробуем создать самый простой Web-сервис, который по запросу пользователя будет выдавать текущую дату или комбинацию даты и времени. Для этого в среде разработки Visual Studio .NET следует выполнить команду меню **File | New | Project** и в появившемся диалоговом окне **New Project** в наборе **Templates** выделить значок **ASP.NET Web Service**. В поле **Name** следует указать наименование создаваемого проекта. Укажем для начала имя `MyService`. После того, как все необходимые приготовления средой разработки будут сделаны, на основном рабочем поле появятся две новые страницы. Одна будет предназначена для разработки визуального дизайна, а на второй будет располагаться код нашего сервиса. Естественно, у создаваемого сервиса не может быть внешнего вида как такового, ему нечего отображать, поэтому страницу для дизайна можно спокойно закрыть.

В состав одного проекта может входить несколько отдельных сервисов. Для нашего примера потребуется всего один Web-сервис, заготовка для которого создается средой разработки Visual Studio. NET автоматически, поэтому ничего изменять не потребуется, и мы можем сразу перейти на страницу с наименованием Service1.asmx.vb. На этой странице мы и разместим код нашего Web-сервиса. Он приведен в листинге 2.1.

Листинг 2.1

```
Imports System.Web.Services

Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
```

```
    Public Sub New()
        MyBase.New()
        InitializeComponent()
    End Sub

    Private components As System.ComponentModel.Container

    <System.Diagnostics.DebuggerStepThrough()> Private
Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    End Sub

#End Region

    <WebMethod()> Public Function MyDate(ByVal ShowTime As Boolean) As
String
        Dim MD As DateTime
        If ShowTime Then
            MyDate = MD.Now
        Else
            MyDate = MD.Today
        End If

    End Function

End Class
```

Из всего этого кода лишь малая часть написана разработчиком. Это функция `MyDate`, расположенная в конце листинга. Ее нам и следует рассмотреть.

Из листинга видно, что объявление функции предваряется тегом `<WebMethod()>`. Он и сигнализирует компилятору, что следующая за ним функция является частью создаваемого сервиса, и результаты ее работы будут передаваться клиенту. В объявлении функции мы указали, что она будет воспринимать параметр `ShowTime` логического типа `Boolean`. В том случае, если функции передано значение `True`, клиенту будет возвращено значение, в которое входит и текущая дата и время вызова функции. Для этого мы используем свойство `Now`, входящее в состав типа `DateTime`. Если же функция

в качестве параметра получает значение `False`, то клиентскому приложению возвращается только дата, что производится при помощи свойства `Today`.

Создание Web-сервиса не явилось сложной задачей. Теперь рассмотрим, как он действует. После компиляции к нему можно обратиться даже из браузера. На рис. 2.1 приведен внешний вид Web-страницы, которая генерируется Web-сервером при попытке обращения к Web-сервису.

Рис. 2.1. Внешний вид Web-страницы, соответствующей Web-сервису

На этой Web-странице расположена ссылка на формальное описание структуры Web-сервиса, а также перечислены все функции, поддерживаемые Web-сервисом. В нашем случае, естественно, указана лишь одна функция `Mydate`. Наименование каждой функции также является гиперссылкой, нажав на которую можно перейти к Web-странице, позволяющей воспользоваться этой функцией. Так мы и поступим. После щелчка мышью на ссылке, указывающей на функцию `MyDate`, в браузере будет отображена Web-страница, предоставляющая доступ к этой функции. Внешний вид этой Web-страницы показан на рис. 2.2.

На этой странице расположено поле ввода, в котором пользователь может указать значение параметра, передаваемого функции, и кнопка **Invoke**, при нажатии на которую этот параметр будет передан функции сервиса, и в браузер будет отправлен результат работы сервиса.

На самом деле, содержимое рассматриваемой Web-страницы не ограничивается органами управления, показанными на рисунке. На ней также показаны блоки кода, которые следует использовать клиентским приложениям для связи с сервисом. В силу того, что эти блоки кода занимают достаточно большое пространство, они не приведены на иллюстрации. Но после того как вы создадите Web-сервис и обратитесь к нему через браузер, можно будет увидеть эти инструкции. Следует отметить, что на Web-странице приве-

дены инструкции для создания клиентов на основе методов GET и POST, а также при помощи языка SOAP. Забота о разработчике налицо. Например, текст запроса к Web-сервису на языке SOAP выглядит следующим образом:

```
POST /MyService/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/MyDate"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <MyDate xmlns="http://tempuri.org/">
      <ShowTime>boolean</ShowTime>
    </MyDate>
  </soap:Body>
</soap:Envelope>
```

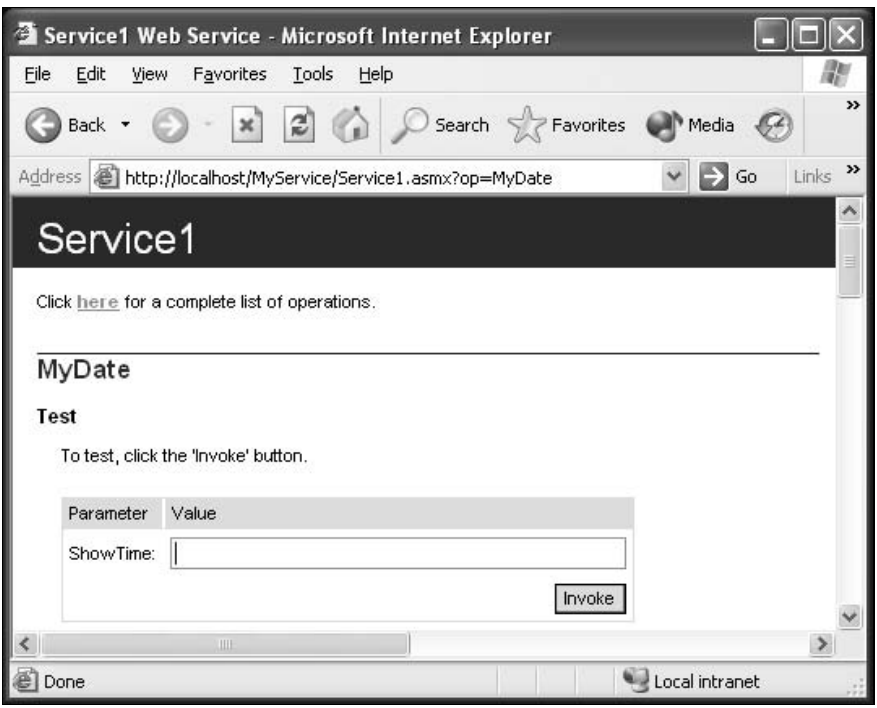

Рис. 2.2. Web-страница, предоставляющая пользователю доступ к функции MyDate

Но вернемся к тестированию созданного Web-сервиса. Как уже было сказано, в поле текстового ввода необходимо внести значение параметра `ShowTime`. Кстати, наименование параметра также указано на странице, рядом с соответствующим полем ввода. Тип параметра указан в блоках кода для клиентских приложений.

Итак, мы знаем, что наша функция воспринимает логический параметр типа `Boolean`. Поэтому в поле ввода можно ввести значение `True`, что мы и сделаем. После того как искомое значение будет введено в текстовое поле и нажата кнопка **Invoke**, браузер отобразит Web-страницу с результатом работы функции сервиса. Внешний вид этой страницы показан на рис. 2.3.

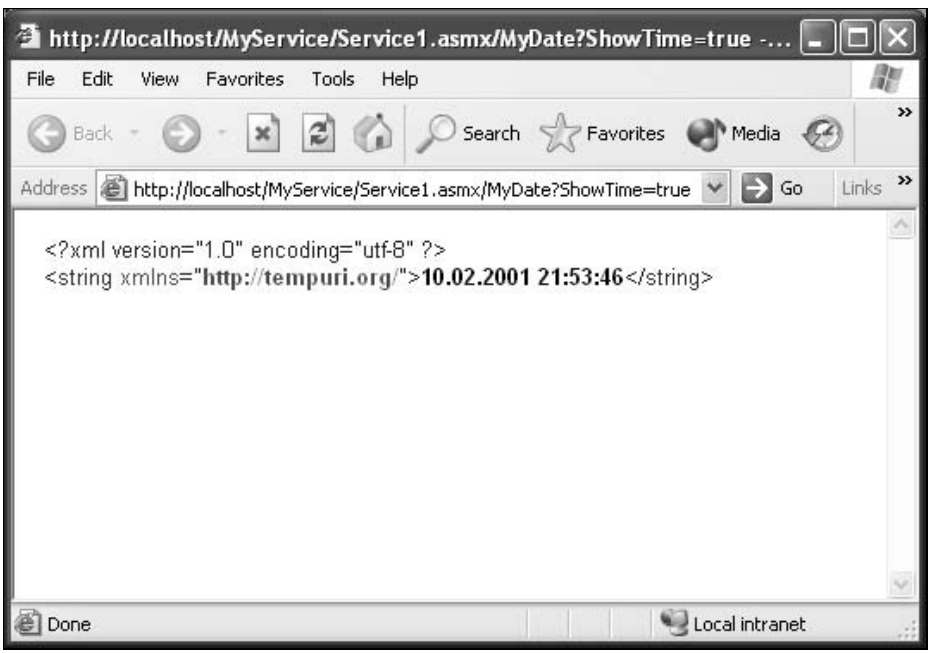

Рис. 2.3. Web-страница с результатом работы функции

Легко заметить, что на последней Web-странице отображен код XML-документа. Этот XML-документ и является результатом работы созданного нами Web-сервиса. Теперь клиентское приложение получит этот XML-документ, само обработает его и выдаст результат пользователю или использует этот результат для собственных целей. Таким образом, практически без всяких усилий с нашей стороны, был создан сервис, который сам документирует себя, принимает данные тремя различными способами по протоколу HTTP и возвращает результаты работы на языке XML. Это действительно удобно.

Теперь обратимся к вопросу самодокументированности Web-сервисов. Дело в том, что при создании Web-сервиса компилятор одновременно создает WSDL-файл, в котором описана структура сервиса. Именно на основе этого файла клиенты получают информацию о функциональности сервиса.

Структуру языка WSDL мы будем рассматривать в *главе 3*, поэтому листинг пока не приводится.

А теперь, когда мы знаем, как создавать сервисы, следует рассмотреть приемы создания клиентских приложений. Но об этом — в следующем разделе.

Клиентская часть

Как мы знаем, клиентские приложения могут использовать три варианта передачи запросов Web-сервису. Это методы `GET` и `POST` протокола HTTP и язык SOAP. Здесь мы рассмотрим создание клиентских приложений, действующих по первым двум вариантам.

Если обращаться к методам `GET` и `POST`, то следует признать, что нет нужды создавать отдельные приложения, которые бы формировали HTTP-запросы по правилам этих методов. У нас уже есть такое приложение — обычный браузер. Достаточно лишь создать для него соответствующие Web-страницы. Это будет намного быстрее и проще, нежели разрабатывать отдельные приложения, которые должны будут самостоятельно создавать HTTP-запросы, отправлять их на сервер, получать ответ на языке XML и извлекать из него данные. Поэтому начнем с создания обычных Web-страниц.

Первый созданный нами клиент для подключения к Web-сервису будет применять метод `GET`. Как известно, при использовании этого метода вся информация, пересылаемая на сервер, передается через строку запроса вместе с URL. То есть на клиентской Web-странице достаточно будет создать ссылку на сервис и в URL поместить необходимую для вызова функции информацию. Так как наш сервис воспринимает один параметр логического типа, нам потребуется создать на клиентской Web-странице две ссылки: одну со значением параметра `True`, а другую — с `False`. Код этой Web-страницы приведен в листинге 2.2.

Листинг 2.2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <title> GET-клиент</title>
  </head>
```

```
  <body>
  <p>Используйте гиперссылки для работы с web-сервисом</p>
<p><a
href="http://localhost/MyService/Service1.asmx/MyDate?ShowTime=True">
Получить дату и время</a></p>
<p><a
href="http://localhost/MyService/Service1.asmx/MyDate?ShowTime=True">
Получить только дату</a></p>
  </body>
</html>
```

Внешний вид этой Web-страницы при отображении ее в браузере показан на рис. 2.4.

Основное внимание в этом листинге следует уделить гиперссылкам, а точнее, значениям атрибута `href` тегов `<a>`. В URL после указания имени файла Service1.asmx указывается наименование функции `MyDate`. После наименования функции ставится вопросительный знак, отделяющий параметры от основного URL. А уже после символа вопросительного знака указывается наименование параметра и его значение. Именно таким образом передаются данные при помощи метода `GET`.

Так как параметр может принимать два значения, то мы и создаем две ссылки, у которых URL отличаются друг от друга только в последней части, где после знака равенства указывается значение параметра.

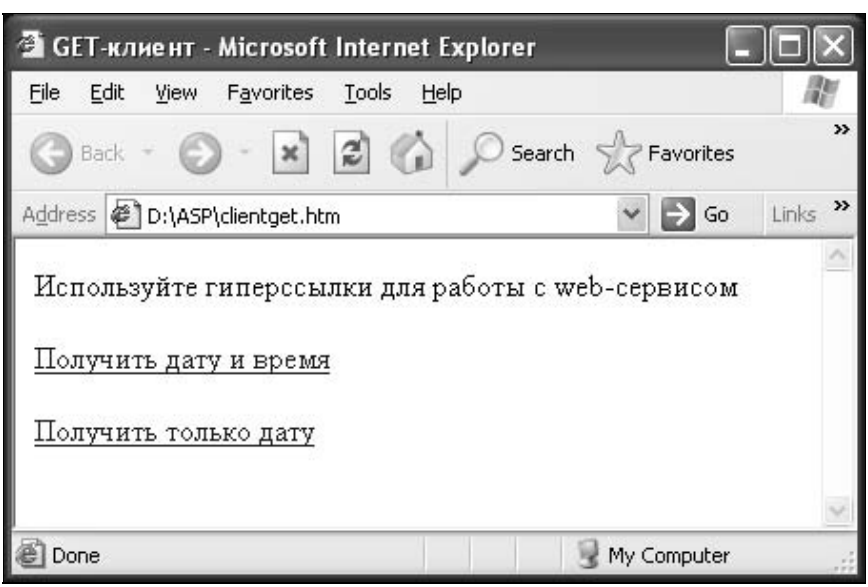

Рис. 2.4. Клиентская Web-страница, работающая с методом `GET`

Теперь перейдем к созданию клиентской Web-страницы, передающей запрос при помощи метода POST. Как известно, метод POST позволяет передавать данные, введенные пользователем в HTML-форме. Следовательно, нам придется на Web-странице создать форму.

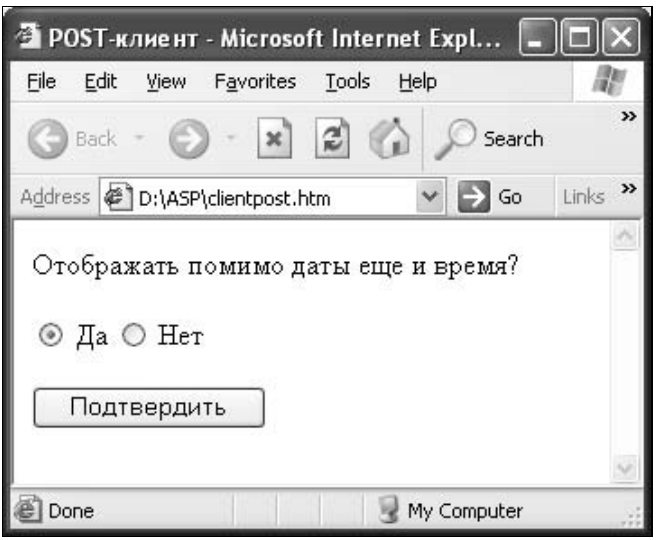

Рис. 2.5. Клиентская Web-страница, работающая с методом POST

Так как параметр имеет логический тип, будет правильно создать группу из двух переключателей. Передаваемое значение может быть или True или False, и именно эти значения будут возвращать переключатель. Однако помимо значения параметра, необходимо передать Web-сервису и его наименование. Как известно, при передаче данных от браузера на сервер, они указываются в виде последовательности пар "имя=значение", где перед знаком равенства указывается наименование органа ввода данных. Следовательно, для передачи правильной пары нам необходимо группу переключателей назвать ShowTime. В результате HTML-код клиентской Web-страницы, передающей информацию при помощи метода POST, будет выглядеть приблизительно так, как это показано в листинге 2.3.

Листинг 2.3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  <title> POST-клиент</title>
```

```
  </head>
  <body>
  <form METHOD="POST"
ACTION="http://localhost/MyService/Service1.asmx/MyDate">
  <p>Отображать помимо даты еще и время?</p>
    <p><input TYPE="RADIO" NAME="ShowTime" VALUE="true" CHECKED> Да
    <input TYPE="RADIO" NAME="ShowTime" VALUE="false"> Нет
    </p>
  <input TYPE="SUBMIT" VALUE="Подтвердить">
</form>
 </body>
</html>
```

Внешний вид этой Web-страницы показан на рис. 2.5.

Итак, мы рассмотрели создание клиентов на основе Web-страниц. Если пользоваться языком SOAP для коммуникации между клиентами и сервисом, придется создавать отдельное приложение, что уже выходит за пределы темы данной главы. Поэтому пока мы остановимся на этом.

Глава 3

Язык WSDL

Краткий обзор

Итак, мы знаем, что Web-сервисы должны документировать свою структуру, чтобы клиентские приложения могли обращаться к их функциям. Для этого и используется язык WSDL (Web Services Description Language). Этот язык создан на базе языка XML, знать который следует любому Web-разработчику, так как он предоставляет достаточно много новых возможностей, облегчающих создание в Сети больших и легко управляемых проектов.

На данный момент WWW-консорциумом (W3C) утверждена версия WSDL 1.1. Именно ей мы и займемся в этой главе. Естественно, мы не станем разбирать официальную спецификацию языка, это заняло бы слишком много времени. Достаточно лишь на примерах познакомиться с его синтаксисом.

В данном разделе мы лишь рассмотрим схему типичного WSDL-документа. Необходимо признать, что в своей логической форме любые XML-документы, к коим, несомненно, следует отнести и WSDL-документы, достаточно сильно отличаются от HTML-документов, так знакомых любому Web-разработчику. Если HTML-документы описывали лишь *отображение* некоего блока информации, то XML описывает прежде всего логическую структуру. Поэтому и WSDL-документы мы будем рассматривать с точки зрения их логической структуры.

Теперь вернемся к первому примеру Web-сервиса, который мы разбирали в предыдущей главе. Так как для разработки было использовано RAD-средство Visual Studio .NET, нам не придется заботиться о собственноручном написании WSDL-файла для этого сервиса. Он был создан автоматически.

Файл WSDL (Web Service Descriptor Language) является обычным XML-файлом, и доступ к нему мы можем получить даже из браузера. Если взглянуть на рис. 2.1, то в верхней части Web-страницы будет видна ссылка **Service Description**, которая отсылает пользователя как раз к упомянутому WSDL-файлу. Код WSDL-файла, соответствующего нашему Web-сервису, приведен в листинге 3.1.

Листинг 3.1

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/" targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="MyDate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="ShowTime"
type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="MyDateResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="MyDateResult"
nillable="true" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </types>
  <message name="MyDateSoapIn">
    <part name="parameters" element="s0:MyDate" />
  </message>
  <message name="MyDateSoapOut">
    <part name="parameters" element="s0:MyDateResponse" />
  </message>
  <message name="MyDateHttpGetIn">
    <part name="ShowTime" type="s:string" />
  </message>
```

```
  <message name="MyDateHttpGetOut">
    <part name="Body" element="s0:string" />
  </message>
  <message name="MyDateHttpPostIn">
    <part name="ShowTime" type="s:string" />
  </message>
  <message name="MyDateHttpPostOut">
    <part name="Body" element="s0:string" />
  </message>
  <portType name="Service1Soap">
    <operation name="MyDate">
      <input message="s0:MyDateSoapIn" />
      <output message="s0:MyDateSoapOut" />
    </operation>
  </portType>
  <portType name="Service1HttpGet">
    <operation name="MyDate">
      <input message="s0:MyDateHttpGetIn" />
      <output message="s0:MyDateHttpGetOut" />
    </operation>
  </portType>
  <portType name="Service1HttpPost">
    <operation name="MyDate">
      <input message="s0:MyDateHttpPostIn" />
      <output message="s0:MyDateHttpPostOut" />
    </operation>
  </portType>
  <binding name="Service1Soap" type="s0:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation name="MyDate">
      <soap:operation soapAction="http://tempuri.org/MyDate"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
```

```
    </operation>
  </binding>
  <binding name="Service1HttpGet" type="s0:Service1HttpGet">
    <http:binding verb="GET" />
    <operation name="MyDate">
      <http:operation location="/MyDate" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <binding name="Service1HttpPost" type="s0:Service1HttpPost">
    <http:binding verb="POST" />
    <operation name="MyDate">
      <http:operation location="/MyDate" />
      <input>
        <mime:content type="application/x-www-form-urlencoded" />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
  <service name="Service1">
    <port name="Service1Soap" binding="s0:Service1Soap">
      <soap:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
    <port name="Service1HttpGet" binding="s0:Service1HttpGet">
      <http:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
    <port name="Service1HttpPost" binding="s0:Service1HttpPost">
      <http:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
  </service>
</definitions>
```

Этот файл полностью описывает функциональность созданного Web-сервиса, и именно на него будут опираться клиентские приложения для установки связи с сервисом. Достаточно часто WSDL-файлы, описывающие структуру Web-сервисов, называют *контрактами*.

Теперь попробуем установить его структуру. В WSDL-файлах информация структурируется при помощи всего пяти основных элементов. Если их кратко перечислить, получится следующий список:

- `<types>`. Элемент содержит определения типов данных, используемых в данном Web-сервисе.
- `<message>`. Элемент содержит абстрактные представления функций Web-сервиса вместе с указанием типов входных и выходных данных.
- `<portType>`. Элемент содержит ссылки на искомые функции, описанные в элементе `<message>` с более подробной структурой их входных и выходных данных.
- `<binding>`. Элемент предназначен для связывания содержимого блока `<port>` с конкретными функциями Web-сервиса. В этих разделах определяются протокол и формат данных для операций, на которые указывает конкретный раздел `<port>`.
- `<port>`. Содержат адреса функций, описанных в элементах `<message>`. По сути дела, порты описывают точки вхождения в Web-сервис.
- `<service>`. Основной элемент, в котором перечисляются конкретные порты, указывающие на прямые URL доступа к функциям сервиса.

Таким образом, мы можем сказать, что первые три элемента — `<types>`, `<message>` и `<portType>` — представляют собой несколько более абстрагированный уровень описания структуры Web-сервиса, а последние два — `<binding>` и `<service>` — увязывают абстрактную логику с конкретными механизмами доступа к функциональности сервиса.

Помимо этих элементов, создающих в WSDL-документе целые разделы, есть и отдельный блок, который нельзя отнести ни к какому разделу. Он выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/" targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
```

Первая строка, естественно, мгновенно узнается каждым, кто знаком с языком XML. Это стандартное объявление XML-документа. Второй тег содер-

жит определения всех используемых в WSDL-документе пространств имен. Легко заметить, что определения большей части объявляемых пространств имен располагаются на ресурсе с доменным именем `schemas.xmlsoap.org`. Особое внимание здесь следует обратить на объявление пространства имен `s0`, так как именно оно будет чаще всего использоваться в нашем примере для объявления типов переменных.

О самих типах переменных мы будем говорить в следующем разделе, а пока что стоит отметить, что WSDL поддерживает все типы, определяемые спецификацией XML Schema, располагающейся по адресу **http://www.w3.org/2001/XMLSchema**. Чаще всего для указания типов данных стоит использовать именно это пространство имен. Его мы и объявили для идентификатора `s` в самом первом параметре тега `<definitions>`.

Если обратиться к листингу 3.1, то в нем с легкостью обнаружатся все вышеописанные структуры. Впрочем, невооруженным глазом видно, что содержимое этих элементов далеко не всегда можно признать интуитивно понятным и прозрачным. Поэтому требуется рассмотреть их тщательнее. Начнем мы с рассмотрения структуры элемента `<types>`.

Структура элемента *<types>*

Элемент `<types>` является основой всего WSDL-файла. Именно в этом разделе определяются все типы данных, которые будет использовать сервис. Если учесть, что хороший сервис должен обладать достаточно богатым набором функций и выполнять сложные операции, станет понятно, что описание всех его типов данных, среди которых, естественно, могут быть и достаточно объемные составные типы, может занять весомую долю всего WSDL-файла.

Рассмотрим подробнее этот блок, позаимствовав его из нашего примера. Вот как он выглядит:

```
  <types>
    <s:schema attributeFormDefault="qualified" elementFormDe-
fault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="MyDate">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="ShowTime"
type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="MyDateResponse">
```

```
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="MyDateResult"
nillable="true" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </types>
```

Легко заметить, что все содержимое блока, ограниченного тегами `<types>` и `</types>`, создано с использованием пространства имен `s`, которое было объявлено в теге `<definitions>`. Это пространство имен, специфицированное, как мы уже знаем, в XML Schema, как раз и позволяет нам использовать типы данных, определенные в этом подмножестве языка XML.

Затем при помощи тегов `<s:schema>` и `</s:schema>` определяется тип для всего сервиса. Это можно сравнить с определением основного класса для обычного приложения. Отдельные элементы этого типа объявляются при помощи тегов `<s:element>`. При помощи атрибутов name объявляются наименования этих элементов. Следует также обратить внимание на параметр `targetNamespace`, входящий в состав объявляющего тега `<s:schema>`, который задает целевое пространство имен, используемое в данном блоке по умолчанию.

Элемент с наименованием `MyDate`, как легко догадаться, объявляет саму основную функцию нашего Web-сервиса. Внутри раздела этого объявления указывается комплексный тип данных для вложенного элемента с наименованием `ShowTime`. В объявлении элемента `ShowTime` присутствует дополнительный атрибут `type`, которому приписано значение "`s:boolean`". Естественно, подобным способом объявляется параметр, передаваемый нашей функции.

Итак, на основе анализа объявления первого элемента, мы знаем, что Web-сервис обладает функцией `MyDate`, которой передается параметр логического типа `Boolean` с наименованием `ShowTime`.

Следующим в блоке `<types>` объявляется элемент с наименованием `MyDateResponse`. Подобным образом обозначается элемент, отвечающий за возвращаемое значение. То есть, первый элемент содержит описание входных данных, а второй элемент описывает выходные данные. Таким образом, анализ объявления элемента `MyDateResponse` показывает, что в нем располагается определение еще одного элемента с наименованием `MyDateResult` типа `String`.

Резюмируем. Блок входных данных содержит элементы, наименования которых опираются на стандартные наименования функций и параметров, указанных в исходном коде самого Web-сервиса. Наименования выходных данных создаются на основе имени функции из первого блока. То есть, для функции `MyDate` элемент, отвечающий за передачу результата ее работы, носит наименование `MyDateResponse`, а результат выполнения функции носит наименование `MyDateResult`.

Теперь следует перейти к рассмотрению различных типов данных, которые могут использоваться в WSDL-документах.

Типы данных WSDL

Все типы данных, входящие в WSDL, унаследованы из спецификации XML Schema. Однако следует обратить внимание на тот факт, что WWW консорциум, известный также как W3C, определил три версии этой спецификации (1999, 2000 и 2001 гг.). Но, учитывая относительную новизну языка WSDL, обычно опираются на наиболее свежую версию спецификации XML Schema, датированную 2001 годом.

В WSDL-описаниях разработчик может применять как простые типы, так и составные. Но составные типы, как известно, создаются из простых. Поэтому для начала рассмотрим наиболее часто употребляемые простые типы данных.

- `anyURI`. Тип обычно применяется для хранения сетевых идентификаторов URI, указывающих на расположение каких-либо ресурсов в Сети. URI являются развитием концепции всем известных стандартных URL и позволяют адресовать не только какие-либо конкретные файлы, но и их фрагменты. В языке Visual Basic .NET этот тип распознается как `String`.
- `boolean`. Это, конечно, логический тип, который полностью соответствует типу `Boolean` из Visual Basic .NET. Возможные значения — `true` и `false`.
- `byte`. Тип предназначен для хранения целых чисел. Как можно понять из его наименования, под хранение одного значения отводится один байт, поэтому значение может располагаться в промежутке от −128 до 127. В Visual Basic .NET соответствует типу `Integer`.
- `date`. Тип предназначен для хранения дат. В Visual Basic .NET конвертируется в тип `Date`.
- `DateTime`. Значения этого типа хранят не только дату, но и время. Так же, как и предыдущий, этот тип в Visual Basic .NET конвертируется в тип `Date`.

- `double`. Тип предназначен для хранения числовых данных. Для каждого значения отводится по восемь байтов, что позволяет хранить числа достаточно большого диапазона. В Visual Basic .NET ему соответствует тип `Double`.
- `float`. Тип предназначен для хранения числовых данных. Для каждого значения отводится по четыре байта. В Visual Basic .NET ему соответствует тип `Single`.
- `int`. Тип используется для представления целочисленных значений. Для хранения каждого числа используются четыре байта. В Visual Basic .NET ему соответствует тип `Long`.
- `language`. Тип предназначен для хранения значений, указывающих на язык некоего текстового содержимого. Все возможные значения этого типа описаны в документе RFC 1766. В языке Visual Basic .NET для хранения подобных значений не предусмотрено отдельного типа, поэтому они хранятся как тип `String`. Разработчик сам должен будет обрабатывать эти значения.
- `long`. Тип также используется для хранения целочисленных значений. Для хранения каждого значения отводится восемь байтов. Соответственно, диапазон хранимых чисел простирается от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807. В Visual Basic .NET хранятся как данные типа `Variant`.
- `negativeInteger`. Тип предназначен для хранения отрицательных целочисленных значений. В Visual Basic .NET хранятся как данные типа `Variant`.
- `nonnegativeInteger`. Тип предназначен для хранения неотрицательных целочисленных значений, то есть положительных чисел и нулевого значения. В Visual Basic .NET хранятся как данные типа `Variant`.
- `positiveInteger`. Этот тип отличается от предыдущего только тем, что не включает в себя нулевое значение — он предназначен только для положительных целых чисел. Как и предыдущие типы, этот в Visual Basic .NET конвертируется в `Variant`.
- `short`. Это тоже целочисленный тип. Его отличием является то, что для хранения каждого значения отводится два байта. В Visual Basic .NET ему соответствует тип `Integer`.
- `string`. Здесь все понятно. Этот тип предназначен для хранения текстовых строк. Он практически идентичен типу `String` из Visual Basic .NET.
- `time`. Этот тип предназначен для хранения значений, указывающих на какое-либо время. В Visual Basic .NET эти значения все равно обрабатываются как `Date`.

- `unsignedByte`. Тип предназначен для целочисленных значений, как и тип byte, но при этом хранит целые беззнаковые числа. В Visual Basic .NET этот тип обрабатывается как обычный `Byte`.
- `unsignedInt`. Как и тип `int`, этот тип хранит целые числа, отводя для них по четыре байта. Отличие заключается в том, что значения этого типа не имеют знака. В Visual Basic .NET обрабатываются как значения типа `Variant`.
- `unsignedLong`. Как и тип `long`, хранит целые числа, отводя для них по восемь байтов. Отличие заключается в том, что значения этого типа не имеют знака. В Visual Basic .NET обрабатываются как значения типа `Variant`.
- `unsignedShort`. Как и тип `short`, хранит целые числа, отводя для них по два байта. Отличие заключается в том, что значения этого типа не имеют знака. В Visual Basic .NET обрабатываются как значения типа `Long`.

На этом мы заканчиваем рассмотрение простейших типов данных, которые чаще всего используются в документах WSDL. Теперь предстоит рассмотреть способы образования составных типов. Проще всего увидеть структуру этого механизма на несложном примере.

Предположим, нам необходимо создать составной тип, в котором бы хранилась информация о зарегистрированном пользователе сервиса. Мы будем хранить имя и фамилию человека, а также его возраст в годах. Для этого нам потребуются два поля, приспособленные для хранения строк, и одно поле для хранения целочисленного значения. Тогда конструкция объявления составного типа будет выглядеть следующим образом:

```
<element name="PERSON">
<xsd:complexType>
   <xsd:sequence>
      <xsd:element name="firstName" type="xsd:string"/>
      <xsd:element name="lastName" type="xsd:string"/>
      <xsd:element name="ageInYears" type="xsd:int"/>
   </xsd:sequence>
</xsd:complexType>
</element>
```

Легко заметить, что определение нового типа заключается в теги `<element>` и `</element>`. Затем, при помощи тегов `<xsd:complexType>` и `</xsd:complexType>`, где префикс `xsd` указывает на используемое пространство имен XML Schema, определяется, что объявляемый тип будет составным. В объявляющем теге `<element>` мы также использовали атрибут `name`, значением которого является наименование создаваемого составного типа. То есть наш объявляемый тип носит наименование `PERSON`.

Само указание структуры типа должно заключаться в теги <xsd:sequence> и </xsd:sequence>. Каждая отдельная часть типа объявляется при помощи тега <xsd:element>, у которого используются атрибуты name и type для указания наименования элемента и его типа соответственно. В нашем случае мы объявили два элемента типа string с наименованиями firstName и lastName, соответственно, и один элемент ageInYears типа int. Как можно видеть, объявление составных типов не является такой уж сложной задачей.

Язык WSDL позволяет объявлять не только отдельные простые и составные типы, но и их массивы, а также перечислимые типы. Для начала рассмотрим создание перечислимых типов.

Основной особенностью перечислимых типов является наличие четко ограниченного и явно объявленного множества значений: все возможные значения, которые может принимать переменная перечислимого типа, объявлены заранее. К перечислимым типам можно отнести перечень наименований дней недели или месяцев в году. Перечислимые типы — это перечни.

Как всегда, для начала рассмотрим простой пример объявления перечислимого типа. Так, для создания типа Sex, который будет состоять из двух возможных значений — Male и Female, требуется использовать следующий фрагмент кода:

```
<element name="Sex">
     <xsd:simpleType>
        <xsd:restriction base="xsd:string">
           <xsd:enumeration value="Male" />
           <xsd:enumeration value="Female" />
        </xsd:restriction>
     </xsd:simpleType>
</element>
```

Начало объявления типа, как мы видим, — традиционно. Используется тег <element>, внутри которого и размещается все объявление. Однако следующим использован тег <xsd:simpleType>, указывающий, что перечислимые типы не относятся к составным. После этого необходимо указать базовый тип, которому будут соответствовать все значения перечислимого типа. Естественно, значения, входящие в состав объявляемого типа, должны быть однотипными. В нашем случае мы используем значения типа string. Для объявления базового типа используется тег <xsd:restriction>. Сам базовый тип указывается в виде значения атрибута base, входящего в состав этого типа. Следует отметить, что тег <xsd:restriction> является не просто объявляющим тегом, он считается *охватывающим* тегом, поэтому разработчику необходимо использовать закрывающий близнец </xsd:restriction> и размещать перечисление возможных значений в промежутке между этой парой тегов.

Теперь можно переходить к перечислению всех возможных значений создаваемого типа. Для этого мы используем теги `<xsd:enumeration>`, в которых присутствует атрибут `value`. Значение этого атрибута и является одним из возможных значений перечислимого типа.

Перейдем к объявлению массивов. Вообще, следует признать, что в WSDL массивы объявляются не при помощи стандартного пространства имен XML Schema. Здесь для этих целей используется тип `Array`, позаимствованный из языка SOAP. Таким образом, для объявления массива следует воспользоваться пространством имен языка SOAP. Вот простой пример объявления целочисленного массива:

```
<xsd:complexType name="ArrayOfInt">
   <xsd:complexContent>
      <xsd:restriction base="soapenc:Array">
         <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int[]"/>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
```

Начнем по порядку рассматривать этот фрагмент кода. Мы объявляем тип с наименованием `ArrayOfInt`. Массивы считаются сложными типами, поэтому для его объявления мы используем тег `<xsd:complexType>`. Так как одна переменная массива содержит все-таки несколько значений, она сама в некотором роде является составной. Поэтому для создания массивов применяется тег `<xsd:complexContent>`, который и открывает секцию объявления содержимого типа.

После тега `<xsd:complexContent>` следует указать тип, на основе которого мы создаем свой массив. Как уже говорилось, в языке WSDL для объявления массивов используется пространство имен языка SOAP, и тип массива заимствуется оттуда. Именно поэтому в теге `<xsd:restriction>` параметр `base` имеет значение `soapenc:Array`. Если обратиться к *разд. "Краткий обзор"* данной главы, мы увидим, что пространство имен `soapenc` действительно было объявлено, и его реализация расположена по адресу **http://schemas.xmlsoap.org/soap/encoding/**.

После того как мы объявили, что создаваемый тип будет массивом, необходимо указать, какого именно типа будут элементы массива. Для этого в нашем объявлении использован следующий тег:

```
<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int[]"/>
```

Для нашего комплексного типа указание типа элементов производится через ссылку на элемент `arrayType` из пространства имен `soapenc`. Эта ссылка связывает искомый элемент с конструкцией `wsdl:arrayType="xsd:int[]"`. То есть тип массива мы задаем через одноименный элемент `arrayType`, но

уже принадлежащий пространству имен `wsdl`, которое также объявляется в начале WSDL-файла. Соответственно, мы используем тип `xsd:int` с проставленными квадратными скобками после его наименования. Эта конструкция и создает массив.

Теперь, когда мы знаем, как распознавать определения типов, собранных в разделе WSDL-документа `<types>`, можно перейти к рассмотрению структуры раздела `<message>`.

Структура элемента *<message>*

В элементах `<message>` описывается структура всех входящих и исходящих сообщений Web-сервиса. Если мы обратимся к WSDL-файлу, описывающему наш первый сервис, то мы увидим там несколько элементов `<message>`. Полностью этот блок выглядит так:

```
<message name="MyDateSoapIn">
    <part name="parameters" element="s0:MyDate" />
</message>
<message name="MyDateSoapOut">
    <part name="parameters" element="s0:MyDateResponse" />
</message>
<message name="MyDateHttpGetIn">
    <part name="ShowTime" type="s:string" />
</message>
<message name="MyDateHttpGetOut">
    <part name="Body" element="s0:string" />
</message>
<message name="MyDateHttpPostIn">
    <part name="ShowTime" type="s:string" />
</message>
<message name="MyDateHttpPostOut">
    <part name="Body" element="s0:string" />
</message>
```

Легко заметить, что в этом блоке содержится шесть разделов `<message>`. Так как наш сервис, как и все стандартные Web-сервисы концепции Microsoft .NET, поддерживает три способа передачи и получения информации (`GET`, `POST` и SOAP), то можно убедиться, что на каждый метод приходится по два элемента `<message>`. То есть для каждого метода один элемент описывает входящее сообщение, а второе — исходящее.

Кстати, эту структуру всего раздела можно распознать и по наименованиям элементов `<message>`. Они складываются из наименования функции

Web-сервиса (MyDate), метода передачи информации (HttpPost, HttpGet, Soap), и направленности сообщения (In, Out). Теперь рассмотрим структуру стандартного элемента <message>. В нашем примере он выглядит следующим образом:

```
<message name="MyDateSoapIn">
    <part name="parameters" element="s0:MyDate" />
</message>
```

В открывающем теге <message> при помощи атрибута name мы указываем наименование сообщения, созданное по образцу, рассмотренному выше. В общем случае совершенно необязательно использовать именно этот алгоритм образования имени. Можно использовать любой уникальный идентификатор. Но для того чтобы клиентское приложение могло самостоятельно распознавать структуру сервиса по его WSDL-файлу, стоит все-таки придерживаться этого правила.

Внутри раздела <message> указываются все передаваемые в нем отдельные элементы. Они объявляются при помощи тегов <part>. В этом теге используются атрибуты name и element для задания наименования передаваемой переменной и типа ее значения, соответственно. В нашем случае мы описываем входящее сообщение на языке SOAP. Поэтому передается переменная с наименованием parameters, тип которой описан в разделе <types> под именем MyDate.

Однако следует отметить, что два вышеописанных атрибута, входящие в состав тега <message>, не являются единственными. Помимо них могут использоваться и другие атрибуты, такие как type для указания типа переменной, и иные атрибуты спецификации XML Schema, описывающие тип и ограничения, налагаемые на переменную. Но чаще всего в дополнительных атрибутах нет нужды.

Вернемся к рассматриваемому нами фрагменту кода. Если в разделах <message>, описывающих входящее и исходящее сообщения на языке SOAP, использовались переменные с наименованием parameters, то сообщения, связанные со стандартными методами протокола HTTP, выглядят несколько иначе. Вот типичный пример входящего сообщения, основанного на методе POST:

```
<message name="MyDateHttpPostIn">
    <part name="ShowTime" type="s:string" />
</message>
```

Как мы помним, вызываемая функция MyDate имеет один параметр с наименованием ShowTime. Именно он и описывается в теге <part>. При этом для указания типа передаваемого параметра используется пространство имен XML Schema. Теперь обратимся к блоку <message>, который описывает ис-

ходящее сообщение. Определение этого сообщения приведено в следующем фрагменте кода:

```
<message name="MyDateHttpPostOut">
    <part name="Body" element="s0:string" />
</message>
```

В этом блоке описывается исходящее сообщение, которое возвращает результат выполнения функции Web-сервиса. Следует обратить внимание, что помимо изменения наименования элемента (и это понятно, потому что возвращаемое значение практически никак не связано с переданным параметром), изменено и пространство имен для указания типа возвращаемого значения. Для возвращаемого значения используется внутреннее пространство имен Web-сервиса с общим наименованием `s0`. Если обратиться к исходному коду контракта Web-сервиса, то можно увидеть, что это пространство имен объявлено следующим образом:

```
xmlns:s0="http://tempuri.org/" targetNamespace="http://tempuri.org/"
```

Те же самые правила образования блоков `<message>` справедливы и для сообщений, связанных со стандартным методом `GET` протокола HTTP.

Каждое объявление сообщения будет связано с портом, описываемым блоком `<port>`. Но прежде чем мы будем рассматривать структуру этих блоков, следует обратить внимание на блоки `<portType>`, которые как раз и призваны задавать структуру портов.

Структура элемента *<portType>*

Начнем мы как всегда с рассмотрения искомого раздела, приведенного в нашем основном примере. Вот этот фрагмент кода:

```
<portType name="Service1Soap">
    <operation name="MyDate">
      <input message="s0:MyDateSoapIn" />
      <output message="s0:MyDateSoapOut" />
    </operation>
</portType>
<portType name="Service1HttpGet">
    <operation name="MyDate">
      <input message="s0:MyDateHttpGetIn" />
      <output message="s0:MyDateHttpGetOut" />
    </operation>
</portType>
<portType name="Service1HttpPost">
```

```
    <operation name="MyDate">
      <input message="s0:MyDateHttpPostIn" />
      <output message="s0:MyDateHttpPostOut" />
    </operation>
</portType>
```

В этом фрагменте объявлено три блока `<portType>`. Если обратить внимание на значения атрибутов name, входящих в состав этого тега, то можно увидеть, что все они описывают методы доступа к описываемому Web-сервису. Так, если взять для рассмотрения блок `<portType>`, описывающий доступ к сервису при помощи метода `POST` протокола HTTP, то мы увидим следующий фрагмент кода:

```
<portType name="Service1HttpPost">
    <operation name="MyDate">
      <input message="s0:MyDateHttpPostIn" />
      <output message="s0:MyDateHttpPostOut" />
    </operation>
</portType>
```

Структура блока `<portType>`, равно как и остальных блоков файла контракта, достаточно прозрачна. При помощи тега `<operation>` задается наименование функции, к которой будет предоставлять доступ какой-либо порт. В нашем случае это функция `MyDate`. Внутри блока `<operation>` описываются входящие и исходящие данные при помощи тегов `<input>` и `<output>`, соответственно. А входящие и исходящие данные, как мы уже знаем, описываются при помощи сообщений. Что и наблюдается в этом блоке. Входные данные для порта, предоставляющего доступ к функции `MyDate` при помощи метода `POST`, описываются сообщением `MyDateHttpPostIn`. Это сообщение было рассмотрено в предыдущем разделе. Соответственно, выходные данные задаются сообщением `MyDateHttpPostOut`. При этом все наименования сообщений указаны с применением внутреннего пространства имен `s0`.

Следует отметить, что бывает несколько основных видов блоков `<portType>`. В нашем случае рассматривались блоки, которые описывают одновременно и входящие, и исходящие сообщения. В терминологии спецификации WSDL такой тип портов называют *запрос-ответ* (request-response). Но помимо этого типа могут встречаться блоки `<portType>`, описывающие функции, которые поддерживают только входящие или только исходящие данные. Такие функции называют *однонаправленными* (one-way). Пример описания подобного типа порта приведен в следующем фрагменте кода:

```
<portType name="myService">
        <operation name="MyFunction">
           <input message="MyFunctionPostIn"/>
```

```
        </operation>
</portType >
```

Этот блок описывает порт, который передает только входные данные.

Следует заметить также, что обычный тип порта может описывать не только входящие и исходящие сообщения. В его состав может включаться описание сообщения, выдаваемого сервисом в случае возникновения какой-либо ошибки. Такой блок может выглядеть следующим образом:

```
<portType name="Service1HttpPost">
    <operation name="MyDate">
      <input message="s0:MyDateHttpPostIn" />
      <output message="s0:MyDateHttpPostOut" />
      <fault message="s0:MyDateHttpPostFault" />
    </operation>
</portType>
```

Естественно, для того, чтобы применять подобный расширенный вариант объявления типа, следует позаботиться и об объявлении соответствующего дополнительного сообщения. Вообще же, дополнительное сообщение, извещающее о возникновении какой-либо ошибки, дублирует концепцию извещения об ошибках языка SOAP. Впрочем, в *главе 4*, посвященной этому языку, мы увидим принцип действия упомянутой концепции.

Теперь следует рассмотреть формат элементов `<binding>`, часто называемых *привязками*, которые определяют формат и протокол передачи данных для каждого объявленного типа порта. Об этом речь пойдет в следующем разделе.

Структура элемента *<binding>*

Как только что было показано в предыдущем разделе, элементы `<binding>` детально описывают формат данных и протокол их передачи для каждого определенного типа порта. Типы порта, как известно, задаются блоками `<portType>`. В нашем примере было три подобных блока. Следовательно, в файле контракта должно находиться и три блока `<binding>`. Так оно и есть. Фрагмент нашего знакомого WSDL-файла с этими блоками выглядит следующим образом:

```
<binding name="Service1Soap" type="s0:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation name="MyDate">
      <soap:operation soapAction="http://tempuri.org/MyDate"
style="document" />
      <input>
```

```
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
<binding name="Service1HttpGet" type="s0:Service1HttpGet">
  <http:binding verb="GET" />
  <operation name="MyDate">
    <http:operation location="/MyDate" />
    <input>
      <http:urlEncoded />
    </input>
    <output>
      <mime:mimeXml part="Body" />
    </output>
  </operation>
</binding>
<binding name="Service1HttpPost" type="s0:Service1HttpPost">
  <http:binding verb="POST" />
  <operation name="MyDate">
    <http:operation location="/MyDate" />
    <input>
      <mime:content type="application/x-www-form-urlencoded" />
    </input>
    <output>
      <mime:mimeXml part="Body" />
    </output>
  </operation>
</binding>
```

Как можно видеть, наименования элементов, объявляющих привязки, совпадают с наименованиями соответствующих типов портов. Впрочем, это совершенно не обязательно, так как помимо наименований привязок, которые сами по себе могут не нести никакой смысловой нагрузки, лишь бы они были уникальными, в объявляющем теге `<binding>` используется атрибут `type`, задающий тип искомой привязки. А значением этого атрибута и будет искомый тип соответствующего порта, определенный при помощи внутреннего пространства имен.

Теперь рассмотрим детально структуру привязки, соответствующую порту, предоставляющему доступ к функциональности Web-сервиса при помощи языка SOAP. Этот фрагмент кода выглядит следующим образом:

```
<binding name="Service1Soap" type="s0:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <operation name="MyDate">
      <soap:operation soapAction="http://tempuri.org/MyDate"
style="document" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
</binding>
```

После объявляющего тега `<binding>`, в котором при помощи атрибутов `name` и `type` задаются наименование привязки и соответствующий тип порта, размещается тег, указывающий протокол, по которому будет происходить обмен информацией с портом. Для SOAP в нашем случае этот тег имеет следующий вид:

```
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
```

Как можно видеть, для объявления протокола используется тег `<binding>`, объявленный в пространстве имен `soap`. Естественно, этот тег отличается от начального тега блока привязки. Сам протокол передачи данных указывается при помощи атрибута `transport`, в качестве значения которого используется объявление HTTP в языке SOAP.

Следует отметить, что для привязок, ориентированных на стандартные методы `GET` и `POST` протокола HTTP, объявление протокола, как, впрочем, и объявления операций, выглядит несколько иначе. Но об этом — чуть позже. Пока же речь идет о привязке, ориентированной на SOAP.

После того как был объявлен протокол передачи данных от сервиса к клиенту и обратно, следует объявление операции, к которой будет привязываться порт. Объявление операции производится при помощи тега `<operation>`. В объявляющем теге `<operation>` присутствует атрибут `name`, значением которого является имя функции, к которой привязывается порт. В нашем случае это, естественно, функция `MyDate`.

После объявления операции следует указание адреса, по которому производится доступ к функции. В нашем случае, для доступа при помощи SOAP, эта конструкция выглядит следующим образом:

```
<soap:operation soapAction="http://tempuri.org/MyDate" style="document" />
```

После этого остается лишь объявить все входящие и исходящие данные при помощи блоков, задаваемых тегами `<input>` и `<output>`, соответственно. В нашем случае входящие и исходящие данные приведены к одному типу, поэтому и содержимое искомых блоков выглядит абсолютно одинаково:

```
<soap:body use="literal" />
```

Теперь рассмотрим структуру блока привязки `<binding>` для использования стандартных методов `GET` и `POST` протокола HTTP. Для метода `POST` блок объявления привязки выглядит следующим образом:

```
<binding name="Service1HttpPost" type="s0:Service1HttpPost">
    <http:binding verb="POST" />
    <operation name="MyDate">
      <http:operation location="/MyDate" />
      <input>
        <mime:content type="application/x-www-form-urlencoded" />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
</binding>
```

Следует обратить внимание на тег, объявляющий протокол передачи данных:

```
<http:binding verb="POST" />
```

То есть, ключевое слово `binding` употреблено в области действия пространства имен `http`. А наименование используемого метода `POST` передается как значение атрибута `verb`.

Указание расположения функции, к которой привязывается искомый порт `Service1HttpPost`, производится, как и в предыдущем случае, при помощи тега `<operation>`. Однако он опять задается при помощи пространства имен `http`, а сам адрес указан в значении атрибута `location`. При этом указывается не полный адрес, а всего лишь виртуальный каталог.

Следует также обратить внимание, как именно объявляется формат передаваемых и принимаемых данных в блоках `<output>` и `<input>`:

```
<mime:mimeXml part="Body" />
```

Здесь для ссылки на возвращаемый XML-код используется пространство имен mime, также объявленное в самом начале файла контракта.

Итак, мы рассмотрели структуру блоков `<binding>`, связывающих порядок передачи данных через порты, которые в свою очередь являются точками доступа к функциям Web-сервиса. Однако структуру объявления портов мы еще не рассматривали. Об этом — в следующем разделе.

Структура элемента *<port>*

Элементы `<port>` являются частями элемента `<service>`. Их структура чрезвычайно проста для понимания, так как единственное их предназначение — указывать адреса для доступа к функциональности Web-сервиса. Таким образом, это точки доступа к функциям. Стандартное объявление одного порта в рассматриваемом нами файле контракта выглядит следующим образом:

```
<port name="Service1HttpGet" binding="s0:Service1HttpGet">
   <http:address location="http://localhost/MyService/Service1.asmx" />
</port>
```

В объявляющем теге `<port>` при помощи параметра `name` задается наименование порта, а при помощи параметра `binding` — наименование привязки для этого порта. Внутри блока `<port>` располагается один тег, указывающий на адрес доступа к функции, с которой этот порт связывает привязка. При этом следует обратить внимание на тот факт, что все три порта, используемые в нашем сервисе, указывают на одну и ту же функцию `MyDate`, следовательно, и адрес точки входа, который указывается в виде значения атрибута `location`, будет для всех трех портов одинаковым.

Основные правила для декларирования портов достаточно просты. Каждый порт может объявлять только один адрес доступа к функции Web-сервиса, и каждый порт может использовать только одну привязку.

Итак, как мы убедились, правила объявления портов достаточно просты, и теперь настало время перейти к рассмотрению наивысшего во всей иерархии контракта элемента — `<service>`.

Структура элемента *<service>*

Как мы уже знаем, элемент `<service>` является основным во всей иерархии компонентов WSDL-файла. Он наименее абстрактен, если можно так выразиться. Вся подготовительная работа была уже проведена предыдущими элементами, и осталось лишь описать структуру Web-сервиса в целом.

В рассматриваемом нами WSDL-файле блок `<service>` выглядит следующим образом:

```
  <service name="Service1">
    <port name="Service1Soap" binding="s0:Service1Soap">
      <soap:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
    <port name="Service1HttpGet" binding="s0:Service1HttpGet">
      <http:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
    <port name="Service1HttpPost" binding="s0:Service1HttpPost">
      <http:address location="http://localhost/MyService/Service1.asmx"
/>
    </port>
  </service>
```

Как можно видеть, блок `<service>` состоит только из портов. И это вполне понятно, так как именно ими и описывается самый верхний уровень абстракции Web-сервиса. Так что, для описания его структуры в целом необходимо лишь перечислить порты, предоставляющие клиентским приложениям доступ ко всем функциям сервиса.

Следует помнить, что в блоке `<service>` перечисляемые порты не должны ссылаться друг для друга. То есть, исходящая информация какого-либо порта не может напрямую переводиться во входящую информацию для другого порта: вся коммуникация должна быть ориентирована только на клиентские приложения и иные Web-сервисы.

Итак, мы рассмотрели структуру WSDL-файлов, которые описывают функциональность Web-сервисов и правила доступа к его функциям. Теперь нам предстоит познакомиться с языком SOAP, который чаще всего используется для общения с серьезными многофункциональными сервисами.

Глава 4

Язык SOAP

Основы

В *главе 2* мы рассматривали создание WEB-сервисов, сообщающихся со своими клиентами посредством платформенно-независимых способов передачи данных. Подобных способов три, но протокол для них используется один — HTTP. Два способа основаны на методах `GET` и `POST`, присущих HTTP, но самым интересным и прогрессивным, пожалуй, является использование языка XML для передачи данных от сервиса к клиенту и обратно. При этом используется не "сырой" XML, а основанный на нем SOAP (Simple Object Access Protocol).

SOAP, как следует из его названия, это простой протокол доступа к объектам. Microsoft давно лелеяла надежду создать сервисы, к которым клиенты могли бы обращаться по общим правилам запроса. Первая попытка была сделана задолго до внедрения технологии Microsoft .NET. Протокол SOAP применялся для создания сервисов на базе сервера BizTalk. Попытку внедрения этой достаточно сильно связанной пары технологических решений вряд ли можно назвать успешной. Но с появлением феноменальной платформы Microsoft .NET протокол SOAP обрел второе дыхание. Он действительно удобен.

Буквально с самых первых версий операционной системы Windows был взят курс на как можно более плотную интеграцию различных приложений. Каждый шаг на этом тернистом пути, будь то DDE (Dynamic Data Exchange) или OLE (Object Linking and Embedding), объявлялся серьезным прорывом. Впрочем, будем объективны, почти всегда именно так и было. Любое усовершенствование в этой области давалось очень нелегко.

Однако впервые свет в конце тоннеля (хотя еще не известно, существует ли вообще из этого тоннеля выход) забрезжил с внедрением технологии COM (Common Object Model). Эта объектная модель позволяла создавать различные функциональные объекты, которые предоставляли иным программам при помощи специализированных интерфейсов средства работы с собой. Так, если некий COM-объект был способен проверять орфографию в тексте,

любое приложение могло использовать эту его способность, воспользовавшись стандартным интерфейсом объекта COM. Таким образом, COM-объекты становились чем-то вроде блоков конструктора Лего, из которых умелый разработчик собирал в короткие сроки мощное приложение.

Основным ограничением подобных объектов служила их привязанность к одной платформе — Windows. Конечно, она очень сильно распространена в компьютерной индустрии (причем, в данном случае, слово "*очень*" может оказаться преуменьшением), но все равно, область распространения COM-объектов ограничена. Следует также осознавать, что существуют объективные трудности с их использованием в среде Интернет.

Однако идея унифицированных вызовов и стандартизованного доступа к самым различным функциональным объектам на самом деле хороша. И ущемлять интернет-приложения в этой области нет никакого резона. Однако в Интернете много ярче, чем в любой другой области компьютерной индустрии, проявляется именно многоплатформенность. А она никогда не была сильной стороной Microsoft. Но если решение необходимо найти — скорее всего, оно будет найдено. Особенно, если это решение действительно нужно такой крупной корпорации. И она его нашла.

Давайте вспомним стандартную архитектуру взаимодействия в Web. Браузер посылает серверу запрос на получение некоей информации при помощи протокола HTTP, сервер получает запрос, находит требуемую информацию и передает ее браузеру, пользуясь все тем же протоколом HTTP. При этом абсолютно неважно, на каких платформах функционируют браузер и сервер, так как для своего взаимодействия они используют *платформеннонезависимый* протокол HTTP. Симптоматично и то, что чаще всего данный протокол используется для передачи HTML-файлов, которые также могут отображаться в любом браузере и в любых условиях, то есть данные файлы тоже не зависят от платформы, применяемой удаленным пользователем.

Следовательно, основываясь на подобной архитектуре, можно создать сервисы и клиенты, которые также могли бы передавать и принимать информацию по протоколу HTTP. И если используется протокол, не зависящий от конкретной платформы, то сервис и клиентская программа тоже не обязаны функционировать в одной и той же операционной системе. Здесь есть еще две нерешенные проблемы. Во-первых, сервис должен быть самодокументируемым, то есть поставлять информацию о себе неким стандартным способом. Клиент может получить эту информацию, а затем на ее основе строить свое дальнейшее взаимодействие с сервисом. Во-вторых, необходимо осознавать, что протокол HTTP является лишь транспортным протоколом и не подходит в качестве *языка* коммуникации. Для отображения Web-страниц использовался язык HTML, но его возможностей явно будет не хватать для взаимодействия программ. Поэтому на роль языка коммуникации был выбран SOAP.

SOAP — это протокол, то есть набор правил для создания приложений, которые могут вызывать методы удаленных объектов. Где именно находятся эти удаленные объекты: в другом каталоге, где-то в корпоративной интрасети или в Интернете — для клиентских программ, использующих SOAP, абсолютно неважно. SOAP основан на XML. По сути дела, каждая передача информации между клиентом и сервисом является отдельным XML-документом, который написан по правилам SOAP.

Как мы знаем, логическая структура каждого XML-документа определяется его DTD-блоком. Так вот, для SOAP заранее определены все возможные теги и типы данных, поэтому SOAP-документы не нуждаются в DTD-блоках. За счет подобной унификации сервисы и клиенты освобождаются от достаточно тяжелой процедуры синтаксического анализа приходящего документа с не определенным заранее набором тегов.

Структура SOAP

Любой документ SOAP (хотя в данном случае лучше все-таки называть их не документами, а сообщениями, так как это лучше передает их предназначение) состоит из двух основных частей: заголовка и собственно тела документа. Причем все это упаковывается в некий "конверт" и в таком виде передается по протоколу HTTP.

Естественно, все эти уровни иерархии создаются при помощи определенных тегов. Простейший пример того, как выглядит SOAP-запрос, упакованный в протокол HTTP, показан в листинге 4.1.

Листинг 4.1

```
POST /MyService HTTP/1.1
Host: www.servicesite.com
Content-Type:text/xml; charset="utf-8"
Contant-Length: xxxx
SOAPAction: "www.servicesite.com/action"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
<SOAP-ENV:Body>
  <m:GetSomeValue xmlns:m=" www.servicesite.com/action ">
    <symbol>SYM</symbol>
  </m:GetSomeValue>
</SOAP-ENV:Body>
</Soap-ENV:Envelope>
```

Естественно, первые пять строк относятся не к самому SOAP-запросу, а к протоколу HTTP, по которому он доставляется. Шестая строка объявляет конверт SOAP-запроса. В данном примере показан запрос без заголовка, так как в седьмой строке объявляется тело SOAP-запроса, внутри которого располагаются некоторые данные.

Теперь, когда мы увидели пример стандартного SOAP-запроса, можно поговорить об основных правилах составления и обработки SOAP-сообщений. Прежде всего, приложение, которое получает SOAP-сообщение, должно выделить все его элементы, предназначенные именно для этого приложения. После выделения подобных компонентов, приложение обрабатывает их согласно заложенной в него логике. Если SOAP-сообщение не содержит некоторых обязательных частей, оно может (но не обязано) быть приложением отвергнуто. Однако в том случае, если отсутствие этих частей не является для приложения критичным, оно может продолжить обработку SOAP-запроса. Если SOAP-сообщение предназначено не только для этого конкретного обрабатывающего приложения, то перед отправкой его следующему получателю, приложение должно удалить из сообщения все элементы, относящиеся только к обрабатывающему приложению. Таким образом, переходя по цепочке обрабатывающих приложений, SOAP-сообщение может "худеть".

Что касается самого SOAP-сообщения, то оно, как мы уже знаем, обязано быть составлено согласно правилам синтаксиса XML. Для всех элементов сообщения и их стандартных атрибутов должны использоваться соответствующие префиксы, указанные в пространствах имен. При этом явные ссылки в теле SOAP-сообщения на используемые пространства имен не являются обязательными. В случае их отсутствия, обрабатывающее приложение считает, что указаны два стандартных пространства имен. Но если в теле SOAP-сообщения находится неверная ссылка на стандартное пространство имен или указано какое-либо дополнительное пространство имен, приложение должно отказаться от обработки SOAP-сообщения. Подобное ограничение действует на уровне конвертов SOAP-сообщений.

Как было сказано, в SOAP используются два стандартных пространства имен. Все содержимое SOAP-конверта использует пространство имен, располагающееся по адресу **http://shemas.xmlsoap.org/soap/envelope**, а применяемые кодировки определены в пространстве имен, находящемся по адресу **http://schemas.xmlsoap.org/soap/encoding**. В нашем первом примере мы видели на них ссылки.

Так как все возможные элементы сообщений SOAP заранее определены, то нет нужды использовать DTD-блоки. Более того, если в сообщении SOAP будет находиться DTD-блок или даже ссылка на него, оно будет считаться некорректным.

Теперь попробуем определить соотношения между тремя основными блоками SOAP-сообщения. Начнем с более или менее формального описания конверта.

Конверт определяется при помощи элемента `Envelope`. Это обязательный компонент сообщений SOAP. В нем могут (но не обязаны) быть размещены ссылки на используемые пространства имен. Если приложение или разработчик размещают в конверте дополнительные элементы, они должны быть описаны в одном из стандартных пространств имен.

Заголовок сообщения SOAP объявляется при помощи элемента `Header`. Данный компонент не является обязательным. Он используется в тех случаях, когда необходимо расширить стандартную функциональность SOAP-сообщения. В заголовке находятся элементы, которые подскажут приложению, обрабатывающему SOAP-запрос, какие именно функции будут реализованы при помощи данного SOAP-сообщения. Заголовок SOAP-сообщения является прямым потомком конверта, то есть на уровне тегов он просто вкладывается в теги с наименованием `Envelope`.

Тело SOAP-сообщения объявляется при помощи элемента `Body`. Этот компонент является обязательным. Так же, как и заголовок, он является прямым потомком конверта. Естественно, как тело, так и заголовок могут существовать в сообщении SOAP только в единственном экземпляре. Если у сообщения есть заголовок, тело сообщения должно объявляться сразу после заголовка. То есть открывающий тег тела сообщения должен вплотную примыкать к закрывающему тегу заголовка сообщения.

Во всех компонентах SOAP-сообщения, на любом уровне вложенности тегов может использоваться атрибут `encodingStyle`, который сообщает, какой набор правил должен применяться для представления данных компонента. Чаще всего при помощи этого атрибута указывают используемую кодировку символов. В качестве значений атрибута используются элементы уже упоминавшегося пространства имен **http://schemas.xmlsoap.org/soap/encoding**. Если для какого-либо компонента SOAP-сообщения не указан явным образом данный атрибут, то используются атрибуты родительских компонентов. Проще говоря, если разработчик или приложение задали в заголовке правила отображения, то они будут распространены на все приложение за исключением тех его элементов, в которых явным образом установлено другое значение данного атрибута.

Теперь перейдем к более детальному рассмотрению правил создания заголовка и тела SOAP-сообщений.

Заголовки сообщений SOAP

Как мы уже знаем, заголовок является необязательной частью SOAP-сообщения. Однако эта часть сообщения позволяет несколько расширять стандартную функциональность SOAP. Необходимо помнить, что любой элемент заголовка должен быть описан со ссылкой на пространство имен,

в котором данный элемент объявляется. Так, если мы хотим использовать в заголовке некий элемент, в котором мы бы передавали, скажем, номер транзакции, то заголовок может выглядеть следующим образом:

```
<SOAP-ENV:Header>
<trans:Transaction
xmlns:trans="http://www.myhost.com/namespaces/myspace/"
SOAP-ENV:mustUnderstand="1">
12
</trans:Transaction>
</SOAP-ENV:Header>
```

В этом маленьком примере мы объявили элемент заголовка с наименованием `Transaction`. При этом пространство имен, в котором описан данный элемент, находится по адресу **http://www.myhost.com/namespaces/myspace/**. Значение элемента `Transaction` в данном случае равно двенадцати. Также данный элемент обладает атрибутом `mustUnderstand` с единичным значением. Об этом атрибуте следует поговорить несколько подробнее.

Когда отправитель SOAP-сообщения его передает, он предполагает, что получатель сообщения будет иметь некоторое представление о структуре сообщения. Так, если функционирует связка из двух приложений, разработанных специально друг для друга, то разработчик, естественно, заложит в них обработку неких элементов заголовка. При этом и получатель, и отправитель знают наименования элементов заголовка, и, соответственно, будут их искать и создавать. Так вот, для тех элементов заголовка, которые получатель должен понимать, устанавливается атрибут `mustUnderstand` с приписанным ему, как в нашем примере, единичным значением.

Подобный атрибут может принимать всего два значения — единичное и нулевое. Атрибут `mustUnderstand` с нулевым значением эквивалентен отсутствию этого атрибута.

Если обрабатывающее приложение не в состоянии правильно распознать элемент заголовка с атрибутом `mustUnderstand`, оно обязано вернуть SOAP-сообщение с указанием кода возникшей ошибки. Точно так же ему следует поступить, если значение этого атрибута будет отличаться от нуля или единицы.

Помимо атрибута `mustUnderstand`, у элементов заголовка SOAP-сообщения может быть атрибут `actor`. Как говорилось несколько ранее, одно и то же сообщение иногда попадает на обработку в несколько приложений. При этом оно передается между ними по цепочке, т. е. приложение, обработавшее сообщение, передает его следующему получателю. При этом в заголовке сообщения могут находиться элементы, предназначенные для разных приложений. И далеко не факт, что элементы, помеченные как `mustUnderstand`,

будут "понятны" каждому приложению в цепочке. Приложения потому и разные, что у каждого своя функциональность, и для каждого есть свой набор элементов, которые они обязаны "понимать". Поэтому необходимо каким-либо образом указывать, для какого именно приложения предназначен тот или иной элемент. Именно для этой цели и существует атрибут `actor`.

Каждому приложению, обрабатывающему SOAP-сообщение, придается определенный уникальный идентификатор. И тогда в качестве значения атрибута `actor` некоего элемента используется идентификатор именно того приложения, которое должно обрабатывать данный элемент. Так как каждое приложение знает собственный идентификатор, выделение именно тех элементов заголовка, которые ему нужны, становится тривиальной задачей.

После того как приложение обработало свои элементы, перед пересылкой SOAP-сообщения следующему приложению из цепочки обработчиков, оно обязано удалить из него все элементы, которые были им обработаны. Таким образом одновременно уменьшается размер пересылаемого пакета и облегчается его синтаксический анализ следующим приложением.

Следует также отметить, что в качестве значения данного атрибута можно использовать идентификатор **http://schemas.xmlsoap.org/soap/actor/next**, который указывает, что сообщение должно быть передано первому приложению из списка получателей. Если же атрибут actor вообще не применялся в объявлении элементов заголовка SOAP-сообщения, это означает, что у данного SOAP-сообщения есть только один получатель.

Теперь, после того, как мы рассмотрели правила, по которым составляется заголовок SOAP-сообщения, перейдем к рассмотрению структуры тела SOAP-сообщения.

Тело SOAP-сообщения

Как мы уже знаем, именно в теле SOAP-сообщения передаются все основные данные, которые необходимо обработать получателю. Эти данные заключены между стартовым и конечным тегами элемента `Body`, которым, собственно, и объявляется тело SOAP-сообщения. Как и в заголовке, каждый элемент должен быть описан при помощи некоего пространства имен. Полное наименование элемента должно состоять из его имени и идентификатора используемого пространства имен.

Элементы тела SOAP-сообщения соответствуют тем элементам заголовка сообщения, которые обладают атрибутом `mustUnderstand` с единичным значением. Однако с практической точки зрения, элементы тела SOAP-сообщения можно рассматривать как удаленные вызовы некоторых методов обрабатывающего приложения. То есть приложение, отсылающее SOAP-сообщение, при помощи элементов его тела заставляет принимающее при-

ложение выполнять те или иные действия. Если мы еще раз посмотрим на листинг 4.1, то увидим, что тело сообщения объявлялось при помощи следующего фрагмента кода:

```
<SOAP-ENV:Body>
  <m:GetSomeValue xmlns:m=" www.servicesite.com/action">
    <symbol>SYM</symbol>
  </m:GetSomeValue>
</SOAP-ENV:Body>
```

Первая и последняя строки содержат открывающий и закрывающий, соответственно, теги тела сообщения. А между ними располагается информация, которая и предназначена для обрабатывающего приложения. Вторая строка, по сути дела, является вызовом некоего метода с внутренним наименованием `GetSomeValue`. При этом в качестве параметра данному методу передается значение `SYM` типа `symbol`. Естественно, имена типов и методов являются условными. Наименование метода может не совпадать с наименованием функции, этот метод реализующей. А тип значения, естественно, распознается самим обрабатывающим приложением.

Однако помимо элементов, специфичных для каждого приложения, в теле SOAP-сообщения может использоваться один стандартный элемент. Он носит наименование `Fault`, и предназначен, как нетрудно догадаться, для передачи информации о возникших ошибках. Этот элемент обязан находиться именно в теле SOAP-сообщения и не может быть повторен два или более раз. То есть, проще говоря, он должен там быть в единственном экземпляре.

Данный элемент может содержать четыре встроенных элемента. Рассмотрим их подробно.

- Элемент `faultcode` предназначен для передачи кода возникшей ошибки. Этот элемент обязательно должен входить в состав своего родительского элемента `Fault`. Его значением должно быть полное наименование ошибки.
- Элемент `faultstring` содержит текстовое описание ошибки. Он также обязан входить в состав родительского элемента `Fault`. На значение этого элемента не накладывается каких-либо жестких условий, но все же рекомендуется вносить в него дополнительную информацию о возникшей ошибке, так как стандартные коды ошибок могут указать лишь тип возникшей проблемы, но не локализовать ее.
- Элемент `faultactor` указывает на приложение, в котором и возникла ошибка. Если у SOAP-сообщения есть всего лишь один адресат, тогда нет нужды использовать этот элемент, так как и без того понятно, что ошибка могла возникнуть лишь при работе единственного адресата. Но вот если для сообщения заготовлена цепочка обработчиков, и ошибка возникла не по вине последнего в этой цепочке приложения, то прило-

жение, в котором возникла ошибка, обязано включить в состав элемента `Fault` данный элемент `faultactor` и указать в качестве его значения свой собственный идентификатор.

- Элемент `detail` служит для передачи детальной информации о возникшей ошибке. Он обязан включаться в состав элемента `Fault`, если ошибка возникла в процессе обработки тела сообщения. Если же ошибка появилась при обработке иных компонентов SOAP-сообщения, наличие данного элемента не является обязательным. В состав данного элемента должны входить иные элементы, определяемые приложением, в которых оно будет указывать локализованную причину ошибки и возможные действия отправителя, которые необходимо произвести в ответ на получение сообщения об ошибке.

Здесь также следует рассмотреть *стандартные классы ошибок*, которые являются значениями элемента `faultcode`. Они определены в пространстве имен **http://shemas.xmlsoap.org/soap/envelope**.

- Код `VersionMismatch` указывает, что приложение-обработчик обнаружило ссылку на неверное пространство имен в элементе `Envelope`.
- Код `MustUnderstand` сигнализирует, что в заголовке SOAP-сообщения обнаружен элемент с атрибутом `mustUnderstand`, которому приписано единичное значение, и этот элемент не распознан обрабатывающим приложением, и, соответственно, не был им обработан.
- Код `Client` свидетельствует о возникновении класса ошибок, которые не могут быть самостоятельно устранены обрабатывающим приложением. Чаще всего возникновение подобных ошибок свидетельствует о том, что в сообщение не была вложена некая важная информация.
- Код `Server` указывает, что ошибка возникла на стороне обрабатывающего приложения, но при этом она возникла не из-за содержимого SOAP-сообщения. Чаще всего это свидетельствует, что обрабатывающее приложение не смогло выполнить ряд своих функций, например, из-за того, что не удалось загрузить некоторые свои удаленные компоненты. Обычно при получении сообщения с ошибкой подобного класса отсылающее приложение может повторно послать SOAP-сообщение в надежде на то, что принимающее приложение все-таки сможет его правильно обработать.

Все данные, передаваемые в теле SOAP-сообщения, должны принадлежать к одному из заранее предопределенных типов. Как мы знаем, изначально в XML нет типов данных. Соответствие между значением и его типом самостоятельно устанавливалось приложением, обрабатывающим XML-документ. В спецификации SOAP заранее определены возможные типы значений. Их мы рассмотрим в следующем разделе.

Типы данных SOAP

В стандарте XML не предусмотрены стандартные типы данных. Из-за его гибкости нет нужды каким-либо особым образом объявлять единый набор типов данных для всех XML-документов. Каждый разработчик может сам объявить именно те типы данных, которые будут применяться в его наборе XML-документов. Однако подобные типы должны, естественно, объявляться в DTD-блоке. Так как в SOAP-сообщениях нет DTD-блоков, то для них заранее объявлены возможные типы данных.

Структура типов для SOAP-сообщений в достаточно большой мере унаследована из словаря XML Schema. Данный словарь был введен в обращение корпорацией Microsoft. Он позволял пользоваться заранее определенным набором тегов и типов данных, таким образом несколько сужая функциональность XML, но приводя документы к некоему стандартизованному виду.

В SOAP мы можем применять простые и составные типы данных. Для начала следует разобраться с *простыми типами*, которые полностью унаследованы из XML Schema.

- Тип `boolean` предназначен для реализации логических значений. По сути является перечислимым типом. В качестве допустимых значений могут использоваться только ключевые слова `true` и `false`, первое из которых обозначает истинность утверждения, второе — ложность.
- Тип `float` предназначен для представления численных данных. Для хранения каждой численной величины этого типа отводится тридцать два бита, т. е. четыре байта. При этом значение представляется в виде произведения целого числа, чье абсолютное значение меньше или равно 16777216, и экспоненты, показатель степени которой может находиться в пределах от 104 до −149.
- Тип `double` также предназначен для численных данных. Но для каждого значения этого типа отводится уже шестьдесят четыре бита, т. е. восемь байтов. Соответственно, изменяется и точность представления чисел, и возможный диапазон представимых значений.
- Если предыдущие два типа рассматривали число в экспоненциальной форме, то тип `decimal` для представления численных данных использует десятичные степени. Естественно, в этом случае число представляется в виде обычной десятичной дроби. По умолчанию XML-процессоры распознают и обрабатывают восемнадцать десятичных знаков после запятой.
- Тип `timeDuration` предназначен для хранения данных о времени. Значение этого типа состоит из шести частей. В нем указывается год, месяц, день, часы, минуты и секунды. Вид значений этого типа описывается в стандарте ISO 8601. Порядок используется именно тот, в котором отдельные части были перечислены. Для указания года используются че-

тыре символа, на остальные части значения времени отводится по два символа.

- Тип `recurringDuration` задает период времени, в течение которого будет с определенной частотой происходить или производиться некое действие. По сути, этот тип является набором значений типа `timeDuration`.
- Тип `binary` предоставляет возможность хранить и обрабатывать данные в двоичном виде, т. е. по сути, это реализация любых объектных вставок — графика, звук, видеоклипы и иные бинарные объекты.
- Тип `uriReference` позволяет задавать URI (Universal Resource Identifier) в качестве содержимого какого-либо элемента. Детально структура URI описана в документах RFC 2396 и RFC 2732.
- Тип `integer` предназначен для хранения целочисленных значений. Создан на основе типа `decimal`. Значения этого типа представляют собой конечную последовательность десятичных цифр с необязательным символом математического знака впереди. Если перед положительным числом ставится знак плюса, то он игнорируется.
- Тип `nonPositiveInteger` создан на основе типа `integer`. Этот тип предназначен для хранения неположительных целочисленных данных. Неположительные целочисленные данные объединяют нуль и все целые отрицательные числа.
- На основе вышеуказанного типа создан тип `negativeInteger`. Используется для представления отрицательных целых чисел. Данные этого типа представляют собой конечную последовательность десятичных цифр со знаком минус впереди.
- Тип `long` создан на основе типа `integer`. Он хранит данные не в форме последовательности символов цифр, а как единое численное значение. Естественно, это налагает ограничения на диапазон хранимых данных. Для хранения одного числа отводится восемь байтов. Так как величины данного типа могут быть как отрицательными, так и положительными, границы допустимого диапазона значений установлены от 9 223 372 036 854 775 807 до −9 223 372 036 854 775 808.
- На основе типа `long` построен тип `int`. Он также предназначен для представления целых чисел, но для хранения каждого числа отведено четыре байта. Таким образом, величины данного типа могут находиться в промежутке от 2 147 483 647 до −2 147 483 648. Лексически его значения представляют собой конечную последовательность десятичных цифр с необязательным символом математического знака в начале.
- Если под хранение целого числа отводить не четыре байта, а два — мы получим тип `short`, декларируемый на основе типа `int`. Промежуток допустимых значений для этого типа лежит от 32 767 до −32 768.

- Тему экономии памяти продолжает тип `byte`, созданный на базе типа `short`. Как следует из его названия, под хранение одного целого числа отводится один байт. Следовательно, мы можем использовать только числа от 127 до −128.
- Тип `nonNegativeInteger` основан на типе `integer` и предназначен для хранения неотрицательных целых чисел. По сути, мы просто жестко задаем нулевое значение свойства `minInclusive`. Все остальное остается без изменений.
- На базе указанного типа создан тип `unsignedLong`. Он предназначен для хранения беззнакового целого числа. Максимальное возможное значение свойства `maxInclusive` — 18 446 744 073 709 551 615. Минимальное, естественно — 0.
- Тип `unsignedInt` является производным от типа `unsignedLong`. Разница между ними заключается в количестве байтов, отводимых под хранение одного числа. Тип `unsignedInt` — вдвое экономнее. Данные этого типа лежат в промежутке от нуля до 42 949 672 95 включительно.
- Тип `unsignedShort`, построенный на базе только что рассмотренного нами типа `unsignedInt`, под одно число отводит всего два байта. Но так как мы используем в данном случае беззнаковые числа, то максимально возможное число этого типа — 65 535.
- Последним из типов, предназначенных для хранения целых беззнаковых чисел, является `unsignedByte`. Легко понять, что для хранения отдельного числа этого типа применяется всего один байт. Таким образом, мы можем использовать рассматриваемый тип для чисел в промежутке от 0 до 255 включительно.
- Тип `positiveInteger` предназначен для хранения и представления положительных целых чисел. Он построен на основе типа `nonNegativeInteger`. При хранении данных этого типа опциональный знак плюса и нули, стоящие в начале числа, игнорируются.
- Тип `timeInstant` предназначается для отображения данных, хранящих информацию о времени и дате. Таким образом, если мы используем рассматриваемый тип для хранения точки времени 13 часов 20 минут 31 марта 2000 года в часовом поясе, смещенном на пять часов относительно времени по Гринвичу, отображаться это значение будет так: `2000-03-31T13:20:00-05:00`.
- Тип `time` позволяет хранить данные только о времени, без указания даты. Таким образом, то же самое время 13:20 в искомом часовом поясе будет отображаться так: `13:20:00-05:00`.
- Тип `timePeriod` используется для обработки и отображения неких промежутков времени, для которых явно задано время начала и окончания.

На самом деле данный тип не рекомендуется напрямую использовать в XML-документах, созданных по спецификации XML Schema. Рекомендуется использовать типы данных, построенные на его основе.

- Тип `date` является первым из рассматриваемых нами типов данных, созданных на основе типа `timePeriod`. Он позволяет указывать промежуток времени величиной в одни сутки, который начинается в полночь указанной даты суток и заканчивается в полночь следующих суток. То есть, значение `2000-12-06` указывает не на точку времени, а на временной промежуток, который начинается в полночь шестого декабря двухтысячного года и длится до полуночи седьмого декабря двухтысячного года. В спецификации XML Schema сказано, что данный тип предназначен для обработки суточного интервала времени "вне зависимости от количества часов в этом дне". В качестве данных этого типа могут использоваться только даты стандартного Григорианского календаря в виде `YYYY-MM-DD`. Если необходимо использовать дату до нашей эры, то перед годом ставится знак минуса.
- Тип `month` основан на типе `timePeriod`. Он предназначен для определения промежутков времени длиною в месяц. Этот промежуток начинается в полночь первого дня указанного месяца и заканчивается в полночь первого дня следующего месяца, не включая ее. Данные этого типа имеют вид строк формата `YYYY-MM`. Таким образом, чтобы обозначить декабрь двухтысячного года, мы должны использовать строку `2000-12`.
- Тип `year` используется для хранения годичных промежутков времени. Данные этого типа указываются в виде строк формата `YYYY`. Искомый промежуток времени начинается в полночь первого дня указанного года, а заканчивается в полночь первого дня следующего года, не включая ее. Таким образом, весь прошедший двухтысячный год мы можем обозначить строкой `2000`. Если необходимо указать годы до нашей эры, то перед номером этого года, в котором, кстати, может быть больше четырех цифр, следует поставить знак минуса.
- Тип `century` предназначен для обработки столетий. При этом значения данного типа представляются в виде двух цифр, являющихся обозначением века в нумерации годов. Таким образом, если мы хотим задать двадцатый век, то мы должны использовать значение `19`. Данный промежуток времени начинается в полночь первого дня указанного столетия, а заканчивается в полночь первого дня следующего века, не включая ее.
- Тип `recurringDate` предназначается для обработки неких ежегодно повторяющихся дат. Значениями данного типа являются строки формата `MM-DD`. То есть, если мы задаем значение `12-06`, то указываем ежегодно повторяющуюся дату — шестое декабря. Естественно, при помощи величин этого типа чрезвычайно удобно задавать самые различные годовщины, такие, как дни рождения и т. п.

- Тип `recurringDay` позволяет задавать ежемесячно повторяющиеся дни. Мы указываем только число, а затем оно будет автоматически обрабатываться каждый месяц. Этот тип генерируется на основе типа `recurringDuration`. Данные этого типа записываются в виде строк из двух символов, которые обозначают число повторяющегося дня.

Теперь перейдем к рассмотрению *сложных типов* данных. К ним относятся массивы, перечислимые списки и составные типы данных. Начнем мы с ознакомления с перечислимыми списками.

Перечислимые списки функционально весьма похожи на Enumeration-классы, которые используются в Visual Basic .NET. В этих списках могут храниться значения любых типов, кроме типа `Boolean`. Естественно, элементы списка должны иметь один и тот же тип. В качестве примера можно привести хранение в подобном списке цветов продаваемых машин. Тогда в SOAP-сообщении будет находиться приблизительно следующий фрагмент кода:

```
<element name="CarColor" type="tns:CarColor" />
  <simpleType name="CarColor" base="xsd:String">
    <enumeration value="Red" />
    <enumeration value="White" />
    <enumeration value="Blue" />
  </simpleType>
<Car>
  <Model>Ferrary</Model>
  <CarColor>Red</CarColor>
</Car>
```

В этом примере мы объявляем перечислимый тип `CarColor`, который создается на основе базового типа `String`. Затем при помощи тегов `<Enumeration>` задаются конкретные значения, входящие в перечислимый список. Естественно, эти теги заключаются между стартовым и закрывающим тегами `<SimpleType>`. А затем уже при указании данных об объекте `Car` мы используем одно из предустановленных значений заранее объявленного перечислимого списка.

Теперь перейдем к *составным типам*. Теоретически, массивы в SOAP тоже относятся к составным типам. Но разработчикам все-таки привычнее под составными типами понимать именно определяемые структуры. Если говорить строго, то структура является составным значением, элементы которого идентифицируются только их наименованиями, причем, эти наименования должны быть уникальны для каждого элемента структуры. По сути, структуры напоминают классы, но их нельзя назвать полноценными классами, так как они не могут включать в себя методы и свойства. Когда мы в предыдущем примере указывали информацию о конкретной машине, то неявно подразумевалось, что объект `Car` как раз и попадает под определение со-

ставного типа. В том случае, когда приложение, принимающее SOAP-сообщение, заранее знает его структуру, нет смысла описывать все составные типы данных. Однако так может быть далеко не всегда, поэтому если в цепочке получателей SOAP-сообщения есть приложение, которое не извещено о структуре сообщения, следует объявлять используемые составные типы.

Но перейдем к примеру определения составного типа данных. Попробуем создать структуру, которая с достаточно малой степенью детализации описывает, скажем, книги, продаваемые неким магазином. На самом деле при продаже книг в корпоративных базах данных хранится достаточно много информации, но сейчас мы ограничимся малой ее долей в пользу более легкой читаемости кода примера.

```
<element name="Book">
  <complexType>
    <element name="author" type="xsd:string">
    <element name="name" type="xsd:string">
    <element name="price" type="xsd:float">
  </complexType>
<element/>
<Book>
  <author>Гибсон У.</author>
  <name>Нейромантик</name>
  <price>58.75</price>
</Book>
```

Легко заметить, что объявление структуры составного типа производится между тегами `<complexType>` и `</complexType>`. Мы задаем два элемента строчного типа, в которых хранятся наименование книги и имя ее автора, и один элемент числового типа, в котором будет храниться цена продаваемой книги. После объявления составного типа располагается сам экземпляр значения этого типа. В SOAP-сообщении передается информация о книге Уильяма Гибсона "Нейромантик", которая продается по цене 58.75.

Теперь обратимся к использованию *массивов*. Массив тоже является составным типом, элементы которого идентифицируются только своим порядковым номером. В качестве элементов массива могут использоваться не только значения простых типов, но также структуры и другие массивы.

Любой массив в SOAP должен объявляться при помощи типа `SOAP-ENC:Array`. Вот простейший пример объявления массива из трех элементов целочисленного типа:

```
<element name="MyArray" type="SOAP-ENC:Array" />
  <MyArray SOAP-ENC:arrayType="xsd:int[3]">
```

```
    <item>1</item>
    <item>2</item>
    <item>3</item>
  </MyArray>
```

Естественно, SOAP позволяет определять и многомерные массивы данных. Объявление массива размером "два на три", состоящего из элементов строкового типа, может выглядеть следующим образом:

```
<MyStringArray SOAP-ENC:arrayType="xsd:String[2,3]">
```

Таким образом, мы разобрали все типы данных, которые применяются в рамках создания SOAP-сообщений.

Глава 5

SOAP-клиенты

Первый клиент

В *главе 2* мы уже создали один Web-сервис и даже попробовали осуществлять доступ к нему при помощи стандартных методов `GET` и `POST` протокола HTTP. При этом в качестве клиентского приложения использовался стандартный браузер. Это, конечно, удобно для разработчика, однако доступ к функциональности достаточно сложного сервиса и правильное отображение полученной информации при подобном методе работы несколько затруднены. Действительно, если мы построим на базе Web-сервиса, скажем, сеть обмена файлами, то использование браузера в качестве клиента явно не приведет к ее популярности.

Действительно, для того чтобы воспользоваться всеми возможностями серьезного сервиса и предоставить пользователю правильно отображенную информацию, придется создать отдельное приложение, которое и будет выступать в роли клиента. Именно этим мы и займемся в этой главе.

Так как все общение с Web-сервисом происходит по протоколу HTTP, то разработчик никак не ограничен в средствах разработки клиентского приложения. Более того, выбор компьютерной платформы и операционной системы для клиентского приложения тоже является прерогативой разработчика. Он может выбрать любую операционную систему и любую платформу, которые поддерживают соединение с использованием протокола HTTP.

Один раз мы уже использовали для разработки Web-сервиса средство Visual Studio .NET. Остановимся на нем же и для разработки клиентского приложения. Поддержка XML является одной из наиболее часто упоминаемых возможностей Visual Basic .NET, поэтому нам не составит труда разработать клиент для Web-сервиса.

Мы можем, конечно, пойти по сложному пути, в клиентском приложении самостоятельно устанавливая соединение с Web-сервисом, подготавливая SOAP-запрос, получая и разбирая ответ сервиса. Но для первого клиентского приложения это будет излишне серьезной задачей. Зачем создавать

себе подобные сложности, если среда разработки Visual Studio .NET позволяет все сделать намного проще?

Итак, для создания своего первого обособленного клиентского приложения придется выполнить несколько несложных действий. Для начала необходимо создать новый проект. Для этого следует выполнить команду меню **File | New | Project**. В появившемся диалоговом окне **New Project** (рис. 5.1) в блоке шаблонов для проектов на Visual Basic выбрать шаблон **Windows Application** и в поле **Name** указать наименование создаваемого приложения. Назовем его `myclient`.

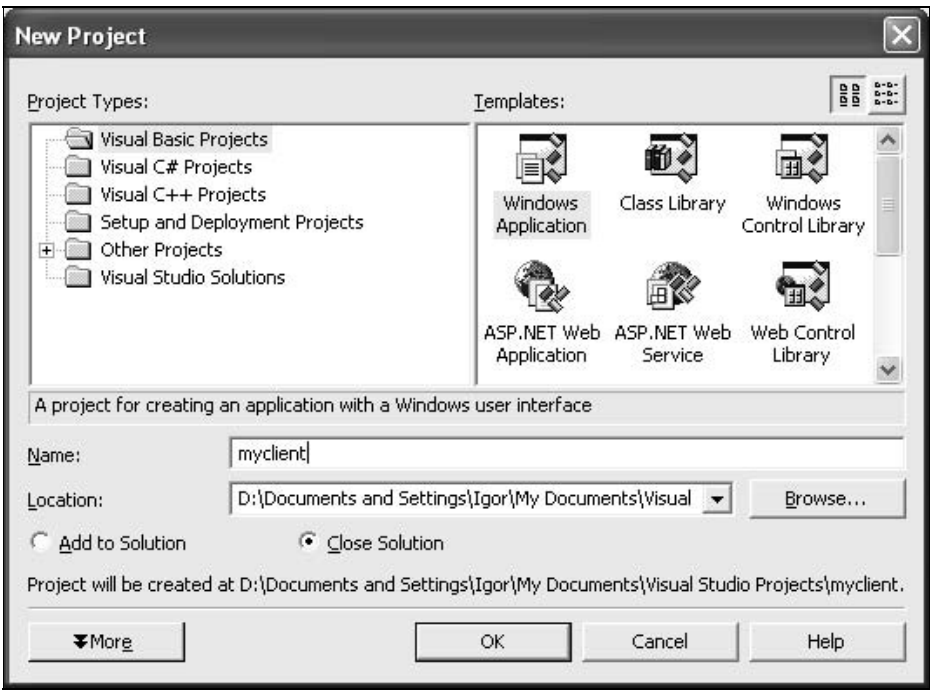

Рис. 5.1. Диалоговое окно **New Project**

После этого Visual Studio .NET создаст форму с именем `Form1`, устанавливаемым по умолчанию. В окне этой формы нам потребуется разместить один элемент `Checkbox`, одно текстовое поле `Label`, и одну кнопку `Button`. При помощи независимого переключателя `CheckBox1` мы будем задавать параметр, передаваемый функции `MyDate`, входящей в состав сервиса `MyService`. Текстовое поле будет содержать результат, возвращаемый этой функцией, а кнопка — запускать процесс установки связи с Web-сервисом и получения от него требуемых данных. Общий вид готовой формы в среде разработки Visual Studio .NET показан на рис. 5.2.

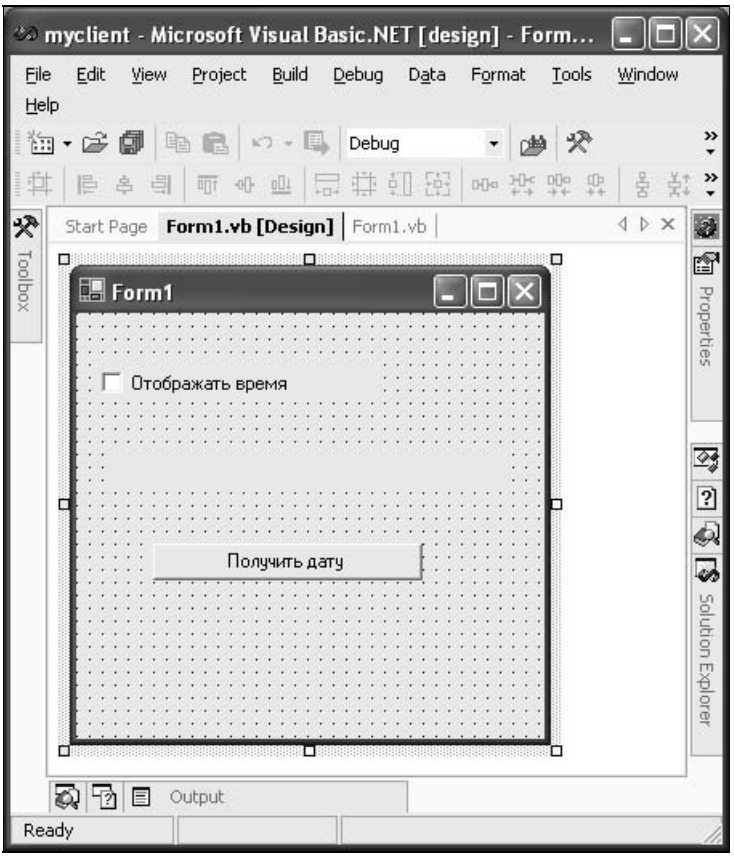

Рис. 5.2. Внешний вид спроектированной формы клиентского приложения в среде разработки Visual Studio .NET

После того как все необходимые компоненты будут отбуксированы с панели инструментов **Toolbox** на разрабатываемую форму, следует установить связь с искомым Web-сервисом. Для этого надо в окне **Solution Explorer** выделить элемент `myclient` и правым щелчком мыши вызвать для него контекстное меню. В этом контекстном меню необходимо выполнить команду **Add Web Reference**, после чего будет активизировано одноименное диалоговое окно, позволяющее установить ссылки на используемые сервисы. Это диалоговое окно унаследовало основную функциональность от стандартного браузера Microsoft Internet Explorer. Для того чтобы получить ссылки на функции Web-сервиса и перенести их в свой проект, нужно в текстовом поле **Address** указать URL необходимого сервиса и загрузить ресурс, располагающийся по этому адресу. После этого в левой части искомого окна **Add Web Reference** будет отображено содержимое стартовой Web-страницы сервера, которую мы уже могли наблюдать при доступе к сервису при помощи

браузера, а в правой части диалогового окна будет показана ссылка на единственную функцию этого сервиса. Внешний вид такого диалогового окна показан на рис. 5.3.

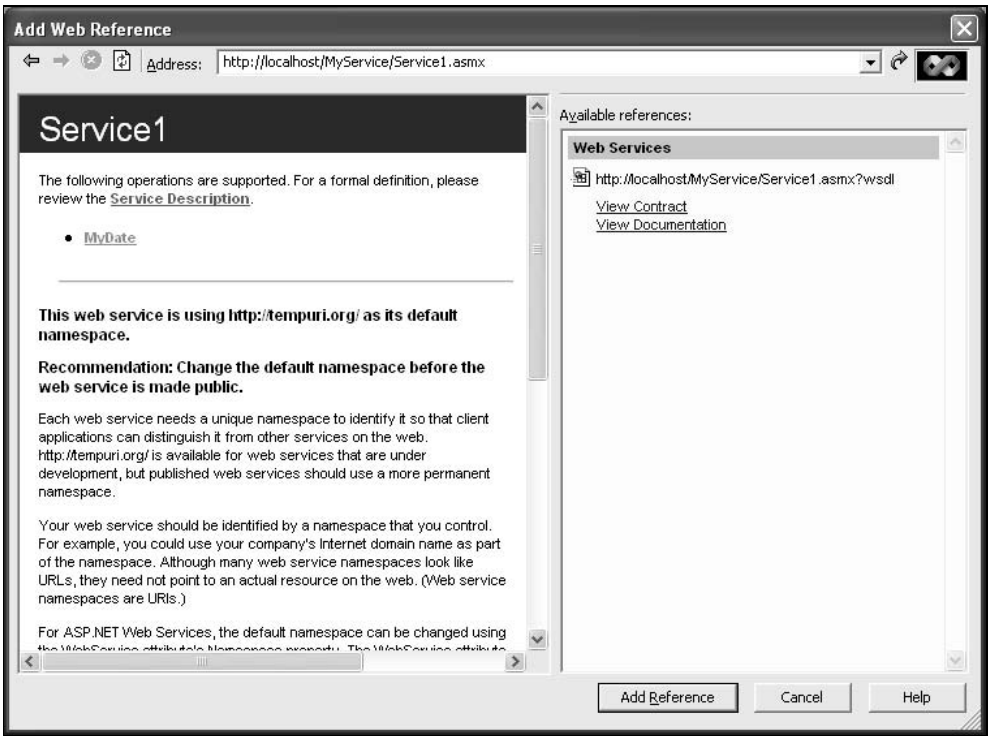

Рис. 5.3. Диалоговое окно **Add Web Reference**

После того как искомые ссылки на функции Web-сервиса будут найдены, при помощи кнопки **Add Reference** их необходимо добавить к разрабатываемому проекту.

Итак, наш проект теперь имеет доступ к функциональности Web-сервиса, причем доступ, практически, прямой. Как осуществляется взаимодействие с Web-сервисом, видно из приведенного кода разрабатываемого клиентского приложения (листинг 5.1).

Листинг 5.1

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
```

```
    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents CheckBox1 As System.Windows.Forms.CheckBox

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.Container

    'NOTE: The following procedure is required by the
Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.CheckBox1 = New System.Windows.Forms.CheckBox()
        Me.SuspendLayout()
        '
        'Label1
        '
```

```
        Me.Label1.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        Me.Label1.Location = New System.Drawing.Point(24, 88)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(248, 23)
        Me.Label1.TabIndex = 0
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(48, 144)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(168, 23)
        Me.Button1.TabIndex = 1
        Me.Button1.Text = "Получить дату"
        '
        'CheckBox1
        '
        Me.CheckBox1.Location = New System.Drawing.Point(16, 32)
        Me.CheckBox1.Name = "CheckBox1"
        Me.CheckBox1.Size = New System.Drawing.Size(176, 24)
        Me.CheckBox1.TabIndex = 2
        Me.CheckBox1.Text = "Отображать время"
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 266)
        Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.CheckBox1, Me.Button1, Me.Label1})
        Me.Name = "Form1"
        Me.Text = "Клиентское приложение"
        Me.ResumeLayout(False)

    End Sub

#End Region

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim t1 = New myclient.localhost.Service1()
```

```
        Label1.Text = t1.MyDate(CheckBox1.Checked)
    End Sub
End Class
```

Сам исполняемый код, вводимый разработчиком, находится в функции `Button1_Click`, которая является обработчиком нажатия на кнопку. Этот фрагмент кода достаточно прост и нагляден:

```
Dim t1 = New myclient.localhost.Service1()
Label1.Text = t1.MyDate(CheckBox1.Checked)
```

В первой строке мы объявляем переменную `t1`, которая является экземпляром класса `Service1`. Полное наименование этого класса, как видно из фрагмента кода, — `myclient.localhost.Service1`. Здесь использование ссылки на сервис добавляет функциональность этого сервиса как класс в состав основного класса приложения. При этом, естественно, указывается и адрес Web-сервера, на котором располагается искомый сервис. В нашем случае, так как мы оперируем локальным Web-сервером, используется его внутреннее имя `localhost`.

После того как был объявлен экземпляр класса, предоставляющего доступ к Web-сервису, осталось лишь использовать его по назначению. Для этого вызывается функция `MyDate`, входящая в состав сервиса. В качестве параметра этой функции передается состояние независимого переключателя. Если флажок в нем взведен, то передается значение логического типа `True`, которое указывает, что помимо даты, клиентское приложение рассчитывает получить еще и текущее время. Если флажок в переключателе не взведен, то, естественно, передается значение `False`.

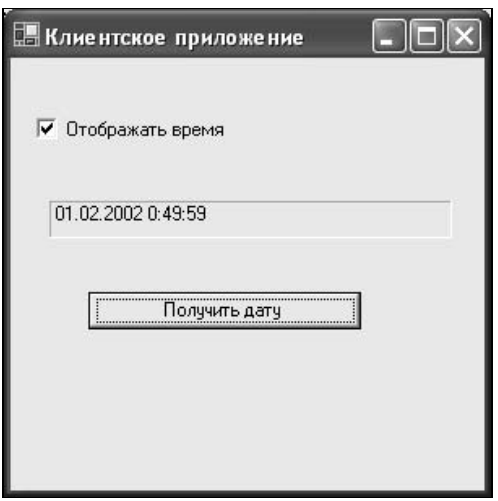

Рис. 5.4. Внешний вид клиентского приложения

Так как функция возвращает значение типа `String`, возвращаемое значение можно сразу отобразить в компоненте `Label`. Именно эта операция и производится во второй строке приведенного фрагмента кода.

После того, как будет создан код, приложение можно компилировать и запускать. Внешний вид нашего первого специализированного клиентского приложения показан на рис. 5.4.

Итак, мы создали свое первое клиентское приложение, которое позволяет получать доступ к функциональности Web-сервиса. Причем, сама процедура разработки не составила особого труда. Действительно, следует признать, что средство разработки Visual Basic .NET позволяет создавать сервисы и клиентские приложения достаточно просто, сосредоточиваясь на разработке именно функциональности, а не на вопросах связи и приема/передачи данных.

Клиент на основе ASP.NET

В жизни разработчика вполне может возникнуть ситуация, когда доступ к Web-сервисам будет необходимо предоставить с Web-страницы. Неужели придется довольствоваться стандартными средствами, описанными в *главе 2*? Следует отметить, что выход есть.

Для того чтобы предоставить удобный и полнофункциональный доступ к Web-сервису, следует воспользоваться возможностями технологии ASP.NET. Перед тем как мы перейдем к рассмотрению примера создания подобной Web-страницы, следует сказать несколько слов об ASP.NET. Конечно, разработчик, собирающийся работать с Web-сервисами Microsoft .NET, скорее всего, что такое ASP.NET, знает. Но для тех, кто не знаком с этой технологией, следует сделать небольшой ее обзор.

Давно прошли те времена, когда сайт был просто набором статических Web-страниц. Теперь этого мало. Сейчас сайты все чаще создают с использованием новейших технологий программирования. Большая часть страниц генерируется автоматически, используя информацию, получаемую из каких-либо баз данных. Естественно, для создания таких страниц одного языка HTML, который все-таки является основой для Web-страниц, явно недостаточно. Даже использование технологии динамического HTML, опирающегося на программы-скрипты, выполняющиеся на стороне пользователя, не поможет в создании подобных сайтов.

Понятно, что работа приложений, динамически формирующих самые разнообразные Web-страницы по запросу пользователя, может происходить только на самом сервере, который поддерживает функционирование искомого Web-сайта. А если в процессе обработки запросов удаленного пользователя приходится обращаться к содержимому базы данных, то здесь уж никак не обойтись без серверных приложений, так как база данных, естественно, может функционировать только на сервере.

Вообще-то, для создания приложений, действующих на стороне Web-сервера, есть достаточно много приемлемых технологий. Но технология ASP (Active Server Pages) стоит в этом ряду особняком. Дело в том, что практически все технологии серверных приложений реализованы в виде дополнительных модулей, подключаемых к Web-серверам. А программы ASP не нуждаются в дополнительных интерпретирующих модулях. Поддержка ASP изначально встроена в Web-сервер IIS (Internet Information Services). Этот достаточно мощный сервер в последнее время автоматически включается в состав операционных систем Windows 2000 и Windows NT.

Таким образом, если вы привыкли работать с продуктами корпорации Microsoft, и перед вами стоит задача создать полнофункциональный Web-сайт, то выбор, пожалуй, очевиден. Следует использовать связку IIS+ASP. Тем более, что ASP-приложения отлично интегрируются с базами данных от Microsoft.

Изначально скрипты ASP создавались при помощи языка Visual Basic. Однако концепция MSIL (Microsoft Intermediate Language), являющаяся частью Microsoft .NET, позволяет писать ASP-приложения на любом языке, поддерживаемом средой разработки Visual Basic .NET. В ее состав, как мы знаем, входят языки Visual Basic .NET, Visual C++ .NET и Visual C# .NET. Таким образом, приложения ASP можно разрабатывать на любом из этих языков.

До появления технологии Microsoft .NET ASP-скрипты приходилось интегрировать с HTML-документами практически вручную. То есть, Web-страницы изначально были отделены от скриптов, реализующих "движок" сайта. Средство разработки Visual Basic .NET позволяет создавать Web-страницы, полностью интегрированные с ASP.NET-кодом. Теперь разработчик может конструировать их точно так же, как и обычные программы (некоторые отличия, несомненно, все равно будут), отбуксировывая на Web-страницу необходимые компоненты и органы управления и создавая исходный код для обработчиков тех или иных событий. Всю остальную "дизайнерскую" работу возьмет на себя Visual Basic .NET, генерируя код, реализующий функциональность разработанной Web-страницы. А после публикации этой Web-страницы на сайте основную работу возьмет на себя WWW-сервер IIS, который и будет обеспечивать функционирование ASP.NET-кода, связанного с данной страницей.

Итак, приступим к разработке Web-страницы, которая сможет выступить в роли клиентского приложения, связывающегося с Web-сервисом при помощи языка SOAP, а не стандартными методами `GET` и `POST` протокола HTTP. Для этого, как и прежде, выполним команду меню **File | New | Project**, и в диалоговом окне **New Project** выберем шаблон `ASP.NET Web Application`. Затем в поле **Name** укажем наименование создаваемого проекта, и Visual Studio .NET создаст новый проект `ASP.NET`.

Вместо формы, как это делается при разработке обычного приложения, среда разработки Visual Studio .NET создаст заготовку для Web-страницы с именем `WebForm1.aspx`, задаваемым по умолчанию. Его, впрочем, всегда можно изменить. В нашем примере мы зададим для Web-страницы имя `Client.aspx`.

Как и в прошлом примере, на Web-странице следует разместить компоненты `Label`, `CheckBox` и `Button`, отбуксировав их с вкладки **Web Forms**, входящей в состав инструментальной панели **Toolbox**. Затем надо добавить к проекту ссылку на Web-сервис точно так же, как это делалось в предыдущем разделе главы при создании отдельного клиентского приложения. То есть, открыть окно **Solution Explorer** и в контекстном меню приложения выполнить команду **Add Web Reference**. Все, теперь мы готовы к созданию Web-страницы, которая сможет функционировать в качестве клиента для Web-сервиса. Однако перед тем, как приступать к написанию кода, следует обратить внимание, что разрабатываемый нами проект разбит на два слоя. Существует HTML-код для Web-страницы, порождаемый Visual Basic .NET, и код класса, который поддерживает функциональность этой Web-страницы. При этом HTML-код отвечает за внешний вид страницы и ее логическую структуру, а исполняемый код, для написания которого мы использовали Visual Basic .NET, ответственен за описание функциональности страницы. Начнем мы с рассмотрения HTML-кода.

Как уже говорилось, этот код формируется Visual Basic .NET автоматически после того, как разработчик отбуксирует необходимые компоненты на создаваемую страницу и, тем самым, задаст ее внешний вид. Этот код приведен в листинге 5.2.

Листинг 5.2

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="Client.aspx.vb" Inherits="WebClient.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <meta name="GENERATOR" content="Microsoft Visual Basic .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
```

```
   <asp:CheckBox id="CheckBox1" style="Z-INDEX: 101; LEFT: 34px;
POSITION: absolute; TOP: 24px" runat="server"></asp:CheckBox>
   <asp:Label id="Label1" style="Z-INDEX: 102; LEFT: 136px; POSITION:
absolute; TOP: 25px" runat="server">Отображать время?</asp:Label>
   <asp:Button id="Button1" style="Z-INDEX: 103; LEFT: 38px; POSITION:
absolute; TOP: 65px" runat="server" Text="Получить дату" Width="148px"
Height="24px"></asp:Button>
   <asp:Label id="Label2" style="Z-INDEX: 104; LEFT: 45px; POSITION:
absolute; TOP: 118px" runat="server" Width="220px"
Height="18px">Сегодняшняя дата</asp:Label>
  </form>
 </body>
</HTML>
```

Теперь перейдем к рассмотрению кода класса, реализующего функциональность Web-страницы. Этот код приведен в листинге 5.3.

Листинг 5.3

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Label2 As System.Web.UI.WebControls.Label
    Protected WithEvents CheckBox1 As System.Web.UI.WebControls.CheckBox


#Region " Web Form Designer Generated Code "


    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()


    End Sub


    Private Sub Page_Init(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub


#End Region
```

```
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        Dim t1 = New WebClient.localhost.Service1()
        Label2.Text = t1.MyDate(False)
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim t1 = New WebClient.localhost.Service1()
        Label2.Text = t1.MyDate(CheckBox1.Checked)
    End Sub
End Class
```

Рассмотрим приведенный код. В нем присутствуют два обработчика событий. Один носит наименование `Page_Load` и выполняется сразу после загрузки страницы, а второй называется `Button1_Click` и выполняется после того, как пользователь нажмет соответствующую кнопку, расположенную на Web-странице. Легко заметить, что код этих двух функций практически идентичен. Более того, принцип его создания мы уже рассматривали в предыдущем разделе. Это стандартные методы доступа к функциональности Web-сервиса и отображения полученного результата. Следует обратить внимание на тот факт, что приемы отображения информации на Web-страницах, созданных при помощи технологии ASP.NET, и использования органов управления, размещенных на этих Web-страницах, практически идентичны разработке обычного приложения.

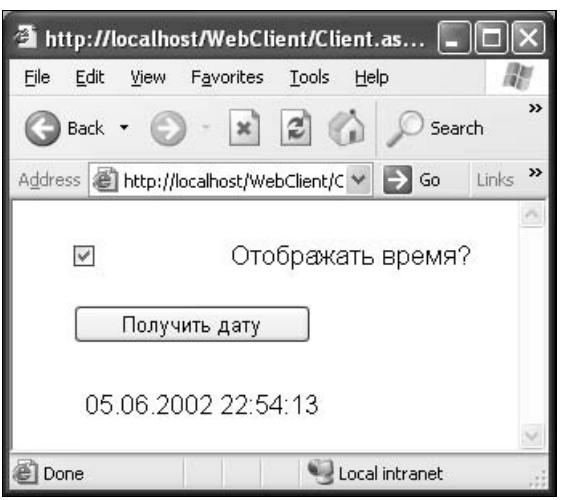

Рис. 5.5. Внешний вид Web-страницы, созданной с использованием технологии ASP.NET

Следует лишь оговориться, что при загрузке страницы мы передаем функции сервиса параметр `False` и сразу отображаем полученную дату на странице, а при нажатии пользователем кнопки — учитываем состояние независимого переключателя.

На рис. 5.5 показан внешний вид разработанной клиентской Web-страницы после того, как пользователь нажал кнопку **Получить дату**.

Естественно, конечный HTML-код этой страницы отличается от того, который был создан Visual Basic .NET. Дело в том, что код, представленный в листинге 5.2, сначала обрабатывается WWW-сервером IIS, формирующим для отправки браузеру удаленного пользователя чистый HTML, который этим браузером и отображается. В листинге 5.4 приведен HTML-код клиентской Web-страницы в том виде, как он был получен браузером.

Листинг 5.4

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <meta name="GENERATOR" content="Microsoft Visual Basic .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form name="Form1" method="post" action="Client.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTU4MDM2NDQ0NDt0PDtsPGk8MT47PjtsPHQ8O2w8aTw3Pjs+O2w8dDxwPHA8bD
xUZXh0Oz47bDwwMi4wMi4yMDAxOz4+Oz47Oz47Pj47Pj47bDxDaGVja0JveDE7Pj4=" />


   <span style="Z-INDEX: 101; LEFT: 34px; POSITION: absolute; TOP:
24px"><input id="CheckBox1" type="checkbox" name="CheckBox1" /></span>
   <span id="Label1" style="Z-INDEX: 102; LEFT: 136px; POSITION:
absolute; TOP: 25px">Отображать время?</span>
   <input type="submit" name="Button1" value="Получить дату" id="Button1"
style="height:24px;width:148px;Z-INDEX: 103; LEFT: 38px; POSITION:
absolute; TOP: 65px" />
   <span id="Label2" style="height:18px;width:220px;Z-INDEX: 104; LEFT:
45px; POSITION: absolute; TOP: 118px">02.02.2001</span>
  </form>
</body>
</HTML>
```

Итак, в этом разделе мы научились создавать Web-страницы, которые могут выступать в роли клиента для Web-сервисов. При этом мы не применяли стандартные методы `GET` и `POST` протокола HTTP. Следует признать, что при использовании Visual Studio .NET разработка как самих сервисов, так и разнообразных клиентов для них не представляет особого труда. Конечно, мы взяли достаточно простые примеры, но необходимо осознавать, что среда разработки Visual Studio .NET избавила нас от необходимости самостоятельно программировать установку соединения с сервисом, отправку запросов, получение информации, ее распознавание и обработку. Впрочем, в следующем разделе главы мы узнаем, как создавать подобные клиентские приложения.

Прокси-классы

Visual Basic .NET позволяет создавать клиентские приложения, которые самостоятельно формируют и разбирают SOAP-сообщения, используемые для общения с Web-сервисом. И все же, если это единственный способ разработки клиентов, то становится непонятно, почему вокруг этих Web-сервисов был поднят такой ажиотаж, ведь подобные сервисы можно создавать и с помощью иных средств разработки, например, Delphi. В чем же секрет?

Дело в том, что в случае разработки клиентских приложений при помощи Visual Basic .NET, программист может просто импортировать классы, используемые Web-сервисом. Эти классы и называются *прокси-классами*. В первых разделах этой главы при разработке клиентов мы присоединяли к проекту ссылку на используемый Web-сервис. Вот именно при помощи этой ссылки в клиентское приложение импортируется прокси-класс Web-сервиса.

Впрочем, разработчик может генерировать прокси-классы самостоятельно, без помощи среды разработки Visual Studio .NET. Как мы уже говорили, эта среда, в принципе, не является необходимым ресурсом для разработки приложений, функционирующих в среде .NET. Вполне можно обойтись обычным блокнотом и некоторыми инструментальными средствами, входящими в состав Framework .NET. В этом случае для разработчика доступ к Web-сервису будет похож на работу с удаленным COM-объектом. Это и показано в следующем примере.

Для получения прокси-класса следует использовать утилиту `wsdl`, входящую в состав Framework .NET. Вызов ее выполняется из командной строки. Для получения прокси-класса, соответствующего нашему Web-сервису, следует выполнить следующую команду:

```
Wsdl /l:vb http://localhost/MailServ/Service1.asmx?WSDL
```

Результат выполнения этой команды в окне командной строки показан на рис. 5.6.

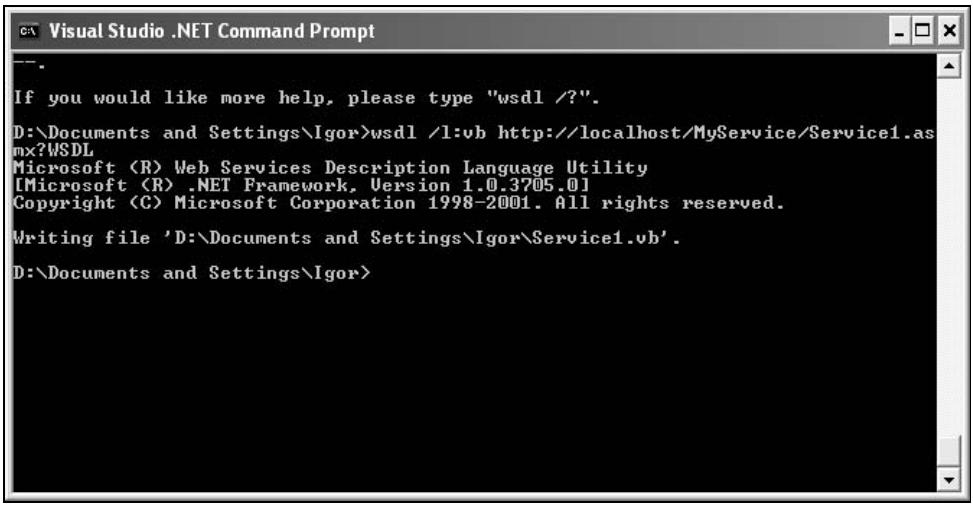

Рис. 5.6. Окно командной строки с результатом выполнения утилиты wsdl

Как можно видеть, утилита создала некий код и записала его в файл, указав его расположение на диске. В этом файле и находится полный код прокси-класса, приведенный в листинге 5.5.

Листинг 5.5

```
'------------------------------------------------------------------------------
' <autogenerated>
'     This code was generated by a tool.
'     Runtime Version: 1.0.3705.0
'
'     Changes to this file may cause incorrect behavior and will be lost
if
'     the code is regenerated.
' </autogenerated>
'------------------------------------------------------------------------------

Option Strict Off
Option Explicit On

Imports System
Imports System.ComponentModel
Imports System.Diagnostics
```

```
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization

'
'This source code was auto-generated by wsdl, Version=1.0.3705.0.
'

'<remarks/>
<System.Diagnostics.DebuggerStepThroughAttribute(),  _
 System.ComponentModel.DesignerCategoryAttribute("code"),  _
 System.Web.Services.WebServiceBindingAttribute(Name:="Service1Soap",
[Namespace]:="http://tempuri.org/")>  _
Public Class Service1
    Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

    '<remarks/>
    Public Sub New()
        MyBase.New
        Me.Url = "http://localhost/MyService/Service1.asmx"
    End Sub

    '<remarks/>
    <System.Web.Services.Protocols.SoapDocumentMethodAttribute
("http://tempuri.org/MyDate", RequestNamespace:="http://tempuri.org/",
ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>
_
    Public Function MyDate(ByVal ShowTime As Boolean) As String
        Dim results() As Object = Me.Invoke("MyDate", New Object()
{ShowTime})
        Return CType(results(0),String)
    End Function

    '<remarks/>
    Public Function BeginMyDate(ByVal ShowTime As Boolean, ByVal callback
As System.AsyncCallback, ByVal asyncState As Object)
As System.IAsyncResult
        Return Me.BeginInvoke("MyDate", New Object() {ShowTime},
callback, asyncState)
```

```
    End Function
      '<remarks/>
    Public Function EndMyDate(ByVal asyncResult As System.IAsyncResult)
As String
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),String)
    End Function
End Class
```

Используя функции, объявленные в этом файле, разработчик может получить доступ к функциональности Web-сервиса напрямую, без разбора SOAP-сообщений. Впрочем, все это тоже не слишком уж легко и не может сравниться по степени удобства с созданием клиентских приложений в среде разработки Visual Studio .NET, где разрешено использовать Web-ссылки на сервисы, позволяющие скрыть от разработчика и этот уровень абстракции.

Следует также отметить, что в приведенном файле объявлены два прокси-класса — синхронный и асинхронный. Синхронный прокси-класс при обращении к функции Web-сервиса блокирует интерфейс пользователя до тех пор, пока не будет получен результат функции. Естественно, это занимает некоторое время, так как Интернет — не то место, где следует ожидать мгновенного отклика. При использовании же асинхронных прокси-классов интерфейс пользователя не блокируется во время ожидания получения результата функции Web-сервиса.

Глава 6

Работа с базами данных

ADO.NET

Наш первый пример Web-сервиса был весьма прост, и для его создания не потребовалось писать достаточно сложный код. Однако в реальной жизни создание Web-сервисов для обслуживания потребностей заказчика не будет настолько легким. Можно с уверенностью сказать, что достаточно большая часть Web-сервисов должна будет взаимодействовать с различными базами данных.

В состав технологии Microsoft .NET входит технология, носящая наименование ADO.NET. Она является наследником технологии ADO (ActiveX Data Object). Если в предыдущей версии ADO упор делался на создание постоянных соединений, то в ADO.NET, ориентированной, как это видно из названия, на работу в сетях, по умолчанию создаются соединения, не поддерживаемые все время в подключенном состоянии.

Одним из объектов ADO.NET, наиболее близким к компонентам **Web Forms**, является `DataSet`. Объект `DataSet` может хранить в себе не только набор записей, а и различные таблицы, связи их друг с другом, отдельные поля и прочую атрибутику баз данных. При этом `DataSet` не взаимодействует напрямую с источником данных. Для заполнения объектов DataSet обычно используются объекты промежуточного звена, такие как, например, класс `SqlDataAdapter`.

В примерах, которые мы рассмотрим в данной главе, в качестве основного SQL-сервера будет принят Microsoft SQL Server 2000. Но Web-сервисы, созданные в Microsoft .NET, могут использовать любую базу данных, для которой есть соответствующий ODBC-драйвер.

Здесь следует сделать некоторое отступление. В ADO.NET информацию из баз данных могут предоставлять два типа источников данных — SQL Managed Provider и ADO Managed Provider. Источник данных SQL Managed Provider предоставляет доступ к базам данных, обслуживаемым серверами Microsoft SQL Server версий 7.0 и 2000. Для всех остальных баз данных используется источник данных ADO Managed Provider.

Следует отметить, что источник данных SQL Managed Provider обеспечивает несколько бóльшую скорость работы, нежели его собрат ADO Managed Provider, поэтому, если есть возможность, следует использовать для работы SQL-серверы от Microsoft, начиная с седьмой версии.

Естественно, все объекты для этих двух типов источников данных имеют одинаковые наименования и практически одинаковую функциональность. Различие лишь в том, что объекты, принадлежащие SQL Managed Provider, располагаются в пространствах имен `System.Data` и `System.Data.SQL`, а их функциональные двойники, работающие в сфере ADO Managed Provider, — в пространствах имен `System.Data` и `System.Data.ADO`. Таким образом, если разработчику необходимо обращаться к СУБД, доступ к которой осуществляется при помощи драйвера ODBC, ему следует использовать два последних пространства имен, а сами объекты останутся теми же самыми.

Чаще всего для взаимодействия Web-сервисов с базой данных используются три компонента — `Connection` (соединение), `Command` (команда) и `DataSet` (набор данных). Эта триада практически идеально подходит для Web-приложений, которые логически разделены на несколько слабосвязанных частей, когда нельзя заранее предсказать, какое именно действие произведет пользователь, и какой блок данных ему потребуется. Поэтому модель кратковременных соединений и иерархия "Соединение—Команда—Набор данных" позволяют при наименьших затратах ресурсов добиться максимальной производительности и свободы действий для разработчика.

Впрочем, в этой главе мы будем рассматривать не только извлечение и обработку данных из стандартных баз данных, но и попробуем использовать в качестве источника данных XML-файл. Но обо всем этом — в следующих разделах данной главы.

Постановка задачи

В этой главе все примеры разрабатываемых Web-сервисов будут связаны с некоей базой данных. Предположим, наш заказчик — книготорговая организация, которая собирается предоставить своим покупателям доступ к внутренней базе данных. Из нее клиенты смогут оперативно узнавать, какие книги есть на складе, и в соответствии с этим создавать заказы. В следующих разделах главы мы расширим этот пример до получения B2B-решения, но пока ограничимся обычным доступом к базе для просмотра данных.

Перед тем как подключать к базе данных Web-сервис, эту базу необходимо еще создать. В качестве SQL-сервера выберем SQL Server 2000 от Microsoft. Для создания базы данных воспользуемся стандартной утилитой управления `Enterprise Manager`. Ее основное рабочее окно показано на рис. 6.1.

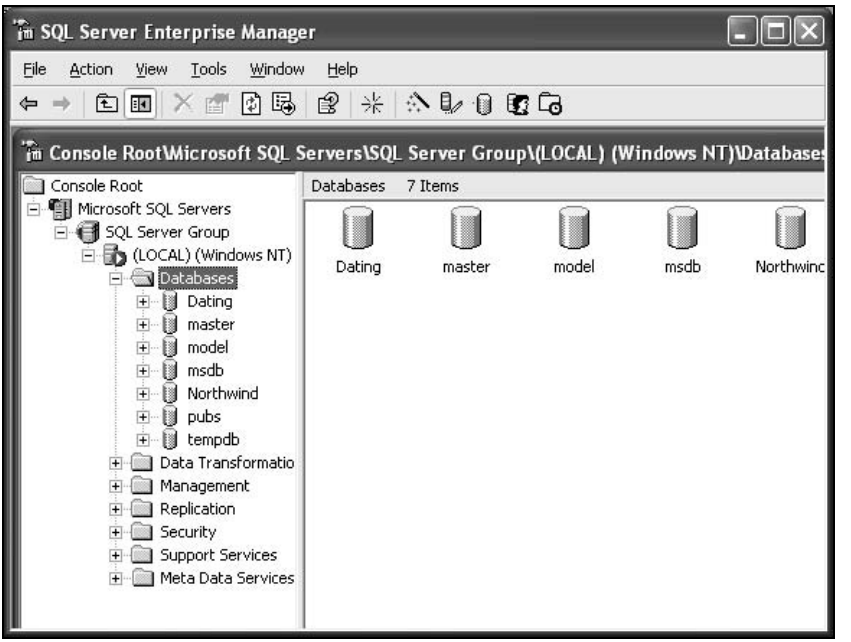

Рис. 6.1. Основное рабочее окно утилиты `Enterprise Manager`

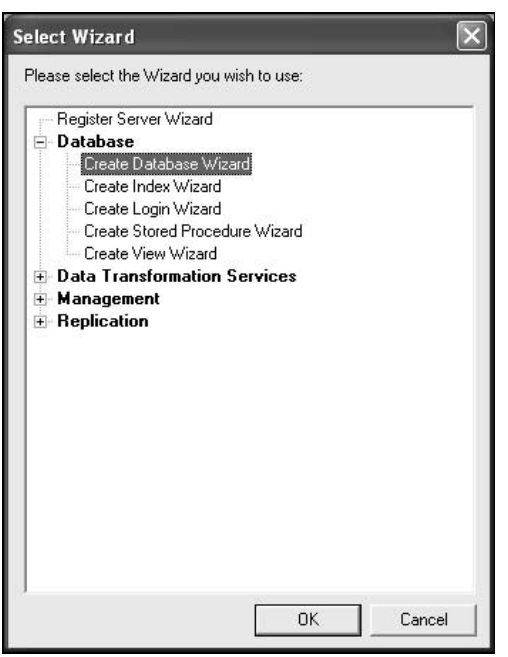

Рис. 6.2. Диалоговое окно **Select Wizard**

Для создания новой базы данных можно активировать список **Databases**, находящийся в левой части окна и являющийся одним из пунктов древовидного списка, вмещающего в себя все компоненты SQL-сервера, и выполнить команду меню **Tools | Wizards** или воспользоваться одноименной кнопкой на стандартной инструментальной панели. При этом будет активировано диалоговое окно **Select Wizard**, показанное на рис. 6.2.

В этом диалоговом окне отображается список всех мастеров, которые доступны для использования в момент вызова окна. Нас интересует мастер **Create Database Wizard**, располагающийся в группе **Database**. После запуска выбранного мастера будет отображено диалоговое окно первого этапа его работы. Для начала необходимо ввести наименование создаваемой базы данных, место расположения ее файлов и LOG-файла всех транзакций в файловой системе сервера. Внешний вид диалогового окна первого этапа работы мастера показан на рис. 6.3.

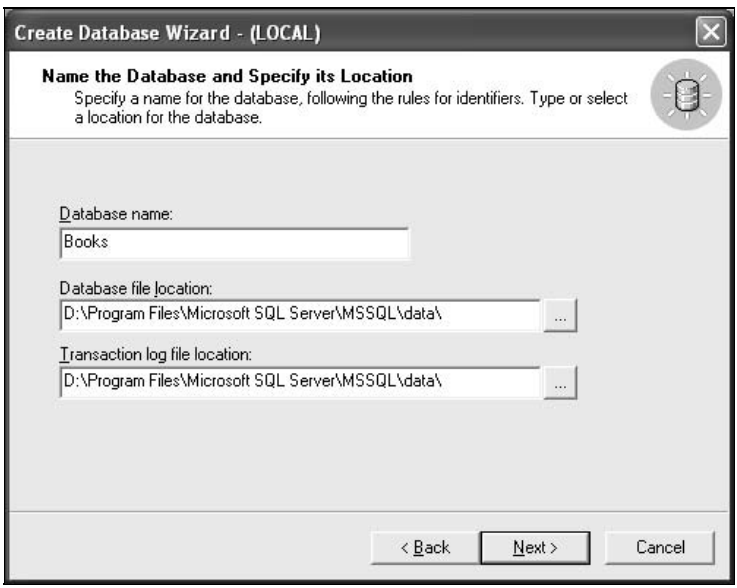

Рис. 6.3. Диалоговое окно первого этапа работы мастера создания новой базы данных

Для новой базы данных мы укажем наименование `Books`, введя его в текстовое поле **Database name**. Расположение файлов каждый может задать самостоятельно, это не должно вызывать затруднений. Стоит только заметить, что на используемом логическом диске должно быть достаточно места для наращивания объема данных.

После того как указано наименование базы данных и расположение требуемых файлов, можно переходить к следующему этапу работы мастера, ис-

пользуя для этого кнопку **Next**. Диалоговое окно второго этапа работы мастера создания новой базы данных показано на рис. 6.4.

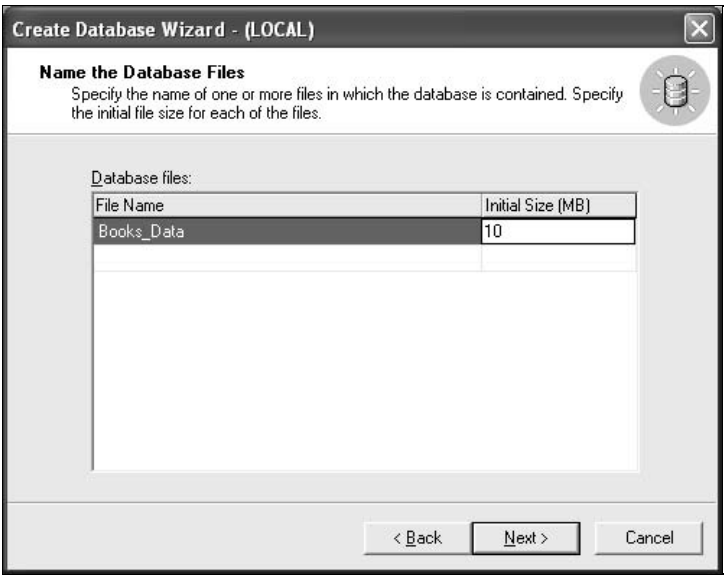

Рис. 6.4. Диалоговое окно второго этапа работы мастера создания новой базы данных

На втором этапе работы мастера следует указать начальные размеры файлов создаваемой базы данных. По умолчанию для основного файла, в котором будут храниться данные, предлагается размер в один мегабайт.

Замечание

Можно, конечно, оставить это размер, ничего страшного не произойдет. Когда размер файла превысит указанный предел, операционная система выделит для него на логическом диске дополнительное пространство. Однако можно с достаточно большой степенью уверенности предположить, что физически место, располагающееся на диске сразу за файлом с данными, займет другая информация, и к искомому файлу будет добавлена цепочка кластеров, находящаяся на некотором удалении от него. Таким образом, файл не будет располагаться в одной цепочке кластеров, и при чтении данных из файла головка жесткого диска будет достаточно интенсивно перемещаться по диску. Естественно, в настоящий момент при существующих скоростях доступа к данным на жестких дисках величины задержек могут казаться несущественными. Однако следует осознавать, что при достаточно интенсивной работе с сервером баз данных эти задержки будут накапливаться, и, в конце концов, суммарная задержка может вылиться в некоторую ощутимую величину. А если еще учесть, что в нашем случае на машине активно функционирует Web-сервер, который тоже добавляет задержки при приеме и передаче данных, станет ясно, что разработчику следует учитывать даже доли секунд. Поэтому рекомендуется заранее указать предполагаемый размер файла, содержащего базу данных. В нашем случае должно хватить десяти мегабайт.

После указания размера файла можно переходить к третьему этапы работы мастера. Его диалоговое окно показано на рис. 6.5.

Это диалоговое окно позволяет указать порядок увеличения размеров файла, содержащего базу данных, в том случае, когда его размер все-таки превысит пределы, установленные на предыдущем этапе работы мастера. Для автоматического увеличения размера файла (а это единственный разумный способ управления файлом, так как иначе тот просто не будет увеличиваться, и ввод новых данных будет заблокирован) следует выбрать переключатель **Automatically grow the database files**. После этого останется лишь указать, как именно будет рассчитываться выделяемый файлу дополнительный объем. Если выбрать переключатель **Grow the files in megabytes (MB)**, то каждый раз файл будет увеличиваться сразу на несколько мегабайт. Размер постоянного приращения задается в поле текстового ввода, связанного с данным переключателем.

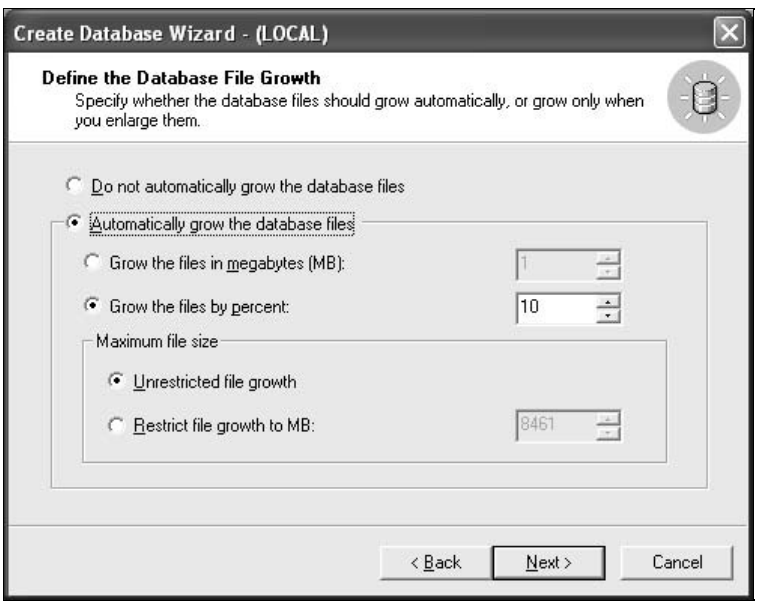

Рис. 6.5. Диалоговое окно третьего этапа работы мастера создания новой базы данных

Также можно указать, что размер приращения будет исчисляться в процентах от размера основного файла. Для этого следует выбрать переключатель **Grow the files by percent**. А в поле, связанном с этим переключателем, необходимо задать размер приращения в процентах.

Группа элементов управления **Maximum file size** позволяет жестко задавать максимально возможный размер файла, содержащего данные. Как мы уже говорили ранее, неразумно задавать конкретный верхний предел размера

файла. Поэтому рекомендуется выбирать переключатель **Unrestricted file growth**, который указывает серверу, что размеры файла можно увеличивать до заполнения всего свободного места на логическом диске.

После того как указана политика увеличения размера файла, содержащего данные, можно переходить к следующему этапу работы мастера. Необходимо сказать, что последующие два этапа его работы практически идентичны второму и третьему, когда устанавливались размеры и политика увеличения файла, содержащего данные. Но на четвертом и пятом этапах точно такие же операции будут производиться в отношении LOG-файла. Следует пояснить, что LOG-файл чаще всего занимает меньше места, чем файл с данными, однако все зависит от специфики базы данных. После указания этих параметров мастер создания базы данных завершит свою работу.

Итак, база данных создана, но это еще далеко не все. Необходимо создать таблицы, входящие в состав базы данных. Для этого необходимо открыть базу данных и активировать группу **Tables**. В базе данных уже находятся несколько служебных таблиц, но нам необходимо сделать свою, в которой будут размещаться данные. Для этого следует выполнить команду меню **Action | New Table**. При этом активируется диалоговое окно **New Table in 'Books'**, которое показано на рис. 6.6.

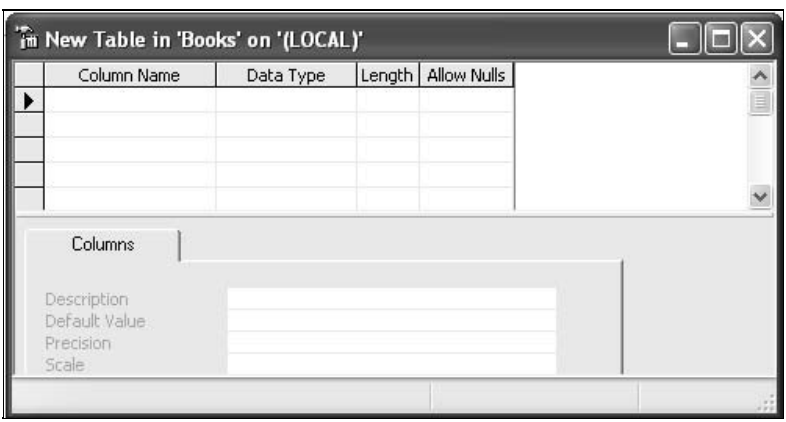

Рис. 6.6. Диалоговое окно создания новой таблицы

Создание новой таблицы может проходить как в визуальном режиме, так и при помощи стандартного SQL-скрипта. Поскольку рассматривается всего лишь тестовый пример, мы создадим облегченную версию таблицы, в которой будут храниться поля, содержащие уникальный код книги ISBN (тип `VarChar`), ее наименование (тип `VarChar`), цену (тип `Int`), наименование издательства, которое выпустило данную книгу (тип `Char`), год выпуска книги (тип `Int`). Все указанные наименования полей вместе с типами полей вводятся в диалоговое окно, показанное на рис. 6.6. После этого следует вы-

звать в текущем диалоговом окне контекстное меню и выполнить его команду **Properties**. Будет отображено окно для установки свойств создаваемой таблицы. В поле **Table Name** необходимо указать наименование для новой таблицы. Назовем ее так же, как и базу данных, — `Books`. После этого в том же контекстном меню надо выполнить команду **Save**, чтобы сохранить созданную таблицу, или воспользоваться для этой цели одноименной кнопкой на основной инструментальной панели.

Итак, база данных создана, и в ней объявлена таблица, в которой будут храниться основные данные. Так как пример упрощенный, нам и не потребуется ничего, кроме этой таблицы.

Естественно, перед тем, как использовать базу данных в паре с Web-сервисом, следует заполнить таблицу данными. Эта задача не будет слишком трудной. Ввод данных в таблицу может осуществляться при помощи все той же утилиты `Enterprise Manager`.

Сервис для просмотра данных

Будем считать, что таблица базы данных заполнена информацией, и можно приступать к работе. Для начала создадим сервис, который позволяет пользователю получать информацию из базы данных. То есть, клиентское приложение просто получит все содержимое необходимой таблицы и самостоятельно отобразит информацию. Итак, перейдем к созданию Web-сервиса.

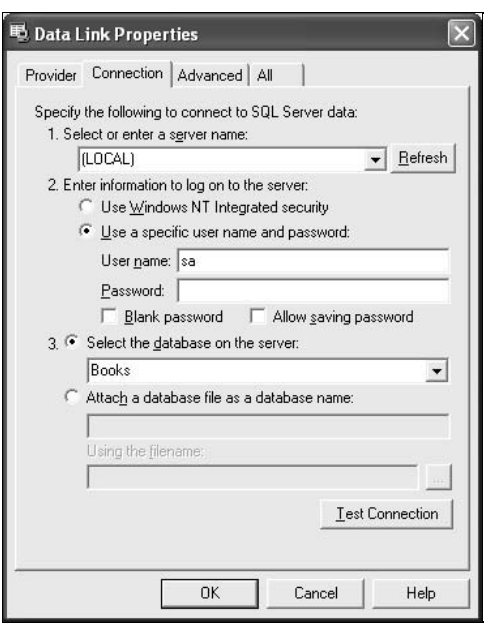

Рис. 6.7. Вкладка **Connection** диалогового окна **Data Link Properties**

Для начала, как всегда, создадим новый проект Web-сервиса. После этого необходимо задать параметры соединения с базой данных. Для этого следует выполнить команду меню **View | Server Explorer**. При этом будет активировано соответствующее окно, позволяющее устанавливать соединения с серверами баз данных. Нас интересует элемент `DataConnections`. Для него нужно вызвать контекстное меню и выполнить команду **Add Connection**. При этом активируется диалоговое окно **Data Link Properties**, позволяющее задавать свойства соединения с базой данных. Основная работа выполняется на вкладке **Connection**, чей внешний вид показан на рис. 6.7.

В поле **Select or enter a server name** следует ввести имя, под которым был зарегистрирован в системе SQL-сервер, к которому будет подключаться разрабатываемый Web-сервис. В нашем случае сервер был зарегистрирован под именем `(LOCAL)`, и именно это имя надо указать в искомом текстовом поле.

Затем следует указать логин и пароль для подключения к серверу. По умолчанию SQL Server 2000 создает для каждой базы данных учетную запись с правами администратора. Такая учетная запись имеет логин `sa` с пустым паролем. Именно этой учетной записью мы и воспользуемся для доступа к базе данных.

Для этого следует активировать переключатель **Use a specific user name and password**, который устанавливает встроенный режим аутентификации SQL Server 2000, а затем в полях **User name** и **Password** указать логин и пароль, соответственно.

Примечание

Следует осознавать, что в реальных условиях не рекомендуется использовать учетную запись с логином `sa`, так как она общеизвестна, и при определенных условиях злоумышленники могут получить доступ к SQL-серверу, и, воспользовавшись этой учетной записью, полностью изменить структуру баз данных. Поэтому, если нет возможности обойтись без этой встроенной учетной записи, следует установить для нее максимально длинный и сложный пароль. Однако стоит помнить, что любые пароли вскрываемы, поэтому при первой же возможности надо избавляться от всех учетных записей, устанавливаемых по умолчанию.

После этого осталось лишь указать базу данных, с которой будет взаимодействовать Web-сервис. Для этого следует использовать выпадающий список **Select the database on the service**. После этого в списке соединений окна **Server Explorer** появится дополнительная ветвь. Теперь достаточно отбуксировать таблицу `Books` на страницу дизайна Web-сервиса, и на этой странице появятся два компонента — `SqlConnection` и `SqlDataAdapter`. `SqlConnection` ответственен за установку соединения с базой данных, а `SqlDataAdapter` позволяет получать данные из таблицы.

Теперь необходимо создать экземпляр объекта `DataSet`, который будет содержать информацию базы данных и передаваться клиентскому приложе-

нию. Для этого следует выполнить команду меню **Data | Generate DataSet**, в появившемся одноименном диалоговом окне активировать переключатель **New**, и в текстовом поле ввода, связанном с этим переключателем, указать наименование создаваемого набора данных. Для нашего примера зададим наименование `DSBooks`. При этом, если взвести флажок в независимом переключателе **Add this dataset to the designer**, то на страницу создаваемого Web-сервиса будет добавлен соответствующий компонент. Осталось лишь написать код данного Web-сервиса. Он приведен в листинге 6.1.

Листинг 6.1

```
Imports System.Web.Services

Public Class Service1
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Web Services Designer.
    InitializeComponent()

    'Add your own initialization code after the InitializeComponent()
call

  End Sub
  Friend WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
  Friend WithEvents SqlDataAdapter1
As System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents DsBooks1 As dataservice.DSBooks

  'Required by the Web Services Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
```

```
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    Me.DsBooks1 = New dataservice.DSBooks()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'SqlSelectCommand1
    '
    Me.SqlSelectCommand1.CommandText = "SELECT ISBN, NAIM, PRICE,
PUBLISHER, BOOKYEAR FROM BOOKS"
    Me.SqlSelectCommand1.Connection = Me.SqlConnection1
    '
    'SqlInsertCommand1
    '
    Me.SqlInsertCommand1.CommandText = "INSERT INTO BOOKS(ISBN, NAIM,
PRICE, PUBLISHER, BOOKYEAR) VALUES (@ISBN, @NAIM, @" & _
    "PRICE, @PUBLISHER, @BOOKYEAR); SELECT ISBN, NAIM, PRICE, PUBLISHER,
BOOKYEAR FRO" & _
    "M BOOKS"
    Me.SqlInsertCommand1.Connection = Me.SqlConnection1
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "NAIM",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "PUBLISHER",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input, True,
CType(0, Byte), CType(0, Byte), "BOOKYEAR",
System.Data.DataRowVersion.Current, Nothing))
```

```
    '
    'SqlConnection1
    '
    Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Books;persist security info=False;user id=sa;" & _
    "workstation id=BHV-9IEMVL4EOER;packet size=4096"
    '
    'SqlDataAdapter1
    '
    Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
    Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
    Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "BOOKS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("PRICE", "PRICE"), New
System.Data.Common.DataColumnMapping("PUBLISHER", "PUBLISHER"), New
System.Data.Common.DataColumnMapping("BOOKYEAR", "BOOKYEAR")})})
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("en-US")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    CType(Me.DsBooks1, Sys-
tem.ComponentModel.ISupportInitialize).EndInit()

  End Sub

  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
  End Sub

#End Region


  <WebMethod()> Public Function GetBooks() As DSBooks
    Dim DS = New DSBooks()
```

```
    SqlDataAdapter1.Fill(DS)
    GetBooks = DS
  End Function


End Class
```

Несмотря на всю объемность приведенного кода, разработчиком был введен лишь фрагмент, входящий в состав функции `GetBooks`. Все остальное — объявление необходимых классов и компонентов, произведенное системой разработки Visual Studio .NET. Если бы код писался с самого начала разработчиком напрямую, без использования Visual Studio .NET, всю работу по объявлению необходимой функциональности ему пришлось бы взять на себя.

Итак, рассмотрим код основной функции `GetBooks`. Он приведен в следующем фрагменте:

```
<WebMethod()> Public Function GetBooks() As DSBooks
    Dim DS = New DSBooks()
    SqlDataAdapter1.Fill(DS)
    GetBooks = DS
  End Function
```

Как обычно, перед объявлением функции вставляется атрибут `<WebMethod()>`, указывающий на то, что данная функция должна быть доступна клиентским приложениям как часть функциональности Web-сервиса. Тип результата, возвращаемого функцией, также указан в ее объявлении. Это тип, задаваемый набором данных, импортированным в разработанный Web-сервис, наименование которого мы установили как `DSBooks`.

Для того чтобы функция могла вернуть этот набор данных, нам потребуется создать его и заполнить. Создание набора данных соответствующего типа производится следующей строкой:

```
Dim DS = New DSBooks()
```

Затем необходимо заполнить только что созданную переменную данными из таблицы. Так как соединение с базой данных уже установлено и сервис знает, какая таблица связывается с любым набором данных типа `DSBooks`, нам остается лишь воспользоваться готовым экземпляром класса `SqlDataAdapter`. Точнее, нам потребуется выполнить его метод `Fill`. В качестве параметра этому методу передается переменная, являющаяся набором данных, в которую и будут перенесены данные из таблицы.

После того как метод выполнен, остается лишь приравнять возвращаемое функцией значение к заполненной данными переменной. Что и делается в последней строке кода функции. Все, Web-сервис, возвращающий данные из базы данных, создан.

Код контракта этого сервиса приведен в листинге 6.2.

Листинг 6.2

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/"
xmlns:i0="http://www.tempuri.org/DSBooks.xsd" targetName-
space="http://tempuri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import namespace="http://www.tempuri.org/DSBooks.xsd"
location="http://localhost/dataservice/Service1.asmx?schema=DSBooks" />
  <types>
  <s:schema attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
    <s:import namespace="http://www.tempuri.org/DSBooks.xsd" />
    <s:element name="GetBooks">
    <s:complexType />
    </s:element>
    <s:element name="GetBooksResponse">
    <s:complexType>
      <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="GetBooksResult"
nillable="true">
        <s:complexType>
        <s:sequence>
          <s:any namespace="http://www.tempuri.org/DSBooks.xsd" />
        </s:sequence>
        </s:complexType>
      </s:element>
      </s:sequence>
    </s:complexType>
    </s:element>
    <s:element name="DSBooks" nillable="true">
    <s:complexType>
      <s:sequence>
      <s:any namespace="http://www.tempuri.org/DSBooks.xsd" />
      </s:sequence>
    </s:complexType>
    </s:element>
```

```
  </s:schema>
  </types>
  <message name="GetBooksSoapIn">
  <part name="parameters" element="s0:GetBooks" />
  </message>
  <message name="GetBooksSoapOut">
  <part name="parameters" element="s0:GetBooksResponse" />
  </message>
  <message name="GetBooksHttpGetIn" />
  <message name="GetBooksHttpGetOut">
  <part name="Body" element="s0:DSBooks" />
  </message>
  <message name="GetBooksHttpPostIn" />
  <message name="GetBooksHttpPostOut">
  <part name="Body" element="s0:DSBooks" />
  </message>
  <portType name="Service1Soap">
  <operation name="GetBooks">
    <input message="s0:GetBooksSoapIn" />
    <output message="s0:GetBooksSoapOut" />
  </operation>
  </portType>
  <portType name="Service1HttpGet">
  <operation name="GetBooks">
    <input message="s0:GetBooksHttpGetIn" />
    <output message="s0:GetBooksHttpGetOut" />
  </operation>
  </portType>
  <portType name="Service1HttpPost">
  <operation name="GetBooks">
    <input message="s0:GetBooksHttpPostIn" />
    <output message="s0:GetBooksHttpPostOut" />
  </operation>
  </portType>
  <binding name="Service1Soap" type="s0:Service1Soap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
  <operation name="GetBooks">
```

```
    <soap:operation soapAction="http://tempuri.org/GetBooks"
style="document" />
    <input>
    <soap:body use="literal" />
    </input>
    <output>
    <soap:body use="literal" />
    </output>
  </operation>
  </binding>
  <binding name="Service1HttpGet" type="s0:Service1HttpGet">
  <http:binding verb="GET" />
  <operation name="GetBooks">
    <http:operation location="/GetBooks" />
    <input>
    <http:urlEncoded />
    </input>
    <output>
    <mime:mimeXml part="Body" />
    </output>
  </operation>
  </binding>
  <binding name="Service1HttpPost" type="s0:Service1HttpPost">
  <http:binding verb="POST" />
  <operation name="GetBooks">
    <http:operation location="/GetBooks" />
    <input>
    <mime:content type="application/x-www-form-urlencoded" />
    </input>
    <output>
    <mime:mimeXml part="Body" />
    </output>
  </operation>
  </binding>
  <service name="Service1">
  <port name="Service1Soap" binding="s0:Service1Soap">
    <soap:address location="http://localhost/dataservice/Service1.asmx"
/>
  </port>
```

```
  <port name="Service1HttpGet" binding="s0:Service1HttpGet">
    <http:address location="http://localhost/dataservice/Service1.asmx"
/>
  </port>
  <port name="Service1HttpPost" binding="s0:Service1HttpPost">
    <http:address location="http://localhost/dataservice/Service1.asmx"
/>
  </port>
  </service>
</definitions>
```

Следует обратить внимание, что в этом файле контракта не описывается явно тип возвращаемого сервисом значения. Вместо этого в блоке `<types>` содержится ссылка на файл DSBooks.xsd. Код этого файла приведен в листинге 6.3.

Листинг 6.3

```
<xsd:schema id="DSBooks" targetName-
space="http://www.tempuri.org/DSBooks.xsd"
xmlns="http://www.tempuri.org/DSBooks.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:
schemas-microsoft-com:xml-msdata" attributeFormDefault="qualified"
elementFormDefault="qualified">
  <xsd:element name="DSBooks" msdata:IsDataSet="true">
  <xsd:complexType>
    <xsd:choice maxOccurs="unbounded">
    <xsd:element name="BOOKS">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ISBN" type="xsd:string" minOccurs="0" />
        <xsd:element name="NAIM" type="xsd:string" minOccurs="0" />
        <xsd:element name="PRICE" type="xsd:int" minOccurs="0" />
        <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0" />
        <xsd:element name="BOOKYEAR" type="xsd:int" minOccurs="0" />
      </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    </xsd:choice>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

В этом листинге уже отчетливо видно, как описывается структура элемента `BOOKS`, состоящего из других элементов, чьи наименования и типы соответствуют полям из используемой сервисом базы данных.

Но вернемся к разработанному нами сервису. Пришло время посмотреть, какие данные он возвращает. Естественно, мы можем осуществить доступ к нему через браузер, воспользовавшись стандартными методами `GET` или `POST`, но в результате мы получим ответ в виде "сырого" XML, который будет не совсем удобно разбирать. Поэтому разработаем обычное приложение при помощи стандартных прокси-классов сервиса.

Создадим новый проект обычного приложения, назвав его `dataclient`. Естественно, первым действием будет добавление к проекту прокси-класса используемого сервиса. Затем на форму потребуется отбуксировать компонент `DataGrid` с вкладки **Web Forms** инструментальной панели **Toolbox** и компонент `Button`. В таблице, создаваемой компонентом `DataGrid`, мы будем отображать полученные данные, а кнопка будет запускать процесс получения этих данных.

Однако для таблицы потребуется задать еще и набор данных в качестве источника данных. Для этого следует отбуксировать на форму разрабатываемого приложения компонент `DataSet` с вкладки **Data** инструментальной панели **Toolbox**. При этом автоматически активируется диалоговое окно **Add Dataset**, приведенное на рис. 6.8.

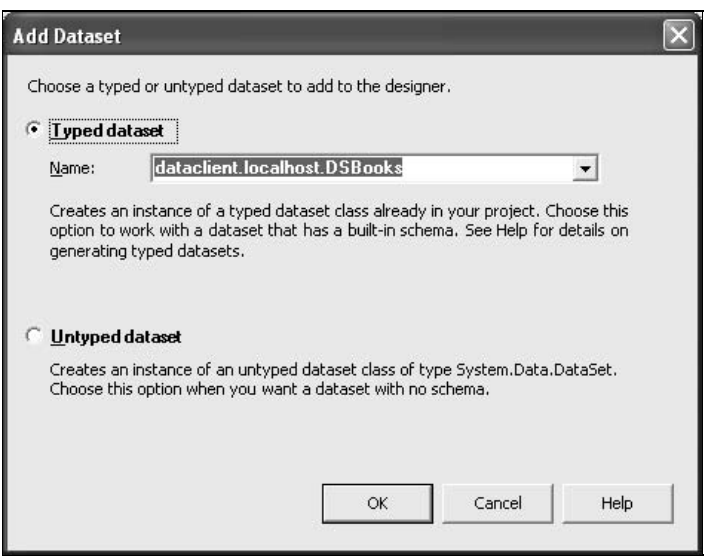

Рис. 6.8. Диалоговое окно **Add Dataset**

В этом диалоговом окне следует, как показано на рисунке, выделить переключатель **Typed dataset** и из связанного с ним выпадающего списка выбрать пол-

ное наименование типа того набора данных, который возвращается интересующим нас Web-сервисом. В нашем случае, при использовании локального WWW-сервера, этот тип имеет наименование `dataclient.localhost.DSBooks`. После указания этого имени остается лишь нажать кнопку **OK**, и необходимый набор данных добавляется к создаваемому проекту.

После того как к проекту будет добавлен набор данных, его наименование следует установить в свойстве `DataSource` компонента `DataGrid`. Наименование таблицы `Books`, которая входит в состав искомого набора данных, необходимо задать в качестве значения свойства `DataMember` все того же компонента `DataGrid`. При этом в пространстве компонента `DataGrid` отображаются поля, входящие в состав указанной таблицы. На этом этап визуального проектирования интерфейса клиентского приложения заканчивается, и можно перейти к написанию исходного кода этого приложения.

Естественно, все действия будут описываться в обработчике нажатия единственной кнопки клиентского приложения. Код нашего приложения приведен в листинге 6.4.

Листинг 6.4

```
Public Class Form1
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
```

```
    End If
    MyBase.Dispose(disposing)
  End Sub
  Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents DsBooks1 As dataclient.localhost.DSBooks

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DsBooks1 = New dataclient.localhost.DSBooks()
    Me.DataGrid1 = New System.Windows.Forms.DataGrid()
    Me.Button1 = New System.Windows.Forms.Button()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    '
    'DataGrid1
    '
    Me.DataGrid1.DataMember = "BOOKS"
    Me.DataGrid1.DataSource = Me.DsBooks1
    Me.DataGrid1.Location = New System.Drawing.Point(24, 48)
    Me.DataGrid1.Name = "DataGrid1"
    Me.DataGrid1.PreferredColumnWidth = 150
    Me.DataGrid1.Size = New System.Drawing.Size(568, 240)
    Me.DataGrid1.TabIndex = 0
```

```
    '
    'Button1
    '
    Me.Button1.Location = New System.Drawing.Point(200, 16)
    Me.Button1.Name = "Button1"
    Me.Button1.Size = New System.Drawing.Size(168, 23)
    Me.Button1.TabIndex = 1
    Me.Button1.Text = "Получить данные"
    '
    'Form1
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(616, 302)
    Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.Button1,
Me.DataGrid1})
    Me.Name = "Form1"
    Me.Text = "Клиентское приложение"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)


  End Sub


#End Region


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New dataclient.localhost.Service1()
    DsBooks1.Merge(serv.GetBooks)
  End Sub
End Class
```

Как можно видеть, основная работа была произведена разработчиком при написании функции `Button1_Click`. Сам код достаточно прост. Сначала мы объявили переменную `serv`, которая является экземпляром класса сервиса. Вторая строка является вызовом метода `Merge` набора данных `DSBooks1`, которому в качестве параметра передается результат выполнения метода `GetBooks` Web-сервиса, который, в свою очередь, возвращает набор данных. Таким образом, набор данных `DSBooks1` заполняется информацией, переданной сер-

висом, и эта информация автоматически отображается в таблице. Все, на этом участие разработчика заканчивается. Остальное сделает среда разработки.

Внешний вид рассмотренного нами клиентского приложения показан на рис. 6.9.

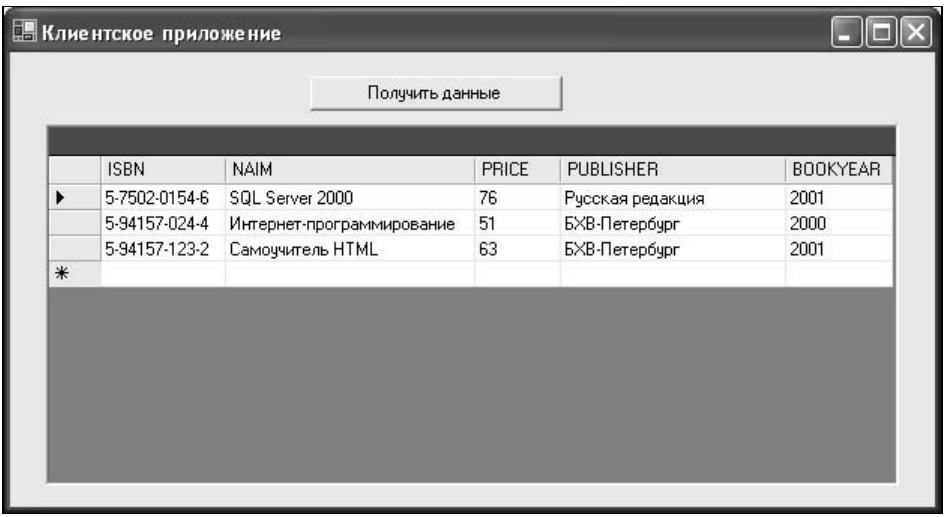

Рис. 6.9. Внешний вид клиентского приложения после получения данных от Web-сервиса

Теперь рассмотрим процедуру создания Web-клиента. Как обычно, воспользуемся возможностями технологии ASP.NET. После создания проекта ASP.NET к нему точно так же необходимо добавить ссылку на прокси-класс, а на создаваемую страницу добавить точно тот же набор органов управления, что и в предыдущем примере. То есть, нам потребуются компоненты `DataGrid`, `Button` и `DataSet`, свойства которых задаются так же, как и в предыдущем проекте. Вся логическая структура, представленная в виде HTML-кода, обрабатываемого WWW-сервером, приведена в листинге 6.5.

Листинг 6.5

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="aspdataclient.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
```

```
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Button id="Button1" style="Z-INDEX: 101; LEFT: 32px; POSITION:
absolute; TOP: 35px" runat="server" Text="Получить данные" Width="222px"
Height="24px"></asp:Button>
   <asp:DataGrid id="DataGrid1" style="Z-INDEX: 102; LEFT: 38px;
POSITION: absolute; TOP: 77px" runat="server" Width="414px"
Height="164px" DataSource="<%# DsBooks1 %>"
DataMember="BOOKS"></asp:DataGrid>
  </form>
 </body>
</HTML>
```

Стоит обратить внимание, что элемент `DataSet` в этом коде явно не объявлен. И это понятно, так как в HTML-коде указываются только визуальные компоненты. Объявление этого набора данных будет произведено в классе, реализующем функциональность данной Web-страницы. А основная ее функциональность, как обычно, сосредоточена в функции, обрабатывающей нажатие на кнопку. Впрочем, это отчетливо видно в листинге 6.6, где приведен код искомого класса.

Листинг 6.6

```
Public Class WebForm1
  Inherits System.Web.UI.Page
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents DsBooks1 As aspdataclient.localhost.DSBooks
  Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid

#Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DsBooks1 = New aspdataclient.localhost.DSBooks()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
```

```
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("en-US")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()


  End Sub


  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub


#End Region


  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New aspdataclient.localhost.Service1()
    DsBooks1.Merge(serv.GetBooks)
    DataGrid1.DataBind()
  End Sub
End Class
```

При первой загрузке этой Web-страницы в браузер таблица на ней не отображается, так как связанный с ней набор данных еще не заполнен. При нажатии пользователем на кнопку выполняется код, содержащийся в функции `Button1_Click`. Этот код предназначен для заполнения набора данных `DSBooks1`. Сам код практически идентичен соответствующему фрагменту кода обычного клиентского приложения, которое мы рассматривали в этом разделе главы несколько ранее. Добавлен еще вызов метода `DataBind` для объекта `DataGrid1`, обновляющего связь между таблицей и ее источником данных и заставляющего WWW-сервер отсылать браузеру удаленного поль-

зователя новую версию Web-страницы, в которой искомая таблица содержит обновленную информацию.

Внешний вид разработанной нами клиентской Web-страницы показан на рис. 6.10.

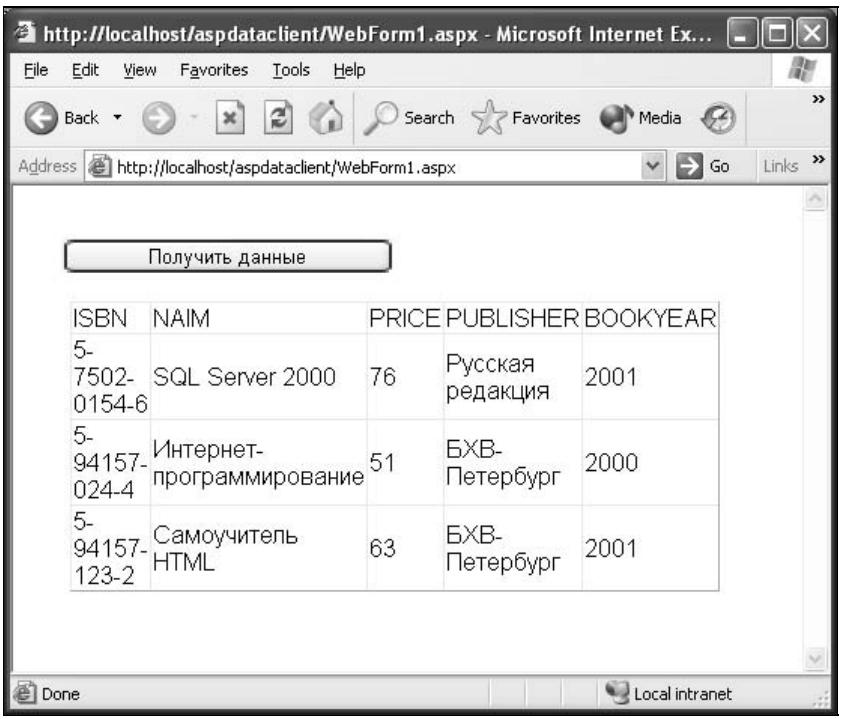

Рис. 6.10. Внешний вид клиентской Web-страницы после того, как пользователь нажал кнопку **Получить данные**

Итак, в этом разделе мы научились создавать Web-сервисы, которые возвращают в результате своей работы наборы данных, и клиентские приложения, которые могут получать переданные наборы данных и адекватно отображать их.

Использование SQL-запроса

Механизм работы, описанный в предыдущем разделе, когда пользователь получает сразу все содержимое базы данных, может удовлетворительно функционировать на тестовой базе данных, которая содержит очень мало информации. В реальной жизни перекачивание через Интернет содержимого всей базы данных — не самое оптимальное решение.

Пользователь должен иметь возможность перед запросом информации задавать критерии, которым она должна удовлетворять. То есть, при достаточно больших объемах данных вариант, когда пользователю передается вся информация из таблицы, а уже потом он проводит операции выборки и сортировки, не может быть признан оптимальным.

Поэтому пользователь должен иметь возможность заранее задать параметры выборки и получить именно те данные, которые им затребованы. Подобная модель работы естественна для распределенных вариантов работы с SQL-сервером. Естественно, что при работе через Интернет подобная модель передачи только затребованных данных приобретает еще большую актуальность, так как загружать и без того не слишком широкий канал доступа пользователя ненужной информацией — слишком большая роскошь.

В качестве параметров поиска сервис сможет воспринимать границы диапазона, в который должна попадать цена книг, интересующих пользователя. На основе этих данных мы построим SQL-запрос к базе данных и передадим клиентскому приложению только результат запроса, а не все содержимое таблицы.

Попробуем создать подобный Web-сервис и набор соответствующих ему клиентских приложений, усовершенствовав пример из предыдущего раздела главы. Код усовершенствованного Web-сервиса `dataservice` приведен в листинге 6.7.

Листинг 6.7

```
Imports System.Web.Services

Public Class Service1
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Web Services Designer.
    InitializeComponent()

    'Add your own initialization code after the InitializeComponent()
call

  End Sub
```

```
  Friend WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
  Friend WithEvents SqlDataAdapter1 As
System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents DsBooks1 As dataservice.DSBooks

  'Required by the Web Services Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DsBooks1 = New dataservice.DSBooks()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    '
    'SqlSelectCommand1
    '
    Me.SqlSelectCommand1.CommandText = "SELECT ISBN, NAIM, PRICE,
PUBLISHER, BOOKYEAR FROM BOOKS WHERE (PRICE >= @p1) AND" & _
    " (PRICE <= @p2)"
    Me.SqlSelectCommand1.Connection = Me.SqlConnection1
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p1", System.Data.SqlDbType.Int, 4,
System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p2", System.Data.SqlDbType.Int, 4,
```

```
System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    '
    'SqlConnection1
    '
    Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Books;persist security info=False;user id=sa;" & _
    "workstation id=BHV-9IEMVL4EOER;packet size=4096"
    '
    'SqlInsertCommand1
    '
    Me.SqlInsertCommand1.CommandText = "INSERT INTO BOOKS(ISBN, NAIM,
PRICE, PUBLISHER, BOOKYEAR) VALUES (@ISBN, @NAIM, @" & _
    "PRICE, @PUBLISHER, @BOOKYEAR); SELECT ISBN, NAIM, PRICE, PUBLISHER,
BOOKYEAR FRO" & _
    "M BOOKS"
    Me.SqlInsertCommand1.Connection = Me.SqlConnection1
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "NAIM",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "PUBLISHER",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input, True,
CType(0, Byte), CType(0, Byte), "BOOKYEAR",
System.Data.DataRowVersion.Current, Nothing))
    '
    'SqlDataAdapter1
    '
    Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
    Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
```

```
    Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "BOOKS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("PRICE", "PRICE"), New
System.Data.Common.DataColumnMapping("PUBLISHER", "PUBLISHER"), New
System.Data.Common.DataColumnMapping("BOOKYEAR", "BOOKYEAR")})})
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()

  End Sub

  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
  End Sub

#End Region



  <WebMethod()> Public Function GetBooks(ByVal pr1 As Integer, ByVal pr2
As Integer) As DSBooks
    Dim DS = New DSBooks()
    SqlSelectCommand1.Parameters("@p1").Value = pr1
    SqlSelectCommand1.Parameters("@p2").Value = pr2
    SqlDataAdapter1.Fill(DS)
    GetBooks = DS
  End Function


End Class
```

Теперь рассмотрим процедуру разработки этого сервиса. Как мы знаем, необходимо создать SQL-запрос `Select`, который будет отбирать из базы данных книги, чьи цены располагаются в заданном промежутке. Здесь следует сделать некоторое отступление.

Дело в том, что компонент `SqlDataAdapter` включает свойство `SelectCommand`. В этом свойстве содержится SQL-запрос `Select`, который выполняется каждый раз, когда компонент получает данные из таблицы. В предыдущем проекте сервиса мы также использовали это свойство, но неявно. По умолчанию такой SQL-запрос получает все содержимое таблицы, поэтому нам не пришлось каким-либо образом изменять текст запроса. Теперь мы можем явно задать SQL-запрос, и каждый раз, когда мы будем заполнять набор данных, этот SQL-запрос будет автоматически выполняться. То есть, если

мы задаем содержимое свойства `SelectCommand`, нам не нужно заботиться о явном выполнении запроса.

Итак, для указания текста SQL-запроса следует на странице конструирования Web-сервиса выделить компонент `SqlDataAdapter1` и активизировать его инспектор свойств. Нас будет интересовать составное свойство `SelectCommand`. Необходимо раскрыть его содержимое и активировать свойство `CommandText`. Содержимое этого свойства является текстом искомого SQL-запроса `Select`. Такой текст можно задать, либо просто вводя текст в поле значения свойства, либо воспользовавшись стандартным инструментом построения запросов в Visual Studio .NET — **Query Builder**. Внешний вид диалогового окна этого инструмента показан на рис. 6.11.

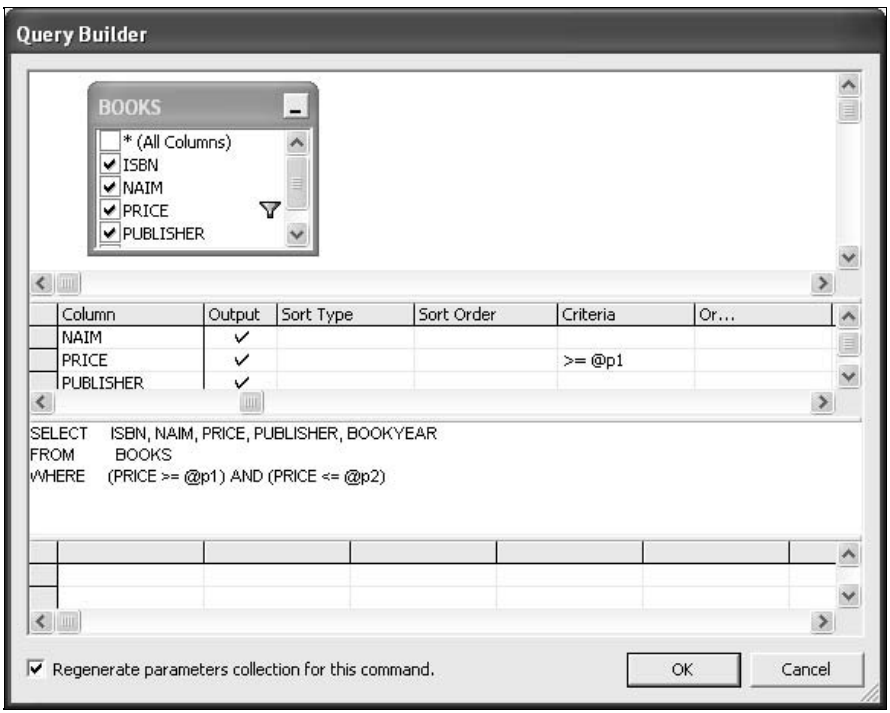

Рис. 6.11. Диалоговое окно **Query Builder**

Само диалоговое окно **Query Builder** разбито на четыре части.

- В верхней его части отображается визуальная схема отношения между таблицами базы данных, которое задается объявляемым SQL-запросом. Здесь показываются все таблицы, входящие в обрабатываемую базу данных, и разработчик может указать, пометив флажками, какие именно поля из каких таблиц должны входить в результат SQL-запроса.

- Вторая часть диалогового окна построителя запросов отображает список всех полей и позволяет указывать условия, накладываемые на них, порядок сортировки содержимого полей при выводе результатов запроса и многие другие параметры.
- Третья часть окна содержит текст SQL-запроса. В том случае, если запрос не слишком сложен, а разработчик достаточно хорошо знаком с синтаксисом SQL, подчас проще напрямую внести текст SQL-запроса в поле текстового ввода.
- Четвертая, самая нижняя часть диалогового окна предназначена для вывода результатов SQL-запроса. То есть, разработчик имеет возможность при создании запроса сразу увидеть результат его выполнения, что, несомненно, является большим подспорьем в деле раннего обнаружения возможных ошибок.

Но вернемся к созданию нашего запроса. По умолчанию используется SQL-запрос `Select`, который сразу выводит содержимое всех записей таблицы. То есть, начальный запрос в нашем случае выглядит следующим образом:

```
SELECT ISBN, NAIM, PRICE, PUBLISHER, BOOKYEAR FROM BOOKS
```

Нам необходимо наложить на этот запрос ограничения на содержимое поля `PRICE`. Для этого нам придется использовать ключевое слово `WHERE`, после которого и указывать ограничения, налагаемые на поле `PRICE`. Но проблема заключается в том, что пользователь каждый раз будет задавать самые разные границы ценового диапазона, поэтому мы не сможем использовать в запросе какие-либо готовые значения.

Можно было бы, конечно, пойти другим путем. При получении искомых границ ценового диапазона динамически формировать текст SQL-запроса, присваивать этот текст в качестве значения свойству `CommandText`, входящему в состав свойства `SelectCommand`, а затем инициировать заполнение набора данных. Однако есть более легкий и правильный путь.

Любая достаточно серьезная система разработки приложений, в том числе, естественно, и Visual Studio .NET, позволяет создавать SQL-запросы с параметрами. В том случае, когда нельзя заранее предсказать, какие конкретные значения будут использоваться в SQL-запросе, можно заменить их параметрами, а затем в ходе выполнения приложения просто изменять значения этих параметров, и SQL-запрос будет выполняться правильно.

В Visual Studio .NET наименования параметров запроса должны начинаться с символа `@`. Нам потребуется использовать два параметра — для верхней и нижней границы ценового диапазона. Назовем их `p1` и `p2`, соответственно. Тогда текст нашего SQL-запроса будет выглядеть следующим образом:

```
SELECT   ISBN, NAIM, PRICE, PUBLISHER, BOOKYEAR
FROM     BOOKS
WHERE    (PRICE >= @p1) AND (PRICE <= @p2)
```

Проверить правильность создания SQL-запроса можно прямо в диалоговом окне **Query Builder**. Для этого следует в контекстном меню панели, в которой отображается текст запроса, выполнить команду **Verify SQL Syntax**. Если сам запрос составлен правильно, то будет отображено соответствующее сообщение. Если запрос успешно не проходит процедуру проверки синтаксиса, отображается сообщение с кратким описанием найденной ошибки.

Однако правильность синтаксиса еще не гарантирует, что SQL-запрос будет соответствовать замыслу разработчика. Запрос может быть синтаксически верным, но при этом содержать логические ошибки в области группировок, сортировок, в налагаемых условиях или в чем-либо еще. Поэтому следует на этапе проектирования SQL-запроса проверять, какие результаты он возвращает.

Мы уже говорили, что самая нижняя область диалогового окна как раз и предназначена для отображения результатов проектируемого запроса. Запуск SQL-запроса на выполнение производится при помощи команды контекстного меню **Run**. При этом, если в SQL-запросе присутствуют, как в нашем случае, параметры, перед выполнением запроса отображается диалоговое окно **Define Query Parameters**, в котором разработчик должен задать значения всех используемых в SQL-запросе параметров.

Итак, мы задали SQL-запрос, который будет выполняться каждый раз, когда потребуется заполнить стандартный набор данных. Теперь необходимо перейти к разработке функциональности сервиса. Основная функция сервиса выглядит следующим образом:

```
<WebMethod()> Public Function GetBooks(ByVal pr1 As Integer, ByVal pr2 As
Integer) As DSBooks
  Dim DS = New DSBooks()
  SqlSelectCommand1.Parameters("@p1").Value = pr1
  SqlSelectCommand1.Parameters("@p2").Value = pr2
  SqlDataAdapter1.Fill(DS)
  GetBooks = DS
End Function
```

Как можно видеть, на этот раз основная функция Web-сервиса получает два целочисленных параметра. С их помощью мы установим значения параметров SQL-запроса. Из приведенного фрагмента кода видно, что для установки значения параметра SQL-запроса используется следующая строка:

```
SqlSelectCommand1.Parameters("@p1").Value = pr1
```

После того как заданы значения всех параметров, остается лишь заполнить набор данных, который будет возвращать функция. Но эту процедуру мы уже обсуждали в предыдущем разделе главы.

Теперь рассмотрим процедуру создания обычного клиентского приложения и Web-страницы, основанной на технологии ASP.NET и предоставляющей

доступ к сервису. Начнем с обычного клиентского приложения. Его мы тоже разработаем на базе клиента из предыдущего раздела.

Но так как сервис несколько изменился, придется немного изменить и клиентское приложение. На форму разрабатываемого приложения необходимо добавить два поля ввода, в которых пользователь будет указывать границы интересующего его ценового диапазона. Весь код этого приложения, в котором объявлен и его дизайн, и его функциональность, приведен в листинге 6.8.

Листинг 6.8

```
Public Class Form1
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub
  Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents DsBooks1 As dataclient.localhost.DSBooks
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
```

```
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DsBooks1 = New dataclient.localhost.DSBooks()
    Me.DataGrid1 = New System.Windows.Forms.DataGrid()
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    '
    'DataGrid1
    '
    Me.DataGrid1.DataMember = "BOOKS"
    Me.DataGrid1.DataSource = Me.DsBooks1
    Me.DataGrid1.Location = New System.Drawing.Point(24, 56)
    Me.DataGrid1.Name = "DataGrid1"
    Me.DataGrid1.PreferredColumnWidth = 150
    Me.DataGrid1.Size = New System.Drawing.Size(568, 232)
    Me.DataGrid1.TabIndex = 0
```

```
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(368, 16)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(168, 23)
Me.Button1.TabIndex = 1
Me.Button1.Text = "Получить данные"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(16, 15)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(72, 23)
Me.Label1.TabIndex = 2
Me.Label1.Text = "Цена книг от"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(96, 16)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.TabIndex = 3
Me.TextBox1.Text = "0"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(208, 15)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(32, 23)
Me.Label2.TabIndex = 4
Me.Label2.Text = "до"
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(248, 16)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.TabIndex = 5
Me.TextBox2.Text = "400"
```

```
    '
    'Form1
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(616, 302)
    Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.TextBox2,
Me.Label2, Me.TextBox1, Me.Label1, Me.Button1, Me.DataGrid1})
    Me.Name = "Form1"
    Me.Text = "Клиентское приложение"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)


  End Sub


#End Region


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim t1, t2 As Integer
    t1 = Convert.ToInt32(TextBox1.Text)
    t2 = Convert.ToInt32(TextBox2.Text)
    DsBooks1.Merge(serv.GetBooks(t1, t2))
  End Sub
End Class
```

Внешний вид приложения, код которого приведен в листинге 6.8, показан на рис. 6.12.

Из листинга 6.8 видно, что процедура вызова функции `GetBooks` практически не изменилась. Но мы завели две целочисленные переменные, чтобы передавать их этой функции в качестве параметров. В них мы заносим числовые значения, внесенные пользователем в поля текстового ввода. Так как по умолчанию введенные значения имеют тип `String`, то их приходится конвертировать при помощи метода `ToInt32`, присущего объекту `Convert`.

Конечно, перед тем как производить операцию конвертирования текстового значения в целочисленное, стоило бы убедиться, что пользователь ввел в соответствующее поле именно целое число. Мы увидим, как это делается, на примере создания Web-страницы, выступающей в качестве клиента для разработанного нами Web-сервиса.

Как и прежде, для разработки Web-клиента мы воспользуемся примером из предыдущего раздела главы, усовершенствовав его. На разрабатываемой Web-странице уже располагается один компонент `DataGrid` и одна кнопка `Button`. Нам дополнительно потребуется разместить на Web-странице два компонента `TextBox`, чтобы пользователь мог указывать в них границы ценового диапазона, и два компонента `Label`, описывающих установленные поля текстового ввода.

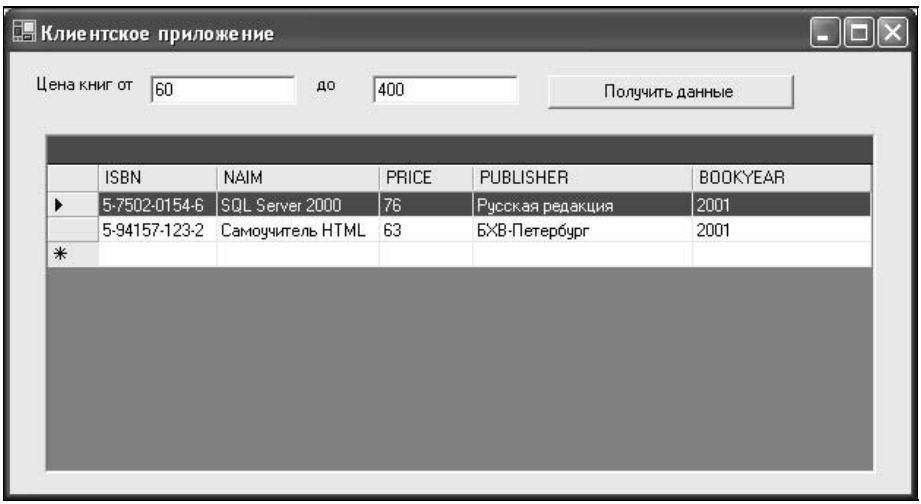

Рис. 6.12. Внешний вид клиентского приложения

Но, как мы говорили, в этом проекте будет производиться проверка значений, вводимых пользователем. Поэтому следует разместить на Web-странице два компонента `RangeValidator`. Каждый из этих компонентов будет контролировать значение, указанное в соответствующем поле ввода. Для этого нужно в свойстве `ControlToValidate` задать наименование того находящегося на Web-странице органа управления, который и будет "курировать" компонент `RangeValidator`. Так как у нас оба этих компонента будут контролировать соответствующие поля ввода, то именно их наименования и следует указать в качестве значения свойства `ControlToValidate`.

Также нужно задать значение свойства `ErrorMessage`, в котором содержится текст предупреждения, отображаемого при вводе неверного значения. Следует отметить, что функциональность компонентов `RangeValidator` реализуется при помощи скриптовых языков, действующих на стороне пользователя, поэтому для проверки валидности введенных значений они на сервер отсылаться не будут. Анализ их допустимости будет происходить на стороне клиента, что, несомненно, ускоряет процесс проверки.

Границы интервала, в котором обязано располагаться вводимое число, задаются при помощи свойств `MinimumValue` и `MaximumValue`. В нашем случае

мы предполагаем, что цена книги будет находиться в промежутке от нуля до тысячи. Соответственно, придется задать значения этих свойств. Следует так же указать и тип вводимого значения. Этот тип указывается при помощи свойства `Type`, и, в нашем случае, этому свойству будет придано значение `Integer`.

Вся структура дизайна разработанной клиентской Web-страницы с указанием использованных объектов и значений их свойств описывается HTML-кодом, который приведен в листинге 6.9.

Листинг 6.9

```
<%@ Page Language="vb" AutoEventWireup="false" Codebe-
hind="WebForm1.aspx.vb" Inherits="aspdataclient.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Button id="Button1" style="Z-INDEX: 100; LEFT: 39px; POSITION:
absolute; TOP: 142px" runat="server" Text="Получить данные" Width="222px"
Height="24px"></asp:Button>
   <asp:RangeValidator id="RangeValidator2" style="Z-INDEX: 108; LEFT:
330px; POSITION: absolute; TOP: 62px" runat="server" Height="17px"
Width="185px" ErrorMessage="Число должно располагаться в промежутке от 0
до 1000" ControlToValidate="TextBox2" MaximumValue="1000"
MinimumValue="0" Type="Integer"></asp:RangeValidator>
   <asp:TextBox id="TextBox2" style="Z-INDEX: 105; LEFT: 321px; POSITION:
absolute; TOP: 26px" runat="server">400</asp:TextBox>
   <asp:Label id="Label2" style="Z-INDEX: 103; LEFT: 286px; POSITION:
absolute; TOP: 32px" runat="server">До</asp:Label>
   <asp:DataGrid id="DataGrid1" style="Z-INDEX: 101; LEFT: 39px;
POSITION: absolute; TOP: 203px" runat="server" Width="414px"
Height="164px" DataSource="<%# DsBooks1 %>"
DataMember="BOOKS"></asp:DataGrid>
   <asp:Label id="Label1" style="Z-INDEX: 102; LEFT: 39px; POSITION:
absolute; TOP: 32px" runat="server">Цена от</asp:Label>
```

```
   <asp:TextBox id="TextBox1" style="Z-INDEX: 104; LEFT: 113px; POSITION:
absolute; TOP: 26px" runat="server">0</asp:TextBox>
   <asp:RangeValidator id="RangeValidator1" style="Z-INDEX: 107; LEFT:
105px; POSITION: absolute; TOP: 62px" runat="server" Height="27px"
Width="186px" ErrorMessage="Число должно располагаться в промежутке от 0
до 1000" ControlToValidate="TextBox1" MaximumValue="1000"
MinimumValue="0" Type="Integer"></asp:RangeValidator>
  </form>
 </body>
</HTML>
```

Естественно, одного HTML-кода недостаточно для функционирования Web-страницы, поэтому следует привести код класса, реализующего эту страницу. Он показан в листинге 6.10.

Листинг 6.10

```
Public Class WebForm1
  Inherits System.Web.UI.Page
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents DsBooks1 As aspdataclient.localhost.DSBooks
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label
  Protected WithEvents Label2 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
  Protected WithEvents RangeValidator1 As
System.Web.UI.WebControls.RangeValidator
  Protected WithEvents RangeValidator2 As
System.Web.UI.WebControls.RangeValidator
  Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DsBooks1 = New aspdataclient.localhost.DSBooks()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'DsBooks1
    '
```

```
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("en-US")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()


  End Sub


  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub


#End Region


  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New aspdataclient.localhost.Service1()
    Dim t1, t2 As Integer
    t1 = Convert.ToInt32(TextBox1.Text)
    t2 = Convert.ToInt32(TextBox2.Text)
    DsBooks1.Merge(serv.GetBooks(t1, t2))
    DataGrid1.DataBind()
  End Sub
End Class
```

Как можно видеть, основная функция, являющаяся обработчиком нажатия на кнопку, фактически идентична подобной функции из предыдущего проекта обычного клиентского приложения.

Созданная клиентская Web-страница действует согласно замыслу разработчика, и ее внешний вид практически идентичен Web-странице из предыдущего раздела этой главы. Поэтому стоит обратить внимание на то, как осуществляется проверка допустимости значений, вводимых пользователем. Как мы уже говорили, эта проверка производится на стороне пользователя

при помощи скриптов. Естественно, HTML-код Web-страницы, переданной браузеру удаленного пользователя, отличается от исходного HTML-кода, который хранится на сервере. Конечный HTML-код клиентской Web-страницы приведен в листинге 6.11.

Листинг 6.11

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form name="Form1" method="post" action="WebForm1.aspx"
language="javascript" onsubmit="ValidatorOnSubmit();" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTI1NTQwNDU0NDt0PDtsPGk8MT47PjtsPHQ8O2w8aTw5Pjs+O2w8dDxAMDw7Oz
s7Ozs7Ozs7Pjs7Pjs+Pjs+Pjs+" />
<script language="javascript"
src="/aspnet_client/system_Web/1_0_2914_16/WebUIValidation.js"></script>
   <input type="submit" name="Button1" value="Получить данные"
onclick="if (typeof(Page_ClientValidate) == 'function')
Page_ClientValidate(); " language="javascript" id="Button1"
style="height:24px;width:222px;Z-INDEX: 100; LEFT: 39px; POSITION:
absolute; TOP: 142px" />
   <span id="RangeValidator2" controltovalidate="TextBox2"
errormessage="Число должно располагаться в промежутке от 0 до 1000"
type="Integer" evaluationfunction="RangeValidatorEvaluateIsValid"
maximumvalue="1000" minimumvalue="0"
style="color:Red;height:17px;width:185px;Z-
INDEX:108;LEFT:330px;POSITION:absolute;TOP:62px;visibility:hidden;">Число
должно располагаться в промежутке от 0 до 1000</span>
   <input name="TextBox2" type="text" value="400" id="TextBox2"
style="Z-INDEX: 105; LEFT: 321px; POSITION: absolute; TOP: 26px" />
   <span id="Label2" style="Z-INDEX: 103; LEFT: 286px; POSITION:
absolute; TOP: 32px">До</span>
   <span id="Label1" style="Z-INDEX: 102; LEFT: 39px; POSITION: absolute;
TOP: 32px">Цена от</span>
   <input name="TextBox1" type="text" value="0" id="TextBox1"
style="Z-INDEX: 104; LEFT: 113px; POSITION: absolute; TOP: 26px" />
```

```
   <span id="RangeValidator1" controltovalidate="TextBox1"
errormessage="Число должно располагаться в промежутке от 0 до 1000"
type="Integer" evaluationfunction="RangeValidatorEvaluateIsValid"
maximumvalue="1000" minimumvalue="0"
style="color:Red;height:27px;width:186px;
ZINDEX:107;LEFT:105px;POSITION:absolute;TOP:62px;visibility:hidden;">Число
должно располагаться в промежутке от 0 до 1000</span>
<script language="javascript">
<!--
 var Page_Validators =  new Array(document.all["RangeValidator2"],
document.all["RangeValidator1"]);
  // -->
</script>
<script language="javascript">
<!--
var Page_ValidationActive = false;
if (typeof(clientInformation) != "undefined" &&
clientInformation.appName.indexOf("Explorer") != -1) {
  if (typeof(Page_ValidationVer) == "undefined")
    alert("Unable to find script library
'/aspnet_client/system_Web/1_0_2914_16/WebUIValidation.js'. Try placing
this file manually, or reinstall by running 'aspnet_regiis -c'.");
  else if (Page_ValidationVer != "121")
    alert("This page uses an incorrect version of WebUIValidation.js. The
page expects version 121. The script library is " + Page_ValidationVer +
".");
  else
    ValidatorOnLoad();
}
function ValidatorOnSubmit() {
  if (Page_ValidationActive) {
    ValidatorCommonOnSubmit();
  }
}
// -->
</script>
  </form>
 </body>
</HTML>
```

На рис. 6.13 показано, как будет выглядеть Web-страница, если пользователь введет в текстовые поля информацию, не удовлетворяющую наложенным условиям. Следует отметить, что активизация компонентов `RangeValidator`

происходит в тот момент, когда опекаемый ими орган управления теряет фокус ввода.

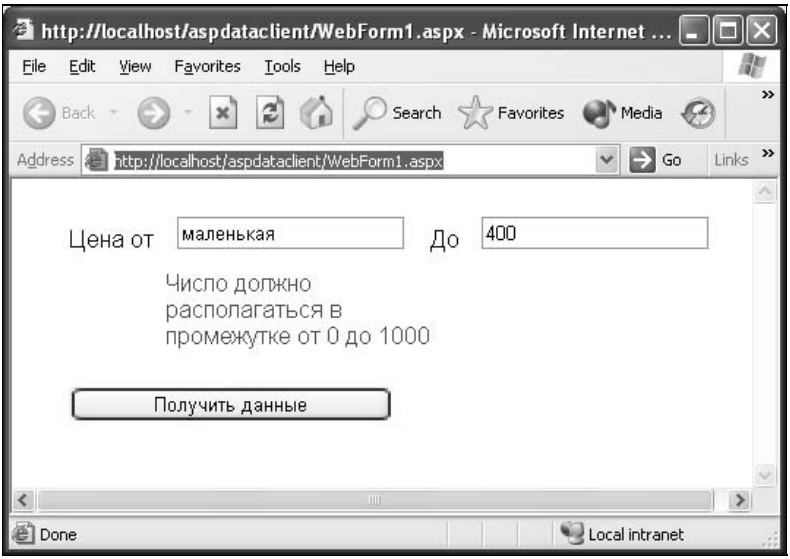

Рис. 6.13. Внешний вид клиентской Web-страницы при попытке ввести неверные данные

Добавление данных

Нельзя сказать, что при создании реальных Web-сервисов, взаимодействующих с базой данных, потребуется только, пользуясь SQL-запросом, получать данные. Чаще всего будет необходимо создавать полноценный инструментарий доступа к базе данных, позволяющий добавлять информацию, изменять записи и удалять их. В этом разделе мы узнаем, как Web-сервис предоставляет клиентским приложениям возможность добавлять информацию в базу данных.

Для добавления данных в таблицу используется SQL-запрос `Insert`. Для его выполнения нам придется в Web-сервисе создать еще одну функцию, которая будет ответственна за добавление информации в таблицу. Сам запрос создается с использованием уже знакомого нам инструмента **Query Builder** при помощи свойства `SqlInsertCommand`, присущего объекту `SqlDataAdapter`. Запрос, связанный с этим свойством, состоит из комбинации запросов `Insert` и `Select`. То есть, данная команда позволяет сначала добавить запись, а потом получить снова весь набор данных. Нам это не нужно. Наш запрос будет состоять только из команды `Insert`. Поэтому окончательная версия SQL-запроса будет выглядеть следующим образом:

```
INSERT INTO BOOKS (ISBN, NAIM, PRICE, PUBLISHER, BOOKYEAR) VALUES (@ISBN,
@NAIM, @PRICE, @PUBLISHER, @BOOKYEAR)
```

Из текста запроса видно, что мы используем пять параметров, наименования которых соответствуют наименованиям полей. Итак, SQL-запрос готов, осталось лишь переписать наш сервис. Его код приведен в листинге 6.12.

Листинг 6.12

```
Imports System.Web.Services

Public Class Service1
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Web Services Designer.
    InitializeComponent()

    'Add your own initialization code after the InitializeComponent()
call

  End Sub
  Friend WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
  Friend WithEvents SqlDataAdapter1 As
System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents DsBooks1 As dataservice.DSBooks

  'Required by the Web Services Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
```

```
    Me.DsBooks1 = New dataservice.DSBooks()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    '
    'SqlSelectCommand1
    '
    Me.SqlSelectCommand1.CommandText = "SELECT ISBN, NAIM, PRICE,
PUBLISHER, BOOKYEAR FROM BOOKS WHERE (PRICE >= @p1) AND" & _
    " (PRICE <= @p2)"
    Me.SqlSelectCommand1.Connection = Me.SqlConnection1
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p1", System.Data.SqlDbType.Int, 4,
System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p2", System.Data.SqlDbType.Int, 4,
System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    '
    'SqlConnection1
    '
    Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Books;persist security info=False;user id=sa;" & _
    "workstation id=BHV-9IEMVL4EOER;packet size=4096"
    '
    'SqlInsertCommand1
    '
    Me.SqlInsertCommand1.CommandText = "INSERT INTO BOOKS (ISBN, NAIM,
PRICE, PUBLISHER, BOOKYEAR) VALUES (@ISBN, @NAIM, " & _
    "@PRICE, @PUBLISHER, @BOOKYEAR)"
    Me.SqlInsertCommand1.Connection = Me.SqlConnection1
```

```
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "NAIM",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, System.Data.ParameterDirection.Input, True, CType(0, Byte), CType(0,
Byte), "PRICE", System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
True, CType(0, Byte), CType(0, Byte), "PUBLISHER",
System.Data.DataRowVersion.Current, Nothing))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input, True,
CType(0, Byte), CType(0, Byte), "BOOKYEAR",
System.Data.DataRowVersion.Current, Nothing))
    '
    'SqlDataAdapter1
    '
    Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
    Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
    Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "BOOKS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("PRICE", "PRICE"), New
System.Data.Common.DataColumnMapping("PUBLISHER", "PUBLISHER"), New
System.Data.Common.DataColumnMapping("BOOKYEAR", "BOOKYEAR")})})
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()

  End Sub

  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
  End Sub

#End Region
```

```
  <WebMethod()> Public Function GetBooks(ByVal pr1 As Integer, ByVal pr2
As Integer) As DSBooks
    Dim DS = New DSBooks()
    SqlSelectCommand1.Parameters("@p1").Value = pr1
    SqlSelectCommand1.Parameters("@p2").Value = pr2
    SqlDataAdapter1.Fill(DS)
    GetBooks = DS
  End Function
  <WebMethod()> Public Function InsertBook(ByVal ISBN As String, ByVal
NAIM As String, ByVal PRICE As Integer, ByVal PUBLISHER As String, ByVal
BOOKYEAR As Integer) As String
    Dim DS = New DSBooks()
    SqlInsertCommand1.Parameters("@ISBN").Value = ISBN
    SqlInsertCommand1.Parameters("@NAIM").Value = NAIM
    SqlInsertCommand1.Parameters("@PRICE").Value = PRICE
    SqlInsertCommand1.Parameters("@PUBLISHER").Value = PUBLISHER
    SqlInsertCommand1.Parameters("@BOOKYEAR").Value = BOOKYEAR
    SqlConnection1.Open()
    SqlInsertCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    InsertBook = "Данные успешно добавлены"
  End Function
End Class
```

Итак, разберем еще одну функцию, вошедшую в состав Web-сервиса. Кстати, следует отметить, что это первый наш сервис, который содержит не одну исполняемую функцию. Новой функции `InsertBook` передается пять параметров, которые содержат информацию о книге, добавляемой в базу данных. Эти параметры мы используем в качестве значения параметров SQL-запроса `Insert`, связанного со свойством `InsertCommand`. После того как мы задали значение всех параметров, следует выполнить искомый SQL-запрос. Так как он не возвращает какого-либо набора данных, следует вызвать метод `ExecuteNonQuery`, присущий свойству `InsertCommand`.

Из листинга видно, что перед вызовом метода `ExecuteNonQuery` выполняется метод `SqlConnection1.Open`. Дело в том, что для выполнения SQL-запроса необходимо, чтобы было установлено соединение с SQL-сервером. Когда мы вызываем метод `Fill` для набора данных, это соединение устанавливается автоматически. А для метода `ExecuteNonQuery` автоматической установки соединения не происходит, поэтому мы явно вызываем метод `SqlConnection1.Open`. Естественно, после выполнения SQL-запроса соединение следует закрыть. Для этого явно вызывается метод `SqlConnection1.Close`.

В качестве результата своей работы функция возвращает обычную строку, где указывает, что вставка записи в таблицу прошла удачно. Конечно, в реальных условиях следует установить блок обработки нештатных ситуаций и в том случае, если по каким-либо причинам не произошло выполнение SQL-запроса, функция должна вернуть соответствующее извещение. Но в нашем тестовом примере это несколько излишне.

Теперь перейдем к созданию соответствующих версий клиентов для этого сервиса. Начнем с обычного Windows-приложения. Основа у нас уже есть, остается лишь добавить к существующему клиентскому приложению модуль для доступа к функции `InsertBook`. Для этого нам придется несколько изменить интерфейс готового приложения. Можно, конечно, создать еще одну форму, которая позволит добавлять данные в базу данных при помощи Web-сервиса, но с расширением функциональности сервиса, нам пришлось бы создавать все больше и больше форм, что может быть достаточно неудобным с точки зрения интерфейса пользователя. Поэтому попробуем уместить все необходимое на одной форме.

Для этого несколько уменьшим размеры таблицы, в которой отображается информация из базы данных, а на освободившееся место отбуксируем компонент `GroupBox` и уже в нем разместим пять полей ввода `TextBox` и одну кнопку `Button`. В качестве текста, отображаемого в полях ввода по умолчанию, мы напишем наименования полей таблицы, которые будут соответствовать этим полям ввода. А кнопка при нажатии на нее будет вызывать функцию Web-сервиса `InsertBook`.

Полностью весь дизайн и функциональность нового клиентского приложения приведены в листинге 6.13.

Листинг 6.13

```
Public Class Form1
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub
```

```
  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub
  Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents DsBooks1 As dataclient.localhost1.DSBooks
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox6 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox7 As System.Windows.Forms.TextBox

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.TextBox4 = New System.Windows.Forms.TextBox()
    Me.TextBox5 = New System.Windows.Forms.TextBox()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.TextBox7 = New System.Windows.Forms.TextBox()
    Me.DsBooks1 = New dataclient.localhost1.DSBooks()
    Me.DataGrid1 = New System.Windows.Forms.DataGrid()
```

```
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Button2 = New System.Windows.Forms.Button()
    Me.TextBox3 = New System.Windows.Forms.TextBox()
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.TextBox6 = New System.Windows.Forms.TextBox()
    Me.GroupBox1 = New System.Windows.Forms.GroupBox()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.GroupBox1.SuspendLayout()
    Me.SuspendLayout()
    '
    'TextBox4
    '
    Me.TextBox4.Location = New System.Drawing.Point(16, 72)
    Me.TextBox4.Name = "TextBox4"
    Me.TextBox4.Size = New System.Drawing.Size(160, 20)
    Me.TextBox4.TabIndex = 8
    Me.TextBox4.Text = "Наименование"
    '
    'TextBox5
    '
    Me.TextBox5.Location = New System.Drawing.Point(16, 112)
    Me.TextBox5.Name = "TextBox5"
    Me.TextBox5.Size = New System.Drawing.Size(160, 20)
    Me.TextBox5.TabIndex = 8
    Me.TextBox5.Text = "Издательство"
    '
    'Label1
    '
    Me.Label1.Location = New System.Drawing.Point(16, 15)
    Me.Label1.Name = "Label1"
    Me.Label1.Size = New System.Drawing.Size(72, 23)
    Me.Label1.TabIndex = 2
    Me.Label1.Text = "Цена книг от"
    '
    'Label2
```

```
'
Me.Label2.Location = New System.Drawing.Point(208, 15)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(32, 23)
Me.Label2.TabIndex = 4
Me.Label2.Text = "до"
'
'TextBox7
'
Me.TextBox7.Location = New System.Drawing.Point(16, 192)
Me.TextBox7.Name = "TextBox7"
Me.TextBox7.Size = New System.Drawing.Size(160, 20)
Me.TextBox7.TabIndex = 8
Me.TextBox7.Text = "Год издания"
'
'DsBooks1
'
Me.DsBooks1.DataSetName = "DSBooks"
Me.DsBooks1.Locale = New System.Globalization.CultureInfo("en-US")
Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
'
'DataGrid1
'
Me.DataGrid1.DataMember = "BOOKS"
Me.DataGrid1.DataSource = Me.DsBooks1
Me.DataGrid1.Location = New System.Drawing.Point(16, 96)
Me.DataGrid1.Name = "DataGrid1"
Me.DataGrid1.PreferredColumnWidth = 150
Me.DataGrid1.Size = New System.Drawing.Size(392, 192)
Me.DataGrid1.TabIndex = 0
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(24, 48)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(168, 23)
Me.Button1.TabIndex = 1
Me.Button1.Text = "Получить данные"
```

```
'
'Button2
'
Me.Button2.Location = New System.Drawing.Point(8, 232)
Me.Button2.Name = "Button2"
Me.Button2.Size = New System.Drawing.Size(184, 23)
Me.Button2.TabIndex = 6
Me.Button2.Text = "Добавить информацию"
'
'TextBox3
'
Me.TextBox3.Location = New System.Drawing.Point(16, 32)
Me.TextBox3.Name = "TextBox3"
Me.TextBox3.Size = New System.Drawing.Size(160, 20)
Me.TextBox3.TabIndex = 7
Me.TextBox3.Text = "ISBN"
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(248, 16)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.TabIndex = 5
Me.TextBox2.Text = "400"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(96, 16)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.TabIndex = 3
Me.TextBox1.Text = "0"
'
'TextBox6
'
Me.TextBox6.Location = New System.Drawing.Point(16, 152)
Me.TextBox6.Name = "TextBox6"
Me.TextBox6.Size = New System.Drawing.Size(160, 20)
Me.TextBox6.TabIndex = 8
Me.TextBox6.Text = "Цена"
```

```
    '
    'GroupBox1
    '
    Me.GroupBox1.Controls.AddRange(New System.Windows.Forms.Control()
{Me.TextBox7, Me.TextBox6, Me.TextBox5, Me.TextBox4, Me.TextBox3,
Me.Button2})
    Me.GroupBox1.Location = New System.Drawing.Point(424, 24)
    Me.GroupBox1.Name = "GroupBox1"
    Me.GroupBox1.Size = New System.Drawing.Size(200, 264)
    Me.GroupBox1.TabIndex = 7
    Me.GroupBox1.TabStop = False
    Me.GroupBox1.Text = "Добавить информацию"
    '
    'Form1
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(640, 310)
    Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.GroupBox1, Me.TextBox2, Me.Label2, Me.TextBox1, Me.Label1,
Me.Button1, Me.DataGrid1})
    Me.Name = "Form1"
    Me.Text = "Клиентское приложение"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.GroupBox1.ResumeLayout(False)
    Me.ResumeLayout(False)


  End Sub


#End Region


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New dataclient.localhost1.Service1()
    Dim t1, t2 As Integer
    t1 = Convert.ToInt32(TextBox1.Text)
    t2 = Convert.ToInt32(TextBox2.Text)
    DsBooks1.Merge(serv.GetBooks(t1, t2))
  End Sub
```

```
  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim serv = New dataclient.localhost1.Service1()
    Dim s, isbn, name, publ As String
    Dim bookyear, price As Integer
    isbn = TextBox3.Text
    name = TextBox4.Text
    publ = TextBox5.Text
    price = Convert.ToInt32(TextBox6.Text)
    bookyear = Convert.ToInt32(TextBox7.Text)
    s = serv.InsertBook(isbn, name, price, publ, bookyear)
    DsBooks1.Clear()
    DsBooks1.Merge(serv.GetBooks(0, 10000))
  End Sub
End Class
```

Итак, рассмотрим структуру новой функции. Сначала мы просто получаем информацию, внесенную пользователем в поля текстового ввода. А затем уже вызываем функцию `InsertBook`, передавая ей в качестве параметров только что полученную информацию. После того, как информация о книге добавлена в базу данных, ее новое состояние следует показать пользователю. Для этого нужно обновить набор данных `DsBooks1`. Но если мы просто вызовем еще раз метод `Merge`, то информация из базы данных будет просто добавлена к уже существующей, поэтому информация обо всех книгах, кроме только что введенной, будет отображена два раза. Поэтому сначала требуется очистить набор данных при помощи метода `Clear`, а потом уже заполнять его снова. Для заполнения набора данных мы опять используем результат выполнения функции Web-сервиса `GetBooks`, которой передадим в качестве параметров границы ценового диапазона от нуля до десяти тысяч. В этот диапазон гарантированно попадут все цены, и, следовательно, мы получим всю информацию из базы данных. Внешний вид созданного клиентского приложения показан на рис. 6.14.

Теперь попробуем добавить доступ к новой функциональности Web-сервиса и в клиентскую Web-страницу, созданную при помощи технологии ASP.NET. Но вместо того чтобы добавлять на существующую клиентскую Web-страницу дополнительные органы управления, увеличивая ее размеры и усложняя дизайн, просто разместим на ней гиперссылку на вторую Web-страницу. А уже на новой Web-странице, которой по умолчанию присваивается наименование WebForm2.aspx, будут размещены дополнительные элементы управления. Так как внешний вид разрабатываемой Web-страницы полностью описывается ее HTML-кодом, в листинге 6.14 приводится хранящийся на сервере HTML-код с ASP-инструкциями.

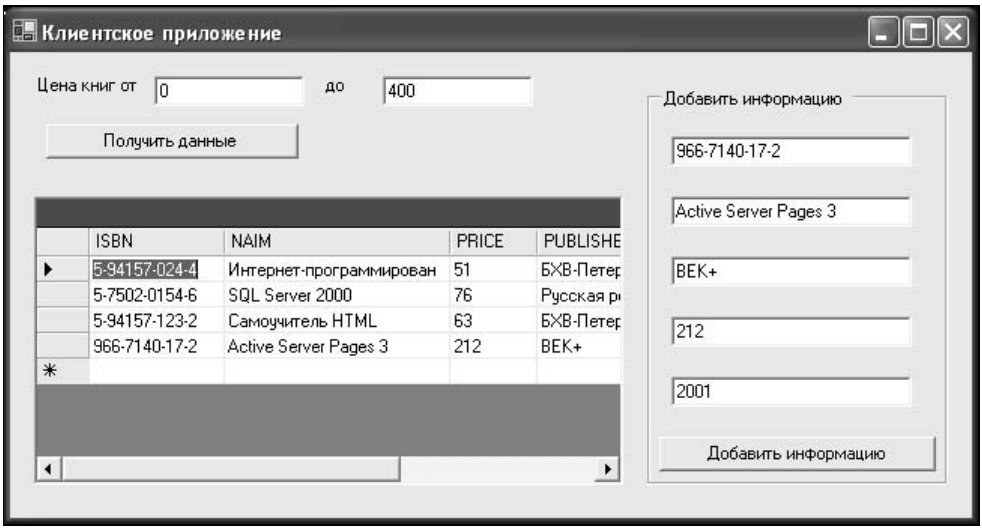

Рис. 6.14. Внешний вид клиентского приложения

Листинг 6.14

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm2.aspx.vb" Inherits="aspdataclient.WebForm2"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title></title>
  <META content="Microsoft Visual Studio .NET 7.0" name="GENERATOR">
  <META content="Visual Basic 7.0" name="CODE_LANGUAGE">
  <META content="JavaScript" name="vs_defaultClientScript">
  <META content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
 </HEAD>
 <BODY MS_POSITIONING="GridLayout">
  <FORM id="Form1" method="post" runat="server">
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 53px; POSITION:
absolute; TOP: 31px" runat="server" Width="171px"
Height="18px">ISBN</asp:Label>
   <asp:TextBox id="TextBox13" style="Z-INDEX: 118; LEFT: 249px;
POSITION: absolute; TOP: 178px" runat="server"></asp:TextBox>
   <asp:TextBox id="TextBox12" style="Z-INDEX: 117; LEFT: 249px;
POSITION: absolute; TOP: 140px" runat="server"></asp:TextBox>
```

```
   <asp:TextBox id="TextBox11" style="Z-INDEX: 116; LEFT: 249px;
POSITION: absolute; TOP: 103px" runat="server"></asp:TextBox>

   <asp:TextBox id="TextBox10" style="Z-INDEX: 115; LEFT: 249px;
POSITION: absolute; TOP: 66px" runat="server"></asp:TextBox>

   <asp:Label id="Label5" style="Z-INDEX: 114; LEFT: 53px; POSITION:
absolute; TOP: 181px" runat="server" Width="171px" Height="18px">Год
выпуска</asp:Label>

   <asp:Label id="Label4" style="Z-INDEX: 113; LEFT: 53px; POSITION:
absolute; TOP: 142px" runat="server" Width="171px"
Height="18px">Издатель</asp:Label>

   <asp:Label id="Label3" style="Z-INDEX: 112; LEFT: 53px; POSITION:
absolute; TOP: 105px" runat="server" Width="171px"
Height="18px">Цена</asp:Label>

   <asp:Label id="Label2" style="Z-INDEX: 111; LEFT: 53px; POSITION:
absolute; TOP: 68px" runat="server" Width="171px"
Height="18px">Наименование</asp:Label>

   <asp:TextBox id="TextBox1" style="Z-INDEX: 110; LEFT: 249px; POSITION:
absolute; TOP: 29px" runat="server"></asp:TextBox>

   <asp:Button id="Button1" style="Z-INDEX: 119; LEFT: 57px; POSITION:
absolute; TOP: 222px" runat="server" Width="221px" Height="24px"
Text="Добавить данные"></asp:Button>

   <asp:Label id="Label6" style="Z-INDEX: 120; LEFT: 56px; POSITION:
absolute; TOP: 262px" runat="server" Width="262px" Height="18px"
ForeColor="Blue"></asp:Label>

  </FORM>

 </BODY>

</HTML>
```

Как можно видеть, на Web-странице размещены пять комплектов, состоящих из связки компонента `Label` и поля ввода `TextBox`. Также на ней размещена кнопка, которая будет отсылать введенные данные Web-сервису, и еще один компонент `Label`, который будет отображать текстовый результат операции, возвращаемый сервисом.

Теперь рассмотрим код класса, реализующего функциональность этой Web-страницы (листинг 6.15).

Листинг 6.15

```
Public Class WebForm2
  Inherits System.Web.UI.Page
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Label2 As System.Web.UI.WebControls.Label
  Protected WithEvents Label3 As System.Web.UI.WebControls.Label
```

```
  Protected WithEvents Label4 As System.Web.UI.WebControls.Label
  Protected WithEvents Label5 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox10 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox11 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox12 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents Label6 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox13 As System.Web.UI.WebControls.TextBox

#Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

  End Sub

  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

#End Region

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New aspdataclient.localhost.Service1()
    Dim s, isbn, name, publ As String
    Dim bookyear, price As Integer
    isbn = TextBox1.Text
    name = TextBox10.Text
    publ = TextBox12.Text
    price = Convert.ToInt32(TextBox11.Text)
```

```
    bookyear = Convert.ToInt32(TextBox13.Text)
    s = serv.InsertBook(isbn, name, price, publ, bookyear)
    Label6.Text = s
  End Sub
End Class
```

Как можно видеть, функциональность клиентской Web-страницы практически не отличается от механизма действия обычного клиентского приложения, и код интересующей нас функции нам уже знаком. Следует отметить лишь тот факт, что в этом примере мы впервые использовали текстовую строку, возвращаемую функцией Web-сервиса `InsertBook`. Эта строка после выполнения операции отображается на Web-странице.

Внешний вид клиентской Web-страницы показан на рис. 6.15.

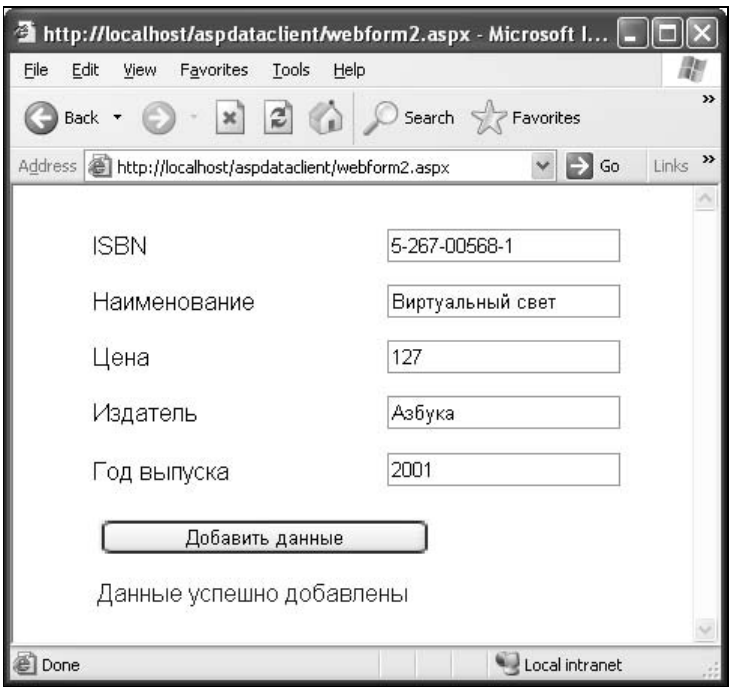

Рис. 6.15. Внешний вид клиентской Web-страницы, предназначенной для добавления информации в базу данных

Изменение и удаление записей

Естественно, при рассмотрении набора операций над базой данных нельзя ограничиваться только добавлением записей в таблицу. Если уж мы начали

предоставлять пользователю возможности редактирования таблицы, следует предоставить ему весь их спектр. То есть, надо дать пользователю Web-сервиса возможность изменять уже существующие записи и удалять записи из базы данных. Для этого нам потребуется соответствующим образом модифицировать используемый Web-сервис.

Реализовать указанные возможности следует при помощи свойств `DeleteCommand` и `UpdateCommand` компонента `SqlDataAdapter`. Однако надо отметить, что эти свойства смогут функционировать только в том случае, если в таблице есть поле, однозначно идентифицирующее каждую запись. То есть поле, используемое в качестве уникального идентификатора. В нашей таблице такого идентификатора нет, поэтому по умолчанию упомянутые свойства функционировать не будут. Более того, они даже не будут доступны для редактирования разработчику.

Впрочем, мы можем использовать в качестве идентификатора поле ISBN. Действительно, ISBN является уникальным идентификатором книги. Хотя в реальной жизни иногда бывают случаи, когда различные книги обладают одним и тем же ISBN-номером, здесь мы этим пренебрежем и укажем, что данное поле будет для нашей таблицы ключевым. Отсюда SQL-скрипт, определяющий структуру основной нашей рабочей таблицы, выглядит следующим образом:

```
CREATE TABLE [dbo].[BOOKS] (
 [ISBN] [varchar] (15) COLLATE Cyrillic_General_CI_AS NOT NULL,
 [NAIM] [varchar] (50) COLLATE Cyrillic_General_CI_AS NULL,
 [PRICE] [int] NULL,
 [PUBLISHER] [varchar] (50) COLLATE Cyrillic_General_CI_AS NULL,
 [BOOKYEAR] [int] NULL
) ON [PRIMARY]
```

Теперь, когда у нас есть поле-идентификатор, можно вернуться в Visual Studio .NET и приступить к модификации Web-сервиса. Начнем мы с создания SQL-запроса, удаляющего запись из таблицы. Легко догадаться, что этот скрипт будет обрабатываться свойством `DeleteCommand`. Соответственно, SQL-скрипт, удаляющий запись из таблицы, будет выглядеть так:

```
DELETE FROM BOOKS WHERE (ISBN = @ISBN)
```

Для изменения записи следует использовать свойство `UpdateCommand`. Соответствующий SQL-скрипт будет выглядеть так:

```
UPDATE  BOOKS
SET  NAIM = @NAIM, PRICE = @PRICE, PUBLISHER = @PUBLISHER,
BOOKYEAR = @BOOKYEAR
WHERE   (ISBN = @ISBN)
```

Теперь, когда заданы все необходимые свойства, можно перейти к обновлению существующего Web-сервиса. Для этого нам достаточно добавить две новые функции с наименованиями `UpdateBook` и `DeleteBook`. Весь код Web-сервиса приведен в листинге 6.16.

Листинг 6.16

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Web Services Designer.
    InitializeComponent()

    'Add your own initialization code after the InitializeComponent()
call

  End Sub

  'Required by the Web Services Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
  Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
  Friend WithEvents SqlDataAdapter1 As
System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents DsBooks1 As DataService.DSBooks
  Friend WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlUpdateCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlDeleteCommand1 As System.Data.SqlClient.SqlCommand
```

```
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    Me.SqlDeleteCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlUpdateCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.DsBooks1 = New DataService.DSBooks()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'SqlConnection1
    '
    Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Books;persist security info=False;user id=sa;" & _
    "workstation id=BHV-9IEMVL4EOER;packet size=4096"
    '
    'SqlDataAdapter1
    '
    Me.SqlDataAdapter1.DeleteCommand = Me.SqlDeleteCommand1
    Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
    Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
    Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "BOOKS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("PRICE", "PRICE"), New
System.Data.Common.DataColumnMapping("PUBLISHER", "PUBLISHER"), New
System.Data.Common.DataColumnMapping("BOOKYEAR", "BOOKYEAR")})})
    Me.SqlDataAdapter1.UpdateCommand = Me.SqlUpdateCommand1
    '
    'SqlDeleteCommand1
    '
    Me.SqlDeleteCommand1.CommandText = "DELETE FROM BOOKS WHERE
(ISBN = @ISBN)"
    Me.SqlDeleteCommand1.Connection = Me.SqlConnection1
    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))
```

```
    '

    'SqlInsertCommand1
    '

    Me.SqlInsertCommand1.CommandText = "INSERT INTO BOOKS(ISBN, NAIM,
PRICE, PUBLISHER, BOOKYEAR) VALUES (@ISBN, @NAIM, @" & _
    "PRICE, @PUBLISHER, @BOOKYEAR); SELECT ISBN, NAIM, PRICE, PUBLISHER,
BOOKYEAR FROM BOOKS WHERE (ISBN = @ISBN)"
    Me.SqlInsertCommand1.Connection = Me.SqlConnection1
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, "ISBN"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, "PRICE"))
    Me.SqlInsertCommand1.Parameters.Add(New

System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, "PUBLISHER"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, "BOOKYEAR"))
    '

    'SqlSelectCommand1
    '

    Me.SqlSelectCommand1.CommandText = "SELECT ISBN, NAIM, PRICE,
PUBLISHER, BOOKYEAR FROM BOOKS WHERE (PRICE >= @p1) AND (PRICE <= @p2)"
    Me.SqlSelectCommand1.Connection = Me.SqlConnection1
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p1", System.Data.SqlDbType.Int, 4,
"PRICE"))
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p2", System.Data.SqlDbType.Int, 4,
"PRICE"))
    '

    'SqlUpdateCommand1
    '

    Me.SqlUpdateCommand1.CommandText = "UPDATE BOOKS SET NAIM = @NAIM,
PRICE = @PRICE, PUBLISHER = @PUBLISHER, BOOKYEAR = @BOOKYEAR WHERE
(ISBN = @ISBN)"
    Me.SqlUpdateCommand1.Connection = Me.SqlConnection1
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
```

```
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, "PRICE"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, "PUBLISHER"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, "BOOKYEAR"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()

  End Sub

  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub

#End Region

  <WebMethod()> Public Function GetBooks(ByVal pr1 As Integer, ByVal pr2
As Integer) As DSBooks
    Dim DS = New DSBooks()
    SqlSelectCommand1.Parameters("@p1").Value = pr1
    SqlSelectCommand1.Parameters("@p2").Value = pr2
```

```
    SqlDataAdapter1.Fill(DS)
    GetBooks = DS
  End Function


  <WebMethod()> Public Function InsertBook(ByVal ISBN As String, ByVal
NAIM As String, ByVal PRICE As Integer, ByVal PUBLISHER As String, ByVal
BOOKYEAR As Integer) As String
    Dim DS = New DSBooks()
    SqlInsertCommand1.Parameters("@ISBN").Value = ISBN
    SqlInsertCommand1.Parameters("@NAIM").Value = NAIM
    SqlInsertCommand1.Parameters("@PRICE").Value = PRICE
    SqlInsertCommand1.Parameters("@PUBLISHER").Value = PUBLISHER
    SqlInsertCommand1.Parameters("@BOOKYEAR").Value = BOOKYEAR
    SqlConnection1.Open()
    SqlInsertCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    InsertBook = "Данные успешно добавлены"
  End Function
  <WebMethod()> Public Function UpdateBook(ByVal ISBN As String, ByVal
NAIM As String, ByVal PRICE As Integer, ByVal PUBLISHER As String, ByVal
BOOKYEAR As Integer) As String
    SqlUpdateCommand1.Parameters("@ISBN").Value = ISBN
    SqlUpdateCommand1.Parameters("@NAIM").Value = NAIM
    SqlUpdateCommand1.Parameters("@PRICE").Value = PRICE
    SqlUpdateCommand1.Parameters("@PUBLISHER").Value = PUBLISHER
    SqlUpdateCommand1.Parameters("@BOOKYEAR").Value = BOOKYEAR
    SqlConnection1.Open()
    SqlUpdateCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    UpdateBook = "Запись изменена"
  End Function
  <WebMethod()> Public Function DeleteBook(ByVal ISBN As String) As
String
    SqlDeleteCommand1.Parameters("@ISBN").Value = ISBN
    SqlConnection1.Open()
    SqlDeleteCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    DeleteBook = "Запись удалена успешно"
  End Function
End Class
```

Итак, добавление функциональности к Web-сервису не представляется особенно сложным. Все функции изменения базы данных возвращают текстовое сообщение о проведенной операции. Конечно, мы не устанавливаем процедур обработки исключительных ситуаций, но это всего лишь тестовый пример, поэтому примем, что все наши операции с базой данных будут выполняться идеально. В реальной жизни это будет так далеко не всегда, поэтому в рабочих приложениях обязательно следует устанавливать блоки обработки исключений.

Теперь перейдем к созданию клиентского приложения. У нас уже есть пространство на форме, где располагаются пять полей текстового ввода. Добавим к этой группе еще две кнопки, которые будут вызывать две новые функции Web-сервиса. Также стоит разместить на форме строку статуса, в которой будем отображать текстовые сообщения, возвращаемые функциями сервиса. Таким образом мы сможем информировать пользователя о результатах выполнения операций.

Полный код клиентского приложения приведен в листинге 6.17.

Листинг 6.17

```
Public Class Form1
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
```

```
    End If
    MyBase.Dispose(disposing)
  End Sub
  Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents DsBooks1 As dataclient.localhost.DSBooks
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox6 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox7 As System.Windows.Forms.TextBox

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.Container

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  Friend WithEvents Button3 As System.Windows.Forms.Button
  Friend WithEvents Button4 As System.Windows.Forms.Button
  Friend WithEvents StatusBar1 As System.Windows.Forms.StatusBar
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.TextBox4 = New System.Windows.Forms.TextBox()
    Me.TextBox5 = New System.Windows.Forms.TextBox()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.TextBox7 = New System.Windows.Forms.TextBox()
    Me.DsBooks1 = New dataclient.localhost.DSBooks()
    Me.DataGrid1 = New System.Windows.Forms.DataGrid()
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Button2 = New System.Windows.Forms.Button()
    Me.TextBox3 = New System.Windows.Forms.TextBox()
```

```
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.TextBox6 = New System.Windows.Forms.TextBox()
    Me.GroupBox1 = New System.Windows.Forms.GroupBox()
    Me.Button3 = New System.Windows.Forms.Button()
    Me.Button4 = New System.Windows.Forms.Button()
    Me.StatusBar1 = New System.Windows.Forms.StatusBar()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.GroupBox1.SuspendLayout()
    Me.SuspendLayout()
    '
    'TextBox4
    '
    Me.TextBox4.Location = New System.Drawing.Point(16, 56)
    Me.TextBox4.Name = "TextBox4"
    Me.TextBox4.Size = New System.Drawing.Size(160, 20)
    Me.TextBox4.TabIndex = 8
    Me.TextBox4.Text = "Наименование"
    '
    'TextBox5
    '
    Me.TextBox5.Location = New System.Drawing.Point(16, 88)
    Me.TextBox5.Name = "TextBox5"
    Me.TextBox5.Size = New System.Drawing.Size(160, 20)
    Me.TextBox5.TabIndex = 8
    Me.TextBox5.Text = "Издательство"
    '
    'Label1
    '
    Me.Label1.Location = New System.Drawing.Point(16, 15)
    Me.Label1.Name = "Label1"
    Me.Label1.Size = New System.Drawing.Size(72, 23)
    Me.Label1.TabIndex = 2
    Me.Label1.Text = "Цена книг от"
    '
    'Label2
```

```
    '
    Me.Label2.Location = New System.Drawing.Point(208, 15)
    Me.Label2.Name = "Label2"
    Me.Label2.Size = New System.Drawing.Size(32, 23)
    Me.Label2.TabIndex = 4
    Me.Label2.Text = "до"
    '
    'TextBox7
    '
    Me.TextBox7.Location = New System.Drawing.Point(16, 152)
    Me.TextBox7.Name = "TextBox7"
    Me.TextBox7.Size = New System.Drawing.Size(160, 20)
    Me.TextBox7.TabIndex = 8
    Me.TextBox7.Text = "Год издания"
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("en-US")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    '
    'DataGrid1
    '
    Me.DataGrid1.DataMember = "BOOKS"
    Me.DataGrid1.DataSource = Me.DsBooks1
    Me.DataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText
    Me.DataGrid1.Location = New System.Drawing.Point(16, 96)
    Me.DataGrid1.Name = "DataGrid1"
    Me.DataGrid1.PreferredColumnWidth = 150
    Me.DataGrid1.Size = New System.Drawing.Size(392, 192)
    Me.DataGrid1.TabIndex = 0
    '
    'Button1
    '
    Me.Button1.Location = New System.Drawing.Point(24, 48)
    Me.Button1.Name = "Button1"
    Me.Button1.Size = New System.Drawing.Size(168, 23)
    Me.Button1.TabIndex = 1
```

```
Me.Button1.Text = "Получить данные"
'
'Button2
'
Me.Button2.Location = New System.Drawing.Point(8, 176)
Me.Button2.Name = "Button2"
Me.Button2.Size = New System.Drawing.Size(184, 23)
Me.Button2.TabIndex = 6
Me.Button2.Text = "Добавить информацию"
'
'TextBox3
'
Me.TextBox3.Location = New System.Drawing.Point(16, 24)
Me.TextBox3.Name = "TextBox3"
Me.TextBox3.Size = New System.Drawing.Size(160, 20)
Me.TextBox3.TabIndex = 7
Me.TextBox3.Text = "ISBN"
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(248, 16)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.TabIndex = 5
Me.TextBox2.Text = "400"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(96, 16)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.TabIndex = 3
Me.TextBox1.Text = "0"
'
'TextBox6
'
Me.TextBox6.Location = New System.Drawing.Point(16, 120)
Me.TextBox6.Name = "TextBox6"
Me.TextBox6.Size = New System.Drawing.Size(160, 20)
Me.TextBox6.TabIndex = 8
```

```
    Me.TextBox6.Text = "Цена"
    '
    'GroupBox1
    '
    Me.GroupBox1.Controls.AddRange(New System.Windows.Forms.Control()
{Me.Button4, Me.Button3, Me.TextBox7, Me.TextBox6, Me.TextBox5,
Me.TextBox4, Me.TextBox3, Me.Button2})
    Me.GroupBox1.Location = New System.Drawing.Point(424, 8)
    Me.GroupBox1.Name = "GroupBox1"
    Me.GroupBox1.Size = New System.Drawing.Size(200, 272)
    Me.GroupBox1.TabIndex = 7
    Me.GroupBox1.TabStop = False
    Me.GroupBox1.Text = "Операции с базой данных"
    '
    'Button3
    '
    Me.Button3.Location = New System.Drawing.Point(8, 212)
    Me.Button3.Name = "Button3"
    Me.Button3.Size = New System.Drawing.Size(184, 23)
    Me.Button3.TabIndex = 9
    Me.Button3.Text = "Удалить запись"
    '
    'Button4
    '
    Me.Button4.Location = New System.Drawing.Point(8, 248)
    Me.Button4.Name = "Button4"
    Me.Button4.Size = New System.Drawing.Size(184, 23)
    Me.Button4.TabIndex = 10
    Me.Button4.Text = "Изменить запись"
    '
    'StatusBar1
    '
    Me.StatusBar1.Location = New System.Drawing.Point(0, 288)
    Me.StatusBar1.Name = "StatusBar1"
    Me.StatusBar1.Size = New System.Drawing.Size(640, 22)
    Me.StatusBar1.TabIndex = 8
    '
    'Form1
    '
```

```
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(640, 310)
    Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.StatusBar1, Me.GroupBox1, Me.TextBox2, Me.Label2, Me.TextBox1,
Me.Label1, Me.Button1, Me.DataGrid1})
    Me.Name = "Form1"
    Me.Text = "Клиентское приложение"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.GroupBox1.ResumeLayout(False)
    Me.ResumeLayout(False)


  End Sub


#End Region


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim t1, t2 As Integer
    t1 = Convert.ToInt32(TextBox1.Text)
    t2 = Convert.ToInt32(TextBox2.Text)
    DsBooks1.Clear()
    DsBooks1.Merge(serv.GetBooks(t1, t2))
  End Sub


  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s, isbn, name, publ As String
    Dim bookyear, price As Integer
    isbn = TextBox3.Text
    name = TextBox4.Text
    publ = TextBox5.Text
    price = Convert.ToInt32(TextBox6.Text)
    bookyear = Convert.ToInt32(TextBox7.Text)
    s = serv.InsertBook(isbn, name, price, publ, bookyear)
    StatusBar1.Text = s
```

```
    DsBooks1.Clear()
    DsBooks1.Merge(serv.GetBooks(0, 10000))
  End Sub

  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s, isbn As String
    isbn = TextBox3.Text
    s = serv.DeleteBook(isbn)
    StatusBar1.Text = s
  End Sub

  Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s, isbn, name, publ As String
    Dim bookyear, price As Integer
    isbn = TextBox3.Text
    name = TextBox4.Text
    publ = TextBox5.Text
    price = Convert.ToInt32(TextBox6.Text)
    bookyear = Convert.ToInt32(TextBox7.Text)
    s = serv.UpdateBook(isbn, name, price, publ, bookyear)
    StatusBar1.Text = s
  End Sub
End Class
```

Следует отметить, что добавление новых возможностей в клиентское приложение уже не должно вызывать каких-либо проблем. Все действительно просто. Поэтому не будем занимать драгоценное пространство книги описанием процедуры создания Web-страницы, выступающей в роли клиента. Пусть это будет неким заданием для самостоятельного выполнения. В конце концов, все необходимые навыки для этого у вас уже есть, не так ли?

Прямое редактирование набора данных

Если вы работали со всеми клиентскими приложениями, описанными в предыдущих разделах этой главы, то должны были заметить основную проблему: изменять информацию, входящую в состав базы данных, неудобно. Действительно, вносить информацию необходимо в различные текстовые

поля, за один раз можно произвести только одну операцию, необходимо тщательно следить, чтобы при удалении и изменении записи ISBN был введен верно — все это не добавляет удобства в работе.

С другой стороны, в клиентском приложении расположена таблица, которая не только отображает данные из базы данных, но и позволяет изменять их прямо в приложении. Одна проблема — изменения в информации, которые пользователь произведет в этой таблице, никак не повлияют на содержимое базы данных. Однако у разработчика есть возможность синхронизировать содержимое клиентского набора данных и таблицы из базы данных, то есть не только отображать информацию как набор данных, но и изменять содержимое базы данных, используя для этого клиентский набор данных.

Итак, основная задача этого раздела — узнать, как можно использовать набор данных, отображающийся в клиентском приложении, в качестве источника данных для удаленной базы данных. Как известно, клиентское приложение получает набор данных от Web-сервиса. Тот, в свою очередь, порождает набор данных от объекта `SqlDataAdapter`. А этот объект позволяет модифицировать базу данных, с которой он связан. До сих пор для этого мы использовали его свойства `InsertComand`, `UpdateCommand` и `DeleteComand`. Однако `SqlDataAdapter` обладает методом `Update`, которому может быть передан набор данных. То есть, мы можем из клиентского приложения передавать Web-сервису набор данных, который пользователь изменит в отображенной таблице, а на стороне Web-сервиса при помощи упомянутого метода этот набор будет перенесен в базу данных.

Для выполнения такой задачи нам придется еще раз изменить наш Web-сервис. Код его последней версии приведен в листинге 6.18.

Листинг 6.18

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Web Services Designer.
    InitializeComponent()
```

```
    'Add your own initialization code after the InitializeComponent()
call

  End Sub

  'Required by the Web Services Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
  Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
  Friend WithEvents SqlDataAdapter1 As
System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents DsBooks1 As DataService.DSBooks
  Friend WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlUpdateCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlDeleteCommand1 As System.Data.SqlClient.SqlCommand
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    Me.SqlDeleteCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlUpdateCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.DsBooks1 = New DataService.DSBooks()
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'SqlConnection1
    '
    Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Books;persist security info=False;user id=sa; workstation
id=BHV-9IEMVL4EOER;packet size=4096"
    '
    'SqlDataAdapter1
    '
    Me.SqlDataAdapter1.DeleteCommand = Me.SqlDeleteCommand1
```

```
    Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
    Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
    Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "BOOKS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("PRICE", "PRICE"), New
System.Data.Common.DataColumnMapping("PUBLISHER", "PUBLISHER"), New
System.Data.Common.DataColumnMapping("BOOKYEAR", "BOOKYEAR")})})
    Me.SqlDataAdapter1.UpdateCommand = Me.SqlUpdateCommand1
    '
    'SqlDeleteCommand1
    '
    Me.SqlDeleteCommand1.CommandText = "DELETE FROM BOOKS WHERE
(ISBN = @ISBN)"
    Me.SqlDeleteCommand1.Connection = Me.SqlConnection1
    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))
    '
    'SqlInsertCommand1
    '
    Me.SqlInsertCommand1.CommandText = "INSERT INTO BOOKS(ISBN, NAIM,
PRICE, PUBLISHER, BOOKYEAR) VALUES (@ISBN, @NAIM, @PRICE, @PUBLISHER,
@BOOKYEAR); SELECT ISBN, NAIM, PRICE, PUBLISHER, BOOKYEAR FROM BOOKS
WHERE (ISBN = @ISBN)"
    Me.SqlInsertCommand1.Connection = Me.SqlConnection1
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, "ISBN"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, "PRICE"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, "PUBLISHER"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, "BOOKYEAR"))
    '
```

```
    'SqlSelectCommand1
    '
    Me.SqlSelectCommand1.CommandText = "SELECT ISBN, NAIM, PRICE,
PUBLISHER, BOOKYEAR FROM BOOKS WHERE (PRICE >= @p1) AND (PRICE <= @p2)"
    Me.SqlSelectCommand1.Connection = Me.SqlConnection1
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p1", System.Data.SqlDbType.Int, 4,
"PRICE"))
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@p2", System.Data.SqlDbType.Int, 4,
"PRICE"))
    '
    'SqlUpdateCommand1
    '
    Me.SqlUpdateCommand1.CommandText = "UPDATE BOOKS SET NAIM = @NAIM,
PRICE = @PRICE, PUBLISHER = @PUBLISHER, BOOKYEAR = @BOOKYEAR WHERE
(ISBN = @ISBN)"
    Me.SqlUpdateCommand1.Connection = Me.SqlConnection1
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, "PRICE"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, "PUBLISHER"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, "BOOKYEAR"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))
    '
    'DsBooks1
    '
    Me.DsBooks1.DataSetName = "DSBooks"
    Me.DsBooks1.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks1.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    CType(Me.DsBooks1,
System.ComponentModel.ISupportInitialize).EndInit()


  End Sub
```

```
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub


#End Region


  <WebMethod()> Public Function GetBooks(ByVal pr1 As Integer, ByVal pr2
As Integer) As DSBooks
    Dim DS = New DSBooks()
    SqlSelectCommand1.Parameters("@p1").Value = pr1
    SqlSelectCommand1.Parameters("@p2").Value = pr2
    SqlDataAdapter1.Fill(DS)
    GetBooks = DS
  End Function


  <WebMethod()> Public Function InsertBook(ByVal ISBN As String, ByVal
NAIM As String, ByVal PRICE As Integer, ByVal PUBLISHER As String, ByVal
BOOKYEAR As Integer) As String
    Dim DS = New DSBooks()
    SqlInsertCommand1.Parameters("@ISBN").Value = ISBN
    SqlInsertCommand1.Parameters("@NAIM").Value = NAIM
    SqlInsertCommand1.Parameters("@PRICE").Value = PRICE
    SqlInsertCommand1.Parameters("@PUBLISHER").Value = PUBLISHER
    SqlInsertCommand1.Parameters("@BOOKYEAR").Value = BOOKYEAR
    SqlConnection1.Open()
    SqlInsertCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    InsertBook = "Данные успешно добавлены"
  End Function
  <WebMethod()> Public Function UpdateBook(ByVal ISBN As String, ByVal
NAIM As String, ByVal PRICE As Integer, ByVal PUBLISHER As String, ByVal
BOOKYEAR As Integer) As String
    SqlUpdateCommand1.Parameters("@ISBN").Value = ISBN
```

```
    SqlUpdateCommand1.Parameters("@NAIM").Value = NAIM
    SqlUpdateCommand1.Parameters("@PRICE").Value = PRICE
    SqlUpdateCommand1.Parameters("@PUBLISHER").Value = PUBLISHER
    SqlUpdateCommand1.Parameters("@BOOKYEAR").Value = BOOKYEAR
    SqlConnection1.Open()
    SqlUpdateCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    UpdateBook = "Запись изменена"
  End Function
  <WebMethod()> Public Function DeleteBook(ByVal ISBN As String) As
String
    SqlDeleteCommand1.Parameters("@ISBN").Value = ISBN
    SqlConnection1.Open()
    SqlDeleteCommand1.ExecuteNonQuery()
    SqlConnection1.Close()
    DeleteBook = "Запись удалена успешно"
  End Function
  <WebMethod()> Public Function UpdateTable(ByVal DS As DSBooks) As
String
    SqlDataAdapter1.Update(DS)
    UpdateTable = "Данные приняты"
  End Function
End Class
```

Как можно видеть, код добавленной функции `UpdateTable` чрезвычайно прост. Функция принимает набор данных объявленного типа `DSBooks` и передает его методу `SqlDataAdapter1.Update` в качестве параметра. Все, более ничего не требуется делать. Компонент `SqlDataAdapter`, руководствуясь полученным набором данных, самостоятельно обновит содержимое базы данных. Остается только возвратить текстовую строку, сигнализирующую, что функция успешно завершила свою работу.

Естественно, помимо Web-сервиса требуется соответствующим образом изменить клиентское приложение. Так как таблица, в которой отображаются и редактируются данные, у нас уже есть, остается лишь добавить кнопку, при нажатии на которую будет вызываться только что рассмотренная нами функция Web-сервиса. Рассмотрим код этого приложения, который приведен в листинге 6.19.

Листинг 6.19

```
Public Class Form1
  Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub
  Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents DsBooks1 As dataclient.localhost.DSBooks
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox6 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox7 As System.Windows.Forms.TextBox

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.Container
```

```
  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  Friend WithEvents Button3 As System.Windows.Forms.Button
  Friend WithEvents Button4 As System.Windows.Forms.Button
  Friend WithEvents StatusBar1 As System.Windows.Forms.StatusBar
  Friend WithEvents Button5 As System.Windows.Forms.Button
  Friend WithEvents DsBooks2 As dataclient.localhost.DSBooks
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.TextBox4 = New System.Windows.Forms.TextBox()
    Me.TextBox5 = New System.Windows.Forms.TextBox()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.TextBox7 = New System.Windows.Forms.TextBox()
    Me.DataGrid1 = New System.Windows.Forms.DataGrid()
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Button2 = New System.Windows.Forms.Button()
    Me.TextBox3 = New System.Windows.Forms.TextBox()
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.TextBox6 = New System.Windows.Forms.TextBox()
    Me.GroupBox1 = New System.Windows.Forms.GroupBox()
    Me.Button4 = New System.Windows.Forms.Button()
    Me.Button3 = New System.Windows.Forms.Button()
    Me.StatusBar1 = New System.Windows.Forms.StatusBar()
    Me.Button5 = New System.Windows.Forms.Button()
    Me.DsBooks2 = New dataclient.localhost.DSBooks()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.GroupBox1.SuspendLayout()
    CType(Me.DsBooks2,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    'TextBox4
    '
    Me.TextBox4.Location = New System.Drawing.Point(16, 60)
    Me.TextBox4.Name = "TextBox4"
```

```
Me.TextBox4.Size = New System.Drawing.Size(160, 20)
Me.TextBox4.TabIndex = 8
Me.TextBox4.Text = "Наименование"
'
'TextBox5
'
Me.TextBox5.Location = New System.Drawing.Point(16, 96)
Me.TextBox5.Name = "TextBox5"
Me.TextBox5.Size = New System.Drawing.Size(160, 20)
Me.TextBox5.TabIndex = 8
Me.TextBox5.Text = "Издательство"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(16, 15)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(72, 23)
Me.Label1.TabIndex = 2
Me.Label1.Text = "Цена книг от"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(208, 15)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(32, 23)
Me.Label2.TabIndex = 4
Me.Label2.Text = "до"
'
'TextBox7
'
Me.TextBox7.Location = New System.Drawing.Point(16, 168)
Me.TextBox7.Name = "TextBox7"
Me.TextBox7.Size = New System.Drawing.Size(160, 20)
Me.TextBox7.TabIndex = 8
Me.TextBox7.Text = "Год издания"
'
'DataGrid1
'
```

```
    Me.DataGrid1.DataMember = "BOOKS"
    Me.DataGrid1.DataSource = Me.DsBooks2
    Me.DataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText
    Me.DataGrid1.Location = New System.Drawing.Point(16, 96)
    Me.DataGrid1.Name = "DataGrid1"
    Me.DataGrid1.PreferredColumnWidth = 150
    Me.DataGrid1.Size = New System.Drawing.Size(392, 192)
    Me.DataGrid1.TabIndex = 0
    '
    'Button1
    '
    Me.Button1.Location = New System.Drawing.Point(24, 48)
    Me.Button1.Name = "Button1"
    Me.Button1.Size = New System.Drawing.Size(168, 23)
    Me.Button1.TabIndex = 1
    Me.Button1.Text = "Получить данные"
    '
    'Button2
    '
    Me.Button2.Location = New System.Drawing.Point(8, 204)
    Me.Button2.Name = "Button2"
    Me.Button2.Size = New System.Drawing.Size(184, 23)
    Me.Button2.TabIndex = 6
    Me.Button2.Text = "Добавить информацию"
    '
    'TextBox3
    '
    Me.TextBox3.Location = New System.Drawing.Point(16, 24)
    Me.TextBox3.Name = "TextBox3"
    Me.TextBox3.Size = New System.Drawing.Size(160, 20)
    Me.TextBox3.TabIndex = 7
    Me.TextBox3.Text = "ISBN"
    '
    'TextBox2
    '
    Me.TextBox2.Location = New System.Drawing.Point(248, 16)
    Me.TextBox2.Name = "TextBox2"
    Me.TextBox2.TabIndex = 5
```

```
    Me.TextBox2.Text = "400"
    '
    'TextBox1
    '
    Me.TextBox1.Location = New System.Drawing.Point(96, 16)
    Me.TextBox1.Name = "TextBox1"
    Me.TextBox1.TabIndex = 3
    Me.TextBox1.Text = "0"
    '
    'TextBox6
    '
    Me.TextBox6.Location = New System.Drawing.Point(16, 132)
    Me.TextBox6.Name = "TextBox6"
    Me.TextBox6.Size = New System.Drawing.Size(160, 20)
    Me.TextBox6.TabIndex = 8
    Me.TextBox6.Text = "Цена"
    '
    'GroupBox1
    '
    Me.GroupBox1.Controls.AddRange(New System.Windows.Forms.Control()
{Me.Button4, Me.Button3, Me.TextBox7, Me.TextBox6, Me.TextBox5,
Me.TextBox4, Me.TextBox3, Me.Button2})
    Me.GroupBox1.Location = New System.Drawing.Point(424, 8)
    Me.GroupBox1.Name = "GroupBox1"
    Me.GroupBox1.Size = New System.Drawing.Size(200, 320)
    Me.GroupBox1.TabIndex = 7
    Me.GroupBox1.TabStop = False
    Me.GroupBox1.Text = "Операции с базой данных"
    '
    'Button4
    '
    Me.Button4.Location = New System.Drawing.Point(8, 282)
    Me.Button4.Name = "Button4"
    Me.Button4.Size = New System.Drawing.Size(184, 23)
    Me.Button4.TabIndex = 10
    Me.Button4.Text = "Изменить запись"
    '
    'Button3
    '
```

```
    Me.Button3.Location = New System.Drawing.Point(8, 243)
    Me.Button3.Name = "Button3"
    Me.Button3.Size = New System.Drawing.Size(184, 23)
    Me.Button3.TabIndex = 9
    Me.Button3.Text = "Удалить запись"
    '
    'StatusBar1
    '
    Me.StatusBar1.Location = New System.Drawing.Point(0, 344)
    Me.StatusBar1.Name = "StatusBar1"
    Me.StatusBar1.Size = New System.Drawing.Size(640, 22)
    Me.StatusBar1.TabIndex = 8
    '
    'Button5
    '
    Me.Button5.Location = New System.Drawing.Point(16, 304)
    Me.Button5.Name = "Button5"
    Me.Button5.Size = New System.Drawing.Size(192, 23)
    Me.Button5.TabIndex = 9
    Me.Button5.Text = "Записать изменения"
    '
    'DsBooks2
    '
    Me.DsBooks2.DataSetName = "DSBooks"
    Me.DsBooks2.Locale = New System.Globalization.CultureInfo("ru-RU")
    Me.DsBooks2.Namespace = "http://www.tempuri.org/DSBooks.xsd"
    '
    'Form1
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(640, 366)
    Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.Button5,
Me.StatusBar1, Me.GroupBox1, Me.TextBox2, Me.Label2, Me.TextBox1,
Me.Label1, Me.Button1, Me.DataGrid1})
    Me.Name = "Form1"
    Me.Text = "Клиентское приложение"
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.GroupBox1.ResumeLayout(False)
```

```
    CType(Me.DsBooks2,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)


  End Sub


#End Region


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim t1, t2 As Integer
    t1 = Convert.ToInt32(TextBox1.Text)
    t2 = Convert.ToInt32(TextBox2.Text)
    DsBooks2.Clear()
    DsBooks2.Merge(serv.GetBooks(t1, t2))
  End Sub


  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s, isbn, name, publ As String
    Dim bookyear, price As Integer
    isbn = TextBox3.Text
    name = TextBox4.Text
    publ = TextBox5.Text
    price = Convert.ToInt32(TextBox6.Text)
    bookyear = Convert.ToInt32(TextBox7.Text)
    s = serv.InsertBook(isbn, name, price, publ, bookyear)
    StatusBar1.Text = s
    DsBooks2.Clear()
    DsBooks2.Merge(serv.GetBooks(0, 10000))
  End Sub


  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s, isbn As String
    isbn = TextBox3.Text
    s = serv.DeleteBook(isbn)
```

```
    StatusBar1.Text = s
    DsBooks2.Clear()
    DsBooks2.Merge(serv.GetBooks(0, 10000))
  End Sub

  Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s, isbn, name, publ As String
    Dim bookyear, price As Integer
    isbn = TextBox3.Text
    name = TextBox4.Text
    publ = TextBox5.Text
    price = Convert.ToInt32(TextBox6.Text)
    bookyear = Convert.ToInt32(TextBox7.Text)
    s = serv.UpdateBook(isbn, name, price, publ, bookyear)
    StatusBar1.Text = s
    DsBooks2.Clear()
    DsBooks2.Merge(serv.GetBooks(0, 10000))
  End Sub

  Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
    Dim serv = New dataclient.localhost.Service1()
    Dim s As String
    s = serv.UpdateTable(DsBooks2)
    StatusBar1.Text = s
    DsBooks2.Clear()
    DsBooks2.Merge(serv.GetBooks(0, 10000))
  End Sub
End Class
```

Сама структура функции не так уж сложна. Поскольку источник данных для основной таблицы клиентского приложения является обычным набором данных, то его мы и можем передать функции `UpdateTable`. После того как набор данных передается Web-сервису, принимается текстовая строка, возвращаемая функцией Web-сервиса, и этот текст отображается в строке статуса. Затем набор данных отображается в таблице еще раз.

Создание клиентской Web-страницы, использующей новую функцию Web-сервиса, будет много сложнее, чем обычного клиентского приложения. Дело

в том, что Web-страницы изначально "заточены" под отображение информации, и если мы хотим позволить пользователю изменять содержимое таблицы, следует в каждой ячейке разместить поле текстового ввода, а также предусмотреть механизмы удаления и добавления строк в таблицу. Задача вполне выполнима средствами ASP.NET, но ее реализация явно выходит за пределы тематики этой книги.

XML-документ в качестве источника данных

Как известно, XML теснейшим образом интегрирован в технологию Microsoft .NET. Одним из предназначений XML-документов может быть хранение данных. Так как все данные, в конечном счете, хранятся все-таки в виде текста, их передача через брандмауэры тоже не представляет труда. Конечно, XML-файлы никоим образом не могут заменить стандартные системы управления базами данных, однако хранение в них небольших объемов данных вполне оправданно. Более того, весьма часто XML-файлы могут содержать данные, предоставляемые сторонними пользователями.

Так или иначе, в этом разделе главы мы рассмотрим использование XML-файлов в качестве источника данных. Строго говоря, этот раздел не должен входить в состав главы, рассказывающей об ADO.NET, так как доступ к XML-файлам осуществляется без помощи ADO.NET. Но если уж мы говорим здесь о работе с данными, то стоит разобрать и этот механизм работы.

Для начала рассмотрим сам XML-файл, который будет служить в качестве источника данных. Для обычного примера мы можем взять файл с не самой сложной структурой. Итак, содержимое XML-файла, используемого в нашем примере, будет выглядеть следующим образом:

```
<Document>
  <Row>
    <Column1>1</Column1>
    <Column2>2</Column2>
    <Column3>Text1</Column3>
    <Column4>Text2</Column4>
  </Row>
  <Row>
    <Column1>3</Column1>
    <Column2>4</Column2>
    <Column3>Text3</Column3>
    <Column4>Text4</Column4>
  </Row>
</Document>
```

Как можно видеть, структура используемого XML-файла более чем прозрачна. Несмотря на то, что этот файл никак нельзя назвать валидным (valid) или хорошо оформленным (well-formed), он, тем не менее, обладает правильной структурой XML-документа. Всю информацию, содержащуюся в файле, можно отобразить в таблице, содержащей две строки по четыре ячейки.

Теперь перейдем к созданию Web-сервиса, который будет использовать этот файл в качестве источника данных. Если быть более точным, наш сервис будет включать единственную функцию, возвращающую обычный набор данных (`DataSet`), содержащий информацию, взятую из XML-файла.

Итак, как обычно, создадим новый проект Web-сервиса. На его страницу отбуксируем один компонент `DataSet` с вкладки **Data**. При этом будет активировано диалоговое окно **AddDataset**, в котором разработчику предлагается указать тип создаваемого набора данных. Так как в нашем случае XML-файл может содержать данные любой структуры, то для набора данных нельзя заранее указать структуру. Следовательно, в этом диалоговом окне разработчик должен выбрать переключатель **Untyped dataset**.

Теперь, когда у нас есть набор данных, следует написать сам Web-сервис. Его код приведен в листинге 6.20.

Листинг 6.20

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
  Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Web Services Designer.
    InitializeComponent()

    'Add your own initialization code after the InitializeComponent()
call

  End Sub
```

```
  'Required by the Web Services Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
  Friend WithEvents DataSet1 As System.Data.DataSet
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DataSet1 = New System.Data.DataSet()
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'DataSet1
    '
    Me.DataSet1.DataSetName = "NewDataSet"
    Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU")
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()

  End Sub

  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub

#End Region


  <WebMethod()> Public Function XmlData() As DataSet
    Dim FS As IO.FileStream
    Dim Reader As IO.StreamReader
```

```
    FS = New IO.FileStream(Server.MapPath("data1.xml"), IO.FileMode.Open,
IO.FileAccess.Read)
    Reader = New IO.StreamReader(FS)
    DataSet1.ReadXml(Reader)
    FS.Close()
    XmlData = DataSet1
  End Function

End Class
```

Рассмотрим структуру работы функции Web-сервиса. Нам потребуется один экземпляр класса `DataSet`, переменная `FS`, которая позволяет работать с файлами, и переменная `Reader`, предназначенная для чтения потоков информации из файла.

Файл Data1.xml открывается в конструкторе объекта `FileStream`, причем, как легко заметить, открывается только на чтение. Затем при помощи метода `ReadXml` мы считываем данные из потока, связанного с XML-файлом. Этого достаточно. Теперь все данные уже находятся в экземпляре объекта `DataSet`. Остается закрыть файл, с которым связана переменная `FS`, и возвратить набор данных.

Стоит обратить внимание на то, как выводится результат действия сервиса. Возвращаемый XML-документ выглядит следующим образом:

```
  <?xml version="1.0" encoding="utf-8" ?>
<DataSet xmlns="http://tempuri.org/">
<xs:schema id="Document" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
<xs:element name="Document" msdata:IsDataSet="true"
msdata:Locale="ru-RU">
<xs:complexType>
<xs:choice maxOccurs="unbounded">
<xs:element name="Row">
<xs:complexType>
<xs:sequence>
  <xs:element name="Column1" type="xs:string" minOccurs="0" />
  <xs:element name="Column2" type="xs:string" minOccurs="0" />
  <xs:element name="Column3" type="xs:string" minOccurs="0" />
  <xs:element name="Column4" type="xs:string" minOccurs="0" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
```

```
  </xs:choice>
  </xs:complexType>
  </xs:element>
  </xs:schema>
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
<Document xmlns="">
<Row diffgr:id="Row1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
  <Column1>1</Column1>
  <Column2>2</Column2>
  <Column3>Text1</Column3>
  <Column4>Text2</Column4>
  </Row>
<Row diffgr:id="Row2" msdata:rowOrder="1" diffgr:hasChanges="inserted">
  <Column1>3</Column1>
  <Column2>4</Column2>
  <Column3>Text3</Column3>
  <Column4>Text4</Column4>
  </Row>
  </Document>
  </diffgr:diffgram>
  </DataSet>
```

Из этого фрагмента кода видно, что передаваемая информация объявлена как текстовые строки. Что ж, это логично. Сам XML-документ является обычным текстом, поэтому нетипизированные данные также считаются строковыми. Если принимающее приложение "знает", какие типы соответствуют различным тегам, то оно сможет соответствующим образом конвертировать данные.

Теперь рассмотрим пример клиентской Web-страницы. На ней мы разместим компонент — `DataGrid`, в котором будет отображаться полученная от Web-сервиса информация, и компонент `DataSet`. Естественно, искомый `DataSet` будет служить источником данных для таблицы. HTML-код этой Web-страницы, хранящийся на сервере, приведен в листинге 6.21.

Листинг 6.21

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="XMLWebClient.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
```

```
 <HEAD>
  <title>WebForm1</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:DataGrid id="DataGrid1" style="Z-INDEX: 101; LEFT: 26px;
POSITION: absolute; TOP: 25px" runat="server" Width="347px"
Height="178px" DataSource="<%# DataSet1 %>">
   </asp:DataGrid>
  </form>
 </body>
</HTML>
```

Так как `DataSet` не является визуальным компонентом, то он, соответственно, не объявлен в HTML-коде. Он объявляется в коде класса, реализующего функциональность этой Web-страницы (листинг 6.22).

Листинг 6.22

```
Public Class WebForm1
  Inherits System.Web.UI.Page
  Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
  Protected WithEvents DataSet1 As System.Data.DataSet


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DataSet1 = New System.Data.DataSet()
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'DataSet1
    '
    Me.DataSet1.DataSetName = "NewDataSet"
    Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU")
```

```
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()


  End Sub


  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub


#End Region


  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim serv = New XMLWebClient.localhost.Service1()
    DataSet1.Merge(serv.XmlData)
    DataGrid1.DataBind()
  End Sub


End Class
```

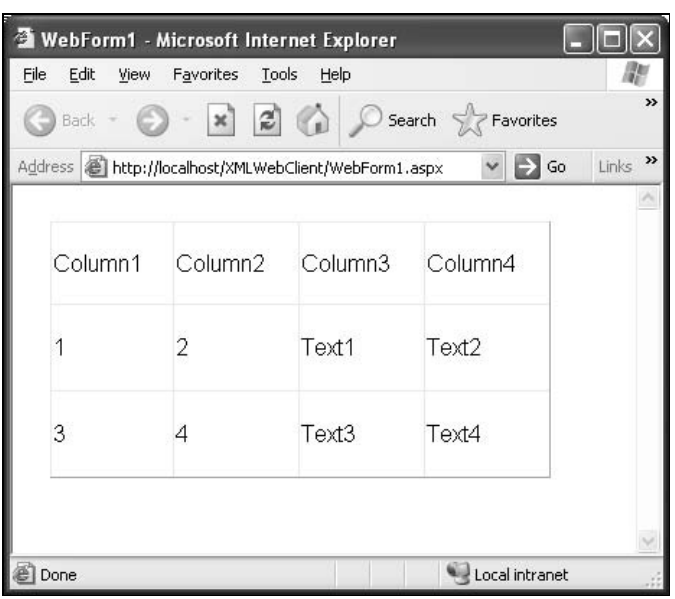

Рис 6.16. Внешний вид клиентской Web-страницы

Как можно видеть, основная работа выполняется при загрузке Web-страницы в браузер пользователя. Мы заполняем `DataSet` при помощи метода `Merge`, которому в качестве параметра передается результат вызова основной функции используемого Web-сервиса, а затем полученные данные отображаются в таблице, для чего используется метод `DataBind` компонента `DataGrid`. Все просто и прозрачно. На рис. 6.16 показан внешний вид нашей Web-страницы.

Пришло время подвести итоги главы. Она была призвана дать некоторое представление о правилах работы Web-сервисов с базами данных при помощи ADO.NET. Конечно, мы рассмотрели лишь основные правила работы. Но это и понятно. ADO.NET, действительно, большая тема, и для ее рассмотрения нужна отдельная книга. Поэтому мы надеемся, что приведенных примеров достаточно для понимания основных принципов взаимодействия с базами данных.

Глава 7

Основные приемы разработки Web-сервисов

Сохранение состояний

Данная глава посвящена рассмотрению приемов, которые часто используются при разработке Web-сервисов. В начале этой главы мы рассмотрим возможность Web-сервисов сохранять состояния. Дело в том, что модель работы типичного Web-сервиса предполагает, что функции выполняются обособленно друг от друга. То есть, клиентское приложение посылает запрос на выполнение какой-либо функции, получает результат выполнения функции, и на этом сеанс работы заканчивается. Другие функции Web-сервиса не знают, каков был результат выполнения предыдущей функции, они не зависят одна от другой. Естественно, если необходимо сделать так, чтобы функции Web-сервиса знали о результатах работы друг друга, разработчик может соответствующим образом передавать им эту информацию в качестве параметров. Однако существует более простой и изящный способ сохранять информацию о результатах выполнения тех или иных функций. Эти результаты, по сути дела, формируют состояние сервисного приложения, и в технологии разработки XML Web-сервисов Microsoft .NET предусмотрен специальный класс, позволяющий сохранять и обрабатывать состояние приложения.

Данные, которые будут одновременно доступны для чтения и модификации всем функциям, входящим в состав Web-сервиса, хранит объект `Application`. Чаще всего подобный объект разработчики и используют как коллекцию для хранения неких данных, которые должны быть доступны сразу нескольким функциям. Таким образом, объект `Application` предоставляет разработчику некий аналог глобальных переменных.

Необходимо осознавать, что к одному и тому же приложению может одновременно обращаться несколько удаленных пользователей. Соответственно, сервер IIS запустит несколько процессов, в каждом из которых будет выполняться копия приложения. Так вот, все эти процессы будут обладать одним объектом `Application`, общим для всех.

Следует отметить, что в иерархии Framework .NET существует несколько различных объектов с наименованием `Application`. Мы рассматриваем объект с полным наименованием `System.Web.Services.Application`. Естественно, в процессе написания кода нет нужды каждый раз указывать его полное имя, так как при первоначальном создании проекта Web-сервиса в код автоматически вставляется конструкция `Imports System.Web.Services`, которая импортирует соответствующее пространство имен, и объект `Application` будет по умолчанию использоваться именно в контексте этого пространства имен. Тип объекта `Application` обозначается как `HttpApplicationState`.

Объект `Application` представляет собой коллекцию разнородных элементов. Доступ к каждому элементу может осуществляться как по его уникальному имени, так и по его порядковому номеру. Соответственно, и список возможных свойств и методов несколько специфичен. Рассмотрим его, начиная со *свойств*.

- `AllKeys`. Свойство представляет собой массив строк, в которых записаны наименования всех элементов коллекции.
- `Contents`. Свойство представляет собой ссылку на весь объект `Application`. Обычно применяется для его копирования в иной объект. Однако в нашем случае нет нужды пользоваться данным свойством, достаточно обратиться к штатным механизмам присваивания значений. Свойство оставлено для обеспечения совместимости с предыдущими версиями ASP-приложений.
- `Count`. Свойство целочисленного типа. В нем указывается количество элементов в коллекции.
- `Item`. Свойство предоставляет доступ к какому-либо конкретному элементу. Доступ может осуществляться по наименованию элемента или его порядковому номеру. Необходимо помнить, что нумерация элементов коллекции начинается с нуля. На самом деле данное свойство используется по умолчанию, поэтому обращение `Application.Item(0)` абсолютно эквивалентно конструкции `Application(0)`.

Естественно, объект обладает и набором *методов*. Рассмотрим и их.

- `Add`. Метод добавляет еще одну глобальную переменную в общую коллекцию. В качестве параметров передается строка с наименованием нового элемента и его значение.
- `Clear`. Метод очищает содержимое коллекции.
- `Get`. Метод позволяет получить значение определенного элемента коллекции. В качестве параметра методу передается порядковый номер элемента или его уникальное наименование.
- `GetKey`. Метод позволяет получить значение элемента коллекции. Доступ осуществляется только по порядковому номеру.

- `Lock`. Метод блокирует коллекцию, позволяя менять ее содержимое только тому потоку, который вызвал этот метод.
- `Remove`. Метод удаляет определенный элемент коллекции. В качестве параметра методу передается наименование удаляемого элемента.
- `RemoveAll`. Метод удаляет все элементы коллекции встроенного объекта `Application`.
- `Set`. Метод позволяет задавать значение для уже существующего элемента коллекции. В качестве параметров методу передается наименование изменяемого элемента и присваиваемое ему значение.
- `UnLock`. Снимает с коллекции предварительно установленную блокировку и позволяет изменять ее содержимое всем потокам приложения.

Теперь, когда мы теоретически знаем, как обращаться с объектом `Application`, стоит рассмотреть практический пример работы с ним. Для этого мы создадим Web-сервис, который умеет только считать количество пользователей, одновременно работающих с ним. Для этого нам потребуется создать две функции — одну для присоединения пользователя к Web-сервису, а другую для отключения от него. Код этого Web-сервиса приведен в листинге 7.1.

Листинг 7.1

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer
```

```
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region

    <WebMethod()> Public Function NewUser() As Integer
        If Application("UserCount") Is Nothing Then
            Application("UserCount") = 1
        Else
            Application("UserCount") = CInt(Application("UserCount")) + 1
        End If
        NewUser = CInt(Application("UserCount"))
    End Function
    <WebMethod()> Public Function ExitUser() As Integer
        Application("UserCount") = CInt(Application("UserCount")) - 1
        ExitUser = CInt(Application("UserCount"))
    End Function
End Class
```

Итак, мы видим, что в состав Web-сервиса входят функции `NewUser` и `ExitUser`. Рассмотрим их поочередно. Функция `NewUser` обрабатывает подключение к сервису нового пользователя и увеличивает на единицу переменную, в которой хранится количество одновременно подключенных пользователей. Для хранения количества пользователей используется элемент коллекции `Application` с наименованием `UserCount`. Следует помнить, что при первом запуске Web-сервиса этого элемента еще не существует, поэтому попытка увеличить его значение на единицу вызовет исключение. Следовательно, перед увеличением счетчика следует проверить существование этой

переменной, и если ее не существует — создать. Для проверки существования мы применяем стандартное выражение `Is Nothing`. После этого код уже не нуждается в комментариях.

Вторая функция еще проще по своей структуре. Так как при выходе пользователя есть гарантия, что до этого он уже присоединялся к сервису, мы можем быть уверены, что переменная-счетчик существует, и ее значение больше нуля. Поэтому нам остается только уменьшить это значение.

Web-сервис без клиентского приложения — это нонсенс, ведь сам по себе он никому не нужен. Поэтому для полноты картины следует рассмотреть пример разработки клиента. Проще всего будет использовать для этой цели технологию ASP.NET. HTML-код простейшей Web-страницы, приспособленной для работы с Web-сервисом, приведен в листинге 7.2.

Листинг 7.2

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="AppStateClient.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>WebForm1</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 17px; POSITION:
absolute; TOP: 23px" runat="server" Width="378px">С сервисом работает x
пользователей</asp:Label>
   <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 25px; POSITION:
absolute; TOP: 58px" runat="server" Width="228px" Text="Отключиться от
сервиса"></asp:Button>
  </form>
 </body>
</HTML>
```

Естественно, одного HTML-кода для работы Web-страницы недостаточно. Потребуется разработать еще и класс, реализующий ее функциональность. Его код приведен в листинге 7.3.

Листинг 7.3

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

    End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        Dim serv = New AppStateClient.localhost.Service1()
        Dim i As Integer
        Dim s As String
        i = serv.NewUser()
        s = Convert.ToString(i)
        Label1.Text = "С сервисом работает " + s + " пользователей"
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim serv = New AppStateClient.localhost.Service1()
        serv.ExitUser()
```

```
    End Sub
End Class
```

Следует отметить, что функцию присоединения нового пользователя мы обрабатываем в тот момент, когда Web-страница только загружается. То есть, новый пользователь присоединяется к Web-сервису автоматически, как только загрузит Web-страницу. На основе числа одновременно подключенных к сервису пользователей формируется текстовое сообщение, отображаемое на Web-странице. Ее внешний вид показан на рис. 7.1.

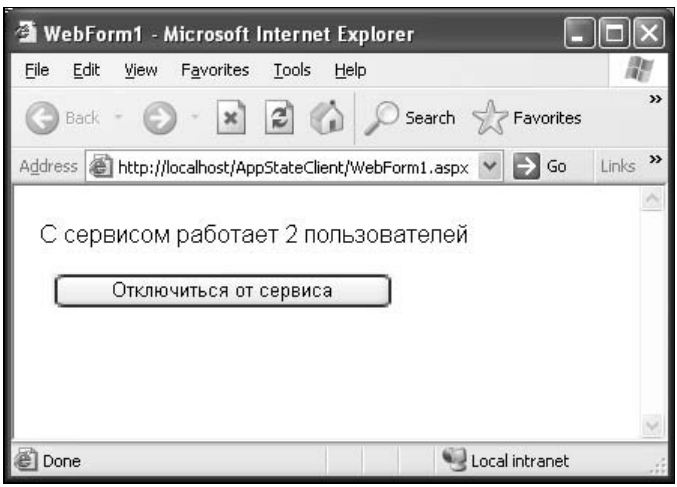

Рис. 7.1. Внешний вид Web-страницы, взаимодействующей с разработанным Web-сервисом

Следует также сказать несколько слов и о блокировках объекта `Application`. Может возникнуть ситуация, когда две или больше функций будут пытаться одновременно изменить содержимое одного и того же элемента коллекции `Application`. В этом случае, естественно, возникнет коллизия. Для того чтобы избежать подобной ситуации, стоит при масштабных и долгих изменениях элементов коллекции блокировать коллекцию, закрывая доступ к ней со стороны других функций и потоков. Тогда если какая-либо функция Web-сервиса установит блокировку на коллекцию `Application`, никакая иная функция или двойник установившей блокировку функции, выполняющийся в другом потоке, не сможет изменить значение того или иного элемента коллекции `Application`.

Установка блокировки выполняется при помощи метода `Lock`. Снятие блокировки, соответственно, производится при помощи метода `UnLock`. Разработчик обязательно должен вызывать метод `Unlock` после установки блоки-

ровки, потому что иначе коллекция `Application` так и останется заблокированной.

Идентификация пользователей

Еще одним весьма важным аспектом разработки Web-сервисов является механизм идентификации пользователей. Сам механизм идентификации можно разбить на две процедуры — аутентификация и авторизация. *Аутентификация* позволяет приложению узнать, какой именно пользователь работает с Web-сервисом. Обычно это прием некоего пароля или иного цифрового идентификатора, такого как номер кредитной карты или цифровое удостоверение, подтвержденное третьей стороной. *Авторизация* позволяет определять, какие именно функции Web-сервиса или их категории будут доступны данному конкретному посетителю. Таким образом, именно процедура авторизации ставит в соответствие опознанному пользователю его права работы в этой системе. Естественно, авторизация может проводиться только после завершения аутентификации.

Нам предстоит в этом разделе рассмотреть обе процедуры, и начнем мы именно с аутентификации.

Для Web-сервисов предусмотрено шесть вариантов аутентификации. Рассмотрим их вкратце.

- **Windows — Basic**. При использовании этого варианта аутентификации логин (имя пользователя) и его пароль передаются в виде обычных текстовых строк без использования шифрования. При этом, если злоумышленник, обладающий средствами перехвата сетевого трафика, находится в интранет-сети, где происходит процесс ввода и передачи логина и пароля, он имеет возможность получить конфиденциальные данные.
- **Windows — Basic over SSL**. Этот вариант аутентификации предназначен для получения логинов и паролей не только от пользователей внутренней интрасети, в которой действует Web-сервис, но и от удаленных пользователей, подключающихся к Web-сервису при помощи глобальной сети. При этом логин и пароль пользователя передаются под защитой протокола SSL (Secure Sockets Layer). Этот протокол позволяет несколько серьезнее защитить передаваемые данные, так как при его работе эти данные шифруются.
- **Windows — Digest**. В этом варианте аутентификации используется хэширование передаваемой информации, что позволяет скрыть пароль пользователя и не передавать его даже в зашифрованном виде. Вместо этого передается только хэш пароля.
- **Windows — Integrated Windows**. Этот вариант аутентификации использует алгоритмы шифрования данных NTLM или Kerberos для шифрования

трафика между Web-сервисом и браузером Internet Explorer. Поскольку этот режим аутентификации может взаимодействовать только с браузером, то для Web-сервиса придется создавать клиентские приложения в виде Web-страниц.

- **Windows — Client Certificates**. В этом варианте каждый пользователь обязан обзавестись собственным удостоверяющим сертификатом от соответствующей организации. Именно этот сертификат и служит цифровым удостоверением клиента, которое передается Web-сервису. Естественно, передачу информации о сертификате следует передавать под защитой SSL.
- **SOAP headers**. Как известно, основные блоки информации от сервиса к клиенту и обратно передаются в виде SOAP-сообщений. Этот вариант аутентификации позволяет клиенту внедрять в эти сообщения свои идентификационные данные и передавать их при помощи заголовков протокола SOAP.

Теперь, когда мы знаем, какие бывают варианты аутентификации пользователей, следует рассмотреть механизм их действия. Прежде всего, следует обратить внимание на то, что применяемый вариант аутентификации обычно указывается в конфигурационном файле Web-сервиса Config.web. Конфигурационный файл не используется только в том случае, когда разработчик применяет аутентификацию, основанную на заголовках протокола SOAP. Структуру этого файла мы рассмотрим в этой главе несколько позже, пока что мы лишь ограничимся одной возможностью этого файла, отвечающей за аутентификацию пользователей.

Файл Web.config является стандартным XML-файлом. Тело его разбито по умолчанию на семь разделов, которые отвечают за те или иные настройки приложения. Нас сейчас интересует секция, объявляемая тегом `<authentification>`. Нетрудно догадаться, что именно эта секция устанавливает параметры аутентификации, используемой в разрабатываемом Web-приложении.

У тега `<authentification>` есть обязательный атрибут `mode`, при помощи которого разработчик устанавливает используемый Web-приложением режим аутентификации. Данный атрибут может принимать одно из следующих четырех значений:

- `None`. Значение указывает, что Web-приложение не использует какой-либо аутентификации.
- `Windows`. Значение свидетельствует, что Web-приложение будет использовать аутентификацию, основанную на стандартном механизме распознавания пользователя операционной системы Windows.
- `Forms`. Значение указывает, что Web-приложение будет использовать аутентификацию на основе cookie-файлов.

- `Passport`. Данное значение свидетельствует, что в Web-приложении будет использоваться аутентификация, основанная на технологии Microsoft Passport. Для того чтобы можно было применить этот вариант идентификации пользователя, следует установить на сервере соответствующий Passport SDK.

При разработке Web-сервисов мы не можем использовать режим, задаваемый значением `Forms`, так как этот режим аутентификации основан на применении cookie-файлов. Естественно, ведь Web-сервисы не генерируют HTML-страницы, которые могли бы устанавливать в локальной системе пользователя cookie-файл. Поэтому чаще всего мы будем использовать в своих примерах Web-сервисов значение `Windows`.

Еще один вариант аутентификации пользователей опирается на использование технологии Microsoft.Passport. Эта технология позволяет хранить все данные каждого конкретного пользователя в одном месте. На самом деле, для аутентификации опять будет применяться cookie-файл или запрос пароля, но всю эту работу будет выполнять сервер Microsoft.Passport. То есть, запись о пользователе в базе данных этого сервера будет являться пропуском ко всем сайтам и Web-сервисам, на которые он заходит (если эти сайты, в свою очередь, могут использовать технологию Microsoft.Passport). Идея, в принципе, хороша. Каждый, кто находится в Сети достаточно долгое время, знает, сколько разных паролей и логинов ему приходилось вводить на самых различных сайтах. Поэтому, когда пользователь заходит в один из своих любимых онлайновых магазинов, ему приходится либо вспоминать свой логин и пароль, либо пользоваться службой напоминания пароля, когда на адрес его электронной почты сайт высылает идентификационные данные. Вот только надо еще вспомнить, какой именно адрес электронной почты вводился на данном сайте.

Microsoft.Passport снимает подобную проблему, так как хранит данные о пользователе, и сайту или сервису необходимо лишь получить эти данные с сервера службы Passport. Помимо идентификационных данных сервер Microsoft.Passport может хранить персональные сведения пользователей, такие как адрес электронной почты и номер кредитной карты. Так что пользователю, который обзавелся подобным цифровым паспортом, уже не придется на сайтах, поддерживающих эту технологию, вообще вводить какие-либо персональные данные. Сайт будет их узнавать от службы Microsoft.Passport. С этой точки зрения использование такой паспортной службы кажется очень хорошим решением. Однако не все так радужно.

Прежде всего, следует задуматься о том, что все персональные данные пользователя, в том числе и критичные, такие, как номер кредитной карты, будут храниться на отдельном сервере в Сети. Что будет, если этот сервер "упадет", как уже бывало с сервером Microsoft? Что будет, если его все-таки взломают, несмотря на обещанную "непробиваемую защиту"? Все мы пре-

красно знаем, что абсолютной защиты не бывает, и вероятность утери или разглашения конфиденциальных данных весьма отлична от нуля. Сама концепция хранения такого массива критичных данных на централизованном сервере (даже если серверов будет несколько, они все равно будут функционировать как один сервер) вызывает некоторую опаску. И неужели можно предположить, что в случае утери или компрометации конфиденциальных данных, корпорация Microsoft возместит ущерб? Честно говоря, в подобный исход верится с трудом.

Есть и еще один подводный камень в использовании этого сервиса. Дело в том, что пользователь должен видеть реальные преимущества от применения цифровых паспортов от Microsoft. То есть, количество сайтов и сервисов, использующих эти паспорта, должно быть достаточно велико. Однако распространение данной технологии среди разработчиков сдерживает тот факт, что Microsoft требует лицензировать применение Passport и просит оплатить свои услуги. Подобные действия весьма типичны для нынешнего курса действий корпорации Microsoft, однако они будут достаточно серьезно сдерживать внедрение Microsoft.Passport в массы.

Впрочем, делать прогнозы сейчас — дело неблагодарное. Microsoft активно продвигает идею Passport при помощи маркетинговых мероприятий. При этом основной упор корпорация делает на развитие сервисов, которые опознают пользователей именно по их цифровым паспортам. Данная инициатива носит наименование HailStorm.

Теперь, после того как мы вкратце познакомились с общими вопросами организации аутентификации пользователей в Visual Studio .NET, стоит перейти к рассмотрению примеров. Начнем мы с разбора механизма стандартной Windows-аутентификации. Это достаточно простой пример, который не требует дополнительных затрат времени разработчика. Основная работа сводится к конфигурированию Web-сервера IIS. Итак, разрабатываем простейший Web-сервис, доступ к которому осуществляется выборочно. Его код приведен в листинге 7.4.

Листинг 7.4

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
```

```
    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()
        'Add your own initialization code after the InitializeComponent()
call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services
Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region

    <WebMethod()> Public Function MyFunc() As String
        MyFunc = "Доступ получен"
    End Function
```

```
End Class
```

В этом варианте основная тяжесть отсеивания нежелательных посетителей перекладывается на Web-сервер IIS. Для этого в Visual Studio .NET следует активировать консоль управления Web-сервером IIS и в группе **Web Sites** выбрать каталог **AuthService**, в котором располагается наш Web-сервис. Затем для выбранного каталога следует вызвать контекстное меню и выполнить его команду **Properties**. В появившемся диалоговом окне **AuthService Properties** следует выбрать вкладку **Directory Security** и в группе **Anonymous access and authentication control** нажать кнопку **Edit**. При этом будет отображено дополнительное диалоговое окно **Authentication Methods**, в котором мы и настроим параметры доступа к Web-сервису. Внешний вид этого диалогового окна показан на рис. 7.2.

Рис. 7.2. Внешний вид диалогового окна **Authentication Methods**

В этом диалоговом окне следует снять флажок в независимом переключателе **Anonymous access**, запрещая таким образом анонимный доступ ко всем ресурсам, располагающимся в каталоге, где базируется наш Web-сервис. Также следует снять флажок в независимом переключателе **Integrated Windows authentication**. Зато флажок **Basic authentication** следует взвести. При этом Web-сервер IIS отобразит информационное окно, в котором выведет предупреждение, что в этом режиме пароль и логин (имя пользователя) будут передаваться в незашифрованном виде, и злоумышленник, обладающий

средствами перехвата сетевого трафика, сможет увидеть эти данные. В нашем случае мы можем игнорировать это предупреждение, так как подобная схема аутентификации обычно используется в интрасетях, где контроль за пользователями несравнимо лучше, чем в Интернете. После этого в текстовом поле **Default domain** можно указать имя домена, пользователи которого будут иметь доступ к ресурсам, располагающимся в защищаемом виртуальном каталоге.

Теперь при попытке соединения с Web-сервисом при помощи браузера напрямую у пользователя будет отображено диалоговое окно для ввода логина и пароля, внешний вид которого показан на рис. 7.3.

Рис. 7.3. Диалоговое окно для ввода логина и пароля удаленного пользователя

Настоящая ценность подобного метода аутентификации пользователей достаточно мала, так как для запуска такого механизма защиты необходимо, чтобы пользователь самостоятельно обращался к Web-сервису через браузер, не прибегая к помощи каких-либо клиентских приложений или соответствующих клиентских сайтов. Поэтому этот пример можно рассматривать лишь в качестве тренировочного.

Следует также отметить, что мы можем разрешить использование встроенного механизма аутентификации Windows. В этом режиме пользователь не сможет самостоятельно указывать свой логин и пароль. Для целей аутентификации будет использоваться имя пользователя, под которым он вошел в операционную систему при загрузке. Для того чтобы включить этот режим аутентификации, следует в диалоговом окне **Authentication Methods** взвести флажок в независимом переключателе **Integrated Windows authentication**. При этом Web-сервис сможет получить это имя и, опираясь на него, проводить авторизацию. Для того чтобы продемонстрировать применение этой возможности, придется несколько видоизменить Web-сервис. Его обновленный код приведен в листинге 7.5.

Листинг 7.5

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent()
call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services
Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
```

```
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region
    <WebMethod()> Public Function MyFunc() As String
        MyFunc = User.Identity.Name
    End Function

End Class
```

Как можно видеть, в новом сервисе функция возвращает имя пользователя, который соединяется с Web-сервисом. При попытке соединиться с сервисом при помощи стандартной Web-страницы, предоставляющей доступ к его функциям, в окне просмотра браузера отображается имя пользователя. Внешний вид этого окна просмотра показан на рис. 7.4.

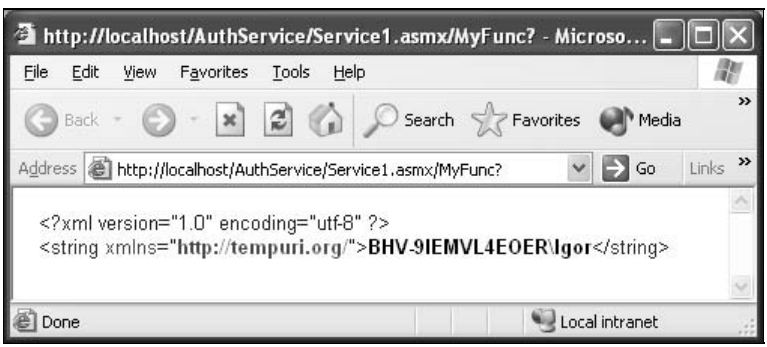

Рис 7.4. Окно просмотра браузера с результатом выполнения функции Web-сервиса

В работе Web-сервера мы используем объект с именем `User`. Этот объект имеет одно свойство `Identity`, определяемое типом `IIdentity`. Объекты подобного типа имеют всего три свойства.

- `AuthentificationType`. Свойство типа `String`. В нем указывается тип аутентификации, применяемой приложением.
- `IsAuthentificated`. Логическое свойство типа `Boolean`, которое показывает, прошел ли уже данный пользователь процедуру аутентификации или еще нет.
- `Name`. Свойство типа `String`, в котором хранится логин пользователя.

Следовательно, для того чтобы получить логин пользователя, надо использовать конструкцию `User.Identity.Name`, что мы, собственно, и сделали.

Однако все рассмотренные конструкции позволяют комфортно работать лишь в интрасети, где все пользователи заранее известны. Если необходимо проводить аутентификацию для пользователей всего Интернета, то следует либо хранить их учетные записи в каком-либо собственном хранилище, либо воспользоваться аутентификацией на основе их цифровых сертификатов. Впрочем, в силу малой распространенности цифровых сертификатов в мире, последний вариант будет использоваться чрезвычайно редко.

Конфигурационный файл Web-сервиса

Любой Web-сервис всегда имеет в своем составе конфигурационный файл Config.web, который и задает параметры приложения. Даже если разработчик не предпримет каких-либо действий по настройке свойств разрабатываемого приложения, среда разработки Visual Studio .NET обязательно создает файл Config.web с набором установок, используемых по умолчанию.

Желательно конфигурационный файл размещать в каталоге, являющимся корневым для Web-приложения. В этом случае действие конфигурационного файла будет распространяться и на все вложенные каталоги. Впрочем, разработчик имеет возможность разместить конфигурационный файл в любом вложенном каталоге приложения, и тогда файл более "глубокого" расположения будет переопределять параметры, заданные корневым конфигурационным файлом. Таким образом, разработчик получает возможность задавать различные параметры для разных каталогов. Чаще всего эта возможность используется для создания систем разграничения доступа пользователей.

Конфигурационный файл имеет, естественно, заранее определенную структуру и опирается на XML-синтаксис. Типичный конфигурационный файл, создаваемый средой разработки Visual Studio .NET по умолчанию, приведен в листинге 7.6.

Листинг 7.6

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <system.Web>

    <!--  DYNAMIC DEBUG COMPILATION
          Set compilation debug="true" to insert debugging symbols (.pdb
information)
```

```
          into the compiled page. Because this creates a larger file that
executes
          more slowly, you should set this_value to true only when
debugging and to
          false at all other times. For more information, refer to the
documentation about
          debugging ASP.NET files.
    -->
    <compilation defaultLanguage="vb" debug="true" />
    <!--  CUSTOM ERROR MESSAGES
          Set customErrors mode="On" or "RemoteOnly" to enable custom
error messages, "Off" to disable.
          Add <error> tags for each of the_errors you want to handle.
    -->
    <customErrors mode="RemoteOnly" />


    <!--  AUTHENTICATION
          This section sets the authentication policies of the
application. Possible modes are "Windows",
          "Forms", "Passport" and "None"
    -->
    <authentication mode="Windows" />



    <!--  AUTHORIZATION
          This section sets the authorization policies of the
application. You can allow or deny access
          to-application resources by user_or role. Wildcards: "*" mean
everyone, "?" means anonymous
          (unauthenticated) users.
    -->
    <authorization>
        <allow users="*" /> <!-- Allow allЂu117 ?sers -->

            <!--  <allow   —  users="[comma separated list of users]"
            —                 roles="[comma_separated list of roles]"/>
            —      <deny      users="[comma_separated list of users]"
            —                 roles="[comma_separated list of roles]"/>
            -->
    </authorization>
```

```
    <!--  APPLICATION-LEVEL TRACE LOGGING
          Application-level tracing enables trace log output for every
page within an application.
          Set trace enabled="true" to enable application trace logging.
If pageOutput="true", the
          trace information will be displayed at the bottom of each page.
Otherwise, you can view the
          application trace log by browsing the "trace.axd" page from
your Web application
          root.
    -->
    <trace enabled="false" requestLimit="10" pageOutput="false"
traceMode="SortByTime" localOnly="true" />



    <!--  SESSION STATE SETTINGS
          By-default ASP.NET uses cookies to identify which requests
belong to a particular session.
          If-cookies are not available, a session can be tracked by
adding a session identifier to the URL.
          To-disable cookies, set sessionState cookieless="true".
    -->
    <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString="data source=127.0.0.1;user
id=sa;password="
            cookieless="false"
            timeout="20"
    />


    <!--  GLOBALIZATION
          This section sets the globalization settings of the
application.
    -->
    <globalization requestEncoding="utf-8" responseEncoding="utf-8" />


  </system.Web>


</configuration>
```

Рассмотрим вкратце структуру этого конфигурационного файла. Вся информация обязана располагаться между тегом `<configuration>` и его закрывающим близнецом `</configuration>`.

Конфигурационный файл можно условно разбить на две части. В первой из них, которая является необязательной, разработчик может создавать свои конфигурационные разделы. Вторая часть содержит стандартные разделы параметров. Перечислим их.

- `<httpmodules>`. Раздел позволяет задавать параметры стандартных HTML-модулей, используемых в приложении.
- `<sessionstate>`. Раздел позволяет конфигурировать состояния сессий.
- `<globalization>`. Раздел позволяет создавать локализованные версии приложений.
- `<compilation>`. Содержимое этого раздела отвечает за параметры компиляции приложения.
- `<trace>`. Раздел отвечает за отладку разрабатываемого приложения.
- `<security>`. Содержимое раздела позволяет задавать параметры модели безопасности приложения.
- `<iisprocessmodel>`. Раздел позволяет настраивать параметры интеграции Web-сервисов с сервером IIS.
- `<browsercaps>`. Раздел предназначен для настройки компонента, определяющего возможности браузера, используемого удаленным пользователем.

Помимо конфигурационных файлов, о которых мы только что кратко рассказали, в работе Web-сервисов может использоваться файл Global.asax. Этот файл позволяет описывать действия, которые относятся к приложению в целом, а не к каким-либо сессиям или сеансам работы отдельных пользователей. Например, при запуске приложения потребуется установить связь с какой-либо базой данных или инициализировать элементы коллекции `Application`. Естественно, это надо делать один-единственный раз. И здесь на помощь разработчику приходит искомый файл Global.asax. Этот файл должен располагаться в корневом для Web-приложения каталоге.

В этом файле описываются процедуры, выполняемые при наступлении в приложении того или иного события. Пример подобного файла приведен в листинге 7.7.

Листинг 7.7

```
<script language="VB" runat="server">

    Sub Application_Start(Sender As Object, E As EventArgs)
```

```
    ' Do application startup code here
End Sub

Sub Application_End(Sender As Object, E As EventArgs)
    ' Clean up application resources here
End Sub

Sub Application_Error(Sender As Object, E As EventArgs)
    Context.ClearError()
    Response.Redirect("errorpage.htm")
End Sub

</script>
```

Если внимательно рассмотреть листинг, станет понятно, что при помощи файла Global.asax мы установили обработчики событий для всего приложения. Процедура `Application_Start` будет выполняться при запуске приложения, процедура `Application_End` предназначена для обработки закрытия приложения, а процедура `Application_Error` будет выполняться при возникновении каких-либо ошибок. Как можно видеть, ничего сложного в использовании файла Global.asax нет.

Глава 8

B2B-решение

Постановка задачи

В последнее время в Сети все чаще стали появляться решения класса B2B (расшифровывается как business-to-business). Под этим выражением обычно понимают некий Web-ресурс, который ориентирован не на отдельных пользователей, а на обслуживание нужд ряда корпоративных клиентов, часто работающих в одной отрасли. Например, машиностроители могут размещать на таком ресурсе информацию о своих предприятиях и списки цен на продукцию. Постепенно формируется единое информационное пространство, где каждый участник сформировавшегося профессионального сообщества может оформить заказ с оптимальным для него соотношением цены и качества.

Итак, основная задача B2B-ресурса — создание единого отраслевого информационного пространства, в котором его участники могут самостоятельно устанавливать контакты друг с другом. Сервисы сопровождения сделок B2B-ресурс, как правило, участникам сообщества не предлагает.

На данный момент B2B-ресурсы обычно оформляются в виде Web-сайтов. Естественно, это динамические сайты с полным набором функциональности, но информация выводится в виде HTML-документов. Соответственно, любая работа с B2B-ресурсом его посетителями производится при помощи браузера.

Само собой разумеется, это не самый удобный способ работы. Достаточно часто у корпоративных членов отраслевого информационного пространства существуют свои информационные системы, с которыми приходится интегрировать искомый B2B-ресурс. Проще говоря, свой прайс-лист на Web-сайт необходимо не только передать, но и постоянно обновлять его, поддерживать в актуальном состоянии. Ручной ввод информации явно неэффективен, так как этот процесс подвержен ошибкам, допускаемым оператором, и занимает слишком большое время.

Поэтому, чаще всего, B2B-сайты предоставляют набор правил для оформления поступающей информации. Соответственно, каждая организация, входящая в состав пользователей сайта, будет вынуждена использовать некое

программное решение, служащее портом между корпоративной информационной системой и самим сайтом. А ведь следует учитывать, что помимо передачи информации на сайт, пользователи рассчитывают еще и получать с него информацию. А как мы знаем, сайт выдает информацию в виде HTML-документов. Следовательно, запрошенную информацию следует выделить из полученного HTML-документа, перевести в формат, используемый корпоративной информационной системой, и только тогда она сможет попасть в базу данных организации. То есть, для полноценной работы с B2B-решением, реализованным в виде Web-сайта, организация должна разработать некое программное обеспечение, функционирующее в качестве шлюза между функциональностью сайта и корпоративной информационной системой. Причем, разработка этого шлюзового программного обеспечения не будет отличаться особой простотой.

В том случае, когда B2B-решение реализуется при помощи Web-сервисов, проблема создания шлюзового модуля не исчезает, но намного облегчается. Действительно, при разработке его с помощью Visual Studio .NET, разработчик всегда может использовать прямой доступ к функциональности Web-сервиса, как это было показано почти во всех примерах из предыдущих глав. При этом практически снимаются вопросы связи с сервисом и извлечения значимых данных из информации, переданных Web-сервисом.

Итак, в этой главе мы рассмотрим пример создания подобного решения класса B2B. В качестве предметной области возьмем книготорговлю. Мы создадим ресурс, при помощи которого издатели и книготорговцы смогут эффективно взаимодействовать друг с другом. Каждый пользователь сервиса сможет указывать список книг, которые есть у него в наличии, и осуществлять поиск тех книг, которые он хотел бы приобрести. Помимо самого Web-сервиса мы рассмотрим пример создания Web-сайта, который будет базироваться на этом сервисе. Все-таки без сайта совсем обойтись нельзя, так как чаще всего новый пользователь будет начинать работу с ресурсом именно с него. А уже потом он сможет создать под свои нужды клиентское приложение или скачать стандартный клиент с сайта. Мы рассмотрим и пример разработки стандартного клиентского приложения, взаимодействующего с Web-сервисом. То есть, у нас получится полный комплект — Web-сервис, сайт и клиентское приложение.

Теперь вкратце рассмотрим список задач, которые будут возложены на Web-сервис. Прежде всего, следует предоставить пользователям возможность регистрироваться в системе и задавать информацию о себе. Первичная регистрация будет производиться только на сайте, связанном с Web-сервисом. Также пользователи должны иметь возможность изменять информацию о себе. Эту операцию уже можно будет производить как с сайта, так и из клиентского приложения.

Пользователи должны иметь возможность передавать в общую базу данных сервиса информацию о книгах, которые их организация готова продавать.

Эта информация также будет изменяться пользователем. Естественно, следует предусмотреть аутентификацию и авторизацию пользователей, чтобы никто не смог изменить чужую информацию.

Помимо передачи данных, пользователи будут собирать информацию. Поэтому следует предусмотреть механизм поиска по базе данных. Полностью критерии поиска мы оговорим несколько позже, в соответствующем разделе главы, но, естественно, поиск должен производиться по издательству, наименованию, фамилии автора, ISBN и ценовому диапазону. Точнее, по любой комбинации этих критериев. Но все это в будущем. А пока перейдем к следующему разделу, в котором рассмотрим структуру базы данных, которая будет связана с Web-сервисом.

Структура базы данных

Саму базу данных мы назовем `BOOKS`. В ее состав будут входить несколько таблиц. Надо сразу предупредить, что структура базы данных может показаться (а, скорее всего, и объективно будет) неоптимальна. Однако мы создаем всего лишь тестовый пример, в котором использование всех возможностей SQL-сервера и ADO.NET лишь неоправданно усложнит разработку. Поэтому создадим достаточно простую структуру, которая будет максимально прозрачна.

Первая таблица предназначена для хранения информации о пользователях. В ней будет храниться уникальный идентификатор пользователей, их пароль для доступа к своим данным, адрес электронной почты, наименование организации и телефон. При этом уникальный логин-идентификатор будет выполнять роль первичного ключа. Таблицу назовем `CLIENTS`. SQL-скрипт, описывающий ее создание, выглядит следующим образом:

```
CREATE TABLE [dbo].[CLIENTS] (
  [IDLOGIN] [char] (30) COLLATE Cyrillic_General_CI_AS NOT NULL,
  [PASSWORD] [char] (10) COLLATE Cyrillic_General_CI_AS NULL,
  [NAIM] [char] (50) COLLATE Cyrillic_General_CI_AS NULL,
  [EMAIL] [char] (30) COLLATE Cyrillic_General_CI_AS NULL,
  [PHONE] [char] (20) COLLATE Cyrillic_General_CI_AS NULL
) ON [PRIMARY]
```

Как можно видеть, все поля в создаваемой таблице имеют символьный тип. Разница лишь в наименовании и длине. Поле `IDLOGIN` предназначено для хранения уникального идентификатора пользователя. Именно по этому полю создается первичный индекс. Для авторизации пользователя используется пароль, вводимый им начале сеанса работы с Web-сервисом. Соответственно, этот пароль хранится в поле `PASSWORD`. А поля `NAIM`, `EMAIL` и `PHONE` предназначены для хранения наименования пользователя, его адреса электронной почты и номера телефона, соответственно.

Теперь перейдем к созданию таблицы, в которой будет храниться информация о книгах. В одной из предыдущих глав мы уже создавали подобную таблицу, но сейчас подойдем к этому вопросу несколько серьезнее. Как уже говорилось, каждая книга идентифицируется при помощи уникального кода ISBN. Помимо этого, следует хранить информацию об авторе книги, наименовании, объеме книги в страницах, издателе и времени выхода в свет. Соответственно, SQL-скрипт, создающий таблицу `BOOKS`, будет выглядеть следующим образом:

```
CREATE TABLE [dbo].[BOOKS] (
     [ISBN] [char] (15) COLLATE Cyrillic_General_CI_AS NOT NULL,
     [NAIM] [char] (50) COLLATE Cyrillic_General_CI_AS NULL,
     [PUBLISHER] [char] (50) COLLATE Cyrillic_General_CI_AS NULL,
     [AUTHOR] [char] (30) COLLATE Cyrillic_General_CI_AS NULL,
     [BOOKVALUE] [int] NULL,
     [BOOKYEAR] [int] NULL
) ON [PRIMARY]
```

Итак, поле `ISBN` содержит ISBN-код книги, поле `NAIM` предназначено для хранения наименования книги, в поле `AUTHOR` хранится имя автора книги, а в поле `PUBLISHER` — наименование издательства, выпустившего книгу. Все эти поля имеют символьный тип. Также в состав таблицы входят два целочисленных поля. Поле `BOOKVALUE` хранит количество страниц в данной книге, а поле `BOOKYEAR` предназначено для хранения номера года, в котором выпущена эта книга. Следует отметить, что значение поля `PUBLISHER` будет совпадать с наименованием организации-издателя из таблицы `CLIENTS`. По всем правилам нам следовало бы явно указать связь между этими таблицами, но в нашем примере мы заставим Web-сервис самостоятельно отслеживать эту связь и контролировать соответствие полей.

Помимо таблиц, содержащих информацию о клиентах и книгах, следует создать таблицу, в которой будут храниться предложения книготорговцев. Предложения торгующих организаций, среди которых есть и издательства, будут иметь достаточно простую структуру. Это ISBN-код книги, цена, по которой она продается данной организацией, и идентификатор организации. Однако таблица эта будет обновляться чаще всего, и к ней будут применяться операции `Insert`, `Update` и `Delete`. Как мы помним, при использовании стандартных компонентов ADO.NET в Web-сервисах необходимо, чтобы таблица, к которой применяются эти команды, имела идентифицирующее поле. Следовательно, нам придется добавить в таблицу `PRICES` и его. Соответственно, SQL-скрипт, создающий новую таблицу `PRICES`, выглядит следующим образом:

```
CREATE TABLE [dbo].[PRICES] (
       [IDPRICE] [int] IDENTITY (1, 1) NOT NULL,
```

```
        [ISBN] [char] (15) COLLATE Cyrillic_General_CI_AS NULL,
        [PRICE] [int] NULL,
        [LOGIN] [char] (30) COLLATE Cyrillic_General_CI_AS NULL
) ON [PRIMARY]
GO
```

Как можно видеть, для поля `IDPRICE` мы использовали тип `int` с модификатором `IDENTITY (1,1)`. Что это означает? Если при определении типа какого-либо поля в таблице разработчик использует такой модификатор, то SQL-сервер будет считать это поле идентификатором и при добавлении в таблицу новой записи автоматически задавать уникальное значение этого поля. То есть, разработчик не должен сам заботиться о его заполнении и поддержании в адекватном состоянии. Для этого не потребуется даже создавать какой-либо хранимой процедуры, SQL-сервер все сделает самостоятельно. Две единицы в скобках после ключевого слова `IDENTITY` означают, что нумерация записей начнется с единицы и будет увеличиваться на единицу для каждой новой записи.

Конечно, подобный модификатор для поля следует установить при проектировании структуры таблицы. Так как наше поле будет одновременно служить и первичным индексом, то нужно щелчком правой кнопки мыши вызвать для этого поля объектное меню и выполнить его команду **Set Primary Key**. Затем перейти в нижнюю часть диалогового окна **Design Table** и в таблице **Columns** обратить внимание на строку **Identity**. В этой строке следует установить значение **Yes**, чтобы поле стало идентификатором. Строки **Identity Seed** и **Identity Increment** позволяют задать начальное значение поля идентификатора и его постоянное приращение.

Структура сервиса

Итак, у нас теперь есть структура базы данных, и на ее основе мы можем разработать структуру Web-сервиса. Прежде всего, как уже говорилось, следует предоставить пользователю возможность зарегистрироваться в системе. Эта функция будет принимать логин, пароль, наименование пользователя, его адрес электронной почты и номер телефона. В качестве результата функция вернет текстовую строку, в которой будет указано, принята учетная запись пользователя или нет.

Также издательства должны иметь возможность добавлять информацию о книгах, изданных этими издательствами, и изменять информацию о них в том случае, если оператором была допущена ошибка. Так как все операции с Web-сервисом должны быть представлены на сайте, то придется создать две процедуры — одну для ввода информации о новой книге и вторую — для изменения информации о книге. При этом, помимо данных о книге, необходимо принимать еще и логин с паролем пользователя, чтобы быть уверенным в том, что информацию о книге изменяет именно ее издатель.

В том случае если для работы с сервисом пользователь использует специализированное клиентское приложение, работу можно несколько облегчить и передавать функции сразу набор данных. То есть, пользователю уже не будет нужды после каждого изменения отсылать пакет данных Web-сервису и ожидать отклика — все изменения можно переслать за один раз.

Помимо издательств, в работе B2B-ресурса примут участие и книготорговцы, которые могут выставлять свои коммерческие предложения. Соответственно, необходимо создать функцию, которая будет принимать эти коммерческие предложения и записывать полученную информацию в базу данных. При работе с сайта эти данные будут приниматься в виде обычных отдельных параметров, а для работы со специализированным клиентским приложением следует предусмотреть функцию, которая могла бы принимать в качестве параметра набор данных.

Помимо добавления информации о продаваемых книгах, книготорговцы могут изменять выставленную информацию и удалять некоторые свои предложения. Соответственно, необходимо будет создать функции, обрабатывающие эти действия.

И конечно, нам пригодится функция поиска книг в общей базе данных по любому признаку, в том числе и по ценовому диапазону. Эта функция будет принимать в качестве параметров критерии поиска и выдавать набор данных, в котором будут содержаться результаты поиска.

Итак, теперь, когда у нас есть приблизительное представление о структуре Web-сервиса, который мы будем создавать, настало время приступить к его разработке.

Разработка сервиса

Начнем мы с разработки функции регистрации нового пользователя. Функция должна принимать в качестве параметров логин и пароль нового пользователя, а также дополнительную информацию о пользователе, которую необходимо будет хранить в базе данных. Кроме того, следует предусмотреть ситуацию, когда пользователь при первичной регистрации использует уже существующий логин. То есть, необходимо разработать функцию, проверяющую вводимый логин на допустимость. Таким образом, для регистрации пользователя мы будем использовать две функции. Начнем мы с функции, проверяющей логин на допустимость. Для ее работы нам потребуется доступ к таблице `CLIENTS`. Следовательно, нам необходимо установить соединение с используемой базой данных и отбуксировать из окна **Server Explorer** на страницу разрабатывамого Web-сервиса таблицу `CLIENTS`. При этом на странице, как мы уже знаем из предыдущей главы, появятся компоненты `SqlConnection` и `SqlDataAdapter`. Для последнего компонента мы зададим наименование `ClientSqlDataAdapter`. Саму проверку мы будем производить при помощи следующего SQL-запроса:

```
SELECT    COUNT(IDLOGIN) AS CID
FROM      CLIENTS
WHERE    (IDLOGIN = @LOG)
```

То есть, в результате его выполнения мы получим набор данных, состоящий из одной записи с одним полем. Можно было бы воспользоваться для проверки и хранимой процедурой, но для тестового примера преимуществ у процедуры перед набором данных практически нет, так как пользователю все равно будет передаваться логическое значение. Текст этого запроса мы поместим в свойство `SelectCommand` объекта `ClientSqlDataAdapter`.

Теперь перейдем к рассмотрению кода функции, проверяющей наличие заданного логина в базе данных. Этот код приведен в листинге 8.1.

Листинг 8.1

```
<WebMethod()> Public Function LogExist(ByVal LOG As String) As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    SqlSelectCommand1.Parameters("@LOG").Value = LOG
    ClientSqlDataAdapter.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") > 0 Then
      t = True
    Else
      t = False
    End If
    DS.Dispose()
    LogExist = t
  End Function
```

Разберем структуру этой функции. Первая строка объявляет и создает экземпляр объекта `DataSet`. Так как мы создаем его программно, нам не потребовалось буксировать соответствующий компонент на страницу дизайна Web-сервиса. Затем мы задаем значение единственного параметра SQL-запроса, при помощи которого будем проверять наличие логина в базе. Лишь после этого мы совершаем запрос, одновременно помещая результаты его выполнения в набор данных `DS` при помощи метода `Fill`, присущего объекту `ClientSqlDataAdapter`.

Так как в результате выполнения запроса создается набор данных, состоящий из одной записи всего с одним полем, то мы можем напрямую получить доступ к значению этого поля при помощи следующей конструкции:

```
DS.Tables(0).Rows(0).Item("CID")
```

То есть, мы пользуемся коллекциями `Tables` и `Rows`. При этом берем самые первые элементы этих коллекций (не следует забывать, что нумерация элементов коллекций начинается с нуля). На основе полученного значения мы формируем переменную логического типа `t`, которую потом используем в качестве возвращаемого значения.

Но перед тем как вернуть полученное значение, мы освобождаем ранее объявленный экземпляр объекта `DataSet` при помощи метода `Dispose`. Строго говоря, этого можно было и не делать, так как при работе в среде `CLR` объявленные, но не освобожденные объекты все равно удаляются после того, как завершается их цикл жизни. Для этого используется весьма полезное новшество Microsoft .NET — автоматический сборщик мусора. Использование метода `Dispose` при работе в .NET является более предпочтительным, нежели использование стандартного деструктора `Finalize` из-за особенностей работы сборщика мусора.

Теперь рассмотрим пример создания функции, регистрирующей нового пользователя, сохраняя информацию о нем в базу данных. Для этого мы используем все тот же объект `ClientSqlDataAdapter`, что и в предыдущей функции. Но теперь нам потребуется его свойство `InsertCommand`. В этом свойстве мы поместим SQL-запрос, выполняющий добавление данных. Сам текст SQL-запроса выглядит следующим образом:

```
INSERT INTO CLIENTS
       (IDLOGIN, PASSWORD, NAIM, EMAIL, PHONE)
VALUES (@IDLOGIN, @PASSWORD, @NAIM, @EMAIL, @PHONE)
```

Теперь рассмотрим саму функцию регистрации нового пользователя. Ее код приведен в листинге 8.2.

Листинг 8.2

```
<WebMethod()> Public Function RegUser(ByVal LOG As String, ByVal PASSWORD
As String, ByVal NAIM As String, ByVal EMAIL As String, ByVal Phone As
String) As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    SqlSelectCommand1.Parameters("@LOG").Value = LOG
    ClientSqlDataAdapter.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") > 0 Then
      t = False
    Else
      SqlInsertCommand1.Parameters("@IDLOGIN").Value = LOG
      SqlInsertCommand1.Parameters("@PASSWORD").Value = PASSWORD
      SqlInsertCommand1.Parameters("@NAIM").Value = NAIM
```

```
      SqlInsertCommand1.Parameters("@EMAIL").Value = EMAIL
      SqlInsertCommand1.Parameters("@PHONE").Value = Phone
      SqlConnection1.Open()
      SqlInsertCommand1.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    DS.Dispose()
    RegUser = t
  End Function
```

Итак, эта функция принимает в качестве параметров пять значений типа `String`, которые будут занесены в таблицу `CLIENTS`. Результатом выполнения функции является логическое значение. Если логин, переданный функции пользователем, уже существует в базе данных, информация не записывается, и функция возвращает значение `False`. В противном случае SQL-запрос `INSERT` все-таки выполняется, и функция возвращает значение `True`. Для проверки существования переданного логина мы используем код, описанный в предыдущей функции.

В том случае, если запись с указанным пользователем логином в базе данных еще не существует, задаются значения параметров SQL-запроса, связанного с объектом `SqlInsertCommand1`, затем принудительно открывается соединение с базой данных, выполняется метод `ExecuteNonQuery`, и соединение с базой данных закрывается. Из-за того что команда `Insert`, как мы уже говорили, не возвращает какого-либо набора данных, ее необходимо выполнять именно при помощи метода `ExecuteNonQuery`, перед этим предварительно открыв соединение, так как оно автоматически устанавливается только для команд `Select`, которые возвращают наборы данных.

Также следует разработать функцию, которая позволяет изменять индивидуальную информацию пользователя. Для этого придется использовать SQL-скрипт `UPDATE`, связанный со свойством `UpdateCommand`. SQL-скрипт, выполняющий изменение информации, будет выглядеть следующим образом:

```
UPDATE  CLIENTS
SET   IDLOGIN = @IDLOGIN, PASSWORD = @PASSWORD, NAIM = @NAIM,
EMAIL = @EMAIL, PHONE = @PHONE
WHERE   (IDLOGIN = @IDLOGIN)
```

Рассмотрим саму структуру этой функции. Перед тем как изменять данные, необходимо убедиться, что пользователь, пославший запрос на изменение, действительно тот, за кого себя выдает. Для проверки мы используем его логин и пароль. Только после того как сервис убедится, что пароль соответствует логину, он выполнит запрос. Соответственно, функция будет возвра-

щать логическое значение, указывающее, успешно ли были произведены изменения. Конечно, можно создать более тщательную функцию проверки, которая распознавала бы тип проблемной ситуации и возвращала значение, указывающее на то, что или логин не существует, или пароль не соответствует указанному логину, но здесь мы ограничимся простейшим вариантом. В конце концов, это всего лишь небольшой пример, от которого не стоит требовать идеальной архитектуры, так как основная наша цель сейчас — простота и наглядность.

Итак, введем код еще одной функции нашего сервиса (листинг 8.3).

Листинг 8.3

```
<WebMethod()> Public Function EditUser(ByVal LOG As String, ByVal
PASSWORD As String, ByVal NAIM As String, ByVal EMAIL As String, ByVal
PHONE As String) As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select count(IDLOGIN) as CID from
CLIENTS where ((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@IDLOGIN", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@IDLOGIN").Value = LOG
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PASSWORD", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@PASSWORD").Value = PASSWORD
    DataCommand.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") = 0 Then
      t = False
    Else
      SqlUpdateCommand1.Parameters("@IDLOGIN").Value = LOG
      SqlUpdateCommand1.Parameters("@PASSWORD").Value = PASSWORD
      SqlUpdateCommand1.Parameters("@NAIM").Value = NAIM
      SqlUpdateCommand1.Parameters("@EMAIL").Value = EMAIL
      SqlUpdateCommand1.Parameters("@PHONE").Value = PHONE
      SqlConnection1.Open()
      SqlUpdateCommand1.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
```

```
    End If
    DS.Dispose()
    EditUser = t
  End Function
```

Теперь пришло время разобрать структуру созданной функции. Для проверки соответствия логина и пароля, введенных пользователем, нам придется получить их из таблицы. Следовательно, мы будем вынуждены использовать SQL-запрос `Select`, который сформирует соответствующий набор данных. Но при этом мы не можем использовать свойство `SelectCommand`, так как оно уже занято одной из предыдущих функций. Что ж, ничего страшного, мы вполне можем самостоятельно создать новые экземпляры классов `DataSet` и `SqlDataAdapter`.

Для создания нового экземпляра объекта `SqlDataAdapter` мы указываем SQL-запрос, выполняемый этим объектом при вызове объекта `Fill`, и соединение с базой данных. Причем, соединение мы используем уже существующее — объект `SqlConnection1`. А SQL-запрос будет иметь следующий вид:

```
select count(IDLOGIN) as CID from CLIENTS where
((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))
```

Основная цель этого запроса — проверка, существуют ли в базе данных указанные пароль и логин, входящие в состав одной записи.

После того как будет выполнен этот SQL-запрос, мы переходим к проверке полученного результата. В том случае, если указанные пароль и логин не существуют, функция Web-сервиса просто вернет логическое значение `False`, и на этом ее работа будет закончена. Однако если существует запись с указанным логином и паролем, функция должна выполнить изменения в базе данных. Для этого, как уже говорилось ранее, мы используем SQL-запрос `Update`. Но перед тем как его выполнить, следует все-таки задать параметры для такого SQL-запроса, использовав для этого параметры самой функции, значения которых были установлены пользователем. Впрочем, эта операция не должна вызывать никаких трудностей, так как мы уже не раз задавали параметры для запросов в теле программы. Сам запрос, как обычно, выполняется при помощи метода `ExecuteNonQuery` с явным открытием и закрытием соединения с базой данных. Итак, еще один функциональный блок нашего сервиса, призванного служить сердцем B2B-ресурса, создан.

Теперь следует разработать функцию, которая позволяет издательствам добавлять в базу данных информацию о выпущенных книгах. Эта функция будет принимать в качестве параметров логин и пароль пользователя, а также ISBN-код книги, ее наименование, фамилию автора, объем книги в страницах и год выпуска. Код этой функции приведен в листинге 8.4.

Листинг 8.4

```
<WebMethod()> Public Function AddBook(ByVal LOG As String, ByVal PASSWORD
As String, ByVal NAIM As String, ByVal ISBN As String, ByVal AUTHOR As
String, ByVal BOOKVALUE As Integer, ByVal BOOKYEAR As Integer) As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    Dim n As String
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim DataCommand2 As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select count(IDLOGIN) as CID from
CLIENTS where ((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@IDLOGIN", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@IDLOGIN").Value = LOG
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PASSWORD", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@PASSWORD").Value = PASSWORD
    DataCommand.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") = 0 Then
      t = False
    Else
      SelectCommand = "Select NAIM from CLIENTS where IDLOGIN=@LOG"
      DataCommand2 = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
      DataCommand2.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 20))
      DataCommand2.SelectCommand.Parameters("@LOG").Value = LOG
      DS.Clear()
      DataCommand2.Fill(DS)
      n = DS.Tables(0).Rows(0).Item("NAIM")
BooksSqlDataAdapter.InsertCommand.Parameters("@ISBN").Value = ISBN
BooksSqlDataAdapter.InsertCommand.Parameters("@NAIM").Value = NAIM
BooksSqlDataAdapter.InsertCommand.Parameters("@PUBLISHER").Value = n
BooksSqlDataAdapter.InsertCommand.Parameters("@AUTHOR").Value = AUTHOR
BooksSqlDataAdapter.InsertCommand.Parameters("@BOOKVALUE").Value =
BOOKVALUE
BooksSqlDataAdapter.InsertCommand.Parameters("@BOOKYEAR").Value =
BOOKYEAR
```

```
      SqlConnection1.Open()
      BooksSqlDataAdapter.InsertCommand.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    AddBook = t
  End Function
```

Следует отметить, что перед тем, как разрабатывать эту функцию, на страницу дизайна сервиса стоит добавить еще один компонент `SQLDataAdapter`, связанный на этот раз с таблицей `BOOKS`, и воспользоваться его свойством `InsertCommand`. Текст SQL-запроса, связанного с этим свойством, выглядит следующим образом:

```
INSERT INTO BOOKS
(ISBN, NAIM, PUBLISHER, AUTHOR, BOOKVALUE, BOOKYEAR)
VALUES (@ISBN, @NAIM, @PUBLISHER, @AUTHOR, @BOOKVALUE, @BOOKYEAR)
```

Теперь рассмотрим код функции. Первая часть ее отрабатывает проверку соответствия логина и пароля, переданных пользователем. Этот фрагмент можно позаимствовать из предыдущих функций.

После того как мы удостоверимся, что пользователь передал верные пароль и логин, следует выполнить SQL-запрос `INSERT`, связанный с компонентом `BookSqlDataAdapter`. Однако для его выполнения нам необходимо получить наименование издателя, которое не было передано вместе с остальными параметрами функции. Действительно, для чего передавать еще и наименование, если оно однозначно идентифицируется логином?

Поэтому для получения наименования издателя нам потребуется выполнить еще один SQL-запрос следующего вида:

```
Select NAIM from CLIENTS where IDLOGIN=@LOG
```

Для выполнения этого запроса мы создаем еще один объект `DataCommand2`, который будет заполнять набор данных `DS` при помощи вышеупомянутого запроса. Правда, перед тем, как заполнять набор данных, необходимо его очистить, так как в нем хранятся результаты выполнения предыдущего запроса. Причем следует не просто убрать из набора предыдущие данные, но и полностью изменить его структуру. Для этого используется метод `Clear`. А после выполнения этого метода можно заново заполнять набор данных результатами искомого SQL-запроса.

Данным запросом мы получим наименование издательства, соответствующего переданному пользователем логину, и присвоим это наименование переменной `n` типа `String` при помощи следующей конструкции:

```
n = DS.Tables(0).Rows(0).Item("NAIM")
```

А затем все просто. Осталось лишь задать значения параметров SQL-запроса `INSERT` и выполнить его.

Теперь, когда у нас есть заполненная таблица с информацией о книгах, следует разработать функцию, позволяющую пользователям системы помещать в базу данных информацию о своих коммерческих предложениях. Для этого нам потребуется еще один компонент `SQLDataAdapter`, который будет связан с таблицей `PRICES`.

Естественно, мы используем свойство `InsertCommand`, которое будет обрабатывать SQL-запрос, добавляющий информацию в таблицу. Сам запрос будет выглядеть следующим образом:

```
INSERT INTO PRICES
       (ISBN, PRICE, LOGIN)
VALUES (@ISBN, @PRICE, @LOGIN)
```

Теперь можно рассмотреть код функции, реализующей добавление коммерческих предложений в базу данных (листинг 8.5).

Листинг 8.5

```
<WebMethod()> Public Function AddPrice(ByVal LOG As String, ByVal
PASSWORD As String, ByVal ISBN As String, ByVal PRICE As Integer)
As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select count(IDLOGIN) as CID from
CLIENTS where ((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@IDLOGIN", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@IDLOGIN").Value = LOG
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PASSWORD", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@PASSWORD").Value = PASSWORD
    DataCommand.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") = 0 Then
      t = False
    Else
  PricesSqlDataAdapter.InsertCommand.Parameters("@ISBN").Value = ISBN
  PricesSqlDataAdapter.InsertCommand.Parameters("@PRICE").Value = PRICE
  PricesSqlDataAdapter.InsertCommand.Parameters("@LOGIN").Value = LOG
```

```
      SqlConnection1.Open()
      PricesSqlDataAdapter.InsertCommand.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    AddPrice = t
  End Function
```

Сначала мы проверяем соответствие логина и пароля, и в том случае, если пользователь прошел аутентификацию, выполняем SQL-запрос `INSERT`. Как видно, код этой функции достаточно прост.

Теперь нам осталось разработать функцию, которая позволяет осуществлять поиск по базе данных на основе заданных пользователем параметров. В качестве параметров искомая функция будет принимать ISBN-код книги, наименование издательства, книги которого интересуют пользователя, фрагмент наименования книги или фамилии автора, а также ценовой диапазон, в котором должны находиться книги. Естественно, пользователь должен иметь возможность задавать любую комбинацию этих параметров поиска. Таким образом, если какой-либо критерий пользователем не был задан, то функции будет передано пустое значение. На основе полученных параметров функция сформирует SQL-запрос и выполнит его. Результатом работы функции будет набор данных, который функция и вернет клиентскому приложению. Код этой функции приведен в листинге 8.6.

Листинг 8.6

```
<WebMethod()> Public Function FindBooks(ByVal ISBN As String, ByVal
PUBLISHER As String, ByVal NAIM As String, ByVal AUTHOR As String, ByVal
PRICE1 As Integer, ByVal PRICE2 As Integer) As DataSet
    Dim DS As New DataSet()
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select PRICES.PRICE AS PRICE,
BOOKS.NAIM AS NAIM, BOOKS.PUBLISHER AS PUBL, BOOKS.AUTHOR AS AUTHOR,
BOOKS.ISBN AS ISBN,CLIENTS.NAIM AS CLNAIM,CLIENTS.EMAIL AS CLEMAIL FROM
PRICES INNER JOIN BOOKS ON PRICES.ISBN = BOOKS.ISBN INNER JOIN CLIENTS ON
PRICES.LOGIN = CLIENTS.IDLOGIN where
    ("SelectCommand = SelectCommand + "(PRICES.PRICE >= @PRICE1) AND
(PRICES.PRICE <=@PRICE2)"
    If ISBN <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.ISBN=@ISBN)"
    End If
    If PUBLISHER <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.PUBLISHER=@PUBLISHER)"
```

```
    End If
    If NAIM <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.NAIM LIKE @NAIM)"
    End If
    If AUTHOR <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.AUTHOR LIKE @AUTHOR)"
    End If
    SelectCommand = SelectCommand + ")"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PRICE1", System.Data.SqlDbType.Int, 4))
    DataCommand.SelectCommand.Parameters("@PRICE1").Value = PRICE1
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PRICE2", System.Data.SqlDbType.Int, 4))
    DataCommand.SelectCommand.Parameters("@PRICE2").Value = PRICE2
    If ISBN <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@ISBN", System.Data.SqlDbType.Char, 15))
      DataCommand.SelectCommand.Parameters("@ISBN").Value = ISBN
    End If
    If PUBLISHER <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PUBLISHER", System.Data.SqlDbType.Char,
50))
      DataCommand.SelectCommand.Parameters("@PUBLISHER").Value =
PUBLISHER
    End If
    If NAIM <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@NAIM", System.Data.SqlDbType.Char, 50))
      DataCommand.SelectCommand.Parameters("@NAIM").Value = "'%" + NAIM +
"%'"
    End If
    If AUTHOR <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@AUTHOR", System.Data.SqlDbType.Char, 30))
      DataCommand.SelectCommand.Parameters("@AUTHOR").Value = "'%" +
AUTHOR + "%'"
    End If
    DataCommand.Fill(DS)
    FindBooks = DS
  End Function
```

Основа всей этой функции — SQL-запрос `SELECT`, текст которого генерируется динамически, на основе полученных параметров. По умолчанию мы считаем, что гарантированно передаются нижняя и верхняя границы ценового диапазона. Действительно, если пользователь даже не задал их, клиентское приложение будет передавать границы, которые заведомо включают в себя цены всех книг, например от нулевого значения до 10 000. Затем мы производим анализ остальных переданных параметров. Если какой-либо параметр является пустым, то мы не включаем его в текст запроса. В ином случае, добавляется еще одно условие.

Сам SQL-запрос в том случае, если пользователь задал все параметры поиска, выглядит следующим образом:

```
select PRICES.PRICE AS PRICE, BOOKS.NAIM AS NAIM, BOOKS.PUBLISHER AS
PUBL, BOOKS.AUTHOR AS AUTHOR, BOOKS.ISBN AS ISBN,CLIENTS.NAIM AS
CLNAIM,CLIENTS.EMAIL AS CLEMAIL

FROM PRICES INNER JOIN BOOKS ON PRICES.ISBN = BOOKS.ISBN INNER JOIN
CLIENTS ON PRICES.LOGIN = CLIENTS.IDLOGIN

where ((PRICES.PRICE >= @PRICE1) AND (PRICES.PRICE <= @PRICE2) AND
(BOOKS.ISBN = @ISBN) AND (BOOKS.PUBLISHER = @PUBLISHER) AND (BOOKS.NAIM
LIKE @NAIM) AND (BOOKS.AUTHOR LIKE @AUTHOR))
```

После того как SQL-запрос будет сформирован, останется только выполнить его и заполнить набор данных. Если обратить внимание на текст SQL-запроса, станет ясно, что каждая запись в наборе данных состоит из полей, в которых указывается цена книги, ее наименование, ISBN-код, издатель и фамилия автора, наименование пользователя, который продает эту книгу по указанной цене, и его адрес электронной почты.

На этом заканчивается процедура разработки Web-сервиса. Его полный код приведен в листинге 8.7.

Листинг 8.7

```
Imports System.Web.Services


<WebService(Namespace:="http://localhost/")> _
Public Class Service1
  Inherits System.Web.Services.WebService


#Region " Web Services Designer Generated Code "


  Public Sub New()
    MyBase.New()


    'This call is required by the Web Services Designer.
```

```
    InitializeComponent()

    'Add your own initialization code after the InitializeComponent()
call

  End Sub

  'Required by the Web Services Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Web Services Designer
  'It can be modified using the Web Services Designer.
  'Do not modify it using the code editor.
  Friend WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlUpdateCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlDeleteCommand1 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
  Friend WithEvents ClientSqlDataAdapter As
System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents SqlSelectCommand2 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand2 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlUpdateCommand2 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlDeleteCommand2 As System.Data.SqlClient.SqlCommand
  Friend WithEvents BooksSqlDataAdapter As
System.Data.SqlClient.SqlDataAdapter
  Friend WithEvents SqlSelectCommand3 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlInsertCommand3 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlUpdateCommand3 As System.Data.SqlClient.SqlCommand
  Friend WithEvents SqlDeleteCommand3 As System.Data.SqlClient.SqlCommand
  Friend WithEvents PricesSqlDataAdapter As
System.Data.SqlClient.SqlDataAdapter
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlUpdateCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlDeleteCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.ClientSqlDataAdapter = New System.Data.SqlClient.SqlDataAdapter()
```

```
    Me.SqlSelectCommand2 = New System.Data.SqlClient.SqlCommand()
    Me.SqlInsertCommand2 = New System.Data.SqlClient.SqlCommand()
    Me.SqlUpdateCommand2 = New System.Data.SqlClient.SqlCommand()
    Me.SqlDeleteCommand2 = New System.Data.SqlClient.SqlCommand()
    Me.BooksSqlDataAdapter = New System.Data.SqlClient.SqlDataAdapter()
    Me.SqlSelectCommand3 = New System.Data.SqlClient.SqlCommand()
    Me.SqlInsertCommand3 = New System.Data.SqlClient.SqlCommand()
    Me.SqlUpdateCommand3 = New System.Data.SqlClient.SqlCommand()
    Me.SqlDeleteCommand3 = New System.Data.SqlClient.SqlCommand()
    Me.PricesSqlDataAdapter = New System.Data.SqlClient.SqlDataAdapter()
    '
    'SqlSelectCommand1
    '
    Me.SqlSelectCommand1.CommandText = "SELECT COUNT(IDLOGIN) AS CID FROM
CLIENTS WHERE (IDLOGIN = @LOG)"
    Me.SqlSelectCommand1.Connection = Me.SqlConnection1
    Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.VarChar,
30, "IDLOGIN"))
    '
    'SqlConnection1
    '
    Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Books;persist security info=False;user id=sa;" & _
    "workstation id=BHV-9IEMVL4EOER;packet size=4096"
    '
    'SqlInsertCommand1
    '
    Me.SqlInsertCommand1.CommandText = "INSERT INTO CLIENTS (IDLOGIN,
PASSWORD, NAIM, EMAIL, PHONE) VALUES (@IDLOGIN, @PASSWORD, @NAIM, @EMAIL,
@PHONE)"
    Me.SqlInsertCommand1.Connection = Me.SqlConnection1
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@IDLOGIN",
System.Data.SqlDbType.VarChar, 30, "IDLOGIN"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PASSWORD",
System.Data.SqlDbType.VarChar, 10, "PASSWORD"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
```

```
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@EMAIL",
System.Data.SqlDbType.VarChar, 30, "EMAIL"))
    Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PHONE",
System.Data.SqlDbType.VarChar, 20, "PHONE"))
    '
    'SqlUpdateCommand1
    '
    Me.SqlUpdateCommand1.CommandText = "UPDATE CLIENTS SET IDLOGIN =
@IDLOGIN, PASSWORD = @PASSWORD, NAIM = @NAIM,
EMAIL = @EMAIL, PHONE = @PHONE WHERE (IDLOGIN = @IDLOGIN)"
    Me.SqlUpdateCommand1.Connection = Me.SqlConnection1
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@IDLOGIN",
System.Data.SqlDbType.VarChar, 30, "IDLOGIN"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PASSWORD",
System.Data.SqlDbType.VarChar, 10, "PASSWORD"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@EMAIL",
System.Data.SqlDbType.VarChar, 30, "EMAIL"))
    Me.SqlUpdateCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PHONE",
System.Data.SqlDbType.VarChar, 20, "PHONE"))
    '
    'SqlDeleteCommand1
    '
    Me.SqlDeleteCommand1.CommandText = "DELETE FROM CLIENTS WHERE
(IDLOGIN = @Original_IDLOGIN) AND (EMAIL =
@Original_EMAIL OR @Original_EMAIL IS NULL AND EMAIL IS NULL) AND (NAIM =
@Original_NAIM OR " & _
    "@Original_NAIM IS NULL AND NAIM IS NULL) AND (PASSWORD =
@Original_PASSWORD OR @Original_PASSWORD IS NULL AND PASSWORD IS NULL)
AND (PHONE = @Original_PHONE OR
@Original_PHONE IS NULL AND PHONE IS NULL)"
    Me.SqlDeleteCommand1.Connection = Me.SqlConnection1
    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_IDLOGIN",
System.Data.SqlDbType.VarChar, 30, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "IDLOGIN",
System.Data.DataRowVersion.Original, Nothing))
    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_EMAIL",
```

```
System.Data.SqlDbType.VarChar, 30, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "EMAIL",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_NAIM",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "NAIM",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_PASSWORD",
System.Data.SqlDbType.VarChar, 10, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "PASSWORD",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_PHONE",
System.Data.SqlDbType.VarChar, 20, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "PHONE",
System.Data.DataRowVersion.Original, Nothing))
    '

    'ClientSqlDataAdapter
    '

    Me.ClientSqlDataAdapter.DeleteCommand = Me.SqlDeleteCommand1

    Me.ClientSqlDataAdapter.InsertCommand = Me.SqlInsertCommand1

    Me.ClientSqlDataAdapter.SelectCommand = Me.SqlSelectCommand1

    Me.ClientSqlDataAdapter.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "CLIENTS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("IDLOGIN", "IDLOGIN"), New
System.Data.Common.DataColumnMapping("PASSWORD", "PASSWORD"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("EMAIL", "EMAIL"), New
System.Data.Common.DataColumnMapping("PHONE", "PHONE")})})

    Me.ClientSqlDataAdapter.UpdateCommand = Me.SqlUpdateCommand1

    '

    'SqlSelectCommand2
    '

    Me.SqlSelectCommand2.CommandText = "SELECT ISBN, NAIM, PUBLISHER,
AUTHOR, BOOKVALUE, BOOKYEAR FROM BOOKS"

    Me.SqlSelectCommand2.Connection = Me.SqlConnection1

    '

    'SqlInsertCommand2
    '

    Me.SqlInsertCommand2.CommandText = "INSERT INTO BOOKS (ISBN, NAIM,
PUBLISHER, AUTHOR, BOOKVALUE, BOOKYEAR) VALUES
(@ISBN, @NAIM, @PUBLISHER, @AUTHOR, @BOOKVALUE, @BOOKYEAR)"
```

```
    Me.SqlInsertCommand2.Connection = Me.SqlConnection1
    Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, "ISBN"))
    Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
    Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, "PUBLISHER"))
    Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@AUTHOR",
System.Data.SqlDbType.VarChar, 30, "AUTHOR"))
    Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKVALUE",
System.Data.SqlDbType.Int, 4, "BOOKVALUE"))
    Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, "BOOKYEAR"))
    '
    'SqlUpdateCommand2
    '
    Me.SqlUpdateCommand2.CommandText = "UPDATE BOOKS SET ISBN = @ISBN,
NAIM = @NAIM, PUBLISHER = @PUBLISHER, AUTHOR =
@AUTHOR, BOOKVALUE = @BOOKVALUE, BOOKYEAR = @BOOKYEAR WHERE (ISBN =
@Original_ISBN) _
AND (AUTHOR = @Original_AUTHOR OR @Original_AUTHOR IS NULL AND AUTHOR IS
NULL) & _
    AND (BOOKVALUE = @Original_BOOKVALUE OR @Original_BOOKVALUE IS NULL
AND BOOKVALUE IS NULL) AND (BOOKYEAR = @Original_BOOKYEAR OR
@Original_BOOKYEAR IS NULL AND & _
    BOOKYEAR IS NULL) AND (NAIM = @Original_NAIM OR @Original_NAIM IS
NULL AND NAIM & _
    IS NULL) AND (PUBLISHER = @Original_PUBLISHER OR @Original_PUBLISHER
IS NULL AND & _
    PUBLISHER IS NULL); SELECT ISBN, NAIM, PUBLISHER, AUTHOR, BOOKVALUE,
BOOKYEAR
    FROM BOOKS WHERE (ISBN = @ISBN)"
    Me.SqlUpdateCommand2.Connection = Me.SqlConnection1
    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, "ISBN"))
    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
```

```
    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PUBLISHER",
System.Data.SqlDbType.VarChar, 50, "PUBLISHER"))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@AUTHOR",
System.Data.SqlDbType.VarChar, 30, "AUTHOR"))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKVALUE",
System.Data.SqlDbType.Int, 4, "BOOKVALUE"))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@BOOKYEAR",
System.Data.SqlDbType.Int, 4, "BOOKYEAR"))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_AUTHOR",
System.Data.SqlDbType.VarChar, 30, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "AUTHOR",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_BOOKVALUE",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "BOOKVALUE",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_BOOKYEAR",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "BOOKYEAR",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_NAIM",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "NAIM",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_PUBLISHER",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "PUBLISHER",
System.Data.DataRowVersion.Original, Nothing))
    '

    'SqlDeleteCommand2
    '

    Me.SqlDeleteCommand2.CommandText = "DELETE FROM BOOKS WHERE
(ISBN = @Original_ISBN) AND (AUTHOR = @Original_AUTHOR OR" & _
```

```
    " @Original_AUTHOR IS NULL AND AUTHOR IS NULL) AND
(BOOKVALUE = @Original_BOOKVALUE OR @Original_BOOKVALUE IS NULL AND
BOOKVALUE IS NULL) AND (BOOKYEAR =
@Original_BOOKYEAR OR @Original_BOOKYEAR IS NULL AND BOOKYEAR IS NULL)
AND (NAIM =
@Original_NAIM OR @Original_NAIM IS NULL AND NAIM IS NULL) AND (PUBLISHER
= @Original_PUBLISHER OR @Original_PUBLISHER IS NULL AND PUBLISHER IS
NULL)"

    Me.SqlDeleteCommand2.Connection = Me.SqlConnection1

    Me.SqlDeleteCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_AUTHOR",
System.Data.SqlDbType.VarChar, 30, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "AUTHOR",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_BOOKVALUE",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "BOOKVALUE",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_BOOKYEAR",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "BOOKYEAR",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_NAIM",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "NAIM",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_PUBLISHER",
System.Data.SqlDbType.VarChar, 50, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "PUBLISHER",
System.Data.DataRowVersion.Original, Nothing))
    '

    'BooksSqlDataAdapter
    '

    Me.BooksSqlDataAdapter.DeleteCommand = Me.SqlDeleteCommand2

    Me.BooksSqlDataAdapter.InsertCommand = Me.SqlInsertCommand2

    Me.BooksSqlDataAdapter.SelectCommand = Me.SqlSelectCommand2

    Me.BooksSqlDataAdapter.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
```

```
System.Data.Common.DataTableMapping("Table", "BOOKS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("PUBLISHER", "PUBLISHER"), New
System.Data.Common.DataColumnMapping("AUTHOR", "AUTHOR"), New
System.Data.Common.DataColumnMapping("BOOKVALUE", "BOOKVALUE"), New
System.Data.Common.DataColumnMapping("BOOKYEAR", "BOOKYEAR")})})
    Me.BooksSqlDataAdapter.UpdateCommand = Me.SqlUpdateCommand2
    '
    'SqlSelectCommand3
    '
    Me.SqlSelectCommand3.CommandText = "SELECT IDPRICE, ISBN, PRICE,
LOGIN FROM PRICES"
    Me.SqlSelectCommand3.Connection = Me.SqlConnection1
    '
    'SqlInsertCommand3
    '
    Me.SqlInsertCommand3.CommandText = "INSERT INTO PRICES (ISBN, PRICE,
LOGIN) VALUES (@ISBN, @PRICE, @LOGIN)"
    Me.SqlInsertCommand3.Connection = Me.SqlConnection1
    Me.SqlInsertCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, "ISBN"))
    Me.SqlInsertCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, "PRICE"))
    Me.SqlInsertCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOGIN",
System.Data.SqlDbType.VarChar, 30, "LOGIN"))
    '
    'SqlUpdateCommand3
    '
    Me.SqlUpdateCommand3.CommandText = "UPDATE PRICES SET ISBN = @ISBN,
PRICE = @PRICE, LOGIN = @LOGIN WHERE (IDPRICE = @Original_IDPRICE) AND
(ISBN = @Original_ISBN OR @Original_ISBN IS NULL AND ISBN IS NULL) AND
(LOGIN = @Original_LOGIN OR @Original_LOGIN IS NULL AND LOGIN IS NULL)
AND (PRICE = @Original_PRICE OR @Original_PRICE IS NULL AND PRICE IS
NULL);
    SELECT IDPRICE, ISBN, PRICE, LOGIN FROM PRICES WHERE
(IDPRICE = @IDPRICE)"
    Me.SqlUpdateCommand3.Connection = Me.SqlConnection1
    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@ISBN",
System.Data.SqlDbType.VarChar, 15, "ISBN"))
```

```
    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PRICE", System.Data.SqlDbType.Int,
4, "PRICE"))

    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOGIN",
System.Data.SqlDbType.VarChar, 30, "LOGIN"))

    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_IDPRICE",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "IDPRICE",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_LOGIN",
System.Data.SqlDbType.VarChar, 30, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "LOGIN",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_PRICE",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "PRICE",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@IDPRICE", System.Data.SqlDbType.Int,
4, "IDPRICE"))
    '

    'SqlDeleteCommand3
    '

    Me.SqlDeleteCommand3.CommandText = "DELETE FROM PRICES WHERE
(IDPRICE = @Original_IDPRICE) AND (ISBN = @Original_ISBN

OR @Original_ISBN IS NULL AND ISBN IS NULL) AND
(LOGIN = @Original_LOGIN OR
@Original_LOGIN IS NULL AND LOGIN IS NULL) AND (PRICE = @Original_PRICE
OR @Original_PRICE IS NULL AND PRICE IS NULL)"

    Me.SqlDeleteCommand3.Connection = Me.SqlConnection1

    Me.SqlDeleteCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_IDPRICE",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "IDPRICE",
System.Data.DataRowVersion.Original, Nothing))

    Me.SqlDeleteCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_ISBN",
System.Data.SqlDbType.VarChar, 15, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "ISBN",
System.Data.DataRowVersion.Original, Nothing))
```

```
    Me.SqlDeleteCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_LOGIN",
System.Data.SqlDbType.VarChar, 30, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "LOGIN",
System.Data.DataRowVersion.Original, Nothing))
    Me.SqlDeleteCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_PRICE",
System.Data.SqlDbType.Int, 4, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "PRICE",
System.Data.DataRowVersion.Original, Nothing))
    '
    'PricesSqlDataAdapter
    '
    Me.PricesSqlDataAdapter.DeleteCommand = Me.SqlDeleteCommand3
    Me.PricesSqlDataAdapter.InsertCommand = Me.SqlInsertCommand3
    Me.PricesSqlDataAdapter.SelectCommand = Me.SqlSelectCommand3
    Me.PricesSqlDataAdapter.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "PRICES", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("IDPRICE", "IDPRICE"), New
System.Data.Common.DataColumnMapping("ISBN", "ISBN"), New
System.Data.Common.DataColumnMapping("PRICE", "PRICE"), New
System.Data.Common.DataColumnMapping("LOGIN", "LOGIN")})})
    Me.PricesSqlDataAdapter.UpdateCommand = Me.SqlUpdateCommand3

  End Sub

  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    'CODEGEN: This procedure is required by the Web Services Designer
    'Do not modify it using the code editor.
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub

#End Region

  <WebMethod()> Public Function LogExist(ByVal LOG As String) As Boolean
```

```
    Dim DS As New DataSet()
    Dim t As Boolean
    SqlSelectCommand1.Parameters("@LOG").Value = LOG
    ClientSqlDataAdapter.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") > 0 Then
      t = True
    Else
      t = False
    End If
    DS.Dispose()
    LogExist = t
  End Function


  <WebMethod()> Public Function RegUser(ByVal LOG As String, ByVal
PASSWORD As String, ByVal NAIM As String, ByVal EMAIL As String, ByVal
Phone As String) As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    SqlSelectCommand1.Parameters("@LOG").Value = LOG
    ClientSqlDataAdapter.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") > 0 Then
      t = False
    Else
      SqlInsertCommand1.Parameters("@IDLOGIN").Value = LOG
      SqlInsertCommand1.Parameters("@PASSWORD").Value = PASSWORD
      SqlInsertCommand1.Parameters("@NAIM").Value = NAIM
      SqlInsertCommand1.Parameters("@EMAIL").Value = EMAIL
      SqlInsertCommand1.Parameters("@PHONE").Value = Phone
      SqlConnection1.Open()
      SqlInsertCommand1.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    DS.Dispose()
    RegUser = t
  End Function
  <WebMethod()> Public Function EditUser(ByVal LOG As String, ByVal
PASSWORD As String, ByVal NAIM As String, ByVal EMAIL As String, ByVal
PHONE As String) As Boolean
    Dim DS As New DataSet()
```

```
    Dim t As Boolean
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select count(IDLOGIN) as CID from
CLIENTS where ((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@IDLOGIN", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@IDLOGIN").Value = LOG
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PASSWORD", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@PASSWORD").Value = PASSWORD
    DataCommand.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") = 0 Then
      t = False
    Else
      SqlUpdateCommand1.Parameters("@IDLOGIN").Value = LOG
      SqlUpdateCommand1.Parameters("@PASSWORD").Value = PASSWORD
      SqlUpdateCommand1.Parameters("@NAIM").Value = NAIM
      SqlUpdateCommand1.Parameters("@EMAIL").Value = EMAIL
      SqlUpdateCommand1.Parameters("@PHONE").Value = PHONE
      SqlConnection1.Open()
      SqlUpdateCommand1.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    DS.Dispose()
    EditUser = t
  End Function


  <WebMethod()> Public Function AddBook(ByVal LOG As String, ByVal
PASSWORD As String, ByVal NAIM As String, ByVal ISBN As String, ByVal
AUTHOR As String, ByVal BOOKVALUE As Integer, ByVal BOOKYEAR As Integer)
As Boolean
    Dim DS As New DataSet()
    Dim t As Boolean
    Dim n As String
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim DataCommand2 As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select count(IDLOGIN) as CID from
CLIENTS where ((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))"
```

```
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@IDLOGIN", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@IDLOGIN").Value = LOG
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PASSWORD", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@PASSWORD").Value = PASSWORD
    DataCommand.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") = 0 Then
      t = False
    Else
      SelectCommand = "Select NAIM from CLIENTS where IDLOGIN=@LOG"
      DataCommand2 = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
      DataCommand2.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 20))
      DataCommand2.SelectCommand.Parameters("@LOG").Value = LOG
      DS.Clear()
      DataCommand2.Fill(DS)
      n = DS.Tables(0).Rows(0).Item("NAIM")
      BooksSqlDataAdapter.InsertCommand.Parameters("@ISBN").Value = ISBN
      BooksSqlDataAdapter.InsertCommand.Parameters("@NAIM").Value = NAIM
      BooksSqlDataAdapter.InsertCommand.Parameters("@PUBLISHER").Value =
n
      BooksSqlDataAdapter.InsertCommand.Parameters("@AUTHOR").Value =
AUTHOR
      BooksSqlDataAdapter.InsertCommand.Parameters("@BOOKVALUE").Value =
BOOKVALUE
      BooksSqlDataAdapter.InsertCommand.Parameters("@BOOKYEAR").Value =
BOOKYEAR
      SqlConnection1.Open()
      BooksSqlDataAdapter.InsertCommand.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    AddBook = t
  End Function
  <WebMethod()> Public Function AddPrice(ByVal LOG As String, ByVal
PASSWORD As String, ByVal ISBN As String, ByVal PRICE As Integer) As
Boolean
    Dim DS As New DataSet()
```

```
    Dim t As Boolean
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select count(IDLOGIN) as CID from
CLIENTS where ((IDLOGIN = @IDLOGIN) AND (PASSWORD=@PASSWORD))"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@IDLOGIN", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@IDLOGIN").Value = LOG
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PASSWORD", System.Data.SqlDbType.Char, 20))
    DataCommand.SelectCommand.Parameters("@PASSWORD").Value = PASSWORD
    DataCommand.Fill(DS)
    If DS.Tables(0).Rows(0).Item("CID") = 0 Then
      t = False
    Else
      PricesSqlDataAdapter.InsertCommand.Parameters("@ISBN").Value = ISBN
      PricesSqlDataAdapter.InsertCommand.Parameters("@PRICE").Value =
PRICE
      PricesSqlDataAdapter.InsertCommand.Parameters("@LOGIN").Value = LOG
      SqlConnection1.Open()
      PricesSqlDataAdapter.InsertCommand.ExecuteNonQuery()
      SqlConnection1.Close()
      t = True
    End If
    AddPrice = t
  End Function
  <WebMethod()> Public Function FindBooks(ByVal ISBN As String, ByVal
PUBLISHER As String, ByVal NAIM As String, ByVal AUTHOR As String, ByVal
PRICE1 As Integer, ByVal PRICE2 As Integer) As DataSet
    Dim DS As New DataSet()
    Dim DataCommand As Data.SqlClient.SqlDataAdapter
    Dim SelectCommand As String = "select PRICES.PRICE AS PRICE,
BOOKS.NAIM AS NAIM, BOOKS.PUBLISHER AS PUBL, BOOKS.AUTHOR AS AUTHOR,
BOOKS.ISBN AS ISBN,CLIENTS.NAIM AS CLNAIM,CLIENTS.EMAIL AS CLEMAIL FROM
PRICES INNER JOIN BOOKS ON PRICES.ISBN = BOOKS.ISBN INNER JOIN CLIENTS ON
PRICES.LOGIN = CLIENTS.IDLOGIN where ("
    SelectCommand = SelectCommand + "(PRICES.PRICE >= @PRICE1) AND
(PRICES.PRICE <=@PRICE2)"
    If ISBN <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.ISBN=@ISBN)"
    End If
    If PUBLISHER <> "" Then
```

```
      SelectCommand = SelectCommand + "AND (BOOKS.PUBLISHER=@PUBLISHER)"
    End If
    If NAIM <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.NAIM LIKE @NAIM)"
    End If
    If AUTHOR <> "" Then
      SelectCommand = SelectCommand + "AND (BOOKS.AUTHOR LIKE @AUTHOR)"
    End If
    SelectCommand = SelectCommand + ")"
    DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PRICE1", System.Data.SqlDbType.Int, 4))
    DataCommand.SelectCommand.Parameters("@PRICE1").Value = PRICE1
    DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PRICE2", System.Data.SqlDbType.Int, 4))
    DataCommand.SelectCommand.Parameters("@PRICE2").Value = PRICE2
    If ISBN <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@ISBN", System.Data.SqlDbType.Char, 15))
      DataCommand.SelectCommand.Parameters("@ISBN").Value = ISBN
    End If
    If PUBLISHER <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@PUBLISHER", System.Data.SqlDbType.Char,
50))
      DataCommand.SelectCommand.Parameters("@PUBLISHER").Value =
PUBLISHER
    End If
    If NAIM <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@NAIM", System.Data.SqlDbType.Char, 50))
      DataCommand.SelectCommand.Parameters("@NAIM").Value = "'%" + NAIM +
"%'"
    End If
    If AUTHOR <> "" Then
      DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@AUTHOR", System.Data.SqlDbType.Char, 30))
      DataCommand.SelectCommand.Parameters("@AUTHOR").Value = "'%" +
AUTHOR + "%'"
    End If
    DataCommand.Fill(DS)
```

```
    FindBooks = DS
  End Function
End Class
```

Разработка сайта

Структура сайта фактически является зеркалом структуры Web-сервиса, с которым он связан. Каждая Web-страница предназначается для работы с той или иной функцией Web-сервиса. На стартовой странице мы разместим набор органов управления для регистрации новых пользователей, а также предоставим возможность уже зарегистрированным пользователям ввести свой пароль и логин и получить доступ ко всем остальным возможностям сайта.

Как мы знаем, логин и пароль передаются каждой функции Web-сервиса для того, чтобы сервис мог убедиться, что пользователь действительно тот, за кого себя выдает, и имеет право выполнять ту или иную операцию. Естественно, было бы неправильным заставлять пользователя раз за разом указывать свой логин и пароль для выполнения каждой операции, поэтому после первичной идентификации эти данные будут храниться в качестве элементов коллекции `SessionState`. Впрочем, эти технические особенности мы рассмотрим на готовых примерах.

Итак, начнем мы с разработки стартовой Web-страницы сайта. Как уже говорилось ранее, она предназначена для регистрации и идентификации пользователей. HTML-код этой Web-страницы с наименованием Default.aspx, в том виде, как он будет храниться на сервере, приведен в листинге 8.8.

Листинг 8.8

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="default.aspx.vb" Inherits="B2BSite.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Стартовая страница</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
```

```
   <TABLE id="Table1" style="Z-INDEX: 101; LEFT: 14px; WIDTH: 567px;
POSITION: absolute; TOP: 16px; HEIGHT: 352px" cellSpacing="1"
cellPadding="1" width="567" border="1">
  <TR>
   <TD style="WIDTH: 287px; HEIGHT: 42px" colSpan="1"
rowSpan="1">Зарегистрироваться</TD>
   <TD style="HEIGHT: 42px">Я уже зарегистрирован</TD>
  </TR>
  <TR>
   <TD style="WIDTH: 287px; HEIGHT: 59px">Логин    
            
    <asp:TextBox id="TextBox1" runat="server"></asp:TextBox></TD>
   <TD style="HEIGHT: 59px">Логин    
    <asp:TextBox id="TextBox6" runat="server"></asp:TextBox></TD>
  </TR>
  <TR>
<TD style="WIDTH: 287px">Пароль     
         
    <asp:TextBox id="TextBox2" runat="server"></asp:TextBox></TD>
   <TD>Пароль 
    <asp:TextBox id="TextBox7" runat="server"></asp:TextBox></TD>
  </TR>
  <TR>
   <TD style="WIDTH: 287px">Наименование
    <asp:TextBox id="TextBox3" runat="server"></asp:TextBox></TD>
   <TD></TD>
  </TR>
  <TR>
   <TD style="WIDTH: 287px">E-mail      
            
    <asp:TextBox id="TextBox4" runat="server"></asp:TextBox></TD>
   <TD></TD>
  </TR>
  <TR>
<TD style="WIDTH: 287px">Телефон      
      
    <asp:TextBox id="TextBox5" runat="server"></asp:TextBox></TD>
   <TD></TD>
  </TR>
```

```
  <TR>
   <TD style="WIDTH: 287px">       
           
    <asp:Button id="Button1" runat="server" Width="155px"
Text="Зарегистрироваться"></asp:Button></TD>
   <TD>           
    <asp:Button id="Button2" runat="server" Width="146px"
Text="Подтвердить"></asp:Button></TD>
  </TR>
   </TABLE>
  </form>
 </body>
</HTML>
```

Внешний вид созданной Web-страницы показан на рис. 8.1.

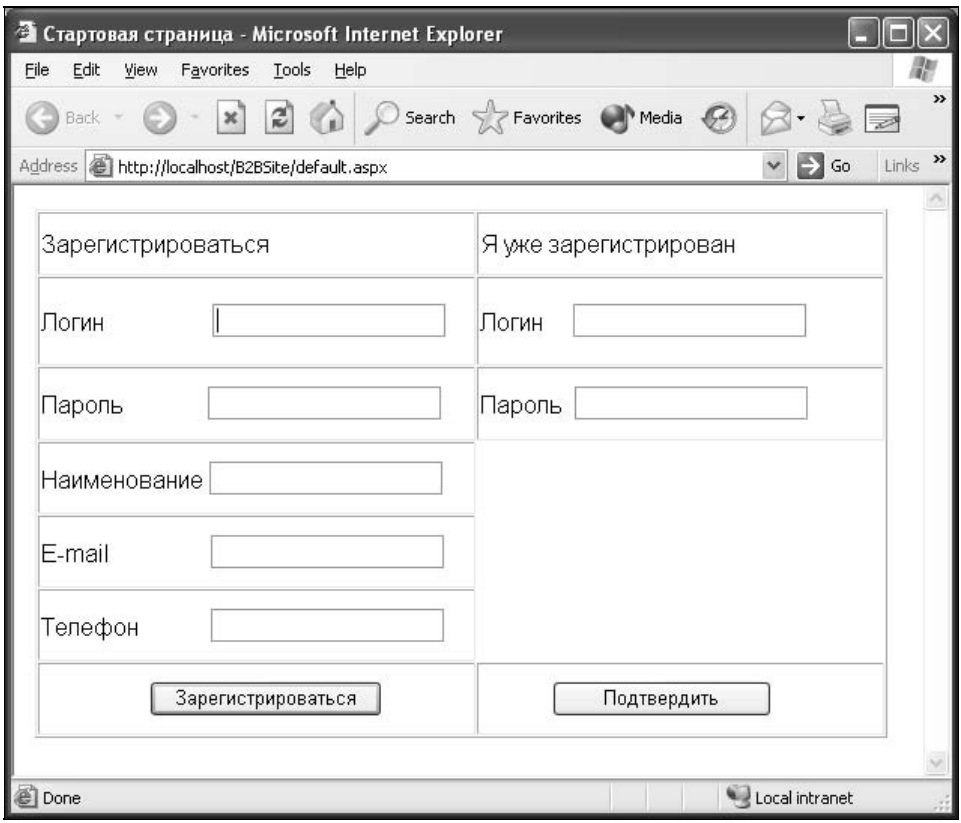

Рис. 8.1. Внешний вид стартовой Web-страницы, осуществляющей регистрацию и идентификацию пользователей

Код класса, реализующего эту Web-страницу, приведен в листинге 8.9.

Листинг 8.9

```
Public Class WebForm1
  Inherits System.Web.UI.Page
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox4 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox5 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents TextBox6 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox7 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Button2 As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()

  End Sub

  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

#End Region

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

```
    Dim serv = New B2BSite.localhost.Service1()
    If serv.LogExist(TextBox1.Text) Then
      Response.Redirect("LogExist.aspx")
    Else
      serv.RegUSer(TextBox1.Text, TextBox2.Text, TextBox3.Text,
TextBox4.Text, TextBox5.Text)
      Session.Clear()
      Session.Add("log", TextBox1.Text)
      Session.Add("pass", TextBox2.Text)
      Response.Redirect("main.aspx")
    End If
  End Sub


  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Session.Clear()
    Session.Add("log", TextBox6.Text)
    Session.Add("pass", TextBox7.Text)
    Response.Redirect("main.aspx")
  End Sub
End Class
```

Вся основная функциональность класса, реализующего рассматриваемую нами Web-страницу, сосредоточена в двух функциях, обрабатывающих нажатия кнопок. Начнем мы с рассмотрения обработчика нажатия кнопки, используемой пользователем для подтверждения регистрационных данных.

При приеме регистрационных данных прежде всего следует проверить, не содержится ли уже в базе данных указанный логин. Для этого мы используем функцию `LogExist`, входящую в состав Web-сервиса. Если функция возвращает значение `True`, то логин, переданный пользователем, уже существует, и регистрацию выполнять нельзя. В этом случае пользователь переадресуется на Web-страницу LogExist.aspx при помощи метода `Response.Redirect`. На этой Web-странице всего лишь отображается текст, сообщающий пользователю о возникшей ошибке.

Если указанного пользователем логина в базе данных еще нет, регистрацию проводить можно. Для этого мы вызываем функцию `RegUser`, передавая ей в качестве параметров информацию, введенную пользователем в текстовые поля. После вызова этой функции следует сохранить логин и пароль пользователя в элементах коллекции `Session`, чтобы использовать их во время всего сеанса работы пользователя с сайтом без дополнительного ввода. Для этого мы сначала принудительно очищаем содержимое этой коллекции при помощи метода `Clear`, а затем с помощью вызовов метода `Add` добавляем в коллекцию элементы `log` и `pass`. В качестве значений этих элементов, мы,

естественно, используем указанные пользователем логин и пароль, соответственно.

После того как указанные пароль и логин сохранены, пользователя следует переадресовать на Web-страницу, которая предоставляет доступ ко всем функциям Web-сервиса. В нашем случае эта Web-страница будет носить наименование Main.aspx. Переадресация пользователя, соответственно, будет производиться при помощи метода `Response.Redirect`.

В том случае если пользователь уже зарегистрирован, нам остается лишь сохранить в коллекции `Session` его логин и пароль, а затем отправить его на основную Web-страницу. Код, используемый для этого, уже встречался в предыдущей функции.

Теперь стоит сделать маленькое отступление и рассказать о несколько раз упомянутом объекте `Session`. Этот объект реализует механизм поддержки сеансов работы пользователей, который снимает с разработчика множество ненужной работы, в частности, позволяет идентифицировать пользователя и отслеживать его работу с сайтом в текущий момент. То есть, разработчик имеет возможность сопровождать пользователя в его непрерывном путешествии по сайту и корректировать действия сайта. Простейший пример — покупка в онлайновом интернет-магазине. С момента выбора пользователем товаров и до подтверждения им покупки необходимо следить за его действиями на сайте, формировать корзину его заказа, составлять список заказанных товаров, считать сумму заказа и скидки. Естественно, при этом пользователь посетит достаточно много различных Web-страниц, и вся накопленная информация при переходах между ними должна сохраняться. Для обеспечения этих возможностей можно было бы использовать технологию cookies или взаимодействие с базами данных. Так, при начале работы пользователя с сайтом можно бы было генерировать уникальный cookie, записывать его на машину пользователя, и любая Web-страница могла бы добавлять в него информацию. Однако, как мы знаем, cookie может быть только текстовой строкой, причем размер ее ограничен. Поэтому при больших объемах сохраняемой информации cookies использовать неудобно. Более того, нельзя забывать о том, что пользователи имеют возможность отключить запись cookies на свои машины.

Ограничения на объем хранимой информации снимает использование баз данных. Тем более что без организации баз данных весьма трудно реализовать онлайновый магазин с его стандартным набором возможностей. Можно, но действительно трудно. Однако применение баз данных для сопровождения сеансов работы пользователей потребует хранить информацию не только о товарах, продающихся в магазине, но еще и обо всех действиях пользователей. Это может занять дополнительное время в работе сайта. Конечно, при современных мощностях Web-серверов обработка двух-трех лишних таблиц не может занять много времени, однако следует осознавать, что в подобные таблицы будет записываться информация не только о со-

вершенных покупках, но и обо всех сеансах работы, которые не закончились покупками. Если учитывать, что обычно только десять процентов посещений сайта заканчиваются покупками (и это еще хороший процент), то объем базы данных, обслуживающей механизм регистрации сеансов пользователей, увеличивается в десять раз по сравнению с ее минимально возможным размером. Эта сложность, естественно, тоже может быть ликвидирована, однако это потребует дополнительных действий со стороны администратора или дополнительного служебного Web-приложения.

Итак, нетрудно догадаться, к чему мы ведем. Действительно, ASP.NET предоставляет разработчику готовый механизм отслеживания и сопровождения сеансов. При этом разработчику даже нет нужды досконально разбираться в его способах работы, достаточно просто пользоваться им как коллекцией и использовать его методы и свойства.

Рассмотрим еще раз пример с организацией сеансов посредством cookies. Даже если не упоминать об ограничениях на объем хранимой информации, все равно есть еще одна проблема. Когда мы записываем в cookie уникальный идентификатор сеанса, это означает, что каждая Web-страница, участвующая в обработке сеанса, получит данный идентификатор и на его основе будет строить свою работу. Но что делать, если пользователь внезапно прервет свою работу и зайдет на сайт две недели спустя? Его идентификатор сеанса сохранился, и сайт будет работать с ним так, как будто никакого разрыва не происходило. А за прошедшее время могут измениться, предположим, цены на заказанные товары. Более того, вполне вероятна ситуация, когда заказанный пользователем две недели назад товар уже закончится на складе. Следовательно, необходимо четко знать, сколько времени может длиться один сеанс.

При использовании объекта `Session`, полное наименование которого `HttpSessionState`, мы можем устанавливать максимальную продолжительность сеанса, а также принудительно завершать его. Возможность жесткого завершения сеанса придется весьма кстати в тех случаях, когда пользователю разрешается проводить несколько сеансов работы с сайтом последовательно. Обратимся опять к примеру онлайнового магазина. Когда пользователь завершил покупку, его следует перевести опять на стартовую страницу сайта, откуда он и начинал работу (вдруг он сделает еще одну покупку?). Но при этом его предыдущий сеанс следует закрыть, так как покупки одного пользователя должны отделяться друг от друга.

Время жизни сеанса задается при помощи свойства `Timeout`, значением которого является целое число. Это и есть максимальная длительность сеанса, выраженная в минутах. Таким образом, если предположить, что пользователь будет оформлять покупку на сайте не более трех часов (а в практике бывали случаи, когда пользователь собирал достаточно объемный заказ около трех часов), то значение этого свойства должно равняться ста восьмидесяти. А принудительно завершить сеанс работы можно при помощи метода `Abandon`.

В нашем случае в стандартной коллекции объекта `Session` сохраняются логин и пароль пользователя, которые будут использованы при выполнении кода иных Web-страниц.

Но вернемся к примеру разработки Web-сайта. Теперь следует рассмотреть структуру Web-страницы с наименованием Main.aspx. Эта Web-страница предназначена для поиска книг. То есть, именно на ней пользователь будет указывать критерии поиска по базе данных и получать из нее информацию. Также на этой странице будут размещены гиперссылки на Web-страницы, которые позволяют изменить учетную запись пользователя и добавить или удалить собственные коммерческие предложения в общую базу данных.

HTML-код Web-страницы с наименованием Main.aspx, в том виде, как он будет храниться на сервере, приведен в листинге 8.10.

Листинг 8.10

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="main.aspx.vb"
Inherits="B2BSite.main"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>main</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 21px; POSITION:
absolute; TOP: 22px" runat="server" Width="329px">Поиск информации в базе
данных</asp:Label>
   <asp:TextBox id="TextBox6" style="Z-INDEX: 113; LEFT: 369px; POSITION:
absolute; TOP: 164px" runat="server"></asp:TextBox>
   <asp:TextBox id="TextBox5" style="Z-INDEX: 112; LEFT: 369px; POSITION:
absolute; TOP: 88px" runat="server"></asp:TextBox>
   <asp:Label id="Label7" style="Z-INDEX: 111; LEFT: 369px; POSITION:
absolute; TOP: 129px" runat="server">Цена до</asp:Label>
   <asp:TextBox id="TextBox4" style="Z-INDEX: 110; LEFT: 183px; POSITION:
absolute; TOP: 170px" runat="server"></asp:TextBox>
   <asp:TextBox id="TextBox3" style="Z-INDEX: 109; LEFT: 183px; POSITION:
absolute; TOP: 130px" runat="server"></asp:TextBox>
   <asp:TextBox id="TextBox2" style="Z-INDEX: 108; LEFT: 183px; POSITION:
absolute; TOP: 90px" runat="server"></asp:TextBox>
```

```
   <asp:Label id="Label6" style="Z-INDEX: 107; LEFT: 369px; POSITION:
absolute; TOP: 53px" runat="server">Цена от</asp:Label>

   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 34px; POSITION:
absolute; TOP: 52px" runat="server" Width="54px">ISBN</asp:Label>

   <asp:Label id="Label3" style="Z-INDEX: 103; LEFT: 34px; POSITION:
absolute; TOP: 91px" runat="server">Издательство</asp:Label>

   <asp:Label id="Label4" style="Z-INDEX: 104; LEFT: 34px; POSITION:
absolute; TOP: 130px" runat="server">Наименование</asp:Label>

   <asp:TextBox id="TextBox1" style="Z-INDEX: 105; LEFT: 183px; POSITION:
absolute; TOP: 50px" runat="server"></asp:TextBox>

   <asp:Label id="Label5" style="Z-INDEX: 106; LEFT: 34px; POSITION:
absolute; TOP: 170px" runat="server">Автор</asp:Label>

   <asp:Button id="Button1" style="Z-INDEX: 114; LEFT: 213px; POSITION:
absolute; TOP: 210px" runat="server" Width="197px"
Text="Поиск"></asp:Button>

   <asp:DataGrid id=DataGrid1 style="Z-INDEX: 115; LEFT: 33px; POSITION:
absolute; TOP: 258px" runat="server" Width="490px" Height="147px"
DataSource="<%# DataSet1 %>" Visible="False">

   </asp:DataGrid>

   <asp:HyperLink id="HyperLink1" style="Z-INDEX: 116; LEFT: 42px;
POSITION: absolute; TOP: 477px" runat="server" Height="21px"
NavigateUrl="adding.aspx">Добавить коммерческое предложение</asp:HyperLink>

   <asp:HyperLink id="HyperLink2" style="Z-INDEX: 117; LEFT: 42px;
POSITION: absolute; TOP: 438px" runat="server"
NavigateUrl="edit.aspx">Изменить учетную запись</asp:HyperLink>

   <asp:HyperLink id="HyperLink3" style="Z-INDEX: 118; LEFT: 42px;
POSITION: absolute; TOP: 510px" runat="server"
NavigateUrl="addbook.aspx">Добавить новую книгу</asp:HyperLink>

  </form>

 </body>

</HTML>
```

Следует отметить, что помимо полей ввода, в которых пользователь указывает критерии поиска книг, кнопки для запуска поиска и таблицы, в которой отображаются результаты поиска, на страницу дизайна мы добавляем также компонент `DataSet`, который будет служить источником данных для таблицы. Так как в Web-сервисе мы не создавали набор данных, напрямую привязанный к элементу `BooksDataAdapter`, импортировать его тип в наше Web-приложение мы не сможем. Поэтому надо создать экземпляр абстрактного набора данных, который приобретет свою структуру в процессе работы функции поиска.

Следует добавить, что при загрузке Web-страницы основная таблица, в которой показываются результаты поиска, не отображается, так как ее содержимого все равно еще нет.

Также на основной Web-странице размещаются три гиперссылки, которые позволяют пользователю переходить к Web-страницам, предоставляющим

доступ к дополнительным функциональным возможностям Web-сервиса. Структуру этих страниц мы рассмотрим несколько позже. Сейчас же сосредоточимся на разработке кода для основной Web-страницы.

Внешний вид этой Web-страницы после того, как пользователь задал параметры поиска и нажал кнопку **Поиск**, приведен на рис. 8.2.

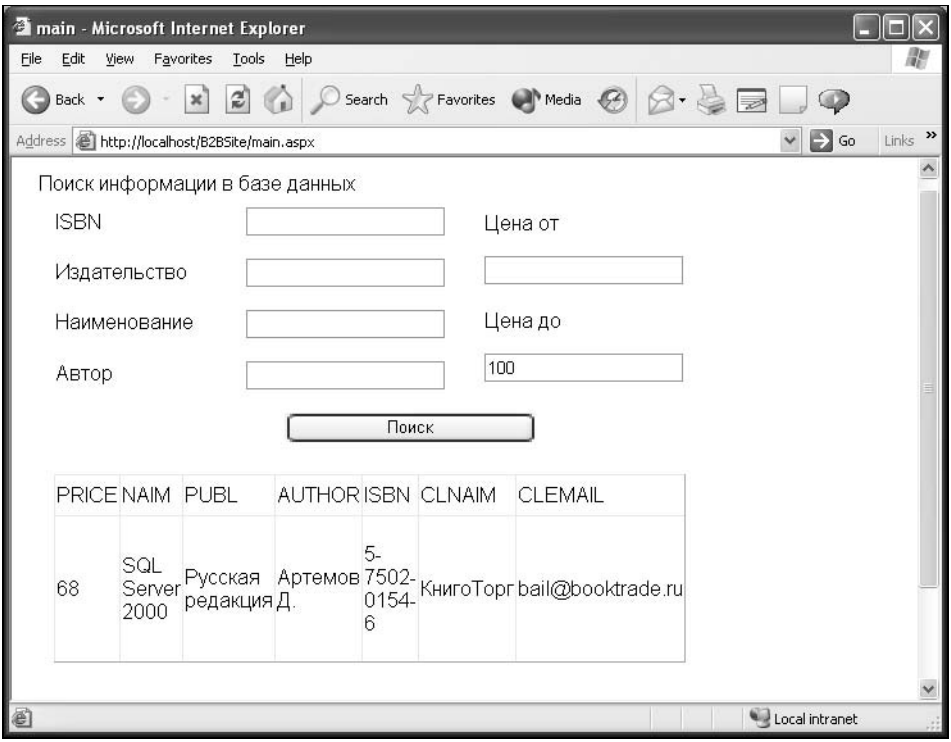

Рис 8.2. Внешний вид Web-страницы Main.aspx

Код класса, реализующего рассматриваемую Web-страницу, приведен в листинге 8.11.

Листинг 8.11

```
Public Class main
  Inherits System.Web.UI.Page
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label
  Protected WithEvents Label2 As System.Web.UI.WebControls.Label
  Protected WithEvents Label3 As System.Web.UI.WebControls.Label
  Protected WithEvents Label4 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
```

```
  Protected WithEvents Label5 As System.Web.UI.WebControls.Label
  Protected WithEvents Label6 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox4 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Label7 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox5 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox6 As System.Web.UI.WebControls.TextBox
  Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
  Protected WithEvents DataSet1 As System.Data.DataSet
  Protected WithEvents HyperLink1 As System.Web.UI.WebControls.HyperLink
  Protected WithEvents HyperLink2 As System.Web.UI.WebControls.HyperLink
  Protected WithEvents HyperLink3 As System.Web.UI.WebControls.HyperLink
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.DataSet1 = New System.Data.DataSet()
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
    '
    'DataSet1
    '
    Me.DataSet1.DataSetName = "NewDataSet"
    Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU")
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()


  End Sub


  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub


#End Region
```

```
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Price1, Price2 As Integer
    Dim serv = New B2BSite.localhost.Service1()
    If TextBox5.Text = "" Then
      Price1 = 0
    Else : Price1 = Convert.ToInt32(TextBox5.Text)
    End If
    If TextBox6.Text = "" Then
      Price2 = 10000
    Else : Price2 = Convert.ToInt32(TextBox6.Text)
    End If
    DataSet1.Clear
    DataSet1.Merge(serv.FindBooks(TextBox1.Text, TextBox2.Text,
TextBox3.Text, TextBox4.Text, Price1, Price2))
    DataGrid1.DataBind()
    DataGrid1.Visible = True
  End Sub
End Class
```

Естественно, без рассмотрения кода функции, обрабатывающей нажатие пользователем на кнопку **Поиск**, не обойтись. Для того, чтобы правильно обработать верхнюю и нижнюю границы ценового диапазона, заданного пользователем, нам потребуются две целочисленные переменные. Также мы объявляем и тут же создаем экземпляр прокси-класса Web-сервиса.

Затем мы осуществляем проверку, ввел ли пользователь границы ценового диапазона. Если соответствующие поля ввода остались пустыми, то для этих переменных мы задаем заведомо подходящие значения. То есть, нижняя граница будет равна нулю, а верхняя — десяти тысячам. С достаточно большой уверенностью можно предполагать, что любая книга уложится в подобный ценовой диапазон.

Если пользователь ввел ценовые значения в соответствующие поля ввода, мы конвертируем их в целочисленный тип. Затем эти значения будут присвоены переменным.

После того как ценовой диапазон определен, можно приступать к формированию набора данных. Перед его заполнением мы принудительно очистим

его при помощи метода `Clear`. А затем остается всего лишь выполнить метод `Merge`, в качестве параметра передав методу вызов функции `FindBooks`. Естественно, функция `FindBooks` тоже требует передачи параметров, и в их качестве мы используем текстовое содержимое четырех полей ввода и две сформированные заранее целочисленные переменные, в которых указаны границы ценового диапазона.

После этого набор данных будет получен, и нам останется лишь связать его с таблицей. Для этого мы вызовем метод `DataGrid1.DataBind`, а затем отобразим таблицу при помощи следующего фрагмента кода:

```
DataGrid1.Visible = True
```

Итак, все опять оказалось не так уж и сложно. Достаточно лишь продумать структуру взаимосвязей и действия, а затем методично разрабатывать приложение.

Рассмотрим теперь структуру Web-страницы, которая позволяет изменять учетную запись пользователя. Эта Web-страница будет называться Edit.aspx. Ее HTML-код, хранящийся на сервере, приведен в листинге 8.12.

Листинг 8.12

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="edit.aspx.vb"
Inherits="B2BSite.edit"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>edit</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 29px; POSITION:
absolute; TOP: 19px" runat="server" Width="204px">Введите новые
данные</asp:Label>
   <asp:TextBox id="TextBox3" style="Z-INDEX: 107; LEFT: 302px; POSITION:
absolute; TOP: 157px" runat="server"></asp:TextBox>
   <asp:TextBox id="TextBox2" style="Z-INDEX: 106; LEFT: 302px; POSITION:
absolute; TOP: 104px" runat="server"></asp:TextBox>
   <asp:Label id="Label4" style="Z-INDEX: 104; LEFT: 29px; POSITION:
absolute; TOP: 163px" runat="server" Width="208px">Телефон</asp:Label>
```

```
   <asp:Label id="Label3" style="Z-INDEX: 103; LEFT: 29px; POSITION:
absolute; TOP: 110px" runat="server" Width="208px">E-mail</asp:Label>
   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 29px; POSITION:
absolute; TOP: 59px" runat="server"
Width="208px">Наименование</asp:Label>
   <asp:TextBox id="TextBox1" style="Z-INDEX: 105; LEFT: 302px; POSITION:
absolute; TOP: 53px" runat="server"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 108; LEFT: 206px; POSITION:
absolute; TOP: 217px" runat="server" Width="163px"
Text="Подтвердить"></asp:Button>
   <asp:Label id="Label5" style="Z-INDEX: 109; LEFT: 44px; POSITION:
absolute; TOP: 264px" runat="server" Width="407px"
Height="32px"></asp:Label>
  </form>
 </body>
</HTML>
```

Внешний вид этой Web-страницы приведен на рис. 8.3.

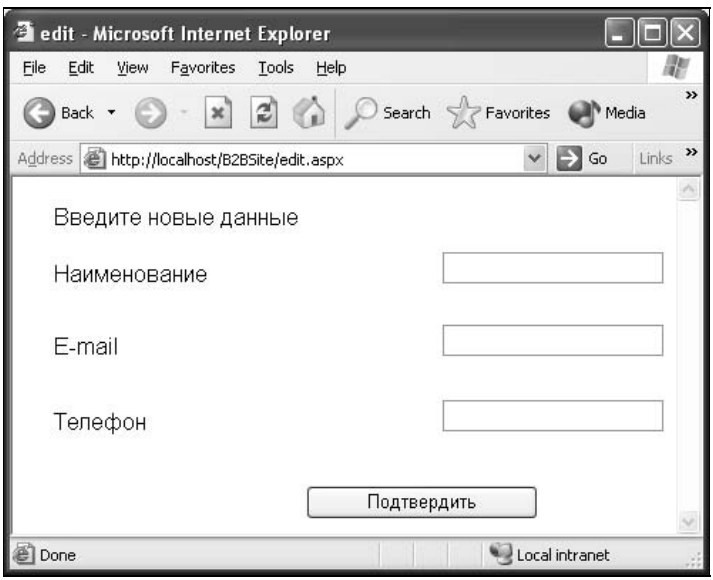

Рис 8.3. Внешний вид Web-страницы Edit.aspx

На этой Web-странице мы размещаем три поля текстового ввода, в которых пользователь сможет указать новое наименование, адрес электронной почты и номер телефона. После нажатия на кнопку осуществится вызов соответствующей функции Web-сервиса. Исходя из результата выполнения этой функции, сформируется текстовое сообщение, которое будет отображено на странице в пространстве компонента `Label5`. Код класса, реализующего

функциональность рассматриваемой нами Web-страницы, приведен в листинге 8.13.

Листинг 8.13

```
Public Class edit
  Inherits System.Web.UI.Page
  Protected WithEvents Label2 As System.Web.UI.WebControls.Label
  Protected WithEvents Label3 As System.Web.UI.WebControls.Label
  Protected WithEvents Label4 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents Label5 As System.Web.UI.WebControls.Label
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()


  End Sub


  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub


#End Region


  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

```
    Dim log, pass As String
    Dim serv = New B2BSite.localhost.Service1()
    Dim Succ As Boolean
    log = Session.Item("log")
    pass = Session.Item("pass")
    Succ = serv.EditUser(log, pass, TextBox1.Text, TextBox2.Text,
TextBox3.Text)
    If Succ Then
      Label5.Text = "Учетная запись успешно изменена"
    Else
      Label5.Text = "Изменить учетную запись не удалось. Повторите
аутентификацию"
    End If
  End Sub
End Class
```

Внешний вид этой Web-страницы после того, как пользователь успешно изменил свою учетную запись, приведен на рис. 8.4.

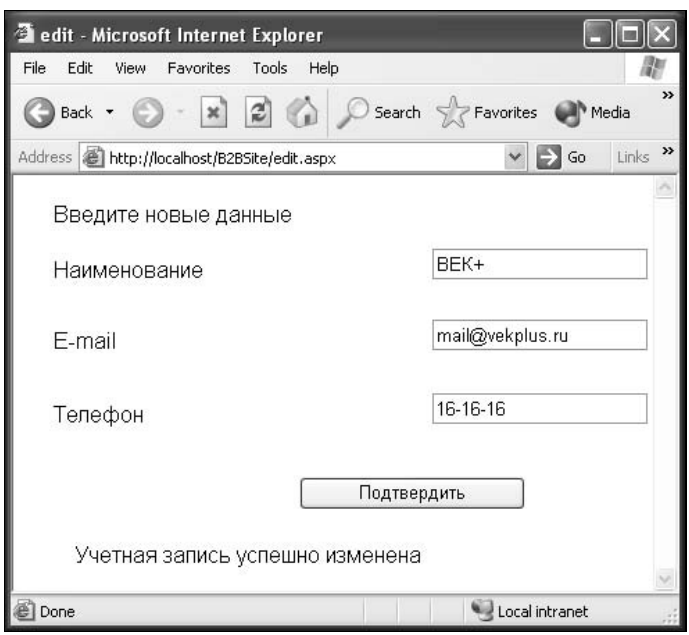

Рис 8.4. Внешний вид Web-страницы Edit.aspx после успешного изменения учетной записи пользователя

Но рассмотрим структуру функции, которая обрабатывает нажатие кнопки пользователем. Для того чтобы изменить учетную запись, необходимо знать

его логин и пароль. Они были введены им на стартовой странице, а затем сохранены в элементах стандартной коллекции `Session`. Соответственно, теперь нам необходимо воспользоваться этими элементами. Доступ к ним осуществляется при помощи свойства `Item` по наименованию элемента коллекции.

После получения логина и пароля пользователя можно вызывать функцию Web-сервиса `EditUser`, которой в качестве параметров мы передаем эти логин и пароль, а также данные, введенные пользователем в текстовые поля ввода. Результат выполнения этой функции присваивается переменной логического типа. Эта переменная будет содержать результат выполнения функции. Основываясь на полученном значении переменной логического типа, мы сформируем текстовое сообщение, отображаемое на Web-странице. Оно будет сигнализировать пользователю о результатах выполнения операции.

Теперь нам осталось лишь разработать последнюю Web-страницу, на которой пользователь сможет помещать в базу данных свои коммерческие предложения. Эта Web-страница носит наименование Adding.aspx. Ее HTML-код приведен в листинге 8.14.

Листинг 8.14

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="adding.aspx.vb" Inherits="B2BSite.adding"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>adding</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 28px; POSITION:
absolute; TOP: 25px" runat="server" Width="287px" Height="13px">Добавить
коммерческое предложение</asp:Label>
   <asp:TextBox id="TextBox2" style="Z-INDEX: 105; LEFT: 150px; POSITION:
absolute; TOP: 106px" runat="server"></asp:TextBox>
   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 28px; POSITION:
absolute; TOP: 68px" runat="server" Width="83px">ISBN</asp:Label>
   <asp:Label id="Label3" style="Z-INDEX: 103; LEFT: 28px; POSITION:
absolute; TOP: 112px" runat="server">Цена</asp:Label>
   <asp:TextBox id="TextBox1" style="Z-INDEX: 104; LEFT: 154px; POSITION:
absolute; TOP: 64px" runat="server"></asp:TextBox>
```

```
   <asp:Button id="Button1" style="Z-INDEX: 106; LEFT: 78px; POSITION:
absolute; TOP: 160px" runat="server" Width="178px"
Text="Подтвердить"></asp:Button>
   <asp:Label id="Label4" style="Z-INDEX: 107; LEFT: 32px; POSITION:
absolute; TOP: 209px" runat="server" Width="265px">Label</asp:Label>
  </form>
 </body>
</HTML>
```

Естественно, одним HTML-листингом не обойтись. Следует рассмотреть код класса, реализующего функциональность этой Web-страницы. Он приведен в листинге 8.15.

Листинг 8.15

```
Public Class adding
  Inherits System.Web.UI.Page
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label
  Protected WithEvents Label2 As System.Web.UI.WebControls.Label
  Protected WithEvents Label3 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button
  Protected WithEvents Label4 As System.Web.UI.WebControls.Label


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()


  End Sub


  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub


#End Region
```

```
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Price As Integer
    Dim serv = New B2BSite.localhost.Service1()
    Dim res As Boolean
    Dim Log, Pass As String
    Price = Convert.ToInt32(TextBox2.Text)
    Log = Session.Item("log")
    Pass = Session.Item("pass")
    res = serv.AddPrice(Log, Pass, TextBox1.Text, Price)
    If res Then
      Label4.Text = "Коммерческое предложение успешно добавлено"
    Else
      Label4.Text = "Внести информацию в базу данных не удалось"
    End If
  End Sub
End Class
```

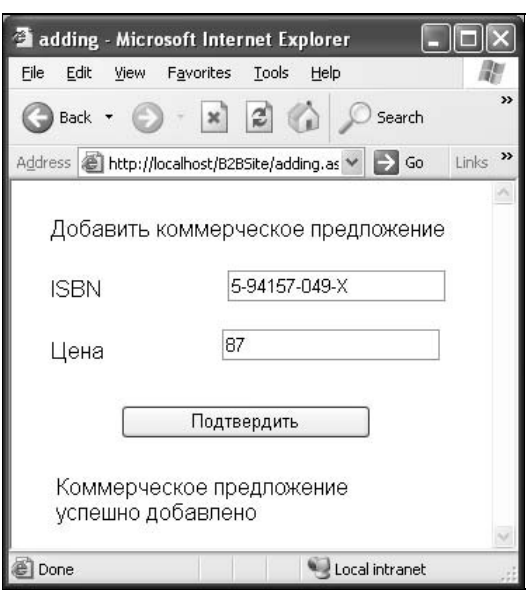

Рис. 8.5. Внешний вид Web-страницы Adding.aspx

Внешний вид этой Web-страницы показан на рис. 8.5.

Нами еще не разработана Web-страница, на которой издательства могут задавать информацию о новых книгах и добавлять ее в базу данных. Назовем ее Addbook.aspx. Ее HTML-код приведен в листинге 8.16.

Листинг 8.16

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="addbook.aspx.vb" Inherits="B2BSite.addbook"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Добавление книги</title>
  <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.0">
  <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
  <meta name="vs_defaultClientScript" content="JavaScript">
  <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 22px; POSITION:
absolute; TOP: 32px" runat="server"
Width="212px">Наименование</asp:Label>
   <asp:TextBox id="TextBox5" style="Z-INDEX: 110; LEFT: 257px; POSITION:
absolute; TOP: 175px" runat="server" Width="198px"></asp:TextBox>
   <asp:TextBox id="TextBox4" style="Z-INDEX: 109; LEFT: 257px; POSITION:
absolute; TOP: 137px" runat="server" Width="198px"></asp:TextBox>
   <asp:TextBox id="TextBox3" style="Z-INDEX: 108; LEFT: 257px; POSITION:
absolute; TOP: 101px" runat="server" Width="198px"></asp:TextBox>
   <asp:TextBox id="TextBox2" style="Z-INDEX: 107; LEFT: 257px; POSITION:
absolute; TOP: 65px" runat="server" Width="198px"></asp:TextBox>
   <asp:Label id="Label6" style="Z-INDEX: 105; LEFT: 22px; POSITION:
absolute; TOP: 179px" runat="server" Width="212px">Год
выпуска</asp:Label>
   <asp:Label id="Label4" style="Z-INDEX: 104; LEFT: 22px; POSITION:
absolute; TOP: 140px" runat="server" Width="212px">Количество
страниц</asp:Label>
   <asp:Label id="Label3" style="Z-INDEX: 103; LEFT: 22px; POSITION:
absolute; TOP: 104px" runat="server" Width="212px">Автор</asp:Label>
   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 22px; POSITION:
absolute; TOP: 68px" runat="server" Width="212px">ISBN</asp:Label>
   <asp:TextBox id="TextBox1" style="Z-INDEX: 106; LEFT: 257px; POSITION:
absolute; TOP: 29px" runat="server" Width="198px"></asp:TextBox>
```

```
   <asp:Button id="Button1" style="Z-INDEX: 111; LEFT: 134px; POSITION:
absolute; TOP: 226px" runat="server" Width="204px"
Text="Подтвердить"></asp:Button>
   <asp:Label id="Label5" style="Z-INDEX: 112; LEFT: 32px; POSITION:
absolute; TOP: 273px" runat="server" Width="252px"
Visible="False"></asp:Label>
  </form>
 </body>
</HTML>
```

Итак, на странице, как обычно, располагается набор полей для ввода информации, текстовая информация, описывающая эти поля, кнопка для подтверждения ввода информации и невидимый до поры компонент `Label`, в котором будет отображаться сообщение о завершении операции. Функциональность этой Web-страницы описывается кодом, приведенным в листинге 8.17.

Листинг 8.17

```
Public Class addbook
  Inherits System.Web.UI.Page
  Protected WithEvents Label1 As System.Web.UI.WebControls.Label
  Protected WithEvents Label6 As System.Web.UI.WebControls.Label
  Protected WithEvents Label4 As System.Web.UI.WebControls.Label
  Protected WithEvents Label3 As System.Web.UI.WebControls.Label
  Protected WithEvents Label2 As System.Web.UI.WebControls.Label
  Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox4 As System.Web.UI.WebControls.TextBox
  Protected WithEvents TextBox5 As System.Web.UI.WebControls.TextBox
  Protected WithEvents Label5 As System.Web.UI.WebControls.Label
  Protected WithEvents Button1 As System.Web.UI.WebControls.Button


#Region " Web Form Designer Generated Code "


  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()


  End Sub
```

```
  Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

#End Region

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Value, Bookyear As Integer
    Dim serv = New B2BSite.localhost.Service1()
    Dim res As Boolean
    Dim Log, Pass As String
    Value = Convert.ToInt32(TextBox4.Text)
    Bookyear = Convert.ToInt32(TextBox5.Text)
    Log = Session.Item("log")
    Pass = Session.Item("pass")
    res = serv.AddBook(Log, Pass, TextBox1.Text, TextBox2.Text,
TextBox3.Text, Value, Bookyear)
    If res Then
      Label5.Text = "Информация о книге добавлена"
    Else
      Label5.Text = "Внести информацию в базу данных не удалось"
    End If
    Label5.Visible = True
  End Sub
End Class
```

Пожалуй, уже нет нужды досконально разбирать код этого класса, так как он достаточно похож на те функциональные модули, которые мы создавали ранее. Как и прежде, объявляется переменная, являющаяся экземпляром прокси-класса Web-сервиса, целочисленные значения, введенные пользователем в текстовые поля, просто конвертируются, и вызывается функция Web-сервиса, в которую передаются параметры, как введенные пользовате-

лем, так и извлеченные из стандартной коллекции объекта `Session`. А затем на основе результата вызванной функции формируется и отображается на Web-странице текстовое сообщение о выполнении операции.

Итак, полностью создан сайт, поддерживающий работу с Web-сервисом, который, в свою очередь, реализует решение класса B2B. В следующем разделе мы рассмотрим пример разработки специализированного клиентского приложения, взаимодействующего с Web-сервисом.

Создание клиентского приложения

Процедура разработки взаимодействующего с Web-сервисом специализированного клиентского приложения начинается с создания ряда форм. Естественно, на одной форме уместить весь набор органов управления для работы со всеми функциями Web-сервиса не получится. Поэтому на основной форме пользователь будет выполнять запросы к общей базе данных, а для доступа к остальным функциям, таким как изменение учетной записи, добавление информации о новых книгах или новых коммерческих предложениях, будут использоваться дополнительные формы, активизируемые по мере надобности. Внешний вид приложения при первом его запуске показан на рис. 8.6.

Рис. 8.6. Внешний вид основного рабочего окна при запуске клиентского приложения

В левой части окна располагаются поля для указания параметров поиска и кнопка, запускающая поиск. Практически все оставшееся пространство будет отдано под таблицу, реализованную в виде компонента `DataGrid`, которая будет отображать результаты поиска. Естественно, перед тем как ото-

бражать его результаты, этот поиск необходимо еще произвести. Поиск будет осуществляться при помощи соответствующей функции Web-сервиса, доступ к которой осуществляется из другой функции, обрабатывающей нажатие кнопки на форме. Код этой функции приведен в листинге 8.18.

Листинг 8.18

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Price1, Price2 As Integer
    Dim serv = New B2BClient.localhost.Service1()
    If TextBox4.Text = "" Then
      Price1 = 0
    Else : Price1 = Convert.ToInt32(TextBox4.Text)
    End If
    If TextBox5.Text = "" Then
      Price2 = 10000
    Else : Price2 = Convert.ToInt32(TextBox5.Text)
    End If
    DataSet1.Clear()
    DataSet1.Merge(serv.FindBooks(TextBox1.Text, TextBox2.Text,
TextBox6.Text, TextBox3.Text, Price1, Price2))
    DataGrid1.Refresh()
End Sub
```

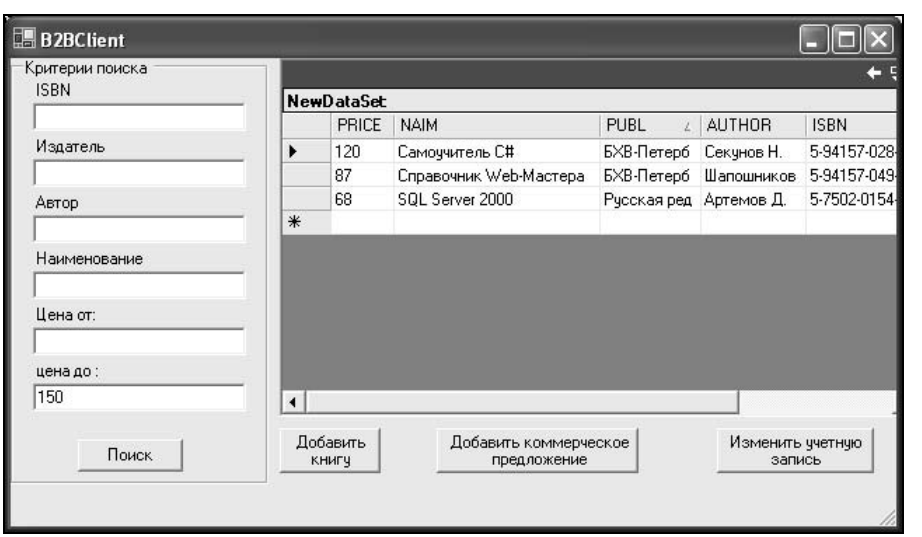

Рис. 8.7. Внешний вид основного окна клиентского приложения с результатами поиска

Легко заметить, что функция поиска практически идентична той, что использовалась при разработке сайта, взаимодействующего с сервисом. Единственное значимое отличие состоит в способе активации таблицы. Так как таблица заранее связана с набором данных, служащим для нее источником, не потребуется выполнять процедуру связывания, которая ранее вызывалась методом `DataBind`. Более того, для объекта `DataGrid`, который используется в обычных приложениях, а не на Web-страницах ASP.NET, подобный метод вообще не предусмотрен. Поэтому после того, как мы заполним набор данных, следует просто обновить таблицу при помощи метода `Refresh`, как это показано в листинге. При этом основное окно приложения приобретет вид, показанный на рис. 8.7.

Теперь рассмотрим пример разработки формы, в которой пользователь сможет добавить информацию о книге в общую базу данных. Для этого в основном окне предусмотрена соответствующая кнопка, обработчик которой активирует дополнительную форму. Ее код приведен в листинге 8.19.

Листинг 8.19

```
Public Class Form2
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub
  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub
```

```
 'Required by the Windows Form Designer
  Private components As System.ComponentModel.IContainer


  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents Label3 As System.Windows.Forms.Label
  Friend WithEvents Label4 As System.Windows.Forms.Label
  Friend WithEvents Label5 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents StatusBar1 As System.Windows.Forms.StatusBar
  Friend WithEvents Label6 As System.Windows.Forms.Label
  Friend WithEvents Label7 As System.Windows.Forms.Label
  Friend WithEvents TextBox6 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox7 As System.Windows.Forms.TextBox
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.Label3 = New System.Windows.Forms.Label()
    Me.Label4 = New System.Windows.Forms.Label()
    Me.Label5 = New System.Windows.Forms.Label()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    Me.TextBox3 = New System.Windows.Forms.TextBox()
    Me.TextBox4 = New System.Windows.Forms.TextBox()
    Me.TextBox5 = New System.Windows.Forms.TextBox()
    Me.Button2 = New System.Windows.Forms.Button()
    Me.StatusBar1 = New System.Windows.Forms.StatusBar()
```

```
Me.Label6 = New System.Windows.Forms.Label()
Me.Label7 = New System.Windows.Forms.Label()
Me.TextBox6 = New System.Windows.Forms.TextBox()
Me.TextBox7 = New System.Windows.Forms.TextBox()
Me.SuspendLayout()
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(264, 312)
Me.Button1.Name = "Button1"
Me.Button1.TabIndex = 0
Me.Button1.Text = "Закрыть"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(24, 104)
Me.Label1.Name = "Label1"
Me.Label1.TabIndex = 1
Me.Label1.Text = "ISBN"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(24, 148)
Me.Label2.Name = "Label2"
Me.Label2.TabIndex = 2
Me.Label2.Text = "Автор"
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(24, 192)
Me.Label3.Name = "Label3"
Me.Label3.TabIndex = 3
Me.Label3.Text = "Наименование"
'
'Label4
'
Me.Label4.Location = New System.Drawing.Point(24, 236)
Me.Label4.Name = "Label4"
```

```
Me.Label4.TabIndex = 4
Me.Label4.Text = "Кол-во страниц"
'
'Label5
'
Me.Label5.Location = New System.Drawing.Point(24, 280)
Me.Label5.Name = "Label5"
Me.Label5.TabIndex = 5
Me.Label5.Text = "Год выпуска"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(144, 104)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.Size = New System.Drawing.Size(160, 20)
Me.TextBox1.TabIndex = 6
Me.TextBox1.Text = ""
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(144, 148)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.Size = New System.Drawing.Size(160, 20)
Me.TextBox2.TabIndex = 7
Me.TextBox2.Text = ""
'
'TextBox3
'
Me.TextBox3.Location = New System.Drawing.Point(144, 192)
Me.TextBox3.Name = "TextBox3"
Me.TextBox3.Size = New System.Drawing.Size(160, 20)
Me.TextBox3.TabIndex = 8
Me.TextBox3.Text = ""
'
'TextBox4
'
Me.TextBox4.Location = New System.Drawing.Point(144, 236)
Me.TextBox4.Name = "TextBox4"
```

```
Me.TextBox4.Size = New System.Drawing.Size(160, 20)
Me.TextBox4.TabIndex = 9
Me.TextBox4.Text = ""
'
'TextBox5
'
Me.TextBox5.Location = New System.Drawing.Point(144, 280)
Me.TextBox5.Name = "TextBox5"
Me.TextBox5.Size = New System.Drawing.Size(160, 20)
Me.TextBox5.TabIndex = 10
Me.TextBox5.Text = ""
'
'Button2
'
Me.Button2.Location = New System.Drawing.Point(40, 312)
Me.Button2.Name = "Button2"
Me.Button2.Size = New System.Drawing.Size(112, 23)
Me.Button2.TabIndex = 11
Me.Button2.Text = "Подтвердить"
'
'StatusBar1
'
Me.StatusBar1.Location = New System.Drawing.Point(0, 344)
Me.StatusBar1.Name = "StatusBar1"
Me.StatusBar1.Size = New System.Drawing.Size(352, 22)
Me.StatusBar1.TabIndex = 12
Me.StatusBar1.Text = "Ready"
'
'Label6
'
Me.Label6.Location = New System.Drawing.Point(24, 16)
Me.Label6.Name = "Label6"
Me.Label6.TabIndex = 13
Me.Label6.Text = "Логин"
'
'Label7
'
Me.Label7.Location = New System.Drawing.Point(24, 60)
```

```
    Me.Label7.Name = "Label7"
    Me.Label7.TabIndex = 14
    Me.Label7.Text = "Пароль"
    '
    'TextBox6
    '
    Me.TextBox6.Location = New System.Drawing.Point(144, 16)
    Me.TextBox6.Name = "TextBox6"
    Me.TextBox6.Size = New System.Drawing.Size(160, 20)
    Me.TextBox6.TabIndex = 15
    Me.TextBox6.Text = ""
    '
    'TextBox7
    '
    Me.TextBox7.Location = New System.Drawing.Point(144, 60)
    Me.TextBox7.Name = "TextBox7"
    Me.TextBox7.Size = New System.Drawing.Size(160, 20)
    Me.TextBox7.TabIndex = 16
    Me.TextBox7.Text = ""
    '
    'Form2
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(352, 366)
    Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.TextBox7,
Me.TextBox6, Me.Label7, Me.Label6, Me.StatusBar1, Me.Button2,
Me.TextBox5, Me.TextBox4, Me.TextBox3, Me.TextBox2, Me.TextBox1,
Me.Label5, Me.Label4, Me.Label3, Me.Label2, Me.Label1, Me.Button1})
    Me.Name = "Form2"
    Me.Text = "Добавление книги"
    Me.ResumeLayout(False)

  End Sub

#End Region

  Protected Overrides Sub Finalize()
    MyBase.Finalize()
  End Sub
```

```
  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Close()
  End Sub

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim Value, Bookyear As Integer
    Dim serv = New B2BClient.localhost.Service1()
    Dim res As Boolean
    Value = Convert.ToInt32(TextBox4.Text)
    Bookyear = Convert.ToInt32(TextBox5.Text)
    res = serv.AddBook(TextBox6.Text, TextBox7.Text, TextBox3.Text,
TextBox1.Text, TextBox2.Text, Value, Bookyear)
    If res Then
      StatusBar1.Text = "Информация добавлена"
    Else
      StatusBar1.Text = "Информацию добавить не удалось"
    End If
  End Sub
End Class
```

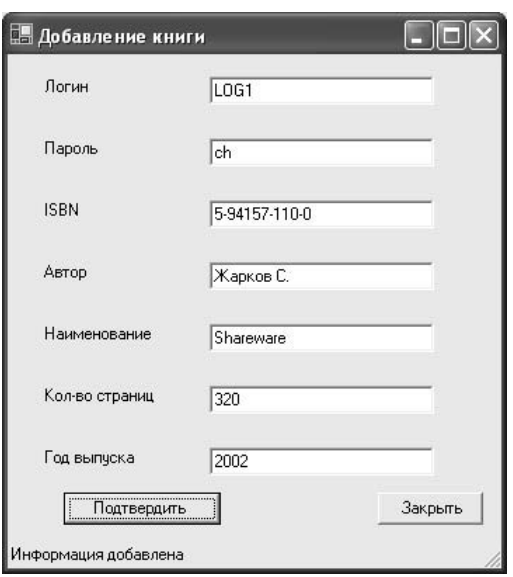

Рис. 8.8. Внешний вид окна, в котором пользователь добавляет информацию о книге в базу данных

Внешний вид этого окна после добавления пользователем книги показан на рис. 8.8.

Как видно из иллюстрации, на форме расположены поля ввода для добавляемой информации и две кнопки. Кнопка **Подтвердить** запускает процесс добавления информации о книге, а кнопка **Закрыть**, соответственно, закрывает окно и возвращает пользователя к основному рабочему окну клиентского приложения. Следует отметить, что исходя из результата, возвращаемого функцией Web-сервиса `AddBook`, формируется текстовое сообщение, отображающееся в строке статуса. Закрытие окна, как можно увидеть из листинга, производится при помощи метода `Close`.

Теперь займемся разработкой формы, в которой пользователь сможет добавить коммерческое предложение в общую базу данных. Для этого нам придется добавить еще одну форму в состав разрабатываемого проекта. Сама технология разработки будет весьма схожа с только что рассмотренным примером. Код этой формы приведен в листинге 8.20.

Листинг 8.20

```
Public Class Form3
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
```

```
    MyBase.Dispose(disposing)
  End Sub

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents Label3 As System.Windows.Forms.Label
  Friend WithEvents Label4 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents StatusBar1 As System.Windows.Forms.StatusBar
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.Label3 = New System.Windows.Forms.Label()
    Me.Label4 = New System.Windows.Forms.Label()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    Me.TextBox3 = New System.Windows.Forms.TextBox()
    Me.TextBox4 = New System.Windows.Forms.TextBox()
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Button2 = New System.Windows.Forms.Button()
    Me.StatusBar1 = New System.Windows.Forms.StatusBar()
    Me.SuspendLayout()
    '
    'Label1
    '
    Me.Label1.Location = New System.Drawing.Point(16, 24)
    Me.Label1.Name = "Label1"
```

```
Me.Label1.TabIndex = 0
Me.Label1.Text = "Логин"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(16, 66)
Me.Label2.Name = "Label2"
Me.Label2.TabIndex = 1
Me.Label2.Text = "Пароль"
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(16, 108)
Me.Label3.Name = "Label3"
Me.Label3.TabIndex = 2
Me.Label3.Text = "ISBN"
'
'Label4
'
Me.Label4.Location = New System.Drawing.Point(16, 150)
Me.Label4.Name = "Label4"
Me.Label4.TabIndex = 3
Me.Label4.Text = "Цена"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(168, 24)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.TabIndex = 4
Me.TextBox1.Text = ""
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(168, 66)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.TabIndex = 5
Me.TextBox2.Text = ""
'
```

```
'TextBox3
'
Me.TextBox3.Location = New System.Drawing.Point(168, 108)
Me.TextBox3.Name = "TextBox3"
Me.TextBox3.TabIndex = 6
Me.TextBox3.Text = ""
'
'TextBox4
'
Me.TextBox4.Location = New System.Drawing.Point(168, 150)
Me.TextBox4.Name = "TextBox4"
Me.TextBox4.TabIndex = 7
Me.TextBox4.Text = ""
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(40, 200)
Me.Button1.Name = "Button1"
Me.Button1.TabIndex = 8
Me.Button1.Text = "Запомнить"
'
'Button2
'
Me.Button2.Location = New System.Drawing.Point(200, 200)
Me.Button2.Name = "Button2"
Me.Button2.TabIndex = 9
Me.Button2.Text = "Закрыть"
'
'StatusBar1
'
Me.StatusBar1.Location = New System.Drawing.Point(0, 240)
Me.StatusBar1.Name = "StatusBar1"
Me.StatusBar1.Size = New System.Drawing.Size(336, 22)
Me.StatusBar1.TabIndex = 10
Me.StatusBar1.Text = "Ready"
'
'Form3
'
```

```
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(336, 262)
    Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.StatusBar1, Me.Button2, Me.Button1, Me.TextBox4, Me.TextBox3,
Me.TextBox2, Me.TextBox1, Me.Label4, Me.Label3, Me.Label2, Me.Label1})
    Me.Name = "Form3"
    Me.Text = "Добавить цену"
    Me.ResumeLayout(False)

  End Sub

#End Region

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Close()
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Price As Integer
    Dim serv = New B2BClient.localhost.Service1()
    Dim res As Boolean
    Price = Convert.ToInt32(TextBox4.Text)
    res = serv.AddPrice(TextBox1.Text, TextBox2.Text, TextBox3.Text,
Price)
    If res Then
      StatusBar1.Text = "Информация добавлена"
    Else
      StatusBar1.Text = "Информацию добавить не удалось"
    End If
  End Sub
End Class
```

При работе с этой частью приложения становится понятно, что ISBN-код книги удобнее выбирать из выпадающего списка, а не вносить в поле текстового ввода вручную. Теоретически это можно сделать, если установить соединение с SQL-сервером, который поддерживает функционирование общей базы данных, и при помощи простого SQL-запроса получить список всех ISBN-кодов. Однако необходимо понимать, что всю работу с SQL-сервером ведет именно Web-сервис, а клиентское приложение может лишь пользоваться теми функциями, которые предоставляются этим Web-

сервисом. Разработчик клиентского приложения может вообще не знать структуры базы данных, с которой он работает. Так как у сервиса нет функции, которая в качестве результата выдавала бы список ISBN-кодов всех книг из базы данных, придется вводить этот код вручную, как это показано на рис. 8.9.

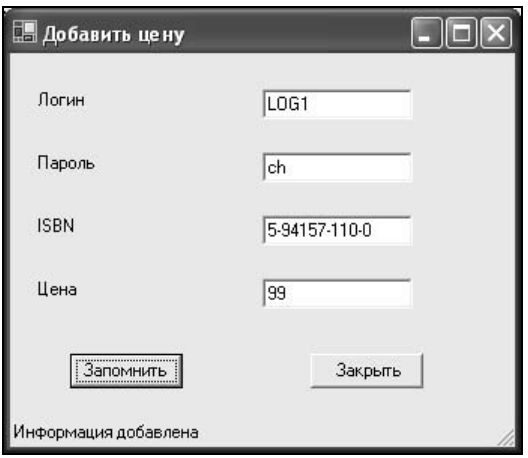

Рис. 8.9. Окно ввода коммерческого предложения

Нам осталось разработать лишь один дополнительный модуль, при помощи которого пользователь сможет изменить свою учетную запись, и после этого привести итоговый код основного окна клиентского приложения. Итак, добавляем к нашему проекту еще одну форму и смотрим на ее код (листинг 8.21).

Листинг 8.21

```
Public Class Form4
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call
```

```
  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents Label3 As System.Windows.Forms.Label
  Friend WithEvents Label4 As System.Windows.Forms.Label
  Friend WithEvents Label5 As System.Windows.Forms.Label
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents StatusBar1 As System.Windows.Forms.StatusBar
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.Label3 = New System.Windows.Forms.Label()
    Me.Label4 = New System.Windows.Forms.Label()
    Me.Label5 = New System.Windows.Forms.Label()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
```

```
Me.TextBox2 = New System.Windows.Forms.TextBox()
Me.TextBox3 = New System.Windows.Forms.TextBox()
Me.TextBox4 = New System.Windows.Forms.TextBox()
Me.TextBox5 = New System.Windows.Forms.TextBox()
Me.Button1 = New System.Windows.Forms.Button()
Me.Button2 = New System.Windows.Forms.Button()
Me.StatusBar1 = New System.Windows.Forms.StatusBar()
Me.SuspendLayout()
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(16, 16)
Me.Label1.Name = "Label1"
Me.Label1.TabIndex = 0
Me.Label1.Text = "Логин"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(16, 56)
Me.Label2.Name = "Label2"
Me.Label2.TabIndex = 1
Me.Label2.Text = "Пароль"
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(16, 96)
Me.Label3.Name = "Label3"
Me.Label3.TabIndex = 2
Me.Label3.Text = "Наименование"
'
'Label4
'
Me.Label4.Location = New System.Drawing.Point(16, 136)
Me.Label4.Name = "Label4"
Me.Label4.TabIndex = 3
Me.Label4.Text = "E-mail"
'
'Label5
```

```
'
Me.Label5.Location = New System.Drawing.Point(16, 176)
Me.Label5.Name = "Label5"
Me.Label5.TabIndex = 4
Me.Label5.Text = "Телефон"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(144, 16)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.TabIndex = 5
Me.TextBox1.Text = ""
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(144, 56)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.TabIndex = 6
Me.TextBox2.Text = ""
'
'TextBox3
'
Me.TextBox3.Location = New System.Drawing.Point(144, 96)
Me.TextBox3.Name = "TextBox3"
Me.TextBox3.TabIndex = 7
Me.TextBox3.Text = ""
'
'TextBox4
'
Me.TextBox4.Location = New System.Drawing.Point(144, 136)
Me.TextBox4.Name = "TextBox4"
Me.TextBox4.TabIndex = 8
Me.TextBox4.Text = ""
'
'TextBox5
'
Me.TextBox5.Location = New System.Drawing.Point(144, 176)
Me.TextBox5.Name = "TextBox5"
```

```
    Me.TextBox5.TabIndex = 9
    Me.TextBox5.Text = ""
    '
    'Button1
    '
    Me.Button1.Location = New System.Drawing.Point(16, 224)
    Me.Button1.Name = "Button1"
    Me.Button1.Size = New System.Drawing.Size(104, 23)
    Me.Button1.TabIndex = 10
    Me.Button1.Text = "Подтвердить"
    '
    'Button2
    '
    Me.Button2.Location = New System.Drawing.Point(184, 224)
    Me.Button2.Name = "Button2"
    Me.Button2.TabIndex = 11
    Me.Button2.Text = "Закрыть"
    '
    'StatusBar1
    '
    Me.StatusBar1.Location = New System.Drawing.Point(0, 256)
    Me.StatusBar1.Name = "StatusBar1"
    Me.StatusBar1.Size = New System.Drawing.Size(320, 22)
    Me.StatusBar1.TabIndex = 12
    Me.StatusBar1.Text = "Ready"
    '
    'Form4
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(320, 278)
    Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.StatusBar1, Me.Button2, Me.Button1, Me.TextBox5, Me.TextBox4,
Me.TextBox3, Me.TextBox2, Me.TextBox1, Me.Label5, Me.Label4, Me.Label3,
Me.Label2, Me.Label1})
    Me.Name = "Form4"
    Me.Text = "Изменение учетной записи"
    Me.ResumeLayout(False)


  End Sub
```

```
#End Region

  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Close()
  End Sub

  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim serv = New B2BClient.localhost.Service1()
    Dim res As Boolean
    res = serv.EditUser(TextBox1.Text, TextBox2.Text, TextBox3.Text,
TextBox4.Text, TextBox5.Text)
    If res Then
      StatusBar1.Text = "Информация добавлена"
    Else
      StatusBar1.Text = "Информацию добавить не удалось"
    End If
  End Sub
End Class
```

Внешний вид разработанного окна показан на рис. 8.10.

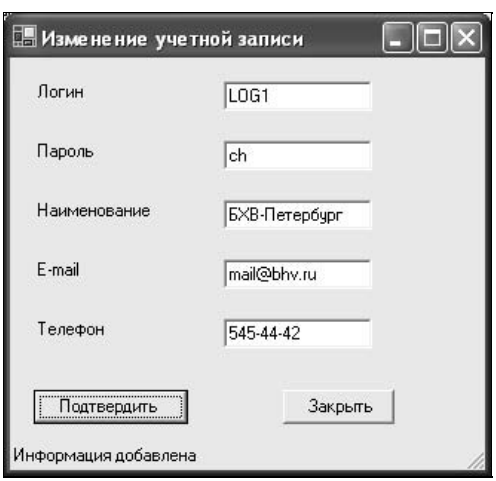

Рис. 8.10. Окно изменения учетной записи пользователя

Теперь, когда у нас есть все дополнительные модули, можно привести полный код основного рабочего модуля клиентского приложения (листинг 8.22).

Листинг 8.22

```
Public Class Form1
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

  Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

  End Sub

  'Form overrides dispose to clean up the component list.
  Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
      If Not (components Is Nothing) Then
        components.Dispose()
      End If
    End If
    MyBase.Dispose(disposing)
  End Sub

  'Required by the Windows Form Designer
  Private components As System.ComponentModel.IContainer

  'NOTE: The following procedure is required by the Windows Form Designer
  'It can be modified using the Windows Form Designer.
  'Do not modify it using the code editor.
  Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
  Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox4 As System.Windows.Forms.TextBox
```

```
  Friend WithEvents TextBox5 As System.Windows.Forms.TextBox
  Friend WithEvents TextBox6 As System.Windows.Forms.TextBox
  Friend WithEvents Label1 As System.Windows.Forms.Label
  Friend WithEvents Label2 As System.Windows.Forms.Label
  Friend WithEvents Label3 As System.Windows.Forms.Label
  Friend WithEvents Label4 As System.Windows.Forms.Label
  Friend WithEvents Label5 As System.Windows.Forms.Label
  Friend WithEvents Label6 As System.Windows.Forms.Label
  Friend WithEvents Button1 As System.Windows.Forms.Button
  Friend WithEvents DataSet1 As System.Data.DataSet
  Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
  Friend WithEvents Button2 As System.Windows.Forms.Button
  Friend WithEvents Button3 As System.Windows.Forms.Button
  Friend WithEvents Button4 As System.Windows.Forms.Button
  Friend WithEvents StatusBar1 As System.Windows.Forms.StatusBar
  <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.GroupBox1 = New System.Windows.Forms.GroupBox()
    Me.Button1 = New System.Windows.Forms.Button()
    Me.Label6 = New System.Windows.Forms.Label()
    Me.Label5 = New System.Windows.Forms.Label()
    Me.Label4 = New System.Windows.Forms.Label()
    Me.Label3 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.TextBox6 = New System.Windows.Forms.TextBox()
    Me.TextBox5 = New System.Windows.Forms.TextBox()
    Me.TextBox4 = New System.Windows.Forms.TextBox()
    Me.TextBox3 = New System.Windows.Forms.TextBox()
    Me.TextBox2 = New System.Windows.Forms.TextBox()
    Me.TextBox1 = New System.Windows.Forms.TextBox()
    Me.DataSet1 = New System.Data.DataSet()
    Me.DataGrid1 = New System.Windows.Forms.DataGrid()
    Me.Button2 = New System.Windows.Forms.Button()
    Me.Button3 = New System.Windows.Forms.Button()
    Me.Button4 = New System.Windows.Forms.Button()
    Me.StatusBar1 = New System.Windows.Forms.StatusBar()
    Me.GroupBox1.SuspendLayout()
```

```
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    'GroupBox1
    '
    Me.GroupBox1.Controls.AddRange(New System.Windows.Forms.Control()
{Me.Button1, Me.Label6, Me.Label5, Me.Label4, Me.Label3, Me.Label2,
Me.Label1, Me.TextBox6, Me.TextBox5, Me.TextBox4, Me.TextBox3,
Me.TextBox2, Me.TextBox1})
    Me.GroupBox1.Name = "GroupBox1"
    Me.GroupBox1.Size = New System.Drawing.Size(184, 304)
    Me.GroupBox1.TabIndex = 0
    Me.GroupBox1.TabStop = False
    Me.GroupBox1.Text = "Критерии поиска"
    '
    'Button1
    '
    Me.Button1.Location = New System.Drawing.Point(48, 272)
    Me.Button1.Name = "Button1"
    Me.Button1.TabIndex = 12
    Me.Button1.Text = "Поиск"
    '
    'Label6
    '
    Me.Label6.Location = New System.Drawing.Point(16, 216)
    Me.Label6.Name = "Label6"
    Me.Label6.Size = New System.Drawing.Size(88, 16)
    Me.Label6.TabIndex = 11
    Me.Label6.Text = "цена до :"
    '
    'Label5
    '
    Me.Label5.Location = New System.Drawing.Point(16, 176)
    Me.Label5.Name = "Label5"
    Me.Label5.Size = New System.Drawing.Size(88, 16)
    Me.Label5.TabIndex = 10
    Me.Label5.Text = "Цена от:"
```

```
'
'Label4
'
Me.Label4.Location = New System.Drawing.Point(16, 136)
Me.Label4.Name = "Label4"
Me.Label4.Size = New System.Drawing.Size(88, 16)
Me.Label4.TabIndex = 9
Me.Label4.Text = "Наименование"
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(16, 96)
Me.Label3.Name = "Label3"
Me.Label3.Size = New System.Drawing.Size(88, 16)
Me.Label3.TabIndex = 8
Me.Label3.Text = "Автор"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(16, 56)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(88, 16)
Me.Label2.TabIndex = 7
Me.Label2.Text = "Издатель"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(16, 16)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(96, 16)
Me.Label1.TabIndex = 6
Me.Label1.Text = "ISBN"
'
'TextBox6
'
Me.TextBox6.Location = New System.Drawing.Point(16, 152)
Me.TextBox6.Name = "TextBox6"
Me.TextBox6.Size = New System.Drawing.Size(152, 20)
```

```
Me.TextBox6.TabIndex = 5
Me.TextBox6.Text = ""
'
'TextBox5
'
Me.TextBox5.Location = New System.Drawing.Point(16, 232)
Me.TextBox5.Name = "TextBox5"
Me.TextBox5.Size = New System.Drawing.Size(152, 20)
Me.TextBox5.TabIndex = 4
Me.TextBox5.Text = ""
'
'TextBox4
'
Me.TextBox4.Location = New System.Drawing.Point(16, 192)
Me.TextBox4.Name = "TextBox4"
Me.TextBox4.Size = New System.Drawing.Size(152, 20)
Me.TextBox4.TabIndex = 3
Me.TextBox4.Text = ""
'
'TextBox3
'
Me.TextBox3.Location = New System.Drawing.Point(16, 112)
Me.TextBox3.Name = "TextBox3"
Me.TextBox3.Size = New System.Drawing.Size(152, 20)
Me.TextBox3.TabIndex = 2
Me.TextBox3.Text = ""
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(16, 72)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.Size = New System.Drawing.Size(152, 20)
Me.TextBox2.TabIndex = 1
Me.TextBox2.Text = ""
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(16, 32)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.Size = New System.Drawing.Size(152, 20)
```

```
    Me.TextBox1.TabIndex = 0
    Me.TextBox1.Text = ""
    '
    'DataSet1
    '
    Me.DataSet1.DataSetName = "NewDataSet"
    Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU")
    '
    'DataGrid1
    '
    Me.DataGrid1.DataMember = ""
    Me.DataGrid1.DataSource = Me.DataSet1
    Me.DataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText
    Me.DataGrid1.Location = New System.Drawing.Point(192, 0)
    Me.DataGrid1.Name = "DataGrid1"
    Me.DataGrid1.Size = New System.Drawing.Size(456, 256)
    Me.DataGrid1.TabIndex = 1
    '
    'Button2
    '
    Me.Button2.Location = New System.Drawing.Point(192, 264)
    Me.Button2.Name = "Button2"
    Me.Button2.Size = New System.Drawing.Size(72, 32)
    Me.Button2.TabIndex = 2
    Me.Button2.Text = "Добавить книгу"
    '
    'Button3
    '
    Me.Button3.Location = New System.Drawing.Point(304, 264)
    Me.Button3.Name = "Button3"
    Me.Button3.Size = New System.Drawing.Size(144, 32)
    Me.Button3.TabIndex = 3
    Me.Button3.Text = "Добавить коммерческое предложение"
    '
    'Button4
    '
    Me.Button4.Location = New System.Drawing.Point(504, 264)
    Me.Button4.Name = "Button4"
    Me.Button4.Size = New System.Drawing.Size(112, 32)
```

```
    Me.Button4.TabIndex = 4
    Me.Button4.Text = "Изменить учетную запись"
    '
    'StatusBar1
    '
    Me.StatusBar1.Location = New System.Drawing.Point(0, 312)
    Me.StatusBar1.Name = "StatusBar1"
    Me.StatusBar1.Size = New System.Drawing.Size(632, 22)
    Me.StatusBar1.TabIndex = 5
    '
    'Form1
    '
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(632, 334)
    Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.StatusBar1, Me.Button4, Me.Button3, Me.Button2, Me.DataGrid1,
Me.GroupBox1})
    Me.Name = "Form1"
    Me.Text = "B2BClient"
    Me.GroupBox1.ResumeLayout(False)
    CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()
    CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
    Me.ResumeLayout(False)


  End Sub


#End Region


  Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Price1, Price2 As Integer
    Dim serv = New B2BClient.localhost.Service1()
    If TextBox4.Text = "" Then
      Price1 = 0
    Else
      Price1 = Convert.ToInt32(TextBox4.Text)
    End If
    If TextBox5.Text = "" Then
      Price2 = 10000
```

```
    Else
      Price2 = Convert.ToInt32(TextBox5.Text)
    End If
    DataSet1.Clear()
    DataSet1.Merge(serv.FindBooks(TextBox1.Text, TextBox2.Text,
TextBox6.Text, TextBox3.Text, Price1, Price2))
    DataGrid1.Refresh()
  End Sub


  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Dim t As New Form2()
    t.Show()
  End Sub



  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    Dim t As New Form3()
    t.Show()
  End Sub


  Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
    Dim t As New Form4()
    t.Show()
  End Sub
End Class
```

В этом коде мы увидим не много новшеств. Помимо визуальной структуры, которая описывается в первой половине листинга, следует обратить внимание на то, как активизируются дополнительные окна. Для этого в функции, обрабатывающей нажатие соответствующей кнопки, объявляется переменная, представляющая собой экземпляр класса, задающего ту или иную форму. Соответственно, после этого остается всего лишь вызвать для этой переменной метод `Show`, который отображает окно и передает ему фокус ввода.

В этом разделе главы мы рассмотрели пример создания приложения, которое выступает в роли клиента по отношению к Web-сервису. Следует отметить, что разработка приложения была не в пример проще, чем разработка самого сервиса. Действительно, именно в Web-сервисе сосредоточена вся функциональность, а клиентское приложение в данном случае служит всего лишь интерфейсом для него, поэтому затраты на разработку клиента были достаточно малы.

Глава 9

Поддержка пиринговых сетей

Структура пиринговых сетей

Пиринговые сети — одна из самых многообещающих и популярных форм обмена информацией между пользователями Сети. Их смысл сводится к тому, что при помощи некоего координирующего центра пользователь указывает список ресурсов, которые он может предоставить иным пользователям, и выбирает из общего списка те ресурсы, которые необходимы именно этому пользователю. А затем клиентские приложения осуществляют обмен информацией напрямую, без участия центрального сервера, не нагружая его. То есть роль центрального сервера сводится только к координированию клиентских приложений, и траффик у этого сервера будет минимален.

Как известно, в настоящее время пиринговые сети становятся все более популярными, предоставляя слабо централизованный сервис для свободного обмена информацией. Одними из наиболее известных сервисов подобного рода являются скандально известные Napster и Kazaa. И если Napster был более ориентирован на обмен MP3-файлами, за что, собственно, и был задушен ассоциацией РИАА, представляющей интересы официальных издателей музыки, то сервис Kazaa позволяет пользователям обмениваться файлами любых типов. При этом весь основной трафик ложится именно на самих пользователей, так как файлы пересылаются между ними напрямую, без участия координирующего центра.

Основа популярности подобного сервиса — распространенность клиентских приложений. Чем больше пользователей в файлообменной сети, тем выше ее ценность, так как соответственно количеству пользователей увеличивается количество файлов, доступных каждому члену сети.

Естественно, Web-сервисы вполне удовлетворяют условиям, предъявляемым координационному центру файлообменной сети. Для работы с Web-сервисом можно разрабатывать клиентские приложения, функционирующие в любой операционной системе. Таким образом разработчик может расши-

рить границы пользователей файлообменной сети. В этой главе мы рассмотрим пример разработки подобного Web-сервиса и клиентского приложения с несколько урезанной функциональностью. Начнем мы с разработки основы нашей файлообменной сети — Web-сервиса.

Обслуживающий Web-сервис

Итак, какова будет структура Web-сервиса, координирующего работу файлообменной сети? Для того чтобы уяснить структуру сервиса, следует точно обозначить механизм работы сети, разграничить сферы ответственности Web-сервиса и клиентских приложений.

Прежде всего, нам потребуется однозначно идентифицировать каждого пользователя и опознавать его при каждом его входе в сеть. Для этого следует предусмотреть функцию регистрации пользователя с проверкой уникальности логина. При каждом входе в файлообменную сеть, то есть при подключении к Web-сервису, помимо аутентификации, пользователь должен передать список файлов, которые он готов предоставить для скачивания остальным пользователям сети. В свою очередь Web-сервис должен предоставить пользователю список файлов, удовлетворяющий его критерию поиска. Поиск будем осуществлять по подстроке, входящей в состав имени файла. В качестве результата выполнения этой функции пользователю будет возвращаться список файлов, их размеры и даты модификации, а также IP-адрес пользователя, у которого эти файлы находятся. Клиентское приложение должно само установить контакт с искомым пользователем сети и загрузить требуемые файлы.

Для хранения списка пользователей файлообменной сети и поддержания актуального списка всех доступных на данный момент файлов стоит использовать SQL-сервер, тем более, что он позволит использовать SQL-запрос для поиска по наименованиям файлов. Для работы Web-сервиса потребуется две таблицы — одна для хранения списка пользователей и вторая для хранения актуального списка файлов, которые пользователи, подключенные в данный момент к сервису, предоставляют иным пользователям для скачивания.

Для хранения информации о пользователях мы создадим таблицу `USERS`, содержащую всего два поля, в которых будут храниться логины и соответствующие им пароли. Больше нам о пользователях сети не потребуется знать ничего. SQL-скрипт, который формирует эту таблицу, выглядит следующим образом:

```
CREATE TABLE [dbo].[USERS] (
    [LOGIN] [char] (30) COLLATE Cyrillic_General_CI_AS NULL,
    [PASS] [varchar] (30) COLLATE Cyrillic_General_CI_AS NULL
) ON [PRIMARY]
```

Помимо этой таблицы, нам потребуется таблица FILES, в которой будут храниться наименования файлов, их размер и дата последней модификации, а также логин пользователя, который предоставляет их для скачивания. Естественно, потребуется еще одно идентифицирующее поле. Соответствующий SQL-скрипт выглядит так:

```
CREATE TABLE [dbo].[FILES] (
    [IDFILE] [int] IDENTITY (1, 1) NOT NULL,
    [NAIM] [char] (50) COLLATE Cyrillic_General_CI_AS NULL,
    [FILESIZE] [int] NULL,
    [LAST_MOD] [datetime] NULL,
    [LOGIN] [char] (30) COLLATE Cyrillic_General_CI_AS NULL
) ON [PRIMARY]
```

Теперь, когда мы знаем состав проектируемого Web-сервиса, можно приступить к его разработке. Начнем с рассмотрения функции, проверяющей наличие в базе данных логина. Пример подобной функции рассматривался в предыдущей главе, поэтому сейчас ее создание не будет слишком сложной задачей. Для проверки существования логина используем следующий SQL-скрипт:

```
SELECT     COUNT(LOGIN) AS CID
FROM       USERS
WHERE     (LOGIN = @LOG)
```

Искомая функция будет возвращать, естественно, логическое значение, а в качестве параметра принимать текстовую переменную, содержимое которой мы и будем проверять на наличие в базе данных. Код этой функции приведен в листинге 9.1.

Листинг 9.1

```
<WebMethod()> Public Function LogExist(ByVal LOG As String) As Boolean
        Dim DS As New DataSet()
        Dim t As Boolean
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select count(LOGIN) as CID from
USERS where (LOGIN = @LOG)"
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 30))
        DataCommand.SelectCommand.Parameters("@LOG").Value = LOG
        DataCommand.Fill(DS)
```

```
        If DS.Tables(0).Rows(0).Item("CID") = 0 Then
            t = False
        Else
            t = True
        End If
        DS.Dispose()
        LogExist = t
    End Function
```

Эта функция использует SQL-запрос типа `Select`, который получает количество логинов в базе данных, совпадающих с переданным значением. Естественно, мы считаем эти логины уникальными, поэтому возвращаемое значение будет либо нулем, либо единицей. Поэтому и анализируем мы его при помощи следующей конструкции:

```
If DS.Tables(0).Rows(0).Item("CID") = 0 Then
```

После того как мы задали значение для переменной `t`, в которой хранится результат выполнения функции, следует уничтожить используемый экземпляр набора данных при помощи метода `Dispose`, чтобы освободить занимаемые им ресурсы.

Теперь рассмотрим функцию, осуществляющую регистрацию нового пользователя. Как мы уже говорили, функция в качестве параметров должна принимать логин и пароль нового пользователя. Впрочем, лучше всего о функции расскажет ее код, который приведен в листинге 9.2.

Листинг 9.2

```
<WebMethod()> Public Function NewUser(ByVal LOG As String, ByVal PASS As
String) As Boolean
        Dim DS As New DataSet()
        Dim t As Boolean
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select count(LOGIN) as CID from
USERS where (LOGIN = @LOG)"
        SqlDataAdapter2.InsertCommand.Parameters("@LOGIN").Value = LOG
        SqlDataAdapter2.InsertCommand.Parameters("@PASS").Value = PASS
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 30))
        DataCommand.SelectCommand.Parameters("@LOG").Value = LOG
        DataCommand.Fill(DS)
```

```
        If DS.Tables(0).Rows(0).Item("CID") = 0 Then
            t = True
            SqlConnection1.Open()
            SqlDataAdapter2.InsertCommand.ExecuteNonQuery()
            SqlConnection1.Close()
        Else
            t = False
        End If
    End Function
```

В этой функции прежде всего проверяется наличие в базе данных логина, переданного пользователем. В том случае, если подобный логин в базе данных не найден, выполняется SQL-запрос `INSERT`, связанный со свойством `InsertCommand`, которое, в свою очередь, принадлежит объекту `SqlDataAdapter2`. Этот объект, как мы знаем, обслуживает таблицу `USERS`. Сам SQL-запрос `Insert` выглядит следующим образом:

```
INSERT INTO USERS (LOGIN, PASS)
VALUES            (@LOGIN, @PASS)
```

Так что все действительно просто. Перейдем к следующему этапу разработки. Когда пользователь подключается к Web-сервису, координирующему работу файлообменной сети, ему необходимо передать список файлов из выбранного каталога. Так как Web-сервис хранит в базе данных информацию обо всех файлах, которые доступны для скачивания в текущий момент, было бы удобно, если бы в соответствующую функцию список файлов подключающегося пользователя передавался в виде набора данных подходящей структуры.

Для того чтобы использовать структуру набора данных, его необходимо явно добавить в состав проекта Web-сервиса. Нас интересует набор данных, связанный с таблицей `FILES`, которая обрабатывается объектом `SqlDataAdapter1`. Поэтому для добавления набора данных следует вызвать для этого объекта контекстное меню и выполнить команду **Generate Dataset**. При этом будет активировано одноименное диалоговое окно **Generate Dataset**

Это диалоговое окно позволяет сконфигурировать создаваемый набор данных. В нашем случае стоит выделить переключатель **New** и в соответствующем текстовом поле указать наименование создаваемого набора данных. В списке **Choose which table(s) to add to the dataset** разработчик может указать, какие именно таблицы из используемой базы данных будут входить в состав набора данных. В нашем случае мы можем ограничиться одной таблицей `FILES`, поэтому в списке именно напротив нее и взведен флажок. И, наконец, последний независимый переключатель **Add this dataset to the designer** позволяет разработчику указывать, следует ли отображать создаваемый набор данных на странице дизайна Web-сервиса. Для удобства работы с на-

бором данных можно взвести флажок и в этом независимом переключателе. После нажатия на кнопку **OK** набор данных будет создан и размещен на странице дизайна.

Следует отметить, что структура любого набора данных, создаваемого подобным образом, обязательно описывается при помощи XSD-файла, на который будет создана ссылка в файле-контракте Web-сервиса. XML-код, задающий наш набор данных, приведен в листинге 9.3.

Листинг 9.3

```
<?xml version="1.0" standalone="yes" ?>
<xs:schema id="DataSet1"
targetNamespace="http://www.tempuri.org/DataSet1.xsd"
xmlns:mstns="http://www.tempuri.org/DataSet1.xsd"
xmlns="http://www.tempuri.org/DataSet1.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:
schemas-microsoft-com:xml-msdata" attributeFormDefault="qualified"
elementFormDefault="qualified">
 <xs:element name="DataSet1" msdata:IsDataSet="true"
msdata:Locale="ru-RU">
  <xs:complexType>
   <xs:choice maxOccurs="unbounded">
    <xs:element name="FILES">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="IDFILE" msdata:ReadOnly="true"
msdata:AutoIncrement="true" type="xs:int" />
       <xs:element name="NAIM" type="xs:string" minOccurs="0" />
       <xs:element name="FILESIZE" type="xs:int" minOccurs="0" />
       <xs:element name="LAST_MOD" type="xs:dateTime" minOccurs="0" />
       <xs:element name="LOGIN" type="xs:string" minOccurs="0" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Итак, у нас есть тип набора данных, и мы можем приступать к разработке функции, которая добавляет в базу данных перечень файлов, предоставленных вновь подключившимся пользователем. Код этой функции приведен в листинге 9.4.

Листинг 9.4

```
<WebMethod()> Public Function AddFiles(ByVal Naims() As String, ByVal
Sizes() As Integer, ByVal LastMods() As DateTime, ByVal LOG As String) As
Boolean
        Dim i, count As Integer
        count = Naims.Length
        For i = 0 To count - 1
            SqlInsertCommand1.Parameters("@NAIM").Value = Naims(i)
            SqlInsertCommand1.Parameters("@FILESIZE").Value = Sizes(i)
            SqlInsertCommand1.Parameters("@LAST_MOD").Value = LastMods(i)
            SqlInsertCommand1.Parameters("@LOGIN").Value = LOG
            SqlConnection1.Open()
            SqlInsertCommand1.ExecuteNonQuery()
            SqlConnection1.Close()
        Next
        AddFiles = True
    End Function
```

В этой функции мы добавляем записи в таблицу при помощи SQL-запроса `INSERT`, связанного с объектом `SqlDataAdapter1`. Соответственно, доступ к самому SQL-запросу осуществляется по его имени `SqlInsertCommand1`. Текст этого запроса выглядит следующим образом:

```
INSERT INTO FILES (NAIM, FILESIZE, LAST_MOD, LOGIN)
VALUES (@NAIM, @FILESIZE, @LAST_MOD, @LOGIN)
```

Однако для того, чтобы вставлять данные в таблицу, необходимо их сначала получить. Эти данные передаются в функцию в виде трех массивов. В первом массиве содержатся наименования файлов, второй предназначен для хранения их размеров, а третий содержит даты их последнего обновления. Также в функцию передается в качестве параметра логин пользователя, предоставляющего эти файлы для обмена.

Для того чтобы добавить информацию в базу данных, следует знать, сколько записей нам придется вставлять. Размер массивов (а они все, естественно, имеют одинаковое количество элементов) определяется при помощи их свойства `Length`. А затем мы просто запустим цикл по элементам массивов, передавая эти элементы в качестве параметров запроса `INSERT`. После того как в каждом проходе цикла будут заполнены все необходимые параметры, следует выполнить метод `ExecuteNonQuery`. Еще раз напомним, что для выполнения SQL-запросов, не возвращающих наборов данных, открывать и закрывать соединение с базой данных следует самостоятельно. Впрочем, в коде листинга эти операции хорошо видны.

После того как будет сформировано содержимое таблицы FILES, следует обратиться к разработке функции, позволяющей пользователю осуществлять поиск по фрагменту наименования файла. Код такой функции FindFiles приведен в листинге 9.5.

Листинг 9.5

```
<WebMethod()> Public Function FindFiles(ByVal Text As String) As DataSet1
        Dim DS = New DataSet1()
        SqlSelectCommand1.Parameters("@TEXT").Value = "%" + Text + "%"
        SqlDataAdapter1.Fill(DS)
        FindFiles = DS
    End Function
```

В этой функции для заполнения набора данных мы используем свойство SelectCommand, связанное с объектом SqlDataAdapter1. Естественно, прежде всего следует обратить внимание на текст SQL-запроса, при помощи которого выбираются записи из базы данных. Поскольку поиск будет производиться по подстроке, входящей в наименование файла, следует воспользоваться предикатом LIKE. Так, если мы хотим найти все файлы с расширением txt, текст SQL-запроса будет выглядеть следующим образом:

```
SELECT     NAIM, FILESIZE, LAST_MOD, LOGIN
FROM       FILES
WHERE     (NAIM LIKE '%txt%')
```

Соответственно, свойство SqlSelectCommand1.CommandText выглядит следующим образом:

```
SELECT     NAIM, FILESIZE, LAST_MOD, LOGIN
FROM       FILES
WHERE     (NAIM LIKE @TEXT)
```

При этом необходимо соответствующим образом сгенерировать значение параметра, используемого в SQL-запросе SELECT. Основой для этого SQL-параметра служит строка, которую задает пользователь. Эта строка является параметром, передаваемым в саму функцию. Затем мы просто добавляем знаки % (процент) перед этой строкой и после нее, чтобы использовать такую конструкцию для поиска имен файлов. А после того как параметр SQL-запроса сформирован, требуется всего лишь выполнить SQL-запрос для заполнения набора данных при помощи вызова метода SqlDataAdapter1.Fill.

Теперь нам осталось разработать всего лишь одну, последнюю функцию, которая будет удалять из базы данных файлы пользователей, которые отключаются от файлообменной сети. Для этого следует использовать SQL-

запрос DELETE. Код соответствующей функции ExitUser приведен в листинге 9.6.

Листинг 9.6

```
<WebMethod()> Public Function ExitUser(ByVal Log As String) As Boolean
        SqlDeleteCommand1.Parameters("@LOG").Value = Log
        SqlConnection1.Open()
        SqlDeleteCommand1.ExecuteNonQuery()
        SqlConnection1.Close()
        ExitUser = True
End Function
```

Для работы мы используем свойство DeleteCommand, с которым связываем SQL-запрос Delete. Сам SQL-запрос выглядит следующим образом:

```
DELETE FROM FILES WHERE (LOGIN = @LOG)
```

Остается лишь задать параметр для этого запроса и выполнить его. При этом из таблицы будут удалены все файлы, которые предоставлял выходящий из сети пользователь. Итак, все функции сервиса разработаны. Полный код класса, реализующего Web-сервис, приведен в листинге 9.7.

Листинг 9.7

```
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent()
call

    End Sub
```

```
    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services
Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    Friend WithEvents SqlConnection1 As
System.Data.SqlClient.SqlConnection
    Friend WithEvents SqlSelectCommand2 As
System.Data.SqlClient.SqlCommand
    Friend WithEvents SqlInsertCommand2 As
System.Data.SqlClient.SqlCommand
    Friend WithEvents SqlDataAdapter2 As
System.Data.SqlClient.SqlDataAdapter
    Friend WithEvents SqlSelectCommand1 As
System.Data.SqlClient.SqlCommand
    Friend WithEvents SqlInsertCommand1 As
System.Data.SqlClient.SqlCommand
    Friend WithEvents SqlDataAdapter1 As
System.Data.SqlClient.SqlDataAdapter
    Friend WithEvents DataSet1 As peer.DataSet1
    Friend WithEvents SqlDeleteCommand1 As
System.Data.SqlClient.SqlCommand
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
        Me.SqlSelectCommand2 = New System.Data.SqlClient.SqlCommand()
        Me.SqlInsertCommand2 = New System.Data.SqlClient.SqlCommand()
        Me.SqlDataAdapter2 = New System.Data.SqlClient.SqlDataAdapter()
        Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
        Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
        Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
        Me.SqlDeleteCommand1 = New System.Data.SqlClient.SqlCommand()
        Me.DataSet1 = New peer.DataSet1()
        CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
        '
        'SqlConnection1
        '
```

```
        Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=Peer;persist security info=False;user id=sa;
workstation id=BHV-9IEMVL4EOER;packet size=4096"
        '
        'SqlSelectCommand2
        '
        Me.SqlSelectCommand2.CommandText = "SELECT LOGIN, PASS FROM
USERS"
        Me.SqlSelectCommand2.Connection = Me.SqlConnection1
        '
        'SqlInsertCommand2
        '
        Me.SqlInsertCommand2.CommandText = "INSERT INTO USERS (LOGIN,
PASS) VALUES (@LOGIN, @PASS)"
        Me.SqlInsertCommand2.Connection = Me.SqlConnection1
        Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOGIN",
System.Data.SqlDbType.VarChar, 30, "LOGIN"))
        Me.SqlInsertCommand2.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PASS", Sys-
tem.Data.SqlDbType.VarChar, 30, "PASS"))
        '
        'SqlDataAdapter2
        '
        Me.SqlDataAdapter2.InsertCommand = Me.SqlInsertCommand2
        Me.SqlDataAdapter2.SelectCommand = Me.SqlSelectCommand2
        Me.SqlDataAdapter2.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "USERS", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("LOGIN", "LOGIN"), New
System.Data.Common.DataColumnMapping("PASS", "PASS")})})
        '
        'SqlSelectCommand1
        '
        Me.SqlSelectCommand1.CommandText = "SELECT IDFILE, NAIM,
FILESIZE, LAST_MOD, LOGIN FROM FILES WHERE (NAIM LIKE @TEXT)"
        Me.SqlSelectCommand1.Connection = Me.SqlConnection1
        Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@TEXT",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
        '
        'SqlInsertCommand1
```

```
        '
        Me.SqlInsertCommand1.CommandText = "INSERT INTO FILES (NAIM,
FILESIZE, LAST_MOD, LOGIN) VALUES (@NAIM, @FILESIZE, @LAST_MOD, @LOGIN)"
        Me.SqlInsertCommand1.Connection = Me.SqlConnection1
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@NAIM",
System.Data.SqlDbType.VarChar, 50, "NAIM"))
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@FILESIZE",
System.Data.SqlDbType.Int, 4, "FILESIZE"))
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LAST_MOD",
System.Data.SqlDbType.DateTime, 8, "LAST_MOD"))
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOGIN",
System.Data.SqlDbType.VarChar, 30, "LOGIN"))
        '
        'SqlDataAdapter1
        '
        Me.SqlDataAdapter1.DeleteCommand = Me.SqlDeleteCommand1
        Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
        Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
        Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "FILES", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("IDFILE", "IDFILE"), New
System.Data.Common.DataColumnMapping("NAIM", "NAIM"), New
System.Data.Common.DataColumnMapping("FILESIZE", "FILESIZE"), New
System.Data.Common.DataColumnMapping("LAST_MOD", "LAST_MOD"), New
System.Data.Common.DataColumnMapping("LOGIN", "LOGIN")})})
        '
        'SqlDeleteCommand1
        '
        Me.SqlDeleteCommand1.CommandText = "DELETE FROM FILES WHERE
(LOGIN = @LOG)"
        Me.SqlDeleteCommand1.Connection = Me.SqlConnection1
        Me.SqlDeleteCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.VarChar,
30, System.Data.ParameterDirection.Input, False, CType(0, Byte), CType(0,
Byte), "LOGIN", System.Data.DataRowVersion.Original, Nothing))
        '
        'DataSet1
        '
        Me.DataSet1.DataSetName = "DataSet1"
```

```
        Me.DataSet1.Locale = New System.Globalization.CultureInfo
("ru-RU")
        Me.DataSet1.Namespace = "http://www.tempuri.org/DataSet1.xsd"
        CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()


    End Sub


    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub


#End Region


    <WebMethod()> Public Function LogExist(ByVal LOG As String)
As Boolean
        Dim DS As New DataSet()
        Dim t As Boolean
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select count(LOGIN) as CID from
USERS where (LOGIN = @LOG)"
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 30))
        DataCommand.SelectCommand.Parameters("@LOG").Value = LOG
        DataCommand.Fill(DS)
        If DS.Tables(0).Rows(0).Item("CID") = 0 Then
            t = False
        Else
            t = True
        End If
        DS.Dispose()
```

```
        LogExist = t
    End Function

    <WebMethod()> Public Function NewUser(ByVal LOG As String, ByVal PASS
As String) As Boolean
        Dim DS As New DataSet()
        Dim t As Boolean
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select count(LOGIN) as CID from
USERS where (LOGIN = @LOG)"
        SqlDataAdapter2.InsertCommand.Parameters("@LOGIN").Value = LOG
        SqlDataAdapter2.InsertCommand.Parameters("@PASS").Value = PASS
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 30))
        DataCommand.SelectCommand.Parameters("@LOG").Value = LOG
        DataCommand.Fill(DS)
        If DS.Tables(0).Rows(0).Item("CID") = 0 Then
            t = True
            SqlConnection1.Open()
            SqlDataAdapter2.InsertCommand.ExecuteNonQuery()
            SqlConnection1.Close()
        Else
            t = False
        End If
    End Function
    <WebMethod()> Public Function AddFiles(ByVal Naims() As String, ByVal
Sizes() As Integer, ByVal LastMods() As DateTime, ByVal LOG As String) As
Boolean
        Dim i, count As Integer
        count = Naims.Length
        For i = 0 To count - 1
            SqlInsertCommand1.Parameters("@NAIM").Value = Naims(i)
            SqlInsertCommand1.Parameters("@FILESIZE").Value = Sizes(i)
            SqlInsertCommand1.Parameters("@LAST_MOD").Value = LastMods(i)
            SqlInsertCommand1.Parameters("@LOGIN").Value = LOG
            SqlConnection1.Open()
            SqlInsertCommand1.ExecuteNonQuery()
            SqlConnection1.Close()
```

```
        Next
        AddFiles = True
    End Function
    <WebMethod()> Public Function FindFiles(ByVal Text As String)
As DataSet1
        Dim DS = New DataSet1()
        SqlSelectCommand1.Parameters("@TEXT").Value = "%" + Text + "%"
        SqlDataAdapter1.Fill(DS)
        FindFiles = DS
    End Function
    <WebMethod()> Public Function ExitUser(ByVal Log As String)
As Boolean
        SqlDeleteCommand1.Parameters("@LOG").Value = Log
        SqlConnection1.Open()
        SqlDeleteCommand1.ExecuteNonQuery()
        SqlConnection1.Close()
        ExitUser = True
    End Function
End Class
```

Итак, разработка Web-сервиса закончена. Настало время рассмотреть пример создания клиентского приложения, взаимодействующего с этим Web-сервисом.

Пример клиентского приложения

Структура клиентского приложения должна быть полностью увязана с функциональностью Web-сервиса, поэтому изобретать велосипед нам не придется. Все достаточно просто.

Приложение должно позволять пользователю осуществлять регистрацию в файлообменной сети, передавать список файлов, которые пользователь предоставляет для скачивания, и осуществлять в сети поиск файлов, необходимых пользователю. После того как пользователь найдет в списке необходимые файлы, клиентское приложение должно самостоятельно осуществить прямое соединение с другим пользователем, у которого располагается необходимый файл, и скачать его. Впрочем, эта часть работы уже выходит за пределы рассматриваемого предмета, поэтому мы здесь не будем заниматься реализацией скачивания файлов от удаленного пользователя. Наша прерогатива — рассмотрение взаимодействия клиентского приложения с Web-сервисом.

Разработка начнется как обычно. В диалоговом окне **Solution Explorer** следует добавить ссылку на используемый Web-сервис, затем переместить на

страницу дизайна приложения компонент `DataSet`, связанный с используемым Web-сервисом. Полное наименование этого набора данных выглядит следующим образом: `PeerClient.localhost.DataSet1`, где `PeerClient` — общее наименование класса, связанного с разрабатываемым клиентским приложением.

Теперь разработаем модуль, который позволяет пользователю регистрироваться в файлообменной сети. Для этого потребуется разместить на форме два поля текстового ввода и две кнопки. Одна будет запускать процесс проверки уникальности выбранного логина, а вторая — осуществлять процесс регистрации. Для начала рассмотрим процедуру, проверяющую уникальность введенного логина. Ее код приведен в листинге 9.8.

Листинг 9.8

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim Serv = New PeerClient.localhost.Service1()
        If Serv.LogExist(TextBox1.Text) Then
            MessageBox.Show("Указанный логин уже существует и не может
быть использован для регистрации")
        Else
            MessageBox.Show("Логин можно использовать")
        End If
    End Sub
```

Здесь все просто. Как обычно, создается экземпляр класса, осуществляющего доступ к функциональности Web-сервиса, а потом вызывается один из его методов — в данном случае это функция `LogExist`. В зависимости от полученного результата выводится окно с тем или иным текстовым сообщением. Практически точно так же выглядит листинг функции, осуществляющей первичную регистрацию пользователя в файлообменной системе. Ее код приведен в листинге 9.9.

Листинг 9.9

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim Serv = New PeerClient.localhost.Service1()
        If Serv.NewUser(TextBox1.Text, TextBox2.Text) Then
            MessageBox.Show("Регистрация прошла успешно")
        Else
            MessageBox.Show("Регистрация не удалась")
        End If
    End Sub
```

Здесь даже не требуется каких-либо комментариев. А вот на функцию входа в пиринговую сеть внимание обратить все-таки следует. Пользователь должен указать список файлов, которые будут предоставлены для скачивания другим пользователям файлообменной сети. А мы затем получим всю необходимую информацию об этих файлах, сформируем набор данных, тип которого был импортирован из Web-сервиса, и передадим его в качестве параметра функции `AddFiles`.

Для выбора файлов используется компонент `OpenFileDialog`. Для его нормального функционирования потребуется сначала установить значения некоторых свойств. Так, для свойства `MultiSelect` следует установить значение `True`. Это свойство позволяет пользователю в диалоговом окне выбирать сразу несколько файлов. Также значение `True` следует задать для свойств `AddExtension`, `CheckFileExists` и `CheckPathExists`, которые отвечают, соответственно, за сохранение имени файла вместе с его расширением, проверку существования файла, имя которого пользователь напрямую вводит в текстовое поле, и проверку существования каталога, который пользователь задает самостоятельно. Если требуется задать начальный каталог, содержимое которого будет отображаться в диалоговом окне по умолчанию, стоит использовать свойство `InitialDirectory`.

Также нам потребуется разместить на форме приложения компонент `ListBox`, в котором будут отображаться наименования файлов, выбранных пользователем для предоставления другим членам файлообменной сети, и одну кнопку, которая и будет запускать процесс выбора файлов. После того как файлы будут выбраны, следует получить дополнительную информацию о них, сформировать набор данных и вызвать функцию `AddFiles`. Этот блок действий будет выполняться при нажатии на еще одну дополнительную кнопку. Для удобства пользователей потребуется также разместить на форме кнопку, нажатие на которую очищает список выбранных файлов. Впрочем, эта операция чрезвычайно проста, и нет смысла рассматривать ее в отдельном листинге. Лучше мы рассмотрим процедуру подготовки набора данных и вызова функции `AddFiles` (листинг 9.10).

Листинг 9.10

```
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        Dim i, q As Integer
        Dim t As Boolean
        Dim FInfo As IO.FileInfo
        q = ListBox1.Items.Count - 1
        Dim Names(q) As String
        Dim Sizes(q) As Integer
```

```
        Dim LastMods(q) As DateTime
        Dim Serv = New PeerClient.localhost.Service1()
        For i = 0 To q
            FInfo = New IO.FileInfo(ListBox1.Items(i))
            Names(i) = ListBox1.Items(i)
            Sizes(i) = FInfo.Length
            LastMods(i) = FInfo.LastWriteTime
        Next
        t = Serv.AddFiles(Names, Sizes, LastMods, TextBox1.Text)
        If t Then
            MessageBox.Show("Информация передана в базу данных")
        Else
            MessageBox.Show("Сбой при передаче информации")
        End If
    End Sub
```

Прежде всего нам надо заполнить массивы, которые затем будут переданы в качестве параметров в функцию `AddFiles`. Для этого в цикле обрабатывается каждая строка, отображенная в списке выбранных пользователем файлов. Как известно, в этом списке содержатся полные имена файлов. Нам же помимо имен требуется передавать размеры файлов и дату их последней модификации. Для извлечения указанной информации следует использовать экземпляр класса `FileInfo`. Этот класс специально предназначен для получения различной служебной информации о файлах.

Экземпляр класса раз за разом пересоздается в теле основного цикла, передавая функции-конструктору полное имя интересующего нас в данный момент файла. Затем мы используем свойства `Length` и `LastWriteTime`, в которых указываются размер файла и дата его последней модификации соответственно. Вся полученная информация записывается в элементы массивов. А уже эти массивы передаются в функцию `AddFiles` в качестве параметров. После чего остается лишь анализировать возвращенное функцией значение и на его основе выводить для пользователя то или иное сообщение.

Функция поиска файлов в пиринговой сети, не в пример только что рассмотренному фрагменту кода, будет намного проще. Ее код приведен в листинге 9.11.

Листинг 9.11

```
Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
        Dim DS = New PeerClient.localhost.DataSet1()
        Dim Serv = New PeerClient.localhost.Service1()
```

```
        DS = Serv.FindFiles(TextBox3.Text)
        DataGrid1.DataSource = DS
        DataGrid1.Refresh()
End Sub
```

Этот фрагмент кода достаточно прост и не нуждается в подробном разборе. Напомним лишь, что после получения результата функции, который, как нам известно, является набором данных, мы явно задаем его в качестве источника данных для таблицы `DataGrid1`. И теперь нам осталось только реализовать функцию выхода пользователя из сети. Это совсем просто. Код функции приведен в листинге 9.12.

Листинг 9.12

```
Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
        Dim Serv = New PeerClient.localhost.Service1()
        Serv.ExitUser(TextBox1.Text)
End Sub
```

На этом мы заканчиваем разбор примера клиентского приложения. Окончательный его код, в котором полностью показана дополнительная функциональность клиентского приложения, объявлен весь интерфейс пользователя и иные используемые в работе элементы, приведен в листинге 9.13.

Листинг 9.13

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub
```

```
    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer
    'NOTE: The following procedure is required by the Windows Form
Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
    Friend WithEvents TextBox2 As System.Windows.Forms.TextBox
    Friend WithEvents Button1 As System.Windows.Forms.Button
    Friend WithEvents Button2 As System.Windows.Forms.Button
    Friend WithEvents OpenFileDialog1 As
System.Windows.Forms.OpenFileDialog
    Friend WithEvents Button3 As System.Windows.Forms.Button
    Friend WithEvents ListBox1 As System.Windows.Forms.ListBox
    Friend WithEvents Button4 As System.Windows.Forms.Button
    Friend WithEvents Button5 As System.Windows.Forms.Button
    Friend WithEvents DataSet1 As PeerClient.localhost.DataSet1
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents TextBox3 As System.Windows.Forms.TextBox
    Friend WithEvents Button6 As System.Windows.Forms.Button
    Friend WithEvents DataGrid1 As System.Windows.Forms.DataGrid
    Friend WithEvents Button7 As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.DataSet1 = New PeerClient.localhost.DataSet1()
        Me.Label1 = New System.Windows.Forms.Label()
        Me.Label2 = New System.Windows.Forms.Label()
```

```
        Me.TextBox1 = New System.Windows.Forms.TextBox()
        Me.TextBox2 = New System.Windows.Forms.TextBox()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.Button2 = New System.Windows.Forms.Button()
        Me.OpenFileDialog1 = New System.Windows.Forms.OpenFileDialog()
        Me.Button3 = New System.Windows.Forms.Button()
        Me.ListBox1 = New System.Windows.Forms.ListBox()
        Me.Button4 = New System.Windows.Forms.Button()
        Me.Button5 = New System.Windows.Forms.Button()
        Me.Label3 = New System.Windows.Forms.Label()
        Me.TextBox3 = New System.Windows.Forms.TextBox()
        Me.Button6 = New System.Windows.Forms.Button()
        Me.DataGrid1 = New System.Windows.Forms.DataGrid()
        Me.Button7 = New System.Windows.Forms.Button()
        CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'DataSet1
        '
        Me.DataSet1.DataSetName = "DataSet1"
        Me.DataSet1.Locale = New System.Globalization.CultureInfo
("ru-RU")
        Me.DataSet1.Namespace = "http://www.tempuri.org/DataSet1.xsd"
        '
        'Label1
        '
        Me.Label1.Location = New System.Drawing.Point(16, 8)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(64, 23)
        Me.Label1.TabIndex = 0
        Me.Label1.Text = "Логин"
        '
        'Label2
        '
        Me.Label2.Location = New System.Drawing.Point(16, 40)
        Me.Label2.Name = "Label2"
```

```
Me.Label2.Size = New System.Drawing.Size(56, 23)
Me.Label2.TabIndex = 1
Me.Label2.Text = "Пароль"
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(104, 8)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.TabIndex = 2
Me.TextBox1.Text = ""
'
'TextBox2
'
Me.TextBox2.Location = New System.Drawing.Point(104, 40)
Me.TextBox2.Name = "TextBox2"
Me.TextBox2.TabIndex = 3
Me.TextBox2.Text = ""
'
'Button1
'
Me.Button1.Location = New System.Drawing.Point(232, 8)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(144, 23)
Me.Button1.TabIndex = 4
Me.Button1.Text = "Проверить уникальность"
'
'Button2
'
Me.Button2.Location = New System.Drawing.Point(232, 40)
Me.Button2.Name = "Button2"
Me.Button2.Size = New System.Drawing.Size(144, 23)
Me.Button2.TabIndex = 5
Me.Button2.Text = "Зарегистрироваться"
'
'OpenFileDialog1
'
Me.OpenFileDialog1.InitialDirectory = "d:\"
Me.OpenFileDialog1.Multiselect = True
```

```
Me.OpenFileDialog1.Title = "Выбор файлов"
'
'Button3
'
Me.Button3.Location = New System.Drawing.Point(24, 72)
Me.Button3.Name = "Button3"
Me.Button3.Size = New System.Drawing.Size(176, 23)
Me.Button3.TabIndex = 7
Me.Button3.Text = "Выбор файлов"
'
'ListBox1
'
Me.ListBox1.Location = New System.Drawing.Point(24, 104)
Me.ListBox1.Name = "ListBox1"
Me.ListBox1.Size = New System.Drawing.Size(176, 173)
Me.ListBox1.TabIndex = 8
'
'Button4
'
Me.Button4.Location = New System.Drawing.Point(24, 288)
Me.Button4.Name = "Button4"
Me.Button4.TabIndex = 9
Me.Button4.Text = "Очистить"
'
'Button5
'
Me.Button5.Location = New System.Drawing.Point(112, 288)
Me.Button5.Name = "Button5"
Me.Button5.Size = New System.Drawing.Size(88, 23)
Me.Button5.TabIndex = 10
Me.Button5.Text = "Подтвердить"
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(240, 72)
Me.Label3.Name = "Label3"
Me.Label3.TabIndex = 11
Me.Label3.Text = "Поиск файлов"
```

```
        '
        'TextBox3
        '
        Me.TextBox3.Location = New System.Drawing.Point(344, 72)
        Me.TextBox3.Name = "TextBox3"
        Me.TextBox3.TabIndex = 12
        Me.TextBox3.Text = ""
        '
        'Button6
        '
        Me.Button6.Location = New System.Drawing.Point(456, 72)
        Me.Button6.Name = "Button6"
        Me.Button6.Size = New System.Drawing.Size(112, 23)
        Me.Button6.TabIndex = 13
        Me.Button6.Text = "Искать"
        '
        'DataGrid1
        '
        Me.DataGrid1.DataMember = ""
        Me.DataGrid1.DataSource = Me.DataSet1.FILES
        Me.DataGrid1.HeaderForeColor =
System.Drawing.SystemColors.ControlText
        Me.DataGrid1.Location = New System.Drawing.Point(240, 104)
        Me.DataGrid1.Name = "DataGrid1"
        Me.DataGrid1.Size = New System.Drawing.Size(336, 176)
        Me.DataGrid1.TabIndex = 14
        '
        'Button7
        '
        Me.Button7.Location = New System.Drawing.Point(344, 296)
        Me.Button7.Name = "Button7"
        Me.Button7.Size = New System.Drawing.Size(160, 23)
        Me.Button7.TabIndex = 15
        Me.Button7.Text = "Выход из сети"
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(600, 334)
```

```
        Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.Button7, Me.DataGrid1, Me.Button6, Me.TextBox3, Me.Label3,
Me.Button5, Me.Button4, Me.ListBox1, Me.Button3, Me.Button2, Me.Button1,
Me.TextBox2, Me.TextBox1, Me.Label2, Me.Label1})
        Me.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle
        Me.Name = "Form1"
        Me.Text = "Клиент файлообменной сети"
        CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)

    End Sub

#End Region

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim Serv = New PeerClient.localhost.Service1()
        If Serv.LogExist(TextBox1.Text) Then
            MessageBox.Show("Указанный логин уже существует и не может
быть использован для регистрации")
        Else
            MessageBox.Show("Логин можно использовать")
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Dim Serv = New PeerClient.localhost.Service1()
        If Serv.NewUser(TextBox1.Text, TextBox2.Text) Then
            MessageBox.Show("Регистрация прошла успешно")
        Else
            MessageBox.Show("Регистрация не удалась")
        End If
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
```

```
        Dim i As Integer
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            For i = 0 To OpenFileDialog1.FileNames.Length - 1
                ListBox1.Items.Add(OpenFileDialog1.FileNames(i))
            Next
        End If
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button4.Click
        ListBox1.Items.Clear()
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button5.Click
        Dim i, q As Integer
        Dim t As Boolean
        Dim FInfo As IO.FileInfo
        q = ListBox1.Items.Count - 1
        Dim Names(q) As String
        Dim Sizes(q) As Integer
        Dim LastMods(q) As DateTime
        Dim Serv = New PeerClient.localhost.Service1()
        For i = 0 To q
            FInfo = New IO.FileInfo(ListBox1.Items(i))
            Names(i) = ListBox1.Items(i)
            Sizes(i) = FInfo.Length
            LastMods(i) = FInfo.LastWriteTime
        Next
        t = Serv.AddFiles(Names, Sizes, LastMods, TextBox1.Text)
        If t Then
            MessageBox.Show("Информация передана в базу данных")
        Else
            MessageBox.Show("Сбой при передаче информации")
        End If
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button6.Click
```

```
        Dim DS = New PeerClient.localhost.DataSet1()
        Dim Serv = New PeerClient.localhost.Service1()
        DS = Serv.FindFiles(TextBox3.Text)
        DataGrid1.DataSource = DS
        DataGrid1.Refresh()
    End Sub

    Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button7.Click
        Dim Serv = New PeerClient.localhost.Service1()
        Serv.ExitUser(TextBox1.Text)
    End Sub
End Class
```

Внешний вид основного окна приложения показан на рисунке 9.1.

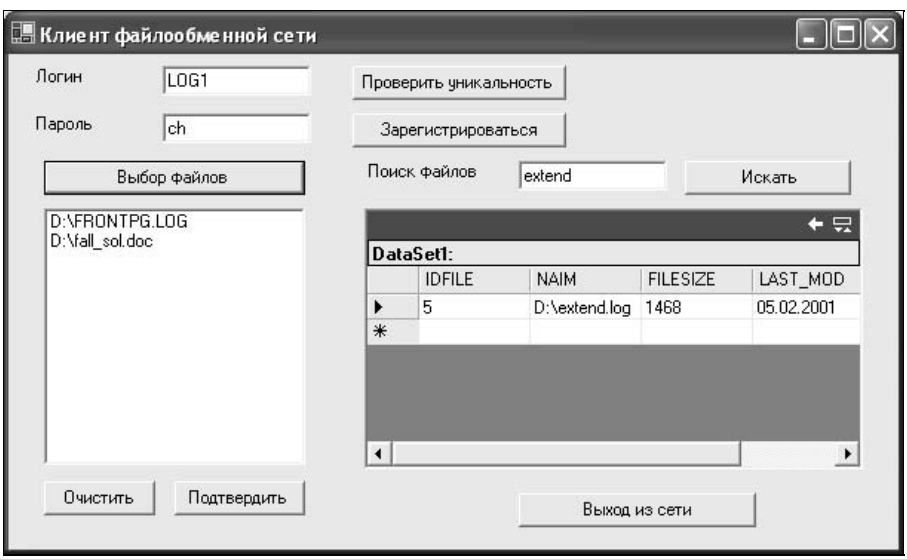

Рис. 9.1. Внешний вид основного окна клиентского приложения

Для полной функциональности, конечно же, нужно разработать для клиентов функции передачи и приема файлов, но это, как мы уже отмечали, выходит за пределы рассматриваемой нами темы.

Обратите внимание на прокси-классы, которые использует клиентское приложение для работы. Хотя от разработчика они скрыты, но ради "общего развития" надо знать, от чего нас избавляет среда разработки Visual Studio .NET. Если бы мы создавали клиентские приложения в иной среде раз-

работки, нам пришлось бы разбираться с ними "вплотную". Код прокси-классов приведен в листинге 9.14.

Листинг 9.14

```
Option Strict Off
Option Explicit On

Imports System
Imports System.ComponentModel
Imports System.Data
Imports System.Diagnostics
Imports System.Runtime.Serialization
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml
Imports System.Xml.Serialization

'
'This source code was auto-generated by Microsoft.VSDesigner, Version
1.0.3705.0.
'
Namespace localhost


    '<remarks/>
    <System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code"),  _
     System.Web.Services.WebServiceBindingAttribute(Name:="Service1Soap",
[Namespace]:="http://tempuri.org/")>  _
    Public Class Service1
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

        '<remarks/>
        Public Sub New()
            MyBase.New
            Me.Url = "http://localhost/peer/Service1.asmx"
        End Sub

        '<remarks/>
```

```
<System.Web.Services.Protocols.SoapDocumentMethodAttribute
("http://tempuri.org/LogExist", RequestNamespace:="http://tempuri.org/",
ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function LogExist(ByVal LOG As String) As Boolean
            Dim results() As Object = Me.Invoke("LogExist", New Object()
{LOG})
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
        Public Function BeginLogExist(ByVal LOG As String, ByVal callback
As System.AsyncCallback, ByVal asyncState As Object) As
System.IAsyncResult
            Return Me.BeginInvoke("LogExist", New Object() {LOG},
callback, asyncState)
        End Function

        '<remarks/>
        Public Function EndLogExist(ByVal asyncResult As
System.IAsyncResult) As Boolean
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
        <System.Web.Services.Protocols.SoapDocumentMethodAttribute
("http://tempuri.org/NewUser", RequestNamespace:="http://tempuri.org/",
ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function NewUser(ByVal LOG As String, ByVal PASS As
String) As Boolean
            Dim results() As Object = Me.Invoke("NewUser", New Object()
{LOG, PASS})
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
        Public Function BeginNewUser(ByVal LOG As String, ByVal PASS As
String, ByVal callback As System.AsyncCallback, ByVal asyncState As
Object) As System.IAsyncResult
```

```
            Return Me.BeginInvoke("NewUser", New Object() {LOG, PASS},
callback, asyncState)
        End Function

        '<remarks/>
        Public Function EndNewUser(ByVal asyncResult As
System.IAsyncResult) As Boolean
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
        <System.Web.Services.Protocols.SoapDocumentMethodAttribute
("http://tempuri.org/AddFiles", RequestNamespace:="http://tempuri.org/",
ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function AddFiles(ByVal Naims() As String, ByVal Sizes()
As Integer, ByVal LastMods() As Date, ByVal LOG As String) As Boolean
            Dim results() As Object = Me.Invoke("AddFiles", New Object()
{Naims, Sizes, LastMods, LOG})
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
        Public Function BeginAddFiles(ByVal Naims() As String, ByVal
Sizes() As Integer, ByVal LastMods() As Date, ByVal LOG As String, ByVal
callback As System.AsyncCallback, ByVal asyncState As Object) As
System.IAsyncResult
            Return Me.BeginInvoke("AddFiles", New Object() {Naims, Sizes,
LastMods, LOG}, callback, asyncState)
        End Function

        '<remarks/>
        Public Function EndAddFiles(ByVal asyncResult As
System.IAsyncResult) As Boolean
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
```

```
<System.Web.Services.Protocols.SoapDocumentMethodAttribute
("http://tempuri.org/FindFiles", RequestNamespace:="http://tempuri.org/",
ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:
=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function FindFiles(ByVal [Text] As String) As DataSet1
            Dim results() As Object = Me.Invoke("FindFiles", New Object()
{[Text]})
            Return CType(results(0),DataSet1)
        End Function

        '<remarks/>
        Public Function BeginFindFiles(ByVal [Text] As String, ByVal
callback As System.AsyncCallback, ByVal asyncState As Object) As
System.IAsyncResult
            Return Me.BeginInvoke("FindFiles", New Object() {[Text]},
callback, asyncState)
        End Function

        '<remarks/>
        Public Function EndFindFiles(ByVal asyncResult As
System.IAsyncResult) As DataSet1
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),DataSet1)
        End Function

        '<remarks/>
        <System.Web.Services.Protocols.SoapDocumentMethodAttribute
("http://tempuri.org/ExitUser", RequestNamespace:="http://tempuri.org/",
ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
        Public Function ExitUser(ByVal Log As String) As Boolean
            Dim results() As Object = Me.Invoke("ExitUser", New Object()
{Log})
            Return CType(results(0),Boolean)
        End Function

        '<remarks/>
        Public Function BeginExitUser(ByVal Log As String, ByVal callback
As System.AsyncCallback, ByVal asyncState As Object) As
System.IAsyncResult
```

```
            Return Me.BeginInvoke("ExitUser", New Object() {Log},
callback, asyncState)
        End Function

        '<remarks/>
        Public Function EndExitUser(ByVal asyncResult As
System.IAsyncResult) As Boolean
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Boolean)
        End Function
    End Class

    <Serializable(), _
     System.ComponentModel.DesignerCategoryAttribute("code"), _
     System.Diagnostics.DebuggerStepThrough(), _
     System.ComponentModel.ToolboxItem(true)> _
    Public Class DataSet1
        Inherits DataSet

        Private tableFILES As FILESDataTable

        Public Sub New()
            MyBase.New
            Me.InitClass
            Dim schemaChangedHandler As
System.ComponentModel.CollectionChangeEventHandler = AddressOf
Me.SchemaChanged
            AddHandler Me.Tables.CollectionChanged, schemaChangedHandler
            AddHandler Me.Relations.CollectionChanged,
schemaChangedHandler
        End Sub

        Protected Sub New(ByVal info As SerializationInfo, ByVal context
As StreamingContext)
            MyBase.New
            Dim strSchema As String = CType(info.GetValue("XmlSchema",
GetType(System.String)),String)
            If (Not (strSchema) Is Nothing) Then
                Dim ds As DataSet = New DataSet
                ds.ReadXmlSchema(New XmlTextReader(New
System.IO.StringReader(strSchema)))
```

```
                If (Not (ds.Tables("FILES")) Is Nothing) Then
                    Me.Tables.Add(New FILESDataTable(ds.Tables("FILES")))
                End If
                Me.DataSetName = ds.DataSetName
                Me.Prefix = ds.Prefix
                Me.Namespace = ds.Namespace
                Me.Locale = ds.Locale
                Me.CaseSensitive = ds.CaseSensitive
                Me.EnforceConstraints = ds.EnforceConstraints
                Me.Merge(ds, false, System.Data.MissingSchemaAction.Add)
                Me.InitVars
            Else
                Me.InitClass
            End If
            Me.GetSerializationData(info, context)
            Dim schemaChangedHandler As
System.ComponentModel.CollectionChangeEventHandler = AddressOf
Me.SchemaChanged
            AddHandler Me.Tables.CollectionChanged, schemaChangedHandler
            AddHandler Me.Relations.CollectionChanged,
schemaChangedHandler
        End Sub

        <System.ComponentModel.Browsable(false),  _
         System.ComponentModel.DesignerSerializationVisibilityAttribute
(System.ComponentModel.DesignerSerializationVisibility.Content)>  _
        Public ReadOnly Property FILES As FILESDataTable
            Get
                Return Me.tableFILES
            End Get
        End Property

        Public Overrides Function Clone() As DataSet
            Dim cln As DataSet1 = CType(MyBase.Clone,DataSet1)
            cln.InitVars
            Return cln
        End Function

        Protected Overrides Function ShouldSerializeTables() As Boolean
```

```
            Return false
        End Function

        Protected Overrides Function ShouldSerializeRelations()
As Boolean
            Return false
        End Function

        Protected Overrides Sub ReadXmlSerializable(ByVal reader As
XmlReader)
            Me.Reset
            Dim ds As DataSet = New DataSet
            ds.ReadXml(reader)
            If (Not (ds.Tables("FILES")) Is Nothing) Then
                Me.Tables.Add(New FILESDataTable(ds.Tables("FILES")))
            End If
            Me.DataSetName = ds.DataSetName
            Me.Prefix = ds.Prefix
            Me.Namespace = ds.Namespace
            Me.Locale = ds.Locale
            Me.CaseSensitive = ds.CaseSensitive
            Me.EnforceConstraints = ds.EnforceConstraints
            Me.Merge(ds, false, System.Data.MissingSchemaAction.Add)
            Me.InitVars
        End Sub

        Protected Overrides Function GetSchemaSerializable() As
System.Xml.Schema.XmlSchema
            Dim stream As System.IO.MemoryStream = New
System.IO.MemoryStream
            Me.WriteXmlSchema(New XmlTextWriter(stream, Nothing))
            stream.Position = 0
            Return System.Xml.Schema.XmlSchema.Read(New
XmlTextReader(stream), Nothing)
        End Function

        Friend Sub InitVars()
            Me.tableFILES = CType(Me.Tables("FILES"),FILESDataTable)
            If (Not (Me.tableFILES) Is Nothing) Then
                Me.tableFILES.InitVars
```

```
            End If
        End Sub

        Private Sub InitClass()
            Me.DataSetName = "DataSet1"
            Me.Prefix = ""
            Me.Namespace = "http://www.tempuri.org/DataSet1.xsd"
            Me.Locale = New System.Globalization.CultureInfo("ru-RU")
            Me.CaseSensitive = false
            Me.EnforceConstraints = true
            Me.tableFILES = New FILESDataTable
            Me.Tables.Add(Me.tableFILES)
        End Sub

        Private Function ShouldSerializeFILES() As Boolean
            Return false
        End Function

        Private Sub SchemaChanged(ByVal sender As Object, ByVal e As
System.ComponentModel.CollectionChangeEventArgs)
            If (e.Action =
System.ComponentModel.CollectionChangeAction.Remove) Then
                Me.InitVars
            End If
        End Sub

        Public Delegate Sub FILESRowChangeEventHandler(ByVal sender As
Object, ByVal e As FILESRowChangeEvent)

        <System.Diagnostics.DebuggerStepThrough()>  _
        Public Class FILESDataTable
            Inherits DataTable
            Implements System.Collections.IEnumerable

            Private columnIDFIL6 As DataColumn

            Private columnNAIM As DataColumn

            Private columnFILESIZE As DataColumn
```

```
            Private columnLAST_MOD As DataColumn

            Private columnLOGIN As DataColumn

            Friend Sub New()
                MyBase.New("FILES")
                Me.InitClass
            End Sub

            Friend Sub New(ByVal table As DataTable)
                MyBase.New(table.TableName)
                If (table.CaseSensitive <> table.DataSet.CaseSensitive)
Then
                    Me.CaseSensitive = table.CaseSensitive
                End If
                If (table.Locale.ToString <>
table.DataSet.Locale.ToString) Then
                    Me.Locale = table.Locale
                End If
                If (table.Namespace <> table.DataSet.Namespace) Then
                    Me.Namespace = table.Namespace
                End If
                Me.Prefix = table.Prefix
                Me.MinimumCapacity = table.MinimumCapacity
                Me.DisplayExpression = table.DisplayExpression
            End Sub

            <System.ComponentModel.Browsable(false)>  _
            Public ReadOnly Property Count As Integer
                Get
                    Return Me.Rows.Count
                End Get
            End Property

            Friend ReadOnly Property IDFILEColumn As DataColumn
                Get
                    Return Me.columnIDFIL_par
                End Get
            End Property

            Friend ReadOnly Property NAIMColumn As DataColumn
```

```
                Get
                    Return Me.columnNAIM
                End Get
            End Property

            Friend ReadOnly Property FILESIZEColumn As DataColumn
                Get
                    Return Me.columnFILESIZE
                End Get
            End Property

            Friend ReadOnly Property LAST_MODColumn As DataColumn
                Get
                    Return Me.columnLAST_MOD
                End Get
            End Property

            Friend ReadOnly Property LOGINColumn As DataColumn
                Get
                    Return Me.columnLOGIN
                End Get
            End Property

            Public Default ReadOnly Property Item(ByVal index As Integer)
As FILESRow
                Get
                    Return CType(Me.Rows(index),FILESRow)
                End Get
            End Property

            Public Event FILESRowChanged As FILESRowChangeEventHandler

            Public Event FILESRowChanging As FILESRowChangeEventHandler

            Public Event FILESRowDeleted As FILESRowChangeEventHandler

            Public Event FILESRowDeleting As FILESRowChangeEventHandler

            Public Overloads Sub AddFILESRow(ByVal row As FILESRow)
```

```
            Me.Rows.Add(row)
        End Sub

        Public Overloads Function AddFILESRow(ByVal NAIM As String,
ByVal FILESIZE As Integer, ByVal LAST_MOD As Date, ByVal LOGIN As String)
As FILESRow
            Dim rowFILESRow As FILESRow = CType(Me.NewRow,FILESRow)
            rowFILESRow.ItemArray = New Object() {Nothing, NAIM,
FILESIZE, LAST_MOD, LOGIN}
            Me.Rows.Add(rowFILESRow)
            Return rowFILESRow
        End Function

        Public Function GetEnumerator() As
System.Collections.IEnumerator Implements
System.Collections.IEnumerable.GetEnumerator
            Return Me.Rows.GetEnumerator
        End Function

        Public Overrides Function Clone() As DataTable
            Dim cln As FILESDataTable =
CType(MyBase.Clone,FILESDataTable)
            cln.InitVars
            Return cln
        End Function

        Protected Overrides Function CreateInstance() As DataTable
            Return New FILESDataTable
        End Function

        Friend Sub InitVars()
            Me.columnIDFILF = Me.Columns("IDFILE")
            Me.columnNAIM = Me.Columns("NAIM")
            Me.columnFILESIZE = Me.Columns("FILESIZE")
            Me.columnLAST_MOD = Me.Columns("LAST_MOD")
            Me.columnLOGIN = Me.Columns("LOGIN")
        End Sub

        Private Sub InitClass()
            Me.columnIDFIL u32 ?= New DataColumn("IDFILE",
GetType(System.Int32), Nothing, System.Data.MappingType.Element)
```

```
                Me.Columns.Add(Me.columnIDFIL@u41 ?
                Me.columnNAIM = New DataColumn("NAIM",
GetType(System.String), Nothing, System.Data.MappingType.Element)
                Me.Columns.Add(Me.columnNAIM)
                Me.columnFILESIZE = New DataColumn("FILESIZE",
GetType(System.Int32), Nothing, System.Data.MappingType.Element)
                Me.Columns.Add(Me.columnFILESIZE)
                Me.columnLAST_MOD = New DataColumn("LAST_MOD",
GetType(System.DateTime), Nothing, System.Data.MappingType.Element)
                Me.Columns.Add(Me.columnLAST_MOD)
                Me.columnLOGIN = New DataColumn("LOGIN",
GetType(System.String), Nothing, System.Data.MappingType.Element)
                Me.Columns.Add(Me.columnLOGIN)
                Me.columnIDFILж.AutoIncrement = true
                Me.columnIDFILR.AllowDBNull = false
                Me.columnIDFILj.ReadOnly = true
            End Sub

            Public Function NewFILESRow() As FILESRow
                Return CType(Me.NewRow,FILESRow)
            End Function

            Protected Overrides Function NewRowFromBuilder(ByVal builder
As DataRowBuilder) As DataRow
                Return New FILESRow(builder)
            End Function

            Protected Overrides Function GetRowType() As System.Type
                Return GetType(FILESRow)
            End Function

            Protected Overrides Sub OnRowChanged(ByVal e As
DataRowChangeEventArgs)
                MyBase.OnRowChanged(e)
                If (Not (Me.FILESRowChangedEvent) Is Nothing) Then
                    RaiseEvent FILESRowChanged(Me, New
FILESRowChangeEvent(CType(e.Row,FILESRow), e.Action))
                End If
            End Sub

            Protected Overrides Sub OnRowChanging(ByVal e As
DataRowChangeEventArgs)
```

```
                MyBase.OnRowChanging(e)
                If (Not (Me.FILESRowChangingEvent) Is Nothing) Then
                    RaiseEvent FILESRowChanging(Me, New
FILESRowChangeEvent(CType(e.Row,FILESRow), e.Action))
                End If
            End Sub

            Protected Overrides Sub OnRowDeleted(ByVal e As
DataRowChangeEventArgs)
                MyBase.OnRowDeleted(e)
                If (Not (Me.FILESRowDeletedEvent) Is Nothing) Then
                    RaiseEvent FILESRowDeleted(Me, New
FILESRowChangeEvent(CType(e.Row,FILESRow), e.Action))
                End If
            End Sub

            Protected Overrides Sub OnRowDeleting(ByVal e As
DataRowChangeEventArgs)
                MyBase.OnRowDeleting(e)
                If (Not (Me.FILESRowDeletingEvent) Is Nothing) Then
                    RaiseEvent FILESRowDeleting(Me, New
FILESRowChangeEvent(CType(e.Row,FILESRow), e.Action))
                End If
            End Sub

            Public Sub RemoveFILESRow(ByVal row As FILESRow)
                Me.Rows.Remove(row)
            End Sub
        End Class

        <System.Diagnostics.DebuggerStepThrough()>  _
        Public Class FILESRow
            Inherits DataRow

            Private tableFILES As FILESDataTable

            Friend Sub New(ByVal rb As DataRowBuilder)
                MyBase.New(rb)
                Me.tableFILES = CType(Me.Table,FILESDataTable)
            End Sub
```

```
            Public Property IDFILE As Integer
                Get
                    Return CType(Me(Me.tableFILES.IDFILEColumn),Integer)
                End Get
                Set
                    Me(Me.tableFILES.IDFILEColumn) = value
                End Set
            End Property

            Public Property NAIM As String
                Get
                    Try
                        Return CType(Me(Me.tableFILES.NAIMColumn),String)
                    Catch e As InvalidCastException
                        Throw New StrongTypingException("Cannot get value
because it is DBNull.", e)
                    End Try
                End Get
                Set
                    Me(Me.tableFILES.NAIMColumn) = value
                End Set
            End Property

            Public Property FILESIZE As Integer
                Get
                    Try
                        Return
CType(Me(Me.tableFILES.FILESIZEColumn),Integer)
                    Catch e As InvalidCastException
                        Throw New StrongTypingException("Cannot get value
because it is DBNull.", e)
                    End Try
                End Get
                Set
                    Me(Me.tableFILES.FILESIZEColumn) = value
                End Set
            End Property

            Public Property LAST_MOD As Date
                Get
```

```
                    Try
                        Return
CType(Me(Me.tableFILES.LAST_MODColumn),Date)
                    Catch e As InvalidCastException
                        Throw New StrongTypingException("Cannot get value
because it is DBNull.", e)
                    End Try
                End Get
                Set
                    Me(Me.tableFILES.LAST_MODColumn) = value
                End Set
            End Property

            Public Property LOGIN As String
                Get
                    Try
                        Return
CType(Me(Me.tableFILES.LOGINColumn),String)
                    Catch e As InvalidCastException
                        Throw New StrongTypingException("Cannot get value
because it is DBNull.", e)
                    End Try
                End Get
                Set
                    Me(Me.tableFILES.LOGINColumn) = value
                End Set
            End Property

            Public Function IsNAIMNull() As Boolean
                Return Me.IsNull(Me.tableFILES.NAIMColumn)
            End Function

            Public Sub SetNAIMNull()
                Me(Me.tableFILES.NAIMColumn) = System.Convert.DBNull
            End Sub

            Public Function IsFILESIZENull() As Boolean
                Return Me.IsNull(Me.tableFILES.FILESIZEColumn)
            End Function
```

```
            Public Sub SetFILESIZENull()
                Me(Me.tableFILES.FILESIZEColumn) = System.Convert.DBNull
            End Sub

            Public Function IsLAST_MODNull() As Boolean
                Return Me.IsNull(Me.tableFILES.LAST_MODColumn)
            End Function

            Public Sub SetLAST_MODNull()
                Me(Me.tableFILES.LAST_MODColumn) = System.Convert.DBNull
            End Sub

            Public Function IsLOGINNull() As Boolean
                Return Me.IsNull(Me.tableFILES.LOGINColumn)
            End Function

            Public Sub SetLOGINNull()
                Me(Me.tableFILES.LOGINColumn) = System.Convert.DBNull
            End Sub
        End Class

        <System.Diagnostics.DebuggerStepThrough()>  _
        Public Class FILESRowChangeEvent
            Inherits EventArgs

            Private eventRow As FILESRow

            Private eventAction As DataRowAction

            Public Sub New(ByVal row As FILESRow, ByVal action As
DataRowAction)
                MyBase.New
                Me.eventRow = row
                Me.eventAction = action
            End Sub

            Public ReadOnly Property Row As FILESRow
                Get
                    Return Me.eventRow
```

```
                End Get
            End Property

            Public ReadOnly Property Action As DataRowAction
                Get
                    Return Me.eventAction
                End Get
            End Property
        End Class
    End Class
End Namespace
```

И на этом мы заканчиваем рассмотрение примера создания Web-сервиса, координирующего действие файлообменной сети. Настало время перейти к последней главе книги, в которой мы познакомимся еще с одним весьма интересным аспектом использования Web-сервисов — разработкой распределенных приложений.

Глава 10

Распределенные приложения

Новая модель

Строго говоря, распределенные приложения создавались и до появления технологии Microsoft .NET. Для этого использовались самые различные технологии работы, от COM-объектов до создания Web-приложений, в которых роль клиентов выполняли обычные браузеры. Но идея оставалась единой. Основная функциональность выносилась за пределы рабочего места пользователя, куда-нибудь на высокопроизводительный сервер. Эта структура задачи весьма похожа на методику работы Web-сервисов, о которой мы говорим на протяжении всей книги.

Но для чего были нужны такие распределенные приложения? Ответ достаточно прост. Создание подобного приложения позволяло упростить его развертывание в больших корпоративных сетях, уменьшить совокупную стоимость (TCO, Total Cost Ownership) владения этим программным продуктом для пользователя и облегчить политику лицензирования для производителя программного обеспечения.

Смысл здесь в том, что достаточно серьезное по своим возможностям ядро распределенного приложения ставится всего один раз на сервер. Клиентские приложения в установке гораздо проще, их инсталляция занимает намного меньше времени, поэтому по сравнению с инсталляцией обычного приложения в корпоративной сети администратор получает серьезное временное преимущество. Тем, кто считает, что проблема затраченного на инсталляции времени не заслуживает внимания, рекомендуем установить пакет Microsoft Office хотя бы на двести машин. После этого к общему времени инсталляции начинаешь относиться весьма критично.

Далее, за счет того, что клиентская часть приложения весьма проста, уменьшаются затраты на ее обслуживание. Таким образом, снижается общая стоимость владения этим программным продуктом, что для организаций с немалым парком машин тоже имеет весьма большое значение.

И, наконец, подобная структура программного продукта упрощает лицензирование для продавца и покупателя. Нет нужды вводить идентификационный номер на каждую отдельную машину. Подобная схема достаточно сильно напоминает уже применяющуюся, когда одна лицензия выдается на весь офис (site license). Но в дополнение к этому, продавец программного обеспечения получает возможность не только продавать приложение, но и передавать его в аренду. И если основная часть базируется на одном сервере, очень сильно упрощается процедура контроля за временем использования сданного в аренду приложения.

В том случае, если распределенное приложение мы создаем на основе Web-сервисов, у нас практически нет ограничений на функциональность разрабатываемой программы. Разработчик вполне может создать программное обеспечение любого профиля, от текстового процессора до сложных систем управления предприятием. Строго говоря, подобные приложения могут взаимодействовать с пользователями даже через Интернет, благо Web-сервисы отлично подходят для этой цели. Однако здесь есть и некоторые сложности.

Предположим, мы разработали текстовый процессор. Для клиентского приложения необходимо уметь лишь загружать, отображать и сохранять текст, который будет обрабатываться функциональным ядром распределенного приложения. Но при этом каждая операция с текстом, будь то изменение шрифта для текстового фрагмента или удаление символов, будет производиться на сервере приложения. Следовательно, для выполнения подобных операций будет необходимо передавать на сервер достаточно большой блок информации, а после выполнения операции этот блок принимать. Таким образом, получается, что для полноценной работы распределенного текстового процессора необходимо передавать огромные количества информации. Естественно, Интернет пока не может предоставить возможность быстрого перемещения таких объемов информации, и работа с распределенным приложением через Интернет будет весьма медленной. Поэтому основное место применения распределенных приложений — скоростные локальные сети организаций с достаточно мощными серверами. Именно в этой среде распределенные приложения наилучшим образом демонстрируют все свои преимущества.

В данной главе мы рассмотрим пример создания подобного распределенного приложения. Чтобы не приводить длинные листинги, без нужды увеличивая объем книги, пример будет не такой уж большой и серьезный. Действительно, разбирать здесь пример разработки распределенного текстового процессора явно не стоит. Попробуем создать приложение, позволяющее пользователям отправлять электронные письма. Это будет не полноценный почтовый клиент, так как мы не будем рассматривать функции приема и обработки электронных сообщений. Мы ограничимся лишь их отправкой.

Распределенное приложение, построенное на основе Web-сервисов, естественным образом разбивается на две части — сервис и клиент. Впрочем, к этой главе подобная структура уже не должна быть для вас новой. Начнем, конечно, с разработки Web-сервиса.

Структура распределенного приложения

Прежде чем приступать к кодированию, всегда необходимо максимально точно и детально поставить задачу — что же именно будет делать Web-сервис.

Примем, что его основное занятие — передача электронных писем пользователей. Каждый пользователь создаст свою учетную запись, в которой укажет почтовый сервер, через который будет отправляться почта, свой электронный адрес и имя, которым будет подписываться электронное сообщение. Нам потребуется хранить логин и пароль пользователя, чтобы никто не смог воспользоваться чужой учетной записью. Каждому пользователю будет соответствовать уникальный идентификатор, передаваемый Web-сервисом клиентскому приложению после того, как пользователь укажет свои логин и пароль. Этот идентификатор клиентское приложение будет передавать сервису для выполнения каждой операции, чтобы ядро приложения могло знать, какую именно учетную запись использовать для отправки сообщений.

Теперь, когда задача поставлена, мы рассмотрим структуру разрабатываемого сервиса. Прежде всего следует отметить, что нам опять потребуется помощь SQL-сервера. Поскольку придется работать с персональными настройками пользователей, следовательно, понадобится хранить их персональные данные. А для хранения данных ничего лучше баз данных еще не придумали.

Итак, основываясь на приведенной информации, мы уже можем сказать, какова будет структура используемой базы данных. В основной таблице будет храниться целочисленный идентификатор пользователя, его логин и пароль, полное имя, которым будут подписываться письма, и адрес используемого почтового сервера. Соответственно, структура основной таблицы `USERS`, входящей в состав базы данных `MAILER`, будет описываться следующим SQL-выражением:

```
CREATE TABLE [dbo].[USERS] (
    [USERID] [int] IDENTITY (1, 1) NOT NULL,
    [LOGIN] [char] (10) COLLATE Cyrillic_General_CI_AS NULL,
    [PASSWORD] [char] (10) COLLATE Cyrillic_General_CI_AS NULL,
    [USERNAME] [char] (50) COLLATE Cyrillic_General_CI_AS NULL,
    [MAILSERVER] [char] (50) COLLATE Cyrillic_General_CI_AS NULL
) ON [PRIMARY]
```

Далее рассмотрим список функций Web-сервиса и их предназначение. Прежде всего, нам потребуется блок аутентификации пользователей. Мы уже не раз разрабатывали его аналог в предыдущих главах, поэтому сейчас ничего нового нам не предстоит. Потребуется так же функция проверки существования логина и функция добавления новой учетной записи в базу данных. Потребуется и еще одна функция, занимающаяся отправкой писем от пользователя.

Что касается клиентской части, то она, естественно, будет создаваться в виде отдельного Windows-приложения. Исходя из идеологии распределенных приложений, нам нет нужды создавать клиентскую часть в виде Web-интерфейса. Вообще, вся та часть разработки, которая опирается на Web, должна быть скрыта от простого пользователя. Он должен считать, что работает с обычным приложением. О том, что ядром распределенного приложения будет Web-сервис, может знать только технический персонал.

Так как в качестве ядра используется Web-сервис, необходимо, чтобы на центральном сервере помимо SQL-сервера функционировал бы еще и WWW-сервер IIS. Естественно, чтобы клиентские приложения могли без лишних сложностей связываться с ядром, необходимо, чтобы корпоративная сеть была построена в виде интрасети. При соблюдении всех вышеперечисленных условий распределенное приложение будет функционировать, не создавая излишних проблем ни конечным пользователям, ни техническому персоналу.

Разработка ядра приложения

Прежде чем приступать к написанию кода ядра распределенного приложения, следует создать новый проект Web-сервиса `MailServ` и отбуксировать на страницу дизайна сервиса все необходимые компоненты. Прежде всего, конечно, нам потребуется установить соединение с SQL-сервером. Эта процедура нам уже знакома — в предыдущих главах мы не раз ее выполняли. После того как соединение будет добавлено в список **Data Connections**, отображаемый в окне **Server Explorer**, на странице дизайна появится компонент `SqlConnection1`. Затем отбуксируем на страницу дизайна таблицу `USERS`, и на этой странице появится компонент `SqlDataAdapter1`, связанный с указанной таблицей. Свойства `SelectCommand` и `InsertCommand` для этого компонента следует изменить. Свойство `SelectCommand` будет содержать следующий SQL-запрос:

```
SELECT    *
FROM      USERS
WHERE     (LOGIN = @LOG) AND (PASSWORD = @PASS)
```

А в свойстве `InsertCommand` будет храниться SQL-запрос вида:

```
INSERT INTO USERS
            (LOGIN, PASSWORD, USERNAME, MAILSERVER)
VALUES      (@LOGIN, @PASSWORD, @USERNAME, @MAILSERVER)
```

После того как будет сконфигурирован компонент `SqlDataAdapter1`, следует на его основе сгенерировать набор данных, который будет представлен компонентом `DataSet1`. Структура набора данных описывается соответствующей XML-схемой, которая хранится в файле DataSet1.xsd. Код этого файла приведен в листинге 10.1.

Листинг 10.1

```
<?xml version="1.0" standalone="yes" ?>
<xs:schema id="DataSet1"
targetNamespace="http://www.tempuri.org/DataSet1.xsd"
xmlns:mstns="http://www.tempuri.org/DataSet1.xsd"
xmlns="http://www.tempuri.org/DataSet1.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata=
"urn:schemas-microsoft-com:xml-msdata" attributeFormDefault="qualified"
elementFormDefault="qualified">
 <xs:element name="DataSet1" msdata:IsDataSet="true" msdata:Locale=
"ru-RU">
  <xs:complexType>
   <xs:choice maxOccurs="unbounded">
    <xs:element name="USERS">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="USERID" msdata:ReadOnly="true"
msdata:AutoIncrement="true" type="xs:int" />
       <xs:element name="LOGIN" type="xs:string" minOccurs="0" />
       <xs:element name="PASSWORD" type="xs:string" minOccurs="0" />
       <xs:element name="USERNAME" type="xs:string" minOccurs="0" />
       <xs:element name="MAILSERVER" type="xs:string" minOccurs="0" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:choice>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Итак, все компоненты, которые необходимы при разработке Web-сервиса, размещены на странице его дизайна и сконфигурированы. Настало время

рассмотреть код функций ядра распределенного приложения. Начнем мы с разработки функции `LogExist`, которая проверяет существование логина, переданного функции в виде параметра. В качестве результата функция возвращает значение логического типа. Код функции приведен в листинге 10.2.

Листинг 10.2

```
<WebMethod()> Public Function LogExist(ByVal LOG As String) As Boolean
        Dim DS As New DataSet()
        Dim t As Boolean
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select count(LOGIN) as CID from
USERS where (LOGIN = @LOG)"
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 30))
        DataCommand.SelectCommand.Parameters("@LOG").Value = LOG
        DataCommand.Fill(DS)
        If DS.Tables(0).Rows(0).Item("CID") = 0 Then
            t = False
        Else
            t = True
        End If
        DS.Dispose()
        LogExist = t
End Function
```

Теперь рассмотрим код функции, добавляющей новую учетную запись в базу данных. Этот код приведен в листинге 10.3.

Листинг 10.3

```
<WebMethod()> Public Function NewUser(ByVal LOG As String, ByVal PASS As
String, ByVal USERNAME As String, ByVal MAILSERV As String) As Boolean
        SqlInsertCommand1.Parameters("LOGIN").Value = LOG
        SqlInsertCommand1.Parameters("PASSWORD").Value = PASS
        SqlInsertCommand1.Parameters("USERNAME").Value = USERNAME
        SqlInsertCommand1.Parameters("MAILSERVER").Value = MAILSERV
        SqlConnection1.Open()
        SqlInsertCommand1.ExecuteNonQuery()
```

```
        SqlConnection1.Close()
        NewUser = True
End Function
```

Рассмотрим далее процедуру аутентификации пользователя в системе. При подключении он должен указать логин и пароль, а на основе их функция Web-сервиса возвратит уникальный целочисленный идентификатор пользователя, который будет храниться в поле `USERID`. Код этой функции приведен в листинге 10.4.

Листинг 10.4

```
<WebMethod()> Public Function LogOnUser(ByVal LOG As String, ByVal PASS
As String) As Integer
        Dim DS As New DataSet()
        SqlSelectCommand1.Parameters("LOG").Value = LOG
        SqlSelectCommand1.Parameters("PASS").Value = PASS
        SqlDataAdapter1.Fill(DS)
        If DS.Tables(0).Rows(0).Item("USERID") Is Nothing Then
            LogOnUser = -1
        Else
            LogOnUser = DS.Tables(0).Rows(0).Item("USERID")
End If
```

Мы используем набор данных, который заполняется при помощи метода `Fill`, присущего объекту `SqlDataAdapter`. Естественно, для этого автоматически используется SQL-запрос `Select`, параметры которому передаются из клиентского приложения. Если логин и пароль соответствуют друг другу, в таблице найдется соответствующая запись, и уникальный целочисленный идентификатор пользователя будет возвращен в качестве результата действия функции. В ином случае возвращается значение `-1`.

После того как клиентское приложение получило целочисленный идентификатор, можно предоставлять пользователю доступ к основной функции распределенного приложения — отправке письма. Для этого в Web-сервисе присутствует еще одна, последняя функция, код которой приведен в листинге 10.5.

Листинг 10.5

```
<WebMethod()> Public Function UserMail(ByVal USERID As Integer, ByVal
MailBody As String, ByVal MailSubject As String, ByVal MailTo As String)
As Boolean
        Dim DS As New DataSet()
```

```
        Dim email As New Web.Mail.MailMessage()
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        'Dim smtp As New System.Web.Mail.SmtpMail()
        Dim SelectCommand As String = "select * from USERS where
(USERID = @USERID)"
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@USERID", System.Data.SqlDbType.Int, 4))
        DataCommand.SelectCommand.Parameters("@USERID").Value = USERID
        DataCommand.Fill(DS)
        email.To = MailTo
        email.From = DS.Tables(0).Rows(0).Item("MAILSERVER").Value
        email.Subject = MailSubject
        email.Body = MailBody
        Web.Mail.SmtpMail.Send(email)
        UserMail = True
    End Function
```

В этой функции нам прежде всего потребуется получить обратный адрес пользователя, который отправляет письмо, для чего мы выполняем SQL-запрос `Select` следующего вида:

```
select * from USERS where (USERID = @USERID)
```

При помощи такого запроса формируется набор данных, состоящий из единственной записи. Причем в ней нас будет интересовать одно поле — в котором хранится обратный адрес электронной почты пользователя.

Этот адрес мы будем использовать для формирования отсылаемого электронного письма. Кроме обратного адреса нам, естественно, понадобится адрес получателя, тема письма и сам текст письма. Эти данные передаются в функцию в качестве параметров типа `String`.

Для отправки письма используется класс `SmtpMail`, располагающийся в пространстве имен `System.Web.Mail`. Метод `Send` отсылает письмо на почтовый сервер. В качестве параметра методу `Send` передается значение типа `MailMessage`. Именно этот тип полностью определяет отсылаемое сообщение. Рассмотрим его подробнее.

- `Attachments`. Свойство задает список файлов, присоединяемых к письму. Свойство имеет значение типа `IList`. Этот тип является коллекцией наименований файлов.
- `Bcc`. Свойство типа `String`. Его значением является список адресов электронной почты, по которым будет отправлено электронное письмо. Следует, однако, заметить, что в этом свойстве задаются адреса, входящие в

список **BCC** (Blind Carbon Copy, скрытая копия), то есть получатели, чьи адреса указаны в полях **To** (Кому) и **Copy** (Копия), не увидят в заголовке письма адреса из списка **BCC**. Адреса, входящие в список **BCC**, разделяются запятой.

- `Body`. Свойство типа `String`. В нем находится само тело электронного письма.
- `BodyEncoding`. Свойство типа `Encoding`. Задает кодировку символов сообщения. Тип `Encoding` является перечислимым типом.
- `BodyFormat`. Свойство типа `MailFormat`, в котором устанавливается формат тела электронного письма. Свойство может принимать значения `Text` и `Html`.
- `Cc`. Свойство типа `String`. В этом свойстве записываются адреса, разделенные запятыми, по которым будет отправлена копия электронного сообщения.
- `From`. Свойство типа `String`, в котором указывается электронный адрес отправителя письма.
- `Priority`. Свойство перечислимого типа `MailPriority`, в котором указывается приоритет отсылаемого письма. Свойство может принимать значения `High`, `Low` и `Normal`.
- `Subject`. Свойство типа `String`. В нем записывается тема отправляемого письма.
- `To`. Свойство типа `String`. В нем указывается адрес электронной почты, по которому будет отправлено создаваемое сообщение.

После того как мы заполнили минимально необходимые поля экземпляра объекта `MailMessage`, письмо необходимо отослать. Для этого, как мы уже говорили, используется метод `Send`. При этом нет нужды создавать экземпляр объекта `SmtpMail`, мы можем вызвать необходимый метод напрямую. При подобном вызове метода `Send` используется стандартный SMTP-сервер, входящий в состав IIS. В том случае, если для отправки писем необходимо обратиться к некому стороннему SMTP-серверу, следует воспользоваться свойством `SmtpServer`, но для этого придется все-таки создать экземпляр объекта `SmtpMail`.

Теперь ядро нашего распределенного приложения готово, и мы можем привести его полностью (листинг 10.6).

Листинг 10.6

```
Imports System.Web.Services


<WebService(Namespace := "http://tempuri.org/")> _
```

```
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent()
call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services
Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    Friend WithEvents SqlSelectCommand1 As
System.Data.SqlClient.SqlCommand
    Friend WithEvents SqlInsertCommand1 As
System.Data.SqlClient.SqlCommand
    Friend WithEvents SqlConnection1 As
System.Data.SqlClient.SqlConnection
    Friend WithEvents SqlDataAdapter1 As
System.Data.SqlClient.SqlDataAdapter
    Friend WithEvents DataSet1 As MailServ.DataSet1
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
        Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
        Me.SqlInsertCommand1 = New System.Data.SqlClient.SqlCommand()
        Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
        Me.DataSet1 = New MailServ.DataSet1()
        CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).BeginInit()
```

```
        'SqlSelectCommand1
        '
        Me.SqlSelectCommand1.CommandText = "SELECT * FROM USERS WHERE
(LOGIN = @LOG) AND (PASSWORD = @PASS)"
        Me.SqlSelectCommand1.Connection = Me.SqlConnection1
        Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.VarChar,
10, "LOGIN"))
        Me.SqlSelectCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PASS",
System.Data.SqlDbType.VarChar, 10, "PASSWORD"))
        '
        'SqlConnection1
        '
        Me.SqlConnection1.ConnectionString = "data source=(LOCAL);initial
catalog=MAILER;persist security info=False;user id=sa & _
        workstation id=BHV-9IEMVL4EOER;packet size=4096"
        '
        'SqlInsertCommand1
        '
        Me.SqlInsertCommand1.CommandText = "INSERT INTO USERS (LOGIN,
PASSWORD, USERNAME, MAILSERVER) VALUES (@LOGIN, @PASSWORD, @USERNAME,
@MAILSERVER)"
        Me.SqlInsertCommand1.Connection = Me.SqlConnection1
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@LOGIN",
System.Data.SqlDbType.VarChar, 10, "LOGIN"))
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@PASSWORD",
System.Data.SqlDbType.VarChar, 10, "PASSWORD"))
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@USERNAME",
System.Data.SqlDbType.VarChar, 50, "USERNAME"))
        Me.SqlInsertCommand1.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@MAILSERVER",
System.Data.SqlDbType.VarChar, 50, "MAILSERVER"))
        '
        'SqlDataAdapter1
        '
        Me.SqlDataAdapter1.InsertCommand = Me.SqlInsertCommand1
        Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
        Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "USERS", New
```

```
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("USERID", "USERID"), New
System.Data.Common.DataColumnMapping("LOGIN", "LOGIN"), New
System.Data.Common.DataColumnMapping("PASSWORD", "PASSWORD"), New
System.Data.Common.DataColumnMapping("USERNAME", "USERNAME"), New
System.Data.Common.DataColumnMapping("MAILSERVER", "MAILSERVER")})})
        '
        'DataSet1
        '
        Me.DataSet1.DataSetName = "DataSet1"
        Me.DataSet1.Locale = New System.Globalization.CultureInfo
("ru-RU")
        Me.DataSet1.Namespace = "http://www.tempuri.org/DataSet1.xsd"
        CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()

    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region
    <WebMethod()> Public Function LogExist(ByVal LOG As String)
As Boolean
        Dim DS As New DataSet()
        Dim t As Boolean
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select count(LOGIN) as CID from
USERS where (LOGIN = @LOG)"
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@LOG", System.Data.SqlDbType.Char, 30))
        DataCommand.SelectCommand.Parameters("@LOG").Value = LOG
```

```
        DataCommand.Fill(DS)
        If DS.Tables(0).Rows(0).Item("CID") = 0 Then
            t = False
        Else
            t = True
        End If
        DS.Dispose()
        LogExist = t
    End Function


    <WebMethod()> Public Function NewUser(ByVal LOG As String, ByVal PASS
As String, ByVal USERNAME As String, ByVal MAILSERV As String) As Boolean
        SqlInsertCommand1.Parameters("LOGIN").Value = LOG
        SqlInsertCommand1.Parameters("PASSWORD").Value = PASS
        SqlInsertCommand1.Parameters("USERNAME").Value = USERNAME
        SqlInsertCommand1.Parameters("MAILSERVER").Value = MAILSERV
        SqlConnection1.Open()
        SqlInsertCommand1.ExecuteNonQuery()
        SqlConnection1.Close()
        NewUser = True
    End Function
    <WebMethod()> Public Function LogOnUser(ByVal LOG As String, ByVal
PASS As String) As Integer
        Dim DS As New DataSet()
        SqlSelectCommand1.Parameters("LOG").Value = LOG
        SqlSelectCommand1.Parameters("PASS").Value = PASS
        SqlDataAdapter1.Fill(DS)
        If DS.Tables(0).Rows(0).Item("USERID") Is Nothing Then
            LogOnUser = -1
        Else
            LogOnUser = DS.Tables(0).Rows(0).Item("USERID")
        End If
    End Function
    <WebMethod()> Public Function UserMail(ByVal USERID As Integer, ByVal
MailBody As String, ByVal MailSubject As String, ByVal MailTo As String)
As Boolean
        Dim DS As New DataSet()
        Dim email As New Web.Mail.MailMessage()
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        'Dim smtp As New System.Web.Mail.SmtpMail()
```

```
        Dim SelectCommand As String = "select * from USERS where
(USERID = @USERID)"
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
SqlConnection1)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@USERID", System.Data.SqlDbType.Int, 4))
        DataCommand.SelectCommand.Parameters("@USERID").Value = USERID
        DataCommand.Fill(DS)
        email.To = MailTo
        email.From = DS.Tables(0).Rows(0).Item("MAILSERVER").Value
        email.Subject = MailSubject
        email.Body = MailBody
        Web.Mail.SmtpMail.Send(email)
        UserMail = True
    End Function
End Class
```

Понятно, что процедура разработки клиентской части приложения принципиально ничем не будет отличаться от примеров, приведенных в предыдущих главах, поэтому мы можем спокойно завершить этот пример.

Послесловие

В этой книге мы рассмотрели самые различные методы применения Web-сервисов, построенных на основе технологии Microsoft .NET. Вне всякого сомнения, ваши сервисы будут больше по размеру и более функциональны, однако следует помнить, что основная ценность этих сервисов отнюдь не в простоте разработки.

Самая ценная часть сервисов — XML. Дело в том, что использование сервисов позволяет самым различным системам обмениваться информацией на одном языке, в едином формате. При правильном использовании эта возможность открывает для внешнего пользователя и удаленных работников многие бизнес-процессы корпораций, позволяет еще больше интегрировать различные ресурсы между собой. Web-сервисы просто облегчают общение различных систем друг с другом, но и это уже немало.

Однако Web-сервисы нельзя назвать панацеей, и нет смысла использовать их для решения самых различных классов задач. Следует осознавать, что сервисы — всего лишь одна технология, для которой есть своя четко очерченная ниша. Но эта ниша не так уж и мала, и места для работы там еще достаточно. Автор очень надеется, что изложенная информация поможет разработчику двигаться дальше в своем самообучении.

Удачных Вам проектов. Встретимся в Сети.

Игорь Шапошников