bhv®
www.bhv.ru
www.bhv.kiev.ua

Игорь Шапошников

ASP.NET

Самоучитель

Visual Studio .NET

Разработка Web-приложений

Взаимодействие с базами данных

XML Web-сервисы

Мощное и гибкое средство создания многофункциональных Web-проектов

Игорь Шапошников

САМОУЧИТЕЛЬ ASP.NET

Санкт-Петербург

«БХВ-Петербург»

2002

УДК 681.3.06

Шапошников И. В.
Самоучитель ASP.NET. — СПб.: БХВ-Петербург, 2002. — 368 с.: ил.

ISBN 5-94157-171-2

Книга представляет собой руководство по созданию мощных полнофункциональных сайтов на основе технологии ASP (Active Server Pages). В книге рассмотрены вопросы установки средств разработки ASP.NET и настройки Web-сервера IIS. Описан процесс создания ASP-приложений, начиная самыми простыми и заканчивая многофункциональными ASP-сценариями. Отдельные главы книги посвящены обзору языков Visual Basic, XML и SOAP. Читатель познакомится с основами создания интернет-сервисов на основе ASP, достаточно новым направлением с многообещающими перспективами.

Для Web-программистов

УДК 681.3.06

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Владислав Борисов*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Татьяна Звертановская*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.03.02.
Формат 70×100 $^{1}/_{16}$. Печать офсетная. Усл. печ. л. 29,67.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

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

ISBN 5-94157-171-2

Содержание

Даниленко Ольге

"Если б не было тебя, я б шел по миру, как слепой,
В гуле сотен чужих голосов узнать пытаясь голос твой
И звук твоих шагов"

Благодарности

Огромное спасибо Вам за то, что Вы держите сейчас книгу в руках и читаете ее. Все книги пишутся именно для их, читателей. На самом деле, мало есть на свете вещей стимулирующих так, как отзывы читателей о книге.

Но автор никогда не пишет книгу один. Всегда есть люди, чей вклад в книгу является не менее весомым, чем авторский. Я, конечно, говорю о редакторском коллективе издательства "БХВ-Петербург". Спасибо нашему главному редактору Кондуковой Екатерине, отдельное спасибо принимающему редактору. На самом деле, упоминать надо всю группу подготовки издания, которая в очередной раз сделала работу просто замечательно.

Спасибо также всем моим друзьям в Сети: Буке, Умке, Лайке, Sweet-dream, Панку, Финисту и многим-многим другим. Ребята, я всех вас помню и люблю.

Огромное спасибо моей Ольге, которая всегда помогала и поддерживала меня во время работы. Спасибо тебе, милая, за терпение и понимание.

Введение

О чем же эта книга? Что такое ASP.NET? По сути дела, в книге рассказывается о создании мощных полнофункциональных сайтов. 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, с использованием информации, получаемой из каких-либо баз данных. Естественно, для создания таких страниц одного языка HTML (Hypertext Markup Language, язык описания гипертекстовых документов), который все-таки является основой Web-страниц, явно недостаточно. Даже использование технологии динамического HTML, опирающегося на сценарии, выполняющиеся на стороне пользователя, не поможет в создании подобных сайтов.

Дело в том, 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 работы с запросами удаленного пользователя приходится работать с содержимым базы данных, то здесь никак не обойтись без серверных приложений, так как база данных, естественно, может функционировать только на сервере.

Для создания приложений, действующих на стороне Web-сервера, есть достаточно много приемлемых технологий. Но технология ASP (Active Server Pages), о которой и рассказывается в этой книге, стоит особняком в этом ряду. 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, пожалуй, выбор будет очевидным. Следует использовать связку IIS+ASP. Тем более что ASP-приложения отлично интегрируются с базами данных от Microsoft.

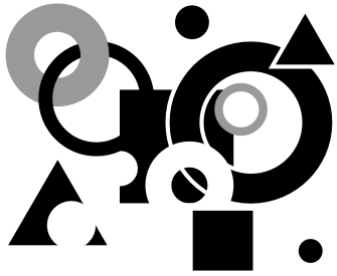
Технология ASP известна достаточно давно. В этой книге рассматривается ее последняя на данный момент модификация — ASP.NET. Разработка приложений ASP.NET производится с помощью пакета Visual Studio .NET.

Структура книги достаточно проста. В первой главе мы рассмотрим вопросы установки средств разработки программ ASP.NET и настройки web-сервера IIS. Во второй главе будет приведен обзор языка Visual Basic, на котором и разрабатываются приложения ASP. Третья глава является, пожалуй, основной в этой книге. Именно в ней будет рассказано о создании ASP-приложений. Начнем мы с самых простых вариантов и закончим созданием многофункциональных ASP-сценариев. В четвертой главе будут освещены вопросы связывания Web-приложений с системами управления базами данных. Затем будут рассмотрены языки XML и SOAP, с которыми очень тесно связана технология Microsoft .NET. А в последней главе мы рассмотрим создание интернет-сервисов на основе ASP. Это достаточно новое направление, но уже с многообещающими перспективами. Собственно говоря, сама концепция ".NET" строится на создании сервисов, которые будут доступны для работы пользователя с любой компьютерной платформой.

Разумеется, сами сервисы будут функционировать на Windows-серверах, но удаленные пользователи могут использовать их вне зависимости от того, на какой платформе они работают. Для связи клиентской системы и .NET-сервиса обычно используется язык XML с его потрясающей адаптивностью и расширяемостью. Естественно, для того чтобы строить подобные сервисы, следует знать основные правила работы с языком XML. Именно поэтому целая глава будет посвящена рассмотрению языка XML.

А теперь перейдем к работе.

Глава 1



Инсталляция и настройка

Среда разработки

Создание приложений ASP.NET производится при помощи средства разработки Visual Studio .NET. Это огромнейший пакет средств и технологий для создания самых разнообразных приложений. У меня просто не хватает слов, чтобы вкратце описать все его возможности. Разработчик может использовать один из трех языков программирования, входящих в состав пакета: Visual Basic .NET, Visual C++ .NET и Visual C# .NET. Все они объединены общим интерфейсом, поэтому разработчик для реализации каждой задачи может выбирать тот язык, который максимально хорошо подойдет к данной ситуации. При этом внешний вид среды разработки, ее основные функции и механизмы не будут изменяться, и в любом случае, разработчик будет действовать в рамках знакомой среды программирования. Подобное единообразие значит очень много, так как удобство разработки является одним из ключевых факторов быстроты создания и качества программного продукта.

Дистрибутив Visual Studio .NET занимает семь дисков CD-ROM. Естественно, на жесткий диск переносится не все их содержимое, но так или иначе, придется выделить не менее 3 Гбайт на основном жестком диске и около 300 Мбайт на системном логическом диске.

Как и все системное программное обеспечение, производимое корпорацией Microsoft, среда разработки Visual Studio .NET является достаточно требовательным приложением. В качестве минимальных системных требований, помимо уже упоминавшегося необходимого объема свободного места на дисках системы, необходимо наличие процессора не ниже уровня Pentium II — 450 и оперативная память от 96 Мбайт. На практике, для более приемлемой скорости работы, следует нарастить объем оперативной памяти до 128 Мбайт. Да и более мощный процессор явно не помешает.

Также стоит обратить внимание на используемую операционную систему. Рекомендуется использовать операционные системы семейства Windows, основанные на технологии NT. Это сама Windows NT, Windows 2000 (желательно оригинальная версия) или Windows XP. Однако последняя в настоящий момент имеет не слишком долгую историю эксплуатации, поэтому, теоретически, может быть несколько нестабильна. На первые две перечисленные системы надо установить самые новые пакеты обновлений. Иначе говоря, машину, на которой будет производиться разработка, необходимо поднять на максимально высокий уровень.

Впрочем, на пятом диске дистрибутива Visual Studio .NET находятся все дополнительные компоненты, необходимые для функционирования среды разработки и среды выполнения программ. И первым действием программы установки будет проверка наличия необходимых дополнительных компонентов в составе операционной системы. Кстати, одним из таких дополнительных компонентов является набор серверных расширений FrontPage, который отказывается корректно устанавливаться на русскую версию Windows 2000. Поэтому я и упоминал, что следует использовать именно оригинальную версию. Впрочем, данный компонент не является критичным для функционирования Visual Studio .NET. И уж тем более, он не нужен для разработки ASP-приложений. Данные расширения позволяют обеспечивать работу некоторых активных элементов, которые FrontPage позволяет встраивать в разрабатываемые HTML-документы. Мы же будем сами разрабатывать подобные активные элементы, поэтому наличие специализированных расширений нам абсолютно не нужно.

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 Studio .NET. В ее состав входят языки Visual Basic .NET, Visual C++ .NET и Visual C# .NET. Приложения ASP можно разрабатывать на любом из этих языков, но традиционно они разрабатывались при помощи языка Visual Basic, и мы тоже будем использовать именно этот язык.

Но вернемся к рассмотрению концепции MSIL. Для того чтобы разработчик мог выбирать язык программирования в зависимости от своих предпочтений или поставленной задачи, необходимо нечто большее, чем единая среда разработки. Необходима общая основа для всех приложений. И она была создана. Как мы помним, при установке Visual Studio .NET были инсталлированы дополнительные элементы. Одним из таких элементов была среда .NET Framework. Этот компонент можно рассматривать как некоторое дополнение к операционной системе, которое несколько расширяет ее возможности. Вместе с .NET Framework технология разработки приложений разбивается на два этапа. Сначала разработчик пишет исходный код на выбранном языке программирования. После этого компилятор Visual

Studio .NET переводит его на язык MSIL. А на втором этапе приложение, записанное на языке MSIL при помощи виртуальной машины CLR (Common Language Runtime), переводится в код, присущий той компьютерной платформе, на которой приложение будет функционировать. При этом приложение получает возможность использовать всю функциональность, описанную в Microsoft .NET Framework.

Естественно, подобная структура компиляции полностью копирует идеологию Java. Как мы помним, Java-компиляторы переводили исходный текст программы не в машинные коды, а в некий байт-код, который затем компилировался виртуальной Java-машиной. Конечно виртуальную Java-машину надо было писать для каждой специфичной платформы. Точно также обстоит дело и с Microsoft .NET Framework. Точнее, с ее основной частью — так называемым *"компилятором по требованию"*, JIT-компилятором, чаще называемым *джиттером*, который и отвечает за перевод кода MSIL в код, специфичный для конкретной платформы. На момент написания книги джиттер был разработан только для операционной системы Microsoft Windows 2000 и Windows XP.

Администрирование IIS

Но вернемся к ASP. Данная технология позволяет создавать приложения, обеспечивающие интерактивность и иные дополнительные возможности для сайтов. ASP-приложения действуют не на стороне удаленного пользователя, а на самом WWW-сервере. Поддержка технологии ASP встроена в WWW-сервер IIS (Internet Information Services), который входит в состав дистрибутива Windows NT 4 и Windows 2000. Впрочем, на диске с дополнительными компонентами, входящими в состав дистрибутива Visual Studio.NET, также находится этот компонент, и если он не установлен в операционной системе, то инсталлятор Visual Studio.NET попробует установить IIS самостоятельно. Но все-таки стоит установить его с самого начала еще перед установкой Visual Studio.NET, чтобы правильно его настроить.

Для установки IIS следует открыть окно **Control Panel** (Панель управления) операционной системы и запустить утилиту **Add/Remove Programs** (Установка и удаление программ). Нам потребуется утилита **Add/Remove Windows Components** (Установка компонентов Windows). Эта утилита активизирует диалоговое окно **Windows Components Wizard** (Мастер установки компонентов Windows), показанное на рис. 1.1.

Компонент Internet Information Services (IIS) состоит из нескольких разнородных частей. Список их можно увидеть, если выделить в диалоговом окне **Windows Component Wizard** (Мастер установки компонентов Windows) строку **Internet Information Services (IIS)** и нажать кнопку **Details** (Состав). В результате будет отображено диалоговое окно **Internet Information Services (IIS)**, показанное на рис. 1.2.

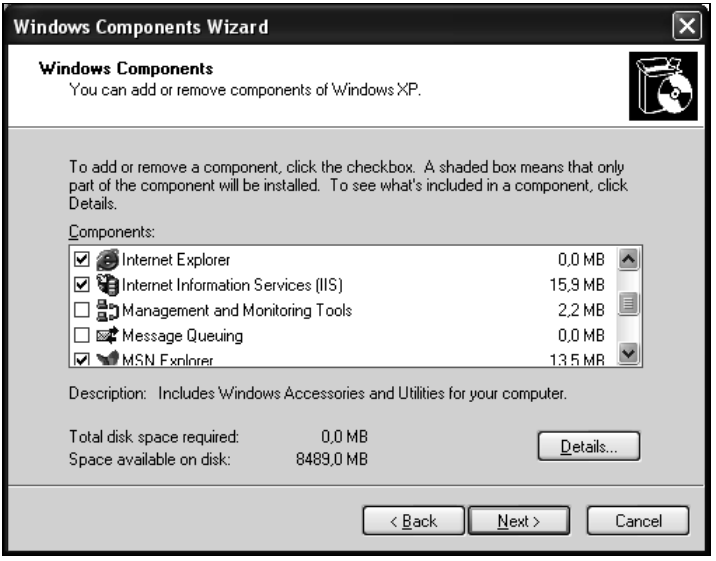


Рис. 1.1. Диалоговое окно **Windows Components Wizard**

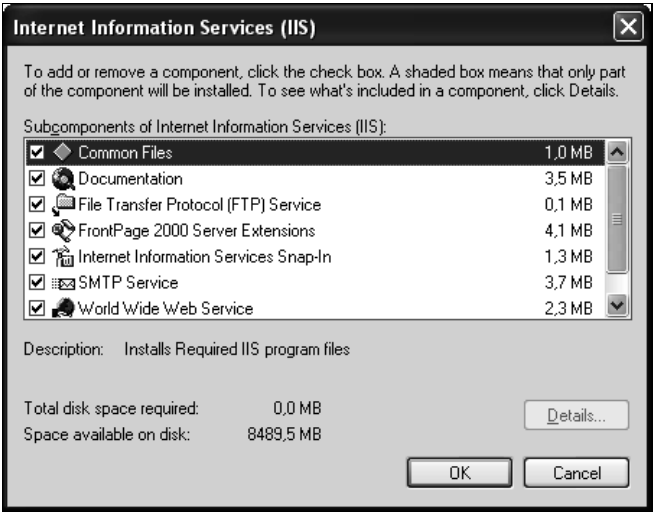


Рис. 1.2. Диалоговое окно **Internet Information Services (IIS)**

В диалоговом окне **Internet Information Services (IIS)** мы можем указать, какие именно функциональные части элемента IIS требуется установить в операционную систему. Для создания полноценного сервера нам потребуются следующие компоненты:

- **Common Files** (Общие файлы). Общие файлы, необходимые для функционирования практически всех элементов IIS;

- **File Transfer Protocol (FTP) Server** (Служба FTP). Сервер приема и передачи файлов по протоколу FTP;
- **Internet Information Services Snap-In** (Оснастка IIS). Система администрирования сервера при помощи утилиты MMC (Microsoft Management Console);
- **Personal Web Manager** (Управление личным Web-сервером). Графический интерфейс администрирования сервера;
- **SMTP service** (Служба SMTP). Служба отправки исходящих сообщений электронной почты по протоколу SMTP;
- **World Wide Web Server** (Служба WWW). Собственно, сам WWW-сервер, который и поддерживает функционирование сайтов.

Для того чтобы установить все эти элементы, необходимо поставить флажки в независимых переключателях, находящихся рядом с наименованиями компонентов, и нажать кнопку **OK**. После этого компонент IIS будет автоматически установлен. Использование остальных элементов не является критичным для создания полноценных сайтов с использованием ASP-приложений, поэтому их установка необязательна.

В процессе установки IIS будет создана структура каталогов, в которых размещается содержимое серверов. По умолчанию, корневой каталог элемента носит наименование `Inetpub`. В каталоге `FTPROOT` размещается содержимое FTP-сервера, каталог `MAILROOT` предназначен для почтового SMTP-сервера, а в каталоге `WWWROOT` находится содержимое WWW-сервера.

Нас будет интересовать администрирование именно WWW-сервера. При установке, автоматически создается некий локальный сайт, в терминологии Microsoft часто называемый Web-узлом. Доступ к нему можно получить, если в строке **Адрес** браузера набрать адрес **http://localhost**.

Необходимо сделать некоторое техническое отступление. В подавляющем большинстве, Web-сайты обладают доменным именем, как, например, **www.bhv.ru**. В том случае, если сайт не имеет подобного имени, доступ к нему осуществляется по IP-адресу того сервера, на котором он установлен. Однако, использование IP-адресов вместо доменных имен явно невыгодно, так как запомнить IP-адрес достаточно трудно, чего нельзя сказать о правильно подобранном доменном имени. Сопоставление доменных имен и IP-адресов серверов, на которых размещены сайты с этими именами производится DNS-серверами при помощи DNS-таблиц. Служба DNS (Domain Name Service) как раз и существует для того, чтобы пользователь WWW мог набрать обычное символьное доменное имя сайта, вместо IP-адреса сервера.

IP-адрес, как известно, представляет собой комбинацию из четырех положительных целых чисел, находящихся в промежутке от 0 до 255, и разделенных точками. При этом все адреса, начинающиеся с числа 127

(обозначается, как 127.*.*.*), являются локальными. Эти адреса нельзя назначить серверу, реально находящемуся в Интернете. Они могут быть назначены только локальным системам. Проще говоря, если необходимо на локальной машине разработать сервер, используя при этом некое доменное имя, например, **www.mysite.com**, то надо сопоставить это имя одному из таких локальных IP-адресов. Казалось бы, для этого необходимо запускать на локальной машине еще и DNS-сервер, чтобы браузер мог в поисках сервера, на котором содержится искомый сайт, обратиться к этому серверу без выхода в Интернет, но на самом деле все проще. Все Windows-системы перед тем, как обратиться к DNS-серверу, проверяют наличие файла hosts, находящегося в системном каталоге Windows. Это текстовый файл, в котором указывается список доменных имен и соответствующих им сайтов. Поэтому, если мы хотим, чтобы на нашем сервере действовал локальный сайт с доменным именем **www.mysite.com**, необходимо в этот файл добавить следующую строку:

```
127.0.0.1 www.mysite.com
```

Теперь, для того чтобы связаться с этим сайтом, браузер не будет обращаться к DNS-серверам, а всего лишь воспользуется файлом hosts. Впрочем, справедливости ради надо сказать, что сайт с доменным именем `localhost` система всегда ищет в корневом каталоге функционирующего WWW-сервера даже без обращения к файлу hosts.

Мы уже говорили, что основной сайт размещается в каталоге Inetpub/WWWROOT. Но в этом каталоге находятся не только HTML-документы и скрипты. Любой сайт, это еще и структура дополнительных каталогов. Всегда удобнее структурировать данные по различным каталогам. Отдельно держать графические файлы, отдельно информацию для различных разделов, и так далее. Поэтому URL графического файла с наименованием picture1.gif может выглядеть следующим образом:

http://www.mysite.com/image/picture1.gif

При этом подразумевается, что графический файл picture1.gif находится в каталоге images, который в свою очередь размещен в корневом каталоге WWW-сервера. На самом деле, в URL не обязательно указывается точное наименование каталога. В нем указывается наименование так называемого "виртуального каталога", чье наименование не обязательно будет совпадать с именем настоящего физического каталога. Так, каталог, в котором находится графический файл picture1.gif, мог бы на самом деле называться img, а не image, как указано в URL. То есть, виртуальные каталоги это всего лишь названия реальных, физических каталогов.

Процедура создания виртуальных каталогов достаточно проста. В группе органов управления, располагающихся в **Control Panel** (Панель управления), существует инструмент настройки основных сервисов системы —

Administrative Tools (Администрирование). Нас будет интересовать один из его компонентов — **Internet Services Manager** (Управление службами Интернета). Двойной щелчок по иконке этого компонента отображает диалоговое окно **Internet Information Services** (Информационные службы Интернета), предназначенное для настройки IIS, внешний вид которого показан на рис. 1.3.

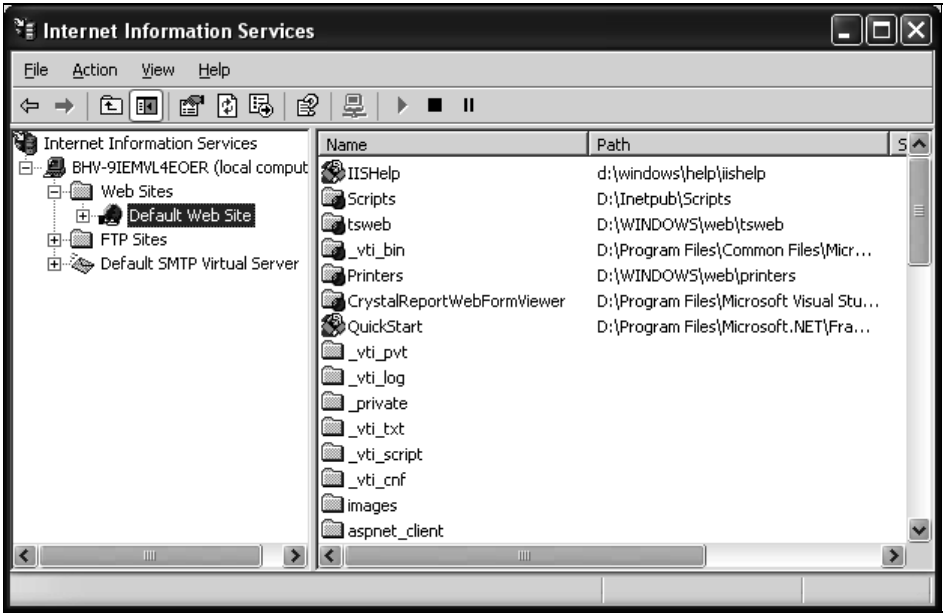


Рис. 1.3. Диалоговое окно настройки **Internet Information Services**

Для того чтобы создать новый виртуальный каталог, необходимо выделить строку с наименованием действующего Web-сайта, чаще всего она носит наименование **Default Web Site** (Веб-сайт по умолчанию), и нажать кнопку **Action** (Действие). В появившемся меню нам следует выбрать пункт **New | Virtual Directory** (Создать | Виртуальный каталог). После этого будет запущен мастер создания виртуальных каталогов. На первом этапе работы мастера будет отображено диалоговое окно **Virtual Directory Creation Wizard** (Мастер создания виртуальных каталогов) с полем текстового ввода, куда необходимо будет внести наименование виртуального каталога, как это показано на рис. 1.4.

После того, как мы ввели в поле **Alias** (Псевдоним) наименование виртуального каталога, следует нажать кнопку **Next** (Далее). Будет отображено диалоговое окно второго шага мастера установки виртуальных каталогов, показанное на рис. 1.5.

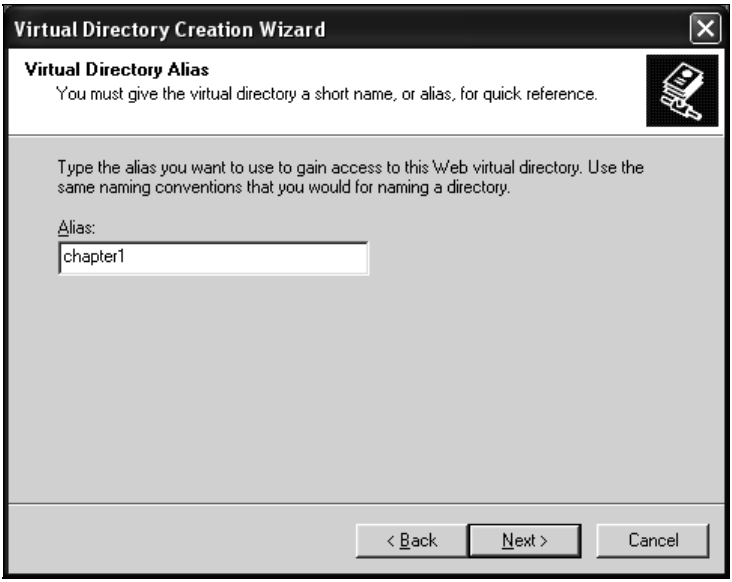


Рис. 1.4. Диалоговое окно настройки **Virtual Directory Creation Wizard**, отображаемое на первом этапе работы мастера

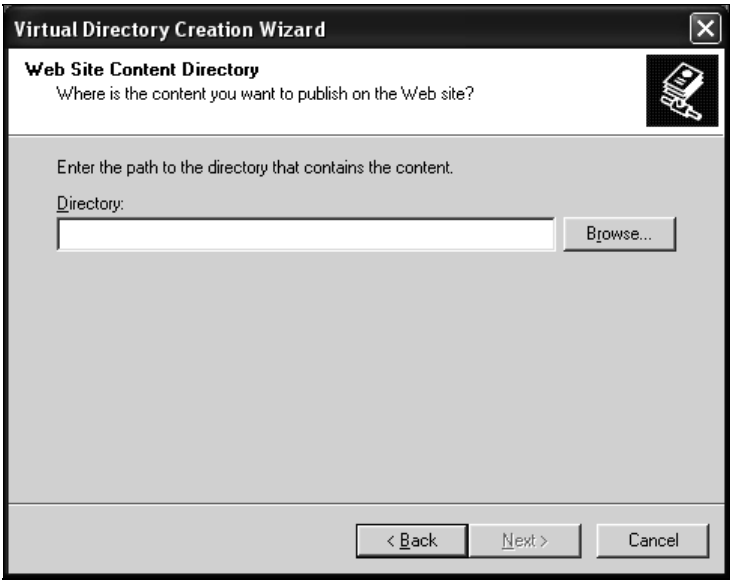


Рис. 1.5. Диалоговое окно настройки **Virtual Directory Creation Wizard**, отображаемое на втором этапе работы мастера

Полный путь к физическому каталогу, который будет связан с создаваемым виртуальным каталогом, следует внести в поле ввода **Directory** (Каталог),

или воспользоваться кнопкой **Browse** (Обзор). После указания пути к физическому каталогу еще раз нажать кнопку **Next** (Далее). Третий этап работы мастера позволяет установить назначение виртуального каталога. Как известно, мы можем гибко устанавливать правила работы с виртуальными каталогами. Можно разрешать чтение файлов из них, запись в файлы данного каталога, исполнение скриптов или приложений, просмотр содержимого каталога. Также возможно использование любой комбинации из этих разрешений. Итак, после того, как мы, указав путь к физическому каталогу, нажали кнопку **Next** (Далее), будет отображено диалоговое окно третьего этапа работы мастера создания виртуальных каталогов, показанное на рис. 1.6.

Рис. 1.6. Диалоговое окно настройки **Virtual Directory Creation Wizard**, отображаемое на третьем этапе работы мастера

Рассмотрим возможные установки разрешений для виртуального каталога. Установка или отключение того или иного разрешения производится при помощи независимых переключателей, связанных с описаниями разрешений. Всего предусмотрено пять разрешений.

- **Read** (Чтение). Разрешение на чтение файлов из создаваемого виртуального каталога.
- **Run scripts (such as ASP)** (Запуск сценариев (например ASP)). Позволяет удаленному пользователю запускать на выполнение скрипты из данного виртуального каталога.
- **Execute (such as ISAPI applications or CGI)** (Выполнение (например приложений ISAPI и CGI)). Разрешение выполнять Web-приложения, осно-

ванные на технологии ISAPI или CGI, находящиеся в данном виртуальном каталоге.

- **Write** (Запись). Разрешение на запись в файлы, размещенные в данном виртуальном каталоге.
- **Browse** (Обзор). Означает, что удаленный пользователь может просматривать содержимое данного каталога.

Так как подключение или отключение любого разрешения производится при помощи независимых переключателей, мы можем использовать любую комбинацию из предлагаемых вариантов. Однако не следует без особой нужды подключать все разрешения к создаваемым виртуальным каталогам, так как это будет являться нарушением элементарных правил безопасности WWW-сервера. Следует использовать для каждого каталога минимально необходимый набор разрешений. То есть, если каталог предназначается только для хранения графических изображений, применяемых в оформлении Web-страниц, входящих в состав сайта, нам потребуется установить лишь разрешение на чтение. Все остальные разрешения нам в данном случае не нужны.

Но основная настройка WWW-сервера производится при помощи его диалогового окна настройки свойств. Для того чтобы получить возможность работать с параметрами сервера, необходимо в диалоговом окне **Internet Information Services** (Информационные службы Интернета) выделить строку, обозначающую Web-сервер, и щелкнуть правой кнопкой мыши. В появившемся контекстном меню следует выбрать команду **Properties** (Свойства). Результатом выполнения этой команды меню будет отображение диалогового окна **Default Web Site Properties** (Свойства веб-узла по умолчанию) с открытой вкладкой **Web Site** (веб-узел), как это показано на рис. 1.7.

Это диалоговое окно позволяет управлять практически всеми свойствами WWW-сервера. Но для наших целей все они не понадобятся. Разберемся лишь с основными параметрами. Начнем мы с вкладки **Web Site** (Веб-узел). Органы управления, собранные в группе **Web Site Identification** (Идентификация веб-узла) идентифицируют сайт, который мы разрабатываем. Общее наименование сайта (но не его доменное имя) указывается в поле **Description** (Описание). В текстовых полях **IP Address** (IP-адрес) и **TCP Port** (TCP-порт) указываются IP-адрес и порт, на которых будет функционировать WWW-сервер. В поле ввода **Connection Timeout** (Время ожидания) мы указываем время тайм-аута, в течение которого сервер может ожидать ответа от удаленного пользователя на посланные запросы. Если по истечении этого времени браузер удаленного пользователя не ответил, WWW-сервер будет считать, что удаленный пользователь сбросил соединение, и разорвет его со своей стороны.

В том случае, если необходимо вести log-файлы, в которых будут записываться сообщения обо всех соединениях удаленных пользователей с WWW-сервером, следует поставить флажок в переключателе **Enable Logging** (Вести

журнал). Правильная установка службы log-файлов может сделать очень многое. С ее помощью можно узнавать IP-адреса, пользователей, их путь по сайту, страницы, с которых они пришли на сайт, время и дату их запросов к серверу и многое другое. Все зависит от того, как настроить log-файлы. Для тонкой настройки службы ведения log-файлов следует нажать кнопку **Properties** (Свойства). При этом будет показано диалоговое окно **Extended Logging Properties** (Раширенные свойства ведения журнала).

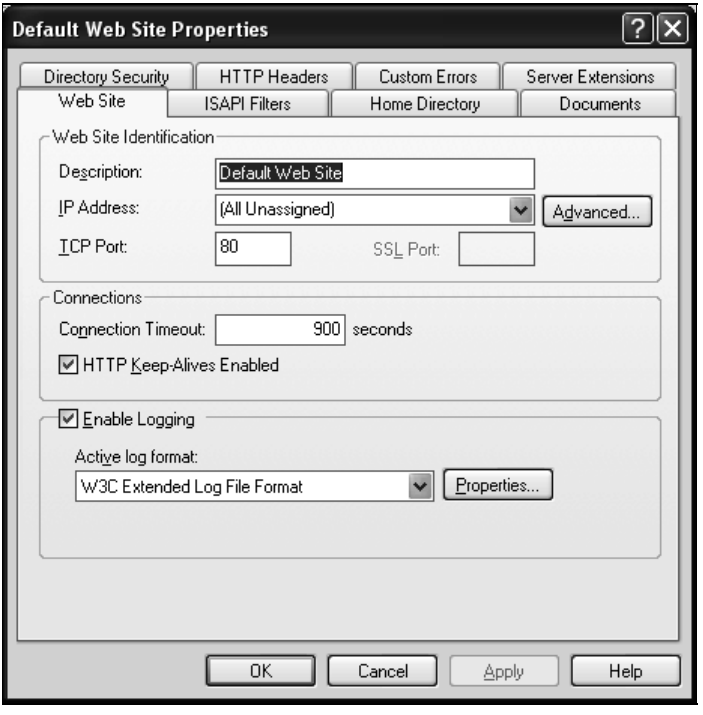


Рис. 1.7. Вкладка **Web Site** диалогового окна **Default Web Site Properties**

Это диалоговое окно состоит из двух вкладок — **General Properties** (Общие свойства) и **Extended Properties** (Расширенные свойства). Вкладка **General Properties** (Общие свойства) содержит органы управления, регулирующие свойства самих файлов, в которых будут содержаться сообщения о подключениях. А формат сообщений о подключениях удаленных пользователей и их действиях регулируется на вкладке **Extended Properties** (Расширенные свойства).

Когда пользователь в качестве URL указывает только доменное имя сайта, WWW-сервер отсылает браузеру стартовую страницу. Обычно HTML-файл с данной страницей носит наименование **index.htm**. Но по умолчанию WWW-сервер IIS отображает страницу **default.htm**. Впрочем, наименование

файла, который будет посылаться удаленному пользователю, естественно, можно изменить. Для этого необходимо активировать вкладку **Documents** (Документы) диалогового окна **Default Web Site Properties** (Свойства веб-узла по умолчанию), показанную на рис. 1.8.

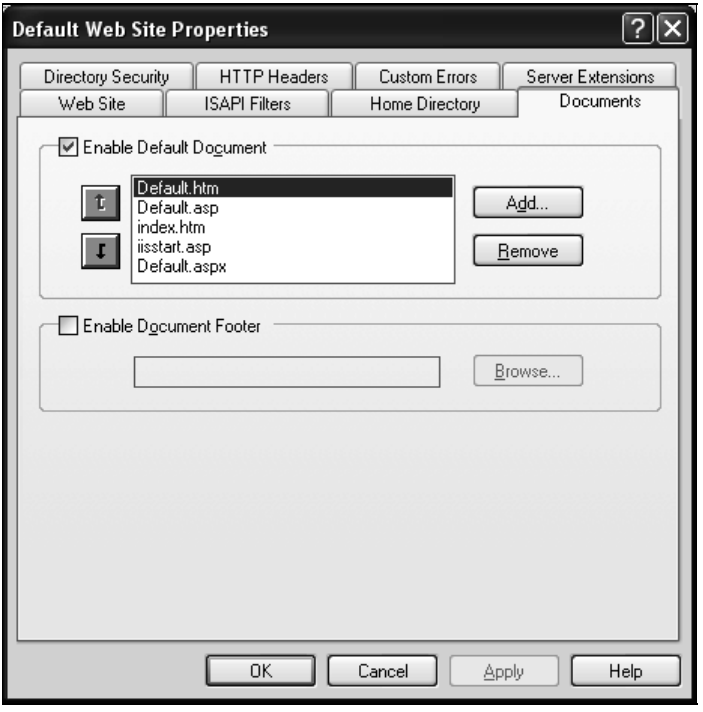


Рис. 1.8. Вкладка **Documents** диалогового окна **Default Web Site Properties**

Для того чтобы при указании удаленным пользователем только доменного имени сайта WWW-сервер выдавал браузеру какую-либо Web-страницу, необходимо поставить флажок в независимом переключателе **Enable Default Document** (Задать документ, используемый по умолчанию). При этом будет активизирован список, входящий в одноименную группу органов управления. В списке находятся наименования файлов, которые будет пытаться отыскать в корневом каталоге сайта WWW-сервер, чтобы обработать их и передать браузеру удаленного пользователя. При этом учитывается порядок расположения наименований файлов в данном списке. Сначала сервер будет пытаться найти файл, находящийся на первой позиции списка. Если этот файл не будет найден в корневом каталоге сайта, то сервер попытается отыскать следующий файл из списка, и так далее. На иллюстрации видно, что мы можем использовать не только чистые HTML-файлы, но и ASP-сценарии. Для изменения порядка следования наименований файлов в списке, слева от списка размещены две кнопки с изображением стрелок,

направленных вверх и вниз. Если необходимо добавить в список наименование еще одного файла, можно использовать кнопку **Add** (Добавить). А для удаления файла из списка необходимо воспользоваться кнопкой **Remove** (Удалить).

В качестве последнего штриха настройки сервера можно использовать страницы обработки ошибок. Как известно, WWW-сервер может обрабатывать ошибки, вызванные действиями удаленного пользователя или локальных сценариев и Web-приложений. Одной из наиболее распространенных ошибок является ошибка с кодом 404, когда пользователь запрашивает документ, которого в действительности нет на сервере. При возникновении какой-либо ошибки, сервер возвращает браузеру удаленного пользователя Web-страницу с описанием возникшей ситуации. Так вот, мы можем самостоятельно создавать подобные Web-страницы с сообщениями об ошибках. Для того чтобы сопоставить созданные Web-страницы с кодами возможных ошибок, следует воспользоваться вкладкой **Custom Errors** (Специальные ошибки) диалогового окна **Default Web Site Properties** (Свойства веб-узла по умолчанию), показанной на рис. 1.9.

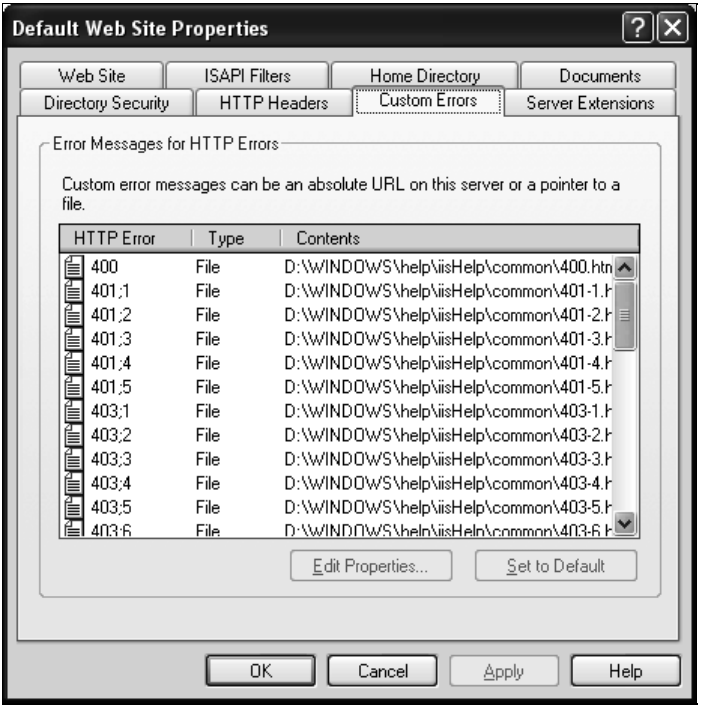


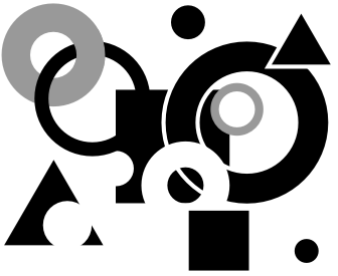
Рис. 1.9. Вкладка **Custom Errors** диалогового окна **Default Web Site Properties**

Основу рассматриваемой нами вкладки диалогового окна настройки свойств сервера составляет список, строки которого состоят из кода ошибки и имени

файла, который будет выводиться сервером при возникновении этой ошибки. Если мы хотим изменить стандартные предупреждения сервера, у нас есть два пути. Мы можем либо самостоятельно изменить те Web-страницы, которые указаны в этом списке, либо создать свои собственные Web-страницы с описанием возникшей критической ситуации и привязать их к кодам ошибок. Для привязки вновь созданных Web-страниц к ошибкам, следует выделить строку с кодом интересующей нас ошибки и нажать кнопку **Edit Properties** (Изменить свойства). После этого будет отображено дополнительное диалоговое окно с полем ввода для полного наименования HTML-файла, который будет посылаться в браузер удаленного пользователя при возникновении искомой ошибки.

Итак, мы настроили свой WWW-сервер. Теперь для того чтобы начать с ним работать в локальном режиме, необходимо его запустить. Обычно запуск WWW-сервера происходит автоматически при загрузке операционной системы. Проверить его функционирование можно в диалоговом окне контроля IIS. Контекстное меню, появляющееся при щелчке правой кнопкой мыши на строке **Default Web Site** (Веб-узел по умолчанию) в диалоговом окне **Internet information Services** (Информационные службы Интернет), содержит три пункта — **Start** (Пуск), **Stop** (Остановить) и **Pause** (Приостановить), которые позволяют принудительно запустить, отключить и приостановить работу сервера соответственно. Нет нужды напоминать, что ASP-сценарии обрабатываются сервером, поэтому при их тестировании необходимо, чтобы сервер был запущен.

Глава 2



Основы языка Visual Basic

Управляющая логика

Почти все программы имеют одинаковую структуру. В хорошей программе явно выделены области объявления переменных, функций и процедур, везде присутствуют одни и те же циклы и управляющие логические конструкции, всегда можно найти операторы сравнений. Разница лишь в ключевых словах, которые используются в том или ином языке программирования. В данной главе мы рассмотрим основные конструкции языка Visual Basic.

Почему мы рассматриваем именно Visual Basic? Дело в том, что именно на этом языке изначально писались сценарии для ASP, поэтому стоит рассматривать именно этот язык в качестве рабочей лошадки для ASP-сценариев. Если быть более точным, надо заметить, что при разработке сценариев ASP.NET мы используем язык Visual Basic .NET, который несколько отличается от канонического Visual Basic. Однако эти отличия не так уж и кардинальны, и там, где они будут важны для создания сценариев ASP.NET, мы упомянем об этих изменениях.

Естественно, мы не будем рассматривать язык Visual Basic .NET полностью. Дело в том, что для нужд создания сценариев ASP.NET нужны далеко не все его возможности. Но все, что действительно необходимо для ASP, мы рассмотрим.

Основа любой программы — это ее управляющая логика, т. е. те самые циклы, операторы сравнений и прочие конструкции, обуславливающие порядок действий программы.

Примечание

Яростные приверженцы объектно-ориентированной модели программирования могут заметить, что основой программы является именно объектная модель.

Но ради объективности надо признать, что это не так. Программы писались еще в те времена, когда об объектах никто и не задумывался. Все-таки управляющая логика это основа, альфа и омега любой программы.

Начнем мы с основного оператора управляющей логики — условного оператора. В языке Visual Basic он имеет предельно детализированный вид и объявляется следующим образом:

```
If условие [ Then ]
   [ блок кода  ]
[ ElseIf условие [ Then ]
   [ блок кода ] ]
[ Else
   [ блок кода ] ]
End If
```

Детально рассмотрим это формальное объявление оператора. Обязательным ключевым словом является, естественно, слово `If`. После него мы указываем либо логическое условие (или их комбинацию), либо переменную булева (логического) типа. После ключевого слова `Then` размещается блок кода, который будет выполнен в том случае, если логическое условие истинно. Или булева переменная имеет значение `True`. Блок, начинающийся с ключевого слова `ElseIf`, позволяет задать еще одно логическое условие, которое будет проверяться на истинность в том случае, когда первое условие, находящееся после ключевого слова `If`, является ложным. Как и в первом блоке оператора, код, выполняемый в случае истинности условия, размещается после ключевого слова `Then`. Если же оба логических условия оказываются ложными, мы можем использовать третий раздел оператора, начинающийся с ключевого слова `Else`. Сразу после него размещается блок кода, выполняемый в том случае, если все логические условия оказались ложными. Завершается весь условный оператор комбинацией ключевых слов `End If`. Необходимо также отметить, что второй и третий блоки оператора не являются обязательными.

Приведем пример использования условного оператора. Рассмотрим следующую конструкцию:

```
If i>5  Then
   i=i*2
 ElseIf i<0 Then
   i=i*(-1)
[ Else
   i=1
End If
```

Все операции в этом блоке кода производятся с переменной `i`. Если значение этой переменной больше пяти, то оно удваивается. Если оно меньше

нуля, то у этого значения принудительно меняется знак. Во всех остальных случаях, т. е. когда значение переменной находится в промежутке от нуля до пяти, это значение становится равным единице.

Но, как мы уже говорили, последние два блока оператора являются необязательными, поэтому без них можно обойтись, если необходимо просто проверить некоторое условие. В этом случае условный оператор будет выглядеть следующим образом:

```
If i>5 Then
   i=i*2
End If
```

Легко увидеть, что реализация условного оператора в Visual Basic .NET явно несколько утяжелена по сравнению с остальными распространенными языками программирования. Как мы увидим позже, подобная избыточность свойственна практически всем управляющим конструкциям Visual Basic .NET.

Логическим продолжением условного оператора является оператор выбора `Select`. Его формальное объявление выглядит следующим образом:

```
Select [ Case ] testexpression
   [ Case expressionlist
      [ блок кода ] ]
   [ Case Else
      [ блок кода ] ]
End Select
```

В теле этого оператора при помощи ключевого слова `Case` мы можем задавать логические условия и сопоставлять им блоки кода. Причем количество ключевых слов `Case` и связанных с ними логических условий ограничивается только требованиями программиста. Более того, в операторе присутствует ключевое слово `Else`. Блок кода, связанный с ним, выполняется в том случае, если ни одно из вышеперечисленных логических условий не было истинным.

Условия, связанные с ключевым словом `Case`, задаются несколько шире, нежели в случае с обычным условным оператором. Можно задавать целый интервал допустимых значений при помощи ключевого слова `To`. Пример такого условия показан в следующем фрагменте кода:

```
Select i
  Case 1 To 10 . . .
  . . . .
End Select
```

Оператор `Select` связывается с переменной `i`. А в первом условии проверяется, входит ли значение этой переменной в промежуток между единицей и десятью.

В логических условиях оператора `Select` мы можем использовать ключевое слово `Is`, применяемое в связке с операторами логического сравнения. Так, например, для того, чтобы выполнить некий блок кода в том случае, если значение основной переменной оператора больше пяти, следует использовать следующий код:

```
Select i
  Case Is>5 . . .
  . . . .
End Select
```

Завершается оператор, как видно из объявления, комбинацией ключевых слов `End Select`.

Приведем пример использования оператора `Select`. Рассмотрим следующий фрагмент кода:

```
Select i
  Case 1 To 5
     s="От единицы до пяти"
  Case 6, 7
     s="Или шесть или семь"
  Case Is>8
     s="Больше восьми"
  Case Else
     s="Судя по всему, меньше единицы"
End Select
```

Рассмотрим работу этого оператора. Все логические условия завязаны на переменную `i`. Первое условие проверяет вхождение значения основной переменной в интервал от единицы до пяти. В том случае, если это условие является истинным, то переменной `s`, очевидно имеющей строковый тип, присваивается значение `От единицы до пяти`. Второе условие проверяет, не равно ли значение основной переменной оператора шести или семи. Легко заметить, что в случае перечисления сравниваемых значений, они просто разделяются запятой. Соответственно, если это условие выполняется, то переменной `s` присваивается другое значение. Третье и последнее логическое условие будет истинным, если значение основной переменной оператора больше восьми. В том случае, если ни одно из перечисленных условий не выполняется, управление переходит к блоку кода, указанному после комбинации ключевых слов `Case Select`. Впрочем, этот блок не является обязательным.

Настало время рассмотрения циклов. Стандартный цикл `For`, связанный с переменной-счетчиком, объявляется следующим образом:

```
For counter = start To end [ Step step ]
   [ блок кода ]
```

```
[ Exit For ]
   [ блок кода ]
Next [ counter ]
```

Процесс выполнения такого цикла связан не с каким-либо логическим условием, а с переменной-счетчиком, на основе которой вычисляется количество проходов цикла. Простейший пример подобного цикла выглядит следующим образом:

```
For i=0 To 9
 my_array(i)=i*2
Next
```

В этом цикле мы каждому элементу массива присваиваем значение, в два раза больше его порядкового номера. Следует отметить, что нумерация массива начата с нуля, что нетипично для Visual Basic. Это нововведение Visual Basic .NET, которое мы рассмотрим в разделе, посвященном созданию массивов и работе с ними.

По умолчанию, после каждого прохода цикла значение переменной-счетчика увеличивается на единицу. Цикл будет выполняться до тех пор, пока значение переменной-счетчика не превысит числа, указанного после ключевого слова `To`. Однако могут возникать случаи, когда необходимо при прохождении цикла изменять значение счетчика не на единицу. В этом случае следует использовать необязательное ключевое слово `Step`, после которого указывается величина изменения счетчика с каждым проходом цикла, т. е. если мы хотим модифицировать цикл из вышеприведенного примера так, чтобы операции присваивания производились только с четными элементами массива, следует использовать следующий фрагмент кода:

```
For i=0 To 9 Step 2
  my_array(i)=i*2
Next
```

После ключевого слова `Next` мы можем указать наименование переменной-счетчика цикла. Это указание переменной не является обязательным, но может помочь в создании более понятного кода, когда применяются встроенные циклы. Посмотрите на такой пример:

```
For i=0 To 9
 For j=0 To 9
  my_array(i,j)=i*2+j
 Next j
Next i
```

В данном случае, дополнительное указание наименований счетчиков в конце цикла помогает читать код.

На основе приведенных примеров может сложиться впечатление, что переменная-счетчик для данного цикла может обладать только целочисленным

типом. На самом деле это не так. Для создания переменных-счетчиков может применяться любой численный тип, для которого поддерживаются арифметические операции сложения и вычитания, а также логические операции "больше" и "меньше".

Существует и еще одна вариация цикла `For`. Она применяется в тех случаях, когда необходимо обработать все элементы некоей группы. Это могут быть элементы коллекций, списков, различных стеков и прочих групп. Пример подобного цикла выглядит следующим образом:

```
For Each myitem In mylist
 myitem.qwant=1
Next
```

Как видно, структура подобного цикла достаточно прозрачна. Начало цикла формируется при помощи комбинации ключевых слов `For Each`, после которых указывается наименование элементов, по которым будет проходить цикл. Затем следует ключевое слово `In`, после которого мы указываем наименование списка, в который входят обрабатываемые элементы. В нашем случае мы производим обработку всех элементов `myitem`, которые являются вложенными частями объекта `mylist`. Естественно, в данном случае нам не нужна переменная-счетчик, для контроля над количеством проходов цикла. Легко заметить, что в приведенном примере мы успешно обошлись без нее. Завершается цикл при помощи знакомого нам уже ключевого слова `Next`.

Также часто применяются циклы, основанные на каких-либо логических условиях. То есть тело цикла будет выполняться до тех пор, пока заранее оговоренное условие не станет истинным или ложным (в зависимости от того, какую разновидность цикла использует разработчик). Подобные циклы объявляются следующим образом:

```
Do { While | Until } условие
   [ блок кода ]
[ Exit Do ]
   [ блок кода ]
Loop
```

Это объявление так называемого цикла с предусловием. То есть, перед выполнением очередного прогона цикла проверяется логическое условие, и в зависимости от результата проверки действие цикла может быть выполнено или нет. Ключевое слово `Until` применяется в тех случаях, когда цикл должен выполняться до тех пор, пока логическое условие является истинным. Примером подобного цикла является следующий фрагмент кода:

```
i=0
Do While i<10
 my_array(i)=i*2
```

```
 i+=1
Loop
```

Следует заметить, что в данном конкретном случае мы все же использовали переменную-счетчик, пусть и опосредованную. Но все зависит от логики программы. Часто подобные циклы применяются и без счетчиков.

Легко увидеть, что в приведенном примере цикл будет выполняться до тех пор, пока условие не станет ложным. Если же мы будем использовать ключевое слово `Until` вместо `While`, то цикл будет выполняться до тех пор, пока условие не станет истинным.

Мы уже говорили, что подобная модификация циклов называется циклом с предусловием, т. е. соответствие логического условия проверяется перед выполнением тела цикла. Таким образом, может возникнуть ситуация, когда еще перед самым первым проходом цикла окажется, что логическое условие не выполняется. В этом случае цикл не будет выполнен. Однако не стоит забывать о циклах с постусловием. В этой разновидности циклов логическое условие проверяется уже после выполнения тела цикла. Другими словами, при любом раскладе можно быть уверенным, что цикл будет пройден хотя бы один раз. Определяется подобный цикл следующим образом:

```
Do
   [ блок кода ]
[ Exit Do ]
   [ блок кода ]
Loop { While | Until } условие
```

Как видно, отличие от первого варианта цикла очень невелико — условие продолжения работы вместе с ключевым словом `Until` или `While` перенесено в конец цикла.

Итак, мы рассмотрели практически все управляющие конструкции языка Visual Basic .NET. Без внимания остались некоторые элементы управляющей логики, которые автор не счел нужным приводить. Одним из таких пропущенных элементов является ключевое слово `GoTo`, обеспечивающее прямую передачу управления какой-либо строке программы. Понятно, что этот пережиток прошлого должен быть как можно быстрее забыт, так как эта разновидность управления ходом выполнения программы устарела даже в момент появления парадигмы модульного программирования. Сейчас это просто дополнительный источник ошибок, которых и без того немало в наших программах, средство ухудшения читабельности кода и просто признак плохого стиля программирования.

Типы данных

Какими бы сложными объектами, массивами или коллекциями мы ни пользовались, следует помнить, что основой их являются единичные переменные.

Это основные кирпичики информации, атомы, из которых будет формироваться впоследствии сложная ткань огромных объектов. Конечно, в конце концов, вся информация в компьютерных системах представляется в численном виде, поэтому теоретически мы могли бы обойтись и одним-единственным типом данных. Причем раньше, во времена ассемблера, так и было. Но сейчас это уж слишком хлопотно. Следует точно знать, какой именно вид информации хранится в той или иной переменной, и в соответствии с этим знанием применять к ней те или иные функции обработки.

Visual Basic .NET все-таки обрел достаточно хорошую типизацию. Рассмотрим краткий перечень возможных типов переменных Visual Basic .NET. Типы перечислены в алфавитном порядке.

- `Boolean`. Логический тип. Для хранения переменной отводятся два байта. Переменная может принимать только два значения: `True` (Истина) и `False` (Ложь).
- `Byte`. Числовой тип. Для хранения переменной отводится один байт. Значение переменной может находиться в промежутке от нуля до двухсот пятидесяти пяти. Знак не используется.
- `Char`. Символьный тип. Для хранения переменной отводятся два байта. В качестве значения переменной обычно применяются символы. Так как используются теперь два байта, разработчики могут применять символы Unicode.
- `Date`. Предназначен для хранения дат. Переменная занимает восемь байт. Возможные значения находятся в промежутке от первого января первого года нашей эры до тридцать первого декабря девять тысяч девятьсот девяносто девятого года[1].
- `Decimal`. Самый мощный числовой тип. Для хранения переменной подобного типа отводится шестнадцать байт. Отрицательная и положительная границы промежутка, в котором располагаются возможные значения этой переменной, одинаковы по модулю и равны +/−79 228 162 514 264 337 593 543 950 335, если использовать целые числа. Если же необходимо хранить дробные величины, то границы возможных значений будут смещены на несколько порядков в зависимости от того, сколько знаков после запятой использует разработчик. Подобный тип может использоваться только для хранения десятичных дробей. Разработчик может использовать до двадцати восьми знаков после запятой.
- `Double`. Числовой тип. Применяется для хранения чисел в экспоненциальной форме. Для хранения переменной отводится восемь байт. Отрицательные значения находятся в промежутке от −1,79769313486231E+308 до −4,94065645841247E−324. Положительные значения входят в промежуток от 4,94065645841247E−324 до 1,79769313486231E+308.

[1] Искренне надеюсь, что грядущую проблему 9999 года решать придется не нам.

- `Integer`. Предназначен для обработки целочисленных значений. Переменная подобного типа занимает четыре байта. Возможные значения находятся в промежутке от −2 147 483 648 до 2 147 483 647.
- `Long`. Предназначен для целочисленных значений. Для хранения переменной отводятся восемь байт. Возможные значения переменной подобного типа находятся в промежутке от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.
- `Object`. По сути, переменная подобного типа является всего лишь ссылкой на некий конкретный экземпляр какого-либо объекта. Для хранения переменной отводятся четыре байта.
- `Short`. Облегченный целочисленный тип. Для хранения переменной отводятся два байта. Возможные значения переменной данного типа находятся в промежутке от −32 768 до 32 767.
- `Single`. Предназначен для хранения чисел в экспоненциальной форме. Для хранения переменной отводятся четыре байта. Отрицательные возможные значения переменной такого типа расположены в промежутке от −3,402823E+38 до −1,401298E−45. Положительные значения укладываются в промежуток от 1,401298E−45 до 3,402823E+38.
- `String`. Строковый тип. Предназначен для хранения строк различной длины. Возможная длина строки может доходить до двух миллионов символов кодировки Unicode. Объем памяти для хранения переменной выделяется в зависимости от длины строки.

Мы рассмотрели все предустановленные типы Visual Basic .NET. На самом деле, разработчик может создавать собственные типы, но в работе с ASP.NET эта возможность, скорее всего, не потребуется. В крайнем случае, можно обойтись объектами. Сейчас уже нет смысла создавать собственный тип переменных просто для экономии оперативной памяти. Учитывая ее типичные объемы на машинах, предназначенных для работы WWW-серверов и возможности автоматического сборщика мусора, действующего в среде Microsoft .NET Framework, можно пользоваться оперативной памятью без оглядки на ее объем. И уж точно не стоит экономить на нескольких байтах, ухудшая тем самым читабельность и стройность кода.

Если есть типы данных, то существуют и переменные. А если мы используем переменные в своих приложениях, следует уметь их объявлять. Объявление переменных производится при помощи комбинации ключевых слов `Dim` и `As`. Типичный пример объявления целочисленных переменных выглядит следующим образом:

```
Dim i,j As Integer
```

При помощи ключевого слова `Dim` мы начинаем объявление переменных, за ним указываются наименования этих переменных, а после ключевого слова `As` записывается их тип. Программы на языке Visual Basic .NET имеют стро-

ковую структуру, т. е. в каждой строке кода может находиться только один оператор. Таким образом, разработчик избавлен от необходимости отделять операторы друг от друга при помощи символов-разделителей, таких, как символ точки с запятой. В нашем случае это означает, что мы должны использовать ключевое слово `Dim` перед каждой строкой объявления переменных, т. е. если необходимо объявить несколько целочисленных переменных и одну булеву, следует использовать следующий фрагмент кода:

```
Dim i,j As Integer
Dim r As Boolean
```

На самом деле, объявление переменных может быть и не таким простым. Мы можем создавать глобальные и защищенные переменные, дружественные, статические и частные. Но все эти дополнительные возможности необходимы лишь для работы с разветвленной объектной иерархией, которая явно не нужна в приложениях ASP.NET.

Естественно, в языке Visual Basic .NET существует ряд функций, которые позволяют преобразовывать типы. Вот их список.

- `CBool`. Возвращает булево значение. В качестве параметра функции может быть передано какое-либо условие или числовое значение.
- `CByte`. Возвращает значение типа `Byte`. В качестве параметра передается соответствующая строка или числовое значение.
- `CChar`. Возвращает символ. В качестве параметра передается целое число, находящееся в промежутке от 0 до 65 535.
- `CDate`. Возвращает значение типа `Date`. В качестве параметра можно использовать любое принятое в операционной системе обозначение даты.
- `CDbl`. Преобразовывает значение параметра к типу `Double`.
- `CDec`. Применяется для преобразования параметра к типу `Decimal`.
- `CInt`. Преобразовывает значение параметра к целочисленному типу `Integer`. Если в качестве параметра передано дробное число, то дробная часть просто округляется.
- `CLng`. Возвращает значение типа `Long`. Обработка параметра производится по образцу функции `CInt`.
- `CShort`. Преобразовывает переменную или значение к типу `Short`.
- `CSng`. Применяется для преобразования параметра к типу `Single`.
- `CStr`. Используется для преобразования данных в строковый тип `String`. Если в качестве параметра функции передается булево значение, то возвращается строка `True` или `False`. Если передаются данные типа `Date`, то функцией возвращается строка, содержащая обозначение даты в принятом для системы формате. Любое числовое значение преобразовывается в символьное представление данного числа.

И на этом мы заканчиваем рассмотрение простых типов данных, принятых в языке программирования Visual Basic .NET.

Массивы и коллекции

Все мы знакомы с понятием массивов. Это просто совокупности переменных одного типа, объединенных одним именем. Доступ к конкретному элементу массива производится при помощи указания его порядковых номеров.

В Visual Basic.NET массивы объявляются точно так же, как и обычные переменные. Отличает их только указание размера массива. То есть, если мы хотим объявить целочисленный массив, содержащий десять элементов, следует воспользоваться следующей конструкцией:

```
Dim my_array(10) As Integer
```

Естественно, разработчик не ограничен одномерными массивами. Язык Visual Basic .NET позволяет создавать многомерные массивы. При этом программист может использовать для своих массивов до тридцати двух измерений. Обычно этого количества более чем достаточно. Если же вам необходимы массивы с большей степенью размерности, то либо вы несколько неправильно спроектировали свое приложение, либо вы — гений, который в состоянии свободно оперировать более чем тридцатью двумя измерениями. Правда, далеко не факт, что с обработкой подобного массива легко справится ваша система.

Объявление многомерных массивов практически ничем не отличается от объявления одномерных массивов. Просто в скобках после имени массива следует перечислить размеры массива по всем измерениям, а именно, если мы хотим объявить целочисленный массив, представляющий собой квадратную матрицу размером десять на десять элементов, следует использовать следующий фрагмент кода:

```
Dim my_array(10,10) As Integer
```

Как мы знаем, все языки, входящие в семейство Microsoft .NET, должны действовать практически одинаково, чтобы создавать идентичный runtime-код. А у языков Visual Basic и C++ есть одно коренное различие в обработке массивов. В C++ нумерация элементов массива начинается с нуля, а в Visual Basic — с единицы. Корпорация Microsoft пошла на серьезный шаг и объявила, что теперь в языке Visual Basic .NET элементы массивов нумеруются с нуля. Честно говоря, меня это известие обрадовало, так как для меня язык C является "родным", и нумерация с нуля кажется мне более естественной. Той же самой точки зрения придерживаются многие разработчики, до знакомства с Visual Basic работавшие с языками C++ или Pascal. На самом деле, такая нумерация более органична для компьютерных систем.

Однако этот, вне сомнения, "дерзкий и революционный" шаг Microsoft практически "убил" совместимость языка Visual Basic .NET с программами,

разработанными на старых версиях языка Visual Basic. Теперь не получится просто перекомпилировать их в новой среде разработки, так как фактически у массивов получится на один элемент меньше, чем ожидали разработчики. Поэтому, чтобы не потерять совместимость с ранее написанными программами, было принято следующее "соломоново" решение. Массивы действительно нумеруются с нуля, но при этом, в момент объявления массива им автоматически добавляется по одному дополнительному элементу по всем измерениям. Таким образом, действительно достигается совместимость со старыми программами, и в то же время предоставляется возможность разработчикам при создании новых приложений пользоваться нумерацией элементов массива с нуля.

Язык Visual Basic .NET предоставляет возможность переобъявить массив в процессе работы программы, изменив его размеры. Сама идея переобъявления массива явно необычна для стиля программирования последних лет, она, скорее, унаследована из начала истории языка Visual Basic. Однако такая возможность есть, поэтому нам следует знать, как использовать ее.

Для переобъявления массива используется ключевое слово `ReDim`. Процедура изменения размеров массива выглядит приблизительно следующим образом:

```
Dim my_array(10,10) As Integer
. . .
ReDim my_array(10,20)
```

После выполнения директивы `ReDim` размер массива `my_array` по второму измерению будет увеличен вдвое. При этом необходимо осознавать, что мы можем изменить лишь объем массива, но не его тип или количество измерений массива. Также по умолчанию при переобъявлении массива значения его элементов теряются. Если же необходимо сохранить их, следует использовать ключевое слово `Preserve`. В этом случае процедура переобъявления размеров массива будет выглядеть следующим образом:

```
Dim my_array(10,10) As Integer
. . .
ReDim Preserve my_array(10,20)
```

Следует также помнить, что при создании любого массива, он автоматически становится объектом. Да, в объектной иерархии Visual Basic .NET существует объект `System.Array`, и любой массив является экземпляром данного класса. Никто не заставляет разработчика относиться к массиву как к классу, и применять соответствующие свойства, методы и приемы программирования, но в определенных ситуациях есть смысл воспользоваться массивом как объектом. Полное рассмотрение методов и свойств объекта `System.Array` находится в разделе, посвященном объектной иерархии языка Visual Basic .NET.

Альтернативой массивам данных с жестко заданными размерами являются так называемые "коллекции". Это просто набор однотипных элементов.

У коллекций нет жестко заданных размеров, и они изначально являются объектами. Поэтому объявление коллекции не обойдется без ключевого слова `New`, при помощи которого выделяется память под те или иные экземпляры различных классов, т. е. стандартное объявление коллекции будет выглядеть следующим образом:

```
Dim my_col As New Collection()
```

Видно, что при объявлении мы не указываем тип элементов коллекции. Этот тип будет определен автоматически, как только будет добавлен к коллекции ее первый элемент. Добавление элемента к коллекции производится при помощи метода `Add`, а удаление элемента при помощи метода `Remove`. Пример работы с этими методами приведен в следующем фрагменте кода:

```
my_col.Add(New my_object, "Пустой элемент")
my_col.Remove(1)
```

Теперь разберем этот фрагмент кода. Метод `Add`, как мы видели, получает два параметра. Мы добавляем в коллекцию переменную `my_object`, которая на самом деле является объектом. Более того, этот экземпляр объекта не был инициализирован, и для него не была выделена память. Поэтому мы в качестве первого параметра метода `Add`, используем связку наименования экземпляра объекта и ключевого слова `New`, которое заставляет систему выделить требуемый объем памяти. Таким образом, мы добавляем в коллекцию новый пустой экземпляр объекта.

Второй параметр метода `Add` является текстовой строкой, которая описывает добавляемый элемент. Дело в том, что доступ к отдельным элементам коллекции может производиться как по их номеру, так и по текстовому описанию. Естественно, текстовые описания различных элементов коллекции не должны совпадать.

Вторая строка рассматриваемого фрагмента кода содержит метод `Remove`, при помощи которого удаляется какой-либо элемент коллекции. В качестве параметра данному методу передается либо порядковый номер удаляемого элемента, либо его текстовое описание. В нашем случае мы удаляем первый элемент коллекции. Необходимо помнить, что нумерация элементов коллекции начинается с единицы, а не с нуля, как у элементов массива.

Для доступа к конкретному элементу коллекции следует воспользоваться методом `Item`. В качестве параметра данному методу может передаваться порядковый номер требуемого элемента коллекции или его текстовое уникальное описание. Использование данного метода показано в следующем фрагменте кода:

```
t=my_col.Item(1)
```

Данная операция присваивает переменной `t` значение первого элемента коллекции. Излишне напоминать, что типы переменной и элемента коллекции должны совпадать. Впрочем, метод `Item` используется для коллекций по

умолчанию, поэтому предыдущая операция может быть записана следующим образом:

```
t=my_col(1)
```

Обработку всех элементов коллекции удобнее всего производить при помощи цикла `For Each`. Однако можно воспользоваться стандартным циклом `For`, если знать количество элементов в коллекции. Это количество хранится в свойстве `Count`. Естественно, это свойство имеет статус "только для чтения".

Встроенные функции

Теперь пришло время рассмотреть список стандартных функций языка Visual Basic .NET. Некоторые из них явно не понадобятся для разработки ASP-приложений, но их очень немного. В конце концов, встроенные функции это возможности языка, которые и создают действующий код в рамках тех или иных управляющих логических конструкций. Поэтому приводится список встроенных функций Visual Basic .NET в алфавитном порядке.

- `Abs`. В качестве параметра передается любое числовое значение или переменная числового типа. Функция возвращает абсолютное значение параметра, т. е. его модуль. Тип возвращаемого значения всегда совпадает с типом переданного параметра.
- `AppActivate`. Передает фокус ввода окну какого-либо открытого приложения. В ASP.NET, естественно, не используется.
- `Asc`. Возвращает числовое представление символа, переданного функции в качестве параметра. По сути дела, функция возвращает числовой код символа-параметра. В качестве параметра может также передаваться значение типа `String`, но возвращаться будет числовой код первого символа переданной строки. Возвращаемое значение имеет тип `Integer`.
- `AscW`. В качестве параметра принимается только значение типа `String`. В остальном механизм действия функции идентичен функции `Asc`.
- `Atn`. Реализует математическую функцию арктангенса. Передаваемый параметр и значение функции имеют тип `Double`.
- `Beep`. Воспроизводит единичный звук при помощи встроенного динамика компьютера. Не требуется никаких параметров, никаких значений не возвращается. Функция явно унаследована из предыдущих версий Visual Basic и в данное время практически не используется. И уж тем более, не потребуется в ASP-приложениях, так как они исполняются на www-сервере, следовательно, и звук будет воспроизводиться на сервере.
- `CallByName`. Выполняет некий метод объекта, переданного в качестве параметра, или устанавливает значения свойств этого объекта. Явно бесполезна в современных условиях программирования с плотным использо-

ванием стандартных методов объектно-ориентированного программирования.

- `ChDir`. Меняет текущий каталог. В качестве параметра передается путь к каталогу, который будет текущим.
- `ChDrive`. Меняет текущий логический диск. В качестве параметра передается символ, обозначающий диск, который будет текущим.
- `Choose`. В качестве параметров передается числовое значение типа `Double` и список элементов, из которых функция будет выбирать возвращаемое значение. Выбор значения производится на основе переданного числового параметра.
- `Chr`. Функция возвращает символ, код которого передан ей в качестве параметра. Параметр имеет тип `Integer`, возвращается значение типа `Char`.
- `Command`. Применяется в разработке приложений, которые будут скомпилированы в отдельный исполняемый exe-файл. Возвращает строку с опциями, введенными в командную строку после наименования исполняемого файла. В ASP-приложениях не используется.
- `Cos`. Реализует математическую функцию косинуса. Передаваемый параметр и значение функции имеют тип `Double`.
- `CreateObject`. Создает COM-объект и возвращает ссылку на него. В качестве обязательного параметра передается идентификатор создаваемого объекта. Также может быть передано наименование объекта, но данный параметр не является обязательным.
- `CurDir`. Возвращает значение типа `String`, в котором содержится путь к текущему каталогу. В качестве необязательного параметра может быть передан символ, идентифицирующий существующий в системе логический диск. В этом случае будет возвращен путь к текущему каталогу на этом диске. Если параметр не задан, то для расчета значения будет использован текущий диск.
- `DateAdd`. Возвращает дату, определяемую на основе параметров, переданных в функцию. Функция обладает тремя обязательными параметрами. Первый параметр обозначает тип интервала времени, который будет отсчитываться от стартовой даты. Этот интервал может иметь тип `DateInterval` или `String`. Значения типа `DateInterval` это всего лишь несколько констант. Им соответствуют строковые константы, которые можно использовать в качестве первого параметра. Полный список констант типа `DateInterval` и соответствующих им строковых значений приведен в приложении. Второй параметр содержит количество отсчитываемых интервалов. А в качестве третьего параметра используется дата, от которой и будет вестись отсчет. Таким образом, если мы используем функцию `DateAdd(DateInterval.Day,1,#Jan 1,2002#)`, то в качестве

результата ее деятельности мы получим значение `#Jan 2,2002#`, т.е. второе января две тысячи второго года.

- `DateDiff`. Возвращает разницу между двумя датами в качестве значения типа `Long`. Принимает три параметра. Первым параметром передается тип интервала времени, в котором будет вестись отсчет разницы между сравниваемыми датами, т. е. если мы указываем тип, ориентированный на дни, то результатом действия функции будет разница именно в днях. Второй и третий параметры имеют тип `Date`. Это соответственно те даты, разница между которыми вычисляется функцией.
- `DatePart`. Возвращает в качестве целочисленного значения определенную часть даты. В качестве параметров функции передаются уже знакомое нам указание типа интервала и дата, из которой необходимо выделить фрагмент. Так, если мы укажем тип промежутка времени, ориентированный на месяцы, то в качестве результата работы функции мы получим номер месяца, в котором находится искомая дата, переданная как второй параметр.
- `DateSerial`. Возвращает значение типа `Data`, созданное на основе трех целочисленных значений, переданных функции в качестве параметра, а именно, в функцию передаются год, месяц и день как значения типа `Integer`. Так, выражение `DateSerial(2002,2,9)` возвращает значение типа `Date`, указывающее на девятое февраля две тысячи второго года.
- `DateValue`. Как и предыдущая функция, создает значение типа `Date`, но при этом опирается не на числовое представление ее частей, а на строковое отображение даты, т. е. в качестве параметра данной функции передается строка, описывающая некоторую дату в установленном формате операционной системы. Для того чтобы получить значение, указывающее на девятое февраля две тысячи второго года, как в предыдущем примере, следует воспользоваться вызовом `DateValue("February 09, 2002")`.
- `Day`. Возвращает конкретное число, т. е. номер дня в месяце, опираясь на дату, переданную в качестве параметра. Возвращаемое значение имеет тип `Integer`, другими словами, выражение `Day(#Feb 9, 2002#)` возвратит число "девять".
- `DeleteSetting`. Удаляет из реестра операционной системы записи о каком-либо приложении. В качестве параметров функции передаются официальное наименование приложения, записи которого будут удалены, и наименования раздела реестра, из которого будут удаляться эти записи. В качестве необязательного параметра может быть добавлено точное наименование конкретной записи, которая подлежит удалению. Естественно, в приложениях ASP.NET функция не используется.
- `Dir`. Возвращает наименование файла, включая его расширение. В качестве необязательных параметром может быть передана строка, включаю-

щая в себя путь к каталогу, в котором необходимо найти файл, и шаблон его имени, а также атрибуты файла. Если эти параметры не задаются, в качестве результата будет возвращено наименование первого найденного файла в текущем каталоге. Так, функция `Dir("C:\docs\*.txt")` вернет наименование первого найденного текстового файла в каталоге docs, находящемся на диске C. Если в данном каталоге нет файлов с подобным расширением, будет возвращена пустая строка.

- `Environ`. Возвращает значение системной переменной окружения, наименование которой передано функции в виде строки. Возвращается также значение типа `String`.
- `EOF`. Функция возвращает логическое значение, сигнализирующее о достижении границы файла, открытого для чтения. В качестве значения функции передается значение типа `Integer`, являющееся дескриптором читаемого файла. В том случае, если указатель позиции, с которой производится чтение, указывает уже на конец файла, функция возвращает значение `True`.
- `ErrorToString`. Функция конвертирует код ошибки в текстовое обозначение ошибки. Естественно, при этом используются официальные англоязычные наименования ошибок. Чаще всего применяется для вывода информации в модальных окнах, сообщающих об ошибках, поэтому используется в приложениях ASP.NET крайне редко.
- `FileAttr`. Функция возвращает режим доступа к открытому ранее файлу. В качестве значения передается целочисленный дескриптор искомого файла. Возвращаемое значение имеет тип `Integer`.
- `FileClose`. Закрывает открытый для доступа файл. В качестве параметра передается целочисленный дескриптор открытого файла.
- `FileCopy`. Копирует файл в другой каталог. В качестве параметра в функцию передаются полное имя копируемого файла и путь к каталогу, куда необходимо скопировать этот файл. Если при этом необходимо сменить имя файла, во втором параметре может быть указан не просто путь к каталогу, а полное наименование создаваемого файла.
- `FileDateTime`. Возвращает значение типа `Date`, в котором показывается дата создания или последней модификации файла, полный путь к которому передается как строка в качестве параметра функции.
- `FileLen`. Возвращает объем файла в байтах. Полный путь к файлу передается в функцию как параметр. Возвращаемое значение имеет тип `Long`.
- `FileOpen`. Открывает файл. В качестве параметра функции передается целочисленное значение, указывающее дескриптор открываемого файла (другими словами об уникальности дескрипторов файла должен заботиться сам разработчик), имя открываемого файла и вариант доступа к нему.

- `FileWidth`. Задает длину строк, которые будут считываться из файла, открытого для чтения. В качестве параметров функции передаются дескриптор открытого файла и целочисленное значение, которое и устанавливает длину читаемых строк.
- `Fix`. Обрезает дробную часть у числового значения, переданного в качестве параметра, и возвращает целое число. Необходимо учитывать, что эта функция не округляет параметр до ближайшего целого, а просто отбрасывает его дробную часть.
- `FormatCurrency`. Функции передается некая строка или объект как параметр, а функция преобразовывает его в финансовый формат. Особенности формата обуславливаются установками операционной системы.
- `FormatDateTime`. Преобразовывает переданный параметр в строку, которая содержит дату. Особенности формата даты, как и в предыдущей функции, определяются установками операционной системы.
- `FormatNumber`. Преобразовывает в строку числовое значение, переданное в качестве параметра.
- `FormatPercent`. Функция форматирует переданный параметр как строку с добавлением знака процента.
- `FreeFile`. Функция не имеет параметров. Выдает целочисленное значение, которое можно использовать как дескриптор для открываемого файла, т. е. данная функция фактически возвращает свободный дескриптор, который затем разработчик должен использовать для открытия файла.
- `GetAllSettings`. Функция возвращает двумерный массив строк, состоящий из наименований записей некоего приложения в реестре операционной системы и их значений. В качестве параметров функции передаются официальное наименование приложения, записи которого необходимо получить в реестре, и наименования раздела реестра, откуда будут извлечены искомые данные.
- `GetAttr`. Функция возвращает атрибуты файла, путь к которому передан функции в качестве параметра. Возвращаемое значение имеет перечислимый тип `FileAttribute`.
- `GetChar`. Функция возвращает символ с указанным порядковым номером из строки. Сама строка, откуда извлекается символ, и порядковый номер извлекаемого символа передаются функции в качестве параметров.
- `GetExeption`. Функция возвращает исключение, которое могло произойти в результате действий приложения. Функция применяется только в составе объекта `Err`. Возвращаемое значение имеет тип `Exception`.
- `GetSetting`. Возвращает значение одного конкретного ключа в реестре операционной системы для некоего приложения. В качестве параметров функции передаются наименование приложения, которому соответствует

данный ключ, секция реестра и наименование ключа, значение которого будет возвращать функция.

- `Hex`. Возвращает строку, в которой записано шестнадцатеричное представление целого числа, переданного функции в качестве параметра. Параметр может иметь любой целочисленный тип, от `Byte` до `Long`. Также может быть передан параметр типа `Object`, но в этом случае ответственность за передачу правильного значения перекладывается на разработчика. В том случае, если передано не целое число, параметр будет округлен до ближайшего целого.
- `Hour`. Функция выделяет из параметра, имеющего тип `DateTime`, час и возвращает его номер в качестве значения, имеющего тип `Integer`. Результат, естественно, лежит в промежутке от нуля до двадцати трех.
- `IIf`. Функция является аналогом условного оператора `If`. В качестве параметров ей передается логическое условие и два значения, одно из которых она вернет в качестве результата. Эти два параметра имеют тип `Object`, т. е. фактически, разработчик может использовать любой тип. В том случае, если переданное как первый параметр, логическое значение истинно, функция возвращает второй параметр. Если логическое значение ложно, возвращается третий параметр.
- `Input`. Функция читает данные из открытого файла. В качестве параметров функции передается целочисленный дескриптор файла и та переменная, в которую будет записан прочитанный блок информации из файла. Соответственно, функция ничего не возвращает.
- `InputBox`. Функция отображает диалоговое окно с одним полем для ввода пользователем текстовых данных. В качестве обязательного параметра функции передается строка текста, которая будет отображена в данном диалоговом окне. Функция возвращает введенный пользователем текст как строку, т. е. используется тип `String`. Естественно, в приложениях ASP.NET данная функция не понадобится.
- `InputString`. Функция читает строку из файла, открытого в режиме `Input` или `Binary`. В качестве параметров функции передаются целочисленный дескриптор файла, из которого будет осуществляться чтение, и количество читаемых символов, тоже представляемое как значение типа `Integer`. Функция возвращает значение типа `String`, т. е. прочитанную строку заданной длины.
- `InStr`. Функция возвращает целочисленное значение, указывающее, с какой позиции одна строка входит в качестве подмножества в другую строку. В качестве первого параметра функции передается целое число, указывающее позицию в основной строке, откуда следует начинать сравнение, вторым параметром является та самая строка, в которой мы ищем вхождение подстроки, третьим параметром передается строка, которую мы ищем в исходной строке.

- `InStrRev`. Эта функция, как и предыдущая, возвращает позицию, с которой одна строка входит в другую, но при этом поиск и сравнение производятся с правого конца искомой строки. Впрочем, в качестве третьего параметра так же можно передать целое число, устанавливающее позицию, с которой будут производиться поиск и сравнение. Естественно, позиция в данном случае отсчитывается от правого края искомой строки.
- `Int`. Отбрасывает дробную часть переданного в качестве параметра числа, приводя его к целому. Сходна с функцией `Fix`. Разница проявляется в обработке отрицательных значений. Функция `Int` приводит значение параметра к меньшему целому числу, а именно, значение −5,5 будет преобразовано данной функцией в −6.
- `IsArray`. Проверяет, является ли переданный ей параметр массивом. Если параметр действительно является массивом, функция возвращает значение `True`. В ином случае функция возвращает значение `False`.
- `IsDate`. Проверяет, может ли переданный функции параметр быть преобразован в дату. Точнее, в значение типа `Date`. Функция возвращает логическое значение типа `Boolean`.
- `IsError`. Функция получает в качестве параметра некое выражение или переменную с данным выражением. В том случае, если выполнение данного выражения создаст нештатную ситуацию, ошибку, которая возбудит соответствующее исключение, функция вернет булево значение `True`. Во всех иных случаях возвращается значение `False`.
- `IsNothing`. В качестве параметра функции передается некая переменная. В том случае, если данной переменной не присвоено какое-либо значение, т. е. она не инициализирована, функция возвращает логическое значение `True`. Иначе будет возвращено значение `False`.
- `IsNumeric`. Функция проверяет, может ли быть переданное в качестве параметра значение преобразовано к какому-либо числовому типу. В тех случаях, когда это возможно, функция возвращает значение `True`. Иначе будет возвращено значение `False`.
- `Join`. В качестве обязательного параметра функции передается массив строк. Сама функция объединяет все строки из массива в одну и возвращает эту большую строку в качестве результата своей работы. Вторым параметром может быть передана строка или символ, которые будут разделять строки из отдельных элементов массива. По умолчанию в качестве разделителя используется обычный пробел. Однако если принудительно в качестве разделителя задать пустую строку, то элементы строчного массива не будут отделены друг от друга.
- `Kill`. Функция принудительно удаляет файл, путь к которому передан функции в качестве параметра, с диска системы пользователя.

- `LCase`. Функция переводит строку или символ, переданные ей в качестве параметра, в нижний регистр символов.
- `Left`. В качестве параметров функция получает строку и целое число, обозначающее количество символов. Результатом работы функции будет строка, содержащая искомое количество символов, которые скопированы у исходной строки, начиная с ее первого символа, т. е. функция возвращает подстроку первого параметра, которая начинается с его левого края.
- `Len`. В качестве параметра функция принимает значения всех числовых и строчных типов. Функцией возвращается длина в символах строчного представления переданного параметра.
- `LineInput`. Читает одну строку из файла, открытого для чтения. В качестве параметра функции передается целочисленный дескриптор файла, из которого будет производиться чтение. Функция возвращает значение типа `String`.
- `Loc`. Функция возвращает позицию, с которой будет производиться запись или чтение в открытом файле. Соответственно, возвращаемое значение имеет тип `Long`. В качестве параметра функции передается целочисленный дескриптор искомого файла.
- `Lock`. Закрывает файл для других процессов. Обычно применяется в тех случаях, когда необходимо произвести запись в файл, и необходимо, чтобы во время записи иные приложения и процессы не изменяли содержимое файла. После завершения работы критического участка, блокировка с файла обычно снимается. В качестве параметра функция принимает целочисленный дескриптор блокируемого файла.
- `LOF`. Функция возвращает значение типа `Long`, в котором содержится объем открытого для доступа файла в байтах. Функция может применяться для измерения длин только открытых файлов. Соответственно, в качестве параметра в функцию передается целочисленный дескриптор файла.
- `LSet`. Функции передается в качестве параметров строка и целое число. Функция возвращает строку, которая является подстрокой первого параметра. Длина подстроки определена вторым параметром-числом. При этом подстрока составляется начиная с левого края строки-параметра.
- `LTrim`. В качестве параметра функции передается строка. Функция обрезает пробелы в начале переданной строки, если таковые есть, и возвращает строку без начальных пробелов.
- `Mid`. В качестве параметра функции передается строка. Функция возвращает подстроку, опираясь на второй и третий параметры, являющиеся целыми числами. Второй параметр обозначает позицию символа исходной строки, начиная с которого будет создаваться возвращаемая строка. Третий параметр является длиной возвращаемой подстроки.

- `Minute`. Функция выделяет минуты из параметра функции, имеющего тип `DateTime`. Возвращаемое значение имеет тип `Integer` и находится в промежутке от нуля до пятидесяти девяти.
- `MkDir`. Функция создает каталог, полное наименование которого передается ей в качестве параметра.
- `Month`. Функция выделяет номер месяца из даты, переданной в качестве параметра. Параметр, естественно, имеет значение `DateTime`. Возвращаемое значение имеет тип `Integer` и находится в промежутке от единицы до двенадцати.
- `MonthName`. Функция возвращает строку с наименованием месяца, порядковый номер которого передан ей в качестве параметра. Данный параметр должен быть целочисленным и находится в промежутке от единицы до двенадцати. В ином случае будет возвращена пустая строка.
- `MsgBox`. Функция отображает диалоговое окно. Текст сообщения диалогового окна передается в качестве параметра типа `String`. Естественно, в приложениях ASP.NET данная функция не используется.
- `Oct`. Возвращает строку, в которой записано представление целого числа в восьмеричной системе исчисления, переданного функции в качестве параметра. Параметр может иметь любой целочисленный тип, от `Byte` до `Long`. Также может быть передан параметр типа `Object`, но в этом случае ответственность за передачу правильного значения перекладывается на разработчика. В том случае, если передано не целое число, параметр будет округлен до ближайшего целого.
- `Print`. Записывает одну или несколько строк в файл, открытый для записи. В качестве параметра функции передается целочисленный дескриптор файла и список записываемых строк, разделенных запятыми.
- `PrintLine`. Фактически идентична предыдущей функции, но при записи строк в файл добавляет к каждой строке символы перевода каретки, что позволяет более корректно отображать записанные ранее строки в текстовых контейнерах.
- `QBColor`. Функция принимает в качестве параметра целочисленное значение от нуля до пятнадцати, ассоциирует с ним один из шестнадцати стандартных цветов WWW и возвращает целое число, соответствующее RGB-коду этого цвета.
- `Rename`. Функция переименовывает файл или каталог. В качестве первого параметра функции передается строка, содержащая текущее имя искомого файла или каталога, а второй параметр содержит строку с новым именем.
- `Replace`. Функция находит в строке, переданной как параметр, некую подстроку и меняет ее на другую заданную последовательность символов.

Первым параметром передается искомая строка, в которой будет производиться поиск и замена. Второй параметр — подстрока, которую будет необходимо заменить. Третий параметр содержит строку, которой будет заменяться найденная подстрока.

- `Reset`. Функция закрывает все файлы, открытые приложением.
- `RGB`. Функция возвращает целочисленное значение, обозначающее некий цвет, RGB-код которого передан функции в качестве параметра. Точнее, передается три параметра — насыщенность красного, зеленого и синего цветов соответственно. Данные параметры являются целыми числами и находятся в промежутке от нуля до двухсот пятидесяти пяти.
- `Right`. В качестве параметров функция получает строку и целое число, обозначающее количество символов. Результатом работы функции будет строка, содержащая искомое количество символов, которые скопированы у исходной строки, начиная с ее последнего символа, т. е. функция возвращает подстроку первого параметра, которая начинается с его правого края.
- `RmDir`. Функция удаляет каталог, наименование которого передано ей в качестве параметра.
- `Rnd`. Возвращает случайное число. Возвращаемое значение имеет тип `Single`.
- `RSet`. Фактически идентична функции `Right`. Но есть одно отличие. Если указанный размер подстроки больше, чем размер исходной строки, возвращаемая подстрока будет дополнена до необходимого размера пробелами слева.
- `RTrim`. В качестве параметра функции передается строка. Функция обрезает пробелы в конце переданной строки, если таковые есть, и возвращает строку без конечных пробелов.
- `SaveSetting`. Функция предназначена для создания записей в реестре операционной системы Windows. Функции передается четыре параметра типа `String`. В качестве первого параметра передается наименование приложения, второй параметр содержит наименование раздела реестра, третий — наименование записи-ключа, а четвертый — собственно, значение этого ключа.
- `Second`. Функция выделяет секунды из параметра функции, имеющего тип `DateTime`. Возвращаемое значение имеет тип `Integer` и находится в промежутке от нуля до пятидесяти девяти.
- `Seek`. Функция возвращает текущую позицию чтения или записи в открытом ранее файле. Возвращаемое значение имеет тип `Long`. В качестве параметра передается целочисленный дескриптор файла, текущую позицию которого разработчик желает получить.

- `SetAttr`. Функция устанавливает атрибуты для какого-либо файла. В качестве параметров передаются строка, содержащая полное наименование файла, включая путь к нему, а также устанавливаемый атрибут для этого файла.
- `Shell`. Запускает приложение как процесс. В качестве обязательного параметра передается полное наименование исполняемого файла. Очевидно, в приложениях ASP.NET данная функция не найдет применения.
- `Space`. Функция возвращает строку, состоящую из пробелов. Количество пробелов задается целочисленным параметром.
- `SPC`. Функция используется вместе с функциями печати `Print` и `PrintLine`. Добавляет некоторое количество пробелов перед следующей выводимой строкой. Количество пробелов определяется целочисленным параметром.
- `Split`. Функция получает в качестве параметра строку, разбивает ее на несколько подстрок и возвращает их в качестве строчного массива. По умолчанию в качестве разделяющего символа используется пробел, но разработчик может воспользоваться вторым необязательным параметром, в котором необходимо указать символ-разделитель.
- `Str`. Преобразовывает переданное в качестве параметра числовое значение в строку.
- `StrComp`. Функция сравнивает две строки, переданные ей в качестве параметров. В том случае, если строки совпадают, возвращается нулевое значение.
- `StrConv`. Функция предназначена для конвертирования строки по установленным правилам. В качестве первого параметра передается строка, предназначенная для обработки, а второй параметр устанавливает механизм ее изменения. В качестве второго параметра используется значение из перечислимого набора `Microsoft.VisualBasic.VbStrConv`. Значение `VbStrConv.None` указывает, что строка останется без изменений. Значение `VbStrConv.UpperCase` преобразовывает все символы обрабатываемой строки в верхний регистр. Для перевода всех символов в нижний регистр используется значение `VbStrConv.LowerCase`. А значение `VbStrConv.ProperCase` переводит в верхний регистр только первый символ каждого слова.
- `StrDup`. Функция создает и возвращает строку, состоящую из нескольких повторений одного и того же символа. В качестве первого параметра передается целое число, указывающее, сколько раз будет повторен символ, по сути, это длина создаваемой строки. Второй параметр — сам символ, из которого будет создаваться строка.
- `StrReverse`. Функция принимает в качестве параметра строку. В результате действия функции возвращается строка, которая состоит из симво-

лов строки-параметра в обратном порядке, другими словами, исходная строка просто прочитывается в обратном порядке, и в таком виде записывается в результирующую строку.

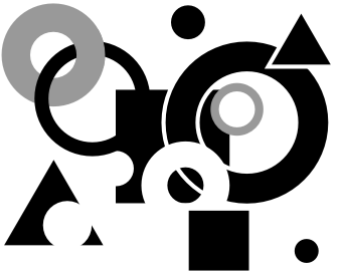
- `Tab`. Функция используется вместе с функциями печати `Print` и `PrintLine`. Добавляет некоторое количество символов табуляции перед следующей выводимой строкой. Количество символов табуляции определяется целочисленным параметром. Впрочем, параметр не является обязательным, поэтому если использовать функцию без него, позиция печати просто будет смещена вправо.
- `TimeSerial`. Функция возвращает значение типа `Date`, для которого явным образом устанавливается только время. Функции передается три целочисленных параметра, которые устанавливают для возвращаемого значения часы, минуты и секунды соответственно.
- `TimeValue`. Функция возвращает значение типа `Date`, для которого явным образом устанавливается только время. При этом в качестве основы для установки времени используется параметр типа `String`, в котором записано представление времени в формате, присущем операционной системе.
- `Trim`. В качестве параметра функции передается строка. Функция обрезает пробелы в конце и начале переданной строки, если таковые есть, и возвращает строку без начальных и конечных пробелов.
- `TypeName`. Функция принимает в качестве параметра значение любого стандартного типа и возвращает наименование типа параметра.
- `UCase`. В качестве параметра функции передается текстовая строка и символ, а функция переводит все символы в верхний регистр и возвращает результирующую строку.
- `Unlock`. Снимает блокировку с файла, наложенную перед этим при помощи функции `Lock`. В качестве параметра функция получает целочисленный дескриптор ранее открытого файла.
- `Val`. Функции передается строка в качестве параметра. Функция выделяет из нее цифры и формирует из них число, которое и возвращает. Возвращаемое значение может иметь тип `Double` или `Long`.
- `WeekDay`. Функция работает с параметром типа `DateTime` и возвращает целочисленное значение, находящееся в промежутке от единицы до семи, обозначающее номер дня недели, на который приходится дата, переданная как параметр.
- `WeekdayName`. Функция обычно применяется в паре с предыдущей. В качестве параметра ей передается целочисленное значение от единицы до семи, обозначающее номер дня недели, а в качестве результата возвращается значение типа `String`, содержащее наименование этого дня недели. Естественно, используются стандартные установки операционной системы.

- `Write`. Функция записывает информацию в открытый файл в виде текста. В качестве параметров передаются целочисленный дескриптор файла и список записываемых значений, разделенных запятой.
- `WriteLine`. Функция фактически идентична только что рассмотренной функции `Write`, однако при записи строк в файл, она добавляет к каждой строке символ возврата каретки.
- `Year`. Функция выделяет номер года из параметра функции, имеющего тип `DateTime`. Возвращаемое значение имеет тип `Integer` и находится в промежутке от нуля до девяти тысяч девятьсот девяноста девяти.

И на этом мы заканчиваем список функций, входящих в состав языка Visual Basic .NET. Некоторые функции мы не рассмотрели в связи с тем, что в приложениях ASP.NET они не смогут найти применения. Впрочем, если быть точным, то в разделе, посвященном конструкциям управляющей логики, мы тоже не рассматривали очень распространенный оператор `Go to`, который явно устарел и может только повредить создаваемым приложениям. Дело в том, что язык Visual Basic .NET в целях совместимости унаследовал достаточно много функций и конструкций от предыдущих версий языка Visual Basic, и разработчикам совершенно нет нужды использовать устаревшие приемы работы.

На этом мы закончим обзор языка Visual Basic. Конечно, он далеко не полон, так как мы не рассмотрели вопросы объявления собственных процедур и функций, объектную модель языка и обработку исключений, но ведь цель этой книги не полное рассмотрение языка Visual Basic .NET, а знакомство с технологией ASP.NET. Для того чтобы рассмотреть этот язык программирования в полном объеме, следует писать отдельную книгу, поэтому ограничимся его основами и перейдем к рассмотрению ASP.NET.

Глава 3



Приложения ASP

Здравствуй, мир

Попробуем создать свое первое ASP.NET-приложение. Теоретически мы могли бы воспользоваться для этого обычным текстовым редактором Блокнот, входящим в состав стандартной поставки Windows. Этого достаточно, если у нас есть функционирующий Web-сервер IIS и среда выполнения .NET Framework. Однако среда разработки Visual Studio предоставляет действительно очень хорошие возможности для разработки приложений ASP.NET. Но перейдем к делу. После запуска Visual Studio .NET необходимо выполнить команду меню **File | New | Project** (Файл | Создать | Проект). Эта команда активизирует диалоговое окно **New Project** (Создать проект), показанное на рис. 3.1. Для того чтобы создать приложение ASP.NET, следует в группе **Project Types** (Типы проектов) выделить элемент с наименованием **Visual Basic Projects** (Проекты Visual Basic), в группе **Templates** (Шаблоны) выделить иконку с наименованием **ASP.NET Web Application** (Веб-приложение ASP.NET), а в поле ввода **Name** (Имя) указать наименование создаваемого проекта. При этом будет создан одноименный виртуальный каталог в файловом пространстве WWW-сервера. Наш первый проект мы назовем **Hello**, как это показано на рис. 3.1. После того, как все необходимые данные указаны, следует, естественно, нажать кнопку **OK**.

Когда будет создан новый виртуальный каталог, в котором, кстати говоря, будут находиться все те же дополнительные папки, что и в корневом каталоге WWW-сервера, среда разработки Visual Studio .NET отобразит страницу с наименованием WebForm1.aspx, присваиваемым по умолчанию. Это и есть основа для той Web-страницы, которую мы будем разрабатывать, и которую будет обслуживать создаваемое нами ASP.NET-приложение. Если вспомнить, что раньше ASP-разработчикам требовалось сначала создавать Web-страницу, а потом вручную привязывать к ней ASP-сценарии, так как практически не было редакторов, которые бы позволяли разрабатывать дизайн и движок одновременно, то становится понятно, что теперь разработчики

получили возможность не только объединить код и дизайн в пределах одной страницы, но и совместить процессы разработки. Внешний вид среды разработки Visual Studio .NET с Web-страницей в режиме создания дизайна показан на рис. 3.2.

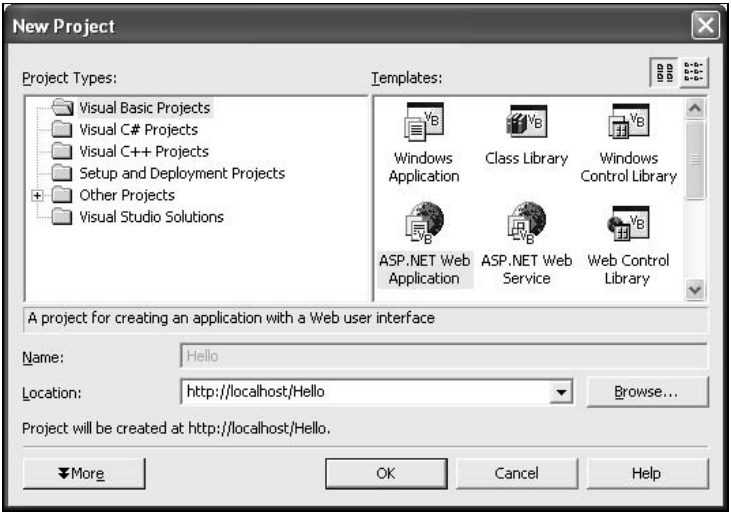


Рис. 3.1. Диалоговое окно **New Project**

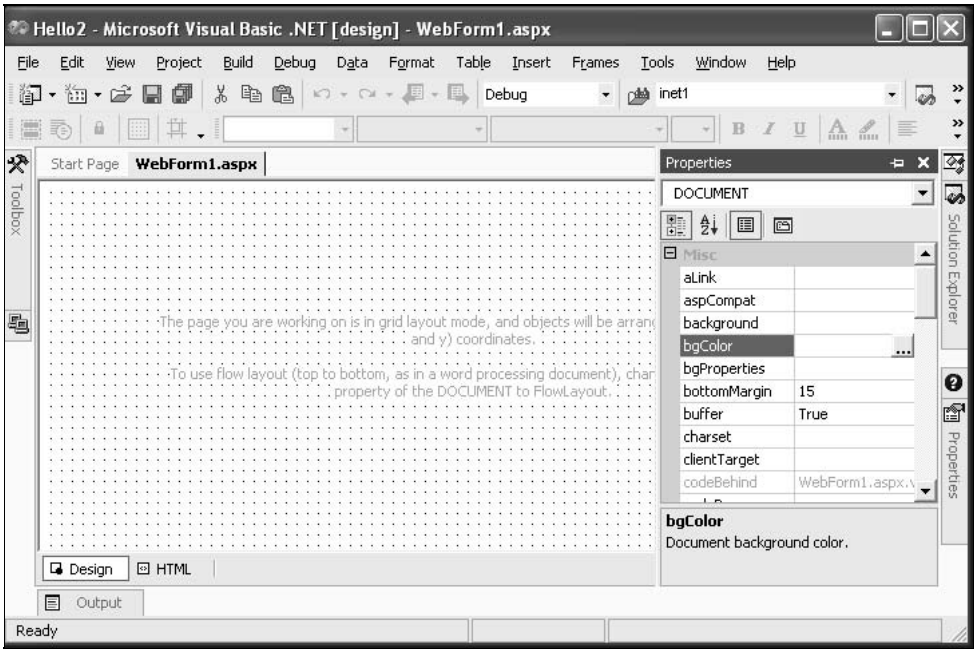


Рис. 3.2. Внешний вид среды разработки Visual Studio .NET

Очевидно, что создаваемую Web-страницу мы можем видеть как в режиме дизайна, так и в виде HTML-кода. Переключение между двумя этими ипостасями производится при помощи вкладок **Design** и **HTML**, отображаемыми в нижней части окна, содержащего разрабатываемую Web-страницу. На первый взгляд, создание Web-страниц в подобном средстве разработки может показаться несколько неудобной, так как средства разметки явно уступают возможностям фаворитов WYSIWUG-редакторов для Web, таких как FrontPage или DreamVeawer. Однако, предполагается, что ASP-разработчик является достаточно квалифицированным специалистом и сможет разобраться, где находятся средства визуального проектирования Web-страниц (обычно они все находятся не дальше одного-двух уровней основного меню), и как подключать дополнительные технологии. Помимо стандарта HTML, поддерживаются, естественно, все языки создания сценариев, действующие на стороне пользователя, CSS, и, соответственно, сплав CSS языков сценариев и объектных моделей браузеров — технология, известная под именем DHTML.

Но перейдем все-таки к созданию своего приложения. Для начала следует активизировать набор компонентов для разработки Web-страниц с наименованием **Toolbox** (Инструментарий). Для этого следует или нажать одноименную кнопку на основной инструментальной панели, или выполнить команду **View | Toolbox** (Вид | Инструментарий), дублируемую комбинацией клавиш <Ctrl>+<Alt>+<X>. При этом основное окно среды разработки Visual Studio .NET примет вид, показанный на рис. 3.3.

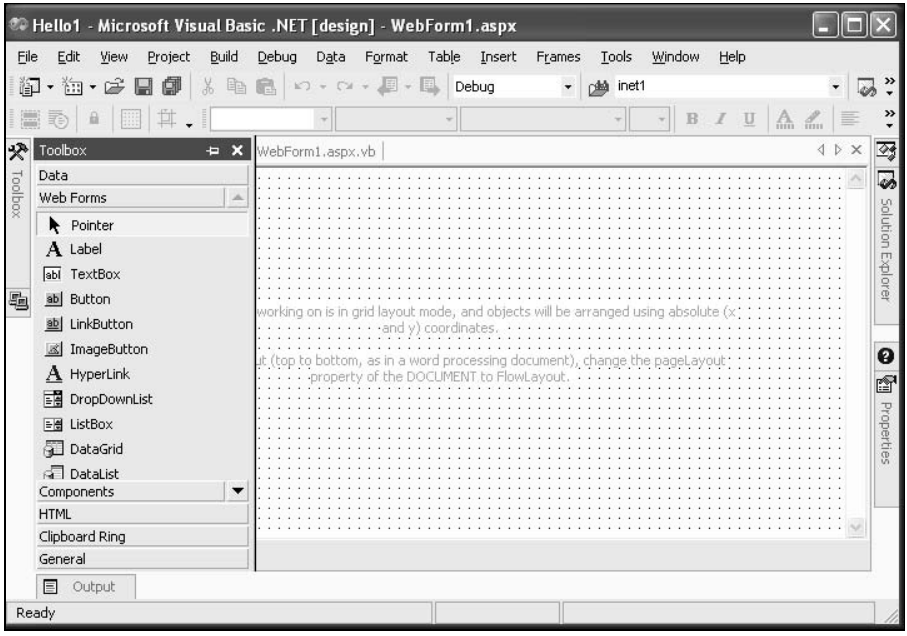


Рис. 3.3. Внешний вид основного окна среды разработки Visual Studio.NET с активизированным набором компонентов **Toolbox**

На панели инструментов **ToolBox** (Инструментарий) находится несколько вкладок с различными типами компонентов, которые мы можем использовать при разработке собственных Web-страниц. Основные элементы дизайна собраны на вкладке с наименованием **HTML**, но нас будет интересовать вкладка **Web Forms** (Web-формы). Дело в том, что элементы, расположенные на этой вкладке, несмотря на то, что они во многом дублируют элементы вкладки **HTML**, на самом деле имеют много большую функциональность. Они обладают собственными событиями и расширенным набором свойств. По сути дела, с использованием этих элементов **Web Forms** (Web-формы) мы можем разрабатывать свои Web-страницы точно в таком же стиле, как мы разрабатывали свои обычные исполняемые приложения в любой другой RAD-среде, такой, как Visual Basic, Visual C++ или Delphi.

Теоретически, для написания простейшего приложения, приветствующего окружающий мир, достаточно было воспользоваться одним компонентом `Label`, который отображает некий текст. Однако, это слишком тривиально. Мы попробуем еще и представиться окружающему миру. Для разработки подобного приложения нам потребуется одно поле текстового ввода (элемент с наименованием `TextBox`), надпись, приглашающая пользователя ввести свое имя, кнопка, подтверждающая окончание ввода текста (элемент `Button`), и еще одна надпись (элемент `Label`), при помощи которой мы и будем приветствовать мир. Все эти элементы надо просто перенести мышью с инструментальной панели на разрабатываемую страницу и расположить их в нужном порядке. После этого следует установить значения свойства `Text`, которые будут отображаться на элементах. Естественно, для элементов `TextBox1` и `Label2` это свойство будет пустым. Внешний вид окна разработки Web-страницы после размещения всех компонент показан на рис. 3.4.

Сама страница тоже является объектом, который носит наименование `DOCUMENT`. Мы можем установить для него значение свойства `title`, которое будет отображаться в заголовке основного окна программы-браузера.

Теперь, после того, как мы разместили все необходимые компоненты, следует написать обработчик нажатия пользователем кнопки. Нам необходимо, чтобы после нажатия на кнопку было сформировано значение свойства `Text` элемента `Label2` с учетом того текста, который был внесен пользователем в поле текстового ввода. Для создания обработчика события достаточно произвести двойной щелчок мышью на кнопке, нажатие на которую мы собираемся обрабатывать. После этого Visual Studio .NET сгенерирует новый файл с наименованием WebForm1.aspx.vb, в котором будет находиться сам код ASP.NET-приложения. Также курсор автоматически будет установлен на функцию, обрабатывающую нажатие пользователем нашей кнопки. Нам останется лишь записать следующую строку кода:

```
Label2.Text="Здравствуй, мир! Меня зовут "+TextBox1.Text
```

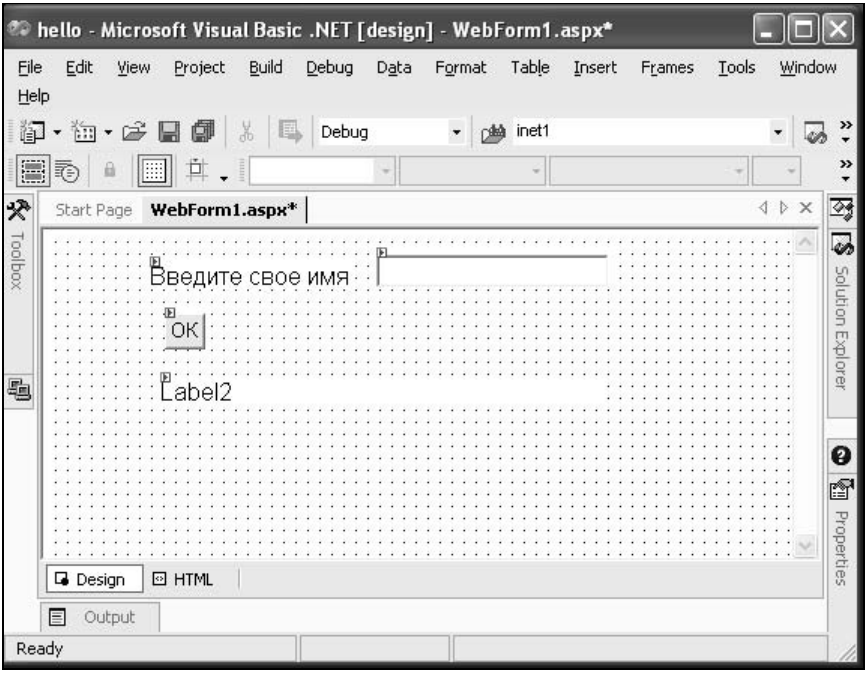


Рис. 3.4. Внешний вид основного окна разработки Web-страницы с размещенными компонентами

Фактически в этом окне объявляется класс, который соответствует нашему ASP.NET-приложению, действующему на странице WebForm1.aspx. Полный вид этого объявления приведен в листинге 3.1.

Листинг 3.1

```
Public Class WebForm1
    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 Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Label2 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
        Label2.Text = "Здравствуй, мир! Меня зовут " + TextBox1.Text
    End Sub
End Class
```

Из листинга легко заметить, что был создан не только обработчик нажатия пользователем на кнопку. Visual Studio .NET также автоматически создал обработчик события загрузки страницы. В нем мы могли бы разместить код, который бы выполнялся сразу после окончания загрузки страницы, но перед ее отображением в окне браузера. В данном случае нам это не нужно. Сам HTML-код созданной нами страницы приведен в листинге 3.2.

Листинг 3.2

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="Hello.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Здравствуй, мир</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: 18px;
                                 POSITION: absolute; TOP: 23px"
              runat="server">Введите свое имя</asp:Label>
   <asp:TextBox id="TextBox1" style="Z-INDEX: 102; LEFT: 168px;
                                 POSITION: absolute; TOP: 21px"
              runat="server"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 103; LEFT: 139px;
                                 POSITION: absolute; TOP: 67px"
              runat="server" Text="OK"></asp:Button>
   <asp:Label id="Label2" style="Z-INDEX: 104; LEFT: 18px;
                                 POSITION: absolute; TOP: 114px"
              runat="server" Width="369px" Height="19px"></asp:Label>
  </form>
 </body>
</HTML>
```

Как видно, все созданные нами органы управления на странице были помещены в форму с параметром `runat`, который явно не входит в спецификацию HTML. Это и есть тег, обрабатываемый ASP-анализатором IIS. Также стоит обратить внимание на первую строку документа, заключенную в брекеты `<%` и `%>`. Эта строка и является опознавательным знаком, что страница является частью ASP.NET-приложения. Легко заметить, что в ее состав входит параметр `Codebehind`, значение которого указывает, в каком файле размещается код ASP.NET-приложения, связанного с этой страницей.

Итак, после того как мы создали обработчик нажатия кнопки пользователем, процесс разработки закончен. Осталось лишь откомпилировать наш проект. Для этого стоит выполнить команду меню **File | Build and Browse** (Файл | Сборка и просмотр), дублируемую комбинацией клавиш <Ctrl>+<F8>. После этого созданное нами приложение будет скомпилировано и отображено в окне предварительного просмотра, которое является функциональным двойником браузера Internet Explorer. В этом же окне можно и проверить работоспособность приложения. Однако мы воспользуемся браузером Internet Explorer для того, чтобы увидеть действие нашего приложения. Если мы не изменяли наименование основной страницы и не устанавливали наименование локального хоста в IIS, оставив стандартное наименование localhost, то URL нашего первого ASP.NET-приложения будет выглядеть следующим образом: **http://localhost/Hello/WebForm1.aspx**.

Естественно, перед тем как просматривать Web-страницу с этим URL, следует все-таки откомпилировать наше приложение. На рис. 3.5 показано, как функционирует наше первое приложение ASP.NET в браузере Internet Explorer версии 6.0. При этом показан завершающий этап работы приложения, когда пользователь уже ввел свое имя в текстовое поле и нажал кнопку.

Что ж, наше первое приложение ASP.NET оказалось не так уж трудно разработать. Однако в реальной жизни нам, скорее всего, не придется создавать такие простые приложения. Все будет намного сложнее. А значит, и материала для изучения у нас еще предостаточно.

HTML-дизайн

Естественно, нет смысла использовать только элементы **Web Forms** для создания Web-страниц. Если нет нужды обрабатывать некие элементы в сценариях ASP.NET, всегда можно воспользоваться обычными элементами HTML, размещенными на одноименной вкладке панели **Toolbox** (Инструментарий). В этом разделе мы рассмотрим возможности дизайна Web-страниц с использованием чистого HTML.

Следует также помнить, что все элементы оформления Web-страниц, даже не входящие в состав **Web Forms**, в терминологии Visual Studio .NET являются объектами со своими свойствами и методами. При помощи свойств задаются параметры тегов, реализующих тот или иной элемент Web-страницы и их значения. Конечно, нет нужды рассматривать досконально весь список свойств и методов, на это есть оперативная справка, но вот основные и наиболее часто используемые методы мы рассмотрим.

Все, что в Web-странице находится между тегами `<BODY>` и `</BODY>` является объектом `DOCUMENT`. Часть свойств этого объекта отображена в окне **Properties** (Свойства). В этом окне собраны свойства, управляющие параметрами тега `<BODY>`.

- `aLink`. Позволяет устанавливать цвет, которым будут отображаться активные гиперссылки на Web-странице.
- `aspCompat`. Данное свойство имеет всего два допустимых значения — `True` и `False`, т. е. это логическое свойство. Если разработчик использовал значение `True`, то код данного Web-приложения будет выполняться по правилам технологии ASP (а не ASP.NET), что позволит использовать в работе приложения компоненты, разработанные до внедрения ASP.NET, т. е. унаследованные из предыдущих проектов.
- `background`. Значение данного свойства содержит URL графического изображения, которое будет использоваться в качестве фона для Web-страницы.

- `bgColor`. Свойство задает цвет фона страницы. По умолчанию цвет вводится как комбинация трех двузначных шестнадцатеричных чисел. Впрочем, можно воспользоваться специализированным диалоговым окном установки цвета **Color Picker** (Указатель цвета), которое активизируется при помощи дополнительной кнопки, размещенной в поле ввода значения свойства. Внешний вид этого диалогового окна показан на рис. 3.5. Легко заметить, что данное диалоговое окно содержит четыре вкладки. Разработчик может выбрать цвет одним из четырех способов. На вкладке **Web Palette** (Web-палитра) размещены цвета, входящие в стандартную палитру Web. Вкладка **Named Colors** (Наименования цветов) предлагает разработчику выбрать один из цветов, для которых установлены стандартные символьные наименования. Помимо стандартных шестнадцати поименованных цветов на вкладке находятся еще пятьдесят два цвета, для которых также введены символьные наименования. Следует помнить, что далеко не все браузеры могут использовать дополнительные символьные наименования цветов. Вкладка **System Colors** (Системные цвета) содержит цвета, определяемые установками операционной системы (например, цвет заголовка активного окна — `ActiveCaption`). В том случае, если разработчик не нашел необходимого ему цвета на первых трех вкладках, всегда можно воспользоваться вкладкой **Custom Color** (Настройка цвета), и при помощи трех ползунков, регулирующих насыщенность трех основных цветов (красный, зеленый, синий), установить нужный цвет.

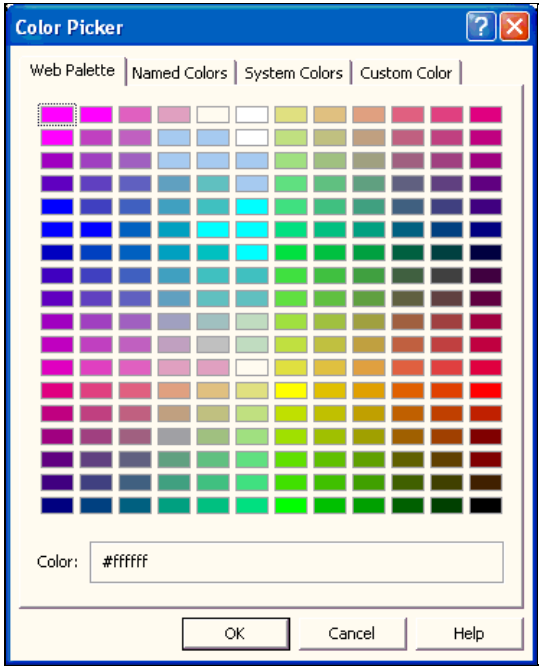


Рис. 3.5. Диалоговое окно **Color Picker**

- `bgProperties`. Свойство задает параметры отображения фонового рисунка, а именно, указывает, будет ли изображение двигаться вместе с содержимым страницы при прокрутке с помощью скроллеров или будет зафиксировано в окне просмотра.
- `bottomMargin`. Свойство устанавливает размеры нижнего поля Web-страницы.
- `charset`. Свойство задает наименование кодировки символов, которая будет использоваться браузером при отображении документа. Учитывая тот факт, что приложение ASP.NET будет выполняться под управлением сервера IIS, и для адекватного отображения наиболее продвинутых возможностей оформления Web-страниц пользователю потребуется использовать браузер Internet Explorer, разработчику для создания русскоязычных Web-страниц имеет смысл использовать кодировку Cyrillic (Windows).
- `codeBehind`. Свойство содержит наименование файла, в котором находится код ASP.NET-приложения, связанного с данной Web-страницей. Разработчик не имеет возможности напрямую изменять значение этого свойства в окне **Properties** (Свойства).
- `contentType`. Свойство задает тип содержимого Web-страницы, указываемый в заголовке протокола HTTP.
- `debug`. Логическое свойство. Его значение указывает, надлежит ли среде разработки внедрять в разрабатываемую Web-страницу специальные символьные последовательности, позволяющие производить отладку ASP.NET-приложения.
- `defaultClientScript`. Данное свойство указывает, какой именно язык сценариев будет использоваться разработчиком данной страницы для их исполнения на стороне пользователя. Свойство имеет всего два допустимых значения — `JScript` и `VBScript`. Смысл этих значений достаточно прозрачен и не требует дополнительного разъяснения.
- `Description`. Свойство содержит строку с кратким описанием разрабатываемой Web-страницы. Значение этого свойства помещается в тег `<META>` с соответствующими параметрами и позволяет хранить дополнительную информацию, обрабатываемую поисковыми машинами.
- `dir`. Свойство указывает направление чтения и, соответственно, вывода текста. Для большинства языков, в которых чтение идет слева направо, используется значение `ltr`. В ином случае будет использовано значение `rtl`.
- `enableSessionState`. Логическое свойство, указывающее, будет ли данная Web-страница поддерживать механизм сеансов работы пользователей. Более подробно об этой технологии сеансов работы (также называемых сессиями) будет рассказано в соответствующем разделе главы.
- `errorPage`. Свойство содержит в качестве своего значения URL-страницу, которая будет посылаться пользователю в случае возникновения каких-либо

ошибок или необработанных приложением исключений (что, в сущности, одно и то же).

- `keywords`. Свойство содержит список ключевых слов, характеризующих содержимое разрабатываемой Web-страницы. Значение этого свойства также может использоваться поисковыми машинами.
- `language`. Значение этого свойства указывает компилятору, какой язык использовался программистом при разработке приложения ASP.NET, связанного с данной Web-страницей.
- `leftMargin`. Данное свойство устанавливает размер левого поля Web-страницы.
- `link`. Свойство задает цвет, которым будут отображаться не посещенные еще пользователем гиперссылки.
- `pageLayout`. Свойство задает порядок позиционирования элементов Web-страницы в окне просмотра. По умолчанию используется значение `GridLayout`, которое создает Web-страницы с использованием абсолютного позиционирования. К сожалению, директивы абсолютного позиционирования внедряются напрямую в HTML-код Web-страницы без использования технологии CSS, поэтому в некоторых браузерах, таких, как старые версии Netscape Communicator, позиционирование может не работать, и страница будет отображена неадекватно. Если же разработчик планирует использовать стандартную "мягкую" верстку Web-страницы, при которой место каждого элемента рассчитывается браузером на основе размеров окна просмотра и внутренней структуры страницы, следует воспользоваться значением `FlowLayout`.
- `responseEncoding`. Свойство задает кодировку символов, которая будет использована сервером для создания Web-страницы, сгенерированной в ответ на действия пользователя с текущей страницей.
- `rightMargin`. Свойство устанавливает размер правого поля разрабатываемой Web-страницы.
- `showGrid`. Свойство логического типа, которое указывает, будет ли отображаться сетка позиционирования в режиме разработки дизайна страницы. Естественно, данное свойство никак не влияет на внешний вид Web-страницы в браузере пользователя. Свойство может иметь только два значения — `True` или `False`. Впрочем, для отображения или сокрытия сетки разметки в режиме дизайна Web-страницы при ее разработке всегда можно использовать кнопку **Show Grid** (Вывести сетку) на инструментальной панели **Formatting** (Разметка).
- `targetSchema`. Свойство указывает, какой именно вариант стандарта языка HTML будет использован при разработке Web-страниц. Соответственно, можно указать целевой браузер для Web-страниц. Свойство может принимать одно из трех предустановленных значений — `Internet`

Explorer 3.02/Navigator 3, Internet Explorer 5.0, используемое по умолчанию, и Navigator 4.

- text. Свойство задает цвет, которым по умолчанию будет отображаться текст на разрабатываемой Web-странице.
- title. Свойство содержит в качестве значения текстовую строку, которая будет использована как наименование Web-страницы, отображаемое в строке заголовка программы-браузера.
- topMargin. Свойство устанавливает размер верхнего поля разрабатываемой Web-страницы.
- transaction. Свойство указывает, будет ли поддерживать создаваемая Web-страница механизм транзакций. Свойство может принимать одно из нескольких предустановленных значений. Значение NotSupported означает, что данная страница не будет поддерживать механизм транзакций вообще. Значение Supported указывает, что страница может работать с транзакциями. Если разработчику необходимо, чтобы страница работала только в режиме обработки транзакций, следует использовать значение Required. В том случае, если при отображении страницы необходимо создавать новую транзакцию, стоит обратить внимание на значение RequiresNew. Если Web-страница входит в состав последовательности страниц, обрабатывающих транзакцию, но сама не должна каким-либо образом менять ее состояние, следует установить значение Disabled.
- vlink. Свойство задает цвет, которым будут отображаться пройденные пользователем гиперссылки.

Впрочем, большую часть свойств, которые связаны со свойствами именно HTML-документа, разработчик может задать при помощи диалогового окна **DOCUMENT Property Pages** (Свойства документа), которое активизируется после выполнения команды меню **View | Property Pages** (Вид | Страницы свойств). Внешний вид этого окна показан на рис. 3.6.

Как видно, данное диалоговое окно содержит три вкладки. Вкладка **General** (Общие) содержит органы управления для задания общих свойств разрабатываемой Web-страницы. Все органы управления для установки различных цветов элементов оформления Web-страницы и полей документа собраны на вкладке **Color and Margins** (Цвет и границы). А ключевые слова, характеризующие содержимое Web-страницы, размещаются на вкладке **Keywords** (Ключевые слова).

Теперь перейдем к элементам, расположенным на вкладке **HTML** панели **Toolbox** (Инструментарий). Они, естественно, имеют несколько меньше свойств, нежели только что рассмотренный нами объект DOCUMENT, причем некоторые свойства присущи всем элементам, поэтому мы будем рассматривать только уникальные свойства для каждого компонента.

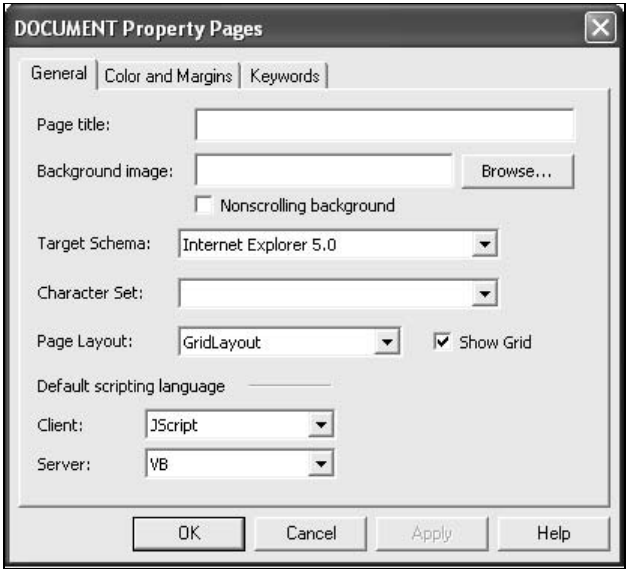


Рис. 3.6. Диалоговое окно **DOCUMENT Property Pages**

Компонент `Label` предназначен для размещения текста на разрабатываемой Web-странице. При этом в HTML-код страницы добавляется не тег нового абзаца `<P>`, как можно было бы ожидать, а новый блок, реализуемый тегом `<DIV>`. Следует также помнить, что в самом начале разработки Web-страницы Visual Studio.NET автоматически добавляет в HTML-код еще пустой страницы теги формы, внутри которой и будет располагаться все содержимое страницы. Естественно, разработчик не ограничен одной этой формой, создаваемой по умолчанию, и может вставить дополнительные формы в разрабатываемый документ, но об этом несколько позже.

Итак, компонент `Label` обладает следующим набором свойств, которые большей частью относятся к отображению данного текста.

- `BackColor`. Свойство задает цвет фона элемента.
- `BorderColor`. Свойство устанавливает цвет границы элемента, если она, конечно, имеет ненулевую ширину.
- `BorderStyle`. Свойство задает тип отображаемой границы элемента.
- `BorderWidth`. Свойство устанавливает ширину границы элемента.
- `CssClass`. Свойство содержит в качестве своего значения наименование класса-селектора CSS-таблицы, с которой связана создаваемая страница. Естественно, для того чтобы использовать данное свойство, необходимо создать и подключить CSS-таблицу.

- `Font`. Составное свойство, задающее шрифт, при помощи которого будет отображаться текстовое содержимое элемента. В его состав входят следующие подсвойства.
  - `Bold`. Логическое свойство, указывающее, будет ли применено полужирное начертание шрифта.
  - `Italic`. Логическое свойство, указывающее, будет ли применено курсивное начертание шрифта.
  - `Name`. Свойство задает наименование шрифта, которым будет отображаться текстовое содержимое элемента. В выборе значения данного свойства следует проявить разумный консерватизм, так как далеко не факт, что удаленный пользователь, просматривающий Web-страницы, входящие в разрабатываемый проект, будет обладать тем же причудливым набором редких шрифтов, которые долго собирал разработчик. Поэтому следует ориентироваться все-таки на стандартные шрифты, входящие в состав стандартных поставок максимально большого количества операционных систем.
  - `Names`. Свойство позволяет задавать наименования шрифтов, которые будут использованы браузером удаленного пользователя в тех случаях, когда шрифт, указанный разработчиком в предыдущем свойстве, все-таки не будет установлен в операционной системе пользователя.
  - `Overline`. Логическое свойство, указывающее, будет ли текстовое содержимое элемента дополнено горизонтальной линией, проходящей поверх символов шрифта.
  - `Size`. Свойство устанавливает размер применяемого шрифта. При этом используются символьные значения, унаследованные стандартом HTML из CSS.
  - `Strikeout`. Логическое свойство, указывающее, будет ли текстовое содержимое элемента перечеркнуто горизонтальной линией.
  - `Underline`. Логическое свойство, указывающее, будет ли текстовое содержимое элемента подчеркнуто.
- `ForeColor`. Значение устанавливает цвет символов шрифта, которым будет отображено текстовое содержимое элемента.
- `Text`. Свойство содержит текст, который и составляет содержимое элемента `Label`.
- `AccessKey`. Свойство устанавливает комбинацию клавиш, по нажатию которой фокус ввода будет перенесен на данный элемент.
- `EnableViewState`. Очень интересное свойство логического типа. Дело в том, что многие элементы могут запоминать свое состояние, измененное пользователем. Очень часто это свойство применяется для элементов форм и элементов группы **Web Forms** (Web-формы). Однако наличествует

и в группе свойств элемента `Label`. В том случае, если свойство установлено в `True`, то оно будет запоминать свои состояния и при перезагрузке страницы отображать себя именно так, как выглядело последний раз. То же самое относится и ко всем остальным элементам с подобным свойством.

- `TabIndex`. Свойство содержит номер элемента в последовательности элементов оформления Web-страницы, перевод фокуса ввода между которыми может осуществляться при помощи табуляции. В данном случае разработчик получает возможность обеспечить доступ к отдельным элементам, так же как и к органам управления обычных приложений Windows — при помощи клавиши табуляции. Для этого надо каждому подобному элементу установить целочисленное значение свойства `TabIndex`. Соответственно, смена фокуса ввода будет переходить по элементам с установленным явно этим свойством от наименьшего значения к наибольшему.
- `ToolTip`. Значение данного свойства будет высвечиваться в виде хинта-подсказки в браузере пользователя, когда тот наведет курсор мыши на элемент.
- `Visible`. Логическое свойство, которое указывает, будет отображаться данный элемент в браузере удаленного пользователя при просмотре Web-страницы, или нет.
- `Height`. Свойство задает высоту создаваемого элемента.
- `Weight`. Свойство устанавливает ширину элемента.
- `ID`. Свойство содержит уникальный идентификатор объекта, который является значением одноименного атрибута `id` в HTML-коде, и служит наименованием объекта для обращения к нему из кода приложения.

Элемент `Button` создает обычную кнопку на разрабатываемой Web-странице. При переносе данного компонента на Web-страницу в режиме дизайна, автоматически добавляется фрагмент HTML-кода следующего вида:

```
<input id="Button1" type="button" value="Button" name="Button1">
```

Таким образом, используется тег `<input>` с типом `button`. Чаще всего данный элемент оформления Web-страниц используется разработчиками для запуска пользователями каких-либо скриптов, выполняющихся на стороне пользователя. Но какими-либо особыми свойствами этот объект не обладает, поэтому его можно не рассматривать детально. Также необходимо осознавать, что данная кнопка не будет обрабатываться сервером IIS, поэтому попытка написать обработчик нажатия на кнопку пользователем приведет лишь к отображению предупреждения, что данный орган управления предназначен для обработки браузером, а не сервером. Для того чтобы перевести данный компонент в разряд элементов, обрабатываемых сервером, можно щелкнуть правой кнопкой мыши на элементе и в появившемся контекстном

меню выполнить команду **Run As Server Control** (Выполнять под управлением сервера).

Также Visual Studio.NET предоставляет возможность использовать кнопки **Reset** и **Submit**, которые реализуются при помощи все того же тега `<input>`, но при этом используются иные значения параметра `type`: `reset` и `submit`, соответственно. Надписи на всех трех видах кнопок задаются при помощи свойства `value`, а стиль отображения регулируется свойством `style`, в качестве значения которого используется набор инструкций CSS. Кнопка типа **Reset**, как мы знаем, обновляет содержимое всех органов управления, входящих в состав той же формы, что и сама кнопка, устанавливая для них те значения, которые были заданы разработчиком по умолчанию. Кнопка типа **Submit** позволяет пользователю отправить введенные в органы управления формы данные на сервер для их обработки.

Компонент с наименованием `Text Field` позволяет создавать однострочное поле символьного ввода. В HTML-коде оно реализуется уже знакомым нам тегом `<input>`, но теперь значение параметра `type` устанавливается в `text`. Компонент обладает некоторыми специфическими свойствами.

- `maxlength`. Свойство устанавливает максимальное количество символов, которые пользователь может ввести в поле.
- `readonly`. Свойство логического типа указывает, есть ли у пользователя возможность изменять информацию, находящуюся в этом поле. Если разработчик установил значение `True` для этого свойства, то текст, находящийся в поле, изменить будет нельзя.
- `size`. Свойство устанавливает видимый размер поля ввода в символах.

Поле текстового ввода имеет и две дополнительные модификации. Так, компонент `File Field` совмещает в себе поле текстового ввода и кнопку **Обзор**, при нажатии на которую пользователем активизируется стандартное диалоговое окно операционной системы для выбора файла. Файл, который выбрал пользователь, обычно пересылается на сервер для обработки. А компонент `Password Field` фактически является функциональным двойником обычного поля текстового ввода, за исключением того, что символы, вводимые пользователем, отображаются в виде звездочек, чтобы никто не смог увидеть вводимых данных. Естественно, подобная модификация поля текстового ввода чаще всего применяется для запроса различных паролей.

Для того чтобы предоставить пользователю возможность вводить достаточно большие объемы текста, разработчик может воспользоваться компонентом `Text Area`, который позволяет вводить несколько текстовых строк. Внешний вид этого элемента регулируется следующим набором свойств:

- `cols`. Свойство задает ширину поля ввода в символах;
- `disabled`. Свойство логического типа указывает, будет ли доступен данный орган управления пользователю. Проще говоря, сможет ли он получить фокус ввода;

- `rows`. Свойство задает высоту поля ввода в символах, т. е. количество одновременно отображаемых строк в поле;
- `wrap`. Свойство устанавливает способ разбиения строк, не умещающихся в поле ввода полностью.

Элемент `Checkbox` является реализацией независимого переключателя. Представляет собой всего лишь позицию для установки или снятия флажка в виде галочки. Естественно, правильнее всего будет комбинировать его с элементом `Label`, чтобы пользователь мог все-таки увидеть назначение переключателя.

- `checked`. Свойство устанавливает начальное состояние переключателя, будет ли установлен в нем флажок или нет. Свойство логическое, поэтому разработчик может использовать только два значения — `True` или `False`.
- `name`. Свойство устанавливает наименование органа управления, которое будет передано на сервер для обработки. Именно по этим именам обрабатывающая программа и будет различать органы управления.
- `value`. Значение этого свойства будет передано на сервер в том случае, если пользователь установит флажок в данном независимом переключателе.

Для того чтобы создать группу зависимых друг от друга переключателей, или, как их еще называют — кнопок выбора, следует воспользоваться соответствующим компонентом с наименованием `RadioButton`. Для того чтобы мы могли объединить эти компоненты в одну группу, необходимо установить для всех кнопок переключателей одно и то же значение свойства `name`. Естественно, встает вопрос, а как программа, обрабатывающая данные, переданные пользователем, будет различать, какая именно кнопка была выбрана пользователем. Для этого используется свойство `value`. В этом случае обрабатывающей программе будет передано наименование группы кнопок переключателей и значение свойства `value` той кнопки, которую выбрал пользователь.

Компонент `Hidden` не отображается на Web-странице, когда ее в браузере просматривает удаленный пользователь. Обычно это поле используется для хранения и передачи некоей дополнительной информации, которую пользователь не должен видеть. Раньше это скрытое поле использовали в разветвленных Web-приложениях для хранения значений различных переменных, которые должны были быть доступны нескольким формам сразу. Этакий аналог глобальных переменных. Конечно, нельзя в подобных полях хранить секретную информацию, так как Web-страницы могут сохраняться в кэше, и при просмотре их кода пользователь сможет увидеть содержимое скрытого поля. На самом деле в приложениях ASP.NET нет нужды использовать скрытое поле подобным образом, так как поддержка глобальных переменных

уже встроена в ASP.NET. Более того, очень часто эти переменные использовались для поддержки механизма сеансов работы удаленных пользователей. А в ASP.NET этот механизм реализован более изящно, и использовать для этой цели глобальные переменные теперь нет нужды.

Еще одним компонентом, часто включаемым в формы, является выпадающий список, реализуемый при помощи элемента `Dropdown`. Элементы списка задаются при помощи диалогового окна **<SELECT> Property Pages** (Страницы свойств <Выбор>), которое активизируется при выполнении команды **Properties** (Свойства) контекстного меню. Внешний вид этого диалогового окна показан на рис. 3.7.

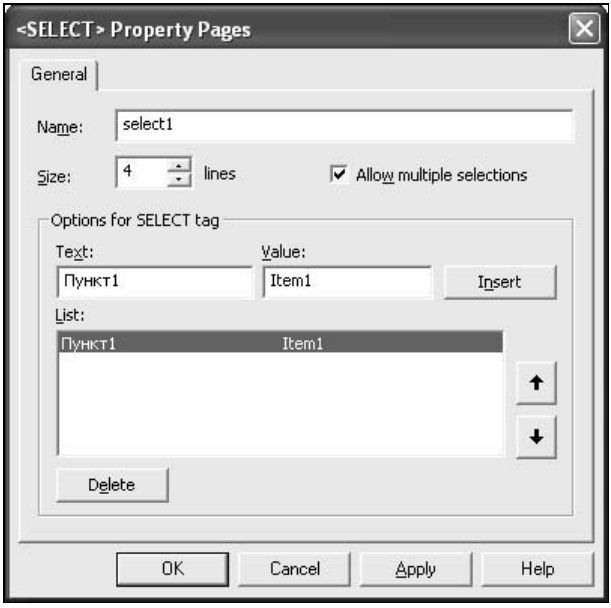


Рис. 3.7. Диалоговое окно **<SELECT> Property Pages**

Элементы выпадающего списка задаются при помощи органов управления, объединенных в группу **Options for SELECT tag** (Элементы выпадающего списка). Каждому элементу списка соответствует некое значение, которое будет передано обрабатывающей программе на Web-сервер, когда пользователь завершит ввод данных. Текст элемента списка указывается в поле ввода **Text** (Текст), а соответствующее ему значение — в поле **Value** (Значение). Затем введенная пара "имя-значение" вносится в состав списка при помощи кнопки **Insert** (Вставить). Порядок отображения элементов регулируется при помощи двух кнопок с изображением стрелок, направленных вверх и вниз. Эти кнопки перемещают соответственно вверх и вниз по списку выделенный элемент. Удаление какого-либо элемента из списка осуществляется, как нетрудно догадаться, при помощи кнопки **Delete** (Удалить). Наименование

самого выпадающего списка, которое будет передано в обрабатывающую программу на сервере, задается в поле текстового ввода **Name** (Имя). Количество строк, отображаемых браузером при нажатии на кнопку, раскрывающую список, указывается в поле **Size** (Размер). Правда, в этом случае выпадающий список может потерять свою привычную форму и превратиться в обычный список с полосами прокрутки. Подобный список также может создаваться при помощи компонента `Listbox`. А в том случае, если разработчик готов предоставить пользователю возможность выбора не одного, а нескольких элементов выпадающего списка сразу, следует установить флажок в независимом переключателе **Allow multiple selections** (Разрешить множественный выбор).

При использовании диалогового окна настройки свойств выпадающего списка может создаться впечатление, что обычное штатное окно свойств **Properties** (Свойства) уже не может предоставить каких-либо новых возможностей. Однако это не так. В нем присутствует свойство `SelectedIndex`, которое указывает номер элемента, выбранного по умолчанию. Кстати, когда пользователь выберет свой элемент из выпадающего списка, соответствующим образом изменится и значение этого свойства.

Таблицы являлись одним из самых часто используемых инструментов в дизайне Web-страниц. Дело в том, что именно эти компоненты могли создать некое подобие верстки. Теперь, если мы используем абсолютное позиционирование элементов Web-страниц, нет нужды использовать хитрые приемы верстки при помощи таблиц. Однако далеко не факт, что разработчик всегда будет иметь возможность использовать абсолютное позиционирование, поэтому забывать о таблицах не стоит.

Вставка таблицы на проектируемую Web-страницу производится либо при помощи компонента `Table`, либо при помощи команды **Table | Insert | Table** (Таблица | Вставить | Таблица). По умолчанию создается таблица из трех строк и трех колонок. Однако всегда есть возможность настроить внешний вид настолько тонко, насколько это позволяет технология HTML. Естественно, все параметры отображения таблицы задаются при помощи ее свойств, которые нам предстоит рассмотреть.

- `align`. Свойство устанавливает горизонтальное выравнивание содержимого ячеек таблицы. В качестве значения свойства разработчик может использовать одно из трех ключевых слов: `left`, `center` и `right`, которые прижимают содержимое ячеек к левому краю ячеек, центрируют его или прижимают к правому краю, соответственно.
- `background`. В качестве значения данного свойства используется URL графического файла, который будет использован в качестве фона таблицы.
- `bgcolor`. Свойство задает цвет фона таблицы. Естественно, не имеет смысла использовать это свойство одновременно с предыдущим.

- `bordercolor`. Свойство задает цвет границ ячеек и таблицы в целом.
- `bordercolordark`. Свойство используется для создания иллюзии трехмерной границы таблицы. Для этого граница таблицы разделяется на две части. Верхняя левая половина отображается светлым цветом, а нижняя правая — темным. При этом подобное разделение относится и к ячейкам. Но у ячеек темная и нижняя границы меняются местами. Данное свойство устанавливает цвет темной половины границы. Впрочем, разработчик может сделать эту границу светлой. Выбор цвета полностью в его власти.
- `bordercolorlight`. Свойство устанавливает цвет светлой стороны границы.
- `cellpadding`. Свойство задает отбивку ячейки, т. е. расстояние между содержимым ячейки и ее границами. В качестве значения свойства используется целое число, обозначающее количество пикселов.
- `cellspacing`. Свойство предназначено для установки отступа ячеек, т. е. расстояния между самими ячейками.
- `height`. Свойство задает высоту таблицы. На самом деле, изначально высота таблицы рассчитывается на основе ее содержимого, как сумма высот всех столбцов. А высота каждого столбца есть высота его самой высокой ячейки. Однако разработчик при помощи данного свойства может сам установить высоту таблицы. Если это значение будет больше, чем высота таблицы, рассчитанная браузером на основе ее содержимого, то будет использоваться значение, установленное разработчиком.
- `width`. Свойство задает ширину таблицы. Как и в случае с высотой, это свойство имеет вес только тогда, когда превышает ширину, рассчитанную на основе содержимого ячеек. При этом в отличие от высоты таблицы, которую разработчик может задавать лишь в пикселах, ширина может указываться как в пикселах, так и в процентах.

На самом деле необязательно устанавливать все свойства таблицы в окне **Properties** (Свойства). Если щелкнуть по выделенной таблице правой кнопкой мыши, и в появившемся контекстном меню выполнить команду **Properties** (Свойства), то будет активизировано диалоговое окно **<TABLE> Property Pages** (Свойства Таблицы), показанное на рис. 3.8. В этом окне можно установить все параметры отображения, что называется, в визуальном режиме.

Все параметры отображения, установленные для таблицы, распространяются на ее отдельные строки и ячейки. Однако если разработчик установит другие параметры для каких-либо ячеек или строк, то вновь установленные свойства перекроют более общие свойства таблицы. Другими словами, свойства для более низких объектов в табличной иерархии имеют несколько больший вес.

Рис. 3.8. Диалоговое окно **<TABLE> Property Pages**

Однако только на уровне ячеек таблиц можно задать некоторые дополнительные свойства отображения их содержимого. Для этого следует перевести курсор именно в ту ячейку, свойства которой необходимо установить, и перейти к встроенному окну редактирования свойств объекта **Properties** (Свойства). Естественно, мы рассмотрим те свойства, которые типичны для ячеек, и доступ к которым на уровне всей таблицы не предусмотрен.

- `colspan`. Свойство используется, если одна ячейка будет объединять в себе несколько ячеек из соседних столбцов, но в пределах одной строки. По сути дела, значение этого свойства указывает, сколько бы ячеек было вместо искомой ячейки, если бы она не объединяла столбцы.
- `nowrap`. Логическое свойство, указывающее, может ли браузер разбивать текстовое содержимое ячейки на несколько строк, если в исходном виде оно превышает горизонтальные размеры ячейки.
- `rowspan`. Свойство весьма похоже на только что рассмотренное свойство `colspan`. Но при этом указывается, сколько объединяется ячеек по вертикали.
- `valign`. Свойство регулирует выравнивание содержимого ячеек по вертикали. В качестве значения данного свойства может быть использовано одно из предустановленных ключевых слов: `baseline`, `bottom`, `middle` и `top`, которые выравнивают содержимое ячейки по базовой линии шрифта, прижимают содержимое к верхней границе ячейки, центрируют по вертикали и прижимают к нижнему краю ячейки соответственно.

Также нам доступны методы объединения соседних элементов Web-страницы в единые блоки. Все элементы оформления HTML-документов разделяются на два типа: inline-элементы, которые чаще всего являются просто элементами текста, и блочные элементы. Inline-элементы могут являться частью строки, а блочные элементы всегда занимают обособленное место на Web-странице, и обязаны начинаться всегда с новой строки. Естественно, блочные элементы могут включать в себя другие блочные элементы и inline-элементы. По вполне понятным причинам inline-элементы не могут включать в себя блочные элементы.

Объединение элементов Web-страницы в блоки позволяет применять к ним единое оформление, осуществлять некое подобие верстки. Достаточно будет изменить расположение блока, изменив один объединяющий тег. Естественно, это удобнее, чем менять расположение каждого элемента Web-страницы по отдельности.

Для объединения элементов блочного типа используется тег с его закрывающим близнецом . А для inline-элементов используется пара тегов и . Учитывая вышесказанное, ясно, что блок с тегом не может располагаться внутри блока с тегом , так как блочные элементы не могут входить в состав inline-элементов.

Visual Studio .NET предоставляет разработчику возможность использовать подобные блочные инструкции. Microsoft, естественно, предпочитает блоки, реализуемые тегом . Элемент Label опирается именно на этот тег. Однако в элементе Label нельзя группировать иные элементы оформления Web-страницы. Поэтому в палитре компонентов HTML находятся еще два элемента, создающие подобные блоки-контейнеры. И называются они практически одинаково — Flow Layout Panel и Grid Layout Panel. Более того, по набору своих свойств они также идентичны. В чем же разница между ними? Дело в том, что элемент Flow Layout Panel применяется в том случае, когда требуется использовать стандартное расположение элементов без применения абсолютной модели позиционирования. В тех случаях, когда разработчик планирует применять подобную модель позиционирования внутри блока-контейнера, следует использовать компонент Grid Layout Panel. А потом уже внутри этих контейнеров разработчик сможет помещать различные элементы оформления Web-страниц.

Компонент с наименованием Image позволяет размещать на разрабатываемых Web-страницах графические изображения и видеоклипы. Естественно, основные параметры данного компонента задаются при помощи все тех же свойств.

- alt. Свойство задает текст, который будет отображаться в виде хинта-подсказки при наведении пользователем курсора мыши на загруженную и отображенную браузером картинку. В том случае, если по каким-либо причинам искомое графическое изображение не будет загружено, напри-

мер, из-за неверной ссылки на него или установок браузера, данная текстовая строка будет отображена вместо картинки или видеоклипа.

- `border`. Свойство устанавливает ширину границы, отображаемой вокруг рисунка или области воспроизведения видеоклипа. Значением данного свойства является неотрицательное целое число, указывающее толщину границы в пикселах.
- `controls`. Свойство логического типа, которое применяется разработчиком в тех случаях, когда на Web-страницу внедряется не простое графическое изображение, а видеоклип. Данное свойство указывает, будут ли вместе с видеоклипом отображаться органы управления его воспроизведением, а именно, кнопки запуска воспроизведения и паузы, а также бегунок, устанавливающий текущую позицию воспроизведения.
- `dynsrc`. Свойство применяется для внедрения в состав Web-страницы видеоклипов. Значением свойства является URL воспроизводимого видеофайла. Естественно, разработчикам предоставляется возможность вставлять файлы с расширениями .avi, .asf и .mpeg, т. е. именно те типы, с которыми работает Windows Media Player. Хотя на самом деле браузеры могут воспроизводить видеофайлы и других форматов, если в системе удаленного пользователя установлены соответствующие видеопроигрыватели.
- `loop`. Это свойство, как и два предыдущих, используется при работе с видеоклипами. Оно задает количество повторов воспроизведения видеоклипа после того, как будет полностью загружен соответствующий видеофайл.
- `lowsrc`. Данное свойство применяется при работе с графическими изображениями. Оно указывает на облегченную версию картинки, которая будет использована браузером в тех случаях, когда скорость работы с Интернетом достаточно низка, и придется слишком долго загружать основное изображение. В качестве значения свойства, естественно, используется URL, указывающий на то самое "облегченное" изображение.
- `src`. Свойство содержит URL, указывающий на графический файл, в котором хранится искомое изображение, предназначенное для отображения на Web-странице.
- `usemap`. Свойство используется, если данное графическое изображение является одновременно местом для размещения одной или нескольких гиперссылок. В качестве значения свойства используется URL.
- `vrml`. Свойство применяется для внедрения на Web-страницу не просто изображения, а сцены виртуальной реальности, созданной при помощи языка VRML (Virtual Reality Modeling Language).
- `hspace`. Свойство устанавливает размер отступа от границы графического изображения до окружающих его остальных элементов оформления Web-

страницы по вертикали. Значением свойства является целое число, указывающее размер отступа в пикселах.

- `vspace`. Свойство устанавливает размер отступа от границы графического изображения до окружающих его остальных элементов оформления Web-страницы по горизонтали. Значением свойства является целое число, указывающее размер отступа в пикселах.

И последним элементом, который мы еще не рассмотрели на вкладке **HTML**, является элемент `Horizontal Rule`, который позволяет размещать на Web-странице горизонтальную линию, являющуюся штатным средством HTML для визуального разделения различных блоков информации. Подобная линия реализуется при помощи тега `<hr>`. Естественно, этот элемент также обладает некоторыми специфичными свойствами.

- `align`. Указывает горизонтальное позиционирование линии. В качестве значения свойства используется одно из трех ключевых слов — `center`, `left` или `right`, которые центрируют линию, или прижимают ее к левому или правому краю окна просмотра браузера, соответственно.
- `color`. Свойство задает цвет отображаемой горизонтальной линии.
- `noshade`. Логическое значение, указывающее, будет ли горизонтальная линия иметь тень, или нет. Если пользователь использует значение `False`, то вместе с линией будет отображена и ее тень.
- `size`. Свойство задает толщину отображаемой горизонтальной линии в пикселах.
- `width`. Задает длину горизонтальной линии. Длина может указываться как в процентах, так и в пикселах.

На этом перечень компонентов вкладки **HTML** заканчивается. Они чаще всего используются просто для оформления Web-страниц, а все основные действия Web-приложений связаны с компонентами вкладок **Web Forms** (Web-формы) и **Components** (Компоненты), которые мы будем рассматривать в следующих разделах этой главы.

Элементы *Web Forms*

Компоненты, размещенные на вкладке **Web Forms** (Web-формы), во многом совпадают по наименованиям и внешнему виду с компонентами вкладки **HTML**. Основное различие между ними заключается в том, что компоненты Web Forms предназначены для работы с сервером, который может изменять их свойства и обрабатывать события, инициируемые этими компонентами. Естественно, достаточно большую часть свойств данные компоненты наследуют от своих обычных предшественников с вкладки **HTML**. Однако существуют и дополнительные свойства, которые регулируют различные рабочие

состояния этих элементов и порядок обработки и передачи информации. Сначала мы просто перечислим компоненты Web Forms (Web-формы) с их кратким описанием, а затем уже перейдем к детальному рассмотрению тех или иных свойств.

- Компонент `Label`. Полное наименование класса, который реализует данный компонент — `System.Web.UI.WebControls.Label`. Предназначен только для отображения текста.
- Компонент `TextBox`. Полное наименование его класса — `System.Web.UI.WebControls.TextBox`. Это обычное однострочное поле текстового ввода.
- Компонент `Button`. Создает кнопку. Полное наименование класса — `System.Web.UI.WebControls.Button`. При работе с компонентами Web Forms (Web-формы) разработчику не нужны специализированные типы кнопок, такие как **Reset** и **Submit**, так как реакцию приложения на нажатие подобных кнопок он определяет самостоятельно.
- Компонент `LinkButton`. Полное наименование класса — `System.Web.UI.WebControls.LinkButton`. Данный компонент при отображении его браузером выглядит как обычная гиперссылка, однако функционально он полностью совпадает с компонентом `Button`.
- Компонент `ImageButton`. Полное наименование класса — `System.Web.UI.WebControls.ImageButton`. Выглядит как обычное графическое изображение, однако на деле является кнопкой.
- Компонент `HyperLink`. Предназначен для создания гиперссылок на разрабатываемой Web-странице. Полное наименование класса — `System.Web.UI.WebControls.HyperLink`.
- Компонент `DropDownList`. Позволяет пользователю ввести свое текстовое значение или выбрать его из выпадающего текстового списка. Полное наименование класса — `System.Web.UI.WebControls.DropDownList`.
- Компонент `ListBox`. Полное наименование класса — `System.Web.UI.WebControls.ListBox`. Позволяет пользователю выбрать одно или несколько значений из списка.
- Компонент `DataGrid`. Достаточно объемный и сложный компонент, который позволяет отображать информацию в табличном виде. Полное наименование класса — `System.Web.UI.WebControls.DataGrid`.
- Компонент `DataList`. Предназначен для отображения информации, получаемой из набора данных, обычно из подключенной базы данных. Подобен таблице, но применяет свои методы отображения различных элементов. Полное наименование класса — `System.Web.UI.WebControls.DataList`.

- Компонент `Repeater`. Предназначен для отображения повторяющихся элементов оформления Web-страницы. Установка параметров отображения производится при помощи тегов HTML, т. е. визуальных средств установки параметров отображения у данного компонента нет. Полное наименование класса — `System.Web.UI.WebControls.Repeater`.
- Компонент `CheckBox`. Создает единичный независимый переключатель. В отличие от своего предшественника с палитры HTML, данный компонент содержит помимо самого переключателя еще и текст, идентифицирующий его. Полное наименование класса — `System.Web.UI.WebControls.CheckBox`.
- Компонент `CheckBoxlist`. Предназначен для создания набора независимых переключателей. Доступ к отдельным переключателям этого компонента осуществляется при помощи свойства `Items`, которое на самом деле является обычной коллекцией. Полное наименование класса — `System.Web.UI.WebControls.CheckBoxList`.
- Компонент `RadioButton`. Предназначен для создания кнопки-переключателя. Обычно в одиночном виде не используется. Полное наименование класса — `System.Web.UI.WebControls.RadioButton`.
- Компонент `RadioButtonList`. Предназначен для создания группы переключателей. Полное наименование класса, реализующего этот компонент — `System.Web.UI.WebControls.RadioButtonList`.
- Компонент `Image`. Позволяет внедрять в состав содержимого Web-страницы графическое изображение. Полное наименование класса — `System.Web.UI.WebControls.Image`.
- Компонент `Panel`. Создает на разрабатываемой Web-странице блок-контейнер, в котором могут быть размещены другие элементы. По сути, является некоторым аналогом компонентов `Layout Panel`, находящихся на вкладке **HTML**. Полное наименование класса — `System.Web.UI.WebControls.Panel`.
- Компонент `PlaceHolder`. Как и предыдущий компонент, он является контейнером для элементов Web-страницы. Однако он предназначен для размещения элементов, которые динамически создаются Web-приложением, т. е. на момент первоначальной загрузки и отображения Web-страницы эти элементы еще не существовали. Полное наименование его класса — `System.Web.UI.WebControls.PlaceHolder`.
- Компонент `Calendar`. Достаточно интересный компонент, который отображает календарную панель на Web-странице. Очень удобно использовать подобный функциональный элемент на сайтах с большим информационным наполнением, разбитым по датам. Полное наименование класса, реализующего этот компонент, — `System.Web.UI.WebControls.Calendar`.
- Компонент `AdRotator`. Позволяет отображать в заданной или случайной последовательности графические изображения. По сути дела, данный

элемент является заготовкой для создания баннерной системы. Полное наименование класса — `System.Web.UI.WebControls.AdRotator`.

- Компонент `Table`. Предназначен для создания таблиц. Чаще всего используют для отображения некоей информации, взятой из базы данных, в табличном виде. Полное наименование класса — `System.Web.UI.WebControls.Table`.
- Компонент `RequiredFieldValidator`. Данный компонент создает текстовое предупреждение, которое будет отображено, если пользователь при работе с Web-страницей не введет некоторые данные, являющиеся обязательными для продолжения работы. Полное наименование класса — `System.Web.UI.WebControls.RequiredFieldValidator`.
- Компонент `CompareValidator`. Применяется для проверки равенства или неравенства (соотношение больше-меньше) некоего значения, введенного пользователем. Полное наименование класса, реализующего этот компонент, — `System.Web.UI.WebControls.CompareValidator`.
- Компонент `RangeValidator`. Позволяет проверять, входит ли введенное пользователем значение, в некий заранее определенный интервал допустимых значений. Полное наименование класса для данного компонента — `System.Web.UI.WebControls.RangeValidator`.
- Компонент `RegularExpressionValidator`. Компонент позволяет проверять введенное пользователем значение на соответствие некоему шаблону, большой набор которых предоставлен разработчику в готовом виде, и, кроме того, естественно, разработчик может сам устанавливать свои собственные символьные шаблоны. Полное наименование соответствующего класса — `System.Web.UI.WebControls.RegularExpressionValidator`.
- Компонент `CustomValidator`. Этот компонент, как и несколько предыдущих, относится к контролирующим элементам. Но если в предыдущие компоненты уже встроены механизмы проверки соответствия введенного значения некоему условию, то данный элемент позволяет разработчику подключать свои собственные функции проверки достоверности введенных данных. Полное наименование класса — `System.Web.UI.WebControls.CustomValidator`.
- Компонент `ValidationSummary`. Предназначен для вывода отчета о несоответствиях данных, введенных пользователем. По сути, данный компонент является итоговым отчетом, который может быть связан с предыдущими компонентами проверки достоверности. Полное наименование класса — `System.Web.UI.WebControls.ValidationSummary`.
- Компонент `Xml`. Предназначен для отображения на Web-странице данных, полученных из XML-документов. Естественно, отображение происходит не напрямую, так как браузер не может знать, как именно следует отображать содержимое тех или иных XML-тегов. Поэтому с документом связывается стилевая таблица XSL, содержащая инструкции по отображению использованных XML-тегов. Полное наименование класса для данного компонента — `System.Web.UI.WebControls.Xml`.

- Компонент `CrystalReportViewer`. Предназначен для внедрения в разрабатываемые Web-страницы различных отчетов, построенных на технологии Crystal Report, которая входит в состав Visual Studio .NET. Полное наименование класса для данного компонента — `System.Web.UI.WebControls.CrystalReportViewer`.

На этом перечень компонентов, расположенных на вкладке Web Forms (Web-формы) заканчивается. На самом деле, этого краткого перечня явно недостаточно для того, чтобы работать с этими компонентами. Однако в следующих разделах мы увидим, как их можно использовать.

Объект *HttpResponse*

Встроенный объект `HttpResponse` выполняет пересылку информации браузеру удаленного пользователя. Разработчик также может обращаться к этому объекту посредством свойства `Response` объекта `Page`. Примеры использования этого встроенного объекта мы увидим в последующих разделах. В этом разделе мы рассмотрим основные свойства и методы данного объекта. Начнем со свойств.

- `Buffer`. Свойство логического типа, в котором указывается, будет ли буферизоваться информация, передаваемая удаленному пользователю, перед отправкой. Если разработчик использует значение `True`, то информация не будет передаваться в браузер после выполнения каждого метода `Write`, а станет накапливаться в буфере, а затем отправляться удаленному пользователю единым пакетом. Данное свойство унаследовано из предыдущих версий ASP.
- `BufferOutput`. Данное свойство является полным функциональным аналогом предыдущего свойства. Однако для приложений ASP.NET рекомендуется использовать именно его.
- `Cache`. В этом свойстве хранятся установки кэширования страницы, такие как, например, срок ее актуальности.
- `CacheControl`. Свойство устанавливает значение одноименного HTTP-заголовка. В качестве значения могут использоваться ключевые слова `Public` или `Private`.
- `Charset`. В данном свойстве указывается наименование кодировки символов, которая должна быть использована браузером для отображения посланной ему информации.
- `ContentEncoding`. В данном свойстве хранится значение одноименного HTTP-заголовка, который устанавливает тип кодирования пересылаемой информации.
- `ContentType`. Свойство предназначено для хранения MIME-типа передаваемой информации.

- `Cookies`. В свойстве хранится коллекция cookies, которая будет установлена на локальную систему пользователя. Более подробно о применении cookies будет рассказано несколько позже в соответствующем разделе главы.
- `Expires`. В данном свойстве указывается срок актуальности передаваемой Web-страницы, попросту, время ее хранения в кэше браузера удаленного пользователя. Значение данного свойства указывается в минутах. Свойство введено в ASP.NET в целях обеспечения совместимости с предыдущими версиями ASP.
- `ExpiresAbsolute`. Данное свойство, как и предыдущее, устанавливает срок хранения страницы в кэше браузера пользователя. Однако для этого свойства значение указывается в виде стандартной даты, после наступления которой страница будет считаться неактуальной.
- `Filter`. Свойство позволяет разработчику создавать некий фильтр (например, переводящий все символы в верхний регистр), через который будет пропускаться все содержимое отсылаемой Web-страницы перед тем как браузер удаленного пользователя получит эту страницу.
- `IsClientConnected`. Свойство логического типа, показывающее, не отключился ли еще удаленный пользователь от сервера.
- `Output`. Ключевое свойство объекта `HttpResponse`. В нем хранится передаваемая удаленному пользователю информация в виде текста.
- `OutputStream`. Данное свойство весьма похоже на предыдущее свойство `Output`, однако в нем информация, отсылаемая пользователю, хранится в двоичном виде, а не как текст.
- `Status`. Текст, указываемый разработчиком в этом свойстве, будет отображен в строке статуса браузера удаленного пользователя, когда тот получит переданную ему страницу.
- `StatusCode`. В данном свойстве хранится код статуса HTTP, передаваемый пользователю. В качестве значения используется целое число, как раз и являющееся кодом статуса передачи информации. По умолчанию используется значение равное 200, обозначающее успешную передачу данных.
- `StatusDescription`. Свойство похоже на предыдущее, но есть, конечно, и отличия. Так, значение имеет тип `String`, т. е. статус передачи информации записывается в виде строки. По умолчанию используется значение `OK`.

Конечно же, помимо свойств, данный встроенный объект обладает и методами. Рассмотрим и их.

- `AddHeader`. Метод позволяет пересылать заголовки протокола HTTP удаленному пользователю. В качестве параметров данного метода передаются

наименование заголовка и значение, приписываемое этому заголовку. Оба параметра передаются как тип `String`. На самом деле этот метод унаследован из предыдущих версий ASP и не рекомендован к употреблению в приложениях ASP.NET.

- `AppendHeader`. Метод полностью идентичен предыдущему. Рекомендован к применению в приложениях ASP.NET. Кстати, всегда следует помнить, что заголовки HTTP добавляются не в стандартный поток, а в самое начало передаваемого документа. В связи с этим лучше при передаче заголовков протокола HTTP использовать буферизацию информации.
- `AppendToLog`. Метод позволяет записывать информацию напрямую в log-файл сервера IIS. Естественно, в качестве параметра передается строка, которую следует записывать в log-файл. Подобный метод позволяет дополнительно документировать действия пользователей.
- `BinaryWrite`. Метод записывает данные в выходной поток, который переправляется удаленному пользователю. Данные передаются в двоичном виде, "как есть". В качестве параметра методу передается массив с элементами типа `Byte`.
- `Clear`. Метод полностью очищает буфер, в котором хранится информация, предназначенная для отправки удаленному пользователю.
- `ClearContent`. Метод полностью функционально идентичен только что рассмотренному методу `Clear`.
- `ClearHeaders`. Метод удаляет все заголовки протокола HTTP, которые находятся в буфере, подготавливаемому к передаче удаленному пользователю.
- `End`. Метод немедленно закрывает установленное соединение с удаленным пользователем. При этом, естественно, никакой информации пользователю не передается.
- `Flush`. Метод немедленно отправляет пользователю всю информацию, накопленную к моменту выполнения метода в буфере. Естественно, для этого свойство `BufferOutput` должно быть установлено в `True`, иначе произойдет ошибка, и будет сгенерировано исключение.
- `Redirect`. Метод перенаправляет пользователя к другому ресурсу. Проще говоря, если необходимо вывести отдельную Web-страницу, которая уже сформирована и ее URL известен, следует воспользоваться этим методом. В качестве параметра методу передается URL ресурса, который будет отправлен удаленному пользователю.
- `Write`. Один из наиболее часто используемых методов объекта `HttpResponse`. Записывает в выводимый поток текстовую информацию. В качестве параметра методу могут передаваться значения типа `Char`, `String` или массивы типа `Char`.

- `WriteFile`. Метод записывает в выводимый поток содержимое файла, имя которого передается методу в качестве параметра типа `String`. В качестве дополнительных параметров можно передать начальную и конечную позиции передаваемого блока информации.

На этом мы заканчиваем краткий обзор встроенного объекта `HttpResponse`, позволяющего передавать информацию удаленному пользователю, которая затем будет отображена его браузером.

Объект *HttpRequest*

Если объект `HttpResponse` позволяет разработчику с максимальным удобством отправлять информацию удаленному пользователю, не заботясь о различных мелочах физического уровня, а сосредотачиваясь именно на логике отсылаемого пакета, то объект `HttpRequest` помогает разработчику легко разгрести ту кучу разнородной информации, которая приходит от удаленного пользователя. В этот объект помещается информация, посылаемая браузером на сервер. В блоке принимаемой информации могут находиться данные, введенные пользователем в элементы управления форм, URL запрошенного ресурса, содержание cookies и многое другое. Объект `HttpRequest` поможет разработчику получить именно ту информацию, которая ему нужна для функционирования приложения.

Мы поступим так же, как и в предыдущем разделе — рассмотрим перечень свойств и методов объекта. Принципы и основные приемы работы с данным объектом все равно будут объяснены в иных разделах, посвященных конкретным задачам Web-приложений. Начнем со свойств.

- `AcceptTypes`. Свойство содержит в качестве значения массив строк типа `String`, в которых записываются типы MIME, которые поддерживаются браузером удаленного пользователя.
- `ApplicationPath`. Свойство содержит путь к виртуальному каталогу, в котором находится ASP.NET приложение, относительно корневого каталога WWW-сервера.
- `Browser`. Составное свойство, в котором указывается список параметров браузера удаленного пользователя. Значение данного свойства имеет тип `HttpBrowserCapabalities`. Соответственно, объект подобного типа имеет свои свойства, которые необходимо перечислить.
  - `ActiveXControls`. Свойство логического типа, указывающее, разрешено ли данному браузеру работать с элементами ActiveX.
  - `AOL`. Свойство логического типа, указывающее, использует ли удаленный пользователь специализированный браузер службы AOL.
  - `BackgroundSounds`. Свойство логического типа, указывающее, разрешено ли данному браузеру воспроизводить звуковые файлы, прикрепленные к Web-страницам.

- `Beta`. Свойство логического типа, сигнализирующее, что браузер удаленного пользователя является всего лишь бета-версией с возможно усеченной функциональностью.
- `Browser`. Значение данного свойства является строкой, в которой содержится условное наименование браузера. Это же наименование передается в виде содержимого заголовка HTTP с наименованием `User-Agent`.
- `CDF`. Свойство логического типа, указывающее, может ли браузер удаленного пользователя обрабатывать push-каналы доставки информации, созданные на основе формата CDF (Channel Definition Format).
- `Cookies`. Свойство логического типа, указывающее, разрешено ли браузеру сохранять cookies на машине удаленного клиента.
- `Crawler`. Свойство логического типа, указывающее, установлено ли в браузере пользователя средство поиска Web crawler.
- `Frames`. Свойство указывает, может ли браузер пользователя отображать фреймы. В настоящее время этот вопрос, очевидно, не имеет особого смысла.
- `JavaApplets`. Свойство логического типа, указывающее, может ли браузер удаленного пользователя корректно работать с апплетами Java.
- `JavaScript`. Логическое свойство, в котором указывается, умеет или нет браузер удаленного пользователя интерпретировать и выполнять Java-сценарии.
- `MajorVersion`. Свойство содержит основной номер версии браузера, т. е. число, стоящее до точки в полном номере версии. Значение данного свойства имеет, естественно, тип `Integer`.
- `MinorVersion`. Свойство указывает дополнительный номер версии, т. е. число, находящееся после первой точки в полном номере версии.
- `MSDomVersion`. В свойстве указывается номер версии объектной модели документа (Microsoft XML Document Object Model), которая поддерживается браузером пользователя.
- `Platform`. В данном свойстве содержится кодовое наименование операционной системы, которая установлена на машине удаленного пользователя. Естественно, значение данного свойства имеет тип `String`.
- `Tables`. Свойство указывает, может ли браузер пользователя отображать таблицы, включаемые в состав HTML-документов. Да, было когда-то такое время, когда браузеры не могли работать с таблицами.
- `Type`. В данном свойстве указываются кодовое наименование браузера и основной номер его версии. Значение свойства имеет тип `String`.

- `VBScript`. Логическое свойство, в котором указывается, умеет или нет браузер удаленного пользователя интерпретировать и выполнять сценарии, написанные на языке VBScript.
- `Version`. Свойство содержит в строковом виде полную версию применяемого пользователем браузера.
- `W3CDOMVersion`. В свойстве указывается номер версии объектной модели документа (W3C Document Object Model), которая разработана консорциумом WWW.
- `Win16`. Логическое свойство, в котором указывается, работает удаленный пользователь на шестнадцатиразрядной версии Windows или нет.
- `Win32`. Логическое свойство, в котором указывается, работает удаленный пользователь на тридцатидвухразрядной версии Windows или нет.

- `ClientCertificate`. Значение данного свойства имеет тип `HttpClientCertificate` и содержит информацию об установках безопасности клиента, если тот использует соединение по протоколу SSL 3.0.
- `ContentEncoding`. В этом свойстве указывается наименование кодировки символов, примененной браузером при отправке информации на сервер.
- `ContentLength`. В данном свойстве указывается размер блока информации, переданного на сервер. Размер рассчитывается в байтах. Естественно, значение свойства имеет тип `Integer`.
- `ContentType`. Свойство содержит наименование MIME-типа для принимаемой сервером информации.
- `Cookies`. Свойство содержит коллекцию cookies, которые передаются на сервер браузером пользователя.
- `FilePath`. В свойстве содержится путь к документу, который запросил пользователь. При этом не учитывается путь для перемещения внутри запрошенного документа, т. е. все закладки, если таковые были указаны в URL, этим свойством игнорируются. Путь указывается только к файлу.
- `Files`. Свойство содержит коллекцию файлов, переданных пользователем на сервер. Естественно, свойство имеет смысл обрабатывать только в том случае, если указан MIME-тип `multipart/form-data`.
- `Filter`. Данное свойство является функционально идентичным своему одноименному близнецу, который применяется к объекту `HttpResponse`. В этом свойстве указывается фильтр, применяемый к входящему потоку информации.
- `Form`. В свойстве содержится коллекция наименований органов управления формы, которая была использована посетителем сайта для ввода информации.
- `Headers`. В свойстве содержится коллекция заголовков протокола HTTP, переданных браузером удаленного пользователя на сервер.

- `HttpMethod`. Свойство указывает, какой именно тип передачи информации на сервер был использован браузером удаленного пользователя. В качестве значений применяются ключевые слова `Get`, `Post` и `Head`. Тип значения свойства, естественно, `String`.
- `InputStream`. В свойстве содержится входящий поток информации в "сыром" виде. А именно таким, каким его принял сервер.
- `IsAuthenticated`. Логическое свойство, указывающее, прошел ли удаленный пользователь аутентификацию или нет.
- `IsSecureConnection`. Логическое свойство, которое применяется для указания, что клиент пользуется защищенным протоколом соединения, таким, как SSL.
- `Params`. Свойство объединяет в себе коллекцию всех переменных из `QueryString`, `Form`, `ServerVariables` и `Cookies`.
- `Path`. В свойстве указывается путь в системе виртуальных каталогов к запрошенному удаленным пользователем ресурсу.
- `PathInfo`. Свойство содержит часть URL, запрошенного пользователем, располагающуюся после расширения файла. Другими словами, это добавочная информация, включенная в состав URL, помимо основного документа.
- `PhysicalApplicationPath`. В свойстве содержится физический (а не виртуальный) путь к каталогу, в котором находится и выполняется действующее приложение ASP.NET.
- `PhysicalPath`. Свойство содержит физический путь, соответствующий виртуальному расположению файла, запрошенного пользователем.
- `QueryString`. В свойстве находится коллекция с наименованиями всех переменных и параметров, переданных в строке запроса URL. Обычно в эту строку добавляются наименования органов ввода информации из форм и значений, введенных в них пользователем, если применен метод передачи `GET`.
- `RawUrl`. В свойстве находится URL, запрошенный пользователем, в сыром виде, не прошедшим процедуру лексического анализа для выделения из него составных частей. Если у разработчика есть такое желание, он может самостоятельно разбирать этот URL.
- `RequestType`. Свойство позволяет получать или устанавливать тип передачи информации от браузера удаленного пользователя на сервер. В качестве значений, естественно, могут быть использованы слова `GET` и `POST`.
- `ServerVariables`. Свойство позволяет приложению получить доступ к коллекции наименований стандартных свойств сервера и браузера.
- `TotalBytes`. В свойстве указывается размер пришедшего на сервер запроса от удаленного пользователя в байтах. Естественно, значение данного свойства имеет тип `Integer`.

- `Url`. В данном составном свойстве собрана самая различная информация об URL, запрошенном пользователем.
- `UrlReferrer`. В данном свойстве содержится информация об URL той страницы, с которой пользователь пришел на текущую.
- `UserAgent`. Свойство содержит неразобранную информацию о браузере, применяемом удаленным пользователем, в виде одной строки.
- `UserHostAddress`. В свойстве содержится IP-адрес удаленного пользователя, который послал данный запрос.
- `UserHostName`. В свойстве содержится доменное имя, приписанное удаленному пользователю.
- `UserLanguages`. Значение содержит отсортированный массив строк, в которых указываются языки, используемые удаленным пользователем, т. е. его лингвистические предпочтения.

На этом список свойств встроенного объекта `HttpRequest` заканчивается. Мы переходим к его уникальным методам.

- `BinaryRead`. Метод позволяет читать определенное количество байтов из входящего потока информации. В качестве параметра методу передается целочисленное значение, указывающее, сколько именно байтов следует прочитать. Метод возвращает массив типа `Byte`.
- `MapImageCoordinates`. Метод возвращает двумерный массив, содержащий координаты активных областей, присущих изображению-гиперссылке.
- `SaveAs`. Метод сохраняет входящий поток информации на диск в виде файла. В качестве параметров методу передаются строка с наименованием создаваемого файла и логическое значение, указывающее, следует ли сохранять помимо основного потока информации еще и заголовки протокола HTTP, пришедшие вместе с ним.

Список уникальных методов объекта `HttpRequest` исчерпан. Настало время перейти к рассмотрению других встроенных объектов.

Объект *HttpApplicationState*

Доступ к встроенному объекту `HttpApplicationState` обычно осуществляется при помощи его укороченного наименования — `Application`. В этом объекте хранятся данные, которые будут доступны всем пользователям, одновременно работающим с Web-приложением. Чаще всего подобный объект разработчики используют как коллекцию для хранения неких данных, которые должны быть доступны сразу нескольким Web-страницам. В одном из следующих разделов мы увидим, как можно использовать этот объект для хранения данных, введенных удаленным пользователем в элементы формы.

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.

Как уже было сказано, 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 ASP.NET нет нужды пользоваться данным свойством, достаточно обратиться к штатным механизмам присваивания значений. Свойство оставлено для обеспечения совместимости с предыдущими версиями ASP.
- `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 передаются 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.

Легко увидеть, что данный объект весьма прост в использовании. Примеры, находящиеся в последующих разделах главы, убедительно докажут это.

Объект *HttpServerUtility*

В коллекции встроенных объектов ASP.NET есть и еще один очень интересный объект, который позволяет приложению напрямую оперировать некоторыми установками самого WWW-сервера. Данный объект носит полное наименование `HttpServerUtility`, к нему можно обращаться и с использованием более короткого (фамильярного) имени — `Server`. Свойств у него немного, но методов вполне достаточно. В общем и целом, можно сказать, что функциональность данного встроенного объекта, как впрочем, и других объектов, должна удовлетворить самого придирчивого разработчика.

Начнем мы рассмотрение функциональности данного объекта с перечисления его свойств.

- `MachineName`. В свойстве содержится уникальное имя той машины, на которой и функционирует WWW-сервер IIS.
- `ScriptTimeout`. В свойстве содержится целочисленное значение, которое указывает величину тайм-аута, т. е. промежутка времени, в течение которого сервер будет ожидать отклика от пользователя, в секундах. Естественно, свойство позволяет как получать искомую величину, так и явно устанавливать ее.

На этом список свойств заканчивается, и мы переходим к рассмотрению методов объекта.

- `ClearError`. Метод удаляет ранее инициированные исключения, а именно сообщения об ошибках и нештатных ситуациях.
- `CreateObjectFromClsid`. Метод создает на сервере некий COM-объект, уникальный идентификатор класса которого (CLSID) передается методу в качестве параметра типа `String`.
- `Execute`. Метод сам посылает запрос серверу на загрузку в браузер какой-либо Web-страницы. В качестве параметра может быть передано либо

имя загружаемой страницы, либо непосредственно отсылаемый в браузер текст и наименование процедуры, пересылающей его.

- `GetLastError`. Метод возвращает значение типа `Exception`, в котором указывается код самой последней ошибки, возникшей при работе приложения.
- `HtmlDecode`. Метод позволяет декодировать информацию, записанную с применением символьных подстановок HTML. Как известно, многие символы, такие, например, как символы сравнения "больше" или "меньше", нельзя напрямую включать в содержимое HTML-страниц, так как они выполняют служебные функции. Поэтому для таких символов установлены специализированные текстовые подстановки. И искомый метод позволяет конвертировать текст с подобными подстановками в обычную текстовую строку с нормальными символами. В качестве параметра методу передается строка с символьными HTML-подстановками. Метод, в свою очередь, возвращает уже преобразованную строку чистого текста.
- `HtmlEncode`. Метод является обратным преобразованием по отношению к только что рассмотренному. Впрочем, преобразование не всегда будет симметричным. Некоторые символы, не входящие в стандартную кодировку, будут указаны при помощи их шестнадцатеричных кодов.
- `MapPath`. Метод получает в качестве параметра виртуальный путь к какому-либо документу, находящемуся на сервере. В качестве результата возвращается физический путь к данному файлу в стандартной файловой системе сервера.
- `Transfer`. Метод принудительно прекращает передачу пользователю запрошенной страницы и начинает передавать ему содержимое иной страницы, URL которой передан методу в качестве параметра типа `String`.
- `UrlDecode`. Метод принимает строку, подвергнутую URL-кодированию, и возвращает ее в виде чистого текста. Следует указать, что URL-кодирование более жесткая процедура, чем HTML-кодирование. Все пробелы заменяются знаком плюса. Все символы, не входящие в стандартный набор ASCII, или имеющие код ниже, чем тридцать три, должны быть представлены в виде их шестнадцатеричных кодов.
- `UrlEncode`. Метод принимает в качестве параметра обычную строку и подвергает ее процедуре URL-кодирования. Результат возвращается методом как тип `String`.

На этом мы заканчиваем рассмотрение функциональности встроенного объекта `HttpServerUtility`. Как обычно, примеры использования тех или иных его особенностей будут приведены в книге несколько позже.

Объект *HttpSessionState*

Все рассмотренные встроенные объекты, несомненно, очень полезны, но не являются такой уж редкостью в технологиях Web-программирования. Прямые аналоги им можно найти практически в каждой системе разработки Web-приложений (за исключением совсем уж древних и примитивных). А вот объект `Session`, если и не является нововведением, появившимся в ASP впервые, тем не менее, представляет одну из наиболее заманчивых его возможностей.

Объект `Session` (полное имя его звучит как `HttpSessionState`) позволяет разработчику отслеживать сеансы работы отдельных пользователей с сайтом, на котором функционирует приложение ASP.NET. Простой пример. Предположим, разработчик создал онлайновый магазин. Тогда ему необходимо будет отслеживать действия каждого пользователя от первого захода на основную страницу сайта, до завершения покупки. Необходимо отслеживать и сохранять состояние его корзины заказа, учитывать, что именно он ищет в базе данных и иным образом индивидуализировать работу сайта. Естественно, можно при помощи каких-либо технологий, связанных с базами данных, создать подобный механизм. Но ведь можно все сделать намного проще, если воспользоваться ASP.NET и его встроенным объектом `Session`. Этот объект сам будет хранить все данные о сеансе работы каждого пользователя. В нем так же, как и в объекте `Application`, можно хранить некие глобальные переменные, но теперь они будут доступны только тому потоку, который и начал сеанс работы. То есть, при работе с объектом `Session` можно не бояться, что удаленные пользователи будут затирать данные друг друга. Принципам работы с сеансами будет посвящен достаточно обширный раздел главы, а пока лишь кратко рассмотрим функциональность объекта в виде его свойств и методов. Начнем со свойств.

- `Count`. Свойство целочисленного типа, в котором указывается количество элементов основной коллекции объекта `Session`.
- `IsCookieless`. Свойство логического типа. Имеет значение `True` в том случае, если данный сеанс идентифицируется без помощи `cookie`.
- `IsNewSession`. Логическое свойство. Приобретает значение `True`, если сеанс, обслуживаемый данным объектом `Session`, был создан последним запросом пользователя. Другими словами, искомый экземпляр объекта `Session` является еще "свежим", он не принимал участия в обработке каких-либо других страниц, кроме текущей.
- `IsReadOnly`. Свойство логического типа. Приобретает значение `True`, если искомый экземпляр объекта `Session` не может редактироваться, предоставляет доступ только для чтения. По умолчанию используется значение `False`.

- Item. Свойство, позволяющее получить доступ к конкретному элементу коллекции элементов объекта Session. В качестве параметра передается или порядковый номер искомого элемента, или его уникальное наименование.
- Keys. Свойство является коллекцией наименований всех переменных основной коллекции объекта Session.
- LCID. В данном свойстве содержится уникальный числовой идентификатор данного экземпляра объекта Session. Свойство имеет тип Integer.
- Mode. В данном свойстве указывается применяемый тип сеанса. Значение свойства имеет тип SessionStateMode. Это перечислимый тип, который состоит из четырех ключевых слов.
  - InProc. Сеанс работает с применением стандартных средств ASP. Используется по умолчанию.
  - Off. Поддержка сеансов отключена.
  - SQLServer. Идентификация сеансов производится при помощи средств SQL-сервера.
  - StateServer. Идентификация сеансов производится при помощи штатных методов операционной системы сервера. Один из самых надежных режимов сохранения состояния сеанса, так как практически невозможно при сбое одного из процессов испортить данные остальных сеансов. Однако за подобную надежность приходится платить некоторым замедлением работы сеанса.
- SessionID. В свойстве содержится уникальный текстовый идентификатор данного экземпляра объекта Session.
- Timeout. Свойство позволяет задавать и получать время тайм-аута для данного сеанса. Свойство имеет целочисленный тип и время, устанавливаемое его значением, измеряется в минутах.

Теперь перейдем к методам объекта Session.

- Abandon. Метод немедленно уничтожает искомый экземпляр объекта Session и, соответственно, прекращает сопровождение пользователя.
- Add. Метод добавляет еще один элемент в коллекцию объекта Session. В качестве параметров методу передаются текстовое наименование вновь создаваемого элемента и его значение.
- Clear. Метод принудительно очищает всю коллекцию объекта Session, удаляя из нее все значения, присваиваемые элементам в процессе работы приложения.
- CopyTo. Метод копирует содержимое всех элементов коллекции объекта Session в одномерный массив. В качестве параметров методу передаются сам массив, в который будет происходить копирование, и целое число,

являющееся порядковым номером элемента массива, с которого и начнется копирование.

- `Remove`. Метод удаляет определенный элемент из коллекции объекта `Session`. В качестве параметра методу передается текстовое наименование удаляемого элемента.
- `RemoveAll`. Метод принудительно удаляет все элементы коллекции объекта `Session`.
- `RemoveAt`. Метод удаляет элементы коллекции объекта `Session`, начиная с некоего элемента и до конца. В качестве параметра методу передается порядковый номер элемента, с которого начнется удаление.

На этом мы заканчиваем обзор функциональности объекта `Session`. Но мы еще встретимся с ним в этой главе.

Формы

Формы являются, пожалуй, единственным средством получения информации от удаленного пользователя. Однако при разработке Web-приложений ASP.NET нет нужды всегда пользоваться стандартными средствами HTML. В конце концов, у нас есть компоненты Web Forms (Web-формы). Они-то нам и пригодятся. Схема работы достаточно проста.

Пользователь вносит данные в органы ввода информации, которые разработчик размещает на странице, и нажимает кнопку, отсылающую данные на сервер. На сервере их принимает программа, обрабатывает, и в результате обработки генерирует новую Web-страницу, которую и отсылает пользователю в качестве ответа на его действия. Схема, как видим, очень проста.

Однако встает следующий вопрос. А как разработчик будет создавать страницу с ответом? Первый пример приложения ASP.NET всего лишь показывает, как можно изменять внешний вид Web-страницы. Неужели потребуется на одной странице разместить два набора элементов оформления — один для формы, второй для ответа на действия пользователя, а затем по очереди делать их видимыми? Естественно, подобную схему работы нельзя назвать изящной.

На самом деле, все будет намного проще, если мы воспользуемся объектом ASP.NET с наименованием `HttpResponse`. Этот объект является встроенным, т. е. разработчику нет необходимости самостоятельно создавать экземпляр объекта для работы, он доступен постоянно. Данный объект предназначен для передачи информации с WWW-сервера в браузер удаленного пользователя по протоколу HTTP. Каждый раз, когда необходимо передать пользователю новую Web-страницу, разработчик может воспользоваться данным объектом, и с его помощью передать динамически сгенерированную страницу.

А как получать данные, введенные пользователем, от формы? Для этих целей существует встроенный объект `HttpRequest`, в котором находятся данные, получаемые сервером.

Попробуем создать проект с наименованием `form`. Он будет состоять из двух страниц. На первой мы разместим только стандартные органы управления HTML. Попробуем запросить у пользователя его имя и адрес электронной почты. После того, как удаленный пользователь введет запрошенные данные и нажмет не подтверждающую кнопку, эти данные будут отправлены на вторую страницу, которая будет заниматься обработкой данных, полученных из формы, размещенной на первой странице.

Так как первая страница не должна будет производить каких-либо действий, ее можно создать при помощи компонентов, размещенных на вкладке HTML. Нам потребуется два компонента `Label`, два однострочных поля текстового ввода `TextField` и одна кнопка `Submit Button`. HTML-код разработанной нами страницы в том виде, как он создан Visual Studio .NET, приведен в листинге 3.3. Было внесено единственное изменение. По умолчанию Visual Studio .NET считает, что любая создаваемая страница является исполняемой частью ASP-приложения, и поэтому в наш листинг была добавлена первая строка следующего вида:

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="WebForm1.aspx.vb" Inherits="form.WebForm1"%>
```

Она сигнализирует WWW-серверу о том, что данная Web-страница должна быть компилирована джиттером, как часть приложения. В нашей странице нет каких-либо органов управления ASP, поэтому ее совершенно необязательно пропускать через процедуру компиляции, тем более, что эта процедура занимает дополнительное время при отправке страницы удаленному пользователю. Следовательно, эту строку можно безболезненно удалить из HTML-кода нашей страницы. Получившийся код приведен в листинге 3.3.

Листинг 3.3

```
<!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"
        action="http://localhost/form/WebForm2.aspx">
   <INPUT style="Z-INDEX: 101; LEFT: 38px; POSITION: absolute; TOP: 54px"
          type="text" name="f1">
   <INPUT style="Z-INDEX: 102; LEFT: 45px; POSITION: absolute;
                 TOP: 183px" type="submit" value="Подтвердить">
   <INPUT style="Z-INDEX: 103; LEFT: 39px; POSITION: absolute;
                 TOP: 128px" type="text" name="f2">
   <DIV style="DISPLAY: inline; Z-INDEX: 104; LEFT: 38px; WIDTH: 320px;
        POSITION: absolute; TOP: 21px; HEIGHT: 19px"
        ms_positioning="FlowLayout">
    Введите Ваше имя
   </DIV>
   <DIV style="DISPLAY: inline; Z-INDEX: 105; LEFT: 43px; WIDTH: 336px;
        POSITION: absolute; TOP: 95px; HEIGHT: 19px"
        ms_positioning="FlowLayout">
    Введите Ваш адрес электронной почты
   </DIV>
  </form>
 </body>
</HTML>
```

Далее в книге мы будем приводить обычно два варианта HTML-кода для каждой Web-страницы: код, который формируется Visual Studio .NET, и тот, который будет получен браузером удаленного пользователя. Естественно, они будут отличаться, но в данном случае мы создали обычную Web-страницу, поэтому ее HTML-код на сервере и в браузере пользователя будут идентичны.

Необходимо, естественно, сделать несколько комментариев к приведенному коду. Основа взаимодействия формы с обрабатывающей Web-страницей кроется в теге `<form>`. В нашем случае он имеет следующий вид:

```
<form id="Form1" method="post"
 action="http://localhost/form/WebForm2.aspx">
```

Особое внимание следует обратить на атрибуты с наименованиями `method` и `action`. Атрибуту `method` приписано значение `post`. Этот атрибут указывает метод, при помощи которого будет передаваться информация обрабатывающему приложению. Всего может быть использовано два метода — `Post` и `Get`. Соответственно, в нашем случае мы выбрали первый вариант. Различия между ними будут отчетливо видны в нашем следующем примере, когда мы рассмотрим механизм передачи данных из формы с методом `Get`.

Атрибут `action` в качестве своего значения содержит URL того приложения, которое будет обрабатывать данные, введенные пользователем в органы управления формы. В нашем случае мы указываем на ASP-страницу с наименованием `WebForm2.aspx`. Именно она примет данные из формы, обработает их и сформирует новую Web-страницу, которая будет отправлена удаленному пользователю.

Для ввода данных удаленным пользователем мы разместили на Web-странице два однострочных поля ввода. В одно из них предполагается ввод имени пользователя, во второе — его адреса электронной почты. Первое поле ввода задано следующей конструкцией:

```
<INPUT style="Z-INDEX: 101; LEFT: 38px; POSITION: absolute; TOP: 54px"
       type="text" name="f1">
```

При помощи атрибута `type` мы задаем тип создаваемого органа ввода данных. Значение `text` указывает, что мы создаем однострочное поле ввода. Помимо этого необходимо обязательно указать наименование поля. Для данного поля мы устанавливаем имя `f1`, при помощи атрибута `name`. Для второго поля, куда пользователь будет вводить свой адрес электронной почты, было установлено наименование `f2`.

После того, как пользователь введет данные в предназначенные для этого поля, и нажмет на кнопку, отсылающую данные на сервер, управление переходит ко второй странице, которая носит наименование `WebForm2.aspx`. Займемся ее созданием.

На второй странице нам достаточно разместить всего два компонента `Label`, в которых мы отобразим полученные данные. Однако перед тем как их отобразить, данные необходимо получить. Для этого мы воспользуемся встроенным объектом `HttpRequest`. В его состав входит коллекция `Form`, которая содержит все данные, введенные пользователем. Для доступа к ним осуществляется обращение при помощи наименования органа управления или его порядкового номера. В листинге 3.4 приведен код класса, реализующего обрабатывающую Web-страницу.

Листинг 3.4

```
Public Class WebForm2
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents Label2 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
        Dim n, email As String
        n = Request.Form("f1")
        email = Request.Form("f2")
        Label1.Text = "Уважаемый " + n
        Label2.Text = "Мы обязательно отправим Вам письмо по Вашему
адресу — " + email
    End Sub


End Class
```

Как следует из листинга, получение и разбор данных будут производиться при загрузке страницы. Единственная функция, измененная разработчиком, является обработчиком события `Page_Load`. В этом обработчике мы объявляем две переменные типа `String`. Переменная `n` предназначена для хранения имени пользователя, а переменная `email`, естественно, будет хранить его адрес электронной почты. При этом мы используем обращение к элементам коллекции `Form` по их наименованию. Так, имя пользователя мы получаем следующим образом:

```
n = Request.Form("f1")
```

Как видим, все крайне просто. Соответственно, после того, как мы получаем введенные данные, мы соответствующим образом изменяем свойство `Text` элементов `Label`.

На рис. 3.9 показан первый этап работы пользователя с нашей формой, т. е. внешний вид Web-страницы WebForm1 в браузере Internet Explorer.

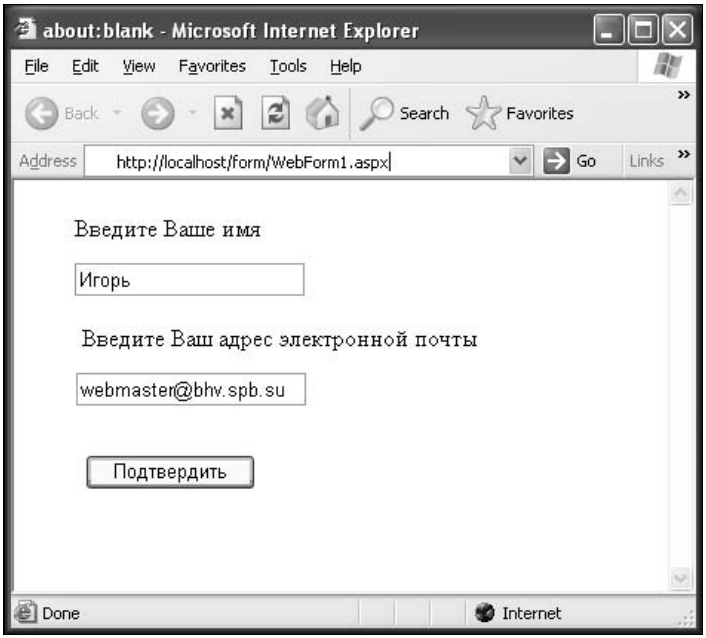


Рис. 3.9. Внешний вид Web-страницы, приведенной в листинге 3.3, при отображении в браузере Internet Explorer

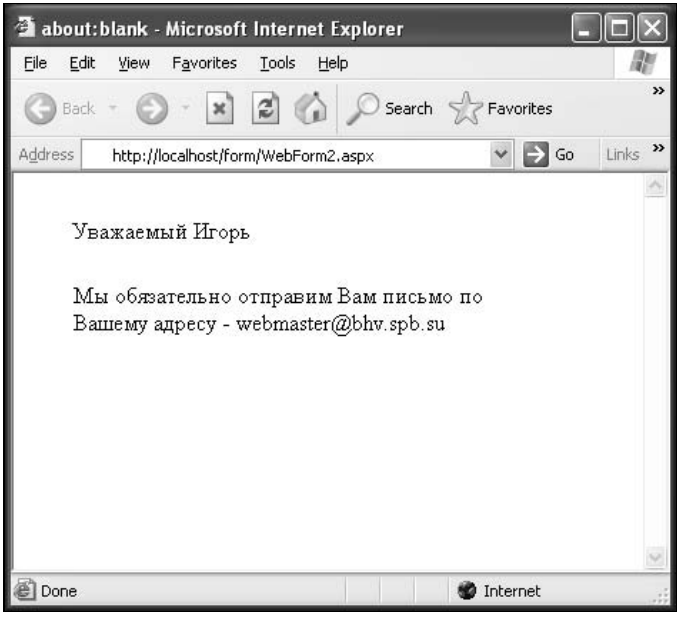


Рис. 3.10. Внешний вид Web-страницы, приведенной в листинге 3.4, при отображении в браузере Internet Explorer

После того, как пользователь введет свои данные и нажмет на кнопку, будет отображена следующая Web-страница. Ее внешний вид показан на рис. 3.10.

Код этой Web-страницы в браузере приведен в листинге 3.5.

Листинг 3.5

```
<!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="WebForm2.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTEwMTE1OTQwMDt0PDtsPGk8MT47PjtsPHQ8O2w8aTwxPjtpPDM+Oz47bDx0PH
A8cDxsPFRleHQ7PjtsPNCj0LLQsNC20LDQtdC80YvQuSDQmNCz0L7RgNGMOz4+Oz47Oz47dDx
wPHA8bDxUZXh0Oz47bDzQnNGLINC+0LHRj9C30LDRgtC10LvRjNC90L4g0L7RgtC/0YDQsNCy
0LjQvCDQktCw0Lwg0L/QuNGB0YzQvNC+INC/0L4g0JLQsNGI0LXQvNGDINCw0LTRgNC10YHRg
yAtIHdlYm1hc3RlckBiaHYuc3BiLnN1Oz4+Oz47Oz47Pj47Pj47Pg==" />


    
   <span id="Label1" style="Z-INDEX: 101; LEFT: 43px; POSITION: absolute;
             TOP: 30px">Уважаемый Игорь</span>
   <span id="Label2" style="Z-INDEX: 102; LEFT: 44px; POSITION: absolute;
             TOP: 75px">Мы обязательно отправим Вам письмо по Вашему
                        адресу —Webmaster@bhv.spb.su</span>
  </form>
 </body>
</HTML>
```

А теперь рассмотрим специфику работы с формой в том случае, если мы используем метод передачи данных `GET`. Изменений придется внести совсем немного. В HTML-коде первой страницы необходимо всего лишь изменить значение атрибута `method` тега `<form>`. Для данного атрибута требуется установить значение `get`. А вот обрабатывающую Web-страницу придется переработать несколько серьезнее. Ее код приведен в листинге 3.6.

Листинг 3.6

```
Public Class WebForm2
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents Label2 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
        Dim n, email As String
        n = Request.QueryString("f1")
        email = Request.QueryString("f2")
        Label1.Text = "Уважаемый " + n
        Label2.Text = "Мы обязательно отправим Вам письмо по Вашему
        адресу — " + email
    End Sub

End Class
```

Как видно, для получения данных, введенных пользователем, мы все так же используем встроенный объект `Request`. Но обращаемся уже не к коллекции `Form`, а к свойству `QueryString`. Это свойство также является коллекцией, и мы можем обращаться к ее элементам просто через указание их наименований. Мы не будем сейчас приводить иллюстраций, на которых будут

показаны эти страницы при отображении их браузером, так как они практически идентичны страницам из прошлого примера. Есть только одно отличие. URL обрабатывающей Web-страницы будет выглядеть следующим образом:

```
http://localhost/form/WebForm2.aspx?f1=%D0%98%D0%B3%D0%BE%D1%80%D1%8C&f2=
Webmaster@bhv.spb.su
```

Таким образом, все данные, введенные пользователем, были переданы через строку URL, а уже потом были извлечены из нее обрабатывающим приложением. При этом следует обратить внимание, что текст, записанный русскими символами, подвергся URL-кодированию, о котором мы уже говорили выше, а именно, все символы русского языка были переданы при помощи их шестнадцатеричных кодов.

Теперь встает вопрос, какой же тип передачи данных из формы предпочтительнее использовать — `Get` или `Post`? Здесь следует опираться как раз на специфику передачи данных. На строку URL изначально установлено ограничение по длине. Более того, вся эта строка видна пользователю вместе с именами полей, в том числе и скрытых, и с их значениями, что позволяет пользователю увидеть некоторую часть логики работы приложения и изменить вводимые значения и наименования полей ввода. Встает вопрос, предоставление подобной свободы пользователю — это хорошо? Даже для обычных приложений предоставление слишком большой свободы пользователю не приветствуется, а уж в Web-приложениях это совсем недопустимо, так как они в силу своей природы много больше уязвимы для вторжения. Поэтому всегда стоит использовать метод передачи данных `Post`, если нет каких-либо особых причин использовать именно `Get`. Кстати, я не могу себе представить хотя бы одной причины, которая жестко обязывала бы разработчика использовать именно метод `Get`.

Есть и еще один вариант работы с данными, введенными пользователями. Для этого мы можем вообще не использовать стандартные формы HTML, а воспользоваться компонентами Web Forms. Разработчику будет достаточно легко получить введенные пользователем данные, но встанет другой вопрос — как передать данные другой странице. Мы уже не помещаем их в выходной поток, который будет принимать и анализировать другая страница. Разработчик может воспользоваться неким аналогом глобальных переменных. Как мы уже знаем, встроенный объект `Application` доступен для всех страниц, входящих в состав одного приложения. В данном объекте приложение может создавать свои глобальные переменные. Этой возможностью мы и воспользуемся.

Для начала объявим новый проект ASP.NET с названием form2. Как и в первом примере с формой, мы создадим две страницы. На первой странице мы разместим компонент `TextBox`, реализующий текстовое поле, метку `Label`, в которой будет указан некий текст подсказки, и кнопку `Button`,

которая и запустит процесс обработки информации. Внешний вид этой страницы в среде разработке Visual Studio .NET показан на рис. 3.11.

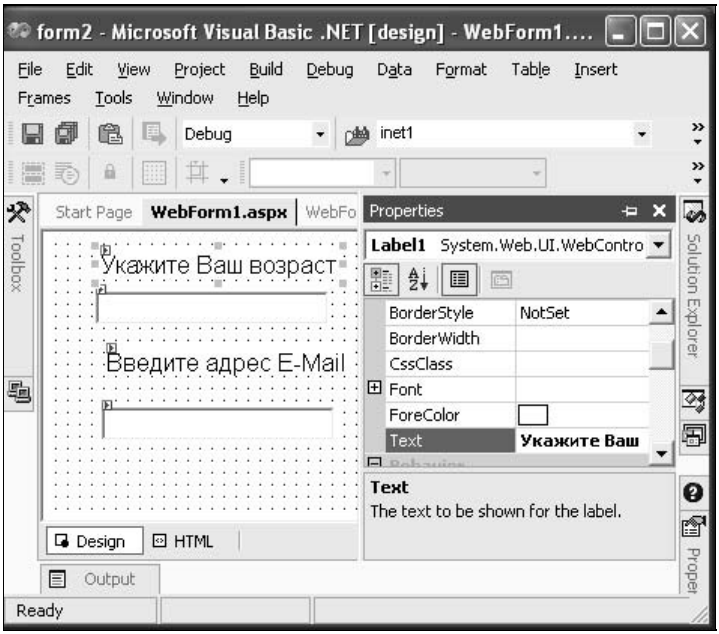


Рис. 3.11. Внешний вид страницы WebForm1 в среде разработки Visual Studio .NET

В нашем случае мы запрашиваем имя пользователя, сохраняем его в аналоге глобальной переменной, а затем вторая страница отображает введенное имя пользователя. Для этого на второй странице мы просто разместим один-единственный компонент `Label`, свойство `Text` которого мы будем задавать при загрузке страницы. Но этот механизм лучше уже рассмотреть при помощи листингов. Начнем мы с первой страницы, на которой размещены основные органы ввода информации. Внешний вид ее в среде разработки был продемонстрирован на рис. 3.11. А в листинге 3.7 приведен HTML-код данной страницы.

Листинг 3.7

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="form2.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:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 35px;
      POSITION: absolute; TOP: 39px" runat="server"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 40px;
      POSITION: absolute; TOP: 86px" runat="server"
      Text="Подтвердить"></asp:Button>
   <asp:Label id="Label1" style="Z-INDEX: 103; LEFT: 38px;
      POSITION: absolute; TOP: 13px" runat="server">
         Укажите Ваше имя</asp:Label>
  </form>
 </body>
</HTML>
```

Однако одного HTML-кода будет явно недостаточно для работы приложения. Необходимо запрограммировать обработчик нажатия пользователем на кнопку. В листинге 3.8 приведен полный код класса, соответствующего первой странице. Естественно, в нем приведен и обработчик нажатия на кнопку.

Листинг 3.8

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    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
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles Button1.Click
        Application.Clear()
        Application.Add("NAME", TextBox1.Text)
        Response.Redirect("WebForm2.aspx")
    End Sub
End Class
```

Нас будет больше всего интересовать код процедуры `Button1_Click`. В ней находятся всего три оператора. Первый оператор очищает ранее установленные глобальные переменные. Второй оператор является вызовом метода `Add` объекта `Application`. Мы просто создаем переменную с именем `NAME`, и приписываем ей значение, указанное пользователем в поле текстового ввода. Последний оператор заставляет браузер пользователя послать запрос серверу на отображение страницы `WebForm2.aspx`.

Теперь обратимся ко второй странице с наименованием `WebForm2.aspx`. На ней мы разместили всего лишь один текстовый объект `Label`. HTML-код этой страницы приведен в листинге 3.9.

Листинг 3.9

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm2.aspx.vb"
Inherits="form2.WebForm2"%>
<!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"3
        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: 42px;
      POSITION: absolute; TOP: 34px" runat="server">Label</asp:Label>
  </form>
 </body>
</HTML>
```

А код класса, соответствующего данной странице, показан в листинге 3.10.

Листинг 3.10

```
Public Class WebForm2
    Inherits System.Web.UI.Page
    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
```

```
        Label1.Text = "Здравствуйте, " + Application.Item(0)
    End Sub

End Class
```

Перед тем, как страница будет отображена в браузере, следует прочитать текст, записанный в элемент объекта `Application`, и с помощью полученного текста сформировать внешний вид текстовой метки, размещенной на странице. Для этого следует воспользоваться обработчиком события `Page_Load`, который автоматически генерируется средой разработки для каждой страницы, входящей в состав какого-либо приложения ASP.NET. Данный обработчик будет выполняться каждый раз при загрузке данной страницы в браузер удаленного пользователя. Поэтому мы и вставили в его тело оператор, который формирует свойство `Text` объекта `Label1` при помощи первого элемента объекта `Application`. Необходимо помнить, что нумерация элементов коллекций начинается с нуля, поэтому первый элемент коллекции будет иметь нулевой индекс.

На рис. 3.12 и 3.13 показана работа приложения в браузере Internet Explorer 6.

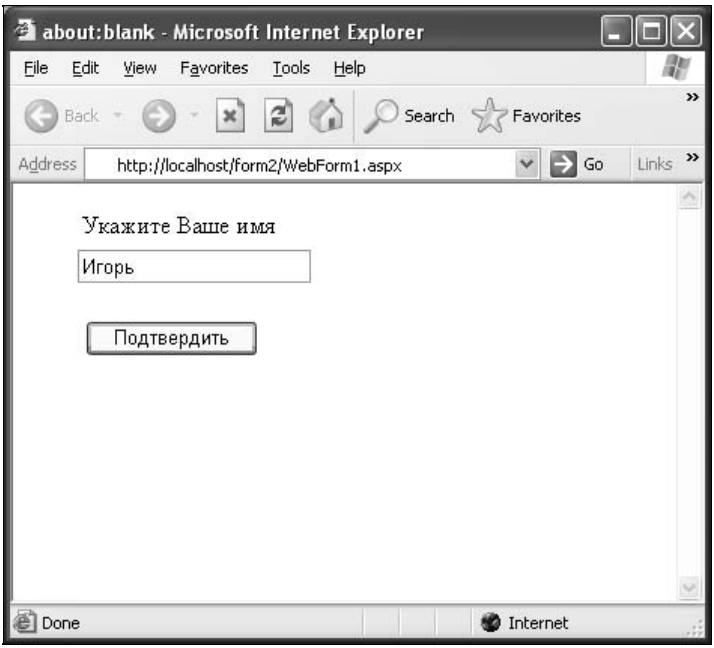


Рис. 3.12. Внешний вид страницы WebForm1 при отображении ее в браузере

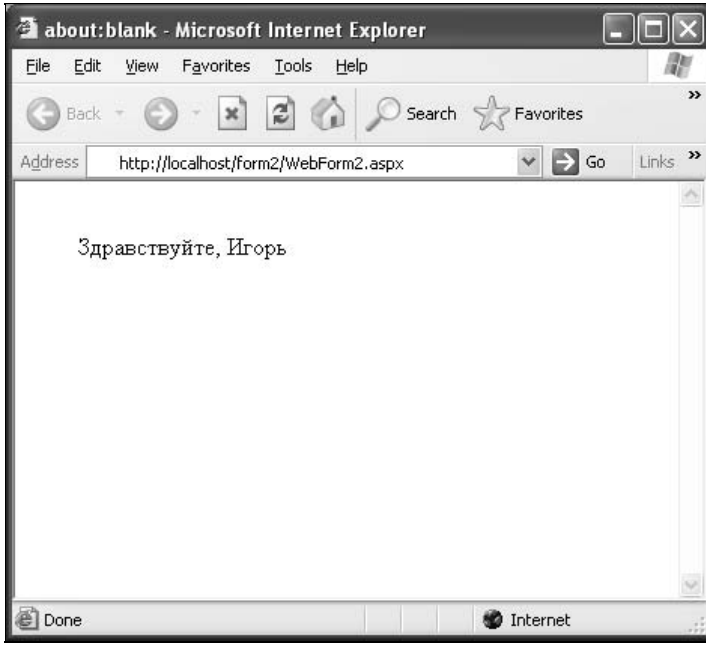


Рис. 3.13. Внешний вид страницы WebForm2, формирующейся в ответ на действия пользователя

Чем хорош подобный способ получения информации от пользователя? Разработчик получает возможность использовать всю мощь функциональных компонентов Web Forms, он не ограничен устаревшими элементами стандартных HTML-форм. Однако проблемы данного способа заключены не в приеме информации от пользователя, а в способе ее передачи на сервер. Использование глобальных переменных может быть хорошо в тех случаях, когда разработчик точно уверен, что в тот промежуток времени, когда один пользователь введет свои данные и передаст их на сервер, а тот возвратит ответ, никто другой не введет иные данные. Потому что переменная в нашем случае — одна и та же для всех пользователей, которые будут запрашивать наши страницы. Следовательно, вполне возможна ситуация, когда первый пользователь получил страницу с формой, начал ее заполнять, но пока он это делает, другой пользователь тоже получил форму и заполнил ее быстрее, чем первый. Тогда второй пользователь первым нажмет кнопку подтверждения, и именно его данные будут сохранены в глобальной переменной. Следовательно, когда первый пользователь отправит свои данные, ему в ответ будет послана информация, базирующаяся на данных второго пользователя. Избавиться от этой ситуации помогает принудительная очистка глобальной переменной перед отправкой данных из формы, как это сделано в нашем примере. Но это все равно не панацея, так как требуемое логикой приложения время жизни данных, хранимых в подобных глобальных пере-

менных, может быть достаточно большим, а большой поток посетителей будет достаточно часто обновлять данные глобальных переменных.

Еще одно решение этой проблемы может заключаться в блокировке объекта `Application`. В разделе, посвященном данному встроенному объекту, были описаны методы `Lock` и `Unlock`. Вызов метода `Lock` закрывает объект `Application` от изменений иными пользователями. После окончания работы с данным объектом, можно (и следует) открыть объект для всех остальных пользователей, при помощи метода `Unlock`. В том случае, если сервер, на котором базируется приложение ASP.NET, достаточно быстро работает, использование подобной методики работы может быть вполне оправданным. Но в этом случае, одна переменная может использоваться цепочкой только из двух страниц. Таким образом нельзя создавать последовательность из трех или более страниц, связанных одним комплектом глобальных переменных. Дело в том, что при такой схеме переменная будет жить не только во время ее обработки сервером, но и во время передачи страниц пользователю и, что самое главное, в то время, пока сервер может ожидать ответа пользователя. Это неприемлемо. Следует получить данные, заблокировать объект `Application`, записать данные в его основную коллекцию, передать их другой странице для обработки и/или записи в базу данных и разблокировать объект `Application`. Эта схема вполне работоспособна и позволяет обойтись без ощутимого замедления работы.

Теперь рассмотрим ситуацию, когда данные, вводимые пользователем, должны удовлетворять неким условиям. Например, приложение может требовать ввода телефона или адреса электронной почты. Естественно, необходимо проконтролировать, какие именно данные вводит пользователь. Возраст должен быть целым положительным числом в четко ограниченном диапазоне, например, от десяти до девяноста. В адресе электронной почты должен присутствовать знак "@" и, как минимум, одна точка в доменном имени. Почтовый индекс России должен состоять из шести цифр. И так далее.

Естественно, ошибки, обусловленные как невнимательностью пользователя, так и преднамеренным вводом неверных данных, просто неизбежны. И, уж поверьте моему опыту, их будет немало. Следовательно, необходимо проверять вводимые пользователем данные хотя бы на предмет соответствия некоему формату. Как это сделать?

Функционально все способы можно разделить на два типа — выполняемые на сервере и выполняемые на стороне пользователя. Проверка на стороне сервера может выполняться самим приложением. По результатам проверки эти данные либо передаются для записи и обработки, либо формируется страница с сообщением об ошибке, которая отправляется пользователю. Этот вариант удобен для разработки, так как программист будет пользоваться тем средством разработки и тем языком, который наиболее привычен для

него. Однако подобная схема имеет очень серьезный недостаток — слишком большое время реагирования на ошибку, допущенную пользователем. А именно, данные сначала уйдут по линиям связи (как всегда, слишком медленным, это, похоже, неотъемлемое свойство всех линий связи), затем будут обрабатываться сервером, а потом снова передаваться по искомым линиям связи обратно пользователю. Другими словами, узким местом будут именно линии передачи информации, которые в данной схеме используются два раза. Неразумно.

Однозначное предпочтение разработчик должен отдавать механизмам проверки, выполняющимся на стороне пользователя. В этом случае ему необходимо воспользоваться соответствующим языком сценариев. Несмотря на то, что ASP-разработчик пользуется языком из семейства Visual Basic, для создания сценариев, функционирующих на стороне пользователя, настоятельно рекомендуется использовать JavaScript. Причины этого мы уже обсуждали.

Visual Studio .NET конечно же позволяет интегрировать в разрабатываемые Web-страницы блоки кода, исполняемого на стороне клиента. Но писать программы проверки значений на языках сценариев, особенно для проверки текстовых шаблонов, является одним из самых нудных занятий в программировании, по моему мнению. И вот здесь нас ожидает один очень приятный сюрприз. Visual Studio .NET предлагает разработчику несколько механизмов проверки достоверности, которые сами автоматически осуществляют контроль введенных пользователем данных. Причем, что самое важное, они делают это без отправки данных на сервер. Вся проверка происходит на стороне пользователя. Рассмотрим тривиальный пример.

Создадим форму для получения возраста пользователя. Ограничим вводимое значение интервалом от десяти до девяноста. В том случае, если пользователь введет неверные данные, необходимо будет его предупредить об ошибке. Нам потребуется практически такая же форма, как и в предыдущем примере с использованием объекта `Application`. Естественно, придется несколько изменить текст подсказки, и добавить компонент с наименованием `RangeValidator`. Основное программирование будет производиться простым заданием свойств данного компонента.

В свойстве `ErrorMessage` следует установить текст предупреждения, которое будет отображаться на странице в случае внесения неверных данных пользователем. Также следует обратить особое внимание на группу свойств `Behavior`. Прежде всего, надо установить какое именно устройство ввода данных будет обрабатываться данным компонентом проверки соответствия шаблону. При разработке действует основное правило — один компонент проверки обрабатывает одно устройство. Разработчик может, конечно, оставить какое-либо устройство ввода данных не подстрахованным компонентом проверки, если данная информация не является критичной, или присоединить к нему два различных компонента проверки соответствия шаблону, но заставить один компонент контролировать несколько органов ввода не получится.

Итак, в свойстве `ControlToValidate` следует установить имя того устройства ввода данных, которое будет контролироваться компонентом проверки соответствия шаблону. Затем в свойстве `Type` надо указать, какой тип данных должен быть введен в контролируемое поле. Компонент проверки позволяет обрабатывать типы `String`, `Integer`, `Double`, `Date` и `Currency`, т. е. практически весь спектр типов данных. Также следует свойству `EnableClientScript` придать значение `True`, чтобы Visual Studio .NET мог сгенерировать код проверки данных на выбранном языке сценариев.

Остается лишь указать минимальное и максимальное возможные значения. Они устанавливаются при помощи свойств `MinimumValue` и `MaximumValue`, соответственно. HTML-код получившейся Web-страницы приведен в листинге 3.11.

Листинг 3.11

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="WebForm1.aspx.vb" Inherits="form2.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:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 35px;
      POSITION: absolute; TOP: 39px" runat="server"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 40px;
      POSITION: absolute; TOP: 86px" runat="server"
      Text="Подтвердить"></asp:Button>
   <asp:Label id="Label1" style="Z-INDEX: 103; LEFT: 38px;
      POSITION: absolute; TOP: 13px" runat="server">
         Укажите Ваш возраст</asp:Label>
   <asp:RangeValidator id="RangeValidator1" style="Z-INDEX: 104;
      LEFT: 229px; POSITION: absolute; TOP: 26px" runat="server"
      ErrorMessage="Возраст должен находиться в промежутке от 10 до 90"
      ControlToValidate="TextBox1" MaximumValue="90" MinimumValue="10"
      Type="Integer"></asp:RangeValidator>
```

```
  </form>
 </body>
</HTML>
```

Код класса, соответствующего разработанной Web-странице, приведен в листинге 3.12. Следует обратить внимание, что обработчик нажатия кнопки несколько отличается от обработчика из предыдущего примера. Теперь добавлен вызов метода `lock` для объекта `Application`, который блокирует его. Разблокировка объекта должна быть произведена при загрузке второй страницы, которая ответит пользователю, что его данные успешно приняты и обработаны.

Листинг 3.12

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents RangeValidator1 As
    System.Web.UI.WebControls.RangeValidator
    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
        Application.Clear()
        Application.Lock()
        Application.Add("AGE", TextBox1.Text)
        Response.Redirect("WebForm2.aspx")
    End Sub
End Class
```

Однако во всех этих листингах не показано, как именно будет контролироваться ввод пользователем данных. Действительно, не следует забывать, что тот HTML-код, который разработчик видит при создании страницы, содержит специфические теги ASP. Поэтому, когда пользователь запросит данную страницу, ее сначала пропустит через себя джиттер, создав приложение, а затем с правильным HTML-кодом отправит удаленному пользователю. На рис. 3.14 показан внешний вид созданной нами HTML-страницы при отображении браузером Internet Explorer в тот момент, когда пользователь ввел неверные данные.

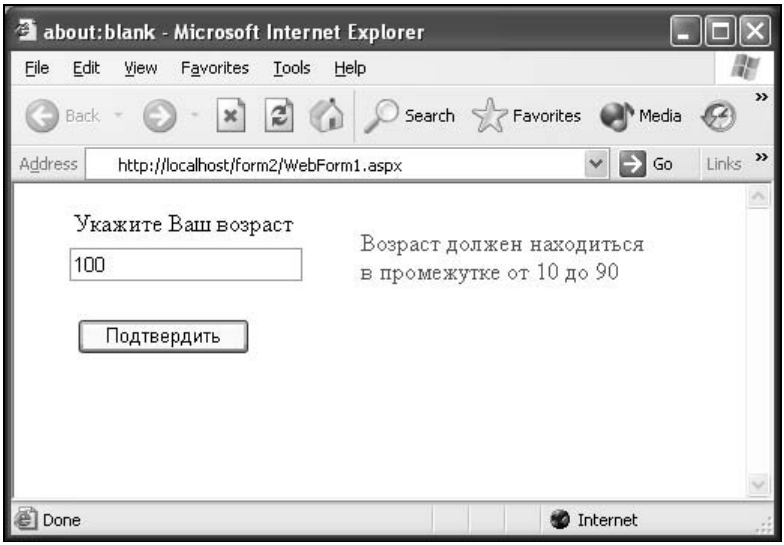


Рис. 3.14. Внешний вид страницы с компонентом системы проверки при попытке отправить неверные данные

А вот код, который передан браузеру удаленного пользователя, выглядит иначе, чем HTML-код у разработчика. Он приведен в листинге 3.13.

Листинг 3.13

```
<!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="dDwxMTkzNzk1ODM7Oz4=" />


<script language="javascript" src="/aspnet_client/system_Web/1_0_2914_16/
WebUIValidation.js"></script>


   <input name="TextBox1" type="text" id="TextBox1" style="Z-INDEX: 101;
LEFT: 35px; POSITION: absolute; TOP: 39px" />
   <input type="submit" name="Button1" value="Подтвердить" onclick="if
(typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); "
language="javascript" id="Button1" style="Z-INDEX: 102; LEFT: 40px;
POSITION: absolute; TOP: 86px" />
   <span id="Label1" style="Z-INDEX: 103; LEFT: 38px; POSITION: absolute;
TOP: 13px">Укажите Ваш возраст</span>
   <span id="RangeValidator1" controltovalidate="TextBox1"
errormessage="Возраст должен находиться в промежутке от 10 до 90"
type="Integer" evaluationfunction="RangeValidatorEvaluateIsValid"
maximumvalue="90" minimumvalue="10" style="color:Red;Z-INDEX:104;
LEFT:229px;POSITION:absolute;TOP:26px;visibility:hidden;">Возраст должен
находиться в промежутке от 10 до 90</span>


<script language="javascript">
<!--
 var Page_Validators =  new Array(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>
```

Как видно, сервер добавил достаточно объемный код JavaScript, который позволяет не только анализировать введенное пользователем значение, но и различные ошибки, которые могут быть вызваны настройками браузера пользователя. То есть, в код также встраивается система самодиагностики.

Для ASP-разработчика подобные компоненты системы проверки являются одним из самых полезных при разработке форм. Естественно, ему предоставлено несколько вариантов подобных компонентов. Мы рассмотрим механизм их применения, несколько расширив предыдущий пример.

Добавим еще одно поле для ввода адреса электронной почты. С ним мы свяжем два компонента системы проверки. Адрес электронной почты будет являться обязательной для ввода информацией. Соответственно, мы свяжем с ним компонент `RequiredFieldValidator`. В нем следует установить значение всего двух уже знакомых нам свойств — `ErrorMessage` и `ControlToValidate`. Этого будет достаточно для работы данного компонента.

Именно собственный адрес электронной почты является одним из самых "трудных" вопросов к пользователю. Слишком уж часто пользователи допускают ошибки в его написании. Поэтому стоит проконтролировать правильность его введения. Обычно достаточно проверить, есть ли в нем символ @ и хотя бы одна точка. Но и эти ограничения не могут гарантировать точного соответствия введенных данных формальным правилам написания адресов e-mail. Поэтому разработчику был предоставлен компонент `RegularExpressionValidator`. Он позволяет проверить соответствие введенных пользователем данных некоему шаблону. Достаточно большой список заранее подготовленных шаблонов предоставляется разработчику, когда тот пытается установить свойство `ValidationExpression`. В списке готовых шаблонов есть и так необходимый нам шаблон адреса e-mail под названием Internet E-mail Address. Именно им мы и воспользуемся.

Компоненты системы проверки допустимости не только блокируют отправку неверных данных на сервер при нажатии подтверждающей кнопки, они отображают предупреждения сразу после того, как контролируемый орган ввода теряет фокус. Если форма достаточно велика, то может сработать несколько систем проверки допустимости. Соответственно, пользователю лучше показать сразу все извещения. Для этого есть компонент `ValidationSummary`, который автоматически собирает вместе все извещения об ошибках и отображает их в виде списка или отдельных абзацев. Достаточно просто разместить этот компонент на странице, и он автоматически подключится ко всем остальным системам проверки допустимости.

HTML-код получившейся страницы до обработки ее сервером приведен в листинге 3.14.

Листинг 3.14

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind= "WebForm1.aspx.vb" Inherits="form2.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:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 35px; POSITION:
absolute; TOP: 39px" runat="server"></asp:TextBox>

   <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 37px; POSITION:
absolute; TOP: 284px" runat="server" Text="Подтвердить"></asp:Button>

   <asp:Label id="Label1" style="Z-INDEX: 103; LEFT: 38px; POSITION:
absolute; TOP: 13px" runat="server">Укажите Ваш возраст</asp:Label>

   <asp:RangeValidator id="RangeValidator1" style="Z-INDEX: 104; LEFT:
229px; POSITION: absolute; TOP: 26px" runat="server" ErrorMessage=
"Возраст должен находиться в промежутке от 10 до 90" ControlToValidate=
"TextBox1" MaximumValue="90" MinimumValue="10" Type="Integer">
</asp:RangeValidator>

   <asp:TextBox id="TextBox2" style="Z-INDEX: 105; LEFT: 39px; POSITION:
absolute; TOP: 116px" runat="server"></asp:TextBox>

   <asp:RequiredFieldValidator id="RequiredFieldValidator1" style="Z-INDEX:
106; LEFT: 223px; POSITION: absolute; TOP: 114px" runat="server"
ControlToValidate="TextBox2" ErrorMessage="Обязательно укажите Ваш
E-Mail"></asp:RequiredFieldValidator>

   <asp:RegularExpressionValidator id="RegularExpressionValidator1"
style="Z-INDEX: 107; LEFT: 227px; POSITION: absolute; TOP: 139px"
runat="server" ControlToValidate="TextBox2" ErrorMessage="Введен неверный
адрес E-Mail" ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\
.\w+([-.]\w+)*"> </asp:RegularExpressionValidator>

   <asp:ValidationSummary id="ValidationSummary1" style="Z-INDEX: 108;
LEFT: 41px; POSITION: absolute; TOP: 212px" runat="server" Width="308px"
Height="29px"></asp:ValidationSummary>

   <asp:Label id="Label2" style="Z-INDEX: 109; LEFT: 42px; POSITION:
absolute; TOP: 78px" runat="server">Введите адрес E-Mail</asp:Label>

  </form>

 </body>

</HTML>
```

Сам код класса здесь не имеет смысла приводить, так как он не изменился кардинальным образом по сравнению с предыдущим примером. Достаточно лишь включить оператор добавления новой глобальной переменной. А вот внешний вид Web-страницы после того, как пользователь ввел неверные данные и, невзирая на показанные предупреждения, которые, напомню, отображаются сразу после того, как орган ввода данных теряет фокус, попробовал отправить на сервер, показан на рис. 3.15.

Естественно, было бы очень интересно увидеть, каким именно HTML-кодом достигается подобная функциональность. Код откомпилированной страницы приведен в листинге 3.15.

Листинг 3.15

```
<!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="dDwtOTEwMjY3Nzg0Ozs+" />
<script language="javascript" src="/aspnet_client/system_Web/1_0_2914_16/
WebUIValidation.js"></script>
   <input name="TextBox1" type="text" id="TextBox1" style="Z-INDEX: 101;
LEFT: 35px; POSITION: absolute; TOP: 39px" />
   <input type="submit" name="Button1" value="Подтвердить" onclick=
"if (typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); "
language="javascript" id="Button1" style="Z-INDEX: 102; LEFT: 37px;
POSITION: absolute; TOP: 284px" />
   <span id="Label1" style="Z-INDEX: 103; LEFT: 38px; POSITION: absolute;
TOP: 13px">Укажите Ваш возраст</span>
   <span id="RangeValidator1" controltovalidate="TextBox1" errormessage=
"Возраст должен находиться в промежутке от 10 до 90" type="Integer"
evaluationfunction="RangeValidatorEvaluateIsValid" maximumvalue="90"
minimumvalue="10" style="color:Red;Z-INDEX:104; LEFT:229px;POSITION:
absolute;TOP:26px;visibility:hidden;">Возраст должен находиться в
промежутке от 10 до 90</span>
   <input name="TextBox2" type="text" id="TextBox2" style="Z-INDEX: 105;
LEFT: 39px; POSITION: absolute; TOP: 116px" />
   <span id="RequiredFieldValidator1" controltovalidate="TextBox2"
errormessage="Обязательно укажите Ваш E-Mail" evaluationfunction=
"RequiredFieldValidatorEvaluateIsValid" initialvalue="" style=
"color:Red;Z-INDEX:106;
LEFT:223px;POSITION:absolute;TOP:114px;visibility:hidden;">Обязательно
укажите Ваш E-Mail</span>
   <span id="RegularExpressionValidator1" controltovalidate="TextBox2"
errormessage="Введен неверный адрес E-Mail" evaluationfunction=
"RegularExpressionValidatorEvaluateIsValid" validationexpression=
"\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" style="color:Red;
Z-INDEX:107; LEFT:227px;POSITION:absolute;TOP:139px;visibility:hidden;">
Введен неверный адрес E-Mail</span>
   <div id="ValidationSummary1" style="color:Red;height:29px;width:
308px;Z-INDEX:108;LEFT:41px;POSITION:absolute;TOP:212px;display:none;">
</div>
```

```
   <span id="Label2" style="Z-INDEX: 109; LEFT: 42px; POSITION: absolute;
TOP: 78px">Введите адрес E-Mail</span>
<script language="javascript">
<!--
 var Page_ValidationSummaries =
new Array(document.all["ValidationSummary1"]);
 var Page_Validators =  new Array(document.all["RangeValidator1"],
document.all["RequiredFieldValidator1"],
document.all["RegularExpressionValidator1"]);
  // -->
</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>
```

Действительно, следует признать, что компоненты системы проверки допустимости могут достаточно серьезно облегчить жизнь разработчику, избавив его от утомительного написания вручную различных проверок допустимости введенных данных и освободив, подобным образом, время для разработки действительно серьезных модулей.

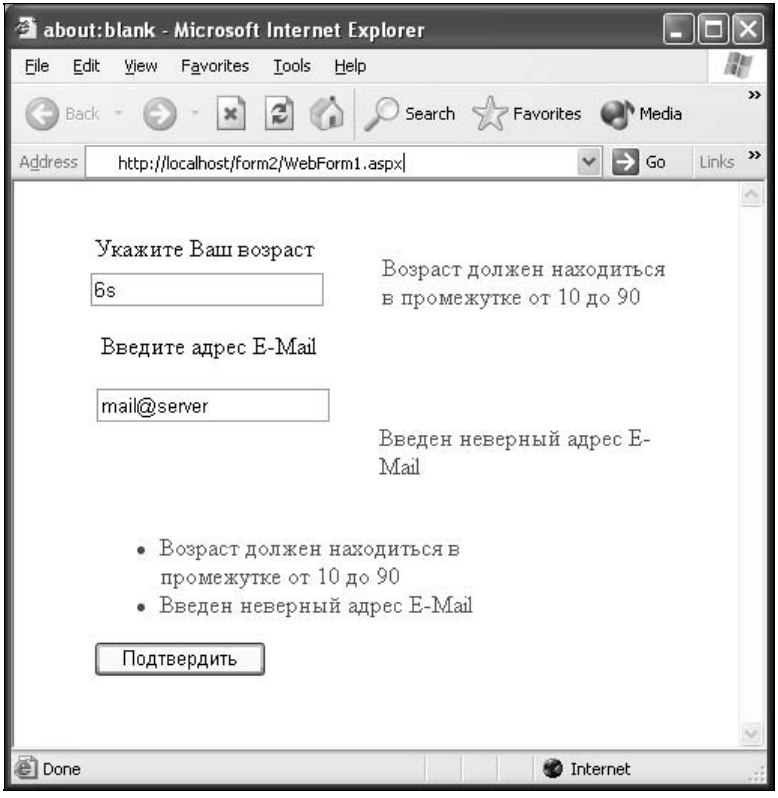


Рис. 3.15. Web-страница с предупреждениями об ошибках во введенных данных

Работа с графикой

Как мы уже знаем, Visual Studio.NET предоставляет разработчику компоненты, предназначенные для работы с графикой. В этом разделе мы рассмотрим основные приемы их использования в приложениях ASP.NET.

Начнем мы с самого простого объекта `Image`. На первый взгляд, он не слишком отличается от своего аналога с вкладки **HTML**, однако это не совсем так. Он является полноценным компонентом Web Forms, поэтому мы можем динамически изменять его свойства.

Попробуем создать простейшую страницу, иллюстрирующую принципы работы с этим компонентом. Предположим, у нас есть небольшая база данных о книгах. Мы хотим дать пользователю возможность выбирать их наименование и просматривать внешний вид обложки соответствующей книги.

Поступим мы максимально просто. Создадим новый объект с наименованием `Image`, и на страницу положим компонент `ListBox`, в котором мы раз-

местим наименования книг, а справа от него расположим объект `Image`, который будет служить для отображения их обложек. Изначально при загрузке страницы не будет выбрана ни одна книга, поэтому свойство `Visible` для компонента `Image` следует установить в `False`.

Теперь перейдем к установке содержимого объекта `ListBox`. Для этого надо в окне **Properties** (Свойства) активировать редактор свойства `Items`. Это диалоговое окно с наименованием **ListItem Collection Editor** (Редактор коллекции ListItem), внешний вид которого показан на рис. 3.16.

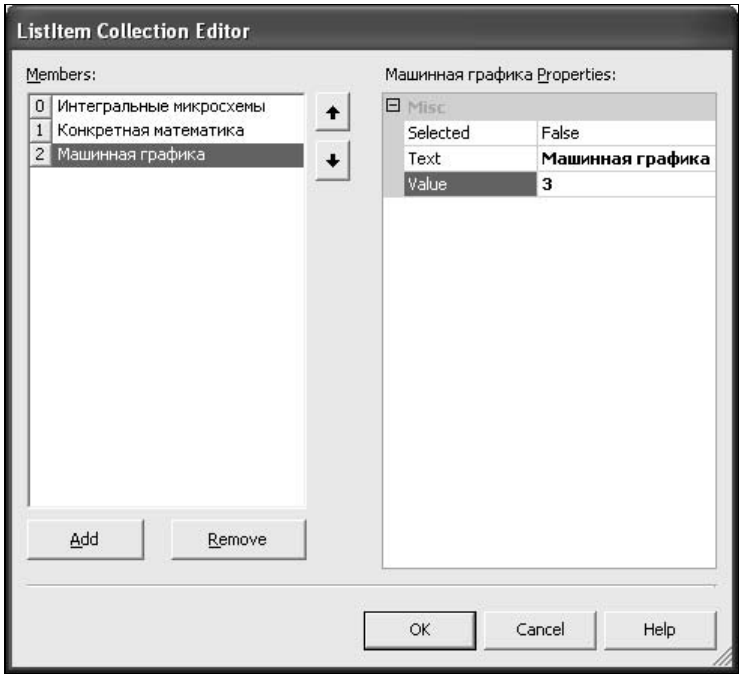


Рис. 3.16. Диалоговое окно **ListItem Collection Editor**

Нажатие на кнопку **Add** (Добавить) добавляет элемент в список. При этом в правой части диалогового окна появляется список свойств для добавляемого элемента. Так как изначально ни один элемент списка не будет находиться в выбранном состоянии, для всех элементов свойство `Selected` следует установить в `False`. В свойстве `Text` следует установить наименования книг, а именно те строки, которые будут отображаться в списке. Свойство `Value` будет автоматически принимать значение свойства `Text`, но разработчик, естественно, может сам изменить значение данного свойства. Именно оно будет передано на сервер для обработки. Впрочем, этот способ работы с органом ввода информации унаследован от обычных HTML-форм, и в некоторых случаях может быть успешно проигнорирован разработчиком.

После того, как список книг будет полностью создан, следует обратить внимание на свойство `AutoPostBack` для компонента `ListBox`, и установить его в `True`, чтобы выбранное пользователем значение автоматически отсылалось серверу, когда пользователь будет переходить от одного элемента списка к другому.

Теперь мы полностью подготовили дизайн своей страницы. Осталось подготовить ее содержимое и запрограммировать логику ее поведения. Для этого скопируем в соответствующий виртуальный каталог сервера все необходимые графические файлы с обложками книг. В моем случае я использовал три книги, и файлы получили имена 1.gif, 2.gif и 3.gif. Да, я знаю — это очень оригинально.

Теперь перейдем к программированию логики поведения Web-страницы. Воспользуемся стандартным обработчиком изменения состояния списка. Напишем функцию, перехватывающую событие `SelectedIndexChanged`. Двойной щелчок по нему на странице в режиме разработки переведет нас в режим написания кода функции-обработчика. Для анализа выбранного пользователем значения, воспользуемся свойством `SelectedIndex`, в котором указывается порядковый номер элемента списка, выбранного пользователем. Следует учитывать, что нумерация ведется с нуля. Разработчику останется проанализировать значение этого свойства, и в соответствии с ним установить свойство `ImageUrl` для компонента `Image`. После того, как все необходимые установки будут выполнены, следует сделать видимым этот рисунок. Для этого значению `Visible` присвоим - `True`. Полный вариант кода класса, реализующего данную страницу, можно увидеть в листинге 3.16.

Листинг 3.16

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Image1 As System.Web.UI.WebControls.Image
        Protected WithEvents ListBox1 As System.Web.UI.WebControls.ListBox
    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 ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
        If ListBox1.SelectedIndex = 0 Then
            Image1.ImageUrl = "1.gif"
        End If
        If ListBox1.SelectedIndex = 1 Then
            Image1.ImageUrl = "2.gif"
        End If
        If ListBox1.SelectedIndex = 2 Then
            Image1.ImageUrl = "3.gif"
        End If
        Image1.Visible = True
    End Sub
End Class
```

Конечно, сам код обработки можно было написать несколько изящнее, но в данном случае я сделал акцент на простоте и понятности кода. Внешний вид разработанной страницы при отображении в браузере показан на рис. 3.17.

HTML-код отображенной страницы приведен в листинге 3.17.

Листинг 3.17

```
<!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" con-
tent="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTM5OTQ2MTM2O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+O2k8Mz47PjtsPHQ8cD
xwPGw8SW1hZ2VVcmw7VmlzaWJsZTs+O2w8Mi5naWY7bzx0Pjs+Pjs+Ozs+O3Q8dDw7O2w8aTw
xPjs+Pjs7Pjs+Pjs+" />


   <img id="Image1" src="/image/2.gif" border="0"
style="height:130px;width:138px;Z-INDEX: 101; LEFT: 348px; POSITION:
absolute; TOP: 55px" />
   <select name="ListBox1" id="ListBox1" size="3"
onchange="__doPostBack('ListBox1','')" language="javascript"
style="height:139px;width:196px;Z-INDEX: 103; LEFT: 44px; POSITION:
absolute; TOP: 57px">
 <option value="1">Интегральные микросхемы</option>
 <option selected="selected" value="2">Конкретная математика</option>
 <option value="3">Машинная графика</option>


</select>
   <span id="Label1" style="Z-INDEX: 104; LEFT: 45px; POSITION: absolute;
TOP: 18px">Выберите интересующую Вас книгу</span>


<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
  var theform = document.Form1;
  theform.__EVENTTARGET.value = eventTarget;
  theform.__EVENTARGUMENT.value = eventArgument;
  theform.submit();
 }
// -->
</script>
</form>
 </body>
</HTML>
```

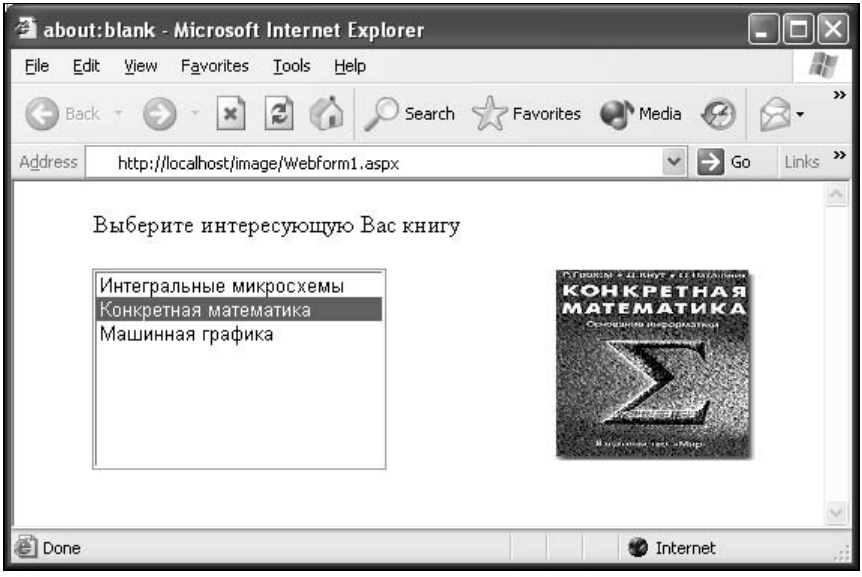


Рис. 3.17. Отображение Web-страницы с динамически изменяемым рисунком

Следует обратить внимание, что в этом примере при выборе пользователем какого-либо пункта списка, данные об этом сначала уходят на сервер, а потом оттуда в браузер пересылается новая страница, т. е. мы имеем дело с интерактивной системой, опирающейся на сервер, а не выполняющейся полностью на стороне пользователя.

Впрочем, компонент `Image` не является единственным компонентом, ориентированным на работу с графикой. На вкладке **Web Forms** (Web-формы) можно обнаружить еще один компонент с наименованием `AdRotator`. Это, по сути дела, заготовка для прокрутки баннеров. По умолчанию ее размеры шестьдесят на четыреста шестьдесят пикселов. Ничего не напоминает?

Даже если судить по наименованию этого компонента, он предназначен для прокрутки баннеров. Естественно, не просто для отображения одного баннера, для этих целей мы могли воспользоваться и компонентом `Image`, а именно прокрутки нескольких заранее заготовленных баннеров. Конечно, разработчик должен иметь возможность управлять количеством отображаемых баннеров, некоторые из них показывать чаще, другие реже. Крайне желательна возможность отключать целые категории баннеров. Все это уже реализовано в компоненте `AdRotator`. Разработчики Visual Studio .NET очень хорошо позаботились о предоставлении хорошего движка для рекламы в будущих проектах сторонних производителей.

Основой управления отображения баннеров служит свойство `AdvertisementFile`. Это обычное свойство строкового типа `String`, в котором содержится путь к определенному XML-файлу. Об XML будет подробно рассказано в сле-

дующей главе, а пока что мы попробуем обойтись без пространных лекций, и на маленьком, но понятном примере рассмотрим применение подобного XML-файла, который позволяет управлять отображением рекламы в приложениях ASP.NET.

В листинге 3.18 приведен код XML-файла с наименованием adv.xml, который был размещен в виртуальном каталоге `images`.

Листинг 3.18

```
<Advertisements>
    <Ad>
        <ImageUrl>1.gif</ImageUrl>
        <NavigateUrl>http://www.computerbook.ru</NavigateUrl>
        <AlternateText>Computerbook.ru — весь мир компьютерных книг
        </AlternateText>
        <Impressions>50</Impressions>
        <Keyword>oursite</Keyword>

    </Ad>
    <Ad>
        <ImageUrl>2.gif</ImageUrl>
        <NavigateUrl>http://www.amazone.com</NavigateUrl>
        <AlternateText>amazone.com — легендарный книжный магазин
        </AlternateText>
        <Impressions>50</Impressions>
        <Keyword>frends</Keyword>

    </Ad>
    <Ad>
        <ImageUrl>3.gif</ImageUrl>
        <NavigateUrl>http://www.computerbook.ru</NavigateUrl>
        <AlternateText>Computerbook.ru — весь мир компьютерных книг
        </AlternateText>
        <Impressions>50</Impressions>
        <Keyword>oursite</Keyword>

    </Ad>
</Advertisements>
```

Легко увидеть, что по своей структуре и синтаксису XML очень похож на HTML. И это вполне понятно, ведь родитель у них общий — язык SGML

(Standard Generalized Markup Language). Чтобы не вдаваться в долгие объяснения, которым место найдется только в одной из следующих глав, скажем пока максимально просто — язык XML позволяет разработчику создавать свои собственные теги, и пользоваться ими для логического форматирования содержимого документа. Как и в HTML, разработчик XML-документов получает возможность располагать одни теги внутри других. Что, собственно говоря, отлично видно в листинге 3.18.

Все содержимое файла заключается между парой тегов `<Advertisements>` и `</Advertisements>`. Это содержимое разбито на отдельные однотипные блоки, каждый из которых описывает один баннер, участвующий в рекламной системе. Эти блоки ограничиваются тегами `<Ad>` и `</Ad>`. Каждый из таких блоков описывает один баннер, который будет отображаться в данном компоненте. Внутри этих блоков располагаются одинаковые наборы тегов, задающие свойства баннеров.

Между тегами `<ImageUrl>` и `</ImageUrl>` располагается текстовая строка, в которой указывается URL графического файла с изображением самого баннера. Следует указать, что данная строка является содержимым файла, а не параметром какого-либо тега, поэтому она не должна заключаться в кавычки.

Каждый баннер является не только простым изображением, он одновременно выполняет роль и гиперссылки. Поэтому между тегами `<NavigateUrl>` и `</NavigateUrl>` располагается URL того ресурса, на который должна будет указывать гиперссылка, связанная с отображаемым баннером.

Давно известно, что для Web-дизайнеров и Web-разработчиков хорошим тоном служит связывание графических изображений с дополнительным текстом, который будет отображаться на месте рисунка в том случае, если у пользователя в браузере отключено отображение графики. В нашем случае, для каждого баннера соответствующий текст будет располагаться между тегами `<AlternateText>` и `</AlternateText>`.

Любая баннерная система может управлять количеством показов того или иного баннера. Частота показа каждого баннера регулируется числом, заключенным между тегами `<Impressions>` и `</Impressions>`. Это не просто некое количество показов, а частота его показа, относительно иных баннеров. Естественно, эти числа являются некими долями относительно их общей суммы.

И, наконец, осталось лишь рассмотреть значение последней пары тегов — `<Keyword>` и `</Keyword>`. Между ними располагается некое ключевое слово, которое на самом деле является наименованием категории. На самом деле разработчик имеет возможность отключать воспроизведение баннеров, принадлежащих той или иной категории.

Помимо XML-файла, который описывает коллекцию баннеров, применяемых в оформлении страницы, следует установить значение свойства `Target`.

Значением этого свойства является наименование фрейма или окна просмотра браузера, в котором будет открыт документ, связанный с отображающимся баннером. Подобные наименования, а точнее ключевые слова, жестко прописаны в спецификации HTML, и являются значениями одноименного параметра `target`, часто применявшегося вместе с тегом гиперссылки `<a>`.

На этом обзор методов работы с различными компонентами, связанными с графическими изображениями, заканчивается.

Работа с файлами

При создании любого мало-мальски объемного сайта, разработчику редко удается обойтись без использования информации, хранящейся в обычных файлах, а не в статичных Web-страницах или в записях таблиц баз данных. В этом разделе мы узнаем, как работать с файлами в приложениях ASP.NET. Для начала мы попробуем усовершенствовать приложение, рассмотренное в предыдущем разделе. Как мы помним, в этом приложении пользователю предлагалось сделать выбор из трех книг, и в зависимости от его выбора отображалась обложка соответствующей книги. Попробуем пойти дальше, и предоставить пользователю кроме обложки еще и аннотацию выбранной книги. Для этого нам потребуется помимо трех графических файлов иметь еще и три текстовых файла, в которых будут находиться искомые аннотации. Чтобы не отступать от наметившейся традиции, назовем эти файлы 1.txt, 2.txt и 3.txt.

Для того чтобы отображать аннотации книг, поместим на разрабатываемую Web-страницу компонент с наименованием `Label`, читать содержимое соответствующего текстового файла с аннотацией, и записывать это содержимое в свойство `Text` элемента `Label`. Естественно, как и с компонентом `Image`, мы сначала сделаем компонент `Label` невидимым, а затем при отображении аннотаций принудительно визуализируем его.

Однако перед тем как приступить к разработке приложения, следует разобраться в механизмах работы с файлами, принятыми в Visual Basic .NET. В любом языке программирования для работы с каким-либо файлом используются некие идентификаторы. Часто используют переменные специализированного файлового типа. Однако в Visual Basic .NET применяется идеология *дескрипторов* файлов. Дескриптором файла называют его номер, т. е. каждому открываемому файлу присваивается некий уникальный номер, и при работе с файлом, функциям передается этот номер в качестве параметра. Вопрос о том, что *правильнее* использовать в языках программирования — дескриптор или переменную файлового типа, на самом деле, не имеет какого-либо реального значения. Неверно считать, что использование переменной файлового типа является признаком хорошо типизированного

языка программирования. Я останавливаюсь на этом вопросе именно потому, что часто приходится слышать мнения, будто работа с файлами при помощи дескрипторов является признаком слабого языка программирования. Увы, это не так. Следует помнить, что механизм дескрипторов использовался и в языке C, который уж никак нельзя упрекнуть в слабости. Однако вернемся к нашим файлам.

Для того чтобы работать с файлом, записывать в него информацию, или читать ее, необходимо сначала открыть это файл. Открытие файла производится при помощи функции `FileOpen`. Естественно, данная функция обладает целым рядом параметров. В качестве первого обязательного параметра типа `Integer` передается номер файла, его уникальный дескриптор. Разработчик должен указать именно тот номер, который еще не присвоен ни одному открытому файлу. Вторым обязательным параметром является имя открываемого файла. Естественно, данный параметр имеет тип `String`. Имя может указываться с использованием полного пути к файлу, включая наименования каталогов и диска, на котором он располагается. В третьем параметре функции указывается режим, в котором открывается файл. В качестве значения данного параметра применяется один из членов перечислимого типа `OpenMode`. Значение `Append` указывает, что файл будет открыт для того, чтобы добавить некоторые данные к нему. Значение `Binary` применяется в том случае, если файл содержит некие неструктурированные в виде текста или иных типов данные. Иначе говоря, файл представляет собой просто поток некоей байтовой информации, которую не следует по умолчанию интерпретировать как текст. Значение `Output` указывает, что в файл будет записываться некая информация, причем, совершенно необязательно она будет добавляться именно в конец открытого файла, как это будет происходить при использовании режима `Append`. Значение `Input` открывает файл для чтения информации из него. Если же планируется совершать с файлом операции как записи, так и чтения, следует воспользоваться значением `Random`, которое открывает приложению полный доступ к файлу.

Четвертый параметр указывает тип доступа к файлу. Файл может быть открыт для чтения, записи, или и для чтения, и для записи одновременно. Соответственно, в качестве значения параметра используется одно из трех ключевых слов: `Read`, `Write` или `ReadWrite`. Все эти слова являются членами перечислимого типа `OpenAccess`. Четвертый параметр не является обязательным для функции `FileOpen`. По умолчанию используется значение `ReadWrite`.

Следует осознавать, что один и тот же файл может одновременно понадобиться нескольким приложениям. Одновременная запись данных в файл может безнадежно перепутать все данные в нем. Следовательно, необходимо каким-либо образом предусмотреть возможные коллизии. Проблемы могут возникнуть не только при одновременной записи данных в файл. Одновременное чтение данных может тоже повредить если не самому файлу,

то читающему его приложению. Дело в том, что при чтении или записи данных в файле используется указатель на то место, в которое будет производиться запись, или откуда будут читаться данные. Поэтому при открытии файла следует заранее устанавливать режим совместного доступа к нему. Данный режим передается в функцию в качестве четвертого параметра. Этот параметр не является обязательным. В качестве значения параметра может использоваться один из четырех членов перечислимого типа `OpenShare`: `Shared`, `LockRead`, `LockWrite` или `LockReadWrite`. Значение `Shared` открывает этот файл (часто говорят, "расшаривает файл") для любых операций из каждого приложения, обратившегося к данному файлу. Именно это значение и этот режим доступа к файлу используются по умолчанию. Значение `LockRead` блокирует указатель позиции, с которой будет производиться чтение. До тех пор, пока приложение, открывшее файл, не закроет его, ни одно другое приложение не сможет ничего прочитать из искомого файла. Значение `LockWrite`, соответственно, закрывает иным приложениям доступ на запись данных в файл, но позволяет им считывать информацию из файла. А значение `LockReadWrite` полностью монополизирует файл для открывшего его приложения, не оставляя никакого доступа для иных процессов.

В состав вызова функции `FileOpen` может входить и еще один атрибут. Дело в том, что достаточно часто в файлах записываются порции данных фиксированной длины. Таким образом, программисту придется прикладывать дополнительные усилия для правильного позиционирования указателя чтения из файла. Конечно, это не самое трудное в программировании, но если есть возможность избавиться хотя бы от одной операции, следует воспользоваться этой возможностью. Поэтому было бы неплохо при перемещении указателя устанавливать не количество байтов, на которое его следует передвинуть, а количество записей. Однако для этого необходимо знать, сколько байтов входит в одну запись. И в качестве последнего атрибута в функции открытия файла мы можем указать целое число, которое и будет размером одной записи в байтах. Максимальный размер записи может составлять 32 676 байтов.

Что касается обязательных параметров функции `FileOpen`, то с ними все понятно. Их три, и они должны быть обязательно указаны в том порядке, в котором мы их рассмотрели. Но что делать с необязательными параметрами. Они тоже должны располагаться в вызове функции по порядку. Если же нам необходимо использовать некий необязательный параметр, перед ним может оказаться другой параметр, который нам не нужен. Как быть в таком случае?

Тогда на месте ненужных параметров можно просто оставить пустое место, которое будет отделено от других параметров при помощи запятых. Рассмотрим это на примере. Предположим, нам надо открыть некий файл для чтения и записи, и при этом заблокировать доступ на чтение и запись

в файл для других приложений. В этом случае вызов функции будет иметь приблизительно следующий вид:

```
FileOpen(1, "c:\temp\tmpfile.txt", OpenMode.Random,,
OpenShare.LockReadWrite)
```

Здесь мы открываем файл с дескриптором, равным единице. Имя файла мы указываем в полном варианте, с указанием пути к файлу. Использование остальных параметров уже должно быть понятно.

Однако есть еще один вопрос. Как разработчик может знать, какой дескриптор еще не занят под открытый файл? Если приложению требуется использовать заранее известное количество файлов, то все в порядке, разработчик сможет в коде программы жестко установить все необходимые номера. Но как поступить, если требуется использовать заранее не определенное количество файлов, причем, некоторые из них могут несколько раз открываться и закрываться в процессе работы, усложняя учет свободных дескрипторов? Конечно, можно вести учет использованных дескрипторов, записывая их номера в некоей таблице, но можно поступить проще. В состав Visual Basic .NET входит достаточно интересная функция `FreeFile`, возвращающая целое число, которое гарантированно может быть использовано в виде дескриптора, т. е. подобное число не используется для идентификации какого-либо файла, открытого в данный момент.

После того, как все операции с файлом завершены, и его повторное использование не планируется в ближайший момент, стоит закрыть его. Закрывать ранее открытые файлы следует обязательно, так как удержание их в открытом виде забирает некоторую часть системных ресурсов. Более того, следует учитывать, что после того как приложение дает команду на запись некоторых данных в файл, эти данные записываются не сразу. Сначала они заносятся в некий буфер, и только после его заполнения сбрасываются на диск. При закрытии файла все буферы, связанные с закрываемым файлом, уничтожаются, и данные физически записываются в файл. Поэтому следует закрывать файлы сразу после окончания серии операций с ним. Закрывается файл при помощи функции `FileClose`, которой в качестве единственного целочисленного параметра передается дескриптор файла, подлежащего закрытию. Впрочем, если необходимо сразу закрыть все открытые файлы, не обязательно использовать для каждого из них функцию `FileClose`. Можно воспользоваться функцией `Reset`, которая сразу закрывает все открытые ранее приложением файлы.

Итак, теперь мы знаем, как открывать и закрывать файлы. Однако, этого мало для настоящей работы. Необходимо еще уметь записывать данные в них, и читать из них информацию. Для чтения данных из текстовых файлов, а нам придется работать именно с такими файлами, обычно применяется функция `LineInput`. Она возвращает значение типа `String`. В него заносится строка символов, которые функция читает с того места, на котором был закончен предыдущий сеанс чтения, и до того момента, пока не встре-

тится знак возврата каретки, обладающий кодом 13, или последовательность символов с кодами 13 и 10, т. е. возврат каретки и переход на новую строку. В качестве параметра функции `LineInput` передается целочисленный дескриптор файла, из которого необходимо производить чтение текста.

В том случае, если в читаемом файле находится не только текст, использование функции `LineInput` уже не будет оправданным. Для чтения разнородной информации из файла следует применять функцию `Input`. Функции `Input` в качестве параметров передается дескриптор файла, из которого производится чтение, и переменная, в которую будет записываться читаемая информация. Если переменная строкового типа, то в нее будет записана строка текста, если переменная числового типа, то в нее будет записано число. Естественно, разработчик должен точно помнить, в каком порядке записывались данные в файл и сам контролировать соответствие типов данных и переменных, в которые эти данные читаются.

Для записи данных в файл используются функции `Write` и `WriteLine`. Функции `Write` в качестве параметров передается дескриптор файла, в который необходимо записать информацию, и переменная, содержимое которой необходимо поместить в файл. Причем, совершенно необязательно, чтобы эта переменная была строкового типа. Записывать в файл можно данные любого типа. Синтаксис функции `WriteLine` практически идентичен синтаксису `Write`, за тем исключением, что в качестве второго параметра передается не одна переменная, а их массив. При этом следует упомянуть, что этот массив также не обязан содержать однотипные элементы. В одном вызове функции `WriteLine` можно записать в файл последовательно данные различных типов.

В языке VisualBasic .NET существует еще одна пара функций, которые позволяют осуществлять запись информации в файл. Они носят наименования `Print` и `PrintLine`. Отличие их от функций `Write` и `WriteLine` заключается в том, что они записывают информацию в том виде, как она будет отображаться на экране. Это не значит, что данные функции могут работать только со строковыми переменными, просто при записи данных в файл разработчик получает возможность использовать некий паллиатив форматирования записываемых строк при помощи некоторых дополнительных выражений. В состав записываемой информации могут входить выражения `SPC(n)` и `TAB(n)`, которые позволяют включать в состав записываемых строк `n` пробелов и `n` символов табуляции, соответственно. Таким образом, если нам необходимо записать в файл строку со словами "Здравствуй, мир", причем не в обычном виде, а чтобы слова были отделены друг от друга десятью пробелами, следует воспользоваться следующим фрагментом кода:

```
PrintLine(1, "Здравствуй,", SPC(10), "мир")
```

Отличие функций `Print` и `PrintLine` друг от друга нетрудно проследить по их аналогам — функциям `Write` и `WriteLine`. В вызов функции `Print` могут

входить только два параметра — дескриптор открытого файла и записываемое значение, а в вызове функции `PrintLine` мы можем указывать несколько записываемых значений.

Теперь, когда мы рассмотрели методы работы с файлами в рамках языка Visual Basic .NET, мы можем перейти к рассмотрению листинга приложения, с описания которого начинался данный раздел. В листинге 3.19 приведен HTML-код разрабатываемой страницы в том виде, как он создан в среде разработки Visual Studio .NET

Листинг 3.19

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="image.WebForm1"%>
<!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:listbox id="ListBox1" style="Z-INDEX: 101; LEFT: 31px; POSITION:
absolute; TOP: 53px" runat="server" AutoPostBack="True" Rows="3"
Height="79px" Width="187px">
    <asp:ListItem Value="1">Интегральные микросхемы</asp:ListItem>
    <asp:ListItem Value="2">Конкретная математика</asp:ListItem>
    <asp:ListItem Value="3">Машинная графика</asp:ListItem>
   </asp:listbox>
   <asp:image id="Image1" style="Z-INDEX: 102; LEFT: 249px; POSITION:
absolute; TOP: 60px" runat="server" Height="128px" Width="135px"
Visible="False"></asp:image>
   <asp:Label id="Label1" style="Z-INDEX: 103; LEFT: 39px; POSITION:
absolute; TOP: 204px" runat="server" Height="46px" Width="339px"
Visible="False"></asp:Label>
  </FORM>
 </BODY>
</HTML>
```

В листинге 3.20 приведен код класса, реализующего данную Web-страницу.

Листинг 3.20

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Image1 As System.Web.UI.WebControls.Image
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents ListBox1 As System.Web.UI.WebControls.ListBox

#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 ListBox1_SelectedIndexChanged(ByVal sender
    As System.Object, ByVal e As System.EventArgs) Handles
    ListBox1.SelectedIndexChanged
        Dim i As Integer
        Dim s As String
        i = FreeFile()
        If ListBox1.SelectedIndex = 0 Then
            Image1.ImageUrl = "1.gif"
            FileOpen(i, "d:\1.txt", OpenMode.Input)
```

```
            s = LineInput(i)
            FileClose(i)
            Label1.Text = s
        End If
        If ListBox1.SelectedIndex = 1 Then
            Image1.ImageUrl = "2.gif"
            FileOpen(i, "d:\2.txt", OpenMode.Input)
            s = LineInput(i)
            FileClose(i)
            Label1.Text = s
        End If
        If ListBox1.SelectedIndex = 2 Then
            Image1.ImageUrl = "3.gif"
            FileOpen(i, "d:\3.txt", OpenMode.Input)
            s = LineInput(i)
            FileClose(i)
            Label1.Text = s
        End If
        Image1.Visible = True
        Label1.Visible = True
    End Sub
End Class
```

На рис. 3.18 показан внешний вид разработанной нами интерактивной Web-страницы в браузере Internet Explorer. На изображении виден вариант содержимого сгенерированного сервером в ответ на выбор пользователем одного из пунктов списка.

При этом HTML-код страницы, отображенной в браузере, естественно выглядит иначе, чем HTML-код при разработке. Он приведен в листинге 3.21.

Листинг 3.21

```
<!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 name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtODY1NzQ5NTM7dDw7bDxpPDE+Oz47bDx0PDtsPGk8MT47aTwzPjtpPDU+Oz47bD
x0PHQ8OztsPGk8MT47Pj47Oz47dDxwPHA8bDxJbWFnZVVybDtWaXNpYmxlOz47bDwyLmdpZjt
vPHQ+Oz4+Oz47Oz47dDxwPHA8bDxUZXh0O1Zpc2libGU7PjtsPNCS0YLQvtGA0LDRjyDQsNC9
0L3QvtGC0LDRhtC40Y87bzx0Pjs+Pjs+Ozs+Oz4+Oz4+Oz4=" />


   <select name="ListBox1" id="ListBox1" size="3"
onchange="__doPostBack('ListBox1','')" language="javascript"
style="height:79px;width:187px;Z-INDEX: 101; LEFT: 31px; POSITION:
absolute; TOP: 53px">
 <option value="1">Интегральные микросхемы</option>
 <option selected="selected" value="2">Конкретная математика</option>
 <option value="3">Машинная графика</option>


</select>
   <img id="Image1" src="/image/2.gif" border="0"
style="height:128px;width:135px;Z-INDEX: 102; LEFT: 249px; POSITION:
absolute; TOP: 60px" />
   <span id="Label1" style="height:46px;width:339px; Z-INDEX: 103; LEFT:
39px; POSITION: absolute; TOP: 204px">Вторая аннотация</span>


<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
  var theform = document.Form1;
  theform.__EVENTTARGET.value = eventTarget;
  theform.__EVENTARGUMENT.value = eventArgument;
  theform.submit();
 }
// -->
</script>
</form>
```

Однако мы рассмотрели всего лишь одну операцию с файлами — чтение. Следует еще понять, как осуществляется запись в файлы. Для этого мы еще немного расширим функциональность нашего примера.

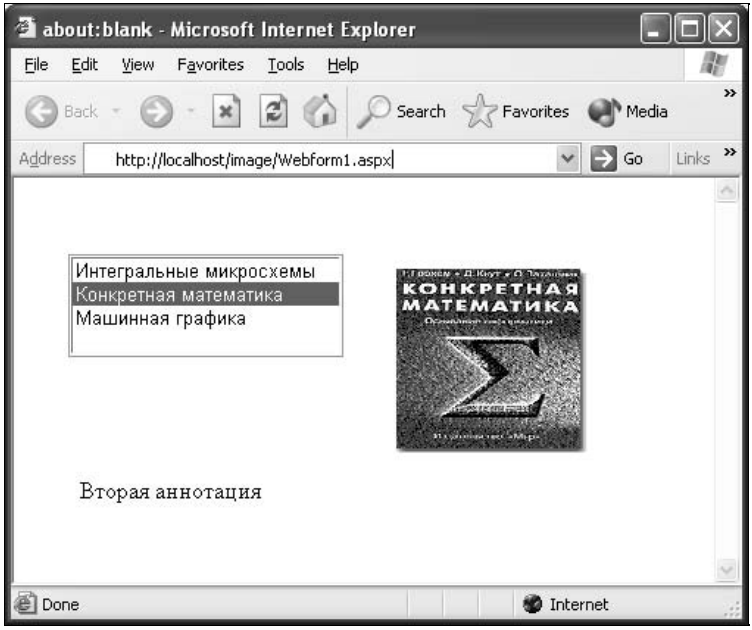


Рис. 3.18. Отображение Web-страницы с динамически изменяемым рисунком и отображаемой аннотацией

Попробуем получить отзывы от читателей о тех книгах, информацию о которых мы показываем. Для этого следует разместить на разрабатываемой Web-странице многострочное поле для ввода текста отзыва и кнопку, отсылающую введенную информацию на сервер. Итак, на страницу выкладывается компонент `Label`, в свойстве `Text` которого записывается строка "Ваш отзыв о книге". Также нам потребуется один компонент `TextBox`. Для его свойства `TextMode` мы установим значение `MultiLine`, чтобы удаленный пользователь мог ввести несколько строк, а не одну. Также потребуется один элемент `Button`, для отсылки данных на сервер. В листинге 3.22 приведен HTML-код страницы в том виде, как она создана в среде разработки Visual Studio.NET.

Листинг 3.22

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="image.WebForm1"%>
<!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:listbox id="ListBox1" style="Z-INDEX: 101; LEFT: 22px; POSITION:
absolute; TOP: 19px" runat="server" Width="187px" Height="79px" Rows="3"
AutoPostBack="True">
    <asp:ListItem Value="1">Интегральные микросхемы</asp:ListItem>
    <asp:ListItem Value="2">Конкретная математика</asp:ListItem>
    <asp:ListItem Value="3">Машинная графика</asp:ListItem>
   </asp:listbox>
   <asp:image id="Image1" style="Z-INDEX: 102; LEFT: 302px; POSITION:
absolute; TOP: 24px" runat="server" Width="135px" Height="128px"
Visible="False"></asp:image>
   <asp:label id="Label1" style="Z-INDEX: 103; LEFT: 22px; POSITION:
absolute; TOP: 104px" runat="server" Width="249px" Height="43px"
Visible="False"></asp:label>
   <asp:label id="Label2" style="Z-INDEX: 104; LEFT: 25px; POSITION:
absolute; TOP: 169px" runat="server">Ваш отзыв о книге</asp:label>
   <asp:textbox id="TextBox1" style="Z-INDEX: 105; LEFT: 29px; POSITION:
absolute; TOP: 193px" runat="server" Width="407px" Height="89px"
TextMode="MultiLine"></asp:textbox>
   <asp:button id="Button1" style="Z-INDEX: 106; LEFT: 29px; POSITION:
absolute; TOP: 298px" runat="server" Width="192px" Height="24px"
Text="Оставить отзыв"></asp:button>
  </FORM>
 </BODY>
</HTML>
```

Сценарий работы с добавленными компонентами будет достаточно прост. Как только пользователь выбирает какой-либо пункт в основном списке, отображаются обложка указанной книги и ее аннотация, кроме того, необходимо подготовиться к записи данных из поля ввода в соответствующий текстовый файл. Для каждой книги отведен отдельный файл, в который последовательно будут записываться комментарии пользователей. Следует осознавать, что блоки кода, ответственные за отображение обложки и прием данных от пользователя, разделены. Отображение аннотации и обложки завязаны на выбор элемента в списке, а отсылка данных из поля ввода инициируется нажатием на кнопку. Но как тогда при нажатии на кнопку отсылки данных, приложение будет знать, какой пункт выбран пользователем. Можно просто обратиться к свойству `SelectedIndex`, присущему компоненту

ListBox. Но мы попробуем продемонстрировать передачу данных через объект Session. Для этого при загрузке страницы мы создадим один элемент коллекции Session, и каждый раз при выборе того или иного элемента списка будем присваивать соответствующее значение этому элементу. Затем, при нажатии пользователем кнопки на Web-странице, приложение будет ориентироваться на значение этого элемента коллекции Session для того, чтобы выбрать имя необходимого файла, в который будет записан отзыв удаленного пользователя. После этого соответствующий файл будет открыт для добавления данных в него, и в него следует записать тот текст, который ввел пользователь в текстовое поле. После того, как данные будут записаны в файл, его следует закрыть, изменить текст элемента Label2, чтобы показать пользователю, что его отзыв принят, а также скрыть поле текстового ввода и кнопку отсылки данных, чтобы предотвратить повторный ввод комментария. На самом деле следует несколько более тщательно отслеживать ввод повторных комментариев, с запоминанием пользователя, который уже установил отзыв, но сейчас мы не будем останавливаться на подобном механизме отслеживания. Приемы идентификации пользователей будут описаны несколько позже.

Однако следует предоставить пользователю возможность ввести отзывы и о других книгах. Поэтому, когда пользователь будет выбирать какой-либо элемент списка, следует привести все видоизмененные и скрытые элементы в исходное состояние. Весь код, реализующий функциональность данного приложения, показан в листинге 3.23.

Листинг 3.23

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Image1 As System.Web.UI.WebControls.Image
    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 Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents ListBox1 As System.Web.UI.WebControls.ListBox

#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
        Session.Add("ch", 0)
    End Sub

    Private Sub ListBox1_SelectedIndexChanged(ByVal sender
    As System.Object, ByVal e As System.EventArgs) Handles
    ListBox1.SelectedIndexChanged
        Dim i As Integer
        Dim s As String
        i = FreeFile()
        Label1.Text = "Ваш отзыв о книге"
        TextBox1.Text = ""
        TextBox1.Visible = True
        Button1.Visible = True
        If ListBox1.SelectedIndex = 0 Then
            Image1.ImageUrl = "1.gif"
            FileOpen(i, "d:\1.txt", OpenMode.Input)
            s = LineInput(i)
            FileClose(i)
            Label1.Text = s
            Session.Item("ch") = 0
        End If
        If ListBox1.SelectedIndex = 1 Then
            Image1.ImageUrl = "2.gif"
            FileOpen(i, "d:\2.txt", OpenMode.Input)
            s = LineInput(i)
            FileClose(i)
            Label1.Text = s
```

```
            Session.Item("ch") = 1
        End If
        If ListBox1.SelectedIndex = 2 Then
            Image1.ImageUrl = "3.gif"
            FileOpen(i, "d:\3.txt", OpenMode.Input)
            s = LineInput(i)
            FileClose(i)
            Label1.Text = s
            Session.Item("ch") = 2
        End If
        Image1.Visible = True
        Label1.Visible = True
    End Sub


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles Button1.Click
        Dim s As String
        Dim fn As String
        Dim i As Integer
        s = TextBox1.Text
        If Session.Item("ch") = 0 Then
            fn = "d:\op1.txt"
        End If
        If Session.Item("ch") = 1 Then
            fn = "d:\op2.txt"
        End If
        If Session.Item("ch") = 2 Then
            fn = "d:\op3.txt"
        End If
        i = FreeFile()
        FileOpen(i, fn, OpenMode.Append)
        WriteLine(i, s)
        Label2.Text = "Ваш отзыв принят"
        TextBox1.Visible = False
        Button1.Visible = False
        FileClose(i)
    End Sub
End Class
```

Рассмотрим листинг чуть более подробно. При загрузке страницы обрабатывается событие `Page_Load`. В этом обработчике мы создаем элемент коллекции объекта `Session` с наименованием `ch`. В данном случае мы используем объект `Session`, а не `Application` для того, чтобы каждый удаленный пользователь обладал собственным элементом коллекции. Если бы мы использовали объект `Application`, то на всех пользователей был бы выделен один элемент коллекции, и они могли бы пытаться использовать его одновременно, портя, таким образом, данные друг для друга.

Когда пользователь выбирает тот или иной пункт списка, приложение вызывает обработчик события `ListBox1_SelectedIndexChanged`. Мы уже сталкивались с ним в предыдущих примерах, однако сейчас в этот обработчик добавлено несколько новых строк кода. Прежде всего, стоит обратить внимание на начало обработчика, где органы управления, ответственные за получение отзыва о книге, приводятся в начальное состояние. Также добавлены операторы, устанавливающие значение используемого элемента коллекции `Session`.

Большая часть функциональности, связанной с обработкой данных, введенных удаленным пользователем, сосредоточена в обработчике нажатия кнопки `Button1_Click`. Сначала в переменную `s` типа `String` заносится содержимое свойства `Text` элемента `TextBox1`. Затем, опираясь на значение элемента коллекции `Session`, в переменную `fn` заносится имя файла, в который будет записываться отзыв посетителя. При помощи функции `FreeFile` в целочисленную переменную `i` заносится значение свободного дескриптора для файла, и при помощи этого дескриптора открывается соответствующий файл. Файл открывается в режиме `OpenMode.Append`, что позволяет дозаписывать в него информацию. А после этого при помощи функции `WriteLine` в открытый файл записывается отзыв посетителя сайта, и файл закрывается. Также прячутся от пользователя компоненты `TextBox1` и `Button1`, а свойству `Text` компонента `Label2` придается значение "Ваш отзыв принят".

HTML-код страницы в тот момент, когда пользователь только выбрал какой-либо элемент списка, ввел комментарий для книги, но не отправил его на сервер, приведен в листинге 3.24.

Листинг 3.24

```
<!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 name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNzUyMjUwNTQ1O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+O2k8Mz47aTw1Pjs+O2
w8dDx0PDs7bDxpPDE+Oz4+Ozs+O3Q8cDxwPGw8SW1hZ2VVcmw7VmlzaWJsZTs+O2w8Mi5naWY
7bzx0Pjs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDtWaXNpYmxlOz47bDzQktGC0L7RgNCw0Y8g0LDQ
vdC90L7RgtCw0YbQuNGPO288dD47Pj47Pjs7Pjs+Pjs+Pjs+" />


   <select name="ListBox1" id="ListBox1" size="3"
onchange="__doPostBack('ListBox1','')" language="javascript"
style="height:79px;width:187px;Z-INDEX: 101; LEFT: 22px; POSITION:
absolute; TOP: 19px">
 <option value="1">Интегральные микросхемы</option>
 <option selected="selected" value="2">Конкретная математика</option>
 <option value="3">Машинная графика</option>


</select>
   <img id="Image1" src="/image/2.gif" border="0"
style="height:128px;width:135px;Z-INDEX: 102; LEFT: 302px; POSITION:
absolute; TOP: 24px" />
   <span id="Label1" style="height:43px;width:249px;Z-INDEX: 103; LEFT:
22px; POSITION: absolute; TOP: 104px">Вторая аннотация</span>
   <span id="Label2" style="Z-INDEX: 104; LEFT: 25px; POSITION: absolute;
TOP: 169px">Ваш отзыв о книге</span>
   <textarea name="TextBox1" id="TextBox1"
style="height:89px;width:407px;Z-INDEX: 105; LEFT: 29px; POSITION:
absolute; TOP: 193px"></textarea>
   <input type="submit" name="Button1" value="Оставить отзыв"
id="Button1" style="height:24px;width:192px;Z-INDEX: 106; LEFT: 29px;
POSITION: absolute; TOP: 298px" />


<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
  var theform = document.Form1;
  theform.__EVENTTARGET.value = eventTarget;
  theform.__EVENTARGUMENT.value = eventArgument;
  theform.submit();
```

```
 }
// -->
</script>
</form>
 </BODY>
</HTML>
```

На рис. 3.19 показан внешний вид этой Web-страницы при отображении ее в браузере Internet Explorer.

Рис. 3.19. Внешний вид Web-страницы, HTML-код которой приведен в листинге 3.24

После того, как пользователь нажмет на кнопку **Оставить отзыв**, внешний вид Web-страницы изменится. Соответственно, изменится и ее HTML-код. Он приведен в листинге 3.25.

Листинг 3.25

```
<!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 name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNzUyMjUwNTQ1O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+O2k8Mz47aTw1PjtpPD
c+O2k8OT47aTwxMT47PjtsPHQ8dDw7O2w8aTwxPjs+Pjs7Pjt0PHA8cDxsPEltYWdlVXJsO1Z
pc2libGU7PjtsPDIuZ2lmO288dD47Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7VmlzaWJsZTs+O2w8
0JLRgtC+0YDQsNGPINCw0L3QvdC+0YLQsNGG0LjRjztvPHQ+Oz4+Oz47Oz47dDxwPHA8bDxUZ
Xh0Oz47bDzQktCw0Ygg0L7RgtC30YvQsiDQv9GA0LjQvdGP0YI7Pj47Pjs7Pjt0PHA8cDxsPF
RleHQ7VmlzaWJsZTs+O2w80JjQvdGC0LXRgNC10YHQvdCw0Y8g0LrQvdC40LPQsC4NCtCc0L3
QtSDQv9C+0L3RgNCw0LLQuNC70LDRgdGMLi4uO288Zj47Pj47Pjs7Pjt0PHA8cDxsPFZpc2li
bGU7PjtsPG88Zj47Pj47Pjs7Pjs+Pjs+Pjs+" />


   <select name="ListBox1" id="ListBox1" size="3"
onchange="__doPostBack('ListBox1','')" language="javascript"
style="height:79px;width:187px;Z-INDEX: 101; LEFT: 22px; POSITION:
absolute; TOP: 19px">
 <option value="1">Интегральные микросхемы</option>
 <option selected="selected" value="2">Конкретная математика</option>
 <option value="3">Машинная графика</option>

</select>
   <img id="Image1" src="/image/2.gif" border="0"
style="height:128px;width:135px;Z-INDEX: 102; LEFT: 302px; POSITION:
absolute; TOP: 24px" />
   <span id="Label1" style="height:43px;width:249px;Z-INDEX: 103; LEFT:
22px; POSITION: absolute; TOP: 104px">Вторая аннотация</span>
   <span id="Label2" style="Z-INDEX: 104; LEFT: 25px; POSITION: absolute;
TOP: 169px">Ваш отзыв принят</span>

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
  var theform = document.Form1;
  theform.__EVENTTARGET.value = eventTarget;
```

```
  theform.__EVENTARGUMENT.value = eventArgument;
  theform.submit();
 }
// -->
</script>
</form>
</BODY>
</HTML>
```

На рис. 3.20 показан внешний вид этой Web-страницы при отображении ее в браузере Internet Explorer.

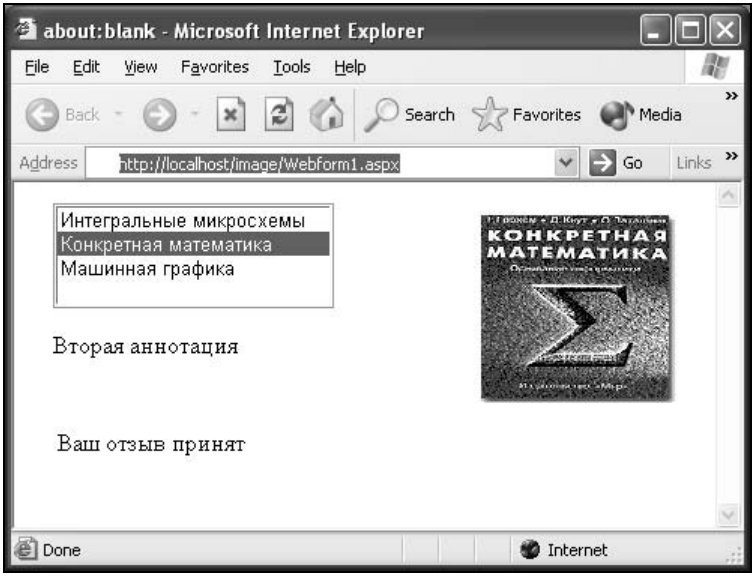


Рис. 3.20. Внешний вид Web-страницы, HTML-код которой приведен в листинге 3.25

Теперь мы видим, что работа с файлами в среде ASP.NET не представляет особого труда. Все это достаточно легко и просто.

Работа с объектами *HttpRequest* и *HttpResponse*

Сами встроенные объекты `HttpRequest` и `HttpResponse` мы уже рассматривали в одном из разделов данной главы. Однако это был лишь обзор их свойств и методов. Сейчас пришло время несколько более подробно рассмотреть механизмы работы с этими встроенными объектами.

Начнем мы с самого простого варианта работы. Как мы помним, объект `HttpRequest` содержит в себе информацию, которая пришла на сервер от браузера удаленного пользователя. Попробуем получить из этого запроса все данные, которые сможем.

Воспользуемся предыдущим примером с передачей данных из формы. Но для обрабатывающей страницы мы установим код, приведенный в листинге 3.26. Так как ее содержимое создается динамически при помощи объекта `Response`, а точнее, его метода `Write`, нам не потребуется размещать на разрабатываемой Web-странице какие-либо компоненты.

Листинг 3.26

```
Public Class WebForm2
    Inherits System.Web.UI.Page

#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 n As String
        Dim t As Boolean
        Dim i As Integer
        n = Request.ApplicationPath
        Response.Write("Путь к приложению — " + n)
        Response.Write("<p>Браузер пользователя")
```

```
n = Request.Browser.Browser
Response.Write("<p>Условное наименование браузера — " + n)
t = Request.Browser.ActiveXControls
If t Then
    Response.Write("<p>Элементы ActiveX включены")
Else
    Response.Write("<p>Элементы ActiveX выключены")
End If
t = Request.Browser.BackgroundSounds
If t Then
    Response.Write("<p>Звук включен")
Else
    Response.Write("<p>Звук выключен")
End If
t = Request.Browser.Cookies
If t Then
    Response.Write("<p>Cookies сохраняются")
Else
    Response.Write("<p>Cookies не сохраняются")
End If
t = Request.Browser.JavaApplets
If t Then
    Response.Write("<p>Java-апплеты выполняются")
Else
    Response.Write("<p>Java-апплеты не выполняются")
End If
n = Request.Browser.Platform
Response.Write("<p>ОС пользователя — " + n)
i = Request.ContentLength
Response.Write("<p>Длина полученного пакета — " + i.ToString() + " байтов")
n = Request.HttpMethod
Response.Write("<p>Использованный метод передачи данных — " + n)
n = Request.PathInfo
Response.Write("<p>Дополнительные данные в URL- " + n)
n = Request.RawUrl
Response.Write("<p>Полный URL — " + n)
n = Request.UrlReferrer.AbsoluteUri
Response.Write("<p>URL страницы, с которой пришел пользователь — " + n)
n = Request.UserHostAddress
```

```
        Response.Write("<p>IP-адрес пользователя — " + n)
    End Sub

End Class
```

Результат работы этого приложения нельзя показать на иллюстрации, так как созданная Web-страница достаточно велика и не может поместиться полностью на экране, поэтому мы просто воспроизведем отображенный текст. Текстовое содержимое обрабатывающей страницы выглядит следующим образом:

```
Браузер пользователя
Условное наименование браузера — IE
Элементы ActiveX включены
Звук включен
Cookies сохраняются
Java-апплеты выполняются
ОС пользователя — WinNT
Длина полученного пакета — 0 байтов
Использованный метод передачи данных — GET
Дополнительные данные в URL-
Полный URL —
/form/WebForm2.aspx?f1=%D0%98%D0%B3%D0%BE%D1%80%D1%8C&f2=Webmaster@bhv.spb.su
URL страницы, с которой пришел пользователь — http://localhost/form/WebForm1.aspx
IP-адрес пользователя — 127.0.0.1
```

Уже из этого маленького примера становится ясно, сколько информации может предоставить встроенный объект `HttpRequest`. Однако хотелось бы несколько подробнее остановиться на одном его составном свойстве. Свойство `UrlReferrer` предоставляет разработчику практически полную информацию о том ресурсе, с которого удаленный пользователь перешел на Web-страницу. Думаю, излишне говорить о том, как важна для владельца сайта статистика действий удаленных пользователей. Опираясь на нее, можно правильно расставить акценты в содержимом сайта и больше внимания уделять именно наиболее часто посещаемым страницам. Опираться на действия сторонних сервисов по сбору статистики не всегда представляется возможным. Помимо того, что они являются платными, нет гарантии, что сервис действует безошибочно[1], и точно так же нет гарантии, что владелец сервиса не будет пользоваться собранными данными в своих целях. Вне всякого сомнения, случаи ошибок сервисов сбора статистики или использования конфиденциальной информации владельцем сервиса чрезвычайно редки, одна-

[1] Билл Гейтс в свое время заявил: "Я могу гарантировать абсолютную безошибочность кода, состоящего из не более, чем пяти строк."

ко вероятность возникновения этих ситуаций отличается от нуля, поэтому нельзя ее игнорировать.

О том, как получить данные о системе пользователя, мы уже знаем. Составное свойство `Browser` объекта `HttpRequest` позволяет получить эту информацию. Состав этого свойства рассматривался нами ранее. Сейчас пришло время увидеть, из чего состоит свойство `UrlReferrer`.

Еще раз повторюсь, данное свойство описывает адрес того интернет-ресурса, с которого пользователь перешел на текущую Web-страницу. Как известно, значение подобного свойства имеет тип `Uri`. Данный тип является составным. Состав значений данного типа приведен ниже.

- `SchemeDelimiter`. Свойство типа `String`. Предназначено только для чтения. В нем хранится набор символов, которые отделяют наименование протокола в адресе ресурса, от его основного имени. Чаще всего это комбинация двоеточия и двух обратных слэшей.
- `UriShemeFile`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если искомый адрес указывает на некий файл.
- `UriShemeFtp`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если доступ к предыдущему ресурсу осуществлялся по протоколу FTP.
- `UriShemeGopher`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если доступ к предыдущему ресурсу осуществлялся по протоколу Gopher.
- `UriShemeHttp`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если доступ к предыдущему ресурсу осуществлялся по протоколу HTTP. Естественно, этот случай будет наиболее часто встречаться, так как переход на Web-страницы обычно производится с других Web-страниц именно по протоколу HTTP.
- `UriShemeHttps`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если доступ к предыдущему ресурсу осуществлялся по протоколу Secure HTTP, т. е. были приняты меры по обеспечению секретности передачи данных.
- `UriShemeMailto`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если доступ к ресурсу осуществлялся по протоколу SNMP, т. е. ссылка на ресурс активирует почтовый клиент, установленный в системе по умолчанию, для создания электронного письма.
- `UriShemeNews`. Свойство типа `String`, предназначенное только для чтения. В него записывается тип протокола, если искомый ресурс является сообщением новостной конференции, и доступ к нему осуществляется по протоколу NNTP (Network News Transport Protocol).

- `UriSchemeNntp`. Свойство абсолютно идентично предыдущему.
- `AbsolutePath`. Свойство типа `String`. В нем содержится часть URL, расположенная после основного адреса ресурса. Чаще всего в нем находится полное наименование файла, включая и путь к нему в среде виртуальных каталогов сервера.
- `AbsoluteUri`. Свойство типа `String` содержит весь URL искомого ресурса полностью, без каких-либо исключений.
- `Authority`. Свойство содержит доменное имя ресурса или его IP-адрес, в зависимости от того, какой именно тип адреса был использован в URL, а также номер порта, если тот был указан в адресе явно. Все это записывается одним значением типа `String`.
- `Fragment`. Свойство типа `String`. В нем находится наименование закладки документа, являющейся идентификатором для локальных гиперссылок, если она находилась в полном URL. В свойстве записывается не только наименование закладки, но и символ, отделяющий ее от имени документа. Обычно, в этом качестве используется символ решетки — `#`.
- `Host`. В свойстве типа `String` находится только доменное имя ресурса или его IP-адрес, в зависимости от того, какой именно тип адресации был использован в URL.
- `HostNameType`. В свойстве указывается тип имени ресурса, примененный в искомом URL. Свойство имеет тип `UriHostNameType`. Данный тип является перечислимым, следовательно, значение свойства может принимать одно из следующих значений:
  - `Basic`. Адрес ресурса успешно обработан, но идентифицировать его тип не представляется возможным;
  - `Dns`. В URL использовано доменное имя ресурса, удовлетворяющее правилам службы DNS;
  - `IPv4`. В URL использовано наименование узла, соответствующее правилам четвертой версии протокола IP;
  - `IPv6`. В URL использовано наименование узла, соответствующее правилам шестой версии протокола IP;
  - `Unknown`. Тип наименования узла не распознан и не поддерживается системой.
- `IsDefaultPort`. Логическое свойство типа `Boolean`. Имеет значение `True`, если сервер, на котором расположен искомый ресурс, функционирует на порте, номер которого соответствует "умолчальному" значению для данного типа сервера.
- `IsFile`. Логическое свойство типа `Boolean`. Имеет значение `True`, если искомый URL указывает на некий файл.

- `IsLoopBack`. Логическое свойство типа `Boolean`. Имеет значение `True`, если URL указывает на некий ресурс, расположенный на том же сервере, что и обрабатывающее приложение.
- `IsUnc`. Логическое свойство типа `Boolean`. Имеет значение `True`, если URL составлен по правилам UNC (Universal Naming Convention). Чаще всего подобные адреса встречаются в локальных сетях.
- `LocalPath`. Свойство типа `String`. В нем содержится путь к искомому ресурсу в физической операционной системе той платформы, на которой функционирует сервер. Естественно, если пользователь пришел к нам с другого сервера, в данном свойстве не будет адекватной информации, так как наш сервер не может получить данные о файловой системе другого сервера.
- `PathAndQuery`. Свойство типа `String`. В нем содержится полное наименование файла из URL, включая и путь к нему, и дополнительный запрос, отделяемый от имени файла символом вопросительного знака.
- `Port`. Свойство типа `Integer`. В нем хранится номер порта, на котором функционирует сервер, указанный в URL.
- `Query`. Свойство типа `String`. В нем находится строка запроса из URL, включая и стартовый символ вопросительного знака.
- `Scheme`. Свойство типа `String`. В нем находится наименование протокола доступа к ресурсу, на который указывает URL. В качестве значений должно быть использовано одно из следующих слов:
  - `file`. Доступ осуществляется к ресурсу, находящемуся на локальном компьютере;
  - `ftp`. Доступ осуществляется по протоколу FTP;
  - `gopher`. Доступ к ресурсу осуществляется по протоколу Gopher;
  - `http`. Доступ осуществляется по протоколу HTTP;
  - `https`. Доступ осуществляется по протоколу Secure HTTP;
  - `mailto`. URL указывает на адрес электронной почты;
  - `news`. URL указывает на ресурс из новостной конференции, и доступ производится по протоколу NNTP.
- `UserEscaped`. Свойство логического типа `Boolean`. Приобретает значение `True`, если полученная строка, составляющая URL, была ранее пропущена через процедуру URL-кодирования браузером или иным приложением.
- `UserInfo`. Свойство типа `String`. В нем располагается информация об удаленном пользователе, который и инициировал данный сеанс работы с Web-приложением. В этом свойстве могут храниться имя пользователя, его пароль доступа или иные специфичные именно для него данные.

Помимо перечисленных нами свойств, для объектов типа `Uri`, существует и ряд методов. Следует рассмотреть наиболее часто используемые методы.

- `CheckHostName`. В качестве параметра методу передается адрес какого-либо хоста в виде значения типа `String`. Метод проверяет полученный адрес и возвращает тип этого адреса. Возвращаемое значение имеет тип `UriHostNameType`. Естественно, применение этого метода к адресу ресурса, с которого на нашу Web-страницу пришел пользователь, абсолютно идентично использованию свойства `Scheme`, которое мы уже рассматривали, однако не следует забывать, что при помощи типа URI мы можем обрабатывать не только значение свойства `UrlReferrer`, но и вообще любой URL.
- `CheckSchemeName`. Данная функция также применима, пожалуй, только для тех URL, которые пользователь вводит самостоятельно. В качестве своего параметра она принимает значение типа `String`, в котором указывается протокол доступа к какому-либо ресурсу в том виде, как его указал пользователь. Метод проверяет, поддерживается ли запрошенный протокол системой. В случае успешного прохождения теста, возвращается значение `True`, иначе — `False`. Как нетрудно догадаться, метод возвращает значение типа `Boolean`.
- `HexEscape`. В качестве параметра метод принимает значение типа `Char`, т. е. один символ, затем выдает строчную запись кода этого символа в шестнадцатеричной системе счисления. Возвращаемое значение имеет тип `String`. Подобный метод может использоваться для проведения процедуры URL-кодирования.
- `HexUnescape`. Метод, по сути, производит обратное действие по отношению к предыдущему. В качестве параметров ему передаются строка и целое число, т. е. параметры имеют тип `String` и `Integer` соответственно. Целое число — это порядковый номер символа в строке, которая является первым параметром. Метод предполагает, что последовательность символов, начиная с указанного места в строке, является шестнадцатеричным кодом некоего символа (т. е. подразумевается, что искомая строка-параметр уже была подвергнута процедуре URL-кодирования), и возвращает этот символ. В том случае, если на указанном месте находится символ, который не может рассматриваться как шестнадцатеричная цифра, будет возвращен именно символ, находящийся на указанной позиции, а само целочисленное значение, указывающее не его расположение, будет увеличено на единицу. Понятно, что метод возвращает значение типа `Char`.
- `IsHexEncoding`. Метод получает точно такой же набор параметров, как и предыдущий. Но при этом он всего лишь проверяет, может ли символ, находящийся в искомой строке на указанной позиции, быть частью шестнадцатеричного числа. Для этого достаточно, чтобы искомый символ был

либо цифрой, либо буквой латинского алфавита в промежутке от "a" до "f" включительно (регистр не имеет значения). Если символ удовлетворяет поставленным условиям, возвращается значение `True`, иначе — `False`.

Теперь, зная, какие параметры можно извлечь из свойства `UrlReferrer`, можно получить из него максимально полную информацию, и на ее основе построить статистику перемещения пользователя по сайту, а также знать, какой именно сайт обеспечивает Вам наибольший приток пользователей.

Но вернемся к объекту `HttpRequest`. В его состав входит еще одно свойство, которое может представлять для нас интерес. Его наименование — `ServerVariables`. Данное свойство является коллекцией, в которой хранятся все значения, получаемые сервером от браузера пользователя. Часть этих значений вынесена в отдельные свойства объекта `HttpRequest`, но давайте посмотрим, что же именно хранится в этом свойстве-коллекции. Мы поступим так же, как и в предыдущем примере — всего лишь изменим код Web-страницы, обрабатывающей данные, введенные пользователем в форму. На самом деле, нам эти данные не нужны, нас будет интересовать лишь дополнительная информация. Итак, измененный код обрабатывающей Web-страницы приведен в листинге 3.27.

Листинг 3.27

```
Public Class WebForm2
    Inherits System.Web.UI.Page

#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 names(),vals() As String
        Dim i,k As Integer
        names=request.ServerVariables.AllKeys
        For i = 0 To names.GetUpperBound(0)
         Response.Write("Переменная: " + names(i) + "<br> ")
         vals = Request.ServerVariables.GetValues(i)
          For k = 0 To vals.GetUpperBound(0)
           Response.Write("Значение " + CStr(k) + ": " + vals(k) +
"<br>")
          Next k
        Next i
     End Sub

End Class
```

Сам код достаточно прост. Мы объявляем два массива типа `String`. В массиве `names` будут храниться наименования переменных сервера, которые мы получим при помощи метода `AllKeys`. Второй массив `vals` предназначен для значений этих переменных. При этом не следует думать, что первый элемент массива `vals` будет однозначно соответствовать первому элементу массива `names`. Все несколько сложнее.

Мы не знаем заранее, будет ли каждая переменная простой строкой, или же это будет массив строк. Поэтому мы используем метод `GetValues`, который собирает все значения, соответствующие той или иной переменной. На самом деле в нашем примере все переменные будут представлены единичными значениями строчного типа, поэтому подобный код страдает некоторыми излишествами. Однако в тех случаях, когда заранее неизвестно, будут ли все значения обычными строками или же массивами, рекомендуется все-таки использовать именно подобную конструкцию.

В результате действия этого приложения, в окно просмотра браузера будет выведен следующий текст:

```
Переменная: ALL_HTTP

Значение 0: HTTP_CONNECTION:Keep-Alive HTTP_ACCEPT:image/gif, image/
x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel,
application/msword, */* HTTP_ACCEPT_ENCODING:gzip, deflate HTTP_ACCEPT_
LANGUAGE:ru HTTP_HOST:localhost HTTP_REFERER:http://localhost/
form/WebForm1.aspx HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; .NET CLR 1.0.2914)

Переменная: ALL_RAW
```

```
Значение 0: Connection: Keep-Alive Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword,
*/* Accept-Encoding: gzip, deflate Accept-Language: ru Host: localhost
Referer: http://localhost/form/WebForm1.aspx User-Agent: Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.2914) Переменная:
APPL_MD_PATH
Значение 0: /LM/W3SVC/1/Root/form
Переменная: APPL_PHYSICAL_PATH
Значение 0: d:\inetpub\wwwroot\form\
Переменная: AUTH_TYPE
Значение 0:
Переменная: AUTH_USER
Значение 0:
Переменная: AUTH_PASSWORD
Значение 0:
Переменная: LOGON_USER
Значение 0:
Переменная: REMOTE_USER
Значение 0:
Переменная: CERT_COOKIE
Значение 0:
Переменная: CERT_FLAGS
Значение 0:
Переменная: CERT_ISSUER
Значение 0:
Переменная: CERT_KEYSIZE
Значение 0:
Переменная: CERT_SECRETKEYSIZE
Значение 0:
Переменная: CERT_SERIALNUMBER
Значение 0:
Переменная: CERT_SERVER_ISSUER
Значение 0:
Переменная: CERT_SERVER_SUBJECT
Значение 0:
Переменная: CERT_SUBJECT
Значение 0:
Переменная: CONTENT_LENGTH
Значение 0: 0
Переменная: CONTENT_TYPE
```

```
Значение 0:
Переменная: GATEWAY_INTERFACE
Значение 0: CGI/1.1
Переменная: HTTPS
Значение 0: off
Переменная: HTTPS_KEYSIZE
Значение 0:
Переменная: HTTPS_SECRETKEYSIZE
Значение 0:
Переменная: HTTPS_SERVER_ISSUER
Значение 0:
Переменная: HTTPS_SERVER_SUBJECT
Значение 0:
Переменная: INSTANCE_ID
Значение 0: 1
Переменная: INSTANCE_META_PATH
Значение 0: /LM/W3SVC/1
Переменная: LOCAL_ADDR
Значение 0: 127.0.0.1
Переменная: PATH_INFO
Значение 0: /form/WebForm2.aspx
Переменная: PATH_TRANSLATED
Значение 0: d:\inetpub\wwwroot\form\WebForm2.aspx
Переменная: QUERY_STRING
Значение 0: f1=&f2=
Переменная: REMOTE_ADDR
Значение 0: 127.0.0.1
Переменная: REMOTE_HOST
Значение 0: 127.0.0.1
Переменная: REQUEST_METHOD
Значение 0: GET
Переменная: SCRIPT_NAME
Значение 0: /form/WebForm2.aspx
Переменная: SERVER_NAME
Значение 0: localhost
Переменная: SERVER_PORT
Значение 0: 80
Переменная: SERVER_PORT_SECURE
Значение 0: 0
```

```
Переменная: SERVER_PROTOCOL
Значение 0: HTTP/1.1
Переменная: SERVER_SOFTWARE
Значение 0: Microsoft-IIS/5.1
Переменная: URL
Значение 0: /form/WebForm2.aspx
Переменная: HTTP_CONNECTION
Значение 0: Keep-Alive
Переменная: HTTP_ACCEPT
Значение 0: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, _
application/vnd.ms-excel, application/msword, */*
Переменная: HTTP_ACCEPT_ENCODING
Значение 0: gzip, deflate
Переменная: HTTP_ACCEPT_LANGUAGE
Значение 0: ru
Переменная: HTTP_HOST
Значение 0: localhost
Переменная: HTTP_REFERER
Значение 0: http://localhost/form/WebForm1.aspx
Переменная: HTTP_USER_AGENT
Значение 0: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.0.2914)
```

Как видно, практически все серверные переменные дублируются отдельными свойствами объекта `HttpRequest`, поэтому разработчик в каждом случае может сам решать, каким способом ему получать необходимую информацию — через серверные переменные или при помощи отдельных свойств, однако можно заметить, что в некоторых случаях серверные переменные несут несколько более детальную информацию.

Говоря о принципах работы с объектом `HttpRequest`, нельзя обойти стороной его коллекцию `Cookies`. Однако для обсуждения данной технологии у нас выделен отдельный раздел, в котором она будет обсуждаться достаточно детально и комплексно, поэтому здесь мы опустим ее рассмотрение. А пока что перейдем к объекту `HttpResponse`.

Этот объект, как мы уже знаем, позволяет отправлять информацию браузеру пользователя. В наших примерах мы пользовались методом `Write`, который позволяет передавать строки. После того, как браузер получит эти строки, он отобразит их. При этом, естественно, следует осознавать, что браузер будет интерпретировать эти строки как HTML-код. Естественно, при использовании данного метода приходится передавать не только значимую информацию, но и все остальное содержимое Web-страницы, что естественно не слишком удобно. Именно поэтому в наших примерах, приведенных

в данном разделе главы, страницы с выводимой информацией выглядели так непрезентабельно.

Но что делать, если нам необходимо выводить информацию на Web-страницах? Дело в том, что компоненты Web Forms (Web-формы) позволяют нам отображать любую информацию, поэтому следует пользоваться в подобных случаях именно ими. Однако может возникнуть еще одна проблема. В зависимости от действий пользователя или иных дополнительных данных, приложению может понадобиться отображать совершенно различные по своей структуре страницы. Можно, конечно, создать на одной странице несколько наборов элементов Web Forms, и в зависимости от ситуации делать видимым один набор и скрывать другие наборы элементов, но очень уж громоздко получается. Действительно, проще все-таки воспользоваться различными Web-страницами. Но остается проблема, как отобразить необходимую страницу в данный момент. Ведь любые переходы по гиперссылкам и формам приводят на некую конкретную страницу. Хорошим решением может быть использование метода `Redirect`. Он позволяет переадресовать запрос, полученный от браузера удаленного пользователя к любой другой Web-странице, в зависимости от нужд приложения. Нам уже известно, что в качестве параметра данному методу передается строка, в которой записан URL той Web-страницы, которая будет отображена. Таким образом, мы получаем достаточно стройную схему. Рассмотрим это на примере.

Предположим, что мы строим сайт, структура которого будет различна для мужчин и женщин. На стартовой странице при помощи формы мы узнаем пол подключившегося удаленного пользователя, а затем в зависимости от полученных данных отобразим ту или иную Web-страницу. Соответственно, нам потребуется три модуля в нашем новом проекте. Но не будем забегать вперед. Все по порядку.

На первой странице нам потребуется, помимо основных информационных элементов, разместить группу кнопок переключателей. Если бы мы использовали на этой странице только чистый HTML-код, тогда бы нам потребовалось вместо трех страниц создавать четыре. Но, используя компоненты Web Forms (Web-формы), можно обойтись всего тремя. Группа кнопок реализована при помощи компонента `RadioButtonList`. В его состав входит всего два элемента, причем ни один из них не будет выбран по умолчанию. А свойство `AutoPostBack` нам следует установить в `True`, чтобы как только пользователь выбрал какую-либо кнопку-переключатель, эти данные были бы переданы на сервер. Итак, HTML-код стартовой страницы приведен в листинге 3.28.

Листинг 3.28

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="redirect.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" con-
tent="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: 32px; POSITION:
absolute; TOP: 36px" runat="server" Width="290px" Height="19px">
Пожалуйста, укажите свой ↳пол</asp:Label>
   <asp:RadioButtonList id="RadioButtonList1" style="Z-INDEX: 102; LEFT:
41px; POSITION: absolute; TOP: 73px" runat="server" AutoPostBack="True">
    <asp:ListItem Value="male">Мужской</asp:ListItem>
    <asp:ListItem Value="female">Женский</asp:ListItem>
   </asp:RadioButtonList>
  </form>
 </body>
</HTML>
```

Для того чтобы узнать, какую именно кнопку переключатель выбрал пользователь, мы воспользуемся свойством `SelectedIndex`. Соответственно, код для загрузки второй страницы сайта мы поместим в функцию, обрабатывающую изменение состояния элемента, которая носит наименование `RadioButtonList1_SelectedIndexChanged`. Но проще посмотреть сам код класса, реализующего данную Web-страницу. Этот код приведен в листинге 3.29.

Листинг 3.29

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents RadioButtonList1 As
System.Web.UI.WebControls.RadioButtonList

#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 RadioButtonList1_SelectedIndexChanged(ByVal sender
    As System.Object, ByVal e As System.EventArgs) Handles
    RadioButtonList1.SelectedIndexChanged
        If RadioButtonList1.SelectedIndex = 0 Then
            Response.Redirect("WebForm2.aspx")
        Else
            Response.Redirect("WebForm3.aspx")
        End If
    End Sub
End Class
```

Достаточно хорошо видно, что когда свойство `RadioButtonList1.SelectedIndex` равно нулю, т. е. когда пользователь активизировал первую кнопку-переключатель, загружается страница WebForm2.aspx. Иначе, браузеру удаленного пользователя передается страница WebForm3.aspx. Видно, что использование метода `Redirect` является очень простым и эффективным приемом.

О чем еще можно упомянуть, рассматривая объект `HttpResponse`? Скорее всего, следует вспомнить о методе `AddHeader`. Как мы знаем, многие действия над Web-страницами производятся при помощи заголовков HTTP, которые передаются в качестве содержимого тегов `<META>`. Так, например, заголовок `Refresh` заставляет браузер по истечении заданного времени перезагрузить Web-страницу или загрузить другую Web-страницу. Для того

чтобы динамически добавлять подобные HTTP-заголовки в тело Web-страницы, отсылаемой пользователю, отлично подойдет искомый метод `AddHeader`. В качестве параметров ему передается две строки — наименование добавляемого HTTP-заголовка и его значение.

Так, например, для того чтобы вставить все тот же заголовок `Refresh`, следует воспользоваться следующим фрагментом кода:

```
AddHeader ("Refresh", "10")
```

При этом в блок заголовка отправляемой пользователю Web-страницы будет вставлен следующий тег:

```
<META HTTP-EQUIV="Refresh" Content="10">
```

Тогда браузер пользователя получит директиву перезагрузить данную страницу через десять секунд после ее полной загрузки.

Однако следует помнить, что HTTP-заголовок `Refresh` позволяет не просто указывать период перезагрузки, но и адрес Web-страницы, которую должен будет загрузить браузер. По умолчанию браузер будет загружать ту же самую страницу, в которой располагался этот HTTP-заголовок. Но мы можем заставить его по истечении указанного времени загрузить и какую-либо иную Web-страницу. Для этого к HTTP-заголовку `Refresh` добавляется еще одно значение — URL страницы, на которую необходимо будет перейти. Но разработчику эта деталь ничуть не прибавляет работы, он должен передать оба значения одной строкой в качестве параметра метода.

Также объект `HttpResponse` обладает несколькими механизмами, которые позволяют разработчику самостоятельно управлять кэшированием Web-страниц. Рассмотрим ситуацию с кэшированием несколько подробнее. Как известно, некоторые Web-страницы редко изменяются с течением времени. Если браузер загружает подобную Web-страницу, то он может одновременно сохранить ее в своем внутреннем кэше. Когда пользователь еще раз запросит эту страницу, браузер может не загружать ее с Web-сервера, а извлечь из внутреннего кэша, существенно ускоряя подобным образом время ее получения. Но браузеру необходимо точно знать, когда Web-страница будет изменена, чтобы не предоставить пользователю ее устаревшую версию. Срок годности страницы можно указать при помощи все тех же заголовков HTTP, но в состав объекта `HttpResponse` входит несколько механизмов, которые позволяют разработчику управлять кэшированием страниц. Их мы и рассмотрим сейчас.

Свойство `Expires` позволяет задавать отрезок времени, в течение которого Web-страница гарантированно не будет изменена, и браузер может предоставлять пользователю ее ранее сохраненную в кэше версию. В качестве значения данного свойства используется целое число, которое указывает срок жизни Web-страницы в минутах.

Если рассчитать срок жизни страницы без изменений не представляется возможным, но точно известна дата, после которой страница будет изменена,

можно использовать свойство `ExpiresAbsolute`. Его значение имеет тип `DateTime`, и, естественно, браузер будет считать, что после наступления даты, указанной в виде значения этого свойства, обязательно следует запрашивать обновленную версию Web-страницы с сервера.

Но страницы кэшируются не только браузером. В Интернете существует огромное множество прокси-серверов, через которые пользователи и ведут свою работу в WWW. Эти прокси-серверы также обладают собственной политикой кэширования Web-страниц, запрашиваемых пользователями, поэтому следует каким-либо образом также контролировать кэширование своих страниц прокси-серверами. Поэтому в состав объекта `HttpResponse` было введено свойство `CacheControl`. Но с его помощью мы не можем указывать сроки действительности сохраненных в кэше прокси-серверов копий наших Web-страниц. Данное свойство позволяет всего лишь указывать прокси-серверу, можно или нельзя ему кэшировать данную Web-страницу. Если в качестве значения свойства `CacheControl` используется строка `Public`, то Web-страница может быть сохранена в кэше. Если же мы используем значение `Private`, то прокси-сервер будет поставлен в известность, что данная страница не подлежит кэшированию.

А вот теперь самое интересное. Дело в том, что все эти свойства, естественно, можно использовать в создаваемых приложениях ASP.NET, но это не будет хорошим стилем программирования. Они оставлены только для совместимости с предыдущими версиями ASP-скриптов. В ASP.NET существует отдельный объект, который предназначен именно для управления политиками кэширования Web-страниц. О нем и пойдет речь в следующем разделе.

Правила кэширования Web-страниц

В предыдущем разделе мы рассмотрели основные ситуации, связанные с кэшированием Web-страниц. Как и всегда, разработчик должен найти компромисс между скоростью загрузки страниц и их динамическим обновлением. Если содержимое Web-страницы динамически обновляется, то, естественно, ни о каком кэшировании речь идти не может. Но подобный жесткий режим обновления содержимого Web-страниц все же будет создавать достаточно интенсивный трафик между браузером удаленного пользователя и WWW-сервером. Поэтому при разработке архитектуры Web-сайта следует по возможности достаточно четко разделить статические и динамические элементы по разным Web-страницам, чтобы статическое наполнение сайта могло быть кэшировано.

Но вернемся к установке правил кэширования Web-страниц. В ASP.NET в состав объекта `HttpResponse` был введен еще один дополнительный объект с наименованием `Cache`. Он реализуется при помощи класса `HttpCachePolicy`.

И этот объект предоставляет разработчику поистине беспрецедентный объем возможностей. Рассмотрим методы и свойства данного класса.

- `AppendCacheExtension`. Метод устанавливает значение для заголовка HTTP Cache-Control. В качестве параметра методу передается значение типа `String`, которое и будет содержимым искомого заголовка HTTP. Необходимо упомянуть, что полностью все возможности, связанные с данным заголовком HTTP, реализованы только в браузере Internet Explorer последних версий. Другие браузеры просто проигнорируют данный заголовок, если в нем будут встречаться неизвестные параметры. Полностью возможные параметры для этого заголовка HTTP описаны в документе RFC 2616. Однако здесь мы приведем два наиболее часто используемых параметра. Один из них носит наименование `pre-check`. Значением данного параметра является целое число, указывающее продолжительность временного промежутка в секундах. Данный параметр указывает время, в течение которого разработчик или приложение не планируют изменять Web-страницу. Но при этом нельзя сказать точно, что она не будет изменена. Это просто планирование. Второй параметр обычно используется в паре с первым и носит наименование `post-check`. Его значением также является целое число, указывающее продолжительность временного промежутка в секундах. В течение этого промежутка времени браузер по запросу пользователя отображает Web-страницу из локального кэша, но при этом в фоновом режиме проверяет исходную версию Web-страницы, находящуюся на WWW-сервере, таким образом уменьшая риск предоставления неадекватной информации в будущем.
- `SetCacheability`. Метод устанавливает правила кэширования Web-страницы в сети, т. е. область действия этого метода распространяется и на прокси-серверы. В качестве параметра данному методу передается значение типа `HttpCacheability`. Данный тип является перечислимым, и в его состав входят следующие константы:
  - `NoCache`. Значение указывает, что данная Web-страница вообще не подлежит кэшированию, и браузер пользователя всегда должен запрашивать Web-страницу с сервера;
  - `Private`. Значение используется по умолчанию. Указывает, что Web-страница может кэшироваться браузером удаленного пользователя, но прокси-серверы не должны хранить ее копию в своем кэше;
  - `Public`. Значение указывает, что Web-страница может кэшироваться как браузером, так и прокси-серверами;
  - `Server`. Указывает, что кэшировать Web-страницу будет сам WWW-сервер. С точки зрения конечного получателя Web-страницы, нет никакой разницы между вариантами `NoCache` и `Server`, так как и в том, и в другом случае ни браузер, ни прокси-сервер не имеют права кэшировать Web-страницу, однако с точки зрения разработчика раз-

ница все-таки достаточно существенная. В конце концов, именно разработчику придется думать, как осуществлять кэширование Web-страниц на своем сервере.

- `SetExpires`. Метод позволяет устанавливать точную дату и время, до наступления которых Web-страница не будет изменена, и браузер может использовать ее локальную версию, сохраненную в собственном кэше или кэше прокси-сервера. В качестве параметра методу передается значение типа `DateTime`.
- `SetLastModified`. Метод позволяет указывать для Web-страницы дату и время ее последнего изменения. При этом, по сути, устанавливается значение заголовка HTTP с наименованием Last-Modified. В качестве параметра методу передается значение типа `DateTime`.
- `SetLastModifiedFromFileDependencies`. Данный метод, подобно предыдущему, устанавливает значение заголовка HTTP с наименованием Last-Modified. Но при этом разработчик не должен передавать устанавливаемое значение в качестве параметра. Метод сам получает его на основе атрибутов файла.
- `SetNoServerCaching`. Метод запрещает серверу производить кэширование искомой Web-страницы в своем кэше. То есть, при выполнении данного метода все запросы к Web-странице должны выполняться в полном объеме, а заранее сохраненная в кэше WWW-сервера копия страницы пользователю предоставляться не будет.

Помимо рассмотренного нами объекта, существует и еще один способ кэширования страниц ASP.NET. Он основан на использовании директивы `OutputCache`. Для того чтобы установить период годности данной Web-страницы равный одной минуте, мы должны в ее HTML-код включить следующую директиву:

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

Однако необходимо понимать, что существует несколько доводов против использования подобного механизма регулирования кэширования Web-страниц. Во-первых, возможности данной директивы все-таки достаточно ограничены. Встроенный объект `HttpCachePolicy` перекрывает их по всем параметрам. Во-вторых, следует осознавать, что ASP.NET все-таки является стройной системой, которая отходит от традиций языков сценариев, к которым, несомненно, относились предыдущие версии ASP, и приближается к системам класса RAD (Rapid Application Development). Следовательно, необходимо соблюдать основные принципы построения приложений и пользоваться все-таки встроенными классами, а не вставлять куски кода вручную в тело разрабатываемой Web-страницы.

Итак, резюмируем. В данном разделе мы рассмотрели основные возможности кэширования Web-страниц при помощи встроенного объекта `HttpCachePolicy`

и директивы `Output Cache`. Каждый разработчик волен сам выбирать, какой именно вариант ему подходит больше.

Cookies

Cookies. Что же это такое, и с чем их едят? Происхождение этого названия достаточно туманно, и по одной из версий, оно пришло из неких мультсериалов, герои которых, съев волшебное печенье, приобретали возможность обмениваться мыслями. Теперь эти cookie-файлы поедают Web-серверы и системы удаленных пользователей. Правда, обмениваются они не мыслями.

Cookies представляют собой достаточно простой механизм обмена информацией между сервером и браузером. Во время соединения сервер просто создает или открывает уже существующий файл в системе удаленного пользователя, в котором сохраняет некую информацию, которая потом будет применяться для идентификации пользователя и для оптимизации работы с ним. Cookies — это своего рода памятные записки, как маленькие желтые листочки, которыми мы обклеиваем свое рабочее место. Это как читательский билет, который хранит информацию о наших вкусах и пристрастиях, показывает нашу "историю взаимодействия" с данным конкретным сервером.

Еще одной проблемой, связанной с данной технологией, является ее наименование в русском языке. Несколько фривольные гастрономические переводы этого слова (пряники, печеньица), естественно, не могут быть использованы в технической литературе, а прямая калька с английского — куки — еще не прижилась, и звучит достаточно необычно, несмотря на то, что именно этот вариант претендует на роль стандартного обозначения в русском языке. Поэтому мы пока что будем обходиться англоязычным наименованием — cookies. Но вернемся к технологическим проблемам.

Давайте посмотрим, как происходит стандартная HTTP-транзакция. Сначала браузер обращается по URL и входит в контакт с сервером. После этого, сервер пересылает браузеру некую затребованную информацию, или сообщение об ошибке. Затем соединение разрывается, и сервер более ничего не помнит о произошедшей транзакции. Таким образом, последующие сеансы связи никак не скореллированы с результатами предыдущих транзакций. Именно этот перекос в некоторой степени выправляют cookies.

Согласно стандарту, cookie представляет собой обычную строку, не превосходящую по размеру 4000 символов, которая отсылается сервером браузеру. Браузер анализирует полученный (ая/ое) cookie, проверяет длину, дату истечения срока годности (best before), после чего сохраняет в отдельном файле. Важная деталь. Cookies не могут содержать исполняемый или интерпретируемый код, а также его фрагменты.

Cookie содержит обязательные поля, опциональные поля, а также любую другую информацию в текстовом формате, обработку которой берет на себя сервер. Рассмотрим поля, которые могут использоваться в cookies.

Стандартный вид заголовка cookie выглядит следующим образом:

```
Set-Cookie: name=<значение>; expires=<дата>; path=<путь>; domain=
<имя домена>; secure.
```

Разберем эти поля несколько более подробно.

- `name=<значение>`. Это определение имени и содержания cookie. Как известно, один и тот же сайт, а в нашем случае, одно и то же Web-приложение, может сохранять в локальной системе удаленного пользователя несколько cookies. Естественно, их необходимо как-то различать. Идентификация каждого cookie происходит именно по его наименованию. Поэтому поле `name` является обязательным.
- `expires=<дата>`. Это срок годности cookie, а точнее момент истечения срока его годности по гринвичскому меридианному времени. Например, `Monday, 9-August-1999 22:00:00 GMT`.
- `path=<путь>`. В случае задания этого параметра, cookie будет возвращен браузером серверу только в том случае, если будет запрошен URL с этого пути. Например, если задано `path=<books>`, то cookie будет выдан только при затребовании документов, лежащих в этом каталоге, или в его подкаталогах. Если этот параметр не будет указан, то он будет устанавливаться автоматически, причем, в качестве пути будет взят путь первого запрошенного URL. При помощи этого параметра мы можем создавать отдельные cookies для каждой Web-страницы, входящей в состав нашего сайта.
- `domain=<имя домена>`. В этом параметре определяется имя домена, куда будут возвращаться cookies. Для доменов com, edu, net, org, gov, mil, int, biz и info необходимо задавать как минимум два периода имени сервера. Например, amazon.com будет означать любые имена вида *.amazon.com. Для других доменов — не менее трех периодов. По умолчанию это имя домена есть имя сервера, приславшего cookie.
- `Secure`. Если в нашем cookie есть это поле, то он будет возвращаться только на сервер, обеспечивающий сертифицированный метод безопасности, обычно SSL (Secure Socket Level).

Теперь, когда мы знаем, как выглядят cookies, и для чего они нужны, рассмотрим методы работы с ними, применяемые в ASP.NET. Следует отметить, что свойство `Cookies` входит в состав объектов `HttpResponse` и `HttpRequest`. Это свойство имеет тип `HttpCookieCollection` и является, как несложно понять, коллекцией. Состоит эта коллекция из элементов типа `HttpCookie`, и перед тем, как рассматривать детально коллекцию, узнаем, как устроены ее элементы. Уникальных методов у данного типа нет, и это понятно. Любой cookie это всего лишь набор данных, и тип `HttpCookie` предназначен главным образом для получения данных из cookie без синтаксического разбора его строки. В свойствах типа `HttpCookie` содержатся зна-

чения всех полей cookie как стандартных, так и введенных разработчиком. Рассмотрим эти свойства.

- `Domain`. Свойство содержит значение типа `String`, в котором записывается доменное имя сайта, с которого устанавливается данный cookie.
- `Expires`. Свойство содержит значение типа `DateTime`, которое устанавливает срок действия cookie.
- `HasKeys`. Свойство логического типа `Boolean`. В том случае, если cookie помимо стандартных полей содержит дополнительную информацию, используется значение `True`. Значение этого свойства нельзя устанавливать программно, оно доступно лишь для чтения.
- `Name`. Свойство типа `String`. Его значением является наименование cookie.
- `Path`. Свойство содержит значение типа `String`, в котором указывается путь к некоему виртуальному каталогу. Как мы уже знаем, если явно установлено значение данного свойства, cookie будет выдаваться системой удаленного пользователя только тем Web-страницам, которые располагаются в указанном каталоге или в его подкаталогах. Можно сказать, что это свойство несколько уточняет область действия свойства `Domain`.
- `Secure`. Свойство логического типа `Boolean`. Значение `True` указывает, что cookie отправляется на систему удаленного пользователя или получается из нее только по защищенному протоколу, такому, как, например, HTTPS. По умолчанию используется значение `False`.
- `Value`. Значение типа `String` является содержимым самого cookie. Данное свойство доступно только для чтения.
- `Values`. Свойство является коллекцией типа `NameValueCollection` и содержит значения всех параметров искомого cookie. Естественно, если мы используем данное свойство, мы предполагаем, что в одном cookie использовано несколько параметров, и тогда свойство `HasKeys` должно иметь значение `True`. Как и предыдущее свойство, эта коллекция может лишь предоставлять приложению данные, но программно их изменить нельзя.

Теперь, когда мы знаем, как именно устроены элементы коллекции `HttpCookieCollection`, мы можем перейти к рассмотрению ее структуры.

Свойства у типа этой коллекции вполне стандартны и присущи в равной мере всем типам-коллекциям, а набор методов несколько специфичен. Именно при помощи этих методов приложение получает возможность оперировать с cookies, содержимое которых задано при помощи типа `HttpCookie`.

- `Add`. Метод добавляет в коллекцию еще один cookie. В качестве параметра передается значение типа `HttpCookie`. Естественно, желательно задать содержимое этого cookie заранее, до выполнения метода `Add`.

- `Clear`. Метод очищает содержимое всей коллекции. При помощи этого метода мы можем уничтожить все cookies, входящие в искомую коллекцию.
- `CopyTo`. Метод позволяет копировать в некий массив содержание всех cookies, входящих в коллекцию. Содержимое cookies, естественно, копируется в массив в виде строк, т. е. используется тип `String`. В качестве параметров методу передается сам массив, в который будет происходить копирование, и целое число, являющееся номером элемента массива, с которого начнется копирование информации.
- `GetKey`. Метод возвращает наименование cookies, входящего в коллекцию, в виде строкового значения типа `String`. В качестве значения методу передается целое число, которое является порядковым номером cookies в коллекции. Естественно, метод вернет наименование именно того cookies, чей порядковый номер был передан в качестве параметра.
- `Remove`. Метод позволяет удалить из коллекции тот или иной cookie. В качестве параметра методу передается наименование того cookie, который подлежит удалению из коллекции. Естественно, передаваемый параметр имеет тип `String`.
- `Set`. Метод позволяет изменить содержимое какого-либо cookie. В качестве параметра методу передается значение типа `HttpCookie`, в котором уже были произведены необходимые изменения.

Теперь, когда мы знаем, какие средства работы с cookies есть в нашем распоряжении, рассмотрим примеры использования этих механизмов. Начнем мы с установки cookies в систему удаленного пользователя. При входе на сайт предложим ему указать свое имя. Для того чтобы в дальнейшем сайт мог "узнать" и поприветствовать этого пользователя при его новых заходах, мы можем сохранить в cookies имя, указанное пользователем, и дату его последнего визита на сайт. Естественно, для того чтобы записать информацию в локальную систему пользователя, нам придется воспользоваться объектом `HttpResponse` и его свойством `Cookies`. Естественно, в состав объекта `HttpResponse` не входит какой-либо метод, который явно записывает cookies в системе удаленного пользователя. Нам достаточно всего лишь создать элементы коллекции `Cookies`, а они будут переданы браузеру пользователя без каких-либо дополнительных директив, так как объект `HttpResponse` как раз и создан для передачи информации удаленному пользователю. Стоит также отметить, что все cookies будут передаваться браузеру при помощи заголовков HTTP с наименованием `Set-Cookie`.

Но перейдем все-таки к рассмотрению листингов. Как всегда, попробуем создать код, опираясь на наши предыдущие наработки. Предположим, что у нас есть маленькая форма, в которой мы запрашиваем у пользователя его имя. Нам потребуются одно поле текстового ввода и кнопка подтверждения. В листинге 3.30 приведен HTML-код этой Web-страницы в том виде, как он показан в среде разработки Visual Studio .NET.

Листинг 3.30

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="cookies.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" con-
tent="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: 40px; POSITION:
absolute; TOP: 26px" runat="server">Укажите Ваше имя</asp:Label>
   <asp:TextBox id="TextBox1" style="Z-INDEX: 102; LEFT: 43px; POSITION:
absolute; TOP: 51px" runat="server" Width="222px"
Height="24px"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 103; LEFT: 50px; POSITION:
absolute; TOP: 88px" runat="server" Text="Подтвердить"></asp:Button>
  </form>
 </body>
</HTML>
```

Естественно, основная работа будет производиться сервером при нажатии пользователем на кнопку. Но для того чтобы увидеть, какие именно действия будет производить приложение, необходимо рассмотреть код класса, реализующего данную Web-страницу. Он приведен в листинге 3.31.

Листинг 3.31

```
Public Class WebForm1
    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 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 MyCookie As New HttpCookie("Cookie1")
        Session.Add("NAME", TextBox1.Text)
        MyCookie.Value = TextBox1.Text
        MyCookie.Values("Time") = DateTime.Now().ToString()
        Response.Cookies.Add(MyCookie)
        Response.Redirect("WebForm2.aspx")
    End Sub
End Class
```

Рассмотрим тщательно всю последовательность операций, которые выполняются в обработчике нажатия на кнопку. Прежде всего, мы объявляем переменную `MyCookie` типа `HttpCookie`, которая обладает наименованием `Cookie1`. После объявления этой переменной мы добавляем в коллекцию объекта `Session` элемент с наименованием `NAME`, в который мы заложим текст, введенный пользователем в текстовое поле. Потом этот элемент будет воспринят второй Web-страницей и обработан ею. Механизм работы второй Web-страницы для нас сейчас абсолютно неинтересен, поэтому сосредоточимся на рассмотрении листинга 3.31. В самом начале работы процедуры мы создали переменную типа `HttpCookie`. Основное значение данного

cookie будет содержать имя пользователя, которое тот указал в поле текстового ввода. Для занесения этого имени в переменную мы используем свойство `Value`.

Но было бы неплохо еще запомнить дату и время его последнего посещения нашего сайта. Поэтому при помощи коллекции `Values` мы создаем в cookie еще одно поле с наименованием `Time`, в которое мы записываем текущее время. После того, как cookie полностью создан, мы добавляем его в коллекцию при помощи оператора `Response.Cookies.Add(MyCookie)`. И последним действием процедуры мы переадресуем пользователя на другую Web-страницу, с наименованием WebForm2.aspx. Именно в этот момент, вместе с содержимым этой Web-страницы браузер пользователя получает инструкцию, записать в локальную систему содержимое cookie.

Но записать cookie — это только половина дела. Необходимо еще получить их при последующем посещении сайта тем же пользователем и правильно его обработать. В рамках уже существующего проекта можно создать третью Web-страницу, которая при обращении пользователя к ней будет получать cookie и обрабатывать его.

На этой странице мы можем расположить всего один компонент `Label` и формировать его текстовое содержимое динамически, опираясь на информацию, заложенную в cookie. Класс, реализующий эту страницу, приведен в листинге 3.32.

Листинг 3.32

```
Public Class WebForm3
    Inherits System.Web.UI.Page
    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
        Dim s, t As String
        Dim MyCookie As HttpCookie
        MyCookie = Request.Cookies.Item("Cookie1")
        s = MyCookie.Values(0)
        t = MyCookie.Values(1)
        Label1.Text = "Здравствуйте, " + s + ". Ваш последний визит на
наш сайт был " + t
    End Sub

End Class
```

Из данного листинга видно, что основные действия производятся в обработчике `Page_Load`, который инициируется при загрузке Web-страницы. Заранее мы объявляем переменную `MyCookie` типа `HttpCookie`. По умолчанию, при загрузке страницы в объект `Request` заранее помещаются все cookies, связанные с данным сайтом. Так как мы точно знаем наименование интересующего нас cookie, мы можем обратиться к нему при помощи этого имени. Также разработчику заранее известна структура данного cookie. Мы точно знаем, что сначала в нем записано имя пользователя, а потом дата и время его последнего посещения сайта. Соответственно, эти данные мы читаем в строковых переменных при помощи коллекции `Values`, обращаясь к этим значениям по их порядковым номерам.

В тот момент, когда писалась эта часть главы, европейский парламент принял решение, ограничивающее использование сайтами технологии cookies. Свое решение парламентарии мотивировали тем, что использование cookies позволяет собирать информацию о действиях человека в Интернете без его ведома. Поэтому все сайты, использующие в своей работе cookies, обязаны предупреждать пользователя, что информация о нем будет сохранена в cookies.

Что можно сказать об этом решении? Во-первых, следует помнить, что пользователь всегда может настроить используемый браузер таким образом, чтобы ограничить или запретить запись cookies в своей системе, и дополнительное напоминание может только раздражать пользователей. Во-вторых, следует упомянуть, что только cookies позволяют идентифицировать пользователя на сайте до того, как он зарегистрируется или укажет свой логин,

заведенный им ранее. Поэтому отключение cookies снимет всю индивидуализацию сайта. Все настройки и предпочтения каждого пользователя, хранящиеся именно в cookies, будут утрачены. Таким образом, все достижения по персонализации сайтов пропадут. Ну, а в-третьих, следует еще раз упомянуть тот факт, что в Интернете нет географических границ, и всегда можно найти себе хостинг вне пределов юрисдикции европейского парламента.

Стоит ли пользователю бояться, что данные о его работе с сайтом будут собираться, храниться и анализироваться? Если я регулярно захожу на сайт туристического агентства и просматриваю информацию о турах в Париж, сайт может "понять", что я интересуюсь именно этой информацией, и в следующий раз, когда я зайду на сайт, предоставит мне информацию о новых турах в Париж. Таким образом, я получаю информацию, интересную именно для меня. Ничего плохого для пользователя в таком методе работы нет. Единственный довод против использования cookies — возможность хранения в них конфиденциальной информации. Вот этого точно не стоит делать. При помощи некоторых ухищрений и совпадении определенных условий, есть возможность получить cookies не тому сайту или Web-приложению, которые эти cookies сохраняли. Таким образом, если злоумышленник получает доступ к cookies пользователя, он может прочитать информацию, хранящуюся в них. Вероятность возникновения подобной ситуации достаточно низка, но она существует, поэтому никогда не следует хранить в cookies конфиденциальную информацию или пароли. В cookies может храниться лишь идентификатор пользователя и дополнительные данные о его предыдущих посещениях. Тогда, даже в том случае, если cookie пользователя будут получены злоумышленником, он не сможет ими воспользоваться в своих целях.[1]

Резюмируя все вышесказанное, следует отметить, что технология cookies при соблюдении некоторых вполне здравых условий приносит разработчику сайта очень ощутимые выгоды и добавляет возможности пользователю. Поэтому все-таки без cookies в хорошем сайте не обойтись.

Стилевое оформление Web-страниц

Одним из важных условий создания действительно хорошего сайта является оформление всех Web-страниц, входящих в его состав, в едином стиле. На первый взгляд, подобная задача является тривиальной. Действительно, что может быть проще — установить одинаковые значения для всех визуальных свойств объектов. Однако при большом объеме Web-страниц, входящих в проект, установка всех значений может занять достаточно большое время. Хотелось бы автоматизировать подобную деятельность. И здесь на помощь

[1] На момент написания книги Европарламент не принял решение об обязательном предупреждении пользователя об использующихся cookies.

приходит технология каскадных таблиц стилей — CSS (Cascading Style Sheets).

В рамках этой технологии создается отдельный внешний файл, в котором собраны правила отображения того или иного тега в браузере. Затем CSS-файл подключался к каждой Web-странице. Таким образом, мы можем легко достигнуть единообразного стилевого оформления каждой Web-страницы, входящей в состав сайта.

Первая версия стандарта CSS содержала правила отображения текста Web-страниц, размещения текстового содержимого на странице и некоторые детали отображения страниц и ячеек таблиц. Вторая версия добавила несколько новых возможностей, таких как визуальные эффекты или автоматическая нумерация элементов.

Стилевая таблица представляет собой обычный текстовый файл специализированного формата. В спецификациях данный формат имеет обозначение `text/css`. В CSS-файле записываются правила отображения для некоторых тегов. Пример записи одного такого правила может выглядеть приблизительно следующим образом:

```
a {color : navy; font-family : Arial}
```

Это правило указывает на то, что текстовое содержимое всех тегов `<a>`, которым в HTML обозначаются гиперссылки, будет отображаться при помощи любого доступного шрифта из семейства Arial, темно-синего цвета (navy).

Все правила CSS состоят из двух частей. В первой части записывается наименование тега, к содержимому которого и будет применено данное правило оформления. Это объявление в спецификации CSS называют `selector`. Иногда это выражение в виде кальки переносят в русскоязычную техническую литературу. Поэтому при описании технологии CSS можно увидеть, что наименования тегов в стилевых таблицах называют селекторами.

После наименования тега в фигурных скобках записывается правило оформления текстового содержимого. Посмотрим еще раз на наш пример.

```
a {color : navy; font-family : Arial}
```

Как видно, в фигурных скобках может записываться сразу несколько правил, разделенных символом точки с запятой. Само правило состоит из двух частей. Сначала записывается наименование параметра отображения, например, вид применяемого шрифта, а затем, после двоеточия, мы пишем значение этого параметра. В примере все это достаточно хорошо видно.

Итак, как мы видели, в фигурных скобках мы можем записать сразу несколько правил отображения содержимого. Но иногда случается и так, что сразу для нескольких тегов задаются одинаковые правила отображения. Мы можем не указывать одинаковые правила для каждого тега отдельно, а объединить теги в одну группу и задать правила отображения для всех те-

гов сразу. При этом объединяемые наименования тегов разделяются запятой. Это можно увидеть на примере следующей конструкции:

```
a, h1 { color : navy }
```

Данное правило указывает, что текстовое содержимое гиперссылок и заголовков первого уровня будет отображаться темно-синим цветом.

В правила синтаксиса CSS входят еще правила оформления комментариев. Комментарии в CSS создаются в стиле языка C. Начинается комментарий с символа наклонной черты и звездочки (`/*`), а завершается обратным сочетанием (`*/`). Все это легко увидеть в следующем примере:

```
/*Это комментарий*/
```

Мы знаем, что полное наименование CSS — каскадные таблицы стилей. Почему они называются таблицами стилей, мы уже поняли. Возникает вполне обоснованный вопрос, при чем тут каскадность, и что это вообще обозначает?

В компьютерной индустрии очень велик темп развития. Следствием этого является множество самых различных тенденций и течений. Часть из них вскоре отмирает, а часть переходит из разряда тенденций в ранг идеологии. Одной из таких тенденций является повторное использование кода и наследование ранее созданных решений. К CSS это имеет самое прямое отношение.

Мы не обязаны для каждого документа использовать только одну стилевую таблицу. Мы можем подключать их сразу несколько. Например, при добавлении новых документов к сайту придется добавлять новые правила отображения, создавать новые стили. Иногда это делается коррекцией уже существующей стилевой таблицы, но никто не запрещает нам создать дополнительный файл и подключить его к документу. Как мы только что говорили, мы можем подключать сразу несколько таблиц.

Иногда может возникнуть ситуация, когда в нескольких CSS-файлах содержатся правила отображения для одного и того же тега. И правила эти могут не совпадать. В этом случае браузером используется самое последнее объявление правила отображения.

Рассмотрим эту концепцию на примере. Предположим, у нас есть два файла стилевых таблиц. В файле style1.css мы размещаем объявления правил отображения для двух элементов HTML-документа с наименованиями `list` и `item`. Получаем следующий код:

```
list {font-family: Times New Roman}
item {color: blue; font-family: Arial}
```

А в дополнительном файле style2.css мы объявляем правила отображения для элементов `item` и `part` следующим образом:

```
item {color: green; font-family: Arial}
part {color: black}
```

Теперь, если мы подключим к HTML-документу эти два стилевых файла, окажется, что для элемента `item` задано два различных правила отображения. В первом файле style1.css указано, что текст элемента `item` будет отображаться синим цветом, а во втором файле правило указывает, что применяться для отображения содержимого этого же элемента будет уже зеленый цвет. То, какой цвет будет использоваться на самом деле, зависит от того, в каком порядке к XML-документу будут подключаться файлы стилевых таблиц.

Используется всегда последнее правило. А именно, если сначала мы подключим файл style1.css, а потом файл style2.css, то текстовое содержимое элемента `item` будет отображаться зеленым цветом.

Но мы можем устанавливать специальный модификатор для правил отображения, который будет заставлять браузер игнорировать порядок подключения стилевых таблиц. Так, если мы приписываем к какому-либо правилу оформления модификатор `important`, то для отображения текстового содержимого элемента будет использоваться именно это правило, вне зависимости от того, были ли еще подключены правила для отображения этого же элемента. Иначе говоря, использование ключевого слова `important` позволяет установить для какого-либо правила привилегированный уровень и убрать его из иерархии каскадных таблиц стилей. Рассмотрим пример. Возьмем знакомый нам файл style1.css и немного модифицируем его код. Получаем следующую совокупность правил отображения:

```
list {font-family: Times New Roman}
item {color: blue ! important; font-family: Arial}
```

Теперь, при том порядке подключения стилевых файлов style1.css и style2.css, который мы рассматривали ранее, текстовое наполнение элемента `item` будет отображаться синим цветом.

Из примера виден порядок использования модификатора `important`. Если нам необходимо какое-либо правило сделать привилегированным, мы после него ставим символ восклицательного знака и записываем ключевое слово `important`, отделенное пробелом.

Впрочем, вполне возможна ситуация, когда в нескольких CSS-файлах существуют правила отображения для одного и того же элемента, и сразу несколько из них имеют модификатор `important`. В этом случае формируется дополнительная иерархия из этих привилегированных правил и используется для отображения самое последнее подключенное правило.

Естественно, для того чтобы успешно применять стилевые таблицы, необходимо знать наименования параметров изображения. Мы не будем сейчас детально описывать все эти параметры, полное описание CSS займет слишком много места, поэтому тем, кто еще не знаком со стилевыми таблицами, придется посоветовать найти литературу, в которой стилевые таблицы рассмотрены детально. Однако в приложении приведен список наиболее часто используемых параметров отображения.

На самом деле, разработчик приложений ASP.NET может устанавливать любые параметры отображения содержимого Web-страниц без стилевых файлов, напрямую используя свойства тех или иных элементов. Но тогда придется писать слишком уж большой объем кода. Подобное решение никак нельзя назвать изящным. Попробуем воспользоваться стилевыми таблицами.

Существует две возможности использования технологии CSS. Мы можем либо установить значение атрибута `style` для тех тегов HTML-страниц, которые будут подвергаться стилевой обработке, либо подключить внешний файл, содержащий определение стилевой таблицы. Увы, ASP.NET не позволяет указывать подключаемую к Web-странице стилевую таблицу как свойство страницы. Поэтому мы создадим стилевую таблицу, в которой правила отображения будут ассоциированы не с наименованиями тегов, а с наименованием классов-селекторов. Для каждого визуального компонента Web Forms предусмотрено свойство `CssClass`, в котором и указывается наименование класса-селектора.

Итак, для начала мы создадим CSS-файл. Для включения в состав проекта внешнего стилевого файла следует выполнить команду меню **Project | Add New Item** (Проект | Добавить новый элемент). После выполнения этой команды на экране будет отображено диалоговое окно **Add New Item** (Добавить новый элемент). В группе **Templates** (Шаблоны) следует выбрать элемент **Style Sheet** (Стиль). После этого в состав проекта будет включен дополнительный файл.

По умолчанию в создаваемую стилевую таблицу включается заготовок для определения стиля, предназначенного для тега `<body>`, т. е. в этом правиле мы можем задавать правила отображения, которые будут влиять на весь документ в целом.

Для того чтобы добавить новое правило оформления в стилевую таблицу, следует нажать на кнопку **Add Style Rule** (Добавить правило оформления), находящуюся на инструментальной панели **Style Sheet** (Стиль). При этом будет активировано диалоговое окно **Add Style Rule** (Добавить правило оформления), чей внешний вид показан на рис. 3.21.

Так как мы планируем создавать правила оформления содержимого Web-страницы, опираясь на классы-селекторы, стоит выбрать кнопку-переключатель **Class Name** (Имя класса). В связанное с ней поле текстового ввода следует ввести наименование класса-селектора, для которого мы будем создавать правило отображения. После того, как наименование класса будет введено, нажатие на кнопку **OK** перенесет в код стилевой таблицы заготовку для создания правил оформления искомого класса.

После того, как мы создадим все заготовки для наших классов, можно будет создать для них правила оформления. Для этого следует установить курсор на том селекторе, на который мы и будем накладывать условия оформления, и нажать кнопку **Build Style** (Создать стиль). Она активирует диалоговое

окно интерактивного построителя стилей **Style Builder** (Построитель стилей), внешний вид которого показан на рис. 3.22.

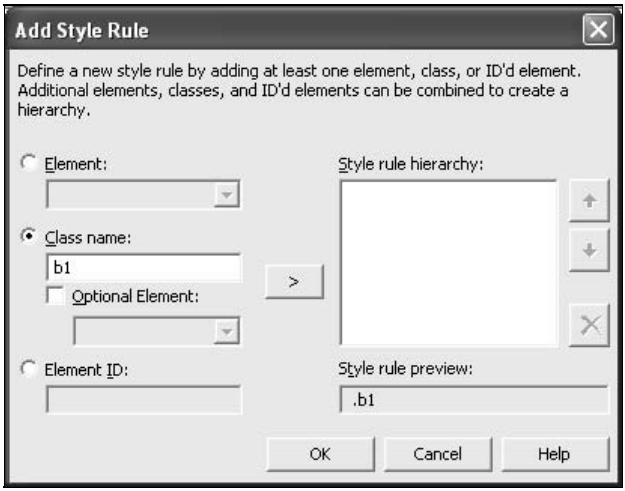


Рис. 3.21. Внешний вид диалогового окна **Add Style Rule**

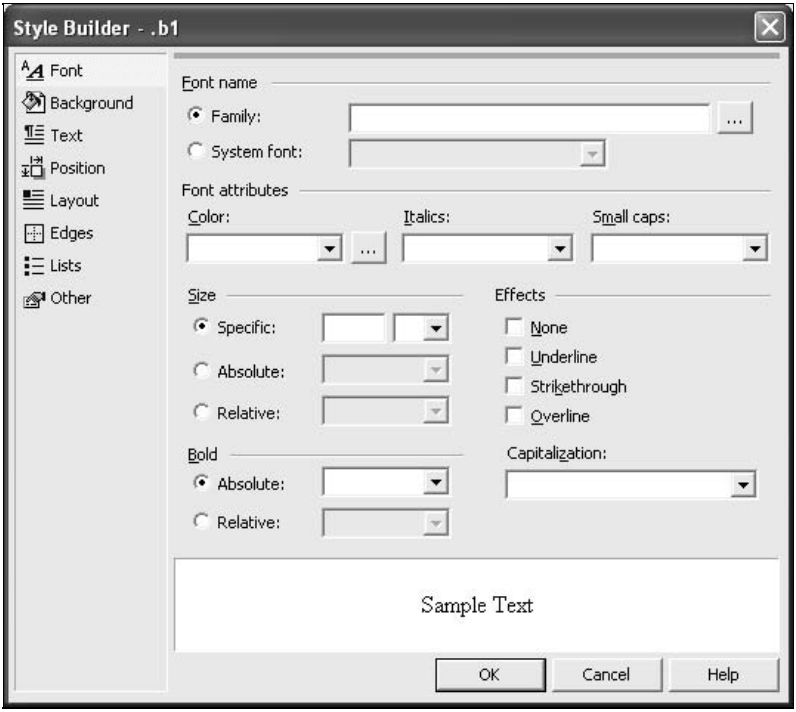


Рис. 3.22. Диалоговое окно **Style Builder**

При помощи диалогового окна построителя стилей можно задать правила отображения для каждого класса-селектора в визуальном режиме. Однако никто не запрещает разработчику просто написать код правил CSS вручную. Все зависит от личных предпочтений. Но следует отметить, что при использовании диалогового окна построителя стилей всегда можно видеть, как будет выглядеть текст с учетом установленных правил отображений, так как в диалоговом окне присутствует панель, на которой отображается пример текста в том виде, как он будет выглядеть в итоговом варианте на Web-странице.

Вернемся к нашему примеру. Код созданного CSS-файла приведен в листинге 3.33.

Листинг 3.33

```
.b1
{
  background-color: deepskyblue;
}
.p1
{
  color: black;
  font-style: italic;
  font-family: 'Arial Black';
  text-decoration: underline;
}
```

После того, как CSS-файл был создан, следует подключить его к нашей Web-странице. Для этого следует перейти на разрабатываемую Web-страницу, к которой мы собираемся подключить стилевую таблицу, и выполнить команду меню **Format | Document Styles** (Формат | Стили документа). Данная команда активизирует диалоговое окно **Document Styles** (Стили документа). В этом окне следует нажать кнопку **Add Style Link** (Добавить ссылку на стилевую таблицу), которая позволит привязать к Web-странице существующий стилевой файл. Для этого будет использовано диалоговое окно **Select Style Sheet** (Выбор стилевой таблицы), чей внешний вид показан на рис. 3.23.

Все, теперь стилевой файл привязан к данной Web-странице. При этом в ее HTML-коде, в разделе заголовка документа появилась строка следующего вида:

```
<LINK href="StyleSheet1.css" type="text/css" rel="stylesheet">
```

Теперь достаточно лишь подключить созданную CSS-таблицу ко всем Web-страницам проекта, установить значения свойств `CssClass` для необходимых компонентов, и задача стилевого оформления сайта решена.

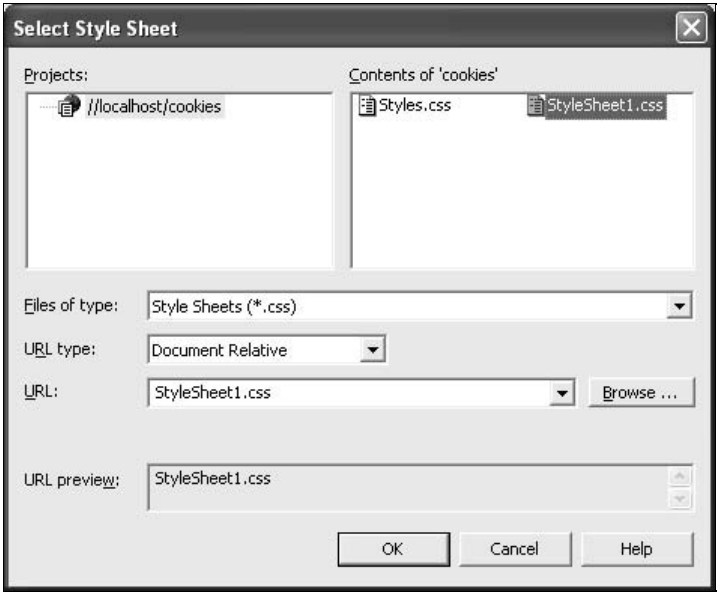


Рис. 3.23. Диалоговое окно **Select Style Sheet**

Впрочем, использование классов-селекторов позволяет выборочно применять классы к элементам оформления Web-страниц. Если необходимо, чтобы правила отображения применялись ко всем элементам оформления Web-страниц, правильнее будет создавать эти правила, опираясь не на классы-селекторы, а на наименования тегов.

Сеансы работы пользователей

Ранее мы уже упоминали об объекте `HttpSessionState`, но в этом разделе мы внимательно рассмотрим механизм поддержки сеансов работы пользователей, который снимает с разработчика множество ненужной работы. Как мы уже знаем, подобный класс позволяет идентифицировать пользователя и отслеживать его работу с сайтом в текущий момент. 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 его подтверждения 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 при переходах между ними, вся накопленная информация должна сохраняться. Для обеспечения этих возможностей можно было бы использовать технологию cookies или взаимодействие с базами данных. Так, в начале работы пользователя

с сайтом можно было бы 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 их нельзя использовать. Более того, нужно помнить о том, что пользователи имеют возможность отключить запись cookies на свои машины.

Ограничения на объем хранимой информации могут быть сняты путем использования баз данных. Тем более что без применения баз данных весьма трудно реализовать онлайновый магазин с его стандартным набором возможностей. Можно, но действительно трудно. Однако использование баз данных для сопровождения сеансов работы пользователей потребует хранить не только информацию о товарах, продающихся в магазине, но и дополнительную информацию обо всех действиях пользователей. Это может занять дополнительное время в работе сайта. Конечно, при современных мощностях Web-серверов обработка двух-трех дополнительных таблиц не займет много времени, однако следует осознавать, что в подобную таблицу будет записываться информация не только о совершенных 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 я веду. 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 применять его методы и свойства. В нескольких примерах этой книги мы уже использовали объект `HttpSessionState` в качестве хранилища данных, которые должны быть переданы от одной Web-страницы к другой, поэтому в этом разделе мы не будем еще раз рассматривать подробно эту процедуру. Нам следует научиться правильно управлять сессиями.

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 сохранится, 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 `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 мы предполагаем, что пользователь будет оформлять покупку на сайте не более трех часов (а в моей практике был случай, когда пользователь собирал достаточно объемный заказ около трех часов), то значение этого свойства должно равняться ста восьмидесяти. Принудительно завершить сеанс работы можно при помощи метода `Abandon`. Однако перейдем от описания объекта к примерам работы с ним. Попробуем сделать очень маленький прототип онлайнового магазина.

Мы не будем использовать базы данных, речь о них пойдет в следующей главе. На первой странице мы просто перечислим список товаров, продающихся в нашем магазине. Каждому наименованию товара мы поставим в соответствие независимый переключатель. Для того чтобы положить товар в свою корзину заказа, пользователю достаточно отметить флажок в соответствующем независимом переключателе. Вторая Web-страница будет отображать текущее состояние корзины заказа с перечнем выбранных товаров и общей суммой покупки. А когда пользователь завершит заказ, будет отображена третья Web-страница, с благодарностью за сделанный заказ. Но она будет нужна не только для того чтобы поблагодарить пользователя, сколько для того чтобы завершить сеанс работы. А теперь перейдем к детальному рассмотрению этого приложения.

На первой странице мы расположим обычную таблицу вкладки компонентов HTML. Дело в том, что компонент `Table` вкладки **Web Forms** (Web-формы) рассчитан на размещение главным образом текста, а нам необходимо разместить в таблице еще и независимые переключатели. Поэтому мы создаем таблицу из шести строк, и в каждой строке будет по три ячейки. Верхняя строка предназначена для наименований столбцов, а в остальных пяти строках мы разместим информацию о товарах.

Левый столбец предназначен для независимых переключателей, во втором столбце будет находиться наименование товара, а в третьем — его цена. А под таблицей мы поместим кнопку, которая будет переадресовывать пользователя к его корзине заказа. HTML-код созданной нами страницы в том виде, как он создан средой разработки Visual Studio .NET, приведен в листинге 3.34.

Листинг 3.34

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb" Inherits="trans.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Обработка сеансов</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:button id="Button1" style="Z-INDEX: 101; LEFT: 52px;
    POSITION: absolute; TOP: 298px" runat="server" Width="292px"
    Height="24px" Text="Посмотреть корзину заказа"></asp:button>
   <TABLE style="Z-INDEX: 102; LEFT: 37px; WIDTH: 540px;
    POSITION: absolute; TOP: 25px; HEIGHT: 214px" cellSpacing="1"
    cellPadding="1" width="540" border="1"
    xmlns="http://schemas.microsoft.com/intellisense/ie5">
    <TR>
     <TD style="WIDTH: 81px">
      Заказать
     </TD>
     <TD style="WIDTH: 266px">
      Наименование товара
     </TD>
     <TD>
       Цена
      </TD>
```

```
</TR>
<TR>
 <TD style="WIDTH: 81px">
  <asp:checkbox id="CheckBox1" runat="server"></asp:checkbox>
 </TD>
 <TD style="WIDTH: 266px">
  Цифровой фотоаппарат
 </TD>
 <TD>
  850
 </TD>
</TR>
<TR>
 <TD style="WIDTH: 81px">
  <asp:checkbox id="CheckBox2" runat="server"></asp:checkbox>
 </TD>
 <TD style="WIDTH: 266px">
  Ноутбук
 </TD>
 <TD>
  2200
 </TD>
</TR>
<TR>
 <TD style="WIDTH: 81px">
  <asp:checkbox id="CheckBox3" runat="server"></asp:checkbox>
 </TD>
 <TD style="WIDTH: 266px">
  Сканер
 </TD>
 <TD>
  120
 </TD>
</TR>
<TR>
 <TD style="WIDTH: 81px">
  <asp:checkbox id="CheckBox4" runat="server"></asp:checkbox>
 </TD>
 <TD style="WIDTH: 266px">
```

```
      Монитор
     </TD>
     <TD>
      250
     </TD>
    </TR>
    <TR>
     <TD style="WIDTH: 81px">
      <asp:checkbox id="CheckBox5" runat="server"></asp:checkbox>
     </TD>
     <TD style="WIDTH: 266px">
      CD-Writer
     </TD>
     <TD>
      40
     </TD>
    </TR>
   </TABLE>
  </form>
 </body>
</HTML>
```

После того, как пользователь выберет необходимые товары, он нажмет на кнопку, размещенную под таблицей. Следовательно, нам необходимо выделить отмеченные пользователем переключатели, и на основе полученной информации сформировать массив, в котором будут помечены выбранные товары. А затем этот массив передать следующей Web-странице в составе объекта `HttpSessionState`. Весь код, реализующий эти действия, приведен в листинге 3.35.

Листинг 3.35

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents CheckBox1 As System.Web.UI.WebControls.CheckBox
    Protected WithEvents CheckBox2 As System.Web.UI.WebControls.CheckBox
    Protected WithEvents CheckBox3 As System.Web.UI.WebControls.CheckBox
    Protected WithEvents CheckBox4 As System.Web.UI.WebControls.CheckBox
    Protected WithEvents CheckBox5 As System.Web.UI.WebControls.CheckBox
    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 tov(5) As Integer
        Dim i As Integer
        For i = 0 To 4
            tov(i) = 0
        Next
        Session.RemoveAll()
        Session.Timeout = 60
        If CheckBox1.Checked Then
            tov(0) = 1
        End If
        If CheckBox2.Checked Then
            tov(1) = 1
        End If
        If CheckBox3.Checked Then
            tov(2) = 1
        End If
```

```
        If CheckBox4.Checked Then
            tov(3) = 1
        End If
        If CheckBox5.Checked Then
            tov(4) = 1
        End If
        Session.Add("goods", tov)
        Response.Redirect("WebForm2.aspx")
    End Sub
End Class
```

Теперь рассмотрим этот код. При нажатии пользователем на кнопку, сначала создается целочисленный массив из пяти элементов. Если пользователь выбрал какой-либо товар, соответствующему элементу массива будет придано единичное значение. Но перед этим очищается вся коллекция объекта `Session`, и устанавливается время жизни этого объекта, равное одному часу.

Затем приложение анализирует переключатели, и в зависимости от значения их свойства, `Checked` изменяет соответствующие элементы массива.

После того, как массив будет сформирован, он передается в коллекцию объекта `Session`, а затем пользователь переадресовывается на следующую Web-страницу с наименованием "WebForm2.aspx".

Вторая Web-страница отображает состояние корзины заказа. На ней нам потребуется отображать список выбранных пользователем товаров и общую сумму заказа. Так как нам заранее неизвестно, сколько именно товаров выберет пользователь, мы не можем использовать стандартную таблицу, как на первой Web-странице. Можно было бы использовать компонент `Table` с вкладки **Web Forms** (Web формы), который поддается динамическому изменению основных его свойств, но о табличных компонентах мы узнаем в следующем разделе. А пока воспользуемся компонентом `Repeater`.

Этот компонент используется в тех случаях, когда разработчику необходимо разместить на Web-странице несколько повторяющихся блоков с одинаковой структурой. При этом структура повторяющихся блоков может быть достаточно сложной, что позволяет практически все возможности HTML, органы ввода информации, формы и т. д.

Единственное неудобство использования компонента `Repeater` состоит в том, что настраивать его структуру придется вручную, изменяя HTML-код страницы. Но о том, как придется изменить этот код, проще будет рассказывать после того, как будет приведен листинг. А пока перейдем к дальнейшему оформлению Web-страницы.

Нам понадобится еще один компонент `Label`, в котором мы будем указывать сумму заказа пользователя, и кнопка, при нажатии на которую пользо-

ватель будет подтверждать покупку. В общем виде HTML-код этой страницы на стадии разработки приведен в листинге 3.36.

Листинг 3.36

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm2.aspx.vb" Inherits="trans.WebForm2"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Корзина заказа</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" con-
tent="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
   <asp:Repeater id="Repeater1" runat="server">
    <ItemTemplate>
     <p>
      <%# Container.DataItem %>
     <p>
    </ItemTemplate>
   </asp:Repeater>
   <asp:Label id="Label1" style="Z-INDEX: 101; LEFT: 31px;
    POSITION: absolute; TOP: 160px" runat="server">
    Сумма заказа</asp:Label>
   <asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 23px;
    POSITION: absolute; TOP: 203px" runat="server"
     Text="Подтвердить заказ" Width="141px" Height="24px"></asp:Button>
  </form>
 </body>
</HTML>
```

Сам код, приведенный в этом листинге, достаточно прост, но необходимо пояснить механизм использования компонента `Repeater`. При переносе его на разрабатываемую Web-страницу, в ее HTML-код вставляется пара тегов — `<asp:Repeater>` и его закрывающая пара `</asp:Repeater>`. Между ними разработчик должен сам описать структуру отображаемых данных.

Компонент Repeater, как мы уже говорили, позволяет отображать на Web-странице последовательность повторяющихся блоков информации. При этом заранее неизвестно, сколько блоков будет входить в состав этого компонента. Так как мы заранее не знаем, сколько товаров выберет покупатель, этот элемент отлично подходит для наших целей.

Образец форматирования элементов, входящих в состав компонента Repeater, размещается между тегами <ItemTemplate> и </ItemTemplate>. В нашем случае мы будем отображать текстовые строки как отдельные абзацы. Поэтому в качестве оформления мы использовали объявление абзаца при помощи тега и его закрывающей пары . А между этими тегами размещается еще один служебный тег — <%# Container.DataItem %>. Именно он является обозначением тех данных, которые будут помещены в компонент Repeater. Конечно, на этом программирование данного объекта не заканчивается, так как нам необходимо еще передать данные в этот объект. Но эта процедура описывается в самом классе, который реализует данную страницу. Его код приведен в листинге 3.37.

Листинг 3.37

```
Public Class WebForm2
    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 Repeater1 As System.Web.UI.WebControls.Repeater


#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 values As New ArrayList()
    Dim tov(5) As Integer
    Dim s As Integer
    s = 0
    tov = Session.Item("goods")
    If tov(0) = 1 Then
        values.Add("Цифровой фотоаппарат")
        s = s + 850
    End If
    If tov(1) = 1 Then
        values.Add("Ноутбук")
        s = s + 2200
    End If
    If tov(2) = 1 Then
        values.Add("Сканер")
        s = s + 120
    End If
    If tov(3) = 1 Then
        values.Add("Монитор")
        s = s + 250
    End If
    If tov(4) = 1 Then
        values.Add("CD-Writer ")
        s = s + 40
    End If
    Repeater1.DataSource = values
    Repeater1.DataBind()
    Label1.Text = "Сумма заказа " + Str(s)
End Sub

Private Sub Repeater1_ItemCommand(ByVal source As System.Object,
ByVal e As System.Web.UI.WebControls.RepeaterCommandEventArgs)
Handles Repeater1.ItemCommand

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
```

```
        Response.Redirect("WebForm3.aspx")
    End Sub
End Class
```

Связывание объекта `Repeater` с данными должно происходить при загрузке страницы, чтобы она могла быть отображена сразу со всеми данными. Поэтому мы использовали обработчик события `Page_Load`. В качестве источника данных для объекта `Repeater` мы приготовили экземпляр объекта `ArrayList` с наименованием `values`. Но перед тем как связать его с искомым объектом `Repeater`, этот массив необходимо заполнить. Для этого мы используем уже знакомый нам по первой Web-странице этого проекта массив `tov`, в котором единицами отмечены элементы, соответствующие товарам, выбранным пользователем. Мы объявляем этот массив в процедуре, но заполняем его, опираясь на его копию, сохраненную в объекте `Session`. Для этого мы используем следующий оператор:

```
tov = Session.Item("goods")
```

А затем, последовательно проверяя значения элементов этого массива, добавляем наименования выбранных пользователем товаров в массив `values` и подсчитываем общую сумму покупки.

После того, как все элементы массива проверены, остается сформировать внешний вид Web-страницы. Сначала мы займемся объектом `Repeater`. Все данные для него находятся в массиве `values`, следовательно, этот массив и будет служить источником данных. Для того чтобы связать эти два объекта, мы используем следующий оператор:

```
Repeater1.DataSource = values
```

А для того чтобы данные были отображены, следует вызвать метод `DataBind`. И все, компонент `Repeater` правильно обработан, и его данные адекватно отображены. После этой операции отображение суммы покупки при помощи компонента `Label` является действительно тривиальной задачей.

Итак, состояние корзины заказа правильно отображено на Web-странице. Что дальше? Пользователь для того чтобы подтвердить заказ, нажимает на кнопку, которая размещена в нижней части Web-страницы. Обработчик нажатия на эту кнопку переадресовывает посетителя сайта к третьей и последней Web-странице, входящей в состав нашего проекта.

На заключительной Web-странице достаточно поместить всего один компонент `Label`, который будет информировать покупателя о том, что его заказ принят. Таким образом, HTML-код страницы получается очень простым. Он приведен в листинге 3.38.

Листинг 3.38

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm3.aspx.vb" Inherits="trans.WebForm3"%>
```

```
<!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:Label id="Label1" style="Z-INDEX: 101; LEFT: 40px; POSITION:
absolute; TOP: 32px" runat="server" Width="324px" Height="19px">
Ваш заказ принят</asp:Label>
  </form>
 </body>
</HTML>
```

Но эта Web-страница нужна не только для того, чтобы отобразить некий текст. Вместе с окончанием процесса покупки необходимо закончить сеанс работы пользователя и уничтожить используемый экземпляр объекта `Session`. Эти действия выполняются при загрузке страницы, что хорошо видно из листинга 3.39.

Листинг 3.39

```
Public Class WebForm3
    Inherits System.Web.UI.Page
    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
        Session.Abandon()
    End Sub

End Class
```

Легко заметить, что для уничтожения сеанса мы используем метод `Abandon`, о котором уже говорилось в этом разделе.

А теперь посмотрим, в каком виде это приложение показывается пользователю. Настоящий HTML-код первой Web-страницы приведен в листинге 3.40.

Листинг 3.40

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Обработка сеансов</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 name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value=
"dDwtNzgwNjMyOTAzOztsPENoZWNrQm94MTtDaGVja0JveDI7Q2hlY2tCb3gzO0NoZWNrQm94
NDtDaGVja0JveDU7Pj4=" />


   <input type="submit" name="Button1" value="Посмотреть корзину заказа"
id="Button1" style="height:24px;width:292px;Z-INDEX: 101; LEFT: 52px;
POSITION: absolute; TOP: 298px" />
```

```
   <TABLE style="Z-INDEX: 102; LEFT: 37px; WIDTH: 540px; POSITION:
absolute; TOP: 25px; HEIGHT: 214px" cellSpacing="1" cellPadding="1"
width="540" border="1" xmlns="http://schemas.microsoft.com/
intellisense/ie5">
    <TR>
     <TD style="WIDTH: 81px">
      Заказать
     </TD>
     <TD style="WIDTH: 266px">
      Наименование товара
     </TD>
     <TD>
      Цена
     </TD>
    </TR>
    <TR>
     <TD style="WIDTH: 81px">
      <input id="CheckBox1" type="checkbox" name="CheckBox1" />
     </TD>
     <TD style="WIDTH: 266px">
      Цифровой фотоаппарат
     </TD>
     <TD>
      850
     </TD>
    </TR>
    <TR>
     <TD style="WIDTH: 81px">
      <input id="CheckBox2" type="checkbox" name="CheckBox2" />
     </TD>
     <TD style="WIDTH: 266px">
      Ноутбук
     </TD>
     <TD>
      2200
     </TD>
    </TR>
    <TR>
     <TD style="WIDTH: 81px">
```

```
      <input id="CheckBox3" type="checkbox" name="CheckBox3" />
     </TD>
     <TD style="WIDTH: 266px">
      Сканер
     </TD>
     <TD>
      120
     </TD>
    </TR>
     <TR>
     <TD style="WIDTH: 81px">
      <input id="CheckBox4" type="checkbox" name="CheckBox4" />
     </TD>
     <TD style="WIDTH: 266px">
      Монитор
     </TD>
     <TD>
      250
     </TD>
    </TR>
    <TR>
     <TD style="WIDTH: 81px">
      <input id="CheckBox5" type="checkbox" name="CheckBox5" />
     </TD>
     <TD style="WIDTH: 266px">
      CD-Writer
     </TD>
     <TD>
      40
     </TD>
    </TR>
   </TABLE>
  </form>
 </body>
</HTML>
```

Внешний вид этой Web-страницы показан на рис. 3.24.

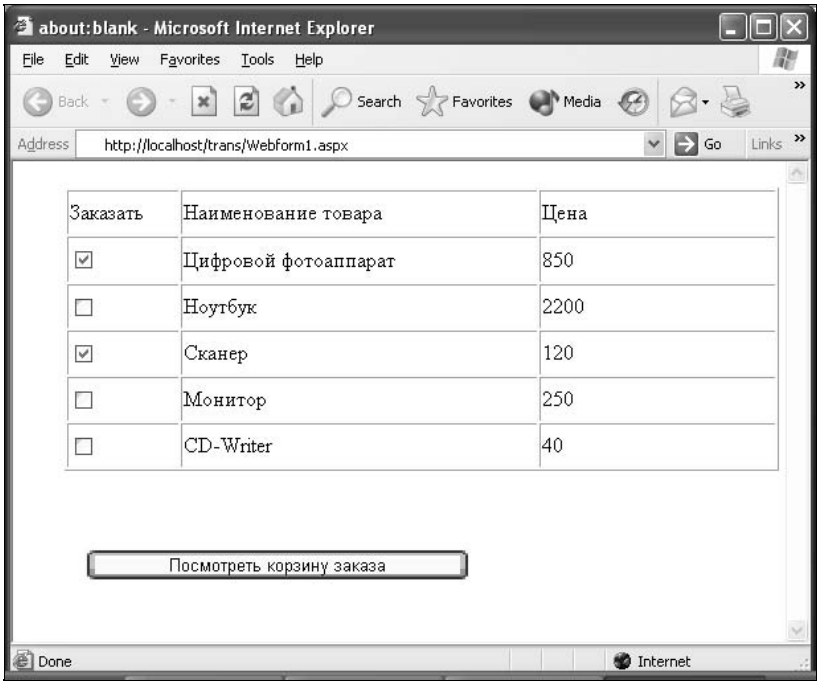


Рис. 3.24. Внешний вид Web-страницы, код которой приведен в листинге 3.40

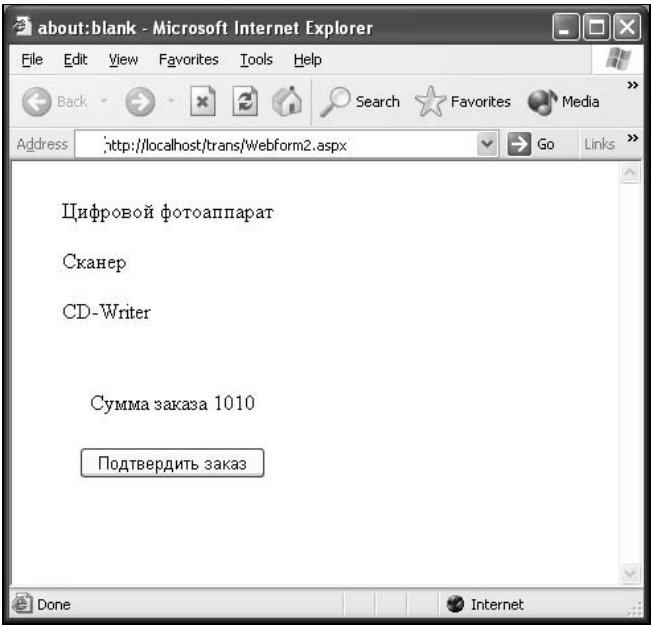


Рис. 3.25. Внешний вид Web-страницы, код которой приведен в листинге 3.41

После того, как пользователь при помощи независимых переключателей выберет необходимые товары и нажмет на кнопку **Посмотреть корзину заказа**, в браузер будет послана вторая Web-страница, код которой приведен в листинге 3.41.

Листинг 3.41

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
  <title>Корзина заказа</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="WebForm2.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value=
"dDwtMTEwODkwMjMwOTt0PDtsPGk8MT47PjtsPHQ8O2w8aTwxPjtpPDM+Oz47bDx0PHA8bDxf
IUl0ZWlDb3VudDs+O2w8aTwzPjs+PjtsPGk8MD47aTwxPjtpPDI+Oz47bDx0PDtsPGk8MD47P
jtsPHQ8QDzQptC40YTRgNC+0LLQvtC5INGE0L7RgtC+0LDQv9C/0LDRgNCw0YI7Pjs7Pjs+Pj
t0PDtsPGk8MD47PjtsPHQ8QDzQodC60LDQvdC10YA7Pjs7Pjs+Pjt0PDtsPGk8MD47PjtsPHQ
8QDxDRC1Xcml0ZXIgOz47Oz47Pj47Pj47dDxwPHA8bDxUZXh0Oz47bDzQodGD0LzQvNCwINC3
0LDQutCw0LfQsCAgMTAxMDs+Pjs+Ozs+Oz4+Oz4+Oz4=" />

     <p>
      Цифровой фотоаппарат
     <p>

     <p>
      Сканер
     <p>

     <p>
      CD-Writer
     <p>

   <span id="Label1" style="Z-INDEX: 101; LEFT: 31px; POSITION: absolute;
TOP: 160px">Сумма заказа  1010</span>
```

```
   <input type="submit" name="Button1" value="Подтвердить заказ"
id="Button1" style="height:24px;width:141px;Z-INDEX: 102; LEFT: 23px;
POSITION: absolute; TOP: 203px" />
  </form>
 </body>
</HTML>
```

Внешний вид этой Web-страницы показан на рис. 3.25.

На этой Web-странице расположена кнопка **Подтвердить заказ**, нажав на которую пользователь завершает покупку. Для этого, как мы знаем, создается третья и последняя Web-страница нашего проекта. Она достаточно проста, но все же приведем для полноты картины и ее HTML-код. Он показан в листинге 3.42.

Листинг 3.42

```
<!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" con-
tent="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form name="Form1" method="post" action="WebForm3.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value="dDwtMTU3ODAzNTQ4MDs7Pg==" />


   <span id="Label1" style="height:19px;width:324px;Z-INDEX: 101; LEFT:
40px; POSITION: absolute; TOP: 32px">Ваш заказ принят</span>
  </form>
 </body>
</HTML>
```

И на этом мы можем закончить рассмотрение механизмов поддержки сеансов работы пользователей, которые используются в ASP.NET.

Табличные компоненты

Теперь рассмотрим использование таблиц в оформлении Web-страниц. В предыдущем разделе мы видели, как используется стандартная таблица, создаваемая компонентом с вкладки **HTML**. Всем хорош этот компонент, кроме одного — у разработчика нет возможности изменять его параметры. Точнее, при помощи некоторых достаточно запутанных ухищрений можно динамически менять содержимое ячеек и управлять их видимостью, используя для этого различные элементы стилевого оформления и свойства видимости, но слишком уж это неблагодарная работа. ASP.NET дал разработчику именно свободу творчества, устранив от необходимости использовать различные ухищрения. Если вся работа в ASP.NET оставляет ощущение стройности, логичности и прозрачности, то почему использование таблиц должно создавать подобные проблемы?

Поэтому в данном разделе мы рассмотрим применение компонента `Table` с вкладки **Web Forms** (Web формы). Начнем мы как обычно с рассмотрения его структуры. Конечно, нет нужды приводить здесь полный список всех свойств или методов, так как немалая их часть унаследована от классов-предков, и они не так уж важны для работы с этим компонентом. Но действительно важные свойства и события, входящие в состав объекта `Table`, мы упомянем. Начнем, как всегда, со свойств.

- `BackColor`. Свойство задает цвет фона для создаваемой таблицы.
- `BackImageUrl`. Свойство типа `String`, в котором указывается URL графического файла. Этот файл должен содержать изображение, которое будет служить фоном создаваемой таблице.
- `BorderColor`. Свойство устанавливает цвет границ ячеек и всей таблицы в целом.
- `BorderStyle`. Свойство имеет значение одноименного перечислимого типа `BorderStyle`. Оно устанавливает тип отображаемых границ ячеек и всей таблицы.
- `BorderWidth`. Свойство задает ширину границ таблицы.
- `CellPadding`. Свойство устанавливает размер отступа между содержимым ячеек и их границами в пикселах.
- `CellSpacing`. Свойство типа `Integer`. Устанавливает дистанцию между ячейками таблицы.
- `GridLines`. Свойство имеет значение одноименного перечислимого типа `GridLines`. Оно указывает, какие именно границы будут отображаться в таблице. Используются следующие значения:
  - `Both`. Отображаются вертикальные и горизонтальные границы;
  - `Horizontal`. Отображаются только горизонтальные линии, разделяющие строки таблицы;

- None. Не отображаются ни горизонтальные, ни вертикальные границы;
- Vertical. Отображаются только вертикальные линии, разделяющие столбцы таблицы.

- HorizontalAlign. Свойство имеет значение одноименного перечислимого типа HorizontalAlign. Оно устанавливает, как будет выравниваться таблица по горизонтали на Web-странице. Используются следующие значения:
  - Center. Таблица будет отображена в центре Web-страницы;
  - Justify. Браузер попытается растянуть таблицу по ширине своего окна просмотра;
  - Left. Таблица будет прижата к левому краю Web-страницы;
  - NotSet. Значение используется по умолчанию. Означает, что никакого горизонтального выравнивания явно для таблицы не задано;
  - Right. Таблица будет прижата к правому краю Web-страницы.
- Rows. Это свойство является коллекцией, в которой содержатся все строки таблицы. Свойство имеет тип TableRowCollection. Коллекция состоит из элементов типа TableRow, который мы рассмотрим несколько позже.
- Width. Свойство задает ширину создаваемой таблицы.

На этом список рассматриваемых свойств объекта Table заканчивается. Теперь упомянем о нескольких событиях, входящих в состав этого объекта.

- DataBinding. Событие инициализируется, когда таблица связывается с ее источником данных. В таблицу разработчик может помещать информацию как в режиме проектирования, так и программно. В программном режиме он может либо самостоятельно указывать данные для каждой ячейки, либо — набор данных, которые и будут отображены в таблице. Вот в тот момент, когда будет установлена связь таблицы с источником данных, и возникнет искомое событие.
- Init. Событие возникает, когда таблица только инициализируется. Это самый первый шаг в отображении таблицы. Сначала создается таблица, как объект, затем она помещается на Web-страницу, в объект Page, и только после этого отображается в окне просмотра браузера. Событие Init возникает на самом первом этапе этой последовательности.
- Load. Событие возникает, когда таблица передается на Web-страницу.
- PreRender. Событие возникает в тот момент, когда сама таблица уже передана Web-странице, но еще не отображена в окне просмотра браузера.

Теперь, как и было оговорено выше, перейдем к рассмотрению структуры объекта TableRow, который реализует отдельную строку таблицы. Мы уже знаем, что сама таблица по своей структуре является именно коллекцией

строк, которые в свою очередь являются коллекциями ячеек. Подобная структура описания таблицы явно унаследована из обычного HTML, поэтому при создании таблицы разработчику будет удобно использовать привычные представления. Но перейдем все-таки к структуре `TableRow`. Как мы уже знаем, это просто коллекция ячеек, которая унаследовала многие свойства от своих предков. Поэтому мы рассмотрим только те свойства, которые регулируют внешний вид строки или описывают ее структуру.

- `BackColor`. Свойство устанавливает цвет фона всех ячеек строки.
- `BorderColor`. Свойство задает цвет границ ячеек, входящих в состав строки.
- `BorderWidth`. Свойство устанавливает ширину границ ячеек строки.
- `Cells`. Свойство имеет тип `TableCellCollection` и является коллекцией объектов `TableCell`, которые реализуют отдельные ячейки таблицы, входящие в состав строки. Этот тип тоже следует пристально рассмотреть, но к нему мы перейдем чуть позже.
- `Font`. Свойство задает шрифт, которым будет отображаться содержимое ячеек таблицы, входящих в состав данной строки.
- `ForeColor`. Свойство устанавливает цвет, которым будет отображаться содержимое ячеек. В нашем случае, это будет цвет текстового содержимого ячеек.
- `Height`. Свойство задает высоту строки таблицы.
- `HorizontalAlign`. Свойство устанавливает горизонтальное выравнивание для содержимого каждой ячейки, входящей в данную строку. По умолчанию используется значение `NotSet`.
- `VerticalAlign`. Свойство устанавливает вертикальное выравнивание для содержимого каждой ячейки, входящей в данную строку. По умолчанию используется значение `NotSet`. Помимо него также используются значения `Top`, `Middle` и `Bottom`.
- `Width`. Свойство задает ширину строки. Однако его явное использование чаще всего будет не нужно, так как все строки наследуют общую ширину таблицы, и нет нужды устанавливать ширину для каждой строки отдельно.

Итак, мы узнали, что основным строительным материалом для таблицы является ячейка, которая реализуется при помощи объекта `TableCell`. Пришла очередь рассмотреть и его.

- `BackColor`. Свойство устанавливает цвет фона данной ячейки. Если для ячейки не задано значение этого свойства, используется цвет фона, который был установлен для всей строки, в состав которой входит эта ячейка.
- `BorderColor`. Свойство задает цвет границ ячейки.
- `BorderWidth`. Свойство устанавливает ширину границ ячейки.

- `ColumnSpan`. Свойство имеет целочисленное значение, которое указывает, сколько столбцов объединено в этой ячейке. По умолчанию, каждая ячейка занимает пространство в одном столбце, но мы можем и объединять несколько соседних столбцов в одной ячейке, как и с обычными HTML-таблицами.
- `Font`. Свойство задает шрифт, которым будет отображаться текстовое содержимое этой ячейки.
- `ForeColor`. В свойстве указывается цвет, которым будет отображаться содержимое ячейки.
- `Height`. Свойство задает высоту ячейки.
- `HorizontalAlign`. Как и одноименное свойство всей строки, это свойство устанавливает горизонтальное выравнивание содержимого ячейки.
- `RowSpan`. Целочисленное значение этого свойства указывает, сколько соседних строк объединяет ячейка.
- `Text`. Свойство имеет тип `String`. Строка, содержащаяся в этом свойстве, отображается в качестве содержимого ячейки.
- `VerticalAlign`. Свойство устанавливает вертикальное выравнивание для текстового содержимого ячейки.
- `Width`. Свойство задает ширину ячейки.
- `Wrap`. Свойство логического типа `Boolean`. В том случае, если содержимое ячейки не умещается в ней как одна строка, следует либо разбить это содержимое на несколько строк без изменения ширины ячейки, либо раздвинуть саму ячейку, увеличив ее размер по горизонтали. В том случае, если свойство имеет значение `True`, используемое по умолчанию, содержимое ячейки будет разбито на несколько строк.

Итак, теперь, когда мы несколько больше знаем о структуре объекта `Table`, можно перейти к рассмотрению примера его использования. Естественно, таблицы идеально подходят для визуализации баз данных, но эту их ипостась мы не будем сейчас рассматривать, так как о взаимодействии ASP.NET с базами данных будет рассказано в следующей главе. Однако следует отметить, что в списке компонентов Web Forms уже есть компонент `DataGrid`, который и предназначен для отображения таблиц, связанных с базами данных. А сейчас нас интересует обычный компонент `Table`, в ячейках которого мы можем размещать текстовую информацию.

На основном пространстве Web-страницы разместим один компонент `Table`. По умолчанию он имеет всего одну ячейку. Попробуем создать два столбца, в каждом из которых будет две ячейки. Для этого нам, естественно, потребуется воспользоваться свойствами таблицы.

В окне свойств объекта **Properties** (Свойства) следует обратить внимание на строку со свойством **Rows** (Строки), и нажать кнопку в этой строке. При

этом будет отображено диалоговое окно **TableRow Collection Editor** (Редактор коллекции строк таблицы), чей внешний вид показан на рис. 3.26.

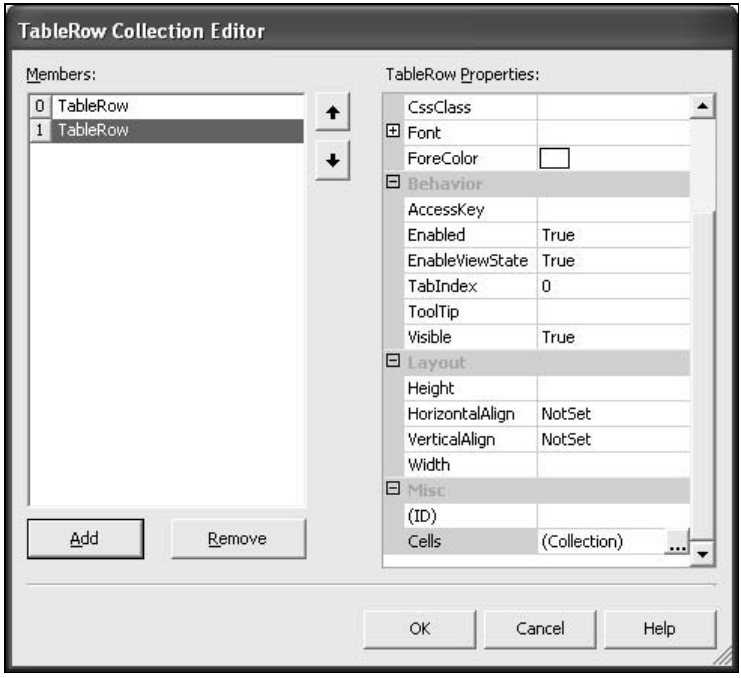


Рис. 3.26. Диалоговое окно **TableRow Collection Editor**

Изначально это окно выглядит несколько иначе, чем на рисунке. Так как в таблице нет ни одной строки, то и в окне ничего не будет отображено. Но как только мы нажмем кнопку **Add** (Добавить), в списке **Members** (Элементы) появится элемент, обозначающий одну строку в таблице. Так как нам необходимо получить изначально две строки, то и кнопку **Add** (Добавить) нам придется нажать два раза.

После того, как мы создадим две строки, следует для каждой из них задать количество ячеек в строке и содержимое этих ячеек. Здесь стоит обратить внимание на список **TableRow Properties** (Свойства строк таблицы), находящийся в правой части диалогового окна **TableRow CollectionEditor** (Редактор коллекции строк таблицы). В этом списке отображаются все свойства выбранной строки таблицы из списка **Members** (Элементы). Нас, естественно, будет интересовать их свойство-коллекция **Cells**, которое располагается на самой нижней строке. Нажатие на кнопку, располагающуюся в этой строке, активирует диалоговое окно **TableCell Collection Editor** (Редактор табличных ячеек), которое позволяет разработчику задавать количество ячеек в строке и все их основные свойства. Внешний вид этого диалогового окна показан на рис. 3.27.

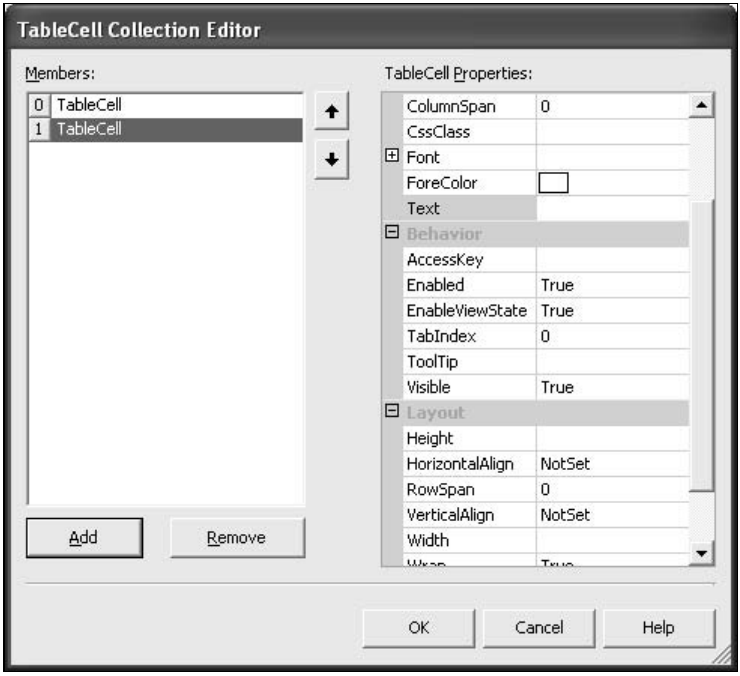


Рис. 3.27. Диалоговое окно **TableCell Collection Editor**

При помощи этого диалогового окна в каждой строке мы создадим по две ячейки, и внесем в них числа от единицы до четырех. Конечно, можно было бы их оставить пустыми, но дело в том, что ячейки без содержимого не будут отображаться, и, следовательно, структура таблицы может от этого нарушиться. Заполнение ячеек будет происходить динамически в момент работы Web-приложения. Для этого нам потребуется разместить на странице три поля текстового ввода, в двух из которых пользователь будет указывать номер строки и столбца, в которых находится редактируемая ячейка, а в третье поле он будет вносить текст, который затем будет отображен в выбранной ячейке. Также, естественно, потребуется одна кнопка. Собственно, весь исходный HTML-код дает достаточно хорошее представление о структуре создаваемой Web-страницы. Он приведен в листинге 3.43.

Листинг 3.43

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind= "WebForm1.aspx.vb" Inherits="Table.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:Table id="Table1" style="Z-INDEX: 101; LEFT: 33px; POSITION:
absolute; TOP: 26px" runat="server" Width="273px" Height="157px"
BorderWidth="1px" BorderStyle="Solid">
    <asp:TableRow BorderWidth="1px">
     <asp:TableCell BorderWidth="1px" Text="1"></asp:TableCell>
     <asp:TableCell BorderWidth="1px" Text="2"></asp:TableCell>
    </asp:TableRow>
    <asp:TableRow>
     <asp:TableCell BorderWidth="1px" Text="3"></asp:TableCell>
     <asp:TableCell BorderWidth="1px" Text="4"></asp:TableCell>
    </asp:TableRow>
   </asp:Table>
   <asp:TextBox id="TextBox2" style="Z-INDEX: 105; LEFT: 180px; POSITION:
absolute; TOP: 240px" runat="server" Height="24px" Width="99px">
</asp:TextBox>
   <asp:Label id="Label1" style="Z-INDEX: 102; LEFT: 39px; POSITION:
absolute; TOP: 207px" runat="server">Номер столбца</asp:Label>
   <asp:Label id="Label2" style="Z-INDEX: 103; LEFT: 180px; POSITION:
absolute; TOP: 209px" runat="server">Номер строки</asp:Label>
   <asp:TextBox id="TextBox1" style="Z-INDEX: 104; LEFT: 40px; POSITION:
absolute; TOP: 239px" runat="server" Height="24px"
Width="99px"></asp:TextBox>
   <asp:Label id="Label3" style="Z-INDEX: 106; LEFT: 45px; POSITION:
absolute; TOP: 280px" runat="server" Height="19px" Width="231px">
Содержимое ячейки</asp:Label>
   <asp:TextBox id="TextBox3" style="Z-INDEX: 107; LEFT: 42px; POSITION:
absolute; TOP: 309px" runat="server"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 108; LEFT: 54px; POSITION:
absolute; TOP: 352px" runat="server" Text="Подтвердить"></asp:Button>
  </form>
 </body>
</HTML>
```

После того, как пользователь укажет координаты изменяемой ячейки и введет ее новое текстовое содержимое, необходимо будет нажать на кнопку для передачи данных на сервер. Следовательно, при разработке класса, реализующего эту Web-страницу, следует вносить код именно в обработчик ее нажатия. Сам код искомого класса приведен в листинге 3.44.

Листинг 3.44

```
Public Class WebForm1
    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 TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Label3 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Table1 As System.Web.UI.WebControls.Table

#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 i, j As Integer
        Dim t As String
        i = Val(TextBox1.Text) - 1
        j = Val(TextBox2.Text) - 1
        t = TextBox3.Text
        Table1.Rows(j).Cells(i).Text = t
    End Sub
End Class
```

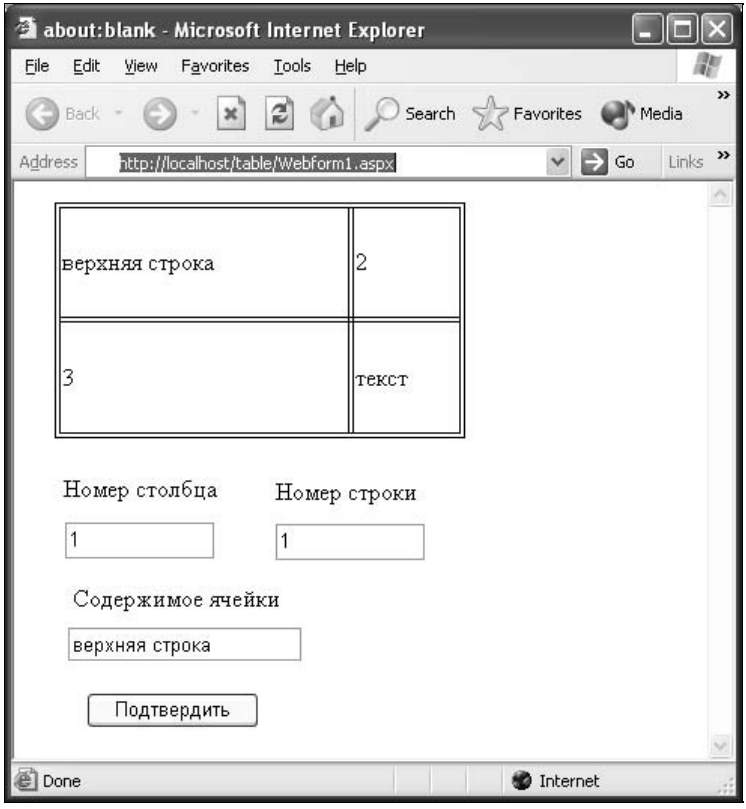


Рис. 3.28. Внешний вид Web-страницы, код которой приведен в листингах 3.43 и 3.44

Сам код достаточно прозрачен. В самом начале подпрограммы мы объявляем две целочисленные переменные, в которых будут храниться номер строки и столбца изменяемой ячейки. Также объявляется переменная строкового типа, в которую мы поместим текст, для размещения пользователем в ячейке таблицы. Сама же подпрограмма достаточно проста. При помощи функции

`Val` мы переводим текстовые значения в целочисленные. Так как пользователь будет нумеровать строки и столбцы, начиная с единицы, а в объекте `Table` их нумерация начинается с нуля, потребуется из полученных значений вычесть по единице. А затем при помощи коллекции строк таблицы и коллекции ячеек, входящих в состав строки, передать в ячейку строку, введенную пользователем.

Внешний вид этой Web-страницы при просмотре ее при помощи браузера Internet Explorer показан на рис. 3.28.

Естественно, конечный HTML-код этой Web-страницы будет отличаться от того, который был создан средой разработки. Та версия кода, которая передается конечному пользователю, приведена в листинге 3.45.

Листинг 3.45

```
<!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" con-
tent="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body MS_POSITIONING="GridLayout">
  <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value=
"dDwtMTM2NjA4OTAyMDt0PDtsPGk8MT47PjtsPHQ8O2w8aTwxPjs+O2w8dDw7bDxpPDA+O2k8
MT47PjtsPHQ8O2w8aTwwPjs+O2w8dDxwPHA8bDxUZXh0Oz47bDzQstC10YDRhdC90Y/RjyDRg
dGC0YDQvtC60LA7Pj47Pjs7Pjs+Pjt0PDtsPGk8MT47PjtsPHQ8cDxwPGw8VGV4dDs+O2w80Y
LQtdC60YHRgjs+Pjs+Ozs+Oz4+Oz4+Oz4+Oz4=" />

   <table id="Table1" border="0" style="border-width:1px;border-style:
Solid;height:157px;width:273px;Z-INDEX: 101; LEFT: 33px; POSITION:
absolute; TOP: 26px">
 <tr style="border-width:1px;border-style:solid;">
  <td style="border-width:1px;border-style:solid;">
   верхняя строка
  </td><td style="border-width:1px;border-style:solid;">
   2
  </td>
 </tr><tr>
```

```
  <td style="border-width:1px;border-style:solid;">
   3
  </td><td style="border-width:1px;border-style:solid;">
   текст
  </td>
 </tr>
</table>
   <input name="TextBox2" type="text" value="1" id="TextBox2"
style="height:24px;width:99px;Z-INDEX: 105; LEFT: 180px; POSITION:
absolute; TOP: 240px" />
   <span id="Label1" style="Z-INDEX: 102; LEFT: 39px; POSITION: absolute;
TOP: 207px">Номер столбца</span>
   <span id="Label2" style="Z-INDEX: 103; LEFT: 180px; POSITION:
absolute; TOP: 209px">Номер строки</span>
   <input name="TextBox1" type="text" value="1" id="TextBox1"
style="height:24px;width:99px;Z-INDEX: 104; LEFT: 40px; POSITION:
absolute; TOP: 239px" />
   <span id="Label3" style="height:19px;width:231px;Z-INDEX: 106; LEFT:
45px; POSITION: absolute; TOP: 280px">Содержимое ячейки</span>
   <input name="TextBox3" type="text" value="верхняя строка"
id="TextBox3" style="Z-INDEX: 107; LEFT: 42px; POSITION: absolute; TOP:
309px" />
   <input type="submit" name="Button1" value="Подтвердить" id="Button1"
style="Z-INDEX: 108; LEFT: 54px; POSITION: absolute; TOP: 352px" />
  </form>
 </body>
</HTML>
```

Итак, на этом простом примере мы научились динамически изменять содержимое таблиц, размещенных на Web-страницах. Необходимо, конечно, отметить, что разработчик имеет возможность изменять не только содержимое таблиц, но и их структуру, добавляя новые столбцы и строки, но подобной возможностью следует пользоваться очень осторожно, так как из-за этого может нарушиться вся верстка Web-страницы.

Аутентификация и авторизация пользователей

Любой более или менее разветвленный сайт должен уметь сортировать своих посетителей по уровню их прав. Так, обычному посетителю будет открываться только информационная часть сайта, но допускать его до административных интерфейсов сайта не рекомендуется. Сайты, практикующие

подписку на предоставляемые ресурсы, дают всем пользователям только часть своего контента, а доступ к полной своей версии предлагают своим подписчикам. Следовательно, Web-разработчик должен уметь использовать механизмы обеспечения безопасности, применительно к своим сайтам.

При разработке сайтов для обеспечения безопасности часто используется связка из двух процедур — аутентификации и авторизации. Аутентификация позволяет приложению узнать, какой именно пользователь посетил сайт. 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 Web-страницы или их категории будут доступны данному конкретному посетителю. Именно процедура авторизации сообщит сайту, что администратору будут предоставляться все Web-страницы, легальному подписчику все страницы с контентом, обычному гостю — страницы с общедоступной информацией, а хакеру, который ввел украденный пароль, — всего одна Web-страница с официальным предупреждением. Естественно, авторизация может проводиться только после завершения аутентификации.

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

В ASP.NET предусмотрено три варианта аутентификации. Разработчик может воспользоваться аутентификацией, основанной на обычной процедуре распознавания пользователя операционной системой Windows, аутентификацией на основе cookie-файлов, и аутентификацией, основанной на технологии Microsoft Passport. И, конечно, разработчик может вообще не использовать проверку подлинности, но это уже удел самых простых сайтов.

Для начала рассмотрим механизм использования аутентификации, основанной на механизме опознания пользователя операционной системой Windows. В этом случае сервер будет выводить диалоговое окно, в котором пользователь введет свое имя и пароль. Введенное имя и пароль должны совпадать с учетной записью используемого домена Windows, т. е. все пользователи сайта должны быть прописаны в операционной системе. Естественно, подобный вариант аутентификации будет плохо уживаться с требованиями обычных сайтов, так как использовать учетные записи операционной системы для всех посетителей сайта — явно не самый лучший вариант. Но вот при работе корпоративного сайта в интрасети это практически лучший выбор, так как все пользователи гарантированно имеют свои учетные записи в домене, и разработчик имеет возможность без лишних хлопот аутентифицировать и авторизовать пользователя.

Второй вариант аутентификации, как мы уже говорили, основан на применении cookies. Когда пользователь первый раз появляется на сайте, Web-приложение запрашивает его регистрационное имя и пароль. После прохождения первичной идентификации приложение формирует cookie-файл,

который записывается в локальной системе удаленного пользователя. Затем, когда пользователь будет запрашивать какую-либо Web-страницу, этот cookie-файл будет автоматически передаваться браузером на сервер, и на основе этого cookie пользователь будет автоматически распознаваться. Таким образом, мы избавляем пользователя от необходимости вводить пароль для доступа к каждой защищенной Web-странице. После завершения сеанса работы приложение может объявить созданный cookie-файл устаревшим, и при следующем появлении этого пользователя на сайте снова запросить его регистрационное имя или пароль, либо приложение может пользоваться единожды созданным cookie постоянно, опознавая пользователя самостоятельно при каждом его последующем заходе на сайт.

Третий 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. Запись о пользователе в базе данных этого сервера будет являться пропуском ко всем сайтам, 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 регистрационных имен ему приходилось вводить на самых различных сайтах. Поэтому, когда пользователь заходит в один из своих любимых онлайновых магазинов, ему приходится или вспоминать свое регистрационное имя и пароль, либо пользоваться службой напоминания пароля, когда на его адрес электронной почты сайт высылает идентификационные данные. Вот только надо еще вспомнить, какой именно адрес электронной почты вводился на этом сайте.

Microsoft Passport снимает подобную проблему, так как он хранит данные о пользователе, и сайту 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. Таким образом, 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 подобной 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 один раз с сервером Microsoft? Или, если его все-таки взломают, несмотря на обещанную "непробиваемую защиту"? Все мы прекрасно знаем, что абсолютной защиты не бывает, и вероятность утери

или разглашения конфиденциальных данных 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, я с трудом верю в подобный исход.

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, т. е. количество сайтов, 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. Естественно, подобные действия весьма типичны для корпорации Microsoft, однако они будут достаточно серьезно сдерживать внедрение Microsoft Passport в массы.

Впрочем, делать прогнозы сейчас — дело неблагодарное. Microsoft активно продвигает с помощью маркетинговых мероприятий идею Passport. При этом основной упор корпорация делает на развитие сервисов, которые опознают пользователей именно по их цифровым паспортам. Данная инициатива носит наименование HailStorm. О создании сервисов речь пойдет в последней главе этой книги, поэтому пока мы не будем останавливаться на этой теме.

Исходя из вышеперечисленного, становится ясно, что на настоящий момент разработчик может использовать только два варианта аутентификации пользователей. Для обычных сайтов стоит применять аутентификацию на основе cookies, а для Web-приложений, действующих в среде intranet, следует использовать стандартную аутентификацию Windows.

Следует упомянуть, что в силу своей достаточно большой функциональности ASP.NET позволяет разработчику создавать свои собственные механизмы аутентификации, но следует весьма аккуратно разрабатывать архитектуру подобных процедур. Дело в том, что обычная пересылка пароля и регистрационного имени может быть перехвачена обычным анализатором трафика TCP. Поэтому, если в базе данных вашего сайта хранятся достаточно критичные для пользователей данные, не стоит уповать только на ввод регистрационного имени и пароля. Более того, даже использование cookie-файлов не всегда решает эту проблему. Если пользователь раз за разом передает один и тот же пароль, пусть даже и зашифрованный, или при пересылке запросов серверу отсылается один и тот же cookie-файл, то злоумышленник, использующий перехватчик сетевых пакетов, сможет вычленить эти данные из общего трафика, а затем воспользоваться ими. Поэтому стоит каким-либо образом менять пароли и cookie-файлы спустя некоторое время их использования, либо вообще после каждого сеанса работы пользователя. Также

при пересылке конфиденциальных данных следует использовать протокол SSL (Secure Socket Layer), который шифрует передаваемые данные. Это, конечно, несколько замедляет работу, но все разумные способы, затрудняющие жизнь злоумышленникам, должны быть использованы.

Итак, мы узнали, какие именно механизмы аутентификации пользователей предлагает нам ASP.NET. Теперь предстоит рассмотреть примеры применения аутентификации на основе cookie-файлов и на основе стандартных механизмов опознания пользователя Windows. Прежде всего, следует отметить, что процессы аутентификации и авторизации пользователей включаются в состав Web-приложения при помощи файла Config.Web. Нетрудно догадаться, что этот файл хранит информацию о конфигурации приложения. Несколько более детально процедуру конфигурирования приложений мы будем рассматривать в одном из следующих разделов этой главы, а пока лишь вкратце рассмотрим структуру этого файла.

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

У тега `<authentication>` 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 использовать этот вариант идентификации пользователя, следует установить на сервере соответствующий Passport SDK.

В нашем случае, естественно, мы используем значение `Forms`. Общий вид конфигурационного файла Config.Web для создаваемого нами приложения приведен в листинге 3.46.

Листинг 3.46

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <system.Web>
  <compilation defaultLanguage="vb" debug="true" />
  <customErrors mode="RemoteOnly" />
  <authentication mode="Forms">
   <forms name="demoauth" loginUrl="login.aspx" protection="All"
timeout="30" path="/">
   </forms>
  </authentication>
  <authorization>
    <allow users="*" /> <!-- Allow all users -->
    <!--  <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>
  <trace enabled="false" requestLimit="10" pageOutput="false"
traceMode="SortByTime" localOnly="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" />
 <httpHandlers>
 <add verb="*" path="*.vb" type=
"System.Web.HttpNotFoundHandler,System.Web" />
 <add verb="*" path="*.cs" type=
"System.Web.HttpNotFoundHandler,System.Web" />
 <add verb="*" path="*.vbproj" type=
"System.Web.HttpNotFoundHandler,System.Web" />
 <add verb="*" path="*.csproj" type=
"System.Web.HttpNotFoundHandler,System.Web" />
 <add verb="*" path="*.Webinfo" type=
"System.Web.HttpNotFoundHandler,System.Web" />
 </httpHandlers>
  <globalization requestEncoding="utf-8" responseEncoding="utf-8" />
 </system.Web>
</configuration>
```

Нас, естественно, будет интересовать именно секция `<authentication>`. Легко заметить, что кроме указания соответствующего режима аутентификации пользователей при помощи атрибута `mode`, мы разместили в ней дополнительный тег `<forms>` с различными атрибутами. Этот тег при помощи атрибутов задает конкретные свойства используемой процедуры аутентификации. Рассмотрим эти атрибуты.

- `loginUrl`. Значением данного атрибута является URL страницы, на которую отсылается пользователь, не прошедший аутентификацию. В нашем случае мы используем страницу с именем "login.aspx".
- `name`. Этот атрибут задает наименование cookie, в котором и будут храниться идентификационные данные удаленного пользователя.
- `timeout`. Свойство задает временной период, в течение которого cookie-файл будет считаться действительным, если даже пользователь не отвечает на запросы. Использование подобного периода тайм-аута позволяет уменьшить риск возникновения ситуации, когда пользователь ввел свое имя и пароль, а затем прервал работу. Если после этого злоумышленник начнет работу с сайтом с локальной машины пользователя, то сайт получит правильный cookie и будет считать злоумышленника легальным пользователем. Если же разработчик явно установит продолжительность периода тайм-аута, то по истечении этого времени с последнего запроса, пользователь вновь будет считаться не прошедшим аутентификацию, и сайт отправит его на страницу ввода имени и пароля. Следует особо отметить, что период тайм-аута начинает отсчитываться не с момента первого ввода регистрационного имени и пароля, а каждый раз начинается заново, когда браузер пользователя отправляет запрос на сервер. Значением данного атрибута является число, указывающее длительность периода тайм-аута в минутах. По умолчанию используется значение, равное тридцати.
- `path`. В этом параметре указывается путь к виртуальным каталогам, для которых данный cookie-файл будет действительным. При помощи этого параметра мы можем создавать несколько cookie-файлов, в каждом из которых будет храниться уникальная информация. Это позволит создать механизм многоступенчатой аутентификации, когда для доступа ко всем более закрытым отделам сайта, пользователь будет вынужден вводить свои дополнительные пароли. Но в последнее время подобной системе ступенчатого ввода паролей все чаще присваивают эпитет "параноидальная". Разработчик должен очень тщательно отнестись к проблеме поиска баланса между защищенностью Web-приложения и удобством работы пользователя. По умолчанию для данного атрибута используется значение "/", которое указывает, что создаваемый cookie-файл будет действителен для всех виртуальных каталогов и Web-страниц, входящих в состав сайта.

- `protection`. Параметр устанавливает правила защиты создаваемого cookie-файла от подделок. Параметр может иметь одно из следующих четырех значений:
  - `None`. Значение указывает, что информация, записываемая в cookie-файл пользователя, никаким образом не защищается от взлома. Естественно, это несколько ускоряет работу приложений, но может использоваться только в тех случаях, когда неверная идентификация пользователя не повлечет за собой каких-либо убытков;
  - `Encription`. При использовании этого значения содержимое cookie-файла шифруется с помощью алгоритмов DES или "тройной DES" в зависимости от настроек операционной системы (как известно, в США существуют ограничения на экспорт программ с "сильной криптографией", поэтому официально за рубеж поставляется версия Windows с ослабленными криптографическими возможностями). Однако использование этого режима не позволяет полностью быть уверенным, что содержимое cookie-файла не изменили, так как он остается подверженным традиционным криптоатакам;
  - `Validation`. Значение указывает, что cookie-файл не будет шифроваться, однако он будет проверяться на подлинность. Ключ подлинности, вырабатываемый Web-приложением, будет добавлен к содержимому самого cookie-файла. К получившимся данным будет добавлен MAC-адрес удаленного клиента;
  - `All`. Значение используется по умолчанию. Оно указывает, что для поддержки безопасности идентификационных данных будет использоваться одновременно и шифрование, и проверка подлинности.

Теперь, после того, как мы узнали, каким образом настраивать приложения для поддержки аутентификации, можно перейти к примеру приложения. Для демонстрации концепции аутентификации на основе cookie-файлов, нам потребуется помимо конфигурационного файла еще две страницы. На стартовой странице мы разместим обычное приветствие, в котором будет упоминаться имя пользователя, а вторая страница будет содержать форму для ввода имени и пароля. По умолчанию будет загружаться стартовая страница, но если пользователь еще не проходил аутентификацию, он будет автоматически отсылаться на страницу ввода имени и пароля без участия самого Web-приложения, т. е. разработчику для этого не придется прикладывать каких-либо дополнительных усилий.

На первой странице мы разместим один компонент `Label`, текстовое содержимое которого будет формироваться автоматически при загрузке страницы. Также на первой странице мы разместим один независимый переключатель, реализуемый при помощи компонента `CheckBox`. Если пользователь установит флажок в этом переключателе, то его учетная запись будет удалена, и в следующий раз ему придется снова проходить аутентификацию. Естест-

венно, свойство AutoPostBack для этого компонента необходимо будет установить в значение True. Также установим для стартовой страницы имя default.aspx, чтобы к ней можно было обращаться по умолчанию.

HTML-код начальной Web-страницы в режиме ее разработки приведен в листинге 3.47.

Листинг 3.47

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="default.aspx.vb" Inherits="cookieauth.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:Label id="Label1" style="Z-INDEX: 101; LEFT: 28px; POSITION:
absolute; TOP: 30px" runat="server" Width="158px" Height="19px">
Здравствуйте,</asp:Label>
   <asp:CheckBox id="CheckBox1" style="Z-INDEX: 102; LEFT: 38px;
POSITION: absolute; TOP: 93px" runat="server"
Text="Удалить мою учетную запись" AutoPostBack="True"></asp:CheckBox>
  </form>
 </body>
</HTML>
```

Текстовую надпись на стартовой странице мы будем формировать при загрузке Web-страницы, следовательно, снова придется перехватывать событие Page_Load. При этом мы будем использовать имя пользователя. Извлекать его придется из соответствующего объекта 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 регистрационное имя пользователя, надо использовать конструкцию `User.Identity.Name`. Именно это и показано в листинге 3.48, в котором приведен код класса, реализующего стартовую Web-страницу.

Листинг 3.48

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents CheckBox1 As System.Web.UI.WebControls.CheckBox
    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
        Label1.Text = "Здравствуйте, " + User.Identity.Name
    End Sub

    Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
        If CheckBox1.Checked Then
            Web.Security.FormsAuthentication.SignOut()
```

```
            Response.Redirect("login.aspx")
        End If
    End Sub
End Class
```

Процедуру, выполняемую при загрузке Web-страницы, мы обсуждали перед листингом. Но в нем находится еще одна процедура, о которой мы еще не говорили. Это обработчик события `CheckBox1.CheckedChanged`. Это событие будет инициироваться, когда пользователь изменит состояние независимого переключателя `Checkbox1`, расположенного на Web-странице. В том случае, если пользователь отметил флажок в этом переключателе, следует исключить его из списка аутентифицированных пользователей, и вновь отправить на страницу с формой для ввода логина и пароля.

Для того чтобы признать аутентификацию пользователя недействительной, следует выполнить метод `SignOut` объекта `FormsAuthentication`. Нужно отметить, что объект `FormsAuthentication` принадлежит пространству имен System.Web.Security, которое по умолчанию не включается в разрабатываемое приложение. Поэтому следует или указывать полное имя объекта, как мы и сделали в примере, либо импортировать это пространство имен для приложения. Импортирование пространства имен System.Web.Security будет производиться при помощи следующей директивы:

```
<%@ Import Namespace="System.Web.Security" %>
```

После того, как аутентификация пользователя будет признана недействительной, его необходимо будет переправить на страницу для ввода имени и пароля. Для этого мы используем метод `Redirect` объекта `Response`.

Теперь перейдем к разработке Web-страницы, предназначенной для ввода пользователем своего имени и пароля. Нам потребуется разместить на ней два поля текстового ввода вместе с текстовыми пояснениями к ним и одну кнопку, подтверждающую ввод данных. HTML-код Web-страницы в режиме ее разработки приведен в листинге 3.49.

Листинг 3.49

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="Login.aspx.vb"
Inherits="cookieauth.Login"%>
<!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:Label id="Label1" style="Z-INDEX: 101; LEFT: 36px; POSITION:
absolute; TOP: 27px" runat="server" Width="219px" Height="19px">
Введите свой логин и пароль</asp:Label>
   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 41px; POSITION:
absolute; TOP: 65px" runat="server">Логин</asp:Label>
   <asp:Label id="Label3" style="Z-INDEX: 103; LEFT: 40px; POSITION:
absolute; TOP: 101px" runat="server">Пароль</asp:Label>
   <asp:TextBox id="Log" style="Z-INDEX: 104; LEFT: 103px; POSITION:
absolute; TOP: 60px" runat="server"></asp:TextBox>
   <asp:TextBox id="Pass" style="Z-INDEX: 105; LEFT: 104px; POSITION:
absolute; TOP: 95px" runat="server"></asp:TextBox>
   <asp:Button id="Button1" style="Z-INDEX: 106; LEFT: 47px; POSITION:
absolute; TOP: 143px" runat="server" Height="24px" Width="169px"
Text="Подтвердить"></asp:Button>
   <asp:Label id="Label4" style="Z-INDEX: 107; LEFT: 47px; POSITION:
absolute; TOP: 190px" runat="server" ForeColor="Red" Visi-
ble="False">Неверный логин или пароль</asp:Label>
  </form>
 </body>
</HTML>
```

Анализируя приведенный листинг, легко заметить, что на Web-странице помимо основных органов управления мы поместили еще один текстовый компонент `Label`, который изначально сделан невидимым. Он предназначен для отображения текстовой информации о вводе неправильного пароля или логина. В том случае, если логин и пароль не будут опознаны, эта надпись будет сделана видимой. Естественно, во избежание сохранения ее видимого состояния при последующих загрузках можно еще при каждой загрузке страницы принудительно делать ее невидимой. Все эти действия реализованы в классе, соответствующем этой Web-странице. Код искомого класса приведен в листинге 3.50.

Листинг 3.50

```
Public Class Login
    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 Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Log As System.Web.UI.WebControls.TextBox
    Protected WithEvents Pass As System.Web.UI.WebControls.TextBox
    Protected WithEvents Label4 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
        Label4.Visible = False
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles Button1.Click
        If (Log.Text = "Igor") And (Pass.Text = "mypassword") Then
  Web.Security.FormsAuthentication.RedirectFromLoginPage(Log.Text, True)
        Else
            Label4.Visible = True
        End If
    End Sub
End Class
```

Рассмотрим действия, выполняемые приложением после того, как пользователь заполнит поля для ввода регистрационного имени и пароля и нажмет

подтверждающую кнопку. Прежде всего, необходимо узнать, какие именно данные ввел пользователь и сравнить их с допустимым именем и паролем. В нашем случае, доступ к основной странице предоставляется только пользователю с именем Igor, которому соответствует пароль mypassword. В том случае, если введенные пользователем данные удовлетворяют этому условию, пользователь переправляется на страницу, которая и затребовала проведение аутентификации. Для этого используется метод `RedirectFromLoginPage` объекта `FormsAuthentication`. Этому методу передаются два параметра. В качестве первого параметра передается имя пользователя. Второй параметр имеет логический тип `Boolean`. В том случае, если второй параметр имеет значение `True`, созданный идентификационный cookie-файл сохраняется на машине пользователя, и в следующий визит ему уже не придется снова проходить процедуру аутентификации.

В том случае, если имя и пароль, введенные пользователем, не соответствуют установленному условию, надпись о запрете доступа визуализируется, и пользователь остается на текущей странице. На рис. 3.29 и 3.30 показан внешний вид разработанных Web-страниц при отображении их браузером Internet Explorer.

Естественно, метод проверки правильности идентификационных данных, показанный в примере, практически никогда не будет использоваться, так как обычно учетных записей пользователей достаточно много и все их сохранять в коде страницы просто невозможно. Обычно эти данные хранятся либо в базе данных, либо в отдельном файле, и процедура аутентификации пользователя будет занимать несколько больше времени.

Рис. 3.29. Страница для ввода регистрационного имени и пароля

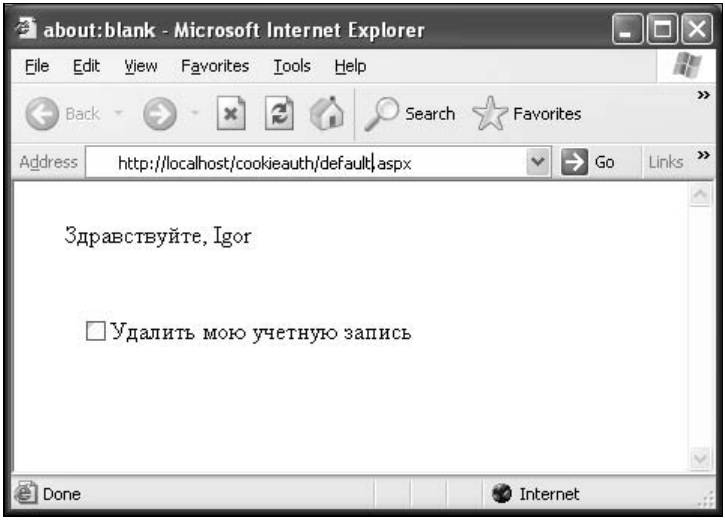


Рис. 3.30. Страница, загруженная после аутентификации пользователя

Впрочем, есть еще один вариант хранения учетных записей пользователей. Их идентификационные данные могут храниться в конфигурационном файле Config.Web. Для этого необходимо в модуль авторизации добавить раздел, объявляемый тегом `<credentials>`. В общем виде этот раздел при хранении в нем учетных записей пользователей будет выглядеть следующим образом:

```
<authentication mode="Forms">
  <forms name="demoauth" loginUrl="login.aspx" protection="All"
timeout="30" path="/">
 <credentials passwordFormat="Clear" >
 <user name="User1" password="pass1"/>
 <user name="User2" password="pass2"/>
 </credentials>
 </forms>
</authentication>
```

Видно, что у тега `<credentials>` существует атрибут `passwordFormat`. С его помощью указывается формат хранения паролей в конфигурационном файле. Этот атрибут должен принимать одно из следующих трех значений:

- `Clear`. Значение указывает, что пароль хранится в виде обычного текста;
- `SHA1`. Значение указывает, что пароли подвергаются хэшированию по алгоритму SHA1;
- `MD5`. Значение указывает, что пароли подвергаются хэшированию по алгоритму MD5.

Естественно, Web-приложению необходимо получать данные о пользователях из конфигурационного файла. Для этого следует использовать метод

`FormsAuthentication.Authenticate`. В качестве параметров методу передаются имя и пароль пользователя как значения типа `String`. Метод возвращает логическое значение типа `Boolean`. В том случае, если переданные методу идентификационные данные есть в конфигурационном файле, метод возвращает значение `True`.

Теперь, когда мы разобрались с аутентификацией пользователей, основанной на применении cookie-файлов, можно перейти к рассмотрению аутентификации, основанной на стандартном механизме распознавания пользователей операционной системой Windows.

Как мы уже знаем, для включения этого режима аутентификации, секция `<authentication>` в конфигурационном файле должна выглядеть следующим образом:

```
<authentication mode="Windows" />
```

Естественно, при использовании такого режима аутентификации Web-разработчику нет нужды создавать свою страницу для ввода имени и пароля, так как пользователь уже ввел свои данные, когда загружал операционную систему. Остается лишь получить эти данные и воспользоваться ими.

Разработчику не нужно знать пароля пользователя, достаточно знать лишь его регистрационное имя. А оно располагается в свойстве объекта `User`. Точнее, в свойстве `User.Identity.Name`, которое мы уже рассматривали. В системах Windows пользователи имеют не только имена. Они еще и могут относиться к разным группам. Каждая группа имеет свои собственные права. Для того чтобы узнать, принадлежит ли пользователь к какой-либо группе, следует воспользоваться методом `User.IsInRole`. В качестве параметра этому методу передается наименование какой-либо группы пользователей в Windows, а метод возвращает логическое значение типа `Boolean`. В том случае, если пользователь действительно входит в указанную группу, возвращается значение `True`.

Этот метод аутентификации хорош тем, что разработчик не должен думать, где и как хранить учетные записи пользователей. Но это преимущество одновременно является и недостатком этого метода аутентификации, так как ее могут проходить только те пользователи, которые зарегистрированы в том же домене, что и IIS. Все это естественным образом ограничивает применение этого метода пределами интрасетей.

Однако следует отметить, что, основываясь на этом методе аутентификации, разработчик получает отличные возможности для авторизации пользователей. Процесс авторизации позволяет определить, разрешено ли пользователю получить какую-либо Web-страницу или нет.

Как мы уже видели, в конфигурационном файле config.Web есть секция `<authorization>`. Именно она и позволяет определять, кому из пользователей разрешено просматривать Web-страницы, которые находятся в зоне

действия конфигурационного файла. Вот как выглядит типичный пример определения параметров авторизации:

```
<authorization>
 <allow users="user1" />
 <allow roles="Administrators" />
 <deny users="*" />
</authorization>
```

Легко заметить, что в секции `<authorization>` находятся теги `<allow>` и `<deny>`. Именно они определяют, кому будет разрешен доступ к страницам, а кому — запрещен. Теги `<allow>`, естественно, описывают разрешения на доступ, а теги `<deny>` — запрещения. Рассмотрим наш пример несколько более внимательно.

Если в теге `<allow>` мы используем атрибут `users`, то этот тег позволяет определять конкретного пользователя, которому разрешен доступ к страницам. В качестве значения атрибута следует указать имя искомого пользователя. Если использовать атрибут `roles`, то можно будет указывать наименования групп пользователей, которым разрешен доступ. В нашем случае мы разрешили доступ к Web-страницам пользователю с регистрационным именем user1 и всем членам группы `Administrators`.

Теперь рассмотрим ситуацию с запрещением доступа. Для установки перечня пользователей, которым доступ к Web-страницам запрещен, используется тег `<deny>`. В примере мы использовали все тот же параметр `users`, но для него установили значение "*". Символ звездочки в данном случае обозначает всех пользователей. Таким образом, доступ к Web-страницам, располагающимся в зоне действия файла config.Web, разрешен только перечисленным ранее пользователям, а всем остальным в доступе отказано.

Разработчик имеет возможность указать права и для анонимных пользователей, т. е. не прошедших процедуру аутентификации. Для обозначения анонимных пользователей используется значение "?", т. е. вопросительный знак.

Естественно, для целей авторизации следует более гибко использовать правила доступа. Например, некоторые Web-страницы должны быть доступны анонимным пользователям, а остальные — нет. Но если мы в единственном файле config.Web заранее откажем в доступе всем анонимным пользователям, то они не смогут получить ни одной Web-страницы. Поэтому следует использовать несколько конфигурационных файлов.

Как известно, изначально конфигурационный файл создается в корневом каталоге Web-приложения, и его действие распространяется на все остальные вложенные каталоги нижних уровней. Но если в каком-либо вложенном каталоге создать свой конфигурационный файл config.Web, то он будет перекрывать действие своего старшего собрата. Таким образом, разработчик

получает возможность сгруппировать в различных каталогах файлы с одинаковыми правами доступа и для каждого каталога создать свой конфигурационный файл.

Использование электронной почты

Весьма часто разработчику Web-приложений приходится использовать в своих проектах функцию автоматической отправки электронной почты. Чаще всего электронные письма, которые создаются и отправляются автоматически, без участия администраторов, содержат какие-либо уведомления. Например, о совершении покупки в онлайновом магазине, или о подписке на рассылку новостей сайта. Так или иначе, практически ни один из серьезных проектов не обходится без использования электронной почты, и, следовательно, нам необходимо рассмотреть механизм ее использования в работе Web-приложений.

Для этого применяется класс `SmtpMail`, располагающийся в пространстве имен System.Web.Mail. Основное его свойство `SmtpServer` типа `String` содержит адрес почтового сервера, на который будет отсылаться сообщение. А метод `Send` отсылает письмо на этот сервер. В качестве параметра этому методу передается значение типа `MailMessage`. Именно этот тип полностью определяет отсылаемое сообщение. Рассмотрим этот тип 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), т. е. 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.

Теперь, когда мы знаем структуру объектов, применяемых для отправки электронных писем, рассмотрим пример использования этой технологии. Создадим Web-приложение, которое будет предназначено для отправки пользователем сообщений по электронной почте.

На стартовой странице, которую мы сохраним под именем default.aspx, следует разместить три однострочных поля текстового ввода. В первое из них пользователь будет вводить адрес получателя письма, во второе — обратный адрес, а третье поле предназначено для указания темы отправляемого сообщения. Также нам потребуется одно многострочное поле, в котором пользователь будет писать текст сообщения, и кнопка, подтверждающая отправку сообщения. К первым двум полям ввода, которые будут предназначены для ввода адресов электронной почты, следует присоединить компоненты под названием RegularExpressionValidator. Они будут следить за правильностью ввода адресов. В листинге 3.51 приведен HTML-код Web-страницы в режиме ее разработки.

Листинг 3.51

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="default.aspx.vb" Inherits="mailer.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" con-
tent="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: 37px; POSITION:
absolute; TOP: 24px" runat="server" Width="257px" Height="19px">
Виртуальный почтамт</asp:Label>
```

```
   <asp:RegularExpressionValidator id="RegularExpressionValidator3"
style="Z-INDEX: 110; LEFT: 34px; POSITION: absolute; TOP: 149px"
runat="server" ErrorMessage="Введен неверный адрес электронной почты"
ControlToValidate="TextBox4" ValidationExpression="\w+([-+.]\w+)*@\
w+([-.]\w+)*\.\w+([-.]\w+)*"></asp:RegularExpressionValidator>

   <asp:TextBox id="TextBox4" style="Z-INDEX: 109; LEFT: 195px; POSITION:
absolute; TOP: 115px" runat="server" Width="193px"
Height="24px"></asp:TextBox>

   <asp:Label id="Label4" style="Z-INDEX: 108; LEFT: 37px; POSITION:
absolute; TOP: 122px" runat="server" Width="151px" Height=
"17px">Адрес отправителя</asp:Label>

   <asp:RegularExpressionValidator id="RegularExpressionValidator2"
style="Z-INDEX: 104; LEFT: 33px; POSITION: absolute; TOP: 91px"
runat="server" ErrorMessage="Введен неверный адрес электронной почты"
↳ControlToValidate="TextBox1" ValidationExpression="\w+([-+.]\w+)*@\
w+([-↳.]\w+)*\.\w+([-.]\w+)*"></asp:RegularExpressionValidator>

   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 37px; POSITION:
absolute; TOP: 59px" runat="server" Width="127px" Height=
"19px">Адрес получателя</asp:Label>

   <asp:TextBox id="TextBox1" style="Z-INDEX: 103; LEFT: 195px; POSITION:
absolute; TOP: 52px" runat="server" Width="193px"
Height="24px"></asp:TextBox>

   <asp:RegularExpressionValidator id="RegularExpressionValidator1"
style="Z-INDEX: 105; LEFT: 33px; POSITION: absolute; TOP: 91px"
runat="server" ErrorMessage="Введен неверный адрес электронной почты"
ControlToValidate="TextBox1" ValidationExpression="\w+([-+.]\w+)*@\
w+([-.]\w+)*\.\w+([-.]\w+)*"></asp:RegularExpressionValidator>

   <asp:Label id="Label3" style="Z-INDEX: 106; LEFT: 37px; POSITION:
absolute; TOP: 186px" runat="server">Тема письма</asp:Label>

   <asp:TextBox id="TextBox2" style="Z-INDEX: 107; LEFT: 195px; POSITION:
absolute; TOP: 177px" runat="server" Width="192px"
Height="26px"></asp:TextBox>

   <asp:Label id="Label5" style="Z-INDEX: 111; LEFT: 37px; POSITION:
absolute; TOP: 220px" runat="server">Текст письма</asp:Label>

   <asp:TextBox id="TextBox3" style="Z-INDEX: 112; LEFT: 37px; POSITION:
absolute; TOP: 246px" runat="server" Width="353px" Height="99px"
Rows="5"></asp:TextBox>

   <asp:Button id="Button1" style="Z-INDEX: 113; LEFT: 41px; POSITION:
absolute; TOP: 358px" runat="server" Width="166px" Height="24px"
Text="Отправить письмо"></asp:Button>

  </form>

 </body>

</HTML>
```

После того, как пользователь заполнит все поля и нажмет на подтверждающую кнопку, необходимо сформировать и отправить письмо. Этим будет заниматься класс, связанный с проектируемой страницей. Код этого класса приведен в листинге 3.52.

Листинг 3.52

```
Public Class WebForm1
    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 TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents RegularExpressionValidator1 As
    System.Web.UI.WebControls.RegularExpressionValidator
    Protected WithEvents Label3 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
    Protected WithEvents RegularExpressionValidator2 As
    System.Web.UI.WebControls.RegularExpressionValidator
    Protected WithEvents Label4 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox4 As System.Web.UI.WebControls.TextBox
    Protected WithEvents RegularExpressionValidator3 As
    System.Web.UI.WebControls.RegularExpressionValidator
    Protected WithEvents Label5 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
    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 email As New Web.Mail.MailMessage()
        Dim smtp As New Web.Mail.SmtpMail()
        email.To = TextBox1.Text
        email.From = TextBox4.Text
        email.Subject = TextBox2.Text
        email.Body = TextBox3.Text
        smtp.SmtpServer = "my.mail.ru" 'Задали имя используемого SMTP-сервера
        smtp.Send(email)
        Response.Redirect("Webform.aspx")
    End Sub
End Class
```

Рассмотрим несколько внимательнее обработчик события `Button1_Click`. Для отправки электронного письма нам потребуются две переменные. Одна из них будет иметь тип `MailMessage`, и в ней будет располагаться полное описание отсылаемого сообщения, а вторая переменная будет иметь тип `SmtpMail`. Вторая переменная предназначена уже для отсылки созданного сообщения.

При объявлении этих переменных мы одновременно и создаем экземпляры соответствующих объектов при помощи оператора `New`. Следует также отметить, что искомые классы находятся в пространстве имен `System.Web.Mail`. Так как прямые вызовы конструкторов этих классов встречаются в нашей процедуре всего два раза, автор счел возможным не импортировать все пространство имен, а вместо этого использовать полные наименования конструкторов.

После того, как переменные созданы, можно приступать к их заполнению. Данные, введенные пользователем в поля ввода, мы присваиваем соответствующим свойствам переменной `email`. После того, как электронное сообщение сформировано, его необходимо отправить. Для этого необходимо воспользоваться переменной `smtp`. Сначала задать доменное имя или IP-адрес используемого SMTP-сервера при помощи свойства `SmtpServer`, а затем, воспользовавшись методом `Send`, отправить письмо.

После того, как письмо будет отправлено, пользователь будет переадресован на другую Web-страницу, на которой отображается всего лишь сообщение о том, что электронное письмо отправлено успешно. В силу того, что эта страница максимально проста, мы не будем приводить ее листинга здесь.

Внешний вид основной страницы нашего Web-приложения показан на рис. 3.31, HTML-код этой страницы приведен в листинге 3.53.

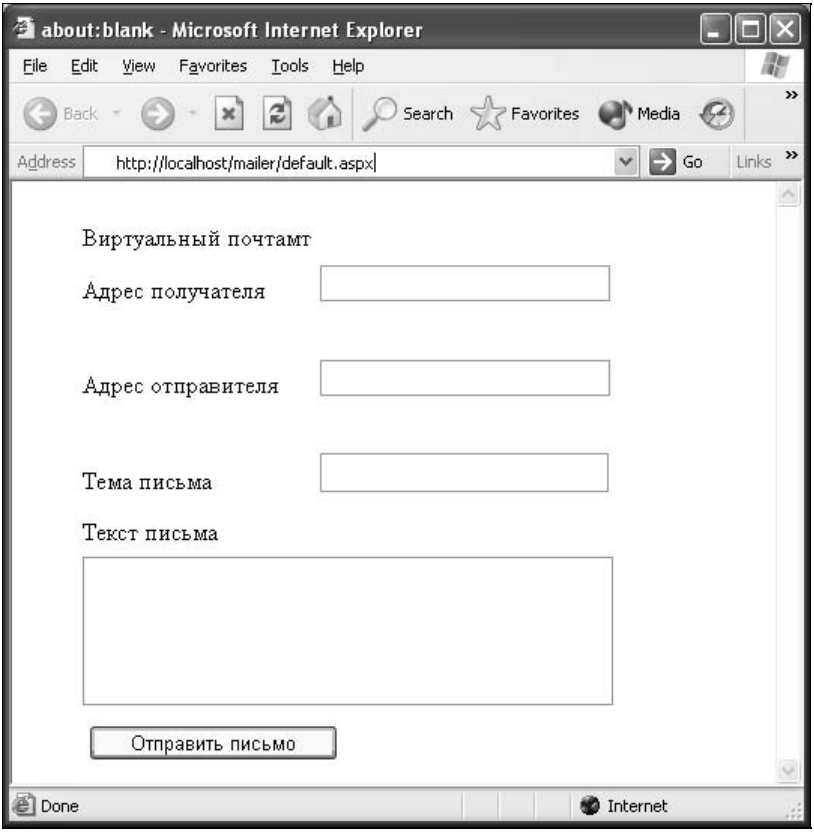


Рис. 3.31. Стартовая страница почтового Web-приложения

Листинг 3.53

```
<!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="default.aspx" language=
"javascript" ↳onsubmit="ValidatorOnSubmit();" id="Form1">

<input type="hidden" name="__VIEWSTATE" value="dDwtNzQ1ODMxMDQ7Oz4=" />

 <script language="javascript" src="/aspnet_client/system_Web/
1_0_2914_16/WebUIValidation.js"></script>

   <span id="Label1" style="height:19px;width:257px;Z-INDEX: 101; LEFT:
37px; POSITION: absolute; TOP: 24px">Виртуальный почтамт</span>

   <span id="RegularExpressionValidator3" controltovalidate="TextBox4"
errormessage="Введен неверный адрес электронной почты" evaluationfunction=
"RegularExpressionValidatorEvaluateIsValid" validationexpression="\
w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" style="color:Red;Z-INDEX:110;
LEFT:34px;POSITION:absolute;TOP:149px;visibility:hidden;">Введен неверный
адрес электронной почты</span>

   <input name="TextBox4" type="text" id="TextBox4"
style="height:24px;width:193px;Z-INDEX: 109; LEFT: 195px; POSITION:
absolute; TOP: 115px" />

   <span id="Label4" style="height:17px;width:151px;Z-INDEX: 108; LEFT:
37px; POSITION: absolute; TOP: 122px">Адрес отправителя</span>

   <span id="RegularExpressionValidator2" controltovalidate="TextBox1"
errormessage="Введен неверный адрес электронной почты" evaluationfunction=
"RegularExpressionValidatorEvaluateIsValid" validationexpression="\
w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" style="color:Red;Z-INDEX:104;
EFT:33px;POSITION:absolute;TOP:91px;visibility:hidden;
">Введен неверный адрес электронной почты</span>

   <span id="Label2" style="height:19px;width:127px;Z-INDEX: 102; LEFT:
37px; POSITION: absolute; TOP: 59px">Адрес получателя</span>

   <input name="TextBox1" type="text" id="TextBox1"
style="height:24px;width:193px;Z-INDEX: 103; LEFT: 195px; POSITION:
absolute; TOP: 52px" />

   <span id="RegularExpressionValidator1" controltovalidate="TextBox1"
errormessage="Введен неверный адрес электронной почты" evaluationfunction=
"RegularExpressionValidatorEvaluateIsValid" validationexpression="\
w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" style="color:Red;Z-INDEX:105;
LEFT:33px;POSITION:absolute;TOP:91px;visibility:hidden;">Введен неверный
адрес электронной почты</span>

   <span id="Label3" style="Z-INDEX: 106; LEFT: 37px; POSITION: absolute;
TOP: 186px">Тема письма</span>

   <input name="TextBox2" type="text" id="TextBox2" style=
"height:26px;width:192px;Z-INDEX: 107; LEFT: 195px; POSITION: absolute;
TOP: 177px" />

   <span id="Label5" style="Z-INDEX: 111; LEFT: 37px; POSITION: absolute;
TOP: 220px">Текст письма</span>

   <input name="TextBox3" type="text" id="TextBox3" style=
"height:99px;width:353px;Z-INDEX: 112; LEFT: 37px; POSITION: absolute;
TOP: 246px" />
```

```
   <input type="submit" name="Button1" value="Отправить письмо" onclick=
"if (typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); "
language="javascript" id="Button1" style="height:24px;width:166px;
Z-INDEX: 113; LEFT: 41px; POSITION: absolute; TOP: 358px" />
 <script language="javascript">
<!--
 var Page_Validators =  new
Array(document.all["RegularExpressionValidator3"],
document.all["RegularExpressionValidator2"],
document.all["RegularExpressionValidator1"]);
  // -->
</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>
```

Естественно, HTML-код страницы получился столь большим из-за того, что при ее создании были использованы компоненты системы проверки

достоверности, которые реализуются при помощи Java-сценариев, действующих на стороне пользователя.

Конфигурирование приложений

В этом разделе мы кратко рассмотрим возможности конфигурирования приложений при помощи дополнительных файлов. Начнем мы с рассмотрения конфигурационных файлов config.Web.

Любое приложение ASP.NET всегда имеет как минимум один 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, в одном из предыдущих разделов этой главы мы уже говорили, что разработчик имеет возможность поместить конфигурационный файл в любом вложенном каталоге приложения, и в этом случае, 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 в ASP.NET имеет, естественно, заранее определенную структуру. Этот файл имеет XML-синтаксис. Пример конфигурационного файла был приведен в разделе "*Аутентификация и авторизация пользователей*", поэтому сейчас мы не будем приводить его листинг еще раз.

Теперь рассмотрим вкратце структуру этих конфигурационных файлов. 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 ASP.NET с сервером IIS.
- <browsercaps>. Раздел предназначен для настройки компонента, определяющего возможности браузера, используемого удаленным пользователем.

Помимо конфигурационных файлов, которые мы только что кратко рассмотрели, в приложениях ASP.NET 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. 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 того или иного события в приложении. Пример подобного файла приведен в листинге 3.54.

Листинг 3.54

```
<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 Session_Start(Sender As Object, E As EventArgs)
        Response.Write("Session is Starting...<br>")
    End Sub

    Sub Session_End(Sender As Object, E As EventArgs)
        ' Clean up session 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 этот листинг, станет понятно, что при помощи файла Global.asax мы установили обработчики событий для всего приложения и для отдельных сессий. 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. Подобным же образом объявляются обработчики начала и завершения сессий в рамках работы приложения. Как видно, ничего сложного в использовании файла Global.asax нет.

Отладка приложений

Следует признать, что наличие ошибок является, пожалуй, неотъемлемой частью программных продуктов. Уильям Гейтс недавно заявил, что "безошибочность кода можно гарантировать, если код занимает не более пяти строк". Утверждение, конечно, нельзя назвать точным, но немалая доля истины в нем есть. Поэтому без отладки приложений ни один разработчик не сможет обойтись.

Традиционно отладка Web-приложений являлась достаточно трудной в силу специфики их работы. Естественно, исполнение Web-приложений в пошаговом режиме так и остается пока недостижимой мечтой, но содержимое различных переменных разработчики могли получить. При этом приходилось либо подавлять вывод основной информации, замещая ее содержимым переменных, либо комбинировать основное содержимое страниц с переменными. После завершения отладки требовалось вычищать все отладочные инструкции из кода приложения, что опять отнимало драгоценное время разработчика.

ASP.NET облегчило жизнь программистам и в этом аспекте. Теперь механизмы отладки встроены в ASP.NET и позволяют без затруднений выводить самую разнообразную справочную информацию вместе с основным содержимым Web-страниц, а по завершении сеансов отладки практически мгновенно избавляться от ее следов.

Отладка в ASP.NET разбита на два уровня. Программист может использовать отладку на уровне отдельных страниц (Page-level tracing) или на уровне

всего приложения (Application-level tracing). Понятно, что отладка на уровне отдельных страниц предназначена для выявления локальных проблем, а отладка на уровне приложения применяется в тех случаях, когда разработчик подозревает, что проблема не локализована в одной Web-странице.

Рассмотрение отладки мы начнем, естественно, с уровня страниц. Достаточно в HTML-код страницы добавить в раздел объявления ASP-страницы `<%Page …%>` параметр `Trace="True"`. И в этом случае вместе с основным содержимым страницы будет отображена отладочная информация. HTML-код простейшей страницы, которая выводит отладочную информацию, приведен в листинге 3.55.

Листинг 3.55

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb"
Inherits="tracing.WebForm1" traceMode="SortByCategory" trace="True"%>
<!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">
  <p>
   Отлаживаемая страница
  </p>
  <form id="Form1" method="post" runat="server">
    
  </form>
 </body>
</HTML>
```

Казалось бы, что эта Web-страница чрезвычайно проста, однако при просмотре ее в браузере отображается просто огромное количество отладочной информации, как это показано на рис. 3.32. Конечный HTML-код этой страницы мы не будем приводить из-за его огромного объема.

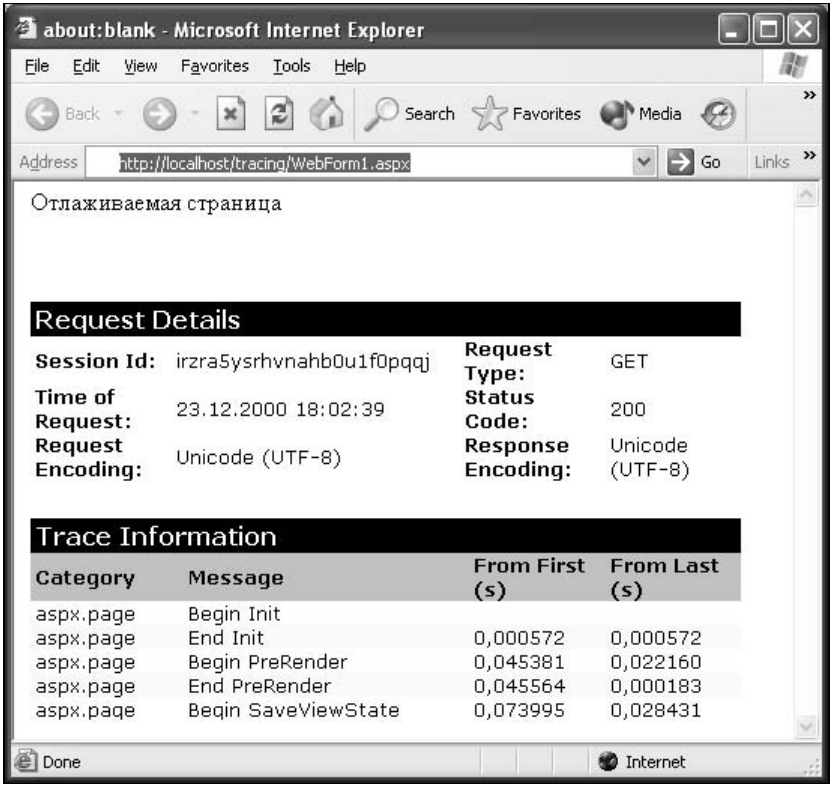


Рис. 3.32. Внешний вид Web-страницы с отладочной информацией в браузере

Помимо этой возможности разработчик имеет возможность создавать свои собственные отладочные сообщения, воспользовавшись методами `Trace.Write` и `Trace.Warn`. Эти методы, естественно, вызываются в различных процедурах класса, реализующего искомую Web-страницу. В качестве параметров этим методам передается два значения типа `String`. Первая строка содержит наименование категории, в которой необходимо будет отобразить отладочное сообщение, а второй параметр содержит текст отладочного сообщения. Единственное отличие между этими методами состоит в том, что метод `Trace.Write` выводит сообщения символами черного цвета, а метод `Trace.Warn` для этого использует символы красного цвета.

Для того чтобы убрать все отладочные сообщения с Web-страниц, достаточно удалить параметр `Trace` из объявления Web-страницы, и приложение готово к выполнению.

В том случае, если разработчику необходимо проверять все запросы, выполняемые в ходе работы приложения, следует использовать отладку на уровне приложения. Этот механизм запускается при помощи конфигурационного файла config.Web. В предыдущем разделе мы упоминали блок `<trace>`,

который позволяет задавать параметры отладки уровня приложения. Для того чтобы эта отладка выполнялась, следует указать некоторые параметры у этого тега. Возможный вариант объявления этого раздела может выглядеть следующим образом:

```
<trace
 enabled="true"
 traceMode="SortByCategory"
 requestLimit="40"
 pageOutput="false"
 localOnly="true" />
```

В этом маленьком примере перечислены все используемые параметры раздела `<tracing>`. Рассмотрим их.

- `enabled`. Параметр имеет только два значения — `true` и `false`. В том случае, если используется значение `true`, механизм отладки запускается.
- `pageOutput`. Параметр позволяет разработчику определять, где именно будут выводиться отладочные сообщения. Если используется значение `true`, то они будут отображаться на Web-страницах, если используется значение `false`, то вся отладочная информация будет отображена на отдельной Web-странице, которая носит наименование trace.axd и расположена в корневом каталоге Web-приложения.
- `requestLimit`. Значением этого параметра является целое положительное число. Оно устанавливает максимальное количество запросов к серверу, отладочная информация о которых будет отображена на страницах приложения.
- `traceMode`. Параметр задает порядок сортировки отладочных сообщений. Если используется значение `SortByTime`, то все сообщения будут отображены в хронологическом порядке, если же разработчик укажет значение `SortByCategory`, то сообщения будут сгруппированы по категориям.
- `localOnly`. Параметр имеет только два значения — `true` и `false`. В том случае, если разработчик установил значение `true`, обрабатываются только локальные запросы.

Если мы используем именно ту комбинацию параметров, которая была приведена в примере, то на Web-страницах приложения отладочная информация не будет отображаться. Для того чтобы получить к ней доступ, необходимо просмотреть в браузере файл с наименованием trace.axd. Внешний вид браузера с этим загруженным файлом показан на рис. 3.33.

Видно, что данная страница показывает краткую информацию об одном запросе. Впрочем, если необходимо получить более детальную информацию о каком-либо запросе, следует воспользоваться гиперссылкой, расположенной в правой части таблицы, напротив наименования искомого запроса.

В этом случае разработчику будет предоставлена самая детальная отладочная информация.

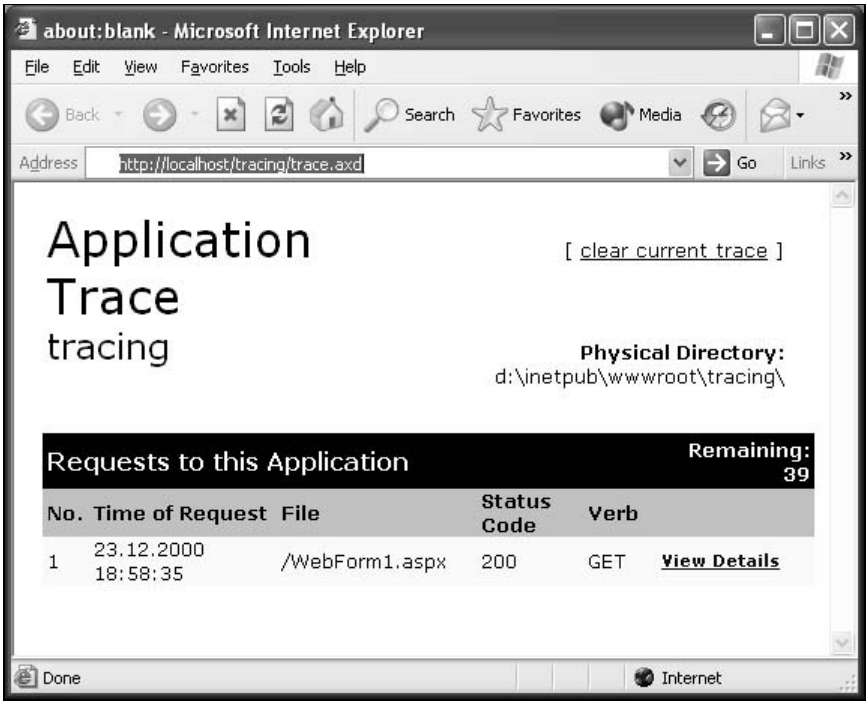
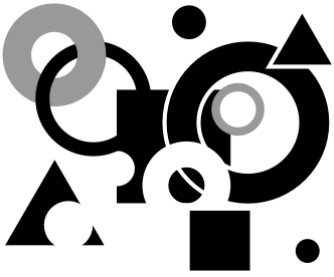


Рис. 3.33. Внешний вид Web-страницы trace.axd в браузере

На этом мы заканчиваем рассмотрение вопросов отладки приложений ASP.NET, а также и всю главу в целом. В следующей главе мы рассмотрим вопросы интеграции приложений ASP.NET с базами данных.

Глава 4



Взаимодействие с базами данных

Постановка задачи

Естественно, при разработке сколько-нибудь серьезного сайта нельзя обойтись без системы управления базами данных. Для сайтов, созданных с помощью технологии ASP.NET, самым естественным выбором, пожалуй, будет сервер MS SQL Server 2000. На самом деле можно использовать практически любую СУБД, для которой в системе установлен ODBC-драйвер, но необходимо осознавать, что сервер MS SQL Server 2000 создан Microsoft, как и технология Microsoft .NET, поэтому связь между Web-приложениями и базами данных, функционирующими под управлением MS SQL Server 2000, будет практически прозрачна.

Установка SQL Server 2000 в конфигурации Enterprise на платформы Windows 2000 или Windows XP достаточно тривиальна и не вызывает никаких проблем. После завершения программы инсталляции в вашем меню **Programs** появится одна или несколько (в зависимости от выбранного комплекта утилит) новых групп программ, а сам SQL Server 2000 успешно стартует, о чем будет сигнализировать соответствующий значок.

В рамках этой главы я не буду рассказывать о настройке и администрировании SQL-сервера, а также о синтаксисе языка SQL. Об этом можно (и нужно) писать отдельные книги. Любой Web-разработчик, если он работает с серверами баз данных, должен представлять хотя бы общий стандарт языка SQL. Поэтому при рассмотрении материала данной главы автор будет исходить из предположения, что с основным синтаксисом SQL читатель знаком.

Естественно, рассмотрение вопросов взаимодействия Web-приложений лучше всего проводить на примере. Чаще всего в литературе рассматривают примеры создания интернет-магазинов. Попробуем взять что-либо не столь распространенное. Например, сетевое агентство знакомств.

Потребуется создать сайт с возможностями поиска по базе данных, а также предусмотреть механизм ввода пользователями информации о себе. Естественно, это будет всего лишь учебный проект, поэтому будем использовать "облегченные" варианты баз данных. А теперь перейдем к непосредственной работе.

Создание базы данных

Прежде всего, необходимо создать саму базу данных, чтобы потом ее можно было подключить к создаваемому Web-приложению. Для этого следует запустить утилиту Enterprise Manager. Внешний вид ее основного окна показан на рис. 4.1.

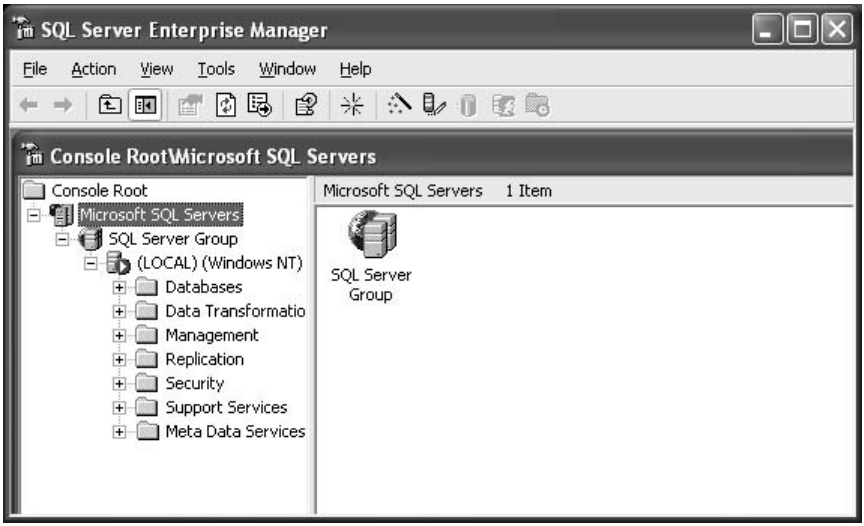


Рис. 4.1. Основное рабочее окно утилиты SQL Server Enterprise Manager

В левой части этой утилиты располагается дерево управления SQL-сервером. Необходимо раскрыть папку **Databases**. Для создания новой базы данных можно воспользоваться встроенным мастером SQL Server 2000, вызов списка встроенных мастеров производится либо нажатием кнопки **Run a Wizard** (Запустить мастер), располагающейся на основной инструментальной панели утилиты управления сервером, либо при помощи команды меню **Tools | Wizards** (Сервис | Мастера). При этом будет отображено диалоговое окно **Select Wizard** (Выбрать мастер), внешний вид которого показан на рис. 4.2.

В списке поставляемых мастеров нас будет интересовать мастер с наименованием **Create Database Wizard** (Мастер создания базы данных). Он позволяет создавать базы данных в визуальном режиме, без объявления их структуры при помощи SQL-выражения.

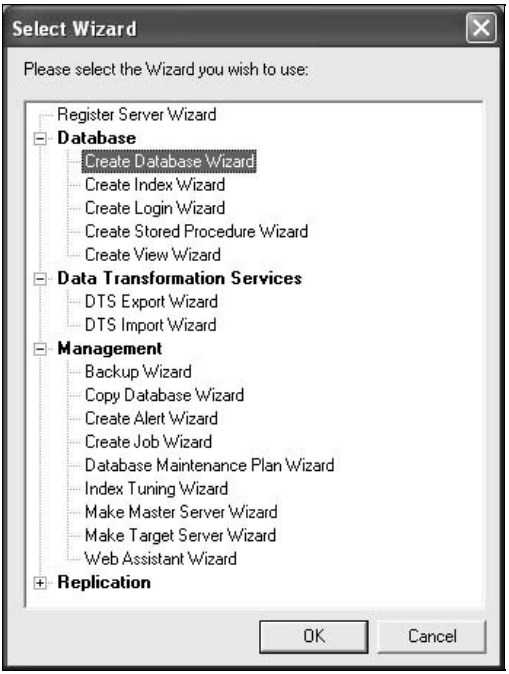


Рис. 4.2. Диалоговое окно **Select Wizard**

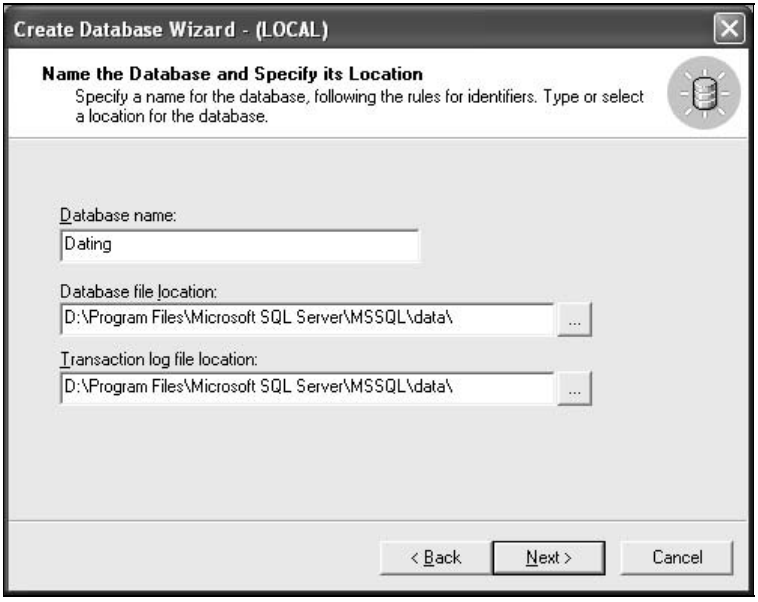


Рис. 4.3. Диалоговое окно первого этапа работы мастера создания новой базы данных

После запуска выбранного мастера, будет отображено диалоговое окно первого этапа его работы. Для начала необходимо будет ввести наименование создаваемой базы данных и расположение файлов самой базы данных и log-файла всех транзакций в файловой системе сервера. Внешний вид диалогового окна первого этапа работы мастера показан на рис. 4.3.

Для новой базы данных мы укажем наименование `Dating`, введя его в текстовое поле **Database Name** (Имя базы данных). 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, используя для того кнопку **Next** (Далее). Диалоговое окно второго этапа работы мастера создания новой базы данных показано на рис. 4.4.

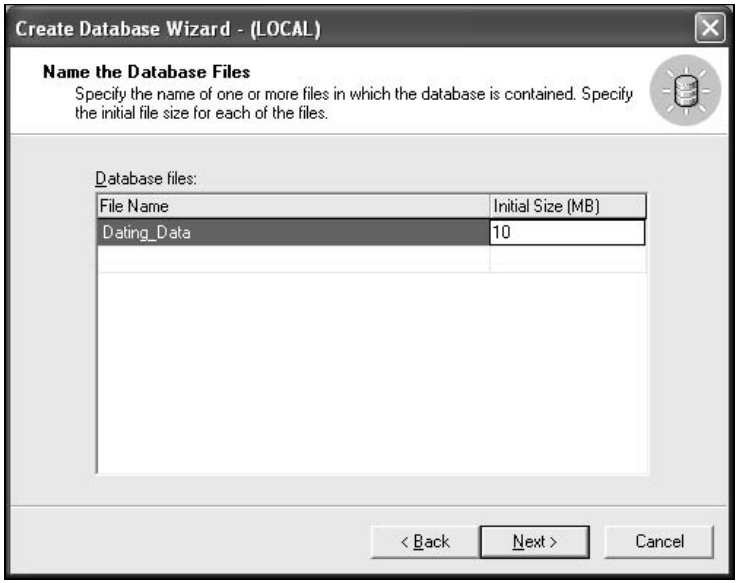


Рис. 4.4. Диалоговое окно второго этапа работы мастера создания новой базы данных

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. Можно, конечно, оставить этот размер, предлагаемый по умолчанию, ничего страшного не произойдет. Когда размер файла превысит указанный предел, операционная система выделит для него дополнительное пространство на логическом диске. Однако можно с достаточно большой степенью уверен-

ности предположить, что физически место, располагающееся на диске сразу за файлом с данными, будет занято другой информацией, и к искомому файлу будет добавлена цепочка кластеров, находящаяся на некотором удалении от него. Таким образом, файл не будет располагаться в одной последовательности 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 показаться несущественными. Однако следует осознавать, что при достаточно интенсивной работе с сервером баз данных эти задержки будут накапливаться, и, в конце концов, суммарная задержка может составить некоторую ощутимую величину. А если еще учесть, что в нашем случае на машине еще и активно функционирует Web-сервер, который тоже добавляет задержки при приеме и передаче данных, то 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 этапу работы мастера. Его диалоговое окно показано на рис. 4.5.

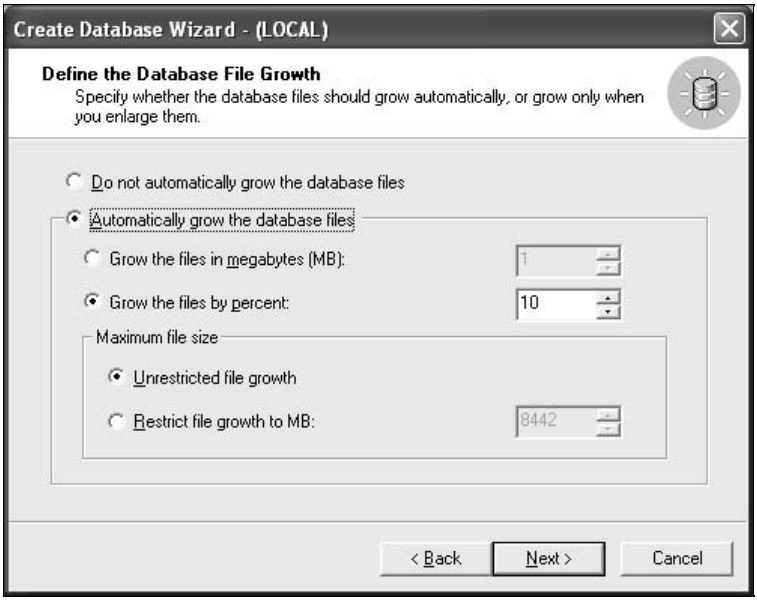


Рис. 4.5. Диалоговое окно третьего этапа работы мастера создания новой базы данных

Это диалоговое окно позволяет администратору указать порядок увеличения размера файла, содержащего базу данных в том случае, когда его размер все-таки превысит пределы, установленные администратором на предыдущем

этапе работы мастера. Для автоматического увеличения размера файла (а это единственный разумный способ управления файлом, так как иначе он просто не будет увеличиваться, и ввод новых данных будет заблокирован) следует выбрать кнопку-переключатель **Automatically grow the database files** (Автоматическое увеличение размера файлов базы данных). После этого останется лишь указать, как именно будет рассчитываться дополнительный объем, выделяемый файлу. Если выбрать кнопку-переключатель **Grow the files in megabytes** (Увеличение размера файлов в мегабайтах), то каждый раз файл будет увеличиваться сразу на несколько мегабайт. Размер постоянного приращения можно задать в поле текстового ввода, связанного с данной кнопкой переключателем.

Также можно указать, что размер приращения будет исчисляться в процентах от размера основного файла. Для этого следует выбрать кнопку **Grow the files by percent** (Увеличение размера файлов в процентах). А в поле, связанном с этой кнопкой переключателем, необходимо задать размер приращения в процентах.

Группа органов управления **Maximum file size** (Максимальный размер файла) позволяет жестко задавать максимально возможный размер файла, содержащего данные. Как мы уже говорили ранее, неразумно задавать конкретный верхний предел размера файла. Поэтому рекомендуется выбирать кнопку **Unrestricted file growth** (Неограниченное возрастание размера файла), которая 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 работы мастера практически идентичны второму и третьему этапу, 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. И после указания этих параметров мастер создания базы данных завершит свою работу.

Итак, база данных создана, но это еще далеко не все. Необходимо создать еще таблицы, входящие в состав этой базы данных. Для этого необходимо открыть базу данных и активировать группу **Tables** (Таблицы). В базе данных уже будет находиться несколько служебных таблиц, но нам необходимо сделать свою, в которой будут размещаться данные. Для этого следует выполнить команду меню **Action | New Table** (Действие | Создать таблицу). При этом будет активировано диалоговое окно **New Table in 'Dating'** (Создать таблицу в 'Dating'), которое показано на рис. 4.6.

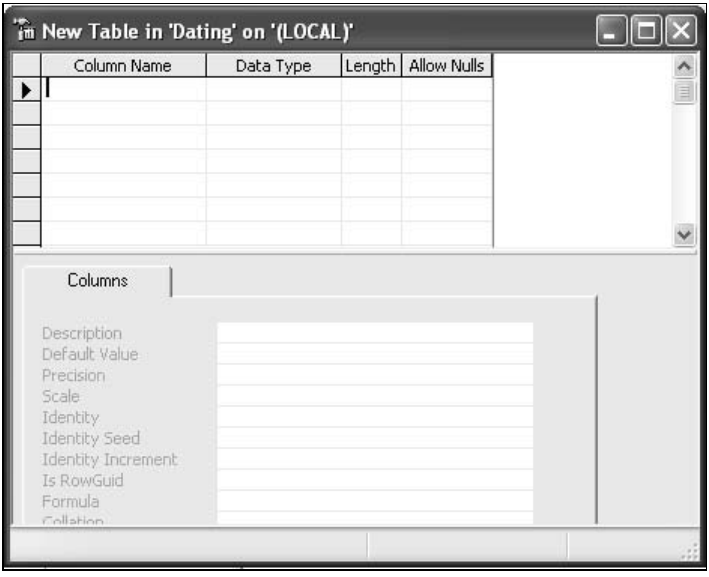


Рис. 4.6. Диалоговое окно создания новой таблицы

Создание новой таблицы может проходить как в визуальном режиме, так и при помощи стандартного SQL-скрипта. Естественно, мы рассматриваем всего лишь тестовый пример, поэтому создадим облегченную версию таблицы, в которой будут храниться поля, содержащие имя посетителя (тип `VarChar`), пол (тип `Int`), адрес электронной почты (тип `Char`), поле, в котором будет храниться информация о пользователе (тип `VarChar`), поле, содержащее возраст пользователя (тип `Int`). SQL-выражение, создающее подобную таблицу, выглядит следующим образом:

```
CREATE TABLE [dbo].[Dating] (
    [USNAME] [varchar] (50) COLLATE Cyrillic_General_CS_AS NULL ,
    [SEX] [int] NULL ,
    [EMAIL] [char] (50) COLLATE Cyrillic_General_CS_AS NULL ,
    [ABOUT] [varchar] (1000) COLLATE Cyrillic_General_CS_AS NULL ,
    [AGE] [int] NULL ,
) ON [PRIMARY]
```

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

Теперь рассмотрим структуру сайта, который будет взаимодействовать с созданной базой данных. Нам потребуется Web-страница для установки критерия поиска человека в общей базе данных, и, соответственно, Web-страница для вывода результатов поиска. Также потребуется и еще одна Web-

страница, при помощи которой пользователи могли бы вводить данные о себе в базу данных. Вот с разработки этой Web-страницы мы и начнем.

Ввод данных с Web-страницы

Для начала нам придется разработать Web-страницу, при помощи которой пользователи смогут ввести информацию о себе в нашу базу данных. Этой Web-странице мы присвоим наименование "insert.aspx". На ней следует разместить поля текстового ввода и одну группу кнопок переключателей, при помощи которой пользователь будет указывать свой пол. Следовало бы также разместить на этой странице и механизм проверки достоверности, для контроля соответствия введенных данных установленным форматам. Иначе говоря, адрес электронной почты должен хотя бы соответствовать правилам написания этих адресов, а возраст должен быть целым положительным числом, находящимся в разумных рамках (допустим от 14 до 85), но не стоит в данном случае так сильно нагружать страницу органами управления. Как их использовать мы уже знаем, поэтому оставим маленький пример без них.

Итак, HTML-код Web-страницы в режиме разработки приведен в листинге 4.1.

Листинг 4.1

```
<%@ Page Language="vb" AutoEventWireup="false" Codebe-
hind="insert.aspx.vb" Inherits="dating.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:Label id="Label1" style="Z-INDEX: 101; LEFT: 49px; POSITION:
absolute; TOP: 20px" runat="server" Width="215px" Height="19px">Введите
персональные данные</asp:Label>
   <asp:Label id="Label2" style="Z-INDEX: 102; LEFT: 49px; POSITION:
absolute; TOP: 49px" runat="server">Имя</asp:Label>
```

```
   <asp:TextBox id="TextBox1" style="Z-INDEX: 103; LEFT: 49px; POSITION:
absolute; TOP: 78px" runat="server" Width="221px"
Height="24px"></asp:TextBox>

   <asp:Label id="Label3" style="Z-INDEX: 104; LEFT: 49px; POSITION:
absolute; TOP: 114px" runat="server">Пол</asp:Label>

   <asp:RadioButtonList id="RadioButtonList1" style="Z-INDEX: 105; LEFT:
49px; POSITION: absolute; TOP: 143px" runat="server">

    <asp:ListItem Value="0" Selected="True">Женский</asp:ListItem>

    <asp:ListItem Value="1">Мужской</asp:ListItem>

   </asp:RadioButtonList>

   <asp:Label id="Label4" style="Z-INDEX: 106; LEFT: 49px; POSITION:
absolute; TOP: 203px" runat="server">Адрес электронной почты</asp:Label>

   <asp:TextBox id="TextBox2" style="Z-INDEX: 107; LEFT: 49px; POSITION:
absolute; TOP: 232px" runat="server"></asp:TextBox>

   <asp:Label id="Label5" style="Z-INDEX: 108; LEFT: 282px; POSITION:
absolute; TOP: 46px" runat="server">Возраст</asp:Label>

   <asp:TextBox id="TextBox3" style="Z-INDEX: 109; LEFT: 284px; POSITION:
absolute; TOP: 79px" runat="server"></asp:TextBox>

   <asp:Label id="Label6" style="Z-INDEX: 110; LEFT: 49px; POSITION:
absolute; TOP: 266px" runat="server">Дополнительная информация</asp:Label>

   <asp:TextBox id="TextBox4" style="Z-INDEX: 111; LEFT: 49px; POSITION:
absolute; TOP: 295px" runat="server" Width="355px" Height="58px"
TextMode="MultiLine"></asp:TextBox>

   <asp:Button id="Button1" style="Z-INDEX: 112; LEFT: 49px; POSITION:
absolute; TOP: 370px" runat="server" Width="130px" Height="24px"
Text="Подтвердить"></asp:Button>

  </form>

 </body>

</HTML>
```

Но эта Web-страница пока еще мертва. Необходимо написать исполняемый код для нее. Идеологически все достаточно просто. Требуется собрать введенные данные, а затем подставить их в SQL-запрос `INSERT`. Еще раз оговорюсь, что в этой главе не будет рассматриваться синтаксис языка SQL, поэтому достаточно будет лишь напомнить, что этот оператор SQL позволяет добавлять данные в таблицы.

Перед тем, как рассматривать механизм соединения с базой данных и выполнения SQL-запросов, приведем листинг кода, реализующего класс создаваемой Web-страницы. Это листинг 4.2.

Листинг 4.2

```
Public Class WebForm1
    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 TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Label3 As System.Web.UI.WebControls.Label
    Protected WithEvents RadioButtonList1 As
    System.Web.UI.WebControls.RadioButtonList
    Protected WithEvents Label4 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Label5 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox3 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Label6 As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox4 As System.Web.UI.WebControls.TextBox
    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 DataConnection As Data.SqlClient.SqlConnection
        DataConnection = New Data.SqlClient.SqlConnection
("server=(LOCAL);uid=sa;pwd=;database=Dating")
        Dim DataCommand As Data.SqlClient.SqlCommand
```

```
        Dim InsertCmd As String = "insert into Dating (USNAME, SEX,
EMAIL, ABOUT, AGE) values (@Usname, @Sex, @Email, @About, @Age)"


        DataCommand = New Data.SqlClient.SqlCommand(InsertCmd,
        DataConnection)
        DataCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@Usname", System.Data.SqlDbType.NVarChar,
50))
        DataCommand.Parameters("@Usname").Value = TextBox1.Text
        DataCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@Sex", System.Data.SqlDbType.Int, 2))
        DataCommand.Parameters("@Sex").Value = RadioButton-
List1.SelectedIndex
        DataCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@Email", System.Data.SqlDbType.NChar, 50))
        DataCommand.Parameters("@Email").Value = TextBox2.Text
        DataCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@About", System.Data.SqlDbType.NVarChar,
1000))
        DataCommand.Parameters("@About").Value = TextBox4.Text
        DataCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@Age", System.Data.SqlDbType.Int, 2))
        DataCommand.Parameters("@Age").Value = Val(TextBox3.Text)
        DataCommand.Connection.Open()
        Try
            DataCommand.ExecuteNonQuery()
            Response.Redirect("Success.aspx")
        Catch Exp As Data.SqlClient.SqlException
            Response.Redirect("failed.aspx")
        End Try
        DataCommand.Connection.Close()


    End Sub
End Class
```

Теперь рассмотрим созданный обработчик нажатия пользователем кнопки. Для работы нам потребуется экземпляр класса `Data.SqlClient.SqlConnection`, который позволяет устанавливать соединение с SQL-сервером, и экземпляр класса `Data.SqlClient.SqlCommand`, позволяющий выполнять SQL-запросы.

После того, как мы объявили экземпляр класса `Data.SqlClient.SqlConnection` с наименованием `DataConnection`, следует его создать. Для этого мы используем конструктор `Data.SqlClient.SqlConnection`, которому

в качестве параметров передается строка, содержащая параметры для установки соединения с базой данных. Из листинга видно, что данная строка состоит из пар "имя=значение", отделенных друг от друга символами точки с запятой. Параметр `server` позволяет указывать наименование SQL-сервера, к которому производится присоединение. Так как в нашем примере мы используем локальный сервер, который регистрируется в системе автоматически при инсталляции Microsoft SQL Server 2000, то мы используем значение "(LOCAL)".

Затем при помощи параметра `uid` мы указываем регистрационное имя, под которым происходит присоединение к серверу. Для этого параметра мы используем значение `sa`, которое указывает на встроенную учетную запись администратора. Пароль для этого регистрационного имени указывается при помощи параметра `pwd`, и в нашем случае этот пароль является пустым. Следует отметить, что подобное подключение к SQL-серверу возможно только в том случае, если тот поддерживает встроенную авторизацию, а не использует авторизацию пользователей, основанную на механизмах идентификации Windows NT. Для того чтобы узнать на какой именно механизм авторизации опирается SQL-сервер, следует в конфигурационной утилите SQL Server Enterprise Manager выделить элемент, соответствующий серверу, и в контекстном меню выполнить команду **Edit SQL Server Registration properties** (Редактор свойств регистрации SQL-сервера). При этом будет активировано диалоговое окно **Registered SQL Server Properties** (Свойства регистрации SQL-сервера), показанное на рис. 4.7. В том случае, если действует собственная аутентификация SQL-сервера, должна быть активирована кнопка-переключатель **Use SQL Server authentication** (Использовать аутентификацию SQL-сервера). Естественно, в этом случае необходимо указать регистрационное имя и пароль для учетной записи, проходящей аутентификацию, в полях **Login Name** (Регистрационное имя) и **Password** (Пароль), соответственно.

Примечание

Следует осознавать, что в реальных условиях не следует использовать учетную запись с именем входа "`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 стоит избавляться от всех учетных записей, устанавливаемых по умолчанию, так как они общеизвестны.

В строке установки соединения также применяется параметр с наименованием `Database`, в качестве значения которого используется наименование базы данных, к которой мы собираемся подключаться. В нашем случае используется значение `Dating`.

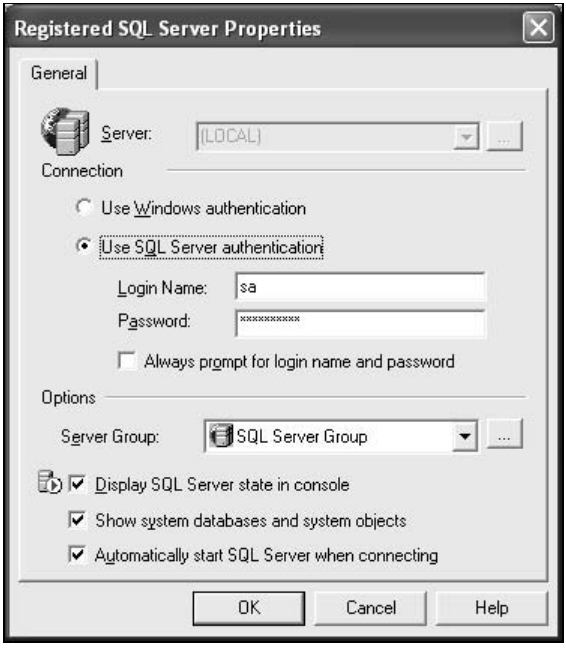


Рис. 4.7. Диалоговое окно **Registered SQL Server Properties**

После того, как было создано соединение с SQL-сервером, следует подготовить для выполнения SQL-запрос. В нашем случае SQL-запрос будет выполнять команду `INSERT`, следовательно, никаких наборов данных от этого запроса мы не получим. Для выполнения запроса будет использован экземпляр класса `SqlCommand`. Однако, для него еще необходимо подготовить саму строку, содержащую команду SQL. Для этого используется переменная `InsertCmd` типа `String`.

В этой строке записывается текст выполняемого SQL-запроса. Так как следует явно указывать значения полей в этой строке, у нас есть два варианта действий. Либо пользуясь данными, введенными пользователем, генерировать эту строку, либо сразу задать текст запроса, но так как нам заранее неизвестны подставляемые значения, воспользоваться параметрами. В нашем примере мы воспользовались вторым способом. При этом строка SQL-запроса приобрела следующий вид:

```
insert into Dating (USNAME, SEX, EMAIL, ABOUT, AGE) values (@Usname,
@Sex, @Email, @About, @Age)
```

Видно, что вместо конкретных значений, сохраняемых в таблице Dating, в строке запроса указаны наименования параметров, начинающиеся с символа `@`. Для доступа к этим параметрам следует использовать коллекцию `Parameters`, входящую в состав класса `SqlCommand`. Для каждого параметра мы используем метод `Add`, который добавляет его в коллекцию. В качестве

параметра этому методу передается конструктор класса `SqlParameter`, который реализует сами параметры SQL-запросов.

Однако перед тем как задать значения этих параметров, следует создать экземпляр класса `SqlCommand`. При этом в качестве параметров передается строка, содержащая текст запроса, и переменная, отвечающая за соединение с SQL-сервером.

В качестве параметров для конструктора класса `SqlParameter` передаются наименование добавляемого параметра, тип данных, которые будут соответствовать этому параметру, и длина поля, как это показано в листинге.

После того, как все параметры добавлены, следует выполнить подготовленный запрос. Для этого сначала следует открыть подготовленное соединение с SQL-сервером. Поэтому используется метод `Open`, применяемый к свойству `Connection`, входящему в состав класса `SqlCommand`. А затем, после того, как соединение открыто, следует выполнить сам SQL-запрос. Для этого мы можем использовать метод `ExecuteNonQuery`. Однако в нашем примере мы не просто вызываем этот метод, а при помощи конструкции `Try … Catch … End Try` подстраховываем себя на случай возникновения каких-либо нештатных ситуаций. В том случае, когда не возникает никаких исключений, пользователь переадресуется на страницу Success.aspx. А если системой было инициировано исключение, пользователь оповещается об этом при помощи переадресации на страницу failed.aspx.

После выполнения запроса следует закрыть соединение при помощи метода `Close`, применяемого к свойству `Connection`, входящему в состав класса `SqlCommand`. На этом работа фрагмента Web-приложения, отвечающего за добавление данных в таблицу, заканчивается. Прежде чем мы перейдем к рассмотрению остальных частей Web-приложения, рассмотрим структуру классов `SqlConnection` и `SqlCommand`.

Объект *SqlConnection*

Объект `SqlConnection` предназначен для установки соединения с SQL-сервером. В предыдущем разделе мы рассмотрели пример использования его конструктора, но это далеко не все, что следует знать об этом объекте. Тот же самый конструктор может применяться с дополнительными атрибутами, входящими в строку, которая объявляет свойства соединения. Впрочем, аналогичная строка может храниться в свойстве `ConnectionString`, с рассмотрения которого мы и начнем обзор класса `SqlConnection`.

- `ConnectionString`. Свойство типа `String`, в котором указываются параметры соединения с SQL-сервером. Строка состоит из пар "имя=значение", отделенных друг от друга символами точки с запятой. Естественно, возможные наименования параметров жестко заданы.

Впрочем, необходимо отметить, что большую часть этих параметров можно задать при помощи отдельных свойств.

- `Application Name`. Параметр позволяет устанавливать наименование приложения, из которого и устанавливается соединение с SQL-сервером.
- `Connect Timeout` или `Connection Timeout`. Значением этого параметра является целое число, указывающее продолжительность периода в секундах, в течение которого приложение будет ожидать ответа от SQL-сервера на свой запрос. Если в течение этого промежутка SQL-сервер не ответит приложению, соединение будет считаться не установленным. По умолчанию используется значение "15".
- `Connection Lifetime`. Это значение позволяет указывать время, в течение которого будет действовать установленное соединение. Значением параметра является целое число, указывающее продолжительность жизни соединения в секундах. По истечении этого промежутка соединение будет принудительно закрыто системой. По умолчанию используется нулевое значение, которое указывает, что соединение имеет неограниченный срок жизни, и его будет закрывать само приложение.
- `Current Language`. Параметр позволяет указывать язык, на котором написано текстовое содержимое открываемой базы данных.
- `Data Source` или `Server`. Параметр позволяет указывать имя SQL-сервера, с которым устанавливается соединение. Если вместо имени необходимо указать сетевой адрес сервера, следует воспользоваться параметрами `Address`, `Addr` или `Network Address`.
- `Database` или `Initial Catalog`. Параметр указывает наименование базы данных, к которой происходит подключение.
- `Trusted_Connection` или `Integrated Security`. Параметр позволяет указывать, что соединение будет устанавливаться по защищенному каналу. В качестве значений могут использоваться ключевые слова `true` или `false`. Если используется значение по умолчанию `false`, соединение устанавливается без соблюдения особых правил безопасности. Впрочем, в тех случаях, когда и SQL-сервер, и приложение находятся на одной и той же машине, нет смысла использовать защищенное соединение.
- `Max Pool Size`. Значением этого параметра является число, указывающее максимальное количество соединений, которое может одновременно поддерживать SQL-сервер. По умолчанию используется значение "100".
- `Min Pool Size`. Значением этого параметра является число, указывающее минимальное количество соединений, которое может одно-

временно поддерживать SQL-сервер. По умолчанию используется нулевое значение.

- `Password` или `Pwd`. Параметр указывает пароль для регистрационного имени, под которым происходит соединение с базой данных.
- `User ID`. Параметр позволяет указывать регистрационное имя, под которым происходит соединение с базой данных.

- `ConnectionTimeout`. Свойство типа `Integer`. Свойство устанавливает продолжительность в секундах периода тайм-аута при соединении с базой данных. По умолчанию используется значение "15".
- `Database`. Свойство типа `String`. В нем указывается наименование базы данных, к которой и присоединяется приложение.
- `DataSource`. Свойство типа `String`. В нем указывается наименование, под которым зарегистрирован SQL-сервер, к которому присоединяется приложение.
- `PacketSize`. Свойство типа `Integer`, в котором указывается размер (в байтах) пакетов информации, пересылаемых от приложения к серверу и обратно. Значение свойства может находиться в пределах от 512 до 32767. По умолчанию используется значение "8192".
- `ServerVersion`. Свойство типа `String`, в котором отображается номер версии сервера, с которым устанавливается соединение.

Теперь перейдем к рассмотрению методов класса `SqlConnection`.

- `BeginTransaction`. Метод вызывает запуск транзакции. Если использовать метод без параметров, то будет применена стандартная транзакция, запускаемая по умолчанию. Однако в качестве параметра можно передать строку, содержащую наименование транзакции. О работе с транзакциями будет более подробно рассказано в одном из следующих разделов этой главы.
- `ChangeDatabase`. Метод позволяет изменить наименование базы данных, к которой при помощи объекта `SqlConnection` было присоединено приложение. В качестве параметра типа `String` методу передается наименование базы данных, к которой будет подключено текущее соединение.
- `Close`. Метод закрывает соединение.
- `CreateCommand`. Метод позволяет создать экземпляр класса `SqlCommand`, который будет связан с открытым соединением.
- `Open`. Метод позволяет открыть соединение. При этом метод использует свойство `ConnectionString` для получения параметров открываемого соединения.

На этом мы заканчиваем рассмотрение структуры класса `SqlConnection` и переходим к рассмотрению класса `SqlCommand`.

Объект *SqlCommand*

Объект `SqlCommand` реализует SQL-запросы в рамках приложений. В одном из предыдущих разделов этой главы мы уже наблюдали пример использования этого объекта, а сейчас пришло время несколько внимательнее рассмотреть его структуру.

Этот объект обладает четырьмя конструкторами. В простейшем случае метод-конструктор не обладает параметрами и основные параметры созданного экземпляра класса придется задавать при помощи свойств. Однако разработчик может сразу при создании экземпляра класса указать строку, содержащую текст SQL-запроса, соединение, в рамках которого будет действовать этот запрос, и транзакцию. Так что программист может выбрать именно тот конструктор, который будет наиболее хорошо удовлетворять его требованиям.

А теперь перейдем к рассмотрению свойств и методов класса `SqlCommand`.

- `CommandText`. Свойство типа `String`, содержащее текст SQL-запроса, связанного с данным экземпляром класса `SqlCommand`.
- `CommandTimeout`. Свойство типа `Integer`, задающее продолжительность периода тайм-аута для искомого запроса в секундах.
- `CommandType`. Свойство указывает, какой именно запрос находится в свойстве `CommandText`, а точнее, тип запроса. Значение должно входить в состав перечислимого типа `CommandType`. Все возможные значения перечислены ниже.
  - `StoredProcedure`. Текст является наименованием хранимой процедуры.
  - `TableDirect`. Текст является наименованием таблицы. Это значение используется только для OLE DB .NET Data Provider.
  - `Text`. Текст обычного SQL-запроса.
- `Connection`. Свойство типа `SqlConnection`. В нем содержится соединение с сервером, при помощи которого и выполняется SQL-запрос.
- `Parameters`. Свойство типа `SqlParameterCollection`, который является коллекцией, обеспечивающей доступ к параметрам SQL-запроса.
- `Transaction`. Свойство типа `SqlTransaction`, содержащее транзакцию, в рамках которой выполняется SQL-запрос.

Теперь перейдем к рассмотрению методов, входящих в состав данного класса.

- `Cancel`. Метод принудительно обрывает выполнение SQL-запроса.
- `CreateParameter`. Метод создает параметр SQL-запроса. В качестве результата своей работы метод возвращает значение типа `SqlParameter`.

- `ExecuteNonQuery`. Метод применяется для выполнения SQL-запросов, которые не возвращают какие-либо наборы данных. Обычно подобные запросы базируются на ключевых словах `Update`, `Insert` или `Delete`.
- `Prepare`. Метод подготавливает SQL-запрос к выполнению. Если один и тот же запрос выполняется несколько раз, стоит воспользоваться этим методом. При "подготовке" запроса, его последующее выполнение производится несколько быстрее.
- `ResetCommandTimeout`. Этот метод сбрасывает продолжительность периода тайм-аута для SQL-запроса до значения, используемого по умолчанию.

Итак, мы рассмотрели структуру объекта `SqlCommand` и можем переходить к рассмотрению иных технологий работы с базой данных.

Поиск и отображение информации

Вернемся к нашему проекту сайта знакомств. Мы уже создали страницу, при помощи которой посетитель сайта может внести в базу данных информацию о себе. Теперь необходимо реализовать поиск по этой базе.

Итак, нам потребуется создать еще две Web-страницы. На одной из них посетитель сайта сможет задать условия поиска по базе данных, а на другой Web-странице ему будет показан результат его запроса.

Естественно, поиск данных будет производиться при помощи SQL-запроса, основанного на ключевом слове `SELECT`. Однако здесь следует сделать некоторое отступление. В предыдущем примере мы использовали SQL-запрос, который не возвращал каких-либо данных. SQL-запрос `SELECT` обязательно будет возвращать некоторое множество записей из основной таблицы. Это множество записей будет являться источником данных для Web-страницы, показывающей результат запроса пользователя. Подобные источники данных в ASP.NET реализуются при помощи типа `DataSet`. Его структуру мы рассмотрим в следующем разделе главы, а в этом разделе мы просто приведем пример его использования.

Итак, для начала необходимо спроектировать Web-страницу, на которой пользователь будет указывать критерии поиска по базе данных. Эту страницу мы назовем search.aspx. Затем, все введенные данные будут переданы Web-странице с наименованием bind.aspx. При загрузке этой страницы будет сформирован запрос `SELECT`, опираясь на критерии пользователя. Затем этот запрос будет выполнен, и его результаты будут отображены в таблице.

Так как наша база данных очень невелика, то и критериев поиска будет немного. Мы предоставим пользователю возможность указать пол пользователей при помощи группы переключателей. В листинге 4.3 приведен HTML-код страницы в режиме ее разработки.

Листинг 4.3

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="search.aspx.vb" Inherits="dating.search"%>
<!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: 41px; POSITION:
absolute; TOP: 20px" runat="server" Height="19px" Width="196px">Укажите
критерии поиска</asp:label>
   <asp:label id="Label2" style="Z-INDEX: 102; LEFT: 41px; POSITION:
absolute; TOP: 51px" runat="server">Пол</asp:label>
   <asp:radiobuttonlist id="RadioButtonList1" style="Z-INDEX: 103; LEFT:
41px; POSITION: absolute; TOP: 81px" runat="server">
    <asp:ListItem Value="0" Selected="True">Мужской</asp:ListItem>
    <asp:ListItem Value="1">Женский</asp:ListItem>
   </asp:radiobuttonlist>
   <asp:button id="Button1" style="Z-INDEX: 109; LEFT: 44px; POSITION:
absolute; TOP: 140px" runat="server" Height="24px" Width="120px"
Text="Искать!"></asp:button>
  </form>
 </body>
</HTML>
```

При нажатии на кнопку **Искать!** пользователь должен быть направлен на другую Web-страницу, которая и будет производить поиск. Поэтому обработчик нажатия кнопки будет достаточно прост. Полный код класса, реализующего созданную Web-страницу, приведен в листинге 4.4.

Листинг 4.4

```
Public Class search
    Inherits System.Web.UI.Page
    Protected WithEvents Label2 As System.Web.UI.WebControls.Label
```

```
    Protected WithEvents RadioButtonList1 As
    System.Web.UI.WebControls.RadioButtonList
    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 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 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
        Session.Add("sex", RadioButtonList1.SelectedIndex)
        Response.Redirect("bind.aspx")
    End Sub
End Class
```

Видно, что при нажатии пользователем кнопки, информация, введенная им, передается на следующую страницу при помощи стандартной коллекции объекта `Session`.

Теперь рассмотрим процедуру создания Web-страницы, которая будет осуществлять поиск и отображение его результатов. Для отображения результатов поиска мы будем использовать компонент `DataGrid`, который позволяет отображать таблицу, связанную с каким-либо набором данных. Этот компонент имеет множество настроек внешнего вида, которые могут устанавливаться в визуальном режиме. Для этого в контекстном меню компонента следует выполнить команду **Property Builder** (Построитель свойств), и будет отображено диалоговое окно, в котором и следует устанавливать свойства. Однако, наша задача сейчас не украсить таблицу, а добиться того, чтобы в ней отображались результаты поиска. Поэтому оставим компонент в том виде, как он создается по умолчанию. В листинге 4.5 показан HTML-код создаваемой Web-страницы в режиме разработки.

Листинг 4.5

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="bind.aspx.vb"
Inherits="dating.bind"%>
<!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:datagrid id="DataGrid1" style="Z-INDEX: 101; LEFT: 28px;
POSITION: absolute; TOP: 22px" runat="server" Height="165px"
Width="348px" ShowHeader="False"></asp:datagrid>
  </form>
 </body>
</HTML>
```

А теперь, после того, как мы увидели HTML-код страницы, следует привести исполняемый код ее класса. Он показан в листинге 4.6.

Листинг 4.6

```
Public Class bind
    Inherits System.Web.UI.Page
```

```
    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()


    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 DS As DataSet
        Dim DataConnection As Data.SqlClient.SqlConnection
        Dim DataCommand As Data.SqlClient.SqlDataAdapter
        Dim SelectCommand As String = "select USNAME, EMAIL, ABOUT, AGE
from Dating where (SEX = @Sex)"
        DataConnection = New
Data.SqlClient.SqlConnection("server=(LOCAL); _
uid=sa;pwd=;database=Dating")
        DataCommand = New Data.SqlClient.SqlDataAdapter(SelectCommand,
DataConnection)
        DataCommand.SelectCommand.Parameters.Add(New
Data.SqlClient.SqlParameter("@Sex", System.Data.SqlDbType.Int, 2))
        DataCommand.SelectCommand.Parameters("@Sex").Value =
        Session.Item("sex")
        DS = New DataSet()
        DataCommand.Fill(DS, "Dating")
        DataGrid1.DataSource = DS
        DataGrid1.DataBind()
    End Sub
End Class
```

Этот код следует детально рассмотреть. Поиск и отображение результатов происходят при загрузке Web-страницы, поэтому мы используем обработчик события `Page_Load`. Также нам потребуется по одному экземпляру трех различных объектов. Мы используем уже знакомый нам объект `SqlConnection` для установки соединения с базой данных, объект `SqlDataAdapter`, который предназначен для выполнения SQL-запроса и получения данных из таблицы, а также объект `DataSet`, который и получит извлеченные данные, а затем послужит источником данных для таблицы. Также в процедуре объявляется переменная `SelectCommand` типа `String`. В этой переменной будет храниться текст SQL-запроса `SELECT`. Следует отметить, что в этом запросе также используется один параметр.

После того, как объявлены все переменные, создается соединение с базой данных при помощи конструктора класса `SqlConnection`. Затем в коллекцию параметров запроса добавляется параметр целочисленного типа. С помощью следующего оператора этому параметру присваивается значение, которое хранилось в элементе коллекции объекта `Session`. После этого выполняется метод `Fill` переменной `DataCommand`. В качестве параметров методу передается переменная типа `DataSet` и наименование таблицы, из которой извлекаются данные. Этот метод заполняет набор данных DS. Осталось только соединить полученный набор данных с таблицей `DataGrid1`.

Соединение набора данных с таблицей происходит при помощи свойства `DataSource`, присущего компоненту `DataGrid`. А после того, как источник данных для таблицы задан, необходимо отобразить данные в ней. Для этого мы используем метод `DataBind`. В результате, после запроса пользователя, все записи из базы данных, удовлетворяющие условию, будут отображены в таблице. Внешний вид этой Web-страницы показан на рис. 4.8.

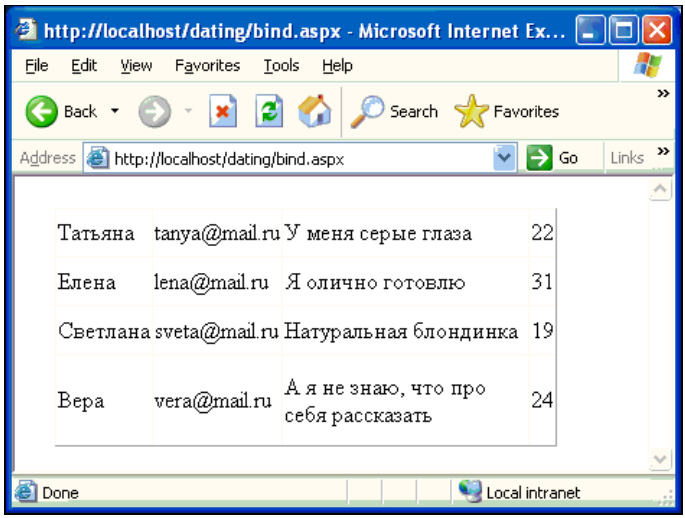


Рис. 4.8. Web-страница с отображением результатов запроса

HTML-код этой страницы приведен в листинге 4.7.

Листинг 4.7

```
<!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 name="Form1" method="post" action="bind.aspx" id="Form1">
   <table cellspacing="0" rules="all" border="1" id="DataGrid1"
style="height:165px;width:348px;border-collapse:collapse;Z-INDEX: 101;
LEFT: 28px; POSITION: absolute; TOP: 22px">
 <tr>
  <td>
   Татьяна
  </td><td>
   tanya@mail.ru
  </td><td>
   У меня серые глаза
  </td><td>
   22
  </td>
 </tr><tr>
  <td>
   Елена
  </td><td>
   lena@mail.ru
  </td><td>
   Я олично готовлю
  </td><td>
   31
  </td>
 </tr><tr>
```

```
  <td>
   Светлана
  </td><td>
   sveta@mail.ru
  </td><td>
   Натуральная блондинка
  </td><td>
   19
  </td>
 </tr><tr>
  <td>
   Вера
  </td><td>
   vera@mail.ru
  </td><td>
   А я не знаю, что про себя рассказать
  </td><td>
   24
  </td>
 </tr>
</table>
  </form>
 </body>
</HTML>
```

В следующем разделе главы мы рассмотрим структуру объекта `DataSet`, который позволяет хранить наборы данных и отображать их на Web-страницах, а пока укажем еще на одну интересную возможность компонента `DataGrid`.

В нашем примере таблица не заняла много места на Web-странице. Но ведь далеко не всегда объемы получаемых данных будут настолько малы, что их можно будет разместить на одной Web-странице. В тех случаях, когда таблица получается слишком большая, ее можно разбить на несколько страниц. Для этого необходимо в режиме разработки страницы выполнить для объекта `DataGrid` команду контекстного меню **Property Builder** (Построитель свойств). На отображенном диалоговом окне следует выбрать вкладку **Paging** (Разбиение на страницы). Внешний вид этой вкладки показан на рис. 4.9.

Для того чтобы таблицу можно было разбивать на несколько частей, каждая из которых отображалась бы на отдельной Web-странице, следует поставить флажок в независимом переключателе **Allow paging** (Разрешить разбиение на страницы). После этого разработчик получает доступ ко всем остальным органам управления, расположенным на этой вкладке.

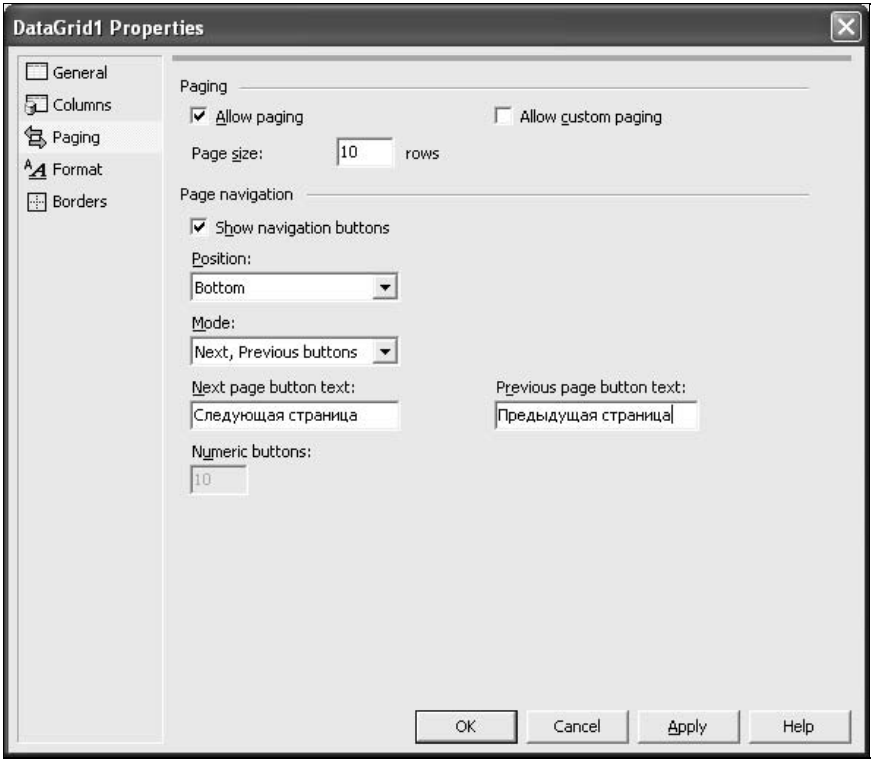


Рис. 4.9. Вкладка **Paging** диалогового окна **DataGrid Properties**

Прежде всего, следует указать, сколько строк таблицы будет размещаться на одной странице. Это количество строк указывается в поле **Page size** (Размер страницы). Для того чтобы пользователь мог перемещаться между страницами, на которых расположена таблица, следует отображать кнопки перехода между страницами. Активация этих кнопок произойдет, если в диалоговом окне отметить флажок в независимом переключателе **Show navigation buttons** (Показать кнопки управления). Расположение кнопок управления между страницами регулируется при помощи пунктов выпадающего списка **Position** (Расположение). А текст надписей на этих кнопках указывается в полях текстового ввода **Next Page Button Text** (Кнопка текста следующей страницы) и **Previous Page Button Text** (Кнопка текста предыдущей страницы).

Итак, мы рассмотрели способы отображения информации из базы данных в таблице. Теперь следует ознакомиться со структурой объекта `SqlDataAdapter`.

Объект *SqlDataAdapter*

Объект `SqlDataAdapter` предназначен для выполнения некоторых команд, которые позволяют заполнять данными объекты типа `DataSet`. В предыду-

щем разделе мы видели в приведенном примере, как при помощи экземпляра класса `SqlDataAdapter` создавался и выполнялся параметрический SQL-запрос, а затем результаты его выполнения перенаправлялись в набор данных `DataSet`. Таким образом, мы можем с полной уверенностью сказать, что в Web-приложениях объект `SqlDataAdapter` чаще всего будет использоваться именно для связи SQL-запросов как команда с отображаемыми результатами этих запросов. А теперь начнем рассмотрение свойств структуры класса `SqlDataAdapter`.

- `DeleteCommand`. Свойство типа `SqlCommand`, которое мы рассматривали в этой главе несколько ранее. В этом свойстве хранится SQL-запрос, основанный на ключевом слове `DELETE`, т. е. предназначенный для удаления одной или нескольких записей из базы данных.
- `InsertCommand`. Свойство типа `SqlCommand`. В нем хранится SQL-запрос, основанный на ключевом слове `INSERT`, т. е. предназначенный для добавления записи в базу данных.
- `SelectCommand`. Свойство типа `SqlCommand`. В этом свойстве находится стандартный SQL-запрос выбора информации из базы данных, основанный, естественно, на ключевом слове `SELECT`.
- `UpdateCommand`. Свойство типа `SqlCommand`. В нем хранится SQL-запрос, основанный на ключевом слове `UPDATE`, т. е. предназначенный для изменения одной или нескольких записей в базе данных.

Как видно, основные свойства рассматриваемого объекта `SqlDataAdapter` предназначены для хранения различных SQL-запросов. Другими словами, одному экземпляру объекта может соответствовать несколько SQL-запросов различного типа, каждый из которых может быть применен в зависимости от ситуации. Но для выполнения SQL-запросов следует использовать все-таки методы объекта. Рассмотрим и их.

- `Fill`. Это один из наиболее часто используемых методов объекта `SqlDataAdapter`. Предназначен для заполнения набора данных `DataSet` или объектов со схожей функциональностью. Метод существует в различных модификациях, отличающихся друг от друга только набором передаваемых параметров. Чаще всего в качестве параметра передается либо экземпляр объекта `DataSet`, либо комбинация этого экземпляра и строки, в которой указывается наименование таблицы, из которой извлекаются данные для занесения в экземпляр `DataSet`. При этом используется SQL-запрос, хранящийся в свойстве `SelectCommand`.
- `Update`. Метод применяется для выполнения SQL-запросов, хранящихся в свойствах `DeleteCommand`, `InsertCommand` и `UpdateCommand`. В качестве параметров методу обычно передается либо экземпляр `DataSet`, либо таблица, к которой будут применяться эти SQL-выражения. Естественно, если какое-либо из перечисленных свойств экземпляра класса

SqlDataAdapter не будет заполнено, то и соответствующий этому свойству SQL-запрос не будет выполняться.

Итак, при помощи этих двух методов мы можем выполнить любой SQL-запрос и применить его к экземпляру класса DataSet. А уже экземпляр класса DataSet мы можем использовать в качестве источника данных для компонентов Web Forms, размещаемых на разрабатываемых Web-страницах.

Таким образом, разработчик получает в свое распоряжение стройную иерархию объектов, которые позволяют устанавливать соединение с базой данных, извлекать данные из нее и модифицировать саму базу данных, а потом сделанную выборку отображать на Web-страницах.

К достоинствам этой методики работы следует отнести тот факт, что соединение с SQL-сервером устанавливается только на то время, которое необходимо для выполнения операции, а затем оно закрывается. Несмотря на то, что постоянное открытие и закрытие соединений несколько замедляет работу сервера, следует признать, что поддержание постоянных соединений забирает все-таки больше ресурсов.

Использование рассмотренных нами компонентов в идеологии Microsoft называется ADO.NET. ADO расшифровывается как AciveX Data Object. Рассмотрим эту концепцию в следующем разделе главы.

ADO.NET

После того, как на протяжении целой главы рассматривалась некая технология работы с базами данных, попробуем подвести под нее теоретическую базу. Как было сказано в предыдущем разделе, использованная нами технология носит наименование ADO.NET. Она является наследником технологии ADO (ActiveX Data Object). Если в предыдущей версии ADO упор делался на создании постоянных соединений, то в ADO.NET, ориентированной, как это видно из названия, на работу в сетях, по умолчанию создаются соединения, не поддерживаемые в подключенном состоянии все время.

Как мы уже знаем, одним из объектов ADO.NET, наиболее близким к компонентам Web Forms является DataSet. Объект DataSet может хранить в себе не только набор записей, а различные таблицы, связи их друг с другом, отдельные поля и прочую атрибутику баз данных. При этом DataSet не взаимодействует напрямую с источником данных. В рассмотренных примерах было показано, как для заполнения объектов DataSet были использованы объекты промежуточного звена. Одним из таких объектов являлся класс SqlDataAdapter.

Следует также отметить, что в качестве источника данных для объекта DataSet могут быть использованы не только стандартные базы данных, но и XML-файлы. Однако, в наших примерах, приведенных в данной главе, мы

использовали в качестве основного SQL-сервера Microsoft SQL Server 2000. Но Web-приложения, созданные на базе ASP.NET, могут использовать любую базу данных, для которой есть соответствующий ODBC-драйвер.

И вот 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 7.0 и Microsoft SQL Server 2000. Для всех остальных баз данных используется источник данных ADO Managed Provider.

Следует отметить, что источник данных SQL Managed Provider обеспечивает несколько большую 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 все 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, располагаются в пространствах имен System.Data и System.Data.ADO. Таким образом, если разработчику необходимо использовать 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-страниц с базой данных используются три компонента — `Connection`, `Command` и `DataSet`. Эта триада практически идеально подходит для использования в Web-приложениях, которые логически разделены на несколько слабосвязанных частей (каждая часть соответствует одной Web-странице), когда нельзя заранее предсказать, какое именно действие произведет пользователь и какой блок данных потребуется ему. Поэтому модель краткосрочных соединений и иерархия Соединение — Команда — Набор данных позволяют при наименьших затратах ресурсов добиться максимальной производительности и свободы действий для разработчика.

Итак, в этой главе мы рассмотрели вопросы работы Web-приложений ASP.NET с базами данных. Несомненно, рассмотренные нами примеры охватывают лишь начальные этапы работы, так как функциональность объектов ADO.NET позволяет создавать и обрабатывать гораздо более сложные структуры данных, однако в рамках книги мы свою задачу выполнили и можем двигаться дальше. На очереди — извлечение данных из XML-файлов.

Извлечение данных из XML-файлов

О самом языке XML будет достаточно подробно рассказано в следующей главе. Однако в процессе изложения материала мы уже сталкивались с файлами XML, поэтому сможем использовать их в достаточно простом примере.

Итак, как нам уже известно, файлы XML являются простыми текстовыми файлами, имеющими такую же структуру, как и HTML-файлы, но при этом набор используемых тегов определяет сам разработчик. Естественно, подобная структура позволяет хранить в XML-файлах не только структурированные тексты, но и информацию из баз данных. Естественно, XML никогда не сможет заменить стандартные механизмы систем управления базами данных, однако при использовании определенных выборок данных для Web эти файлы могут быть полезны. Достаточно всего лишь настроить параметры отображения XML-файлов в браузере пользователя, и можно передавать ему весь файл как есть, без дополнительной обработки.

Впрочем, иногда дополнительная обработка может понадобиться, но стоит вспомнить, что технология Microsoft .NET очень плотно связана с XML, поэтому в ней присутствуют средства достаточно прозрачной работы с XML-файлами. В рамках этого раздела мы, естественно, не сможем рассмотреть всю технологию использования XML в проектах ASP.NET, для этого пришлось бы писать отдельную книгу, но мы рассмотрим механизм использования XML-файла в качестве источника данных для таблицы, отображаемой на Web-странице.

Для начала нам придется создать простейший XML-файл, содержащий какую-либо информацию. Код этого файла приведен в листинге 4.8.

Листинг 4.8

```
<Document>
    <Row>
        <Column1>1</Column1>
        <Column2>2</Column2>
        <Column3>3</Column3>
        <Column4>4</Column4>
    </Row>
    <Row>
        <Column1>5</Column1>
        <Column2>6</Column2>
        <Column3>7</Column3>
        <Column4>8</Column4>
    </Row>
</Document>
```

Как видно, структура и содержимое XML-файла максимально прозрачны. Для простоты доступа этот файл стоит разместить в том же каталоге, в котором будет функционировать создаваемое Web-приложение. Впрочем, это не критично.

Теперь перейдем к созданию самого Web-приложения. На разрабатываемой Web-странице достаточно будет разместить один компонент `DataGrid`. HTML-код этой Web-страницы в режиме разработки приведен в листинге 4.9.

Листинг 4.9

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.spx.vb"
Inherits="dataxml.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:DataGrid id="DataGrid1" style="Z-INDEX: 101; LEFT: 30px;
POSITION: absolute; TOP: 13px" runat="server" Width="282px"
Height="162px"></asp:DataGrid>
  </form>
 </body>
</HTML>
```

Естественно, это не самый сложный листинг, который приходилось нам рассматривать в этой книге. Но вот код класса, который будет считывать данные из XML-файла и помещать их в таблицу, будет несколько сложнее. Этот код приведен в листинге 4.10.

Листинг 4.10

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    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()


    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 DS As New 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)
        DS.ReadXml(Reader)
        FS.Close()
        Dim Source As DataView
        Source = New DataView(DS.Tables(0))
        DataGrid1.DataSource = Source
        DataGrid1.DataBind()
    End Sub
End Class
```

Как видно, все операции будут выполняться в обработчике загрузки таблицы. 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`. Затем, мы объявляем переменную типа `DataView`, которая обраща-

ется к начальному элементу коллекции `Tables` из созданного набора данных. Как мы уже знаем, в объектах `DataSet` может храниться несколько таблиц сразу, но в данном случае, там точно всего одна таблица, поэтому можно смело применить начальный и единственный элемент коллекции `Tables`. А затем эту извлеченную таблицу мы используем в качестве источника данных для компонента `DataGrid1` и отдаем команду на отображение полученных данных при помощи метода `DataBind`. В результате этих действий в браузере отображается Web-страница, внешний вид которой показан на рис. 4.10.

Следует обратить внимание на то, с какой легкостью мы подключили XML-файл к таблице без какого-либо описания данных. Вся прелесть механизма ASP.NET в том, что он автоматически распознает структуру корректных XML-файлов и может отлично их обрабатывать.

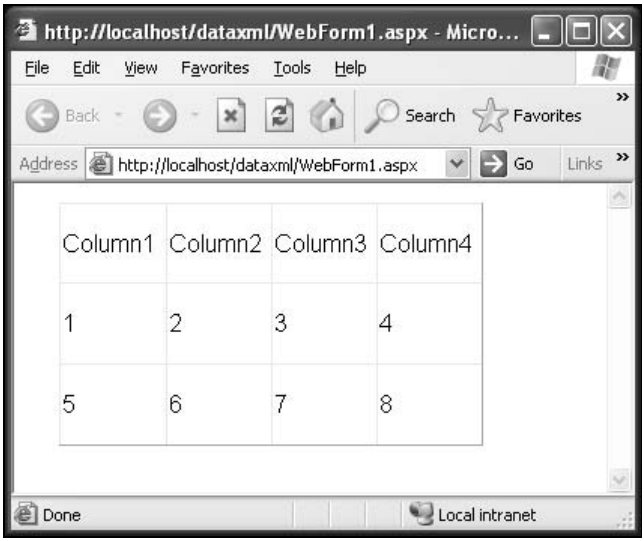
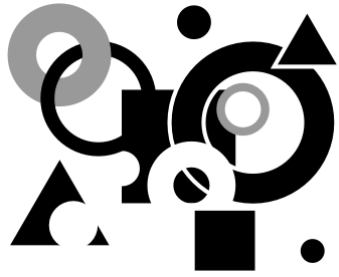


Рис. 4.10. Внешний вид Web-страницы, на которой отображаются данные из XML-файла

Итак, на этом мы заканчиваем рассмотрение работы с данными для Web-приложений в рамках технологии ASP.NET. Настало время несколько внимательнее рассмотреть язык XML. Этому и будет посвящена следующая глава.

Глава 5



XML

История создания

Основой WWW является HTML (HyperText Markup Language). Этот язык представляет собой набор тегов, которые позволяют создавать *разметку* документа, т. е. помимо указания содержимого документа, мы можем при помощи тегов HTML управлять отображением этого документа. Технология проста. Браузер получает HTML-документ и анализирует его. Как только в коде документа встречается какой-либо тег, он распознается, и фрагмент документа, к которому относился тег, оформляется соответствующим образом.

На данный момент доступна уже четвертая версия стандарта языка HTML. Первоначального набора тегов не хватало для нормальной работы. Поэтому компании Microsoft и Netscape, владельцы двух наиболее популярных браузеров дополняли наборы тегов, распознаваемых их браузерами. В силу конкуренции этих двух платформ, указанные дополнения, естественно, не совпадали. Поэтому разработчики либо не использовали дополнительных возможностей, либо отказывались адекватно отображать свой документ в любом браузере. Создавался документ, предназначенный для просмотра в конкретном браузере, а часть посетителей сайта, использующих иной браузер, не могли увидеть данный HTML-документ. Проблемой HTML стала его врожденная ограниченность. Даже самый большой список тегов не в состоянии полностью удовлетворить запросы создателей документов просто в силу того, что этот список ограничен.

Еще одной проблемой стала излишняя популярность WWW. На основе этой технологии стали строить даже локальные сети, которые, соответственно, получили название *интрасетей*. Все корпоративные документы переводились в HTML-формат. Общение внутри сети происходило при помощи электронной почты, создавались сайты, не имеющие выхода в Интернет, служащие хранилищем для корпоративных документов и средством совместной деятельности рабочих групп. Но HTML не мог служить интерфейсом

между пользователями и данными. HTML-документы ориентированы прежде всего на отображение, а не на автоматическую обработку. Из базы данных можно перевести данные в HTML-документ. Обратная операция намного сложнее.

Исходя из этого, стало ясно, какими свойствами должен обладать преемник HTML. Требовалось, чтобы новый язык был расширяемым, чтобы он не был зависим от конкретного набора тегов. Требовалась возможность расширять язык автоматически, исходя из нужд пользователя. Также было необходимо создавать файлы, которые могли бы не только отображаться, но и обрабатываться сторонними приложениями. Другими словами, структура документов должна была напрямую привязываться к ним. Каждый документ должен содержать как информацию, так и ее структуру.

Исходя из этих и многих других соображений Консорциум World Wide Web (W3C) в 1998 году приступил к разработке спецификаций нового языка, который должен был прийти на смену HTML. Его назвали XML (eXtensible Markup Language). В этой книге мы рассмотрим эту спецификацию версии 2.0, воспользовавшись документацией самого W3C.

Корни XML

Прежде всего, давайте разберемся, откуда появился XML. Очень давно (по меркам компьютерной индустрии, естественно), в 1986 году организацией ISO (International Organization for Standardization) язык SGML (Standard Generalized Markup Language) был принят в качестве официального стандарта. А использоваться он начал еще раньше. SGML применяется в качестве стандарта до сих пор. Этот язык описывает практически все возможности разметки информации. Язык SGML позволяет разработчику создавать свои конструкции разметки. Потрясающая гибкость и объем, охватывающий практически все случаи, возникающие при работе, казалось бы, делали этот язык идеальным кандидатом для принятия его в качестве основного языка Web, но существовали и другие обстоятельства, которые помешали ему занять это место.

Любые документы из Web прежде всего просматриваются при помощи специализированных программ — браузеров. Браузеры анализируют код полученного документа, и на основе инструкций разметки, называемых также *тегами*, отображают полученный документ в окне просмотра соответствующим образом. Но спецификация SGML занимает более 500 страниц. Сколько же труда потребовалось бы, чтобы заставить браузеры правильно отображать документы, написанные на SGML. Требовалось что-то более компактное. Поэтому и был создан язык HTML, являющийся очень ограниченным и нерасширяемым подмножеством SGML. Так как набор тегов HTML был невелик, браузеры писать было достаточно легко.

Но потом это достоинство HTML превратилось в его недостаток. Посетителям и владельцам сайтов хотелось получать от HTML все больше и больше. Компании — участники "браузерных войн" добавляли все новые теги во множество распознаваемых своими браузерами инструкций. Основная проблема была в том, что эти добавленные теги у различных компаний тоже были разными. Возникли проблемы с совместимостью. Принятие стандарта HTML версии 4.0 не решило проблемы. Так как HTML представляет собой ограниченный и нерасширяемый набор тегов, то рано или поздно любая версия стандарта окажется недостаточной.

На смену пришел стандарт (а точнее, рекомендация к стандарту) XML (eXtensible Markup Language). Это *расширяемый* язык разметки. Набор тегов XML много меньше по объему, чем у HTML, но в данном случае это неважно. Изменилась сама парадигма создания документов. Теперь теги XML используются в качестве строительных блоков конструктора Lego. Мы теперь можем при помощи тегов XML создавать свой язык для каждого типа документов или даже для каждого документа отдельно.

Подобная гибкость языка позволяет практически прозрачно стыковать документы с различными источниками данных для них. При помощи XML стало максимально легко интегрировать данные из различных приложений. А ведь именно интеграции различной информации из разнородных приложений подчас не хватает настоящим рабочим, а не презентационным, корпоративным сайтам. XML подоспел вовремя.

XML, как и его предшественник HTML, является подмножеством XGML. Но XML является достаточно компактным языком. Поэтому реализация браузеров для XML-документов не будет являться излишне сложной задачей.

Структура XML-документов

В XML-документах можно выделить две основные части. Первая часть XML-документа предназначена для описания структуры, а во второй находится сам контент документа. В описании структуры документа мы можем использовать инструкции для XML-процессора, объявление элементов структуры документа, атрибуты для каждого элемента и так называемые сущности. Каждую из этих структурных единиц мы рассмотрим отдельно и подробно.

Описание структуры документа называют DTD-блоком (Document Type Definition). В нем мы указываем древовидную структуру элементов документа. Наиболее близкой аналогией для этих элементов могут служить объекты. Как и объекты, наши элементы разметки могут иметь свойства, которые в данном случае называются атрибутами.

Как и в любой объектной иерархии, в XML-документе существует некий корневой элемент, от которого наследуются все остальные элементы.

Содержимое XML-документа форматируется при помощи тегов, которые определяются в описании типа документа. Наименования тегов полностью совпадают с наименованиями элементов. А параметры тегов позволяют устанавливать значения атрибутов элементов.

Инструкции XML-процессора

В качестве первой строки каждого XML-документа должна использоваться исполняемая инструкция для XML-процессора. В своем минимально применимом виде она выглядит следующим образом:

```
<?xml version="1.0"?>
```

Как видно из примера, исполняемые инструкции ограничиваются угловыми скобками с вопросительными знаками. Ключевым словом является конструкция `xml`. Следом за ней указываются параметры инструкции и соответствующие им значения. Иногда эти стартовые строки инструкций называют прологом XML-документа.

Приведенный пример предназначен для указания браузеру, что данный документ написан на XML. Параметр `version` с приведенным значением указывает на то, что будет использоваться первая версия стандарта XML. А другой версии у нас пока и нет.

В этих инструкциях мы можем не только указывать номер версии стандарта. Для XML-документов заготовлено несколько видов кодировок, состоящих из символов набора Unicode. Большинство кодировок предложено международной организацией по стандартизации (ISO).

Для указания конкретной кодировки, которая будет использоваться для создания содержимого документа, применяется параметр `encoding`. Это показано в следующем примере:

```
<?xml encoding="UTF-8"?>
```

В качестве значения параметра используется текстовая строка `"UTF-8"`. Кодировка UTF-8 является одной из наиболее часто используемых кодировок.

Следующие параметры инструкций XML-процессора напрямую связаны с обработкой блоков определения типа документа. Забегая немного вперед, необходимо сказать, что подобные блоки могут внедряться в сам XML-документ или находиться отдельно от того документа, к которому они относятся. Для того чтобы XML-процессор смог правильно их обработать, применяется параметр `standalone`. При помощи этого параметра можно указать, где именно находится блок определения типа документа (DTD-блок) для данного XML-документа. В следующем примере мы используем объявление XML-документа, в котором указывается, что DTD-блок для него сделан в виде отдельного файла.

```
<?xml standalone='no'?>
```

У параметра `standalone` есть два предопределенных значения: `yes` и `no`. Значение `no`, как мы видели, извещает XML-процессор, что для данного документа DTD-блок выделен в отдельный файл. Значение `yes` указывает на то, что DTD-блок внедрен в сам XML-файл.

Мы рассмотрели возможные варианты создания исполняемых инструкций, помещаемых в начале документа. На самом деле, эти инструкции могут находиться не только в начале документа, но и в любом другом его месте. Но использование исполняемой инструкции в качестве первой строки XML-документа является обязательным условием.

Любое рассмотрение языка на примерах, без четких определений, будет всего лишь обзором. Чтобы избежать этой участи, для каждой из рассматриваемых нами структурных единиц XML-документа мы будем приводить их определение из спецификации XML. Для исполняемых инструкций, помещаемых в пролог документа, это выглядит следующим образом:

```
[23]  XMLDecl ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24]  VersionInfo ::= S 'version' Eq (' VersionNum ' | " VersionNum ")
[25]  Eq ::= S? '='S?
[26]  VersionNum ::=  ([a-zA-Z0-9_.:] | '-')+
```

На первый взгляд смотрится это все достаточно устрашающе. Все спецификации XML выглядят подобным образом. Это запись правил XML в форме Бэкуса-Наура EBNF (Extended Backus-Naur Form). Разберемся с правилами чтения деклараций в форме EBNF. В левой части каждой декларации указывается имя конструкции, затем следует оператор эквивалентности (`::=`), а в правой части следует расшифровка имени, которая содержит его формат и правила оформления. Перед каждой спецификацией в квадратных скобках указывается номер строки.

Итак, из приведенного фрагмента мы видим, что определение исполняемой инструкции находится в 23-й строке спецификации языка XML. При чтении спецификаций необходимо помнить несколько правил записи определений в форме EBNF.

Если в определении записано несколько терминов, разделенных вертикальной чертой (`|`), то необходимо выбрать только один из них. Подобным образом определяется перечислимое множество компонентов.

Для указания диапазона символов, которые могут использоваться в каком-либо месте определения термина, этот диапазон символов помещается в квадратные скобки.

Круглые скобки ограничивают так называемые регулярные выражения. Проще всего их воспринимать как обычное средство группировки элементов. Когда согласно синтаксису вместо одиночного литерала необходимо указать несколько, используются круглые скобки, группирующие их.

Знак звездочки (*) устанавливается после некоего символьного выражения, если необходимо, чтобы это выражение в строке могло встречаться несколько раз или не встречаться там вообще. Другими словами, символьное выражение, после которого установлен модификатор звездочки, может находиться в результирующей строке ноль или более раз.

Знак плюса (+) применяется в качестве модификатора символьного выражения подобно знаку звездочки. Но есть некоторое отличие. Символьные выражения, после которых установлен этот модификатор, могут встречаться в результирующей строке один или более раз.

Вопросительный знак (?) используется для указания, что тот или иной элемент могут не находиться в результирующей строке, т. е. при помощи этого модификатора указываются опциональные элементы.

Если в выражении не могут встречаться некоторые символы, то вставляется последовательность [^, после которой указываются исключаемые символы. Например, правило [^&] исключает символ амперсанта.

Если внутри какого-либо литерала необходимо вставить пробел, то этот пробел обозначается при помощи символа S.

Любая последовательность символов, заключенная в двойные или одиночные кавычки является литералом, и в результирующей строке должна появляться именно в том виде, в котором она указана в декларации.

Зная эти правила, рассмотрим наши декларации. Начнем, естественно, с декларации [23]. В ней объявляется элемент XMLDecl, который является объявлением XML-документа. После оператора эквивалентности находится литерал '<?xml'. Это означает, что строка объявления XML-документа должна начаться с открывающей угловой скобки, вопросительного знака и последовательности символов xml. Впрочем, этот факт мы знали и до разбора синтаксиса декларации. После этой последовательности символов находится элемент VersionInfo, указывающий номер версии применяемого стандарта XML. Помимо этого в объявлении могут находиться элементы EncodingDecl и SDDecl. После них может находиться или не находиться пробел, и завершается конструкция последовательностью символов '?>'.

В приведенном нами фрагменте спецификации XML не приведено определение конструкции SDDecl. Приведем его сейчас.

```
[32]  SDDecl ::=  S 'standalone' Eq (("'" ('yes' | 'no') "'" |
('"' ('yes' | 'no') '"'))
```

Как видно из этой декларации, значения параметра standalone могут заключаться как в двойные, так и в одиночные кавычки.

Объявление типа документа

Сразу после исполняющей инструкции, указывающей на то, что данный документ создан с применением языка XML, в документе помещается объ-

явление типа документа, называемое также DTD (Document Type Definition). DTD-блок является основой структуризации содержимого XML-файла. В нем находятся правила, по которым происходит разметка содержимого документа. В этом блоке определяются элементы документа, атрибуты для этих элементов, сущности, комментарии. Другими словами, DTD-блок не менее важен для XML-документа, чем его значимое содержимое. Рассмотрим пример объявления типа документа.

```
<!DOCTYPE body [
<!ELEMENT body (#PCDATA)>
<!ENTITY name "Igor">
]>
```

В первой строке примера начинается объявление типа документа. Как видно, используется ключевое слово `DOCTYPE`, помещаемое после открывающей угловой скобки и восклицательного знака. Необходимо отметить, что все конструкции XML помещаются в угловые скобки. Но перед большинством из ключевых слов ставится восклицательный знак. А перед исполняемыми инструкциями XML-процессора, как мы помним, ставится вопросительный знак.

После ключевого слова `DOCTYPE` пишется наименование типа. Это, по сути, имя корневого элемента объектной иерархии данного документа. Здесь необходимо сделать некоторое отступление и рассказать немного об элементах XML-документа, которые более полно мы будем рассматривать в следующей части. Но в XML все настолько переплетено и хорошо увязано, что есть только один путь абсолютно последовательного и непротиворечивого изложения правил оформления XML-документов. Это официальная спецификация, части которой мы приводим в тексте этой книги. Так как читать ее трудно, придется немного переплетать изложение различных структурных единиц, принося последовательность изложения в жертву его доступности.

Итак, элементы в XML являются основными структурными единицами. В качестве одной из аналогий можно вспомнить теги в HTML. Там мы могли создавать абзацы, элементы списков, ссылки и прочие элементы разметки. Так вот, их можно назвать прародителями элементов XML. Но в HTML элементы разметки не были взаимосвязаны. В XML был сделан шаг вперед. Теперь элементы могут быть структурированы. Все элементы зависят друг от друга. В XML-документе есть основной элемент, в который в качестве его частей входят иные элементы. То есть достаточно упрощенно моделируется объектная иерархия, в которой в качестве объектов выступают элементы XML. Как мы увидим немного позже, это достаточно хорошая аналогия, и мы еще не раз воспользуемся ею.

Таким образом, в качестве наименования типа XML-документа мы используем имя самого старшего элемента, который включает в себя все остальные документы. В нашем примере это элемент с именем `body`.

После указания этого имени в квадратных скобках описывается структура всего документа. А после закрывающей квадратной скобки следует закрывающая угловая скобка. Получается, что все определение типа документа находится в рамках одного тега `DOCTYPE`. Все это легко увидеть в коде рассмотренного нами примера.

DTD-блоки могут находиться как внутри XML-документа, так и вне его. Для нескольких XML-документов можно создать один файл с описанием DTD и использовать его для каждого из этих XML-документов. Для подключения внешнего DTD-блока можно использовать конструкцию, подобную этой:

```
<!DOCTYPE main PUBLIC "-//New Boundaries//Sweet Immersing//
EN" "http://www.newboundaries.com/sweet/dtd/main.dtd">
```

Разберем этот пример. Мы объявляем тип документа `main`, спецификация которого находится в отдельном DTD-файле. В данном случае объявлено, что это DTD не принято международной организацией по стандартизации (ISO) в качестве стандарта, оно принадлежит компании New Boundaries, создано для проекта Sweet Immersing, ориентировано на англоязычные документы, и файл со спецификацией находится по адресу **http://www.newboundaries.com/sweet/dtd/main.dtd**.

Ключевое слово `PUBLIC` применяется для так называемых "публичных" DTD. Для них помимо URL указывают еще и наименование. Делается это для того, чтобы XML-браузер мог отыскать эти DTD на других серверах, которые, может быть, будут ближе. Конечно, в критериях Интернета понятие "ближе" относится к географии весьма опосредованно, но выбирается всегда тот DTD-файл, который будет быстрее всего загружен на локальную систему удаленного пользователя. Публичные DTD отличаются от обычных обособленных DTD-файлов тем, что могут находиться на нескольких различных серверах.

Еще одна особенность публичных DTD состоит в том, что их рассматривает международная организация по стандартизации (ISO). В том случае, если этот DTD-блок принят в качестве стандарта, перед именем файла (а не перед его URL) ставится префикс `ISO`. Если в качестве стандарта этот DTD не принят, но, тем не менее, рассматривается группой стандартизации, в качестве префикса используется знак плюса (+). Если же он не принят группой стандартизации, используется префикс в виде минуса (-), как мы это видели в нашем примере.

Если же DTD-блок вообще не отправлялся в ISO, то он объявляется "приватным". В таком случае вместо ключевого слова `PUBLIC` используется ключевое слово `SYSTEM`. При этом применяется конструкция следующего типа:

```
<!DOCTYPE main SYSTEM "http://www.newboundaries.com/sweet/main.dtd">
```

Теперь, когда мы знаем, как использовать внешние файлы, содержащие DTD-блоки, следует научиться создавать эти файлы. Любой DTD-файл под-

чиняется спецификациям XML, т. е. по сути, он является обычным XML-документом, но без значимого содержимого. Поэтому первой строкой снова будет исполняемая инструкция XML-процессора. Но теперь совсем необязательно указывать номер версии стандарта XML. Обычно во внешних DTD-файлах указывают используемую кодировку. После этого определяются все структурные элементы. Так как ссылка на DTD-файл уже встраивается в соответствующий XML-документ, нет нужды в DTD-файле использовать объявление типа документа с ключевым словом `DOCTYPE`.

Мы уже рассматривали параметр `standalone` для исполняемой инструкции, которая объявляет использование XML. В том случае, если используется внешний DTD-файл, необходимо использовать значение `no`.

В зависимости от того, как используется DTD-блок, какие правила оформления DTD применяются в XML-документах, эти документы могут называться *"хорошо оформленными"* (well-formed) и *"правильными"* (valid). Кстати, некоторые правила спецификации XML являются обязательными для оформления документа, если мы хотим, чтобы этот документ являлся хорошо оформленным или правильным. Для таких правил в спецификации указываются индексы `WFC` (Well-Formedness Constraint) и `VC` (Validity Constraint), соответственно. Но чтобы избавить себя от штудирования неудобочитаемой спецификации, рассмотрим эти правила сами.

Итак, для того чтобы документ считался хорошо оформленным, необходимо выполнение нескольких правил.

- У документа должен быть один элемент верхнего уровня, и содержимое документа должно полностью располагаться между тегами, соответствующими этому элементу-прародителю.
- В одном теге не могут использоваться несколько раз одни и те же атрибуты элемента, соответствующего этому тегу.
- Все сущности должны объявляться до их использования.
- Все теги должны быть правильно вложены друг в друга, согласно иерархии элементов, и у каждого тега должен быть его закрывающий двойник.

По сути дела, все XML-документы, которые будут использоваться в публичных целях, просматриваться в различных браузерах и обрабатываться различными приложениями, должны быть, как минимум, хорошо оформленными. Обязательное использование вышеперечисленных правил обусловлено тем, что при несоблюдении какого-либо из них, обычный XML-процессор выдаст сообщение о фатальной ошибке, после чего приложение, в котором действует XML-процессор, будет остановлено. Более того, кое-где встречается информация о том, что отдельные приложения и браузеры "зависают" при попытке просмотра документов с ошибками.

Следует обратить внимание на то, что в перечне правил для хорошо оформленных документов нет ни слова о DTD-блоках. Связано это с тем, что по

спецификации XML DTD-блоки не являются обязательными для использования в XML-документах. Поэтому следующий XML-документ считается правильным, так как не нарушено ни одно из правил.

```
<?xml version="1.0" standalone="yes"?>
<body>well-formed document </body>
```

Еще раз напоминаю, что XML-процессоры чувствительны к регистру символов.

Правильным считается XML-документ, который удовлетворяет требованиям, предъявляемым к хорошо оформленным документам, и при этом имеет соответствующий DTD-блок и подчиняется всем правилам, описанным в нем.

Для того чтобы вышеприведенный пример XML-документа можно было бы считать правильным, необходимо внести в его код некоторые изменения. Теперь он будет выглядеть следующим образом:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE body [<!ELEMENT body (#PCDATA)>
]>
<body>valid document </body>
```

Теперь, после рассмотрения примеров DTD-блоков и всех правил, связанных с ними, приведем блок спецификации, описывающий определение типа документа.

```
[28]  doctypedecl ::=  '<!DOCTYPE' S Name (S ExternalID)? S?
   ('['(markupdecl | PEReference | S)* ']' S?)? '>' [VC: Root Element
   Type ]
[29]  markupdecl ::=  elementdecl | AttlistDecl | EntityDecl
   | NotationDecl | PI | Comment  [ VC: Proper Declaration/PE Nesting ]
      [  WFC: PEs in Internal Subset ]
```

В спецификации видно, как в 28-й строке накладывается условие наличия корневого элемента, необходимое для создания правильного документа.

Элементы XML-документа

Мы уже говорили в предыдущих частях об элементах. Элементы являются основными структурными единицами XML-документов. Мы объявляем все элементы документа в DTD-блоке, а потом при разметке значимого содержимого документа мы используем теги, наименования которых совпадают с наименованиями элементов.

Объявление элемента в самом упрощенном виде мы уже наблюдали в одном примере. Приведем его еще раз.

```
<!DOCTYPE body [
<!ELEMENT body (#PCDATA)>
]>
```

Здесь мы объявляем тип документа `body`, в котором объявляется одноименный элемент. Как видно, объявление элемента производится при помощи ключевого слова `ELEMENT`, после которого указывается наименование элемента и его тип. Все эти значимые и обязательные части разделяются пробелами. В данном случае мы объявили элемент с символьным содержимым, применив для этого стандартную конструкцию `(#PCDATA)`.

На самом деле элементы в качестве своего содержимого могут использовать не только символьные данные. В них могут входить другие элементы, которые в свою очередь могут тоже содержать элементы, и так далее. То есть, выстраивается некая иерархия элементов, вся вложенная как в матрешку в основной элемент, объявленный как тип документа при помощи ключевого слова `DOCTYPE`.

Рассмотрим пример такого сложного элемента. Предположим, что мы создаем XML-документы для некоей организации, которая ведет базу данных предприятий города. Нам потребуется создать основной элемент `firm`, в котором будут находиться вложенные элементы, отражающие название фирмы, ее род деятельности, адрес, телефоны, электронные реквизиты, и так далее. Все это можно проиллюстрировать следующим примером:

```
<!ELEMENT firm (name+, address, phone*, fax*, email, info)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT info (#PCDATA)>
```

В этом примере мы создаем элемент, в состав которого входят шесть других элементов с содержимым символьного типа. При этом для группировки субэлементов используются как обычно круглые скобки. Все, казалось бы, достаточно прозрачно, но если приглядеться, видно, что после наименований некоторых элементов, входящих в состав элемента `firm`, установлены дополнительные модифицирующие знаки. Назовем их модификаторами. Всего в XML применяется три различных модификатора.

Знак плюса (`+`) применяется для тех элементов, чье вхождение в родительский элемент является обязательным. Данные элементы могут встречаться в рамках родительского элемента один или более раз. Но хотя бы один раз они должны быть использованы обязательно.

Знак звездочки (`*`) указывает на то, что данный элемент может встречаться в описании родительского элемента любое количество раз. Впрочем, может там вообще не встречаться.

Вопросительный знак (`?`) используется для тех элементов, которые могут появляться в описании родительского элемента только один раз, или вообще не появляться.

Все остальные символы, которые могут встречаться в описании типа элемента, и в то же время не входят в наименования субэлементов, служат либо для группировки, либо в качестве разделителей.

Группировка элементов, как мы знаем, осуществляется при помощи круглых скобок. Применяя запятую, мы фактически устанавливаем порядок следования элементов. Таким образом, в значимом содержимом XML-документа все субэлементы должны следовать именно в том порядке, в котором они записаны в DTD-блоке.

В том случае, если нам необходимо использовать один из некоторого множества субэлементов, это множество дополнительно заключается в круглые скобки, и все наименования субэлементов разделяются вертикальной чертой (|). Например, если бы мы хотели, чтобы в нашем примере для элемента `firm` можно было бы использовать либо номер факса, либо номер телефона, можно применить следующую декларацию:

```
<!ELEMENT firm (name+, address, (phone | fax), email, info)>
```

Также элементы могут состоять не только из других элементов. Всегда, в конце концов, должны найтись конкретные элементы, в которых не будет содержаться дочерних элементов. Подобные элементы называются *атомарными*.

Это, впрочем, не означает, что в атомарных элементах будут находиться единичные и неделимые фрагменты информации. Каждый элемент может обладать атрибутами, которые в свою очередь и будут содержать подобные информационные атомы. Хорошей естественной аналогией для атрибутов являются свойства объектов. Но в этой части мы рассматриваем только элементы. Атрибуты мы будем рассматривать в следующей части.

Итак, нас интересует, как задавать тип для атомарных элементов. Всего есть три варианта.

Если мы собираемся сохранять в элементе какие-либо данные так называемого смешанного типа (mixed content), мы должны указать в круглых скобках тип `#PCDATA`. По сути, под эвфемизмом "смешанный тип" подразумевается использование любых символьных данных.

Если заранее не определено, какой тип данных будет содержать этот элемент, то для указания его типа используется ключевое слово `ANY`.

Для обозначения типа пустых элементов (заглушки) используется ключевое слово `EMPTY`.

Все типы могут комбинироваться в объявлении элемента таким образом, чтобы достигнуть именно того результата, который нам необходим. В качестве примера можно рассмотреть следующие декларации:

```
<!ELEMENT b (#PCDATA)>
<!ELEMENT br EMPTY>
```

```
<!ELEMENT container ANY>
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
```

Здесь видно, что элемент `b` содержит символьные данные, используется, так называемый, смешанный тип содержимого, элемент `br` изначально задан пустым, элемент `container` предназначен для хранения данных любого типа, а элемент `p` может содержать либо символьные данные, либо один из других элементов, чьи имена, разделенные вертикальной чертой, указаны в скобках.

И в заключение обзора элементов мы должны привести фрагмент спецификации, который описывает объявление элементов.

```
[45]  elementdecl ::=  '<!ELEMENT' S Name S contentspec S? '>'
[  VC: Unique Element Type Declaration ]

[46]  contentspec ::=  'EMPTY' | 'ANY' | Mixed | children

[47]  children ::=  (choice | seq) ('?' | '*' | '+')?

[48]  cp ::=  (Name | choice | seq) ('?' | '*' | '+')?

[49]  choice ::=  '(' S? cp ( S? '|' S? cp )* S? ')'
[  VC: Proper Group/PE Nesting ]

[50]  seq ::=  '(' S? cp ( S? ',' S? cp )* S? ')'
[  VC: Proper Group/PE Nesting ]
```

Атрибуты элементов

В предыдущей части мы уже говорили о том, что неделимые порции информации записываются и хранятся в атрибутах элементов. При использовании аналогии элементной модели с объектной иерархией атрибуты можно считать свойствами объектов. При помощи атрибутов мы можем максимально полно детализировать информацию, предназначенную для отображения или использования в данном элементе.

Список атрибутов задается при помощи ключевого слова `ATTLIST`. Рассмотрим маленький пример.

```
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
    create_date    CDATA #REQUIRED
    last_modified  CDATA #IMPLIED
    author         CDATA #REQUIRED
    editor         CDATA "Kondukova E.V.">
```

В этом примере мы объявили элемент `text`, в котором содержится четыре атрибута. Как видно, после ключевого слова `ATTLIST` указывается наименование элемента, к которому присоединяются атрибуты, определяемые в данном теге ниже. После наименования элемента записывается список атрибутов. Для каждого атрибута указываются наименование, тип содержимого атрибута и модификатор атрибута.

Что касается наименования, то с ним все ясно. Наименования атрибутов полностью подчиняются правилам формирования имен в XML. Напомним их. Имя обязано начинаться с буквенного символа, символа подчеркивания или двоеточия, и может содержать в себе любые буквы, цифры и знаки подчеркивания, двоеточия, дефиса и точки.

После имени атрибута указывается его тип. Всего может быть три типа. В нашем примере мы использовали атрибуты символьного типа. Для обозначения этого типа применяется ключевое слово `CDATA`. В этих атрибутах могут храниться только данные в виде строк. В приведенном выше примере объявлены атрибуты именно такого типа.

Также атрибуты могут быть перечислимого типа. Для подобных атрибутов при их объявлении заранее указывается список возможных значений, которые заключаются в круглые скобки и разделяются вертикальной чертой. В качестве примера можно привести следующее объявление атрибута:

```
<!ATTLIST my_element
    type (a|b|c) "c">
```

В этом примере объявляется атрибут `type`, который может принимать только значения `a`, `b` и `c`, причем по умолчанию используется значение `c`.

Применяется также и третий тип атрибутов. Подобные атрибуты называются маркерами. Это специализированные атрибуты, значения которых несут заранее предопределенный тип информации об элементе. При использовании атрибутов-маркеров в качестве типа атрибута мы записываем одно из семи ключевых слов, которые идентифицируют вид маркера. Так, например, даже в последней версии стандарта HTML в каждом теге присутствует параметр `ID`, при помощи которого задают идентификатор блока, обрабатываемого данным тегом. В XML этот параметр задан изначально и является атрибутом-маркером. Приведем пример задания подобного атрибута.

```
<!ATTLIST my_element
    id ID #REQUIRED>
```

В этом примере видно, как мы задаем для элемента `my_element` параметр `id`, для которого указываем тип `ID`.

Как мы уже говорили, есть семь предопределенных типов элементов-маркеров. Рассмотрим их все.

Атрибут типа `ID` предназначен для указания уникального идентификатора данного элемента. Значение этого атрибута должно полностью удовлетворять соглашению об именах в XML. Естественно, идентификаторы всех экземпляров элементов в содержимом XML-документа должны быть разными, иначе нарушается условие уникальности. Если же это условие нарушается, то XML-процессор укажет на ошибку в документе.

Атрибут `IDREF` содержит идентификатор другого элемента, с которым данный элемент связан по смыслу. Подобный атрибут предназначен для реали-

зации двунаправленных ссылок, одной из наиболее широко разрекламированных возможностей, вошедших в стандарт XML версии 1.0. Естественно, элемент, идентификатор которого указан в данном атрибуте, должен присутствовать в XML-документе, иначе XML-генератор объявит о недействительности данного документа.

Если данный элемент связан смысловыми взаимоотношениями с несколькими другими документами, то мы можем указать все их идентификаторы сразу в одном атрибуте. Для этого используется атрибут типа `IDREFS`. В его значении все идентификаторы связанных элементов записываются, разделенные пробелами.

Атрибут `ENTITY` указывает на внешнюю сущность, связанную с данным элементом. В качестве значения этого атрибута указывается имя сущности. Более подробно механизм сущностей мы рассмотрим в следующей части.

Атрибут `ENTITIES` весьма похож на предыдущий. Разница лишь в том, что в его значении мы можем записать сразу несколько имен внешних сущностей.

Атрибут `NMTOKEN` содержит любую последовательность символов имени данного элемента.

Идею этого атрибута расширяет атрибут с именем `NMTOKENS`, который в своем значении может содержать сразу несколько подстрок имени данного элемента.

Мы рассмотрели типы атрибутов, которые записываются в их объявлениях сразу после наименования атрибута. Но у каждого атрибута есть еще и некая характеристика. Мы можем задать атрибут, который будет обязателен для применения в каждом экземпляре элемента, задать значение по умолчанию для каждого атрибута, и так далее. Есть три стандартных модификатора для атрибутов.

Модификатор `#IMPLIED` указывает на то, что для данного атрибута значение может быть не определено, т. е. этот атрибут не является обязательным для применения.

Модификатор `#REQUIRED` наоборот, применяется для указания обязательных атрибутов. А именно, если у данного атрибута некоего элемента есть этот модификатор, то во всех экземплярах элемента в значимом содержимом XML-документа должно быть обязательно присвоено значение этому атрибуту.

Модификатор `#FIXED` применяется в тех случаях, когда заданное для атрибута значение по умолчанию является фиксированным, т. е. не поддается изменению. Проще говоря, единожды объявленное значение атрибута в тексте XML-документа изменить не удастся. Понятно, что этот модификатор не применяется для атрибутов перечислимого типа.

Для каждого атрибута можно задать значение, применяемое по умолчанию. Причем оно может как замещать стандартный модификатор, так и использоваться вместе с ним. Рассмотрим еще раз наш пример.

```
<!ELEMENT text (#PCDATA)>
<!ATTLIST text
    create_date    CDATA #REQUIRED
    last_modified  CDATA #IMPLIED
    author         CDATA #REQUIRED
    editor         CDATA "Kondukova E.V.">
```

Здесь видно, что атрибуты `create_date` и `author` являются обязательными, а атрибут `last_modified` может и не использоваться в элементах `text`. А для атрибута `editor` предусмотрено значение по умолчанию `Kondukova E.V.`, которое ограничивается двойными кавычками, как строковая переменная. В этом примере значение по умолчанию не комбинируется с модификатором. Пример такого сочетания выглядит следующим образом:

```
<!ATTLIST hyperlink
xlink:type  CDATA  #FIXED  "simple">
```

В этом примере мы объявляем элемент `hyperlink` с атрибутом `xlink:type`, для которого установлено фиксированное значение `simple`. Следовательно, каждый раз, когда в тексте XML-документа будет использоваться элемент `hyperlink`, его атрибут `xlink:type` будет всегда иметь значение `simple`, и изменить это значение в тексте XML-документа для какого-либо экземпляра элемента будет нельзя.

Атрибуты перечислимого типа мы можем комбинировать со списком условных обозначений (`NOTATION`). Механизм подобных списков мы будем рассматривать в одной из следующих частей. А сейчас нам необходимо просто привести пример подобного комбинирования. После наименования атрибута мы указываем в качестве его типа ключевое слово `NOTATION`. Затем, как обычно в скобках, указываем сами условные обозначения, входящие в данный список. А после перечисления вписываем значение, используемое по умолчанию. Все это можно увидеть в следующем примере:

```
<!ATTLIST new_element
    attr NOTATION (first|second|third) "third">
```

Естественно, в DTD-блоке должно присутствовать определение подобного списка, в котором определяются все элементы списка.

Итак, мы рассмотрели все правила создания атрибутов. Теперь приведем фрагмент спецификации, который описывает создание атрибута.

```
[52]  AttlistDecl ::=  '<!ATTLIST' S Name AttDef* S? '>'
[53]  AttDef ::=  S Name S AttType S DefaultDecl
[54]  AttType ::=  StringType | TokenizedType | EnumeratedType
[55]  StringType ::=  'CDATA'
[56]  TokenizedType ::=  'ID' [  VC: ID ]
    [  VC: One ID per Element Type ]
```

```
    [  VC: ID Attribute Default ]
   | 'IDREF' [  VC: IDREF ]
   | 'IDREFS' [  VC: IDREF ]
   | 'ENTITY' [  VC: Entity Name ]
   | 'ENTITIES' [  VC: Entity Name ]
   | 'NMTOKEN' [  VC: Name Token ]
   | 'NMTOKENS' [  VC: Name Token ]
[57]  EnumeratedType ::=  NotationType | Enumeration
[58]  NotationType ::=  'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'
[  VC: Notation Attributes ]
[59]  Enumeration ::=  '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')' [  VC:
Enumeration ]
[60]  DefaultDecl ::=  '#REQUIRED' | '#IMPLIED'
   | (('#FIXED' S)? AttValue) [  VC: Required Attribute ]
    [  VC: Attribute Default Legal ]
    [  WFC: No < in Attribute Values ]
    [  VC: Fixed Attribute Default ]
```

Сущности

Сущности являются одним из ключевых элементов XML. Под ними обычно понимают единый и неделимый блок информации, который не анализируется XML-генератором. Сущности могут представлять собой файлы с бинарной информацией, такой как рисунки, аудио- и видеофайлы, документы, обрабатываемые специализированными приложениями, и многое другое. Также в качестве сущностей используются часто применяемые блоки информации. Достаточно их объявить в качестве сущности в DTD-блоке и потом во всех XML-документах, подключенных к этому DTD-блоку, можно применять обозначение этой сущности. Таким образом, если необходимо изменить этот элемент, мы можем изменить его в одном месте, в объявлении сущности, и изменения во всех связанных с этим DTD-блоком XML-документах будут сделаны автоматически. Также сущности могут применяться и в самом DTD-блоке для оформления часто встречающихся последовательностей атрибутов или элементов, т. е. выполняют обычную функцию текстовой подстановки. При помощи все тех же сущностей мы можем вставлять в текст XML-документа неотображаемые символы. По сути, любая сущность является контейнером для каких-либо данных, помещаемых в XML-документ. Все эти ипостаси мы тщательно рассмотрим, и приведем соответствующие примеры.

Итак, официально сущности являются некими элементами разметки, которые содержат объявление текста или данных, которые вставляются в значимое содержимое XML-документа при помощи связанных с ними псевдонимов.

Сущности по своему типу делятся на текстовые, бинарные и параметрические. По своему месту определения, декларирования мы можем сущности делить на внешние и внутренние.

Любая сущность объявляется в DTD-блоке при помощи ключевого слова `ENTITY`. После него, отделенное пробелом, следует наименование сущности, ее псевдоним. А затем, уже в самом конце декларации мы вписываем содержимое сущности, заключенное в двойные кавычки. В качестве примера наиболее простой текстовой сущности приведем следующую конструкцию:

```
<!ENTITY cp "copyright disclaimer">
```

Теперь, если в тексте XML-документа мы вставим наименование этой сущности, обрамленное знаками амперсанта и точки с запятой, то оно при отображении документа браузером, будет заменено на соответствующее словосочетание. То есть в тексте для вызова сущности мы должны использовать конструкцию `&cp;`.

Также существуют и предопределенные текстовые сущности. В значимом содержимом XML-документа мы можем применять только символы из набора ASCII. Все остальные символы будут вставляться в текст только при помощи сущностей. По существу, все кодировки представляют собой набор сущностей. Наиболее распространенные кодировки приведены в приложении.

В ASCII-наборе есть несколько символов, которые не могут самостоятельно вставляться в текст XML-документа. Это служебные символы, применяемые в разметке кода XML-документа, такие как знаки "больше" и "меньше", знак амперсанта (&), а также одинарные и двойные кавычки. Для вставки этих символов в текст значимого содержимого XML-документа используются сущности `>`, `<`, `&`, `'` и `"` соответственно.

Бинарные сущности позволяют вставлять в текст XML-документа любые внешние файлы, от графических изображений, до файлов Microsoft Office, не забывая, естественно, мультимедиа-файлы. Вообще, любые файлы можно присоединить к XML-документу, необходимо лишь указать какие приложения должны вызываться для обработки того или иного файла. Приведем пример сущности, которая подключает графический GIF-файл:

```
<!ENTITY picture SYSTEM "images/pic1.gif" NDATA gif>
```

Все двоичные сущности будут *внешними*, так как они встраивают внешние по отношению к XML-документу файлы. Поэтому после объявления имени, так называемого псевдонима сущности, следует ключевое слово `SYSTEM`, следом за которым указывается URL подключаемого файла. Все это достаточно прозрачно, так как не противоречит ничему, что мы узнали о разметке XML ранее. Но есть одно достаточно тонкое место. XML-процессор "не знает" о том, какой именно тип файла мы используем. Он не обязан автоматически распознавать тип файла по расширению. Поэтому для каждого типа

файлов, используемого в оформлении, необходимо указать программу, применяемую для обработки данного типа файла. Для этого используется механизм условных обозначений (NOTATION), который мы рассмотрим в следующей части детально. Здесь мы приведем маленький пример. Если мы в объявлении сущности picture использовали тип gif, указав его после ключевого слова NDATA, то в DTD-блоке должно присутствовать следующее объявление условного обозначения:

```
<!NOTATION gif SYSTEM "http://www.mysite.ru/progs/gifview.exe">
```

В этом объявлении мы указываем, что для обработки файлов типа gif необходимо вызвать приложение, URL которого указан после ключевого слова SYSTEM в двойных скобках.

В объявлении бинарной сущности необходимо внимательно следить за типом файла, указываемом после ключевого слова NDATA. Каким бы ни было расширение подключаемого файла, XML-процессор не будет принимать его во внимание. Тип устанавливается по соответствующему обозначению.

Однако есть одна возможность создавать и использовать бинарные внешние сущности без указания типа присоединяемого файла. Это имеет место, если вставляются сторонние XML-файлы. При помощи этого средства мы можем собирать объемный XML-документ из других документов. Другими словами, если у нас уже есть три XML-файла, содержимое которых необходимо объединить в одно целое, следует объявить их как внешние бинарные сущности, и вызвать в основном документе при помощи их псевдонимов. Приведем соответствующий пример.

```
<!xml version="1.0">
<!DOCTYPE body [
<!ENTITY doc1 SYSTEM "http://www.parts.ru/part1.xml">
<!ENTITY doc2 SYSTEM "http://www.parts.ru/part2.xml">
<!ENTITY doc3 SYSTEM "http://www.parts.ru/part3.xml">
]>
А в теле документа следует использовать следующую конструкцию:
<body>
&doc1;
&doc2;
&doc3;
</body>
```

При этом мы можем использовать в объявлении внешней бинарной сущности не только ключевое слово SYSTEM, но и PUBLIC. Но в этом случае мы должны использовать только те файлы, которые рассматривались международной организацией по стандартизации. Но с другой стороны, так ли уж необходимо Вам стандартизировать свои данные?

Мы еще не рассмотрели третий вид сущностей — *параметрические*. Они применяются для обозначения часто применяемых групп атрибутов или содержания элемента, т. е. здесь используется понятие тривиальной текстовой подстановки.

Эти сущности объявляются и вызываются внутри DTD-блока. Естественно, параметрическую сущность сначала необходимо объявить, а уже потом — использовать.

Есть и еще одно отличие от обычных сущностей. Для вызова параметрических сущностей, перед их псевдонимом мы должны поставить не знак амперсанта, а знак процента. Этот же знак процента ставится и при объявлении параметрической сущности.

Приведем маленький пример. Нам необходимо объявить два разных элемента, причем у каждого из них частично будут совпадать наборы атрибутов. Эту-то совпадающую часть мы и объявим как параметрическую сущность, а затем используем ее в декларировании самих элементов.

```
<!ENTITY % common_atts "
    last_modified      CDATA    #IMPLIED
    description        CDATA    #IMPLIED
    id                 ID       #REQUIRED">
<!ELEMENT el_first (#PCDATA)>
<!ELEMENT el_second (#PCDATA)>
<!ATTLIST el_first
    special_attribut   CDATA    #IMPLIED
    %common_atts;>
<!ATTLIST el_second
    sex               (male|female)
    %common_atts;>
```

В примере видно, как объявляются и используются параметрические сущности. Необходимо отметить, что при декларировании параметрической сущности необходимо знак процента отделять пробелом от псевдонима сущности. Если пробела не будет, то XML-процессор выдаст сообщение об ошибке.

Теперь детально рассмотрим механизмы объявления внутренних и внешних сущностей. Объявление внутренних сущностей не представляет особых проблем. Вся информация, входящая во внутреннюю сущность, находится внутри одного DTD. Для декларирования внутренней сущности необходимо после псевдонима сущности указать само содержимое сущности. В случае использования внешних сущностей после псевдонима необходимо указать тип ссылки на содержимое сущности. Как и все внешние ресурсы, содержимое сущности может быть как обычным, так и стандартизованным. Если

используется обычный внешний ресурс, то в объявлении после псевдонима мы указываем ключевое слово SYSTEM, а затем URL ресурса, являющегося содержимым ссылки. Если же содержимое сущности по какому-либо недоразумению было отослано в ISO и прошло там рассмотрение, то после псевдонима мы указываем ключевое слово PUBLIC, затем официальное наименование ресурса, а уже после него — URL содержимого.

Приведем маленький пример. В нем находится три декларации. Первая объявляет обычную внутреннюю сущность, вторая — нестандартизованную внешнюю бинарную сущность, третья предназначена для внешней сущности, чье содержимое было оценено международной организацией по стандартизации.

```
<!ENTITY cp "All rights are protected">
<!ENTITY pic SYSTEM "http://www.graphics.com/images/c3.gif" NDATA gif>
<!ENTITY firm PUBLIC "+//New Boundaries//Sweet Immersing//EN"
"http://www.bound.com/dtd/f.xml">
```

Теперь, когда мы рассмотрели правила объявления и применения сущностей, узнали, что это такое и для чего они нужны, разобрали несколько примеров, необходимо привести фрагмент официальной спецификации XML, в котором описываются сущности. Что ж, вот он.

```
[67]  Reference ::=  EntityRef | CharRef
[68]  EntityRef ::=  '&' Name ';' [  WFC: Entity Declared ]
    [  VC: Entity Declared ]
    [  WFC: Parsed Entity ]
    [  WFC: No Recursion ]
[69]  PEReference ::=  '%' Name ';' [  VC: Entity Declared ]
    [  WFC: No Recursion ]
    [  WFC: In DTD ]
[70]  EntityDecl ::=  GEDecl | PEDecl
[71]  GEDecl ::=  '<!ENTITY' S Name S EntityDef S? '>'
[72]  PEDecl ::=  '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef ::=  EntityValue | (ExternalID NDataDecl?)
[74]  PEDef ::=  EntityValue | ExternalID
[75]  ExternalID ::=  'SYSTEM' S SystemLiteral
   | 'PUBLIC' S PubidLiteral S SystemLiteral
[76]  NDataDecl ::=  S 'NDATA' S Name [  VC: Notation Declared ]
```

Комментарии и условные обозначения

В этой части мы рассмотрим все остальные элементы XML. Правда, их осталось всего два. Это комментарии и условные обозначения, которые мы

упоминали немного выше по тексту. Начнем с условных обозначений, они нам нужны больше.

Итак, условные обозначения являются специальными элементами DTD, которые позволяют ассоциировать тип файла, включаемого в значимое содержимое XML-файла, и приложение, которое будет обрабатывать эти файлы. XML-браузеры не занимаются отображением иных файлов, таких как графика, аудио- или видеофайлы, самостоятельно, в отличие от известных нам HTML-браузеров. Для этих целей используются их "родные" приложения.

Условное обозначение задается при помощи ключевого слова `NOTATION`, после которого указывается наименование типа файла (тип файла совсем не обязательно должен совпадать с его расширением, хотя это очень удобно) без ограничивающих кавычек, а затем — URL обрабатывающего приложения. Так как все приложения являются внешними ресурсами по отношению к данному XML-файлу, то, естественно, необходимо использовать ключевые слова `SYSTEM` или `PUBLIC`.

Приведем пример связки внешней бинарной сущности и соответствующего ей условного обозначения. Внешняя сущность будет ссылаться на графический gif-файл, а условное обозначение задаст URL приложения, которое будет отображать этот файл.

```
<!ENTITY pict SYSTEM "http://www.graphix.net/images/c2.gif" NDATA gif>
<!NOTATION gif SYSTEM "file://c:/Program Files/acdsystem/acdsee.exe">
```

При этом нужно помнить, что если в объявлении внешней бинарной сущности после ключевого слова `NDATA` будет указан неверный тип файла, например "avi", то будет вызван обработчик не для gif-файлов, а именно для avi-ресурсов, несмотря на расширение самого файла.

В официальной спецификации объявление условных обозначений регламентируется следующим образом:

```
[82]  NotationDecl ::=  '<!NOTATION' S Name S (ExternalID |  PublicID) S? '>'
```

Теперь переходим к комментариям. Сейчас уже ни у кого не возникает сомнений в их необходимости. Любой достаточно объемный код необходимо тщательно документировать. Иначе уже через месяц понять, что значат ранее такие понятные и простые обозначения, будет весьма сложно.

В XML комментарии обозначаются очень просто. Они ограничиваются сочетаниями символов `<!--` и `-->`. В качестве примера можно привести следующую конструкцию:

```
<!-- Это комментарий -->
```

Официальная спецификация XML описывает комментарии следующим образом:

```
[15]  Comment ::=  '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

На этом мы заканчиваем рассмотрение элементов DTD и можем переходить к оформлению самих XML-документов.

Тело XML-документа

Теперь, после того, как мы полностью обозначили структуру документа, можно приступать к созданию самого документа. Главное в нем, все-таки, не разметка, а его наполнение. Как и в HTML, разметка значимого содержимого производится при помощи тегов. Наименования тегов совпадают с наименованиями элементов, объявленных в DTD-блоке. При этом должна полностью соблюдаться последовательность вложенности тегов. Иначе говоря, все содержимое документа заключено между тегами, которые определены основным элементом — типом документа. Если какие-либо элементы входят в качестве составных частей в другие элементы, то и соответствующие теги должны точно так же вкладываться друг в друга с соблюдением иерархии. Все это можно проиллюстрировать одним примером цельного XML-документа.

```
<?xml version="1.0"?>
<!DOCTYPE body [
<!ELEMENT book (chapter)*>
<!ELEMENT chapter (#PCDATA)>
<!ENTITY name "Igor">
]>
<body>
<book>
<chapter>Hello, world. My name is &name;</chapter>
<chapter>Hello, world. It's me again</chapter>
</book>
</body>
```

В этом примере мы объявляем DTD-блок, в котором объявлен тип документа `body`, затем объявляем элемент `book`, составной частью которого является атомарный элемент `chapter`. Этот атомарный элемент может несколько раз встречаться в одном экземпляре элемента `book`.

После DTD-блока мы начинаем работать с самим документом. Так как основной элемент объявлен типом документа, мы, соответственно, сначала открываем тег `<body>`, а в самом конце документа закрываем его. В этот блок мы вкладываем тег `<book>`, а в него уже вкладываем два набора тегов `<chapter>`.

В этом же примере видно, как используется обычная внутренняя текстовая сущность. В DTD-блоке мы объявляем ее с псевдонимом `name`, а затем используем в текстовой строке, относящейся к одному экземпляру элемента `chapter`.

Естественно, та информация, которую мы ограничиваем теми или иными тегами, должна полностью соответствовать по типу тому элементу, к которому

она относится. Есть только один тип элементов, для которого приведенный пример не будет работать. В том случае, если мы объявили пустой элемент, т. е. при указании его типа в декларации этого элемента мы использовали ключевое слово EMPTY, то и тег, соответствующий этому элементу, должен быть пустым. Но экземпляры пустых элементов, это не просто открывающий и закрывающий теги, без какой-либо информации между ними. Для обозначения пустых элементов мы должны использовать соответствующие пустые теги. Их применение показано в следующем примере.

```
<!ELEMENT nothing EMPTY>
. . . . .
<nothing/>
```

В примере видно, что пустые элементы отражаются в XML-документе при помощи одного тега, в конце которого стоит наклонная черта.

Возникает вполне резонный вопрос, а для чего вообще нужны пустые элементы? На самом деле экземпляр пустого элемента может хранить информацию. Она не будет отображаться в XML-браузере, но она может быть обработана при помощи специализированных XML-приложений. Для того чтобы занести некую информацию в пустые элементы, используются их атрибуты.

Как вообще мы можем использовать атрибуты элементов в XML-документе? Рассмотрим пример.

```
<!ELEMENT computer EMPTY>
<!ATTLIST computer
    stone     CDATA       #IMPLIED
    memory    CDATA       #IMPLIED
    sound     (yes|no)    #IMPLIED>
. . . . .
<computer stone="Coppermine" memory="256" sound="yes"/>
```

В этом примере видно, как мы сначала объявляем пустой элемент computer с набором атрибутов, которые мы потом используем в содержимом XML-документа. Мы применяем тег пустого элемента, в котором между наименованием тега и завершающей наклонной чертой находятся все наши атрибуты.

Атрибуты в тегах содержимого XML-документа используются для задания дополнительных данных, относящихся к существующему экземпляру элемента. Как видно из примера, данные для атрибутов помещаются в тело тега в виде имя=значение, причем значение заключается в двойные кавычки.

Теперь, после того, как мы знаем, как создавать значимое содержимое XML-документа, мы можем рассмотреть законченный документ в качестве примера.

Листинг 5.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE body [
<!ELEMENT firm (name+, address+, management, phone*, e-mail, description)>
<!ELEMENT name        (#PCDATA)>
<!ELEMENT address     (#PCDATA)>
<!ELEMENT management  (#PCDATA)>
<!ELEMENT phone       (#PCDATA)>
<!ELEMENT e-mail      (#PCDATA)>
<!ELEMENT description(#PCDATA)>
<!ELEMENT paragraph   (#PCDATA)>
<!ATTLIST address
    property (real|official) #REQUIRED>
<!ATTLIST management
    owner    CDATA #IMPLIED
    CEO      CDATA #IMPLIED
    CFO      CDATA #IMPLIED
    CIO      CDATA #IMPLIED>
<!ENTITY copyright "Businee SuperBase Inc, 2000, All rights reserved">
]>
<body>
<firm>
<name>IBM</name>
<address property="real">unknown</address>
<management>respectable men</management>
<e-mail>ibm@ibm.com</e-mail>
<description>small hardware company</description>
</firm>
<firm>
<name>Microsoft</name>
<address property="real">Redmond, USA</address>
<management owner="Billy, great and awful">cool</management>
<e-mail>info@microsoft.com</e-mail>
<description>big software company</description>
</firm>
<paragraph>&copyright;</paragraph>
</body>
```

Прежде всего, автор должен заявить, что при составлении документа он не придерживался исторической правды и реального положения дел в компьютерной индустрии.

Итак, в этом примере мы объявляем элемент `body` в качестве типа данного документа, в качестве корневого элемента. В него уже входят обычные элементы `paragraph` и `firm`. Элемент `paragraph` является атомарным, а в элемент `firm` входят дочерние субэлементы. Так как все субэлементы имеют "говорящие" имена, поэтому в особой расшифровке не нуждаются.

Легко заметить, что для двух субэлементов объявлены наборы атрибутов, которые позволяют детализировать информацию элемента.

Также в DTD-блоке объявлена текстовая сущность с информацией об авторских правах. Так как мы не можем напрямую использовать какие-либо текстовые данные в XML-документе, мы специально создали элемент `paragraph`, который и будет служить контейнером для всех текстовых данных. В предпоследней строке кода XML-документа видно, как используется предварительно объявленная текстовая сущность.

На самом деле, DTD-блок не является обязательной частью для XML-документов. Если мы удалим из нашего примера весь DTD-блок, он, тем не менее, будет отображаться практически также. Правда, вместе с DTD-блоком мы должны будем удалить из значимого содержимого XML-документа все ссылки на сущности, объявленные в DTD-блоке. Необходимо учитывать, что без DTD-блока XML-документ не будет считаться правильным (valid). А отображаться документ будет точно так же.

А пока мы закончим рассмотрение стандарта XML, и перейдем к технологии XLink, которая разработана для создания гиперссылок нового поколения.

XLink. Умные гиперссылки

Для создания гиперссылок в XML-документах существует свой язык — XLink. Основной идеей XML было радикальное расширение возможностей HTML. Не менее революционный прорыв был сделан и в области гиперссылок. Теперь гиперссылки не являются однонаправленными. При помощи XLink мы можем создавать двунаправленные ссылки (туда — обратно), мультинаправленные ссылки, создавать кольцо ссылок на однородные ресурсы. Мы можем управлять поведением XML-браузера при осуществлении перехода по гиперссылке. При помощи гиперссылок, созданных в XLink, мы в состоянии автоматически устанавливать режим просмотра ресурса, на который эта гиперссылка указывает. Теперь мы способны сделать если не все, то очень многое.

Все гиперссылки в XML-документе являются обычными элементами. Но в них обязаны быть некоторые предустановленные параметры. Все эти детали мы рассмотрим в следующих частях этой главы. Одна из важных частей

концепции новых гиперссылок, которые в XML теперь называют "умными ссылками" (smart links), это указание ресурса, на который ссылается ссылка. Мы можем как напрямую указывать целый документ, на который будет указывать гиперссылка, так и указывать относительное положение ресурса. Для этого был создан специализированный язык для целеуказания, который называется XPointer.

Одной из основных проблем Интернета являются "висячие ссылки", которые возникают в том случае, если ресурс, на который они указывают, был удален, изменен или перемещен. В этом случае ссылка ведет в никуда. Возникает стандартная "ошибка 404", которая регистрируется при переходе по оборванной ссылке. Знакомая ситуация, не правда ли?

Так вот, XLink если и не решает эту проблему полностью, то, по крайней мере, снимает некоторые проблемы. Уже за это стоит использовать XML, если всех вышеперечисленных достоинств Вам не хватило.

Создание гиперссылок в XML

Как мы уже говорили, гиперссылки являются обычными элементами. Но необходимо оформить их должным образом. У элементов, которым отведена роль гиперссылок, должны быть совершенно специфические атрибуты.

В XML мы можем создавать как обычные гиперссылки (simple links), знакомые нам еще по HTML, так и расширенные гиперссылки (extended links). А уже из расширенных ссылок мы можем создавать группы расширенных ссылок (extended link groups) и иные виды гиперссылок.

Рассмотрим атрибуты элементов, необходимые для создания различных видов таких ссылок.

У каждого элемента, которому отводится роль ссылки, причем, вне зависимости от типа создаваемой ссылки, должен быть атрибут `xlink:href`, значение которого содержит точку назначения гиперссылки, и атрибут `xlink:type`, в котором указывается тип создаваемой гиперссылки. Самый простой вариант объявления обычной гиперссылки выглядит следующим образом:

```
<!ELEMENT my_link ANY>
<!ATTLIST my_link
    xlink:type    CDATA    #FIXED    "simple"
    xlink:href    CDATA    #REQUIRED>
```

Как видно из примера, атрибут `xml:type` всегда будет иметь модификатор `#FIXED`, так как тип ссылки не меняется. Атрибут `xml:href`, естественно, всегда должен присутствовать во всех экземплярах элемента, следовательно, он всегда будет объявляться с модификатором `#REQUIRED`.

Теперь, после разбора маленького примера, мы рассмотрим основные варианты создания ссылок.

Ссылки бывают разные

Как мы уже говорили, ссылки в XLink делятся на простые и расширенные. Естественно, это не единственная классификация ссылок, но чаще всего применяется именно такое разделение.

Итак, простые ссылки (simple links) являются прямыми наследниками ссылок из HTML, так хорошо нам знакомых. В силу их простоты, на них не накладывается особых ограничений по внутреннему синтаксису. Нам достаточно указать их тип и ресурс, на который она ссылается.

С расширенными ссылками (extended links) все немного сложнее. В связи с тем, что их функциональность намного выше, чем у простых ссылок, на синтаксис расширенных ссылок накладываются некоторые правила. Итак, прежде всего, необходимо указать тип при помощи атрибута `xlink:type`, использовав для него фиксированное значение `extended`.

Каждая расширенная ссылка является элементом, который включает в себя другие элементы в качестве содержимого. А уже с помощью включаемых в расширенную ссылку субэлементов мы управляем действием нашей ссылки. Рассмотрим пример.

```
<!ELEMENT exlink (source)+>
<!ELEMENT cource ANY>
<!ATTLIST exlink
    xlink:type CDATA #FIXED "extended">
<!ATTLIST source
    xlink:type CDATA #FIXED "locator"
    xlink:href CDATA #REQUIRED>
. . . . . .
<exlink>
<source xlink:href="http://www.site1.com">First site</source>
<source xlink:href="http://www.site2.com">Second site</source>
<source xlink:href="http://www.site3.com">Third site</source>
</exlink>
```

В этом примере мы объявляем расширенную ссылку `exlink`, в которую входят один или несколько субэлементов `source` типа `locator`. Наша расширенная гиперссылка позволяет адресовать сразу три внешних ресурса. Именно для этого и применяются элементы типа `locator`.

Помимо подобных указателей, в расширенную гиперссылку могут входить элементы типа `arc`, которые устанавливают правила прохождения гиперссы-

лок, элементы типа `title`, позволяющие устанавливать отображаемые метки для расширенных гиперссылок, и элементы типа `resource`, адресующие внутренние ресурсы для нашей гиперссылки. Субэлементы этого типа могут входить поодиночке или группами в каждую расширенную гиперссылку. Рассмотрим пример, включающий в себя все эти типы элементов.

```
<!ELEMENT courseload ((tooltip|person|course|gpa|go)*)>
<!ATTLIST courseload
  xlink:type      (extended)      #FIXED "extended"
  xlink:role      CDATA           #IMPLIED
  xlink:title     CDATA           #IMPLIED>
<!ELEMENT tooltip ANY>
<!ATTLIST tooltip
  xlink:type      (title)         #FIXED "title"
  xml:lang        CDATA           #IMPLIED>
<!ELEMENT person EMPTY>
<!ATTLIST person
  xlink:type      (locator)       #FIXED "locator"
  xlink:href      CDATA           #REQUIRED
  xlink:role      CDATA           #IMPLIED
  xlink:title     CDATA           #IMPLIED
  xlink:label     NMTOKEN         #IMPLIED>
<!ELEMENT course EMPTY>
<!ATTLIST course
  xlink:type      (locator)       #FIXED "locator"
  xlink:href      CDATA           #REQUIRED
  xlink:role      CDATA           #FIXED
"http://www.example.com/linkprops/course"
  xlink:title     CDATA           #IMPLIED
  xlink:label     NMTOKEN         #IMPLIED>
<!ELEMENT gpa ANY>
<!ATTLIST gpa
  xlink:type      (resource)      #FIXED "resource"
  xlink:role      CDATA           #FIXED
"http://www.example.com/linkprops/gpa"
  xlink:title     CDATA           #IMPLIED
  xlink:label     NMTOKEN         #IMPLIED>
<!ELEMENT go EMPTY>
<!ATTLIST go
  xlink:type      (arc)           #FIXED "arc"
```

```
  xlink:arcrole   CDATA           #IMPLIED
  xlink:title     CDATA           #IMPLIED
  xlink:show      (new
                  |replace
                  |embed
                  |other
                  |none)          #IMPLIED
  xlink:actuate   (onLoad
                  |onRequest
                  |other
                  |none)          #IMPLIED
  xlink:from      NMTOKEN         #IMPLIED
  xlink:to        NMTOKEN         #IMPLIED>
```

А использовать это объявление можно при помощи следующего кода:

```
<courseload xlink:title="Course Load for Pat Jones">
  <person
    xlink:href="students/patjones62.xml"
    xlink:label="student62"
    xlink:role="http://www.example.com/linkprops/student"
    xlink:title="Pat Jones" />
  <person
    xlink:href="profs/jaysmith7.xml"
    xlink:label="prof7"
    xlink:role="http://www.example.com/linkprops/professor"
    xlink:title="Dr. Jay Smith" />
  <!-- more remote resources for professors, TAs, etc. -->
  <course
    xlink:href="courses/cs101.xml"
    xlink:label="CS-101"
    xlink:title="Computer Science 101" />
  <!-- more remote resources for courses, seminars, etc. -->
  <gpa xlink:label="PatJonesGPA">3.5</gpa>
  <go
    xlink:from="student62"
    xlink:to="PatJonesGPA"
    xlink:show="new"
    xlink:actuate="onRequest"
    xlink:title="Pat Jones's GPA" />
  <go
```

```
    xlink:from="CS-101"
    xlink:arcrole="http://www.example.com/linkprops/auditor"
    xlink:to="student62"
    xlink:show="replace"
    xlink:actuate="onRequest"
    xlink:title="Pat Jones, auditing the course" />
  <go
    xlink:from="student62"
    xlink:arcrole="http://www.example.com/linkprops/advisor"
    xlink:to="prof7"
    xlink:show="replace"
    xlink:actuate="onRequest"
    xlink:title="Dr. Jay Smith, advisor" />
</courseload>
```

Это достаточно сложный пример, и его необходимо рассмотреть детально. В начальном блоке мы объявляем головной элемент `courseload`, который является расширенной гиперссылкой. В него могут входить один или несколько субэлементов, каждый из которых мы объявляем несколько позже.

Мы используем три атрибута для расширенной гиперссылки, такие как `xlink:type`, `xlink:role` и `xlink:title`. Атрибут `xlink:type` мы уже знаем, а остальные мы будем тщательно рассматривать в следующих частях этой главы. Пока что можно сказать, что атрибут `xlink:role` позволяет задавать роль ссылки, а атрибут `xlink:title` — ее заголовок.

Теперь рассмотрим субэлементы, используемые в гиперссылке. Элемент `tooltip` носит тип `title`, и у него существует еще один атрибут `xml:lang`, при помощи которого указывается язык элемента.

Следующий субэлемент — `person`. Он является пустым, как это видно из его объявления. Общий тип этого элемента, входящего в качестве составной части в расширенную гиперссылку, объявлен как `locator`. Локаторами обычно называют фрагменты расширенных гиперссылок, которые идентифицируют ресурс. На этот ресурс указывает или может указывать расширенная гиперссылка. По сути, локаторы являются адресами ресурсов.

В качестве необязательных атрибутов данного элемента используются `xlink:role`, `xlink:title` и `xlink:label`.

Данный субэлемент в нашем примере используется для указания на ресурс, который содержит информацию о каком-либо человеке, как видно из кода значимого содержимого XML-документа. Если быть более точным, то этим элементом адресуется какой-либо студент или преподаватель, так как наш пример направлен именно на предметную область высшего образования.

Следующий субэлемент, входящий в состав расширенной ссылки, предназначен для указания на ресурс, в котором содержится описание какого-либо обучающего курса. Из этих курсов складывается учебная программа студентов. В субэлементе `course`, который, как и предыдущий, имеет тип `locator`, мы помимо обязательного атрибута `xlink:href`, объявляем набор дополнительных атрибутов. Как и в предыдущем субэлементе, это атрибуты `xlink:role`, `xlink:title` и `xlink:label`. Но есть и отличие. Атрибут `xlink:role` теперь имеет модификатор `#FIXED` и значение по умолчанию, которым является URL некоего ресурса. Данный атрибут описывает так называемую "роль" субэлемента, а по этому URL находится ресурс с ее описанием.

Следующий субэлемент `gpa` входит в расширенную ссылку в качестве ресурса, о чем говорит значение `resource`, приписанное фиксированному атрибуту `xlink:type`. В данном случае этот ресурс описывает некие свойства того объекта, на который указывает расширенная ссылка. Вообще говоря, ресурсами называют некие сервисы или объекты, на которые указывает гиперссылка. Но в данном случае мы видим пример так называемого "участвующего ресурса", который является частью ссылки.

В субэлемент `gpa` мы, как обычно, включили набор из трех дополнительных атрибутов.

Нам осталось рассмотреть объявление последнего субэлемента, входящего в состав расширенной ссылки. Он носит наименование `go` и применяется для описания процесса прохождения данной гиперссылки. Так как вся информация о процессе прохождения гиперссылки задается обычно при помощи атрибутов, то данный элемент объявлен пустым (`EMPTY`). Для этого субэлемента атрибут `xlink:type` имеет значение `arc` (контейнер). В подобных элементах хранится некая служебная информация о расширенной гиперссылке.

В объявлении этого элемента мы применяем специализированный атрибут `xlink:arcrole`, который, по существу, является дубликатом атрибута `xlink:role`, но при этом используется для элементов типа `arc`. Необязательный атрибут `xlink:title` нам уже знаком по объявлениям предыдущих субэлементов. А вот остальные атрибуты требуют отдельного рассмотрения.

Атрибут перечислимого типа `xlink:show` определяет принцип отображения ресурса, на который указывает гиперссылка в XML-документе. Предустановленные значения, которые могут быть приписаны этому атрибуту, мы рассмотрим в нижеследующих частях.

Атрибут `xlink:actuate` тоже имеет перечислимый тип. Но он позволяет указывать момент перехода по ссылке. В HTML переход осуществлялся только тогда, когда пользователь производил щелчок по гиперссылке. В XML мы можем обойти это ограничение как раз при помощи этого атрибута расширенной гиперссылки.

Атрибуты `xlink:from` и `xlink:to` позволяют указывать информацию о том, откуда и куда осуществляется переход, т. е. это некие аналоги кнопок **Back** (Назад) и **Forward** (Вперед) обычных HTML-браузеров. При помощи этих атрибутов обычно формируются кольца гиперссылок.

Теперь, после того, как мы рассмотрели объявление этой расширенной ссылки, можно увидеть, как она используется в XML-документе. С ее помощью выстраивается целая система взаимоотношений в объектной области обучения в институте.

Сначала мы используем открывающий тег `<courseload>` с параметром `xlink:title`. В него мы вкладываем пустой тег `person`, который описывает студента Pat Jones, с идентификационным номером 62. Атрибут этого тега `xlink:href` указывает на XML-документ с URL `students/patjones62.xml`. При этом роль данного экземпляра элемента `person` описывается документом, который находится по адресу **http://www.example.com/linkprops/student**.

Затем мы создаем еще один тег `person`, в котором описываем профессора Jay Smith. По аналогии с предыдущим тегом `person` можно легко разобраться в атрибутах данного экземпляра элемента.

После этих двух тегов мы устанавливаем ссылку на курс, входящий в программу обучения данного студента, при помощи тега `course`.

При помощи тега `gpa` мы указываем среднюю оценку нашего студента по этому курсу. Следует обратить внимание на то, что этот элемент не является пустым, поэтому в коде должны присутствовать как открывающий, так и закрывающий теги `gpa`.

А после этого тега мы вписываем три тега `go`, которые устанавливают связи между всеми сущностями нашей предметной области. Первый тег устанавливает прохождение ссылки от студента к его средней оценке, второй — от курса к студенту, указывая при этом, что данный курс проходит наш студент Pat Jones, третий — от студента к его научному руководителю, которым является все тот же Jay Smith.

Для двух последних экземпляров элементов `go` принудительно указывается роль переходов при помощи атрибута `xlink:arcrole`.

Теперь, после того, как мы рассмотрели этот достаточно объемный пример, мы уже готовы к созданию собственных расширенных ссылок, которые, как мы убедились, позволяют связывать воедино множество ресурсов, указывая при этом их отношение друг к другу, правила адресации и переходов между ними.

А сейчас перейдем к более тщательному описанию элементов, входящих в расширенные гиперссылки.

Локальный ресурс

Локальным ресурсом называют тот ресурс, который является частью самой расширенной гиперссылки. Он реализуются при помощи субэлементов,

включаемых в состав расширенной гиперссылки, для которых атрибут `xlink:type` выставлен в значение `resource`. В нашем примере такой элемент носил наименование `gpa`.

В подобные элементы мы можем помещать некоторое содержимое (в нашем примере этот элемент тоже не был пустым), которое в общем случае может и не относиться к самой ссылке, т. е. это содержимое может и не быть служебной информацией, необходимой для обработки ссылки.

Тем не менее, элементы подобного типа могут иметь семантические атрибуты, такие как `xlink:role` и `xlink:title`, или атрибут `xlink:label`, управляющий прохождением ссылки.

Внешние ресурсы

Внешний ресурс, как и локальный, является частью расширенной ссылки, но при этом позволяет гиперссылке адресовать другие XML-документы и иные ресурсы.

Объявляются внешние ресурсы при помощи элементов, у которых атрибут `xlink:type` установлен в значение `locator`.

Подобные элементы могут иметь любое содержание. Рассмотрим пример создания группы расширенных ссылок, каждая из которых указывает на внешний ресурс.

```
<!ELEMENT exlink (source)+>
<!ELEMENT cource ANY>
<!ATTLIST exlink
    xlink:type CDATA #FIXED "EXTENDED">
<!ATTLIST source
    xlink:type CDATA #FIXED "LOCATOR"
    xlink:href CDATA #REQUIRED>
. . . . . .
<exlink>
<source xlink:href="http://www.site1.com">First site</source>
<source xlink:href="http://www.site2.com">Second site</source>
<source xlink:href="http://www.site3.com">Third site</source>
</exlink>
```

Как мы можем видеть, между открывающими и закрывающими тегами `source` находится некий тег, который и идентифицирует для пользователя какой-либо внешний ресурс.

Элементы-локаторы обязаны иметь в своем объявлении атрибут `xlink:href`, который подлежит обязательному заполнению в каждом теге этого элемента.

Подобные элементы могут иметь атрибуты `xlink:role`, `xlink:title` и `xlink:label`.

Локаторы указывают на удаленные ресурсы при помощи URI (Universal Resource Identifier), который расширяет понятие URL. URI какого-либо ресурса формируется из его URL и выражения языка XPointer, который был создан в рамках XML специально для указания местоположения какого-либо ресурса.

Правила прохождения ссылок

В HTML у нас было только одно правило прохождения ссылки. При щелчке на гиперссылке, размещенной в теле документа, в браузер загружается ресурс, на который указывает ссылка. Единственным "глотком свободы" была возможность указывать окно просмотра или фрейм, в который предстояло загружать связанный документ.

XML дал нам новые возможности и в этой области. Теперь мы можем полностью управлять процессом перехода по ссылке, или даже, по нескольким ссылкам сразу, если использовать расширенную ссылку на группу ресурсов.

Подобные правила прохождения (traversal rules) оформляются в виде элементов, входящих в состав расширенной ссылки. Их атрибуту `xlink:type` обязательно приписывается значение `arc`.

Подобные элементы могут иметь любое содержимое, но обычно эта возможность не используется. Правила прохождения являются сугубо служебной информацией, поэтому далеко не всегда есть смысл их отображать в XML-браузере.

Элементы типа `arc` могут иметь атрибуты прохождения, такие как `xlink:from` и `xlink:to`, атрибуты поведения, такие как `xlink:show` и `xlink:actuate` и семантические атрибуты `xlink:arcrole` и `xlink:title`.

Атрибуты прохождения характеризуют пары ресурсов, между которыми осуществляется переход. При этом в качестве значений атрибутов `xlink:from` и `xlink:to` записываются значения атрибутов `xlink:label` соответствующих ресурсов.

Атрибуты поведения описывают действия XML-приложения, которое обрабатывает данный XML-документ, в процессе перехода по ссылке. Стандартные варианты поведения XML-приложений и XML-браузеров мы рассмотрим ниже, когда придет пора обсудить сами атрибуты поведения.

В нашем большом примере, который мы рассматривали в разделе "Ссылки бывают разные", в качестве элементов, управляющих правилами прохождения расширенной ссылки, мы использовали элементы `go`. Но в том примере их роль и механизм действия видны не так уж четко. Приведем дополнительный пример.

```
<extendedlink xlink:type="extended">
  <loc xlink:type="locator" xlink:href=".  " xlink:label="parent"
xlink:title="p1" />
  <loc xlink:type="locator" xlink:href="..." xlink:label="parent"
xlink:title="p2" />
  <loc xlink:type="locator" xlink:href="..." xlink:label="child"
xlink:title="c1" />
  <loc xlink:type="locator" xlink:href="..." xlink:label="child"
xlink:title="c2" />
  <loc xlink:type="locator" xlink:href="..." xlink:label="child"
xlink:title="c3" />
  <go xlink:type="arc" xlink:from="parent" xlink:to="child" />
</extendedlink>
```

В данном случае мы при помощи элемента `go`, задающего правила прохождения расширенной гиперссылки, позволяем осуществлять переходы от любого элемента-локатора с меткой `parent` к любому элементу-локатору с меткой `child`. При этом все остальные направления перехода заблокированы. Таким образом и осуществляется управление переходами между ресурсами, входящими в сферу действия расширенной гиперссылки.

Если бы мы удалили из тега `go` параметр `xlink:from`, оставив только `xlink:to`, то разрешили бы переход от всех элементов-локаторов ко всем элементам, у которых атрибут `xlink:label` имеет значение `child`.

Есть некоторое ограничение, которое накладывается на подобные элементы. У каждого arc-элемента должна быть уникальная пара значений атрибутов `xlink:from` и `xlink:to`. Все остальное — в ваших руках.

Идентифицирующие элементы ссылок

Расширенная гиперссылка всегда может быть идентифицирована неким образом. Для этого применяется специализированный элемент, входящий в ее состав. Подобные элементы имеют значение `title` у атрибута `xlink:type`. Необходимо отличать элементы типа `title` от одноименных атрибутов. Атрибуты применяются для идентификации какого-либо элемента для самого XML-процессора, а элемент `title` содержит информацию, которая предназначена для пользователя. Иначе говоря, с помощью подобных элементов организуется отображение информации о расширенной гиперссылке, ресурсе или локаторе в XML-браузере. Изначально подобные элементы задумывались для нужд локализации XML-документов. Я бы хотел обратить внимание на то, что теперь мы локализуем не только программные приложения, но и документы. Связано это, естественно, с тем, что XML-документы имеют как минимум равную ценность контента и механизмов работы, заложенных в структуру документа.

В качестве примера можно привести следующую конструкцию:

```
<!ELEMENT advisorname (name)>
<!ATTLIST advisorname
  xlink:type      (title)         #FIXED "title"
  xml:lang        CDATA           #IMPLIED>
```

Атрибут типа элемента

Мы переходим к рассмотрению всех предопределенных атрибутов, которые используются в элементах-ссылках.

Первым из них является, естественно, атрибут с наименованием `xlink:type`. Чаще всего с этим атрибутом используется модификатор `#FIXED`, что, впрочем, не является необходимым условием. У данного атрибута есть список предустановленных значений. Иные значения не будут распознаваться XML-генератором, и, следовательно, данный элемент не будет считаться ссылкой.

Атрибут `xlink:type` позволяет задавать и обозначать тип ссылки, которая будет создаваться данным элементом. Для этого атрибута могут быть использованы значения `simple`, `extended`, `locator`, `arc`, `resource`, `title` или `none`.

Механизм действия всех значений, кроме последнего, мы уже рассмотрели в предыдущих частях. Значение `none` применяется в тех случаях, когда в XML-документе данный элемент будет использоваться не только как ссылка, но и в качестве обычного элемента. Объявление подобного элемента будет выглядеть следующим образом:

```
<!ATTLIST commandname
  xlink:type      (simple|none)   #REQUIRED
  xlink:href      CDATA           #IMPLIED>
```

В этом примере мы объявляем, что в тексте XML-документа элемент `commandname` может использоваться либо в качестве простой ссылки, либо как обычный элемент разметки содержимого.

Атрибут `xlink:type` является обязательным для всех элементов, которые будут действовать в качестве ссылок.

Атрибут целеуказания

Еще одним обязательным атрибутом для всех элементов-ссылок является атрибут целеуказания с именем `xlink:href`. В качестве значения для этого атрибута мы указываем URI того ресурса, на который указывает данная ссылка. Кстати, полное описание спецификации URI содержится в документе RFC 2396, который, как и все RFC, доступен через Интернет.

Мы уже не раз рассматривали примеры декларирования и использования данного атрибута, но эта часть не может считаться полной, если мы все-таки не поместим в нее хоть какой-нибудь пример.

```
<!ATTLIST simplelink
  xlink:href      CDATA          #REQUIRED
```

Так как этот атрибут является обязательным для любого элемента, который используется в качестве ссылки, то в его объявлении используется модификатор `#REQUIRED`.

Семантические атрибуты

Существует группа необязательных атрибутов, которые позволяют неким образом идентифицировать ту или иную ссылку, тот или иной экземпляр соответствующего элемента. Их принято называть семантическими атрибутами. В эту группу входят атрибуты `xlink:title`, `xlink:role` и `xlink:arcrole`.

Значение атрибута `xlink:title` представляет собой символьную информацию, которая используется для дополнительной, чаще всего визуальной, идентификации экземпляра элемента-ссылки. Если говорить более понятным языком, это значение является описанием ссылки, которое показывается пользователю, т. е. попросту подсказка. Чаще всего XML-браузеры отображают эту подсказку, если пользователь наводит курсор мыши на отображение экземпляра элемента-ссылки в окне просмотра.

Атрибуты `xlink:role` и `xlink:arcrole` идентичны по своему значению и механизму действия. Единственное их отличие в том, что атрибут `xlink:role` применяется для элементов типа `simple`, `extended`, `locator` и `resource`, а атрибут `xlink:arcrole` — для элемента типа `arc`.

Значения этих атрибутов обязаны быть указателями на некие ресурсы, которые описывают роль данных ссылок, т. е. иметь формат URI.

Все три вышеописанных атрибута являются необязательными. Поэтому при их декларировании мы применяем модификатор `#IMPLIED`.

Поведенческие атрибуты

В эту группу входят два атрибута, которые управляют поведением XML-приложения, в том числе и XML-браузера, при отображении ресурса ссылки. К поведенческим атрибутам относятся атрибуты `xlink:show` и `xlink:actuate`. Эти атрибуты применяются к элементам-ссылкам, имеющим тип `simple` или `arc`.

Ресурс, на который указывает ссылка, может быть обработан несколькими способами. Выбор метода отображения ресурса зависит от значения атрибу-

та xlink:show. У этого атрибута существует пять фиксированных значений, которые регулируют обработку и отображение целевого ресурса ссылки. Значение embed указывает на то, что содержимое целевого ресурса необходимо встроить в текущий документ, прямо с того места, где была размещена ссылка. Если нам необходимо целевой ресурс отобразить в новом окне браузера, применяется значение new. А если мы используем значение replace, то содержимое текущего документа будет заменено содержимым целевого ресурса ссылки при ее активации. Значение other мы используем, если наше XML-приложение будет отображать ресурс, на который указывает ссылка, каким-либо иным способом. Описание поведения XML-браузера при использовании этого значения атрибута xlink:show в текущую версию стандарта не вошло, и вряд ли войдет, так как это значение явно добавлено в спецификацию как некий слот для нестандартных реакций. Если же ресурс, на который указывает ссылка, мы вообще не собираемся отображать, используется значение none (но с другой стороны, кому и зачем нужна такая ссылка?).

В XML мы можем также управлять процессом активации ссылок. В HTML ссылка активировалась только пользователем. Переход осуществлялся только в том случае, если удаленный пользователь отдавал команду. В XML мы, в дополнение к этому механизму, можем автоматически активировать ссылку. Для этого используется параметр xlink:actuate, у которого есть четыре предустановленных значения. Значение onRequest указывает на то, что загрузка содержимого целевого ресурса ссылки не будет осуществляться до тех пор, пока пользователь сам явно не отдаст команду. А вот значение onLoad следует использовать в тех случаях, когда мы хотим, чтобы переход по ссылке и загрузка документа произошли автоматически, как только XML-анализатор обнаружит их при загрузке и отображении текущего ресурса. При помощи подобных автоматических ссылок мы можем без особых усилий формировать объемные XML-документы из отдельных небольших частей.

Также мы можем для этого атрибута использовать значения other и none, которые дублируют поведение своих двойников, использовавшихся в атрибуте xlink:show.

Атрибуты прохождения ссылки

Мы уже упоминали пресловутые "правила прохождения ссылки" (traversal rules). Они задают поведение XML-приложения при перемещении по ссылке или целой цепочке ссылок. Для управления этим процессом применяются атрибуты прохождения (traversal attributes).

Атрибут label предназначен для использования в элементах типа resource и locator. Атрибуты to и from могут использоваться только в элементах типа arc. Все атрибуты, входящие в эту группу, являются опциональными,

и их наличие в объявлении элемента-ссылки не требуется, но иногда бывает полезно. Немного ранее мы видели, как эти атрибуты позволяют организовывать группу взаимосвязанных элементов-ссылок.

Атрибут `label`, как мы видели, имеет тип `NMTOKEN`, и благодаря этому позволяет идентифицировать тот или иной экземпляр элемента, например, при помощи значения атрибута `ID`.

Атрибуты `from` и `to` содержат значения атрибутов `label` того элемента, откуда производится переход по ссылке, и элемента, куда осуществляется переход, соответственно.

Теперь мы знаем, как объявляются элементы ссылки, и какие у них бывают атрибуты. Кстати, об атрибутах. Мы перечислили лишь обязательные и предустановленные атрибуты для элементов-ссылок. Не следует забывать, что все ссылки являются обычными (ну, почти обычными) элементами, и, следовательно, мы всегда можем добавить туда те атрибуты, которые нам нужны. Это же XML! Теперь нас почти ничто не ограничивает. Мы можем создавать настолько сложные иерархические структуры, насколько захотим. Главное, не забывать их документировать.

Итак, мы рассмотрели правила и методы создания гиперссылок в XML. Но это еще далеко не все. Каждая гиперссылка должна куда-то вести (хороший девиз для Интернета, правда?). Теперь помимо обычных URL, которые указывают на конкретные файлы, мы можем использовать URI (Universal Resource Identifier), создающиеся при помощи языка целеуказания для XML — XPointer.

XPointer

XPointer (XML Pointer Language), как мы уже говорили, является особым языком, описывающим местонахождение тех или иных ресурсов. С одной стороны, у нас уже есть механизм, который позволяет описывать местонахождение ресурсов. Мы называем его URL. Зачем понадобилась его модификация?

Выражения языка XPointer позволяют создавать так называемые URI (Universal Resource Identifier), которые являются некими надстройками над URL. В связи с тем, что HTML не производит контроля целостности ссылок, мы вынуждены раз за разом натыкаться на оборванные гиперссылки, ведущие в никуда. Со ссылками, адресованными по технологии XPointer, подобная ситуация будет встречаться намного реже. Язык XPointer позволяет не только адресовать внешние документы, но и максимально точно описывать местонахождение отдельных частей внутри какого-либо документа путем создания некоей иерархической структуры. При этом можно создавать ссылки не только на какие-либо поименованные закладки, как

в HTML, но и на некое содержание. Например, мы можем указать ссылку на конкретное слово, экземпляр элемента или значение атрибута в XML-документе.

XPointer является надстройкой над языком XPath. В отличие от него XPointer позволяет еще адресовать отдельные экземпляры элементов и целые разделы, умеет обрабатывать текстовые строки и устанавливать точку перехода в зависимости от результата анализа текстовой строки, использовать идентификаторы URI без ограничений в XML-документе.

Основные правила

Итак, сначала — несколько фактов, которые указаны в спецификации XPointer. Язык XPointer позволяет адресовать документы и их фрагменты, которые являются XML-документами. Если говорить официальным языком, эти документы должны иметь тип `text/xml` или `application/xml`. Все выражения XPointer должны записываться с использованием символов Unicode. Но если их не хватает (хотя трудно представить ситуацию, когда для кодирования текстовой информации не хватает набора символов Unicode) или нет возможности использовать их напрямую, мы можем использовать так называемые Escape-последовательности. На самом деле, они не являются истинными Escape-последовательностями, как команды для принтеров, но, тем не менее, они так называются в спецификации. Будем придерживаться официальной терминологии. В XML мы их называли символьными сущностями. Так, мы не можем напрямую использовать в выражениях XPointer символы больше-меньше (которые, кстати, в англоязычной терминологии называют брекетами) и символ амперсанта. Для них могут использоваться стандартные символьные сущности, которые мы рассмотрели в предыдущих главах. Но мы можем представить любой символ, входящий в выражения языка XPointer, в виде Escape-последовательности. Для этого ставится символ процента, а затем номер кода символа Unicode в шестнадцатеричном виде. Другими словами, правила создания Escape-последовательностей полностью совпадают с правилами построения символьных сущностей.

Теперь рассмотрим основные термины, которые используются в XPointer.

Субресурс (sub-resource) — основная единица адресации языка XPointer. Это обычно часть целого XML-ресурса, такая, как отдельная глава или абзац. Именно для адресации подобных мелких частей и предназначен XPointer.

Набор указателей (location-set) — упорядоченный список точек адресации XML-документа, в который входят все адресуемые суб-ресурсы различных типов.

Точка (point) — место в XML-документе, где и находится адресуемый субресурс.

Интервал (range) — часть содержимого XML-документа, расположенная между двумя точками, которые адресованы при помощи выражений XPointer.

Именно в этих терминах ведется объявление конструкций XPointer.

Теперь разберем типы ошибок, которые распознает XML-процессор при анализе выражений языка XPointer.

Синтаксическая ошибка (syntax error) возникает в том случае, если выражение XPointer содержит конструкцию, не поддерживающую какое-либо правило оформления выражений XPointer.

Ошибка адресации (resource error) возникает, если синтаксически правильное выражение XPointer указывает на несуществующий XML-документ, т. е. оборванные ссылки считаются ошибками, распознаваемыми и обрабатываемыми на уровне XML-процессора, на уровне самого engine.

Ошибка адресации субресурса (subresource error) возникает, если синтаксически правильное выражение XPointer указывает на существующий XML-документ, но конкретный субресурс, на который и должен осуществляться переход по ссылке, отсутствует.

Абсолютные указатели

Существует несколько ключевых слов, которые позволяют в XPointer адресовать субресурсы в абсолютном, а не относительном порядке. Рассмотрим их.

Ключевое слово `root` указывает на точку местонахождения открывающего тега самого главного элемента, который объявлен как тип документа (`DOCTYPE`).

Ключевое слово `here` является указателем (и, строго говоря, функцией) на текущее местоположение самого указателя, т. е. отсчет будет вестись от самой ссылки. Естественно, адресовать подобным образом можно только те субресурсы, которые находятся в том же XML-документе, что и сама ссылка.

Конструкция `ID(name)` позволяет адресовать элемент в XML-документе, у которого значение атрибута `id` установлено в `name`.

Если мы используем два идентификатора, построенные при помощи средств XPointer, например, для адресации некоего интервала, то мы можем для второго идентификатора использовать ключевое слово `ditto`, которое указывает, что для него начальная точка адресации является такой же, как и для первого идентификатора.

Эти четыре ключевых слова задают первую точку адресации в XML-документе. Уже начиная с них, мы можем выстраивать цепочку уточняющих инструкций при помощи относительных указателей. Вот эти относительные указатели мы и рассмотрим в следующей части.

Относительные указатели

При помощи ключевых слов, которые мы рассмотрим в этой части, мы сможем производить адресацию по иерархии элементов, из которых и состоит содержание XML-документа.

Так как при движении по иерархической структуре нам хотелось бы иметь полную свободу адресования, то и использование относительных указателей немного отличается от абсолютных указателей. Относительные указатели после ключевого слова, указывающего направление перемещения, в круглых скобках содержат список значений параметров, которые позволяют более точно найти необходимый нам субресурс. Примеры подобных указателей мы сможем увидеть в разделе "Относительная адресация". Там будет показаны их структура и механизм применения. В этой части мы лишь рассматриваем ключевые слова, позволяющие создавать подобные указатели.

Ключевое слово `child` дает возможность адресовать все дочерние элементы источника, на который мы адресовали при помощи абсолютной адресации. При этом адресуются не все потомки исходного элемента, а только те, которые находятся на один уровень ниже, т. е. — дети.

Для адресации всех потомков начального элемента поиска мы можем использовать ключевое слово `descedant`. При перемещении относительно ключевого элемента мы уже не учитываем иерархию потомков, а просто идем по содержанию и используем порядковый номер.

Если же нам необходимо пройти по иерархии объектов вверх от ключевого элемента, к его предкам, мы должны использовать ключевое слово `ancestor`.

Если же нас не интересует иерархическая зависимость элементов друг от друга, и мы хотим просто пройти по содержимому XML-документа и отсчитать то или иное количество экземпляров элементов вниз от исходного, то надо использовать ключевое слово `following`.

А для смещения вверх по содержимому XML-документа от начального элемента поиска, можно применить ключевое слово `preceding`. Есть еще два ключевых слова, которые применяются для относительной адресации. Ключевое слово `psibling` используется для указания на "предшествующих братьев", т. е. на элементы, у которых родитель совпадает с родителем исходного элемента адресации, и которые в содержимом XML-документа находятся перед исходным элементом.

Ключевое слово `fsibling` позволяет адресовать "последующих братьев". Иначе говоря, это братья исходного элемента, которые находятся в содержимом XML-документа после исходного адресуемого элемента.

Кроме того, мы всегда можем использовать ключевое слово `ALL`, которое позволяет адресовать все экземпляры элементов, входящих в содержимое

XML-документа. Естественно, при использовании этого ключевого слова мы добавляем ограничения в скобках, как операнд функции, но, так или иначе, использование этого ключевого слова в указателе в качестве результата выдает интервал, в котором располагаются подходящие значения.

Теперь разберем способы задания дополнительных условий поиска для относительных способов адресации. После каждого ключевого слова, задающего тип выбора, мы в круглых скобках задаем уточняющие параметры.

Первый из них — порядковый номер субресурса. Если заданное число является положительным, то отсчет ведется по направлению к концу документа. Если задается отрицательное число, то отсчет идет к началу документа.

Второй дополнительный уточняющий элемент позволяет задавать тип элемента, который мы ищем. Это может быть наименование элемента. Значение `CDATA` указывает на то, что адресоваться будут элементы, содержащие обычный текст. Символ звездочки "`*`" указывает, что мы ищем любой субресурс. Символ точки "." указывает, что мы ищем только элементы.

Затем мы можем указать в скобках дополнительные условия поиска по атрибуту элемента. На третьем месте мы можем записать наименование атрибута, по которому производится поиск, а на четвертом месте — значение этого атрибута.

Естественно, все, о чем мы узнали на протяжении этой главы, пока что выглядит очень голословно, но без примеров мы не обойдемся. Все примеры мы рассмотрим в следующих разделах. Оставайтесь с нами.

Абсолютная адресация

В этой части мы рассмотрим примеры абсолютной адресации субресурсов средствами XPointer.

Для начала возьмем пример некоего XML-документа.

```
<!DOCTYPE body[
<!ELEMENT p (#PCDATA)>
<!ELEMENT group (item*)>
]>
. . . . . .
<body>
<p>First paragraph</p>
<p>Second paragraph</p>
</body>
```

Будем считать, что URL этого XML-файла имеет вид `http://www.site.com/xml/f1.xml`. Если мы в каком-либо файле собираемся

установить ссылку на этот файл, то мы можем использовать следующую конструкцию:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#root()">link </hlink>
```

В этой строке мы в открывающем теге ссылки указываем в качестве значения атрибута `xlink:href` не URL XML-документа, а URI некоего субресурса. Легко увидеть, что после URL подключаемого XML-документа мы ставим знак решетки. А после него уже записываем выражение XPointer. В данном случае мы использовали выражение `root()`, следовательно, будет загружен наш документ из файла f1.xml, а фокус будет установлен на открывающий тег `<body>`, так как именно элемент `<body>` является корневым для нашего документа.

Как мы знаем, в спецификации XML описан предопределенный маркерный атрибут `id`. Этот атрибут позволяет задавать уникальные идентификаторы для каждого экземпляра любого элемента. Естественно, при помощи конструкций XPointer мы можем использовать эти идентификаторы в целях адресации. Рассмотрим соответствующий пример. Предположим, у нас есть XML-документ приблизительно следующего вида:

```
<!DOCTYPE body[
<!ELEMENT p (#PCDATA)>
<!ELEMENT group (item*)>
<!ATTLIST p
    id    ID #REQUIRED>
]>
. . . . . .
<body>
<p id='p1'>First paragraph</p>
<p id='p2'>Second paragraph</p>
</body>
```

Теперь рассмотрим ссылку. Если нам необходимо установить гиперссылку на первый параграф с содержимым 'First paragraph', мы должны будем использовать приблизительно следующую конструкцию:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#ID(p1)">
link on first paragraph</hlink>
```

Среди ключевых слов абсолютной адресации мы видели ключевое слово `here`. Оно позволяет элементу-ссылке адресовать самого себя. Таким образом, если в содержимом XML-документа, который находится, скажем, все в том же файле f1.xml, мы установим экземпляр элемента-ссылки `<hlink xlink:href= "http://www.site.com/xml/f1.xml#here()">link</hlink>`, то гиперссылка будет направлена на этот же экземпляр элемента.

Естественно, сам по себе компонент `here()` практически бесполезен. Он всегда используется в качестве основы для относительной адресации. Вообще, это стандартное правило. Практически всегда элементы абсолютной адресации являются базой, от которой ведут отсчет элементы относительной адресации. Они потому и называются относительными, что указывают на элементы относительно некоей заранее установленной точки.

Относительная адресация

После того, как мы установили точку отсчета при помощи элементов абсолютной адресации, необходимо произвести более точное "целеуказание" при помощи средств относительной адресации. Их мы рассматривали в разделе "Относительные указатели". В этом разделе мы увидим примеры, которые разъясняют действие каждого ключевого слова, относящегося к относительным указателям.

Как мы видели, относительные указатели позволяют адресовать достаточно отдаленные элементы, находящиеся в самых различных иерархических отношениях друг с другом. Поэтому нам потребуется создать достаточно сложный документ. Вот как он будет выглядеть.

```
<!DOCTYPE body [
<!ELEMENT list (item*)>
<!ELEMENT item (#PCDATA|part)>
<!ELEMENT part (#PCDATA)>
<!ATTLIST item
    id ID #REQUIRED>
<!ATTLIST part
    id ID #REQUIRED>
]>
. . . . . .
<body>
<list id='l1'>
<item id='i1'>
<part id='p1'>Part 1</part>
<part id='p2'>Part 2</part>
</item>
<item id='i2'>Second item</item>
</list>
</body>
```

Теперь, когда у нас есть XML-документ с некоей иерархической структурой, мы можем начать рассмотрение выражений XPointer, использующих относительные указатели.

Для начала попробуем установить указатель на второй экземпляр дочернего элемента корневого элемента body. Для этого мы установим указатель на корневой элемент при помощи абсолютного указателя root, а затем используем относительный указатель child. При этом указатели отделяются друг от друга символами точки. В конечном счете, это будет выглядеть следующим образом:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#root().child(2)">
link to second item</hlink>
```

Эта гиперссылка указывает на элемент item с идентификатором i2.

Теперь мы попробуем показать, как на один и тот же элемент можно ссылаться при помощи разных конструкций XPointer. В качестве цели ссылки мы выберем элемент part с идентификатором p2. Мы можем адресовать его как при помощи последовательных приближений относительных указателей child, так и посредством одного-единственного относительного указателя descedant. Вариант с последовательными приближениями выглядит так:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#root().child(1).
child(1).child(2)">link to part 2</hlink>
```

Но мы можем использовать более простой способ адресации при помощи относительного указателя descedant, который позволяет адресовать любого наследника. Это будет выглядеть следующим образом:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#root().descedant(4)">
link to part 2</hlink>
```

Есть и еще один вариант. Мы сначала установим абсолютный указатель на элемент item с идентификатором i1, а уже от него двинемся к искомому элементу part. Для этого мы используем следующую конструкцию:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#ID(i1).child(2)">
link to part 2</hlink>
```

Мы рассмотрели относительные указатели, которые смещают точку адресации вниз по иерархии элементов с соблюдением правил наследования. Теперь мы увидим, как можно смещаться по этой иерархии вверх, против правил наследования. Для этого применяется ключевое слово ancestor.

В этом примере мы будем смещаться от самого глубокого элемента part к элементу body. Конечно, для того, чтобы установить точку адресации на элемент body, мы могли бы напрямую использовать абсолютный указатель root(), но сейчас у нас несколько иная цель. Нам всего лишь надо продемонстрировать правила применения выражений XPointer. Итак, для установки гиперссылки на элемент body относительно элемента part мы должны использовать приблизительно следующую конструкцию:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#ID(p2).ancestor(3)">
  link to root
</hlink>
```

В нашем примере есть элемент `part`, который находится в самом низу иерархии элементов. Экземпляры этого элемента являются "братьями" друг для друга, так как у них есть общий родитель — экземпляр элемента `item` с идентификатором `i1`. Рассмотрим примеры адресования подобных "братских" элементов.

Так как у нас есть два ключевых слова, которые позволяют указывать на предшествующего и на последующего "брата", то и примеров будет два. В качестве базовой точки отсчета мы будем использовать поочередно оба экземпляра элемента `part`.

Для начала мы укажем на последний экземпляр элемента `part` с идентификатором `p2`. Мы можем использовать следующую конструкцию:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#ID(p1).fsibling(1)">
  link to part 2
</hlink>
```

Адресация от первой части ко второй может производиться при помощи следующей конструкции:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#ID(p2).psibling(1)">
  link to part 1
</hlink>
```

Если же у нас возникает необходимость произвести относительную адресацию вне зависимости от иерархических отношений элементов, мы можем использовать ключевые слова `following` и `preceding`.

Итак, для установки гиперссылки на элемент с идентификатором `i2`, воспользуемся следующей конструкцией:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#root().following(4)">
  link to part 2
</hlink>
```

Мы можем также установить абсолютный указатель на элемент, находящийся достаточно близко к концу документа, и от него уже двинуться вверх по содержимому по направлению к корневому элементу. А точка назначения гиперссылки останется такой же, как и в предыдущем примере. Это будет выглядеть следующим образом:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#ID(i1).preceding(1)">
  link to part 2
</hlink>
```

Итак, мы рассмотрели примеры относительной адресации. Мы видели, что в выражениях XPointer сначала записываются абсолютные указатели, а уже на их основе действуют относительные указатели. Но ведь нас никто не ограничивает в количестве последовательных приближений. После установки одного относительного указателя мы можем добавить еще один. И еще.

Серия последовательных приближений позволяет создавать указатели, максимально приближенные к семантической структуре сколь угодно разветвленных документов.

Адресация интервалов

Рассмотренные нами примеры позволяли устанавливать гиперссылку на тот или иной семантический субресурс XML-документа. Этот субресурс, по сути, являлся лишь единичной точкой входа в общий ресурс. Но мы можем адресовать не только единичные элементы, но и целые фрагменты XML-документов. Фрагменты тоже считаются субресурсами.

Ссылка на некий фрагмент производится при помощи двух выражений XPointer. При этом в качестве результата действия гиперссылки выдается блок XML-документа, находящийся между двумя точками, на которые и указывают выражения XPointer. Естественно, первое выражение должно адресовать элемент, находящийся ближе к началу документа, нежели элемент, адресуемый вторым выражением. В ином случае XML-процессор выдает сообщение об ошибке.

В спецификации XPointer субресурс в виде интервала называется `range`. Создание ссылки на подобный ресурс производится при помощи ключевого слова `range-to`.

Самым простым примером установки гиперссылки на некий интервал можно считать следующее выражение:

```
<hlink xlink:href="http://www.site.com/xml/f1.xml#id("chap1")/
range-to(id("chap2"))">link to chapter 1</hlink>
```

Эта гиперссылка ориентирована на фрагмент XML-документа, который находится между элементами с идентификаторами `chap1` и `chap2`. Как видно из приведенного примера, `range-to` является обычной функцией, для которой в качестве параметра передается конечная точка интервала. При этом мы можем в качестве параметра функции передавать не только абсолютные указатели, как в нашем примере, но и любые выражения XPointer, в том числе и многоэтажные конструкции с использованием любых относительных указателей и строчных функций поиска, которые мы рассмотрим в следующей части.

Существует еще несколько "продвинутых" функций, которые позволяют оперировать в выражениях XPointer с интервалами. Рассмотрим их вкратце.

Функция `range`, как и ее предшественник `range-to`, в качестве результата выдает набор точек входа, или, как мы привыкли говорить, точек адресации. В официальной спецификации XPointer подобные наборы называют `location-set`. Но функция `range` в отличие от `range-to` в качестве парамет-

ра принимает не одну точку входа, а их набор. Декларация этой функции будет выглядеть следующим образом:

```
location-set range(location-set )
```

Данная функция применяется для создания указателя на фрагмент, который будет объединять все точки адресации, указанные в наборе, переданном в функцию в качестве параметра. Таким образом, функция `range` является более общим вариантом функции `range-to`.

Во всех функциях задания параметра мы можем использовать дополнительные функции, которые помогают объявлять стартовую и конечную точку интервала.

Функция `start-point` принудительно задает начальную точку интервала. Параметр и результат имеют тип `point`. Для принудительной установки конечной точки интервала используется функция `end-point`.

Необходимо осознавать, что обе этих функции применяются для управления неким набором точек адресации, т. е. в конечном счете, действия должны производиться над неким операндом с типом `location-set`. При этом, если тип операнда не совпадает с `location-set`, результат применения функции не всегда будет таким, каким он задумывался разработчиком.

Если операнд на самом деле является отдельной точкой входа (`point`), то его значение меняется на значение параметра, переданного функции `start-point` или `end-point`, в зависимости от того, какая функция была применена к операнду.

Если операнд является атрибутом или неким идентификатором (например, псевдонимом сущности), то XML-процессор сообщает о возникновении ошибки.

Адресация строчных субресурсов

При помощи функций, которые мы рассмотрим в этой части, мы сможем осуществлять адресацию, ориентируясь на строковые элементы. По сути, это механизм текстового поиска в документе, встроенный непосредственно в язык XPointer.

Строчные субресурсы являются частным случаем адресуемых интервалов, которые мы обсуждали в предыдущем разделе. Их адресация происходит при помощи функции `string-range`. Рассмотрим несколько примеров применения этой функции.

Для начала мы разберем одну из самых простых конструкций.

```
string-range(//title,"Thomas Pynchon")[17]
```

Данное выражение XPointer разыскивает в тексте XML-документа семнадцатый элемент типа `title`, значение которого равно `Thomas Pynchon`.

На примере этого выражения XPointer мы можем увидеть, какие параметры используются в функции `string-range`.

Первый параметр имеет тип `location-set`. В качестве первого параметра функции `string-range` мы можем передать как наименование элемента, по экземплярам которого будет проводиться поиск, так и результат действия других конструкций XPointer, которые в качестве результата возвращают интервал или набор точек адресования. В нашем случае, мы указали, что поиск будет вестись по экземплярам элемента `title`.

Второй параметр имеет тип `string`. Это, собственно, основной параметр поиска, в котором указывается разыскиваемая подстрока.

После того, как мы указали операцию поиска, в квадратных скобках мы указываем, какой по счету элемент войдет в наш результат поиска, и станет точкой адресации гиперссылки. Мы можем не использовать эту возможность, и тогда в результат войдут все элементы, удовлетворяющие условию поиска.

Но рассмотренные нами параметры не являются единственно возможными для функции `string-range`. Мы можем добавить еще два параметра. Рассмотрим соответствующий пример.

```
string-range(//title,"Thomas Pynchon",8,4)
```

Эта функция возвращает подстроку `"Pync"`, которая начинается с восьмого символа, входящего в строку поиска, и содержит четыре символа. Причем, в данном случае, мы можем получить в качестве результата не одну подстроку, а несколько, если у нас найдется не один элемент типа `title` с содержимым `Thomas Pynchon`.

Как мы помним, первый параметр нашей функции имеет тип `location-set`, поэтому мы можем создавать вложенные конструкции, как в следующем примере.

```
string-range(string-range(//P,"Thomas Pynchon")[17],"P",1,0)
```

Здесь основной функции `string-range` в качестве базы поиска выдаются результаты первичного поиска, т. е. семнадцатый экземпляр элемента `P`, который содержит строку `Thomas Pynchon`, на основе которого и производится уточняющий поиск. Нам требуется найти символ `P`, от которого мы возьмем подстроку, начинающуюся с первого символа и не содержащую ни одного символа. В данном случае мы получаем в качестве результата вырожденный интервал, не содержащий никакого текстового наполнения. В спецификации подобный случай называется *collapsed rang*e.

Адресация элементов

Идеология умных гиперссылок в XML ориентирует составителей XML-документов на создание семантических, а не навигационных ссылок. При помощи относительных указателей мы можем создавать ссылки не на конкретный URL, который может измениться, а напрямую, на искомый

элемент, используя иерархические отношения элементов содержимого XML-документа. В этой части мы рассмотрим все возможные способы адресации отдельных экземпляров элементов содержимого XML-документа.

Для начала рассмотрим способ адресации элемента по значению некоего атрибута. Мы можем задать выражение XPointer, которое адресовало бы только тот экземпляр элемента, у которого значение некоего атрибута совпадало бы с предустановленным значением. Для этого мы можем использовать следующую конструкцию:

```
root().following(2,paragraph,shadow,"none")
```

Этим выражением мы адресуем второй элемент типа `paragraph`, у которого значение параметра `shadow` установлено в `none`.

Как видно из примера, для адресации элементов по значению атрибута используются уже знакомые нам относительные указатели. До сих пор мы применяли всего один параметр для относительных указателей, который давал их порядковый номер. Необходимо напомнить, что этот номер не обязан быть положительным. Если мы укажем отрицательный индекс, то отсчет пойдет к началу содержимого XML-документа.

В качестве второго параметра мы передаем наименование типа элемента, на который мы хотим установить точку адресации. Этот параметр является необязательным, равно как и третий с четвертым параметры.

Третий параметр содержит наименование атрибута, по которому производится уточняющий поиск. Естественно, одного наличия атрибута для уточняющего поиска недостаточно, поэтому в четвертом параметре передается значение данного атрибута, по которому будет производиться поиск.

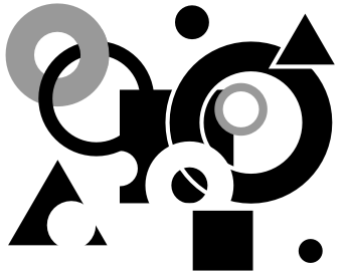
В разделе "Абсолютные указатели" мы рассматривали функцию `here()`, которая применяется для абсолютного позиционирования. Следующий пример показывает ее действие.

```
<hlink href="http://www.site.com/xml/x1.xml#here().following(3)"> link to
second item</hlink>

<list>

<item id="i1">First item</item>

<item id="i2">Second item</item>

</list>
```

В данном примере мы устанавливаем гиперссылку на элемент с идентификатором `i2`. Сначала гиперссылка указывает на саму себя при помощи функции `here()`, а затем посредством условного указателя смещает точку на три элемента вниз, попадая прямо на второй элемент списка.

Мы не упомянули еще одну функцию XPointer. Функция `origin()` указывает на точку, из которой произошла ссылка на данный субресурс. Естественно, эта функция предназначена для XML-приложений, так как браузеры пользуются параметром ссылки `from`.

Глава 6



SOAP

Основы

В следующей главе мы будем рассматривать одну из самых заманчивых и интересных возможностей технологии Microsoft .NET — Web-сервисы, осуществляющие связь со своими клиентами посредством независимых от платформы 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 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 вынашивала идею 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.

Подробно рассматривать идеологию Web-сервисов мы будем в следующей главе, однако для того, чтобы понять, какую роль играет SOAP в их создании, следует провести краткий экскурс уже сейчас.

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

Однако первый свет в конце тоннеля (хотя неизвестно еще, существует ли вообще выход у этого тоннеля) забрезжил с внедрением технологии COM (Common Object Model). Эта объектная модель позволяла создавать различные функциональные объекты, которые предоставляли при помощи специализированных интерфейсов иным программам средства работы с собой. Другими словами, если некий COM-объект мог проверять орфографию в тексте, то 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 мощное приложение в короткие сроки.

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. Причем следует 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 многоплатформенность в Интернете проявляется много ярче, чем в любой другой области компьютерной индустрии. А кросс-платформность никогда не была сильной стороной Microsoft. Однако если необходимо найти решение — скорее всего оно будет найдено. Особенно, если это решение действительно нужно такой крупной корпорации. Специалисты Microsoft его нашли.

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, также, что чаще всего данный протокол используется для передачи HTML-файлов, которые также могут отображаться в любом браузере и в любых условиях, т. е. 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. Так как используется протокол, не зависящий от конкретной платформы, то сервис и клиентская программа тоже не обязаны функционировать в одной и той же операционной системе. Однако есть еще две нерешенных проблемы. Во-первых, сервис должен быть самодокументируемым, т. е. 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 HTTP, но его возможностей явно будет не хватать для взаимодействия программ. Поэтому на роль языка коммуникации был выбран SOAP.

SOAP — это протокол, т. е. 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 из предыдущей главы, 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 неопределенным 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 этом 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 6.1.

Листинг 6.1

```
POST /MyService HTTP/1.1
Host: www.servicesite.com
Content-Type:text/xml; charset="utf-8"
Content-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, которые предназначены именно для этого приложения. После выделения подобных компонентов, приложение обрабатывает их согласно логике, заложенной в него. Если SOAP-сообщение не содержит некоторых обязательных частей, оно может (но не обязано) быть отвергнуто приложением. Однако в том случае, если отсутствие этих частей не является критичным для приложения, оно может продолжить обработку SOAP-запроса. В тех случаях, когда 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 соответствовать 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 применяются 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 ссылки на них.

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-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. В том случае, если в конверте приложение или разработчик размещают дополнительные элементы, эти элементы 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 будут подсказывать приложению, обрабатывающему SOAP-запрос, какие именно функции будут реализованы при помощи данного SOAP-сообщения. Заголовок SOAP-сообщения является прямым потомком конверта, т. е. 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. В том случае, если у сообщения есть заголовок, тело сообщения должно объявляться сразу после заголовка, т. е. 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 правила отображения в заголовке, 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 расширить 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 передает его, он "предполагает", что получатель 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 единичным значением, как в нашем примере.

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

Если обрабатывающее приложение не может правильно распознать элемент заголовка с атрибутом `mustUnderstand`, то оно обязано вернуть SOAP-сообщение с указанием кода возникшей ошибки. Точно также 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 может обрабатываться несколькими приложениями. 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 некий 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 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. Другими словами, приложение, отсылающее SOAP-сообщение, при помощи элементов его тела заставляет принимающее приложение выполнять те или иные действия. Если мы еще раз посмотрим

на листинг 6.1, 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 вот между ними уже располагается некая 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. Проще говоря, он там должен 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 приложение, 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.

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 некоторые свои функции, например, из-за того, что не смогло 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 16 777 216, 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 указываются 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 применяются четыре 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 — графики, звука, видеоклипов и иных бинарных объектов.

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. В том случае, 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 этого типа создан тип `negativeInteger`. Используется для представления отрицательных целых чисел. Данные этого типа представляют собой конечную последовательность десятичных цифр со знаком минуса впереди.

Тип `long` создан на основе типа `integer`. Этот тип 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 значения данного типа 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 этого 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 0 до 429 4967 295 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 как "`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 как "`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. Иначе говоря, 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 применяться 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. Данные этого типа имеют формат "YYYY-MM". Таким образом, чтобы обозначить декабрь двухтысячного года, мы должны записать 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. 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, мы должны 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.

Тип recurringDate предназначается для обработки неких ежегодно повторяющихся дат. Значениями данного типа являются строки формата "MM-DD", т. е., если мы задаем значение "12-06", то мы указываем ежегодно повторяющуюся дату — шестое декабря. Естественно, при помощи величин этого типа чрезвычайно удобно задавать самые различные годовщины, такие, как дни рождения.

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 мы использовали 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 составного 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 читабельности 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:

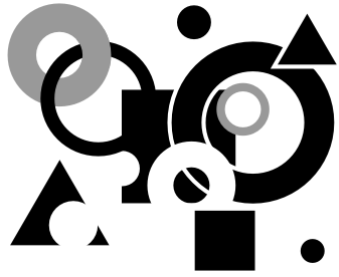
```
<element name="MyArray" type="SOAP-ENC:Array" />
<MyArray SOAP-ENC:arrayType="xsd:int[3]">
<item>1</item>
<item>2</item>
<item>3</item>
</MyArray>
```

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. Теперь, когда мы рассмотрели структуру SOAP, мы можем переходить к рассмотрению процедуры разработки Web-сервисов в рамках концепции Microsoft .NET.

Глава 7



Сервисы ASP

Постановка задачи

Вообще-то идея создания сервисов в Web так или иначе должна была появиться. Она не так уж и нова, но с появлением технологии Microsoft .NET сервисы в Web должны получить большее распространение. Идеологически, сервисы в Web являются наследниками COM-объектов в обычных и распределенных приложениях.

Как мы все помним, COM-объекты предоставляли любому приложению доступ к своим функциональным возможностям при помощи стандартизованных интерфейсов. Любое приложение вместо того, чтобы выполнять некие действия (например, проверку орфографии в написанном пользователем тексте), могло дать задание COM-объекту, созданному специально для соответствующей работы, и тот сам выполнил бы требуемые действия.

Web-сервисы тоже обладают некоей функциональностью, и, что самое важное, связываются с клиентской частью, передают результаты работы и получают команды от клиента по протоколу HTTP. При этом Web-сервис предоставляет клиентским приложениям стандартизованные интерфейсы. А использование независимого от платформы протокола HTTP позволяет создавать один Web-сервис и множество клиентских приложений на самых различных операционных системах.

Web-сервисы могут принимать запросы, основанные на методах `GET` и `POST` протокола HTTP, а также запросы на языке SOAP, который мы рассматривали в предыдущей главе. Структура этого решения достаточно проста. Сервис работает на сервере, и он понимает запросы, отправляемые по протоколу HTTP при помощи одного из его методов или на языке SOAP, и передает клиентскому приложению информацию о выполняемых сервисом действиях на языке XML. Клиентское приложение анализирует полученную информацию о функциональности сервиса и вызывает одну из его функций, пользуясь все тем же языком XML и протоколом HTTP. Сервис

выполняет затребованную клиентом функцию, а результаты ее выполнения передает клиенту. Еще раз повторюсь, указывая на тот факт, что использование в качестве способа общения между сервисом и клиентом языка XML и общепринятого протокола HTTP, позволяет создавать клиентские приложения практически на любой компьютерной платформе. Если как следует поработать, клиентское приложение можно разместить даже на сотовом телефоне.

Но давайте перейдем все-таки к рассмотрению конкретных примеров создания Web-сервисов.

Простейший Web-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** (Файл | Создать | Проект), и в появившемся диалоговом окне **New Project** (Создать проект) в наборе **Templates** (Шаблоны) выделить значок **ASP.NET Web Service** (Web-сервис ASP.NET). Также в поле **Name** (Имя) 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 поместим код нашего Web-сервиса. Он приведен в листинге 7.1.

Листинг 7.1

```
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 входят 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`.

Как видно, создание Web-сервисов не является сложной задачей. Теперь рассмотрим, как он действует. После того, как этот Web-сервис будет откомпилирован, к нему можно будет обратиться даже из браузера. На рис. 7.1 приведен внешний вид Web-страницы, которая генерируется Web-сервером при попытке обращения к Web-сервису.

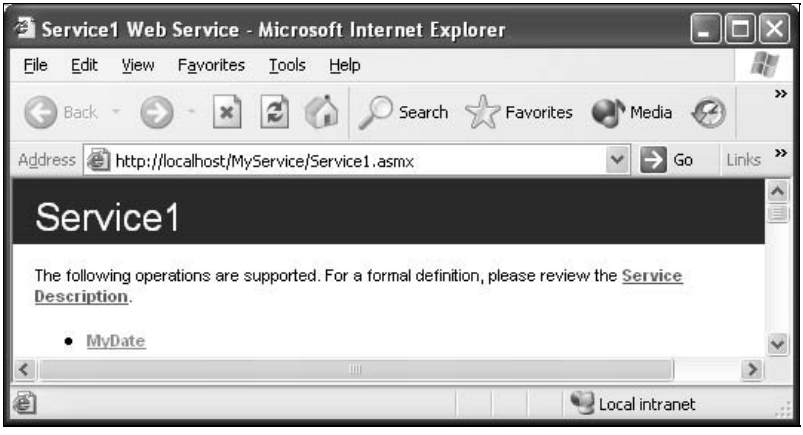


Рис. 7.1. Внешний вид Web-страницы, соответствующей Web-сервису

На этой Web-странице расположена ссылка на формальное описание структуры Web-сервиса, а также перечислены все функции, поддерживаемые этим 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 того, как будет произведен щелчок 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. 7.2.

На этой странице расположено поле ввода, в котором пользователь может указать значение параметра, передаваемого функции, и кнопка **Invoke** (Инициировать), 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, вы сможете увидеть эти инструкции. Следует отметить, что на Web-странице приведены 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. 7.2. 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** (Инициировать), браузер отобразит Web-страницу с результатом работы функции сервиса. Внешний вид этой страницы показан на рис. 7.3.

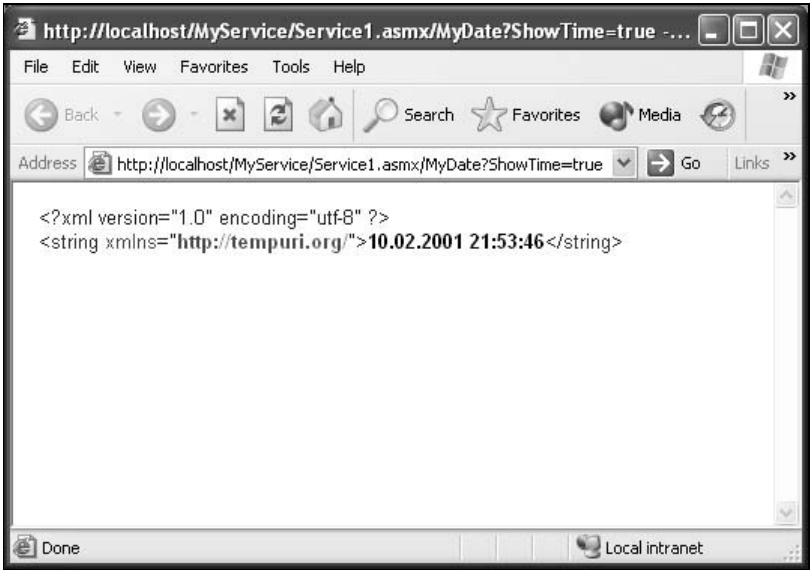


Рис. 7.3. Web-страница с результатом работы функции

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.

Файл WSDL (Web Service Descriptor Language) является обычным XML-файлом, и доступ к нему мы можем получить даже из браузера. Если мы обратим внимание на рис. 7.1, то увидим в верхней части Web-страницы ссылку **Service Description**, которая отсылает пользователя как раз к упомянутому WSDL-файлу. Код этого WSDL-файла, соответствующего нашему Web-сервису, приведен в листинге 7.2.

Листинг 7.2

```
<?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/" 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. 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. В этом разделе 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 — это 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 использовать метод `GET`. Как известно, при использовании этого метода вся информация, передаваемая на сервер, передается через строку запроса, вместе с URL. Таким образом, 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 7.3.

Листинг 7.3

```
<!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. 7.4.

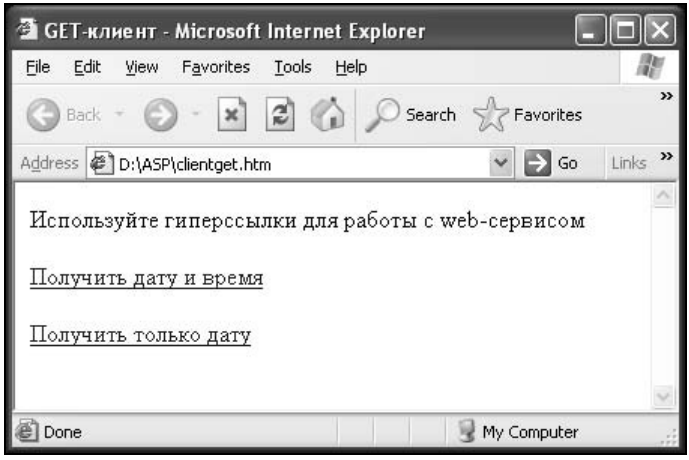


Рис. 7.4. Клиентская Web-страница, работающая с методом `GET`

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 указываются наименование параметра и его значение. Именно таким образом передаются данные при помощи метода `GET`.

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

Теперь перейдем к созданию клиентской Web-страницы, передающей запрос при помощи метода `POST`. Как известно, метод `POST` позволяет переда-

вать данные, введенные пользователем в HTML-форме. Следовательно, нам придется на Web-странице создать форму.

Так как параметр имеет логический тип, правильно будет создать группу из двух переключателей. Передаваемое значение может быть или `True` или `False`, и именно эти значения будут возвращать кнопки-переключатели. 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 кнопок-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 7.4.

Листинг 7.4

```
<!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. 7.5.

Итак, мы рассмотрели создание клиентов на основе Web-страниц. Если пользоваться языком SOAP для коммуникации между клиентами и серви-

сом, то придется создавать отдельное приложение, что уже выходит за пределы темы книги. Поэтому на этом мы и остановимся.

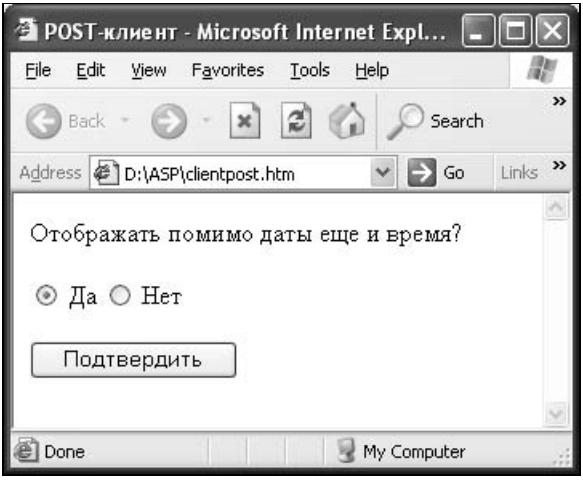


Рис. 7.5. Клиентская Web-страница, работающая с методом `POST`

Послесловие

Пришло время оглянуться назад. На протяжении всей книги мы рассматривали стандартные механизмы работы с технологией ASP.NET. Теперь можно с уверенностью сказать, что мы обладаем необходимым багажом знаний и навыков для того, чтобы создавать интерактивные и разветвленные сайты. Эта гибкая и удивительно мощная технология, как теперь выяснилось, позволяет создавать сколь угодно серьезные Web-проекты, и автор надеется, что смог помочь вам в овладении этим инструментом.

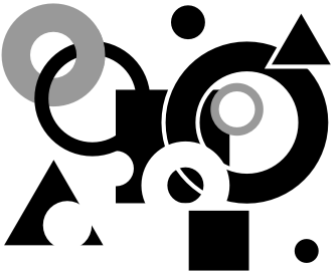
Конечно, в этой книге мы рассмотрели далеко не все возможности, которые предоставляет ASP.NET. Но это ведь просто невозможно. Сама технология Microsoft .NET настолько многогранна, что для ее полного описания потребуется несколько очень толстых книг. Поэтому в этой книге были отражены основные вопросы, встающие у Web-разработчика, доселе незнакомого с ASP.NET. Опираясь на материал, изложенный здесь, программист сможет создавать свои проекты и двигаться дальше в изучении ASP.

Удачных вам проектов. Встретимся в Сети.

Шапошников Игорь

Webmaster@bhv.spb.su

Приложение



В табл. П1 и П2 приведены характеристики константы типа `DateInterval` и основные параметры отображения в CSS.

Таблица П1. *Константы типа* `DateInterval`

Константа `DateInterval`	**Строка**	**Временной интервал**
`DateInterval.Day`	`d`	День. Результат указывается с точностью до целого числа
`DateInterval.DayOfYear`	`y`	День. Учитывается порядковый номер дня в году
`DateInteval.Hour`	`h`	Час. Результат указывается с точностью до миллисекунды
`DateInterval.Minute`	`n`	Минута. Результат указывается с точностью до миллисекунды
`DateInterval.Month`	`m`	Месяц. Результат учитывается с точностью до дня
`Dateinterval.Quarter`	`q`	Квартал. Результат учитывается с точностью до дня
`DateInterval.Second`	`s`	Секунда. Результат указывается с точностью до миллисекунды
`DateInterval.Weekday`	`w`	День. Отсчет ведется по дням недели
`DateInterval.WeekOfYear`	`ww`	Неделя. Отсчет ведется по количеству недель в году
`DateInterval.Year`	`yyyy`	Год. Отсчитывается по целочисленным значениям

Таблица П2. *Основные параметры отображения в CSS*

Наименование параметра	Описание параметра	Значения параметра
`position`	Указывает модель позиционирования элемента Web-страницы в окне просмотра браузера	`static`, `relative`, `absolute`, `fixed`
`top`	Вертикальная координата верхнего левого угла элемента	Количество пикселов или процентное соотношение
`bottom`	Вертикальная координата нижнего левого угла элемента	Количество пикселов или процентное соотношение
`left`	Горизонтальная координата верхнего левого угла элемента	Количество пикселов или процентное соотношение
`right`	Горизонтальная координата верхнего правого угла элемента	Количество пикселов или процентное соотношение
`float`	Горизонтальное выравнивание элемента	`left`, `right`, `none`
`width`	Ширина элемента	Количество пикселов или процентное соотношение
`min-width`	Минимальная ширина элемента	Количество пикселов или процентное соотношение
`max-width`	Максимальная ширина элемента	Количество пикселов или процентное соотношение
`length`	Высота элемента	Количество пикселов или процентное соотношение
`min-length`	Минимальная высота элемента	Количество пикселов или процентное соотношение
`max-length`	Максимальная высота элемента	Количество пикселов или процентное соотношение
`line-height`	Высота элемента, размещаемого в пределах одной строки	Количество пикселов, процентное соотношение, или коэффициент увеличения основного размера шрифта
`vertical-align`	Вертикальное выравнивание элемента	`baseline`, `sub`, `super`, `top`, `text-top`, `middle`, `bottom`, `text-bottom`, количество пикселов или процентное соотношение
`margin-left`	Размер левого поля элемента	Количество пикселов или процентное соотношение

Таблица П2 *(продолжение)*

Наименование параметра	Описание параметра	Значения параметра
`margin-right`	Размер правого поля элемента	Количество пикселов или процентное соотношение
`margin-top`	Размер верхнего поля элемента	Количество пикселов или процентное соотношение
`margin-bottom`	Размер нижнего поля элемента	Количество пикселов или процентное соотношение
`margin`	Размеры всех полей элемента	Количество пикселов или процентное соотношение
`padding-left`	Размер левого отступа элемента от его границы	Указание размера или процентное соотношение
`padding-right`	Размер правого отступа элемента от его границы	Указание размера или процентное соотношение
`padding-top`	Размер верхнего отступа элемента от его границы	Указание размера или процентное соотношение
`padding-bottom`	Размер нижнего отступа элемента от его границы	Указание размера или процентное соотношение
`padding`	Размеры всех отступов элемента	Указание размера или процентное соотношение
`border-left-width`	Ширина левой границы элемента	`thin`, `medium`, `thick` или указание размера в пикселах
`border-right-width`	Ширина правой границы элемента	`thin`, `medium`, `thick` или указание размера в пикселах
`border-top-width`	Ширина верхней границы элемента	`thin`, `medium`, `thick` или указание размера в пикселах
`border-bottom-width`	Ширина нижней границы элемента	`thin`, `medium`, `thick` или указание размера в пикселах
`border-width`	Ширина всех границ элемента	`thin`, `medium`, `thick` или указание размера в пикселах
`border-left-color`	Цвет левой границы элемента	Указание цвета
`border-right-color`	Цвет правой границы элемента	Указание цвета

Таблица П2 *(продолжение)*

Наименование параметра	Описание параметра	Значения параметра
`border-top-color`	Цвет верхней границы элемента	Указание цвета
`border-bottom-color`	Цвет нижней границы элемента	Указание цвета
`border-color`	Цвет всех границ элемента	Указание цвета
`border-left-style`	Стиль левой границы элемента	`none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`
`border-right-style`	Стиль правой границы элемента	`none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`
`border-top-style`	Стиль верхней границы элемента	`none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`
`border-bottom-style`	Стиль нижней границы элемента	`none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`
`border-style`	Стиль всех границ элемента	`none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`
`border-top`	Задает все правила отображения верхней границы элемента	Указание цвета, стиля и размера
`border-left`	Задает все правила отображения левой границы элемента	Указание цвета, стиля и размера
`border-bottom`	Задает все правила отображения нижней границы элемента	Указание цвета, стиля и размера
`border-right`	Задает все правила отображения правой границы элемента	Указание цвета, стиля и размера
`border`	Задает все правила отображения границ элемента	Указание цвета, стиля и размера
`z-index`	Указывает номер слоя, в котором будет отображаться содержимое элемента	`auto` или порядковый номер слоя

Таблица П2 *(продолжение)*

Наименование параметра	Описание параметра	Значения параметра
direction	Указывает направление по горизонтали, в котором будет отображаться текстовое содержимое элемента	rtl, ltr
overflow	Указывает браузеру правила отображения элемента, если его содержимое выходит за пределы отведенного ему пространства	visible, hidden, scroll, auto
visibility	Управляет видимостью элемента	visible, hidden, collapse
color	Задает цвет, которым будет отображаться содержимое элемента	Указание цвета
background-color	Задает цвет фона элемента	transparent или стандартное указание цвета
background-image	Задает графическое изображение, использующееся в качестве фона элемента	none или URL графического файла, который будет служить фоном
background-repeat	Задает повторяемость графического изображения, служащего фоном для элемента	repeat, repeat-x, repeat-y, no-repeat
background-attachment	Задает "привязку" фонового изображения к содержимому элемента	scroll, fixed
background-position	Задает координаты фонового изображения	top, center, bottom, left, right, указание координат в пикселах или в процентном соотношении
background	Задает все параметры фонового изображения	URL графического рисунка, координаты фона, привязка к элементу, повторяемость фонового рисунка, цвет фона
font-family	Указывает наименование семейства шрифта, используемого для отображения текстового содержимого элемента	serif, sans-serif, cursive, fantasy, monospace, наименования шрифтов
font-style	Задает стиль отображения символов используемого шрифта	normal, italic, oblique
font-variant	Устанавливает регистр символов используемого шрифта	normal, small-caps

Таблица П2 *(продолжение)*

Наименование параметра	Описание параметра	Значения параметра
font-weight	Задает ширину и насыщенность символов используемого шрифта	normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900
font-stretch	Задает межсимвольный интервал для текстового содержимого элемента	normal, wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expanded
font-size	Задает размер символов используемого шрифта	xx-small, x-small, small, medium, large, x-large, xx-large, larger, smaller, размер в пунктах или процентное соотношение
font-size-adjust	Задает коэффициент увеличения символов шрифта	none или коэффициент масштабирования
font	Задает свойства используемого шрифта	Все перечисленные параметры шрифта
text-decoration	Устанавливает эффекты оформления текста	none, underline, overline, line-through, blink
text-shadow	Накладывает тень на текстовое содержимое элемента	Цвет и размеры тени
text-transform	Принудительно приводит символы к указанному регистру	capitalize, uppercase, lowercase, none
text-indent	Устанавливает отступ для первой строки абзаца	Размер отступа в пикселах или в процентном соотношении
text-align	Устанавливает выравнивание строк абзаца	left, right, center, justify
white-space	Регулирует вид используемого пробела	normal, pre, nowrap
caption-side	Устанавливает расположение заголовка таблицы относительно ее тела	top, bottom, left, right

Таблица П2 *(окончание)*

Наименование параметра	Описание параметра	Значения параметра
table-layout	Задает алгоритм раскладки таблицы	auto, fixed
border-collapse	Устанавливает модель отображения границ ячеек	collapse, separate
border-spacing	Задает размер отступа между ячейками таблицы	Указание размеров
cursor	Задает внешний вид курсора при прохождении его над элементом	auto, crosshair, default, pointer, move, e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize, text, wait, help