37

Страницы ASP.NET

Если вы — новичок в C# и .NET, то можете удивиться, почему в эту книгу включена глава о ASP.NET. Это же совершенно новый язык, правильно? Ну, не совсем. На самом деле, как вы скоро убедитесь, C# можно использовать для создания страниц ASP.NET.

ASP.NET — часть среды .NET Framework, и представляет собой технологию, позволяющую динамически создавать документы на Web-сервере, когда они запрашиваются по протоколу HTTP. Главным образом, речь идет о HTML- и XHTML-документах, хотя также возможно создавать документы XML, файлы каскадных таблиц стилей (CSS), изображения, документы формата PDF или что-то еще, что поддерживает типы MIME.

В некотором роде ASP.NET подобно многим другим технологиям — таким как PHP, ASP или ColdFusion. Однако имеется одно ключевое отличие — ASP.NET, что должно быть понятно по ее названию, разработана как полностью интегрированная составляющая .NET Framework, часть которой включает поддержку C#.

Возможно, вы уже знакомы с технологией Active Server Pages (ASP), которая позволяет создавать динамическое содержимое для Web. Если это так, то, вероятно, вы знаете, что при программировании в этой технологии используются сценарные языки, такие как VBScript и JScript. Результат не всегда блестящий — по крайней мере, для тех, кто имеет опыт работы с "правильными" компилирующими языками программирования, и почти всегда связан со снижением производительности.

Одним главным отличием, связанным с применением более развитых языков программирования, является обеспечение полноценной объектной модели на стороне сервера для использования во время выполнения. ASP.NET предоставляет доступ ко всем элементам управления страницы как к объектам в многофункциональном окружении. На стороне сервера вы получаете доступ ко всем классам .NET, что позволяет интегрировать множество полезных служб. Элементы управления, размещаемые на страницах, предлагают богатый набор функциональности; фактически, вы получаете возможность делать почти все то, что делают классы Windows Forms с их невероятной гибкостью. ASP.NET, генерирующие содержимое HTML, часто называют *Web Forms* (Web-формы).

В этой главе мы детально рассмотрим технологию ASP.NET, включая то, как она работает, что можно делать с ее помощью и как в нее вписывается язык C#. Ниже следует краткий обзор тех вопросов, которые будут рассмотрены в настоящей главе:

- введение в ASP.NET;
- как создаются Web-формы ASP.NET с помощью серверных элементов управления;

- как осуществляется привязка данных к элементам управления ASP.NET посредством ADO.NET;
- конфигурирование приложений.

Введение в ASP.NET

Для доставки содержимого в ответ на HTTP-запросы ASP.NET взаимодействует с информационным сервером Internet (Internet Information Server — IIS). Страницы ASP.NET находятся в файлах `.aspx`. На рис. 37.1 показана базовая архитектура этой технологии.

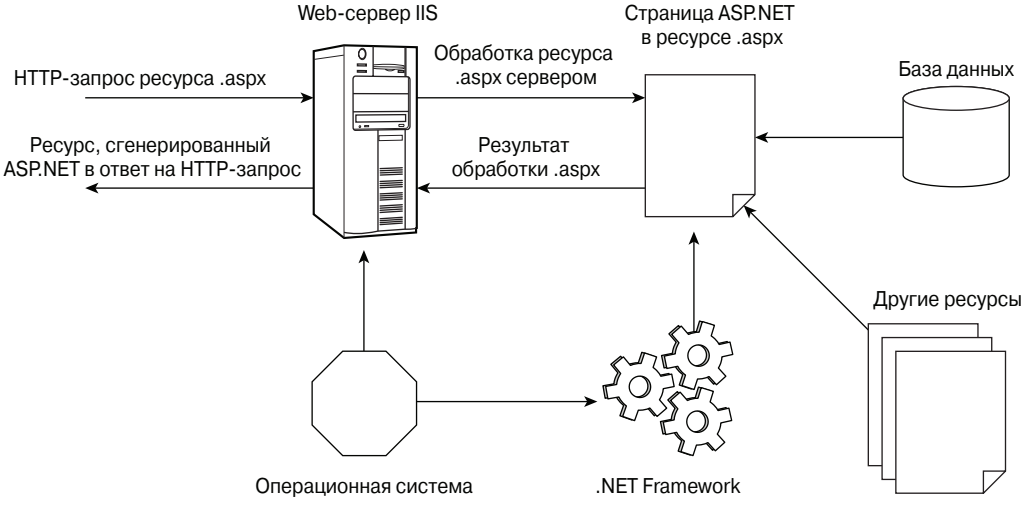

Рис. 37.1. Базовая архитектура технологии ASP.NET

Во время обработки ASP.NET вы имеете доступ к классам .NET, пользовательским компонентам, созданным на C# и других языках, базам данных и т.д. Фактически, в вашем распоряжении находится весь арсенал, доступный приложениям C#; использование C# в ASP.NET дает эффект работающего приложения C#.

Файл ASP.NET может содержать:

- инструкции обработки для сервера;
- код на C#, Visual Basic .NET, JScript .NET или любом другом языке, поддерживаемом .NET Framework;
- содержимое в любой подходящей форме для генерируемого ресурса, такого как HTML;
- сценарный код клиентской стороны, например, JavaScript;
- встроенные серверные элементы управления ASP.NET.

Таким образом, вы можете иметь файл ASP.NET, настолько простой, как показан ниже:

```
Hello!
```

Это просто даст результат в виде возвращенной HTML-страницы (поскольку HTML — это вывод по умолчанию страниц ASP.NET), которая содержит представленный выше текст.

Как будет показано далее в этой главе, определенные части кода можно вынести в другие файлы, что позволяет организовать более логичную структуру.

Управление состоянием в ASP.NET

Одно из ключевых свойств страниц ASP.NET состоит в том, что они в действительности не поддерживают концепцию состояния. По умолчанию никакая информация не сохраняется на сервере между пользовательскими запросами (хотя, как мы увидим позже, есть методы, чтобы сделать это). На первый взгляд это выглядит несколько странно, поскольку управление состоянием — нечто такое, что выглядит обязательным условием для дружественных к пользователю интерактивных сеансов. Однако ASP.NET предлагает способ обойти эту проблему, так что управление сеансом становится почти прозрачным.

Короче говоря, информация о состоянии элементов управления Web Forms (включая данные, введенные в текстовые поля, или выбор в выпадающих списках) сохраняются в скрытых полях *viewstate* (состояние представления), являющихся частью сгенерированной сервером и переданной пользователю страницы. Следующие друг за другом действия, такие как возбуждение событий, требующих обработки со стороны сервера, вроде отправки данных формы, инициируют передачу этой информации обратно на сервер, что называется *postback* (обратной отправкой). На сервере эта информация используется для повторного наполнения объектной модели страницы, что позволяет работать, как будто изменения были выполнены локально.

Вскоре мы увидим все это в действии и разберем в деталях.

ASP.NET Web Forms

Как упоминалось ранее, б*о*льшая часть функциональности ASP.NET достигается с применением Web Forms. Прежде чем погрузиться в детали, попробуем создать простую Web-форму, чтобы получить некоторое начальное представление перед изучением этой технологии. Сначала в этом разделе мы представим обзор некоторых ключевых моментов, относящихся к проектированию Web-форм. Следует отметить, что некоторые разработчики ASP.NET просто используют текстовый редактор — вроде Notepad — для создания файлов. Мы не являемся сторонниками такого подхода, потому что те преимущества, которые дают такие интегрированные среды разработки, как Visual Studio или Web Developer Express, весьма существенны, однако стоило упомянуть и такой способ, поскольку он вполне допустим. Если вы изберете такой путь, то сможете гибко организовывать размещение частей вашего Web-приложения по файлам. Это позволит вам, например, скомбинировать весь код в одном файле, если заключать код в элементы `<script>`, используя два атрибута в открывающем дескрипторе:

```
<script language="c#" runat="server">
  // Здесь размещается код серверной стороны.
</script>
```

Атрибут `runat="server"` здесь является ключевым, поскольку инструктирует механизм ASP.NET выполнять этот код на сервере вместо отправки его клиенту, таким образом, предоставляя в ваше распоряжение богатые возможности, упомянутые ранее. В блоки серверной стороны можно помещать функции, обработчики событий и тому подобное.

Если вы пропустите атрибут `runat="server"`, то это будет означать код клиентской стороны, который даст сбой, если станет использовать кодирование в стиле серверной стороны, обсуждаемое далее в настоящей главе. Тем не менее, вы можете использовать элементы `<script>` для того, чтобы создать сценарий клиентской стороны на таких языках, как JavaScript.

Например:

```
<script language="JavaScript" type="text/JavaScript">
  // Клиентский код размещается здесь; можно также использовать "vbscript".
</script>
```

Обратите внимание, что атрибут `type` не обязателен, но его нужно применять, если вы хотите обеспечить совместимость с XHTML.

Может показаться странным то обстоятельство, что в ASP.NET имеется возможность добавлять код JavaScript в ваши страницы. Однако JavaScript позволяет добавлять в ваши Web-страницы динамическое поведение на стороне клиента, что может быть очень полезным. Это особенно справедливо для программирования с применением Ajax, о чем будет сказано в главе 39.

Можно также создавать файлы ASP.NET в среде Visual Studio, которая отлично подойдет, если вы уже знакомы со средой программирования C#. Однако настройка проекта по умолчанию для Web-приложений в этой среде представляет собой несколько более сложную структуру, чем единственный файл `.aspx`. Это не будет большой проблемой, просто сместит акцент в сторону логики (больше программирования и меньше разработки для Web). По этой причине в данной главе мы будем использовать Visual Studio для программирования ASP.NET (вместо редактора Notepad).

Файлы `.aspx` также могут включать блоки кода, заключенные в дескрипторы `<%` и `%>`. Однако определения функций и объявления переменных в них появляться не могут. Вместо этого сюда вставляется код, который выполняется немедленно при достижении такого блока, что удобно для вывода простого HTML-содержимого. Такое поведение подобно ASP-страницам старого стиля, но с одним важным отличием: код компилируется, а не интерпретируется. Это обеспечивает гораздо более высокую производительность.

Теперь приступим к реализации примера. В Visual Studio создадим новое Web-приложение, используя пункт меню File⇨New⇨Web Site (Файл⇨Новый⇨Web-сайт). В появившемся диалоговом окне выберем язык Visual C# и шаблон ASP.NET Web Site. В этот момент у нас появляется выбор. Visual Studio может создавать Web-сайты во многих разных местах:

- на локальном Web-сервере IIS;
- на локальном диске, настроенном для использования встроенного сервера Visual Studio Developer Web Server;
- в любом месте, доступном по FTP;
- на удаленном Web-сервере, поддерживающем серверные расширения Front Page Server Extensions.

Последние два варианта используют удаленные серверы, поэтому нам остается первых два. Вообще говоря, IIS — это лучшее место для инсталляции Web-сайтов ASP.NET, потому что, вероятно, это будет ближе всего к той конфигурации, которая потребуется при поставке Web-сайта заказчику. Альтернативный вариант с использованием встроенного Web-сервера отлично подходит для тестирования, но имеет ряд ограничений.

- Web-сайт можно видеть только на локальном компьютере.
- Доступ к таким службам, как SMTP, ограничен.
- Модель безопасности отличается от принятой в IIS — приложение выполняется в контексте текущего пользователя, а не специфической учетной записи ASP.NET.

Последний пункт требует пояснения, потому что безопасность очень важна, когда дело касается доступа к базам данных или чего-то другого, что требует аутентификации. По умолчанию Web-приложения, исполняющиеся на сервере IIS, делают это от имени учетной записи с именем `ASPNET` на Web-серверах Windows XP или Windows 2000, или же от имени учетной записи `NETWORK SERVICES` — в Windows Server 2003. Это все настраивается, если используется IIS, но не в случае применения встроенного Web-сервера.

Для иллюстрации, и поскольку у вас может не быть инсталлированного сервера IIS на вашем компьютере, можно использовать встроенный Web-сервер. На этой стадии нам не нужно беспокоиться о безопасности, поэтому для простоты так и поступим.

Создадим новый Web-сайт ASP.NET с именем `PCSWebApp1`, используя расположение File System (Файловая система) и каталог `C:\ProCSharp\Chapter37`, как показано на рис. 37.2.

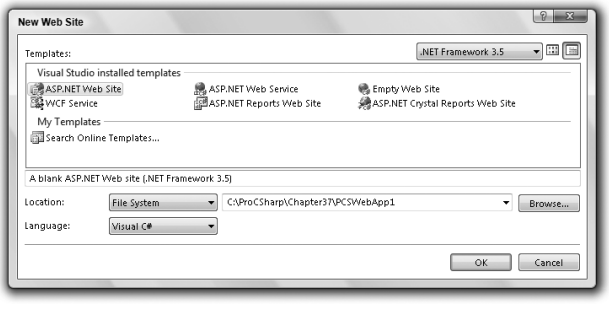

Рис. 37.2. Создание нового Web-сайта ASP.NET с именем `PCSWebApp1`

Через несколько мгновений среда Visual Studio .NET должна настроить следующее:

- `PCSWebApp1` — новое решение, содержащее приложение C# Web Application PCSWebApp1;
- резервную папку `App_Data` для данных, таких как файлы XML и файлы базы данных;
- первую страницу ASP.NET в Web-приложении — `Default.aspx`;
- `Default.aspx.cs` — отделенный код класса для `Default.aspx`.
- `Web.config` — конфигурационный файл для Web-приложения.

Все это можно увидеть в проводнике решений Solution Explorer, как показано на рис. 37.3.

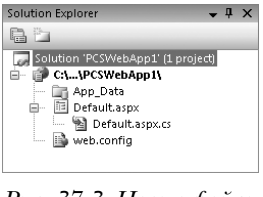

Рис. 37.3. Новые файлы в проводнике решений Solution Explorer

Файлы `.aspx` можно видеть в представлении дизайнера или исходного текста (HTML). Все выглядит так же, как и в Windows Forms (см. главу 31). Начальное представление в Visual Studio — это либо представление дизайнера (design view), либо представление исходного текста для `Default.aspx`. (Переключаться между ними или просматривать их вместе в раздельном отображении можно с помощью кнопок внизу слева.) Представление дизайнера показано на рис. 37.4.

Ниже формы (пока пустой) вы можете видеть, где в HTML-коде формы позиционирован курсор. Здесь курсор находится на элементе `<div>` внутри элемента страницы `<body>`. Элемент `<form>` отображается как `<form#form1>`, чтобы обозначить элемент по его атрибуту `id`, который вы вскоре увидите. Элемент `<div>` тоже отмечен в представлении дизайнера.

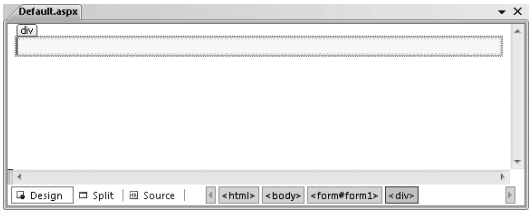

Рис. 37.4. Представление дизайнера для `Default.aspx`

Представление исходного текста страницы отобразит код, сгенерированный в файле `.aspx`:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
     <div>
     </div>
  </form>
  </body>
</html>
```

Если вы знаете синтаксис языка HTML, это покажется вам знакомым. Здесь представлен базовый код, необходимый для HTML-страницы, которая следует схеме XHTML, с несколькими небольшими дополнениями. Самое важное дополнение — элемент `<form>`, который будет содержать наш код ASP.NET. В нем самое главное — это атрибут `runat`. Точно так же, как это было с блоком серверной стороны, который мы видели в начале раздела, для него установлено значение `server`, означающее, что обработка формы будет происходить на сервере. Если не включить эту ссылку, то никакой обработки со стороны сервера не произойдет и форма ничего не сделает. На странице ASP.NET может присутствовать только один элемент `<form>` серверной стороны.

Другой интересный момент в этом коде — это дескриптор `<%@ Page %>` в верхней части. Он определяет характеристики страницы, которые важны для нас как разработчиков Web-приложений на C#. Здесь присутствует атрибут `Language`, указывающий, что на странице будет использован код на языке C# — точно так же, как мы видели раньше в блоке `<script>` (стандартным для Web-приложений является Visual Basic .NET, хотя это можно изменить в файле `Web.config`, который мы рассмотрим в этой главе позднее). Остальные три атрибута — `AutoEventWireup`, `CodeFile` и `Inherits` — служат для того, чтобы ассоциировать Web-форму с классом в файле отделенного кода, в данном случае — частичным (`partial`) классом `_Default` в файле `Default.aspx.cs`. Это заставляет нас поговорить о модели кода ASP.NET.

Модель кода ASP.NET

В ASP.NET комбинация кода разметки (HTML), элементов управления ASP.NET и кода C# используется для генерации HTML, представляемого пользователю. Разметка и код ASP.NET размещаются в файле `.aspx` — вроде того, что мы видели в предыдущем разделе. Код C#, добавляемый для настройки поведения формы, находится либо

в файле .aspx, либо, как в предыдущем примере, в отдельном файле .aspx.cs, который обычно называют "файлом отделенного кода".

Когда обрабатывается Web-форма ASP.NET, то обычно, когда пользователь запрашивает страницу, несмотря на то, что сайты могут быть предварительно компилированы, происходят описанные ниже вещи.

- Процесс ASP.NET просматривает страницу и определяет, какие объекты должны быть созданы для того, чтобы сконструировать экземпляр модели страницы.
- Базовый класс для страницы создается динамически, включая члены для элементов управления страницы вместе с их обработчиками событий (такими как события щелчков на кнопках).
- Дополнительный код, включенный в страницу .aspx, комбинируется с этим базовым классом для завершения построения объектной модели.
- Полный код компилируется и помещается в кэш, готовый к обработке последующих запросов.
- Генерируется HTML-код, который возвращается пользователю.

Файл отделенного кода, сгенерированный средой разработки для нашего Web-сайта PCSWebApp1 и помещенный в Default.aspx, очень прост. Если мы заглянем в него, то в начале увидим набор ссылок на пространства имен, которые вы, возможно, будете использовать в Web-страницах ASP.NET:

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Xml.Linq;
```

Под этими ссылками находится почти пустое частичное определение класса:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }
}
```

Здесь обработчик событий Page_Load() может быть использован для добавления любого необходимого в момент загрузки страницы кода. По мере добавления новых обработчиков событий этот класс будет наполняться кодом. Обратите внимание, что здесь нет кода, который связывает обработчик событий со страницей — как уже говорилось ранее, это делает исполняющая система ASP.NET. Такое поведение задано атрибутом AutoEventWireUp — присваивание ему значения false означало бы необходимость ассоциировать обработчики с событиями вручную.

Этот класс является частичным классом, поскольку этого требует процесс, описанный выше. Когда выполняется предварительная компиляция страницы, то на основе кода ASP.NET вашей страницы создается отдельное определение частичного класса. Оно включает все элементы управления, которые были добавлены на страницу. На этапе проектирования компилятор распознает это определение частичного класса, что позволяет вам использовать IntelliSense в вашем коде для ссылочных элементов управления на вашей странице.

Серверные элементы управления ASP.NET

Наш сгенерированный код пока мало на что пригоден, поэтому следующее, что потребуется сделать — добавить некоторое содержимое. Это можно сделать в Visual Studio с помощью дизайнера Web Forms, который поддерживает перетаскивание элементов мышью — почти таким же способом, как это делается в дизайнере Windows Forms.

На страницы ASP.NET можно добавлять три типа элементов управления.

- **Серверные элементы управления HTML** — имитируют элементы HTML, которые должны быть знакомы разработчикам на HTML.
- **Серверные элементы управления Web —** новый набор элементов, часть которого имеют ту же функциональность, что и элементы HTML. Эти элементы управления используют общую схему именования свойств и прочих элементов для облегчения разработки и согласования с аналогичными элементами управления Windows Forms. Как увидим позднее, доступны также некоторые другие совершенно новые и очень мощные элементы управления. Среди Web-серверных элементов управления есть стандартные — такие как кнопки, элементы проверки достоверности для контроля пользовательского ввода, элементы регистрации (Login), упрощающие управление пользователями, — а также более сложные элементы, предназначенные для работы с источниками данных.
- **Настраиваемые и пользовательские элементы управления** — эти элементы определяются разработчиком и могут быть созданы разными способами, как описано в главе 38.

В следующем разделе представлен список многих часто используемых Web-серверных элементов управления вместе с замечаниями по их применению. Некоторые дополнительные элементы управления будут рассмотрены в следующей главе. Мы не описываем в этой книге элементы управления HTML. Они не делают ничего более того, что доступно серверным элементам управления Web, к тому же последние предлагают расширенные возможности разработчикам, привыкшим к программированию, а не дизайну с использованием HTML. Изучение применения серверных элементов управления Web дает достаточно знаний для использования серверных элементов управления HTML.

Теперь добавим несколько Web-серверных элементов управления к созданному в предыдущем разделе Web-сайту `PCSWebApp1`. Все Web-серверные элементы управления используются в следующей форме XML-элемента:

```
<asp:имяЭлемента runat="server" атрибут="значение">Содержимое</asp:имяЭлемента>
```

Здесь *`имяЭлемента`* — имя серверного элемента управления ASP.NET, *`атрибут`*`="`*`значение`*`"` — одна или более спецификаций атрибутов, а *`Содержимое`* — содержимое, если оно есть. Некоторые элементы позволяют устанавливать свойства посредством атрибутов и содержимого элемента управления, такого как `Label` (используемого для простого отображения текста), где `Text` может быть специфицирован обоими способами. Другие элементы управления могут применять схему содержания элементов для определения их иерархии, например, `Table` (определяет таблицу), который может включать в себя элементы `TableRow` для декларативного описания строк таблицы.

Поскольку синтаксис элементов управления базируется на XML (хотя они могут быть включены и в не-XML-код, такой как HTML), считается ошибкой, если не указан закрывающий дескриптор и `/>` для пустых элементов или же определения элементов управления перекрываются.

И, наконец, мы опять видим здесь атрибут runat="server". Его смысл здесь — такой же, как и везде, и часто допускаемой ошибкой является пропуск этого атрибута, в результате чего получаются неработающие Web-формы.

Первый пример будет очень простым. Изменим представление HTML-дизайна Default.aspx следующим образом:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
    Inherits="_Default" %>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
    <title>Untitled Page</title>
    </head>
    <body>
    <form id="form1" runat="server">
      <div>
         <asp:Label runat="server" ID="resultLabel" /><br />
         <asp:Button runat="server" ID="triggerButton" Text="Click Me" />
      </div>
    </form>
    </body>
</html>
```

Здесь добавлены два элемента управления Web Forms: метка (Label) и кнопка (Button).

Обратите внимание, что когда вы будете это делать, средство IntelliSense в Visual Studio .NET предвосхитит ввод нужного кода, как это имеет место и в редакторе кода C#. Кроме того, если вы отредактируете свой код в раздельном представлении, а затем синхронизируете представления, то элемент, который вы редактируете в панели исходного кода, будет выделен в панели дизайнера.

Вернувшись на экран дизайнера, мы увидим, что элементы управления добавлены и поименованы в соответствии со значение атрибутов ID (атрибут ID часто называют *идентификатором* (identifier) элемента управления). Как и в Windows Forms, здесь предоставляется полный доступ к свойствам, событиям и тому подобному — через окно свойств, причем все внесенные изменения немедленно отображаются как на экране дизайнера, так и в коде.

Вы можете использовать окно CSS Properties *(Свойства CSS) и любые другие окна стилей для выбора стилей для ваших элементов управления. Однако если вы знакомы с CSS, то вы, вероятно, пока что можете не прибегать к этому способу, а сосредоточиться на функциях других элементов управления.*

Все добавленные серверные элементы управления автоматически становятся частью объектной модели разрабатываемой формы. Это приятное открытие для разработчиков приложений Windows Forms — они обнаружат здесь полное сходство.

Чтобы заставить наше приложение делать что-то полезное, нам нужно добавить обработчик события щелчка кнопки. Здесь мы можем либо ввести имя метода в окне свойств кнопки, либо просто дважды щелкнуть на кнопке, чтобы получить обработчик события по умолчанию. Если сделать так, то автоматически будет добавлен метод обработки события следующего вида:

```
void triggerButton_Click(object sender, EventArgs e)
{
}
```

Этот метод будет привязан к кнопке с помощью кода, добавленного в `Default.aspx`:

```
<div>
  <asp:Label Runat="server" ID="resultLabel" /><br />
  <asp:Button Runat="server" ID="triggerButton" Text="Click Me"
       onClick="triggerButton_Click" />
</div>
```

Здесь атрибут `onClick` дает знать исполняющей системе ASP.NET, что при генерации модели кода формы событие щелчка на кнопке нужно связать с вызовом метода `triggerButton_Click()`.

Модифицируем код этого метода следующим образом (заметьте, что метка, как элемент управления, выводится из кода ASP.NET, поэтому вы можете использовать ее непосредственно из отделенного кода):

```
void triggerButton_Click(object sender, EventArgs e)
{
  resultLabel.Text = "Button clicked!";
}
```

Теперь все готово. Нет необходимости даже выполнять сборку проекта; следует только убедиться, что все было сохранено, и адресовать Web-браузер на местоположение нашего Web-сайта. Однако, поскольку для этого примера мы используем встроенный Web-сервер, его нужно как-то запустить. Быстрее всего это можно сделать, нажав комбинацию клавиш <Ctrl+F5> — при этом запустится сервер и откроется браузер с нужным URL.

Когда встроенный Web-сервер запущен, в правой части линейки внизу экрана появляется пиктограмма. Если дважды щелкнуть на ней, можно увидеть, какой Web-сервер запущен, и остановить его при необходимости. Диалог, появляющийся при этом, показан на рис. 37.5.

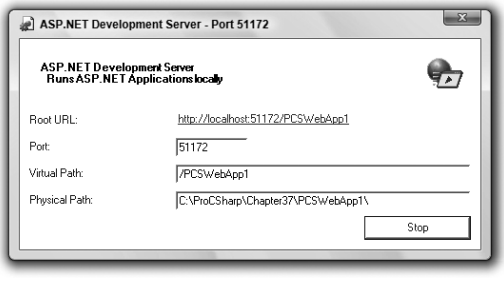

Рис. 37.5. Запуск встроенного Web-сервера

Здесь виден номер порта, на котором работает Web-сервер, а также URL, обратившись к которому можно увидеть только что созданный Web-сайт.

В открывшемся браузере на Web-странице можно видеть кнопку Click Me. Прежде чем щелкать на ней, посмотрим на код, полученный браузером, используя пункт меню Page⇨View Source (Страница⇨Исходный код) (в браузере IE7).

Раздел `<form>` должен выглядеть примерно так:

```
<form method="post" action="Default.aspx" id="form1">
  <div>
     <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
       value="/wEPDwUKLTE2MjY5MTY1NWRkzNjRYstd1OK5KcJ9a8/X3pYTHvM=" />
  </div>
```

```
    <div>
       <span id="resultLabel"></span><br />
       <input type="submit" name="triggerButton" value="Click Me" id="triggerButton" />
    </div>
    <div>
      <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
       value="/wEWAgK39qTFBwLHpP+yC4rCCl22/GGMaFwD0l7nokvyFZ8Q" />
    </div>
  </form>
```

Web-серверные элементы управления содержат обычный HTML-код — `<span>` и `<input>` для `<asp:Label>` и `<asp:Button>` соответственно. Есть также поле `<input type="hidden">` с именем `__VIEWSTATE`, которое инкапсулирует состояние формы, как упоминалось ранее. Эта информация используется при обратной отправке формы на сервер для пересоздания пользовательского интерфейса, отслеживания изменений и т.д. Отметим, что элемент `<form>` сконфигурирован для этого; он отправляет данные обратно `Default.aspx` (как указано в `action`) через HTTP-операцию `POST` (указано в `method`). Форме присвоено имя `form1`.

После щелчка на кнопке и появления текста опять посмотрим на исходный HTML-код (пробелы добавлены для наглядности):

```
<form method="post" action="Default.aspx" id="form1">
  <div>
     <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
       value="/wEPDwUKLTE2MjY5MTY1NQ9kFgICAw9kFgICAQ8PFgIeBFRleHQFD0J1dHR
       vbiBjbGlja2VkIWRkZExUtMwuSlVTrzMtG7wrmj98tVn7" />
  </div>
  <div>
     <span id="resultLabel">Button clicked!</span><br />
     <input type="submit" name="triggerButton" value="Click Me" id="triggerButton" />
  </div>
  <div>
    <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
     value="/wEWAgKTpL7LBALHpP+yC0Ymqe9SgScfB2yHTGjnlQKtbudV" />
  </div>
</form>
```

На этот раз viewstate-значение содержит больше информации, поскольку HTML-результат дает больше, чем вывод по умолчанию страницы ASP.NET. В сложных формах состояние может описываться очень длинными строками, но это не должно вас беспокоить, поскольку многое делается “за кулисами”. На самом деле вы можете вообще забыть об управлении состоянием, сохранении значений полей между вызовами и тому подобном. Если длина строки состояния оказывается слишком большой, то эту строку можно вообще не использовать для тех элементов управления, которые не должны хранить информацию о состоянии. Это можно сделать даже для всей страницы, что, кстати, очень удобно в плане повышения производительности, если страница не должна сохранять состояние между обратными отправками.

Более подробно о состоянии представления можно прочитать в главе 38.

Чтобы убедиться, что нет необходимости ни в какой ручной компиляции, попробуем изменить текст `"Button clicked!"` в `Default.aspx.cs` на что-нибудь другое, сохраним файл и щелкнем на кнопке снова. Текст, который появится на Web-странице, должен соответствующим образом измениться.

Палитра элементов управления

В этом разделе мы кратко рассмотрим некоторые из доступных элементов управления, прежде чем использовать большинство из них вместе в полноценном, более

интересном приложении. Этот раздел разбит на части в соответствии с организацией панели инструментов, которую можно видеть при редактировании страниц ASP.NET, как показано на рис. 37.6.

Рис. 37.6. Панель инструментов, доступная при редактировании страниц ASP.NET

Обратите внимание, что описания элементов управления ссылаются на свойства — во всех случаях соответствующие атрибуты в коде ASP.NET называются идентично. Отметим также, что этот раздел не претендует на то, чтобы служить полным руководством, поэтому многие элементы управления и свойства опущены. Здесь показаны только наиболее часто используемые из них. Элементы управления, рассматриваемые в этой главе, вы сможете найти в категориях Standard (Стандартные), Data (Данные) и Validation (Проверка). Категории Navigation and Login (Навигация и вход) и WebParts (Web-части) будут рассмотрены в следующей главе, а элементы управления AJAX Extensions — в главе 39. Элементы управления Reporting (Формирование отчетов), которые позволяют генерировать отчетную информацию, включая Crystal Reports, и которые должны быть представлены на Web-страницах, в этой книге не рассматриваются.

Стандартные Web-серверные элементы управления

Почти все Web-серверные элементы управления (в этой и других категориях) унаследованы от класса `System.Web.UI.WebControls.WebControl`, который, в свою очередь, унаследован от `System.Web.UI.Control`. Те же из них, которые не используют это наследование, наследуются либо напрямую от `Control`, либо от более специализированных базовых классов, которые в конечном итоге происходят от `Control`. Этим объясняется, что Web-серверные элементы управления имеют множество общих свойств и событий, которые можно использовать при необходимости. Их достаточно много, поэтому даже не предпринимается попытка описать их все здесь.

Многие из часто используемых унаследованных свойств имеют отношение к стилю отображения. Внешним видом элементов управления можно легко управлять, используя такие свойства, как `ForeColor`, `BackColor`, `Font` и т.д.; управлять можно также с помощью классов каскадных таблиц стилей (CSS). Это достигается установкой строкового свойства `CssClass` равным имени класса CSS из отдельного файла. Для работы с каскадными таблицами стилей можно использовать окно CSS Properties (Свойства CSS). Другие достойные упоминания свойства — это `Width` и `Height`, с помощью которых можно управлять размером элемента управления, `AccessKey` и `TabIndex` — для облегчения взаимодействия с пользователем, а также `Enabled` — для включения/отключения функциональности элемента управления в Web Forms.

Некоторые элементы управления могут содержать в себе другие элементы управления, таким образом, образуя иерархию в пределах одной страницы. Получить доступ к вложенным элементам управления можно через свойство `Controls`, а к контейнерному элементу управления — через свойство `Parent`.

Вероятно, вам придется использовать унаследованное событие `Load` чаще других — для инициализации элемента управления, и `PreRender` — для выполнения самых последних модификаций перед выводом HTML.

Существует большое количество других событий и свойств, и мы более подробно рассмотрим многие из них в следующей главе. В частности, следующая глава посвящена более сложным технологиям отображения и стиля. В табл. 37.1 описаны стандартные Web-серверные элементы управления.

Таблица 37.1. Стандартные Web-серверные элементы управления

Элемент управления	Описание
`Label`	Простое отображение текста; для установки и программной модификации отображаемого текста необходимо использовать свойство `Text`.
`TextBox`	Представляет текстовое поле, текст в котором можно редактировать. Для доступа к введенным данным следует использовать свойство `Text`, а событие `TextChanged` — для обработки изменений. Если необходима автоматическая обратная отправка (как альтернатива применению кнопки), установите свойство `AutoPostBack` равным `true`.
`Button`	Стандартная кнопка, на которой пользователь может щелкать. Свойство `Text` представляет текст на кнопке, а событие `Click` — реакцию на щелчки (обратная отправка на сервер выполняется автоматически). Можно также использовать событие `Command` для ответа на щелчки, что дает доступ к дополнительным свойствам `CommandName` и `CommandArgument` при приеме.
`LinkButton`	Аналогично `Button`, но отображает кнопку как гиперссылку.
`ImageButton`	Выводит графическое изображение, ведущее себя подобно кнопке, на котором можно щелкать. Свойства и события унаследованы от `Button` и `Image`.
`HyperLink`	Гиперссылка HTML. Адрес назначения устанавливается свойством `NavigateUrl`, а отображаемый текст — `Text`. Можно также использовать `ImageUrl` для указания графического изображения ссылки и `Target` — для указания целевого окна браузера. Этот элемент управления не имеет нестандартных событий, поэтому если нужна дополнительная обработка при переходе по ссылке, следует применять `LinkButton`.
`DropDownList`	Выпадающий список; дает возможность пользователю выбирать один элемент из списка возможных либо указанием его в списке, либо вводом первой буквы. Для установки списка элементов служит свойство `Items` (класс `ListItemCollection`, содержащий элементы типа `ListItem`), а свойства `SelectedItem` и `SelectedIndex` применяются для определения выбранного элемента. Событие `SelectedIndexChanged` может использоваться для определения того, что выбор изменился. Кроме того, доступно свойство `AutoPostBack`, указывающее на необходимость выполнения действия обратной отправки при смене выбора.
`ListBox`	Окно списка; дает возможность пользователю выбирать одну или более позиций списка. Для указания того, может ли быть выбрана только одна позиция или много, свойству `SelectionMode` присваивается значение, соответственно, `Multiple` или `Single`. Свойство `Rows` задает количество отображаемых элементов (строк). Все прочие свойства и события — как у `DropDownList`.
`CheckBox`	Отображает окошко, которое может быть помечено (флажок). Состояние сохраняется в свойстве булевского типа с именем `Checked`, а ассоциированный с ним текст — в свойстве `Text`. Свойство `AutoPostBack` может использоваться для инициирования автоматической обратной отправки, а событие `CheckedChanged` — для обработки изменений.
`CheckBoxList`	Группа флажков. Свойства и события идентичны другим списочным элементам управления, таким как `DropDownList`.

Продолжение табл. 37.1

Элемент управления	Описание
RadioButton	Отображает кнопку, которая может быть включена и выключена (переключатель). Обычно объединяется в группы, в которых может быть включена только одна кнопка. Для присоединения элемента управления RadioButton к группе используется свойство GroupName. Прочие свойства и события — как у CheckBox.
RadioButton List	Создает группу переключателей, из которых может быть выбран только один. Свойства и события — как у других списочных элементов управления, таких как DropDownList.
Image	Выводит графическое изображение. Для ссылки на изображение используется ImageUrl, а AlternateText — для вывода текста при невозможности загрузки изображения.
ImageMap	Подобно Image, но позволяет специфицировать определенные действия, которые нужно инициировать, когда пользователь щелкает на определенных "горячих областях" изображения. Эти действия могут быть связаны либо с обратной отправкой, либо с переадресацией по другому URL-адресу. Горячие области определяются встроенными элементами управления, унаследованными от HotSpot — такими как RectangleHotSpot и CircleHotSpot.
Table	Специфицирует таблицу. Используется в сочетании TableRow и TableCell на этапе проектирования либо позволяет программно присваивать строки через свойство Rows типа TableRowCollection. Это свойство также можно применять для модификаций во время выполнения. Этот элемент управления имеет некоторые стилистические свойства, уникальные для таблиц, вроде TableRow и TableCell.
BulletedList	Форматирует список с метками. В отличие от других списочных элементов управления имеет событие Click, которое можно применять при обратной отправке для определения того, на каком элементе списка щелкнул пользователь. Прочие свойства и события — как у DropDownList.
HiddenField	Применяется для создания скрытого поля, служащего для сохранения неотображаемых по любой причине значений. Это может оказаться очень удобным для сохранения настроек, для которых в противном случае понадобился бы специальный механизм сохранения. Свойство Value дает доступ к сохраненному значению.
Literal	Выполняет ту же функцию, что и Label, но не имеет свойств, задающих стиль, а только Text (поскольку унаследован от Control, а не от WebControl). Отображаемый текст задается с помощью свойства Text.
Calendar	Позволяет выбирать дату из графического календаря. Этот элемент управления имеет множество свойств, связанных со стилем, но основная функциональность обеспечивается свойствами SelectedDate и VisibleDate (типа System.DateTime), предоставляющим доступ к дате, выбранной пользователем, и отображаемому месяцу (который всегда содержит VisibleDate). Ключевое событие — SelectionChanged. Обратная отправка — автоматическая.

Окончание табл. 37.1

Элемент управления	Описание
`AdRotator`	Отображает несколько графических изображений последовательно, причем каждое следующее отображается после обращения к серверу. Свойство `AdvertisementFile` позволяет указать XML-файл, описывающий возможные изображения, а событие `AdCreated` выполняет обработку перед тем, как каждое изображение отправляется обратно. Можно также использовать свойство `Target` для именования окна, открываемого после щелчка.
`FileUpload`	Этот элемент управления предоставляет пользователю текстовое поле с кнопкой Browse (Обзор), позволяющей выбрать загружаемый файл. После того, как пользователь это сделает, можно обратиться к свойству `HasFile`, чтобы убедиться, что файл выбран, а затем применить метод `SaveAs()` для выполнения загрузки файла на сервер.
`Wizard`	Мастер. Сложный элемент управления, упрощающий общую задачу получения нескольких страниц пользовательского ввода за один прием. Вы можете добавить множество шагов к мастеру, которые будут представлены пользователю последовательно либо не последовательно, и положиться на него в сохранении состояния и тому подобного.
`Xml`	Наиболее сложный элемент управления для отображения текста. Используется для показа содержимого XML, которое может быть трансформировано с применением таблицы стилей XSLT. Содержимое XML устанавливается с использованием одного из свойств — `Document`, `DocumentContent` или `DocumentSource` (в зависимости от формата исходного XML) и таблицы стилей XSLT (не обязательно), используя либо `Transform`, либо `TransformSource`.
`MultiView`	Элемент управления, содержащий в себе один или более элементов `View`, причем в конкретный момент времени отображается только один `View`. Текущий активный элемент `View` указывается с помощью `ActiveViewIndex`, и можно определить, изменялся ли он (возможно, через ссылку `Next`) с помощью события `ActiveViewChanged`.
`Panel`	Контейнер для других элементов управления. Для указания размещения элементов можно использовать свойства `HorizontalAlign` и `Wrap`.
`PlaceHolder`	Этот элемент управления не отображает никакого вывода, но может оказаться удобным для группирования других элементов управления или программного добавления их в заданную позицию. Включенные элементы управления доступны через свойство `Controls`.
`View`	Контейнер для элементов управления, подобный `PlaceHolder`, но включенный в `MultiView`. Можно указать его видимость через атрибут `Visible`, а также использовать события `Activate` и `Deactivate` для обнаружения изменений в состоянии активности.
`Substitution`	Специфицирует раздел Web-страницы, который не кэшируется вместе с остальным выводом. Это усовершенствованное средство, связанное с поведением кэширования ASP.NET, которое в настоящей книге нам не понадобится.
`Localize`	Выполняет ту же функцию, что и `Literal`, но разрешает локализовать текст с помощью ресурсов проекта, позволяющих задать текст для отображения в разных местах.

Web-серверные элементы управления для работы с данными

Эти элементы управления подразделяются на два типа:

- элементы управления источников данных (`SqlDataSource`, `AccessDataSource`, `ObjectDataSource`, `XmlDataSource` и `SiteMapDataSource`);
- элементы управления отображения данных (`GridView`, `DataList`, `DetailsView`, `FormView`, `ListView`, `Repeater` и `DataPager`).

Обычно на страницу помещается один из (невизуальных) элементов управления источников данных для связи с хранилищем данных; затем добавляются элементы управления для отображения данных, привязанные к источнику. Некоторые из наиболее усовершенствованных элементов управления для отображения данных вроде `GridView` также позволяют редактировать данные.

Все элементы источников данных наследуются от `System.Web.UI.DataSource` или `System.Web.UI.HierarchicalDataSource`. Эти классы имеют методы, подобные `GetView()` (или `GetHierarchicalView()`), обеспечивающие доступ к внутренним представлениям данных и обладающие возможностями настройки внешнего вида.

В табл. 37.2 описаны различные элементы управления источников данных. Обратите внимание, что здесь мы менее детально описываем свойства, чем в других разделах — в основном потому, что конфигурирование этих элементов лучше выполнять графически, применяя мастера. Далее в этой главе мы покажем некоторые из них в действии.

Таблица 37.2. Элементы управления источников данных

Элемент управления	Описание
`SqlDataSource`	Служит каналом для данных, хранимых в базе SQL Server. Поместив этот элемент на страницу, можно манипулировать данными SQL Server с помощью элементов управления, отображающих данные. Далее в этой главе мы покажем его в действии.
`AccessDataSource`	Как `SqlDataSource`, но работает с данными из базы Access.
`LinqDataSource`	Позволяет манипулировать объектами в модели данных, разрешающей использование LINQ.
`ObjectDataSource`	Этот элемент управления позволяет манипулировать данными, сохраненными в созданных вами объектах, которые могут быть сгруппированы в класс коллекции. Это может быть очень быстрый способ представить пользовательскую объектную модель на странице ASP.NET.
`XmlDataSource`	Позволяет осуществлять привязку к данным XML. Например, его удобно применять в связке с элементом управления `TreeView` (один из элементов навигации). Можно также при необходимости трансформировать данные XML с помощью таблицы стилей XSL.
`SiteMapDataSource`	Позволяет связаться с иерархическими данными карты сайта. См. далее раздел, посвященный Web-серверным элементам управления для навигации.

В табл. 37.3 представлено описание элементов управления, предназначенных для отображения данных. Некоторые из них подходят для разных целей. Некоторые более функциональны, чем другие, но их всегда можно использовать упрощенным образом (например, когда нет необходимости в редактировании элементов данных).

Таблица 37.3. Элементы управления для отображения данных

Элемент управления	Описание
`GridView`	Отображает множественные элементы данных (такие как записи базы данных) в форме строк, где каждая строка состоит из столбцов, отображающих поля данных. Манипулируя свойствами этого элемента управления, можно выбирать, сортировать и редактировать элементы данных.
`DataList`	Отображает множественные элементы данных, где для каждого элемента можно применять произвольный шаблон отображения. Как и в случае `GridView`, можно выбирать, сортировать и редактировать элементы данных.
`DetailsView`	Отображает единственный элемент данных (запись базы данных) в табличной форме, где каждая строка данных представляет отдельное поле. Позволяет добавлять, редактировать и удалять элементы данных.
`FormView`	Отображает единственный элемент данных, используя шаблон. Как и `DetailsView`, позволяет добавлять, редактировать и удалять элементы данных.
`ListView`	Подобен `DataList`, но с дополнительной поддержкой разбивки на страницы посредством `DataPager` и дополнительных возможностей шаблонов.
`Repeater`	Подобно `DataList`, но без возможностей выбора и редактирования.
`DataPager`	Позволяет выполнять разбивку на страницы в элементах управления `ListView`.

Web-серверные элементы управления для проверки достоверности

Элементы управления для проверки достоверности предоставляют метод контроля пользовательского ввода в большинстве случаев без необходимости написания кода. Когда инициируется обратная отправка, каждый такой элемент управления проверяет связанный с ним элемент управления и соответствующим образом изменяет его свойство `IsValid`. Если это свойство принимает значение `false`, значит, пользовательский ввод в этом элементе управления не удовлетворяет требованиям проверки. При этом свойство `IsValid` всей включающей страницы получает значение `false`. Код серверной стороны может проверить это свойство и соответствующим образом обработать.

Элементы управления для проверки достоверности имеют также и другую функцию. Они не только проверяют содержимое во время выполнения, но также могут выдавать полезные подсказки пользователю. Простое присвоение текста свойству `ErrorMessage` позволяет информировать пользователя о том, что он ввел некорректные данные.

Текст, сохраненный в `ErrorMessage`, может быть выдан в той точке, где находится элемент управления для проверки, или же в любой другой точке вместе с сообщениями от других проверяющих элементов, расположенных на странице. Такое поведение обеспечивается применением элемента `ValidationSummary`, который по требованию отображает сообщения об ошибке вместе с необходимым дополнительным текстом.

В браузерах, поддерживающих их, эти элементы управления даже генерируют функции JavaScript, выполняемые на стороне клиента, чтобы упростить поведение проверки. Это значит, что в некоторых случаях даже не возникает необходимость в обратной отправке, поскольку проверяющие элементы управления могут справиться с некорректным вводом и выдать сообщение, вообще не обращаясь к серверу.

Все проверяющие элементы управления унаследованы от `BaseValidator`, а потому разделяют ряд важных свойств. Возможно, наиболее важное из них — `ErrorMessage`, упомянутое ранее, а также свойство `ControlToValidate`, указывающее

программный идентификатор проверяемого элемента управления. Другим важным свойством является Display, которое определяет, нужно ли помещать текст в позиции итоговой проверки страницы (если установлено none) или же в позиции проверяющего элемента. Кроме того, есть возможность оставить пространство для сообщения об ошибке, даже если она не отображается (установить Display в значение Static), или же динамически выделять место при необходимости, что может несколько сдвинуть содержимое страницы (установить Display в значение Dynamic). В табл. 37.4 описаны элементы управления для проверки достоверности.

Таблица 37.4. Элементы управления для проверки достоверности

Элемент управления	Описание
RequiredFieldValidator	Применяется для проверки, ввел ли пользователь данные в такой элемент управления, как TextBox.
CompareValidator	Используется для проверки введенных данных на соответствие некоторым простым требованиям с помощью набора операций, заданных свойством Operator и ValueToCompare, с которым нужно сравнивать. Operator может принимать значения Equal, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, NotEqual и DataTypeCheck. DataTypeCheck просто сравнивает тип данных ValueToCompare с данными в проверяемом элементе управления. Свойство ValueToCompare является строковым, но его тип данных интерпретируется в соответствии с его содержимым. Чтобы можно было выполнять такую проверку, необходимо установить свойство Type в значение Currency, Date, Double, Integer или String.
RangeValidator	Проверяет попадание данных в элементе управления в диапазон значений, заданный свойствами MinimumValue и MaximumValue. Как и CompareValidator, имеет свойство Type.
RegularExpressionValidator	Проверяет содержимое поля на основе регулярного выражения, содержащегося в свойстве ValidationExpression. Это может быть удобно для проверки известных последовательностей — таких как почтовые коды, телефонные номера, IP-адреса и тому подобное.
CustomValidator	Используется для проверки данных в элементе управления с применением пользовательской функции. Свойство ClientValidationFunction позволяет указать пользовательскую функцию на стороне клиента, используемую для проверки (что, к сожалению, исключает возможность применения C#). Функция должна возвращать булевское значение, обозначающее успешность проверки. В качестве альтернативы можно использовать событие ServerValidate для указания функции проверки на стороне сервера. Эта функция — обработчик события типа bool, принимающая строку с проверяемыми данными вместо параметра EventArgs. Возвращает true, если проверка прошла успешно, и false — в противном случае.
ValidationSummary	Отображает ошибку проверки для всех проверяющих элементов управления, у которых установлено свойство ErrorMessage. Отображение может быть форматировано установкой свойств DisplayMode (BulletList, List или SingleParagraph) и HeaderText. Отображение может быть отключено установкой ShowSummary в false или выглядеть как всплывающее окно сообщений, если установить свойство ShowMessage равным true.

Пример применения серверных элементов управления

В данном примере мы создадим каркас Web-приложения для заказа комнаты совещаний (как и в случае других примеров этой книги, код этого примера доступен на прилагаемом компакт-диске). Для начала включим в приложение только один интерфейс и простую обработку событий; позже мы расширим его средствами ADO.NET и привязкой к данным, чтобы подключить бизнес-логику на стороне сервера.

Web-форма, которую мы создадим, будет содержать поля для имени пользователя, имени события, комнаты совещаний и списка участников совещания, вместе с календарем для выбора даты (в этом приложении предполагается, что комната будет заказываться на весь день). Ко всем полям кроме календаря подключим проверяющие элементы управления серверной стороны, а также предусмотрим ввод даты по умолчанию, если никакая дата не введена.

Для тестирования пользовательского интерфейса на форме используем также элемент `Label`, служащий для отображения результатов подтверждения.

Начнем с создания нового Web-сайта в Visual Studio .NET в каталоге `C:\ProCSharp\Chapter37\` и назовем его `PCSWebApp2`. Далее изменим код в файле `Default.aspx`, как показано ниже:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
     <title>Meeting Room Booker</title>
  </head>
  <body>
     <form id="form1" runat="server">
       <div>
            <h1 style="text-align: center;">
                 Enter details and set a day to initiate an event.
            </h1>
       </div>
```

После заголовка страницы (заключенного в HTML-дескрипторы `<h1>` для получения увеличенного стиля текста заголовка) основное тело формы заключено в HTML-дескриптор `<table>`. Можно было бы применить таблицу — Web-серверный элемент управления — но это привнесло бы излишнюю сложность, поскольку пока мы используем таблицу только для форматирования отображения, а не как часть элемента динамического интерфейса пользователя. Это важный момент, который нужно иметь в виду при разработке Web Forms — не добавлять Web-серверных элементов управления без необходимости. Таблица содержит три столбца: первый хранит простые текстовые метки, второй — поля ввода, соответствующие меткам первого столбца (вместе с проверяющими элементами управления); третий столбец содержит календарь для выбора даты, который распространяется на четыре строки таблицы. Пятая строка содержит кнопку отправки, растянутую на ширину всех столбцов, а шестая — элемент управления `ValidationSummary` для отображения сообщений об ошибках при необходимости (все остальные проверяющие элементы управления имеют значение свойства `Display="None"`, потому что они будут использовать этот итоговый элемент для отображения сообщений). Ниже таблицы находится простая метка, которую пока мы сможем использовать для отображения результата, прежде чем позднее добавим доступ в базу данных.

```
<div style="text-align: center; ">
  <table style="text-align: left; border-color: #000000; border-width: 2px;
       background-color: #fff99e;" cellspacing="0" cellpadding="8"
       rules="none" width="540">
       <tr>
          <td valign="top">
             Your Name:</td>
          <td valign="top">
             <asp:TextBox ID="nameBox" Runat="server" Width="160px" />
             <asp:RequiredFieldValidator ID="validateName" Runat="server"
             ErrorMessage="You must enter a name."
             ControlToValidate="nameBox" Display="None" />
          </td>
          <td valign="middle" rowspan="4">
             <asp:Calendar ID="calendar" Runat="server" BackColor="White" />
          </td>
       </tr>
       <tr>
          <td valign="top">
             Event Name:</td>
          <td valign="top">
             <asp:TextBox ID="eventBox" Runat="server" Width="160px" />
             <asp:RequiredFieldValidator ID="validateEvent" Runat="server"
             ErrorMessage="You must enter an event name."
             ControlToValidate="eventBox" Display="None" />
          </td>
       </tr>
```

Бо́льшая часть кода ASP.NET в этом файле совершенно проста, и бо́льшая его часть может быть понята после первого прочтения. Единственное, что следует отметить — это способ, которым элементы списка присоединены к элементам управления для выбора комнаты совещаний, а также способ указания множества участников совещания:

```
       <tr>
          <td valign="top">
            Meeting Room:</td>
          <td valign="top">
             <asp:DropDownList ID="roomList" Runat="server" Width="160px">
             <asp:ListItem Value="1">The Happy Room</asp:ListItem>
             <asp:ListItem Value="2">The Angry Room</asp:ListItem>
             <asp:ListItem Value="3">The Depressing Room</asp:ListItem>
             <asp:ListItem Value="4">The Funked Out Room</asp:ListItem>
            </asp:DropDownList>
            <asp:RequiredFieldValidator ID="validateRoom" Runat="server"
              ErrorMessage="You must select a room."
              ControlToValidate="roomList" Display="None" />
          </td>
       </tr>
       <tr>
          <td valign="top">
            Attendees:</td>
          <td valign="top">
            <asp:ListBox ID="attendeeList" Runat="server" Width="160px"
              SelectionMode="Multiple" Rows="6">
              <asp:ListItem Value="1">Bill Gates</asp:ListItem>
              <asp:ListItem Value="2">Monica Lewinsky</asp:ListItem>
              <asp:ListItem Value="3">Vincent Price</asp:ListItem>
              <asp:ListItem Value="4">Vlad the Impaler</asp:ListItem>
              <asp:ListItem Value="5">Iggy Pop</asp:ListItem>
              <asp:ListItem Value="6">William Shakespeare</asp:ListItem>
            </asp:ListBox>
```

Здесь объекты ListItem ассоциируются с двумя Web-серверными элементами управления. Строго говоря, сами по себе они не являются Web-серверными элементами управления (наследуются просто от System.Object), а потому для них не нужно указывать Runat="server". Когда страница обрабатывается, элементы <asp:ListItem> используются для создания объектов ListItem, добавляемых в коллекцию родительского списочного элемента управления. Это облегчает написание кода инициализации списков, избавляя нас от ручного кодирования (иначе пришлось бы создавать объект ListItemCollection, добавлять ему объекты ListItem и потом передавать коллекцию элементу управления — списку). Конечно, при желании все это можно сделать программно:

```
                <asp:RequiredFieldValidator ID="validateAttendees" Runat="server"
                 ErrorMessage="You must have at least one attendee."
                 ControlToValidate="attendeeList" Display="None" />
                </td>
            </tr>
            <tr>
                <td align="center" colspan="3">
                 <asp:Button ID="submitButton" Runat="server" Width="100%"
                     Text="Submit meeting room request" />
                </td>
            </tr>
            <tr>
                <td align="center" colspan="3">
                 <asp:ValidationSummary ID="validationSummary" Runat="server"
                     HeaderText="Before submitting your request:" />
                </td>
            </tr>
           </table>
       </div>
       <div>
           <p>
           Results:
           <asp:Label Runat="server" ID="resultLabel" Text="None." />
           </p>
       </div>
 </form>
</body>
</html>
```

В представлении дизайнера созданная нами форма будет выглядеть так, как показано на рис. 37.7. Это полнофункциональный пример пользовательского интерфейса, который поддерживает свое собственное состояние между запросами и проверяет пользовательский ввод.

Учитывая краткость приведенного выше кода, это впечатляет. Фактически, нам мало чего остается сделать, по крайней мере, с этим примером; нужно только специфицировать событие обработки щелчка на кнопке подтверждения.

На самом деле, это еще не все. До сих пор мы не предусмотрели проверки для элемента управления — календаря. Все что нужно, для этого сделать — присвоить ему начальное значение. Это можно сделать в обработчике события Page_Load() в файле отделенного кода:

```
private void Page_Load(object sender, EventArgs e)
{
   if (!this.IsPostBack)
   {
     calendar.SelectedDate = System.DateTime.Now;
   }
}
```

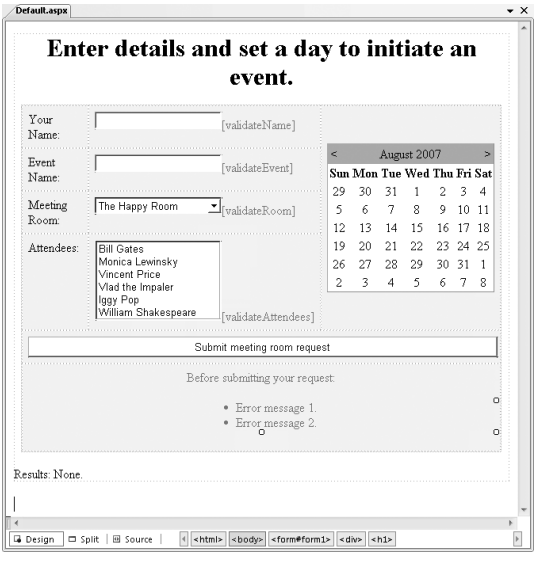

Рис. 37.7. Созданная форма для заказа комнаты совещаний

Здесь для начала выбирается текущая дата. Обратите внимание, что свойство `IsPostBack` страницы используется для того, чтобы определить, был ли вызван метод `Page_Load()` в результате действия обратной отправки. Если обратная отправка все еще выполняется, значение этого свойства будет `true`, тогда мы оставляем выбранную дату в покое (поскольку не хотим потерять выбор, сделанный пользователем).

Для добавления обработчика щелчка на кнопке необходимо просто дважды щелкнуть на кнопке в визуальном дизайнере и ввести следующий код:

```
private void submitButton_Click(object sender, EventArgs e)
{
   if (this.IsValid)
   {
      resultLabel.Text = roomList.SelectedItem.Text +
            " has been booked on " +
            calendar.SelectedDate.ToLongDateString() +
            " by " + nameBox.Text + " for " +
            eventBox.Text + " event. ";
      foreach (ListItem attendee in attendeeList.Items)
      {
            if (attendee.Selected)
            {
                  resultLabel.Text += attendee.Text + ", ";
            }
      }
      resultLabel.Text += " and " + nameBox.Text + " will be attending.";
}
```

Здесь мы просто устанавливаем свойство `Text` элемента управления `resultLabel` в значение результирующей строки, которая затем появится под главной таблицей.

В браузере Internet Explorer результат такой передачи может выглядеть примерно так, как показано на рис. 37.8, если только не возникнут ошибки, которые вместо этого приведут к активизации `ValidationSummary` (рис. 37.9).

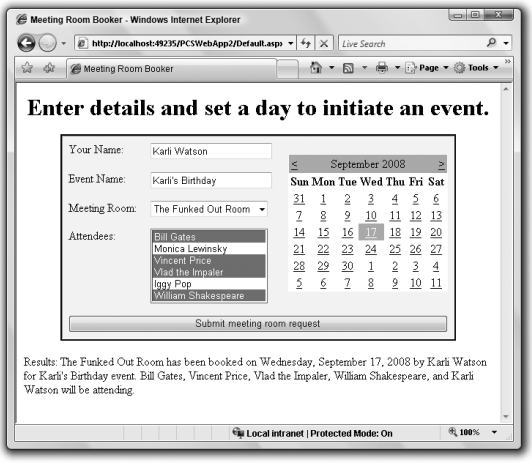

Рис. 37.8. Передача информации о заказе комнаты совещаний

Рис. 37.9. Ошибки, связанные с отсутствием сведений о совещании и об участниках

ADO.NET и привязка данных

Приложение Web Forms, которое мы разработали в предыдущем разделе, вполне функционально, но содержит только статические данные. В дополнение процесс заказа не предусматривает хранения постоянных данных о событиях. При решении обеих проблем мы можем воспользоваться ADO.NET для организации доступа к информации, хранящейся в базе данных, с тем, чтобы получить возможность сохранять и извлекать данные о событиях вместе со списками комнат и участников. Привязка данных делает процесс извлечения информации еще проще. Такие элементы управления, как окна списков (и некоторые из более специализированных элементов, которые мы рассмот-

рим позже), готовы для применения этой технологии. Они могут быть привязаны к любому объекту, реализующему интерфейс IEnumerable, ICollection или IlistSource, а также к любому источнику данных Web-серверных элементов управления.

В этом разделе мы начнем с обновления нашего приложения, чтобы оно было способно работать с данными, а затем перейдем к рассмотрению других результатов, которых можно достичь с помощью привязки данных к некоторым другим Web-серверным элементам управления.

Усовершенствование приложения заказа мероприятий

Чтобы отделить нововведения от предыдущего примера, создадим новый Web-сайт с именем PCSWebApp3 в каталоге C:\ProCSharp\Chapter37\ и скопируем код ранее созданного нами приложения PCSWebApp2. Прежде чем начать писать новый код, взглянем на базу данных, с которой нам придется иметь дело.

База данных

Для данного примера мы используем базу данных Microsoft SQL Server Express под названием MeetingRoomBooker.mdf, которая представляет часть кода примеров этой книги, доступного для загрузки. Для приложений масштаба предприятия имело бы смысл использовать полную базу данных SQL Server, но почти идентичная технология разработки, которую предполагает SQL Server Express, позволяет несколько упростить тестирование. Код также будет идентичен.

Если вы добавляете собственную версию этой базы данных, то для этого придется добавить новую базу в папку App_Data в проводнике Solution Explorer. Для этого щелкните правой кнопкой мыши на папке App_Data и выберите в контекстном меню пункт Add New Item (Добавить новый элемент). Выберите базу данных, назвав ее MeetingRoomBooker, и щелкните на кнопке Add (Добавить). При этом в Solution Explorer также будет настроено соединение с источником данных, готовое к использованию. Далее вы сможете добавить необходимые таблицы, как показано в следующих разделах, и ввести свои собственные данные. В качестве альтернативы можно воспользоваться готовой базой данных, загруженной вместе с кодом; просто скопируйте ее в каталог App_Data вашего Web-сайта.

Предложенная база данных содержит три таблицы:

- Attendees, содержащая список участников мероприятия;
- Rooms, содержащая список возможных комнат для проведения мероприятий;
- Events, содержащая список запланированных мероприятий.

Attendees

Таблица Attendees содержит столбцы, перечисленные в табл. 37.5.

Таблица 37.5. Структура таблицы Attendees

Столбец	Тип	Примечания
ID	Identity, первичный ключ	Идентификационный номер участника
Name	varchar, обязательное, 50 символов	Имя участника
Email	varchar, не обязательное, 50 символов	Адрес электронной почты участника

Поставляемая база данных включает 20 записей участников, все с их собственными (фиктивными) адресами электронной почты. Можно предположить, что в более развитом приложении сообщения электронной почты могут рассылаться автоматически

участникам при приеме заказа, но это мы оставляем вам в качестве дополнительного упражнения.

Rooms

Таблица Rooms включает в себя столбцы, перечисленные в табл. 37.6.

Таблица 37.6. Структура таблицы Rooms

Столбец	Тип	Примечания
ID	Identity, первичный ключ	Идентификационный номер комнаты
Room	varchar, обязательное, 50 символов	Наименование комнаты

В базе данных содержится 20 записей о комнатах.

Events

Таблица Events содержит столбцы, перечисленные в табл. 37.7.

Таблица 37.7. Структура таблицы Events

Столбец	Тип	Примечания
ID	Identity, первичный ключ	Идентификационный номер мероприятия
Name	varchar, обязательное, 255 символов	Наименование мероприятия
Room	int, обязательное	ID комнаты проведения мероприятия
AttendeeList	text, обязательное	Список имен участников
EventDate	datetime, обязательное	Дата события

В загруженной базе данных присутствует несколько записей о мероприятиях.

Привязка к базе данных

Мы привяжем к данным два элемента управления — attendeeList и roomList. Прежде чем сделать это, необходимо добавить Web-серверные элементы управления SqlDataSource, отображающиеся на таблицы из базы данных MeetingRoomBooker.mdf. Самый быстрый способ сделать это — перетащить с помощью мыши из панели инструментов на Web-форму Default.aspx и настроить его посредством мастера конфигурации. На рис. 37.10 показано, как использовать этот мастер для элемента управления SqlDataSource.

На первой странице мастера настройки источника данных необходимо выбрать соединение с созданной ранее базой. Затем сохраните строку соединения как MRBConnectionString, после этого выберите * (все поля) из таблицы Attendees базы данных.

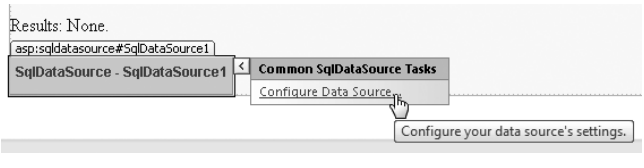

Рис. 37.10. Вызов мастера конфигурации для элемента управления SqlDataSource

После завершения работы мастера измените ID элемента управления SqlDataSource на MRBAttendeeData. Вам нужно также добавить и сконфигурировать еще два элемента управления SqlDataSource для получения данных из таблиц Rooms и Events — это элементы MRBRoomData и MRBEventData, соответственно. Для их подключения используйте сохраненную строку MRBConnectionString.

Добавив эти источники данных, мы увидим в коде формы очень простой синтаксис:

```
<asp:SqlDataSource ID="MRBAttendeeData" runat="server"
   ConnectionString="<%$ ConnectionStrings:MRBConnectionString %>"
   SelectCommand="SELECT * FROM [Attendees]"></asp:SqlDataSource>
<asp:SqlDataSource ID="MRBRoomData" runat="server"
   ConnectionString="<%$ ConnectionStrings:MRBConnectionString %>"
   SelectCommand="SELECT * FROM [Rooms]"></asp:SqlDataSource>
<asp:SqlDataSource ID="MRBEventData" runat="server"
   ConnectionString="<%$ ConnectionStrings:MRBConnectionString %>"
   SelectCommand="SELECT * FROM [Events]"></asp:SqlDataSource>
```

Определение используемой строки соединения находится в файле web.config, который мы рассмотрим в настоящей главе чуть позже.

Далее потребуется установить свойства привязки элементов управления roomList и attendeeList. Для roomList необходимые настройки будут следующими:

- DataSourceID — MRBRoomData
- DataTextField — Room
- DataValueField — ID

Аналогично следует установить свойства привязки для attendeeList:

- DataSourceID — MRBAttendeeData
- DataTextField — Name
- DataValueField — ID

В результате запуска этого приложения получим полный доступ к информации о комнатах и участниках совещаний через привязанные к данным элементы управления. Элемент MRBEventData используем чуть позже.

Настройка календаря

Прежде чем добавлять события в базу данных, нам нужно несколько модифицировать отображение календаря. Было бы неплохо иметь возможность показать другим цветом дни, на которые уже приняты заказы на конкретную комнату, дабы предотвратить повторный выбор этих дней. Требуемые модификации касаются способа установки дат в календаре и способа отображения ячеек-дней.

Начнем с выбора дат. При заказе мероприятия даты нужно проверять в трех местах и соответствующим образом изменять выбор: когда устанавливается начальная дата в Page_Load(), когда пользователь пытается выбрать дату из календаря, и когда мероприятие заказывается, а мы хотим установить новую дату для предотвращения повторного заказа на один и тот же день. Поскольку, похоже, что это должно быть часто используемым средством, можно создать приватный метод для реализации таких вычислений. Этот метод должен принимать в качестве параметра пробную дату и возвращать доступную, которая должна либо совпадать с пробной, либо представлять следующий свободный день после пробной даты.

Прежде чем добавить этот метод, нашему коду нужно предоставить доступ к таблице Event. Для этого можно воспользоваться элементом управления MRBEventData, потому что он способен наполнить DataView. Чтобы облегчить это, добавим следующий приватный метод и свойство:

```
private DataView eventData;
private DataView EventData
{
  get
  {
       if (eventData == null)
       {
            eventData =
             MRBEventData.Select(new DataSourceSelectArguments()) as DataView;
       }
       return eventData;
  }
  set
  {
     eventData = value;
  }
}
```

Свойство `EventData` присваивает члену `eventData` значение требуемой даты с результатом, кэшированным для последующего использования. Здесь мы вызываем метод `SqlDataSource.Select()` для получения `DataView`.

Далее добавим этот метод `GetFreeDate()` в файл отделенного кода:

```
private DateTime GetFreeDate(DateTime trialDate)
{
  if (EventData.Count > 0)
  {
       DateTime testDate;
       bool trialDateOK = false;
       while (!trialDateOK)
       {
            trialDateOK = true;
            foreach (DataRowView testRow in EventData)
            {
                testDate = (DateTime)testRow["EventDate"];
                if (testDate.Date == trialDate.Date)
                {
                  trialDateOK = false;
                  trialDate = trialDate.AddDays(1);
                }
            }
       }
  }
  return trialDate;
}
```

Этот простой код использует `DataView` по имени `EventData` для извлечения данных о мероприятиях. Сначала мы проверяем тривиальный случай, когда на заказанный день ничего не запланировано — при этом можно просто подтвердить заказанную дату, возвратив ее. Затем выполняем итерацию по датам в таблице `Event`, сравнивая их с пробной датой. Если находим совпадение, добавляем один день к пробной дате и повторяем поиск.

Извлечение даты из `DataTable` достаточно просто:

```
testDate = (System.DateTime)testRow["EventDate"];
```

Приведение данных столбца к `System.DateTime` работает хорошо.

Первое место, где будет использоваться `getFreeDate()` — опять-таки `Page_Load()`. Это означает внесение небольших изменений в код, устанавливающий свойство календаря `SelectDate`:

```
if (!this.IsPostBack)
{
  DateTime trialDate = DateTime.Now;
  calendar.SelectedDate = GetFreeDate(trialDate);
}
```

Далее потребуется ответить на выбор даты в календаре. Чтобы сделать это, просто добавим обработчик события `SelectionChanged` календаря и выполним в нем проверку данных по существующим записям о мероприятиях. Для этого потребуется выполнить двойной щелчок на календаре в дизайнере и добавить следующий код:

```
void calendar_SelectionChanged(object sender, EventArgs e)
{
  DateTime trialDate = calendar.SelectedDate;
  calendar.SelectedDate = GetFreeDate(trialDate);
}
```

Этот код практически идентичен тому, что есть в `Page_Load()`.

Третье место, где необходимо выполнять эту проверку — обработчик щелчка на кнопке заказа. Мы вернемся к этому ниже, поскольку перед этим нужно внести еще несколько изменений.

Далее потребуется окрасить ячейки занятых дней в календаре, чтобы отметить запланированные мероприятия. Чтобы сделать это, добавим обработчик события `DayRender` объекта календаря. Это событие возбуждается всякий раз при отображении отдельного дня, и дает доступ к объекту отображаемой ячейки и соответствующей ей дате через свойства `Cell` и `Date` параметра `DayRenderEventArgs`, принимаемого функцией обработчика. Дату отображаемой ячейки мы просто сравним с датами в объекте `eventTable` и если найдем соответствие, окрасим, используя свойство `Cell.BackColor`:

```
void calendar_DayRender(object sender, DayRenderEventArgs e)
{
   if (EventData.Count > 0)
   {
     DateTime testDate;
     foreach (DataRowView testRow in EventData)
     {
          testDate = (DateTime)testRow["EventDate"];
          if (testDate.Date == e.Day.Date)
          {
               e.Cell.BackColor = System.Drawing.Color.Red;
          }
     }
   }
}
```

Здесь используется красный цвет и получается в результате внешний вид календаря, показанный на рис. 37.11, где на 12, 15 и 22 июня 2008 г. запланированы мероприятия, а пользователь выбрал 24 июня.

<	June 2008					>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Рис. 37.11. Внешний вид календаря

С добавлением логики выбора даты теперь невозможно выбрать день, окрашенный красным; если предпринять такую попытку, то выбирается следующая свободная дата (например, при попытке указать 15 июня будет выбрано 16 июня).

Добавление мероприятий в базу данных

Обработчик события `submitButton_Click()` сейчас собирает строку из характеристик мероприятия и отображает ее

в элементе управления `resultLabel`. Чтобы добавить запись о мероприятии в базу данных, мы просто переформатируем созданную строку в SQL-запрос `INSERT` и выполним его.

Обратите внимание, что в среде разработки вам не нужно особо беспокоиться о безопасности. Добавление базы SQL Server 2005 Express через решение Web-сайта и конфигурирование элементов управления `SqlDataSource` для автоматического применения дает вам строку соединения, которую можно использовать для осуществления записи в базу данных. В более сложных ситуациях вам может понадобиться обращаться к ресурсам, используя другие учетные записи — например, учетную запись домена, которая применяется для доступа к экземпляру SQL Server, находящемуся где-то в сети. Такая возможность (через заимствование прав, службы COM+ или другим способом) присутствует в ASP.NET, но не входит в круг тем, освещаемых в этой главе. В большинстве случаев конфигурирование строки соединения не сложнее тех вещей, которые нужно получить.

Большая часть следующего кода выглядит знакомо:

```
void submitButton_Click(object sender, EventArgs e)
{
  if (this.IsValid)
  {
     System.Text.StringBuilder sb = new System.Text.StringBuilder();
     foreach (ListItem attendee in attendeeList.Items)
     {
        if (attendee.Selected)
        {
           sb.AppendFormat("{0} ({1}), ", attendee.Text, attendee.Value);
        }
     }
     sb.AppendFormat(" and {0}", nameBox.Text);
     string attendees = sb.ToString();

     try
     {
       System.Data.SqlClient.SqlConnection conn =
            new System.Data.SqlClient.SqlConnection(
             ConfigurationManager.ConnectionStrings["MRBConnectionString"]
                  .ConnectionString);
       System.Data.SqlClient.SqlCommand insertCommand =
            new System.Data.SqlClient.SqlCommand("INSERT INTO [Events] "
             + "(Name, Room, AttendeeList, EventDate) VALUES (@Name, "
             + "@Room, @AttendeeList, @EventDate)", conn);
       insertCommand.Parameters.Add(
            "Name", SqlDbType.VarChar, 255).Value = eventBox.Text;
       insertCommand.Parameters.Add(
            "Room", SqlDbType.Int, 4).Value = roomList.SelectedValue;
       insertCommand.Parameters.Add(
            "AttendeeList", SqlDbType.Text, 16).Value = attendees;
       insertCommand.Parameters.Add(
             "EventDate", SqlDbType.DateTime, 8).Value = calendar.SelectedDate;
```

Самое интересное здесь — обращение к созданной ранее строке соединения с применением следующего синтаксиса:

```
ConfigurationManager.ConnectionStrings["MRBConnectionString"].ConnectionString
```

Класс `ConfigurationManager` предоставляет доступ ко всей систематизированной конфигурационной информации нашего Web-приложения, сохраненной в файле `Web.config`. Позднее в этой главе мы рассмотрим это в деталях.

После создания команды SQL мы применяем ее для вставки новой записи о мероприятии:

```
conn.Open();
int queryResult = insertCommand.ExecuteNonQuery();
conn.Close();
```

`ExecuteNonQuery()` возвращает целочисленное значение, представляющее количество строк таблицы, обработанной запросом. Если это значение равно `1`, значит, вставка записи выполнена успешно. В этом случае сообщение об успехе помещается в `resultLabel`, очищается `EventData`, поскольку она устарела, и выбор в календаре перемещается на новую свободную дату. Поскольку `GetFreeDate()` предполагает использование `EventData`, а свойство `EventData` автоматически обновляется, если не содержит даты, сохраняемая дата мероприятия обновляется:

```
if (queryResult == 1)
{
  resultLabel.Text = "Event Added.";
  EventData = null;
  calendar.SelectedDate = GetFreeDate(calendar.SelectedDate.AddDays(1));
}
```

Если `ExecuteNonQuery()` возвращает число, отличное от `1` — это признак возникновения проблемы. Код в этом примере возбуждает исключение, если возвращается число, отличное от `1`. Это исключение перехватывается главным блоком `catch`, куда помещен код доступа к базе данных. Этот блок просто отображает общее уведомление об ошибке в `resultLabel`.

```
      else
      {
          throw new System.Data.DataException("Unknown data error.");
      }
    }
    catch
    {
       resultLabel.Text = "Event not added due to DB access " + "problem.";
    }
  }
}
```

На этом завершается наша версия программы заказа мероприятий с доступом к данным.

Дополнительные сведения о связывании данных

Как упоминалось ранее в этой главе, среди доступных Web-серверных элементов управления несколько предназначены для отображения данных (`GridView`, `DataList`, `DetailsView`, `FormView` и `Repeater`). Все они исключительно полезны, когда речь идет об отображении данных на Web-странице, поскольку автоматически выполняют многие задачи, которые в противном случае потребовали бы значительного объема кодирования.

Для начала посмотрим, насколько просто их использовать, добавив в приложение отображаемый список мероприятий, который будет отображаться в нижней части экрана `PCSWebApp3`.

Перетащим элемент управления `GridView` из панели инструментов в нижнюю часть `Default.aspx` и выберем источник данных `MRBEventData`, который был добавлен ранее (рис. 37.12).

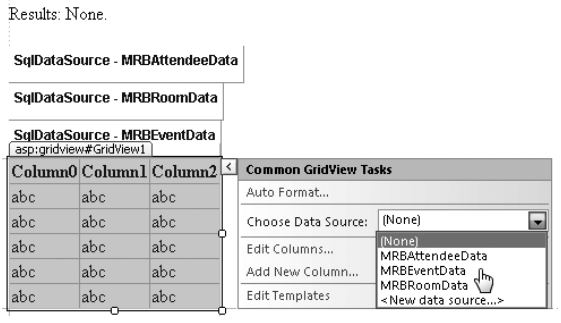

Рис. 37.12. Добавление элемента управления `GridView`

Теперь щелкните на ссылке Refresh Schema (Обновить схему), и список мероприятий будет отображен под формой. Попробуем теперь просмотреть Web-сайт снова и увидим их, как показано на рис. 37.13.

ID	Name	Room	AttendeeList	EventDate
1	My Birthday	4	Iggy Pop (5), Sean Connery (7), Albert Einstein (10), George Clooney (14), Jules Verne (18), Robin Hood (20), and Karli Watson	9/17/2008 12:00:00 AM
2	Dinner	1	Bill Gates (1), Monika Lewinsky (2), and Bruce Lee	8/5/2008 12:00:00 AM
3	Discussion of darkness	6	Vlad the Impaler (4), Myra Hindley (13), and Beelzebub	10/29/2008 12:00:00 AM
4	Christmas with Pals	9	Dr Frank N Furter (11), Bobby Davro (15), John F Kennedy (16), Stephen King (19), and Karli Watson	12/25/2008 12:00:00 AM
5	Escape	17	Monika Lewinsky (2), Stephen King (19), and Spartacus	5/10/2008 12:00:00 AM
6	Planetary Conquest	14	Bill Gates (1), Albert Einstein (10), Dr Frank N Furter (11), Bobby Davro (15), and Darth Vader	6/15/2008 12:00:00 AM
7	Homecoming Celebration	7	William Shakespeare (6), Christopher Columbus (12), Robin Hood (20), and Ulysses	6/22/2008 12:00:00 AM
8	Dalek Reunion Ball	12	Roger Moore (8), George Clooney (14), Bobby Davro (15), and Davros	6/12/2008 12:00:00 AM
9	Romantic meal for two	13	George Clooney (14), and Donna Watson	3/29/2008 12:00:00 AM

Рис. 37.13. Отображение списка мероприятий

Чтобы обеспечить автоматическое обновление этого списка при добавлении новой записи, внесем следующее изменение в код `submitButton_Click()`:

```
if (queryResult == 1)
{
  resultLabel.Text = "Event Added.";
  EventData = null;
  calendar.SelectedDate = GetFreeDate(calendar.SelectedDate.AddDays(1));
  GridView1.DataBind();
}
```

Все привязываемые к данным элементы управления поддерживают этот метод, обычно вызываемый при вызове метода `DataBind()` верхнего уровня (`this`).

Один момент, на который вы, вероятно, обратили внимание на рис. 37.13 — это то, что отображение даты/времени поля `EventDate` выглядит не особенно изящно. Поскольку мы просматриваем только даты, составляющая времени всегда будет

00:00:00, что совершенно излишне. В следующих разделах будет показано, как отобразить эту информацию в более дружественной к пользователю манере в контексте элемента управления `ListView`. Как можно было ожидать, элемент управления `DataGrid` содержит множество свойств, которые можно применить для форматирования отображаемых данных, но мы оставляем это вам для самостоятельных исследований.

Отображение данных по шаблонам

Многие из элементов управления, служащих для отображения данных, позволяют применять форматирующие шаблоны. Шаблоны в том смысле, как они понимаются ASP.NET — это параметризованные разделы HTML, используемые в качестве элементов вывода в определенных элементах управления. Они позволяют точно настроить вывод данных в браузере с тем, чтобы добиться профессионального вида приложений, не затрачивая на это много усилий.

Несколько шаблонов доступны для настройки различных аспектов поведения списков, но один из них важен для элементов управления `Repeater`, `DataList` и `ListView` — это `<ItemTemplate>`, который применяется для отображения каждого элемента данных. Этот шаблон (как и все прочие) объявляется внутри объявления элемента управления. Например:

```
<asp:DataList runat="server" ... >
  <ItemTemplate>
     ...
  </ItemTemplate>
</asp:DataList>
```

Внутри объявления шаблона обычно выводятся фрагменты HTML-кода вместе с параметрами данных, привязанных к элементу управления. Для вывода таких параметров можно использовать специальный синтаксис:

```
<%# выражение %>
```

Здесь вместо *выражение* может быть подставлено выражение, связывающее параметр со свойством страницы или элемента управления, но более вероятно, что оно будет состоять из выражений `Eval()` или `Bind()`. Эти функции могут применяться для вывода данных таблицы, привязанной к элементу управления простой спецификацией столбца. Для `Eval()` применяется следующий синтаксис:

```
<%# Eval("ColumnName") %>
```

Есть также необязательный второй параметр, позволяющий форматировать возвращенные данные, который имеет синтаксис, идентичный выражениям форматирования строк, применяемых повсюду. Например, это может быть использовано для форматирования строк данных в более читабельном виде — как раз то, чего не доставало предыдущему примеру.

Выражение `Bind()` аналогично, но позволяет вставлять данные в атрибуты серверных элементов управления. Например:

```
<asp:Label runat="server" ID="ColumnDisplay" Text='<%# Bind("ColumnName") %>' />
```

Обратите внимание, что поскольку в параметрах `Bind()` используются двойные кавычки, для ограничения значения атрибута применяются одинарные.

В табл. 37.8 представлен список доступных шаблонов вместе с пояснениями касательно их использования.

Таблица 37.8. Доступные шаблоны

Шаблон	Применяется к	Описание
`<ItemTemplate>`	`DataList`, `Repeater`, `ListView`	Используется для элементов списков.
`<HeaderTemplate>`	`DataList`, `DetailsView`, `FormView`, `Repeater`	Используется для вывода перед элементами.
`<FooterTemplate>`	`DataList`, `DetailsView`, `FormView`, `Repeater`	Используется для вывода после элементов.
`<LayoutTemplate>`	`ListView`	Используется для вывода окружающих элементов.
`<SeparatorTemplate>`	`DataList`, `Repeater`	Используется между элементами списка.
`<ItemSeparatorTemplate>`	`ListView`	Используется между элементами в списке.
`<AlternatingItemTemplate>`	`DataList`, `ListView`	Используется для замены элементов; может помогать видимости.
`<SelectedItemTemplate>`	`DataList`, `ListView`	Используется для выбранных элементов списка.
`<EditItemTemplate>`	`DataList`, `FormView`, `ListView`	Используется для редактируемых элементов.
`<InsertItemTemplate>`	`FormView`, `ListView`	Используется для вставляемых элементов.
`<EmptyDataTemplate>`	`GridView`, `DetailsView`, `FormView`, `ListView`	Используется для отображения пустых элементов, например, когда никакие записи недоступны в `GridView`.
`<PagerTemplate>`	`GridView`, `DetailsView`, `FormView`	Используется для форматирования нумерации страниц.
`<GroupTemplate>`	`ListView`	Используется для определения выходных окружающих групп элементов.
`<GroupSeparatorTemplate>`	`ListView`	Используется между группами элементов.
`<EmptyItemTemplate>`	`ListView`	Если используются группы элементов, то этот шаблон применяется для предоставления выходных данных для пустых элементов в группе. Этот шаблон применяется, если в группе нет достаточного количества элементов для заполнения группы.

Проще всего понять, как их использовать, с помощью примера.

Использование шаблонов

Мы расширим таблицу в верхней части страницы `Default.aspx` приложения `PCSWebApp3` так, чтобы включить `ListView`, отображающий все мероприятия, хранящиеся в базе данных. Сделаем их выбираемыми, чтобы подробности о каждом мероприятии отображались в элементе управления `FormView` в результате щелчка на его названии.

Для начала нам понадобится создать новые источники данных для привязанных к данным элементов управления. Хорошим тоном (и мы настоятельно рекомендуем его придерживаться) является наличие отдельного источника данных для каждого из таких элементов управления.

Элемент SqlDataSource, необходимый для ListView — MRBEventData2 — в основном похож на MRBEventData за исключением того, что он должен возвращать только данные столбцов Name и ID. Необходимый код представлен ниже:

```
<asp:SqlDataSource ID="MRBEventData2" runat="server"
  SelectCommand="SELECT [ID], [Name] FROM [Events]"
  ConnectionString="<%$ ConnectionStrings:MRBConnectionString %>">
</asp:SqlDataSource>
```

Источник данных MRBEventDetailData, используемый для элемента управления FormView, более сложен, хотя его можно построить достаточно легко с помощью мастера конфигурации источника данных. Этот источник данных использует выбранный элемент элемента управления ListView, который мы назовем EventList, чтобы получать данные только этого выбранного элемента. Это делается за счет применения параметра в SQL-запросе:

```
<asp:SqlDataSource ID="MRBEventDetailData" runat="server"
  SelectCommand="SELECT dbo.Events.Name, dbo.Rooms.Room, dbo.Events.AttendeeList,
                 dbo.Events.EventDate FROM dbo.Events INNER JOIN dbo.Rooms
                 ON dbo.Events.ID = dbo.Rooms.ID WHERE dbo.Events.ID = @ID"
  ConnectionString="<%$ ConnectionStrings:MRBConnectionString %>">
  <SelectParameters>
     <asp:ControlParameter Name="ID" DefaultValue="-1" ControlID="EventList"
       PropertyName="SelectedValue" />
  </SelectParameters>
</asp:SqlDataSource>
```

Здесь вместо параметра @ID в запросе подставляется значение ID элемента, выбранного в списке. Элемент ControlParameter получает это значение из свойства SelectedValue списка EventList или же использует значение -1, если ни одного элемента не выбрано. На первый взгляд такой синтаксис кажется несколько странным, но он обеспечивает высокую степень гибкости, и если вы воспользуетесь мастером, у вас не возникнет с ним никаких проблем.

Далее нам нужно добавить элементы управления ListView и FormList. Необходимые изменения кода Default.aspx в проекте PCSWebApp3 соответствующим образом выделены:

```
<tr>
  <td align="center" colspan="3">
     <asp:ValidationSummary ID="validationSummary" runat="server"
       HeaderText="Before submitting your request:" />
  </td>
</tr>
<tr>
  <td align="left" colspan="3" style="width: 40%;">
  <table cellspacing="4" style="width: 100%;">
  <tr>
     <td colspan="2" style="text-align: center;">
        <h2>Event details</h2>
     </td>
  </tr>
  <tr>
  <td style="width: 40%; background-color: #ccffcc;" valign="top">
     <asp:ListView ID="EventList" runat="server"
       DataSourceID="MRBEventData2" DataKeyField="ID"
```

```
                OnSelectedIndexChanged="EventList_SelectedIndexChanged">
                <LayoutTemplate>
                   <ul>
                     <asp:Placeholder ID="itemPlaceholder"
                       runat="server" />
                   <ul>
                </LayoutTemplate>
                <ItemTemplate>
                   <li>
                     <asp:LinkButton Text='<%# Bind("Name") %>'
                       runat="server" ID="NameLink" CommandName="Select"
                    CommandArgument='<%# Bind("ID") %>'
                    CausesValidation="false" />
                   </li>
                </ItemTemplate>
                <SelectedItemTemplate>
                   <li>
                      <b><%# Eval("Name") %></b>
                   </li>
                </SelectedItemTemplate>
               </asp:ListView>
             </td>
             <td valign="top">
               <asp:FormView ID="FormView1" runat="server"
                  DataSourceID="MRBEventDetailData">
                  <ItemTemplate>
                       <h3><%# Eval("Name") %></h3>
                       <b>Date:</b>
                       <%# Eval("EventDate", "{0:D}") %>
                       <br />
                       <b>Room:</b>
                       <%# Eval("Room") %>
                       <br />
                       <b>Attendees:</b>
                       <%# Eval("AttendeeList") %>
                  </ItemTemplate>
             </asp:FormView>
           </td>
         </tr>
       </table>
    </td>
  </tr>
  </table>
```

Здесь мы добавили новую строку таблицы, которая сама содержит таблицу с элементом управления `ListView` в одном столбце и `FormView` — в другом.

`ListView` использует `<LayoutTemplate>` для вывода списка с буллитами, а также `<ItemTemplate>` и `<SelectedItemTemplate>` для отображения подробностей о мероприятии. В `<LayoutTemplate>` определен контейнерный элемент для элементов посредством элемента управления `PlaceHolder`, имеющего атрибут `ID="itemPlaceHolder"`. Чтобы облегчить выбор, выполняется команда `Select` по ссылке-названию мероприятия, отображенного в `<ItemTemplate>`, что автоматически изменяет выбор. Также возбуждается событие `OnSelectedIndexChanged`, когда команда `Select` изменяет выбор, дабы гарантировать, что отображаемый список покажет выбранную строку в другом стиле. Код обработчика события представлен ниже:

```
protected void EventList_SelectedIndexChanged(object sender, EventArgs e)
{
  EventList.DataBind();
}
```

Также потребуется обеспечить добавление новых мероприятий в список:

```
if (queryResult == 1)
{
  resultLabel.Text = "Event Added.";
  EventData = null;
  calendar.SelectedDate =
       GetFreeDate(calendar.SelectedDate.AddDays(1));
  GridView1.DataBind();
  EventList.DataBind();
}
```

Теперь подробности о запланированных мероприятиях доступны в таблице, как показано на рис. 37.14.

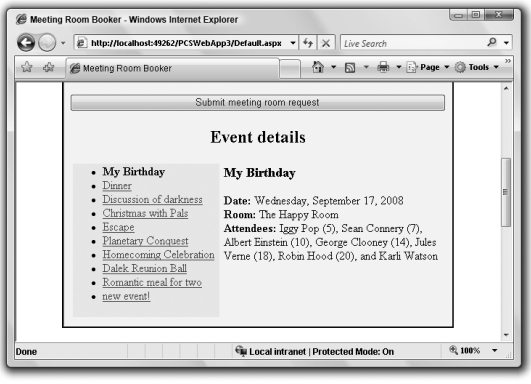

Рис. 37.14. Отображение подробной информации о мероприятии

С шаблонами и привязанными к данным элементами управления можно делать множество других вещей, для описания которых понадобилась бы целая книга. Однако изложенного должно быть достаточно для того, чтобы вы начали самостоятельно экспериментировать.

Конфигурация приложения

Одной из вещей, которые мы постоянно упоминали на протяжении этой главы, было существование некоего концептуального приложения, состоящего их Web-страниц и конфигурационных настроек. Это важная концепция, которую необходимо понять, особенно, если есть необходимость в индивидуальной настройке вашего Web-сайта для множества параллельно работающих пользователей.

Приведем несколько заметок по терминологии и времени жизни приложения. *Приложение* определяется как все файлы проекта и конфигурируется с помощью файла `Web.config`. Объект `Application` создается, когда приложение запускается первый раз, что происходит при появлении первого HTTP-запроса. Также в этот момент возбуждается событие `Application_Start` и создается пул экземпляров `HttpApplication`. Каждый входящий запрос получает один из этих экземпляров для выполнения обработки. Обратите внимание, что это значит, что объекты `HttpApplication` не обязаны работать с параллельным доступом, в отличие от глобального объекта `Application`. Когда все экземпляры `HttpApplication` завершают свою работу, генерируется событие `Application_End` и приложение завершается, уничтожая объект `Application`.

Обработчики упомянутых ранее событий (вместе со всеми обработчиками, рассмотренным в этой главе) могут быть определены в файле `Global.asax`, который легко добавить к любому проекту Web-сайта. Сгенерированный файл содержит пустые места, подлежащие заполнению разработчиком. Например:

```
protected void Application_Start(Object sender, EventArgs e)
{
}
```

Когда индивидуальный пользователь обращается к Web-приложению, запускается *сеанс* (session). Как и все приложение, это включает создание специфичного для пользователя объекта `Session` вместе с возбуждением события `Session_Start`. Внутри сеанса отдельные *запросы* инициируют события `Application_BeginRequest` и `Application_EndRequest`. Это может происходить несколько раз в рамках сеанса, по мере обращения к разным ресурсам внутри приложения. Индивидуальные сеансы могут быть прерваны вручную или по таймауту, если не поступает новых запросов. Прерывание сеанса возбуждает событие `Session_End` и уничтожает объект `Session`.

На фоне этого процесса вы можете выполнить некоторые операции со своим приложением. Если все его экземпляры используют единственный ресурсоемкий объект, то задачу создания его экземпляра можно вынести на уровень приложения. Это увеличит производительность и снизит затраты памяти при большом количестве пользователей, поскольку для обработки большинства запросов создавать этот экземпляр не придется.

Другая технология, которую можно применить — сохранять между запросами информацию уровня сеанса для индивидуальных пользователей. Это может включать специфичную для пользователя информацию, извлеченную из хранилища данных при первом его подключении (в обработчике события `Session_Start()`), и оставлять ее доступной до тех пор, пока сеанс не будет прекращен (или вследствие истечения времени, или по запросу пользователя.

Описание всех этих приемов выходят за пределы тем этой книги, но надеемся, общее представление вы получили.

И, наконец, нам нужно рассмотреть файлы `Web.config`. Обычно Web-сайт будет иметь один из них в своем корневом каталоге (хотя он и не создается для вас по умолчанию) и несколько дополнительных в подкаталогах — для конфигурирования специфичных для каталогов установок (вроде настроек безопасности). Web-сайт `PCSWebApp3`, который мы разработали в этой главе, принял автоматически сгенерированный файл `Web.config`, когда мы добавили хранимую строку соединения с базой данных, которую можно увидеть в файле:

```
<connectionStrings>
  <add name="MRBConnectionString" connectionString="Data Source=.\SQLEXPRESS;
     AttachDbFilename=|DataDirectory|\MeetingRoomBooker.mdf;
     Integrated Security=True;User Instance=True"
     providerName="System.Data.SqlClient" />
</connectionStrings>
```

Если запустить проект в режиме отладки, то в файле `Web.config` появятся некоторые дополнительные настройки.

Файлы `Web.config` можно редактировать вручную, но Web-сайт (и его конфигурационные файлы) также можно настроить с помощью инструментального средства. Этот инструмент доступен через пункт меню Website⇨ASP.NET Configuration (Web-сайт⇨Конфигурация ASP.NET) в Visual Studio. Экрана этого инструмента показан на рис. 37.15.

Как видите, этот инструмент позволяет конфигурировать множество настроек, включая безопасность. Намного больше о нем вы узнаете в следующей главе.

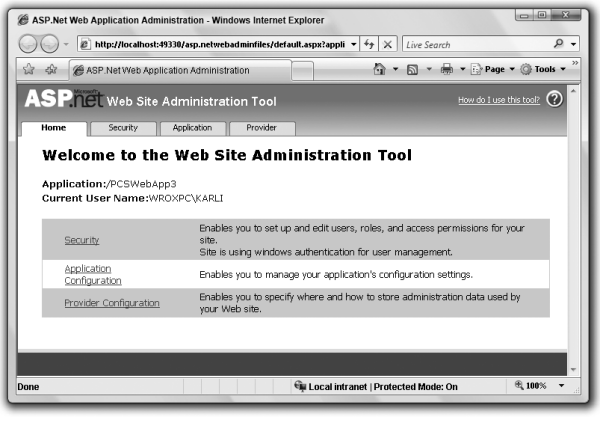

Рис. 37.15. Инструментальное средство администрирования Web-сайта

Резюме

В этой главе был представлен обзор создания Web-приложений ASP.NET. Вы увидели, как можно использовать C# в комбинации с Web-серверными элементами управления для обеспечения действительно полнофункциональной среды разработки. Мы разработали пример приложения заказа мероприятий, чтобы проиллюстрировать многие из доступных приемов — такие как многообразие серверных элементов управления и связывание данных с ADO.NET.

В частности, рассматривались перечисленные ниже вопросы.

- Представление ASP.NET и как эта технология вообще стыкуется с разработкой .NET.
- Как работает базовый синтаксис ASP.NET, как осуществляется управление состоянием, а также как интегрировать код C# со страницами ASP.NET.
- Как создать Web-приложение ASP.NET в среде Visual Studio, а также, какие доступны варианты для хостинга и тестирования Web-сайтов.
- Перечень элементов управления Web, доступных разработчикам ASP.NET, и как они работают вместе для обеспечения доставки динамического и/или управляемого данными содержимого.
- Как работать с обработчиками событий для обнаружения и обработки пользовательского взаимодействия с элементами управления, и как настроить их через события страниц и отображения.
- Как привязать данные к элементам управления Web, и как форматировать отображаемые данные, используя шаблоны и выражения связывания данных.
- Как собирать все перечисленное вместе для построения приложения заказа комнат для совещаний.

Обладая всей этой информацией, вы теперь готовы собирать собственные мощные Web-приложения. Однако до сих пор мы лишь слегка коснулись тех возможностей, которые предоставляет разработка ASP.NET. Поэтому прежде чем отложить эту книгу и погрузиться в собственно разработку для Web, проявите терпение и прочитайте немного больше. В главе 38 вы сможете расширить свои знания относительно ASP.NET, ознакомившись с некоторыми наиболее важными темами, относящимися к Web, включая ведущие страницы, настройку внешнего вида и персонализацию. Результаты того стоят!