



主要特色

- 以大量的范例演示ASP.NET 4.0技术的应用
- 知识体系完整,涉及.NET 4.0 大量的新技术

超值光盘

- 每章的范例代码
- 12个使用ASP.NET 4.0和C#开发的项目案例
- 69堂多媒体技术讲座,全面阐述ASP.NET 4.0动态网站开发技术
- 超过3000页的技术文档,包括HTML、ASP.NET Ajax、C#网络编程技术、C#数据库高级技术、Access、SQL Server 2000、SQL Server 2005数据库使用技术等内容

ASP.NET 4.0

从入门到精通



36小时教学视频
共69课全程语音讲解

张正礼 王坚宁 编著



清华大学出版社



ASP.NET 4.0 从入门到精通

本 书12个完整项目，
代码可以直接重用：

网上个人博客
网上音乐商店
图书管理系统
网上校友录
考勤管理系统
新闻发布系统
绩效管理系统
博客管理系统
医院管理系统
仓库管理系统
学生宿舍管理系统
机票预订系统

超 值光盘

- 每章的范例代码
- 12个使用ASP.NET 4.0和C#开发的项目案例
- 69堂多媒体技术讲座，全面阐述ASP.NET 4.0动态网站开发技术
- 超过3000页的技术文档，包括HTML、ASP.NET Ajax、C#网络编程技术、C#数据库高级技术、Access、SQL Server 2000、SQL Server 2005数据库使用技术等内容

上架指导：计算机/程序开发
封面设计：王 翔





ASP.NET 4.0

从入门到精通



36小时教学视频
共69课全程语音讲解

张正礼 王坚宁 编著

清华大学出版社
北 京

目 录

第 1 章 ASP.NET 4.0 开发入门1

1.1 网页基础知识1

1.1.1 网页基础理论.....1

1.1.2 静态页面.....2

1.1.3 动态页面.....3

1.1.4 CGI 接口.....4

1.1.5 脚本语言.....4

1.2 ASP.NET 4.0 框架5

1.2.1 .NET 框架的发展历程.....5

1.2.2 .NET 4.0 语言6

1.2.3 公共语言运行时.....6

1.2.4 动态语言运行时.....7

1.2.5 .NET 类库.....9

1.3 ASP.NET 应用程序9

1.3.1 ASP.NET 页面与服务器交互10

1.3.2 ASP.NET Web 窗体.....10

1.3.3 ASP.NET 4.0 的新特性10

1.4 Visual Studio 2010 开发环境13

1.4.1 安装 Visual Studio 2010.....13

1.4.2 创建 Web 项目15

1.4.3 Web 项目管理17

1.4.4 Visual Studio 2010 的新特性.....20

1.5 配置 Web 服务器.....22

1.6 配置 ASP.NET 4.0 应用程序26

1.7 上机练习29

1.8 上机题30

第 2 章 C# 语言基础31

2.1 C# 语言概述31

2.1.1 第一个 C# 程序.....32

2.1.2 C# 代码结构.....34

2.2 基本语法36

2.2.1 数据类型.....36



ASP.NET 4.0 开发入门 1.wmv/32 分钟

ASP.NET 4.0 开发入门 2.wmv/11 分钟



C# 语言基础 1.wmv/30 分钟

C# 语言基础 2.wmv/55 分钟

C# 语言基础 3.wmv/17 分钟

C# 语言基础 4.wmv/44 分钟

2.2.2	变量和常量	44
2.2.3	运算符	46
2.2.4	转义字符	52
2.2.5	C# 中的控制语句	52
2.3	面向对象编程	62
2.3.1	类	63
2.3.2	属性、方法和事件	64
2.3.3	构造函数	65
2.3.4	继承和多态	66
2.3.5	委托	69
2.3.6	事件	71
2.4	泛型	73
2.4.1	使用系统的泛型类	73
2.4.2	创建泛型	74
2.5	C# 4.0 的新特性	76
2.5.1	大整数类型 BigInteger	76
2.5.2	动态数据类型 dynamic	78
2.5.3	命名参数和可选参数	79
2.6	上机题	81

第 3 章 ASP.NET 4.0 常用内置对象 82

3.1	Page 类	82
3.1.1	页面的生命周期	82
3.1.2	Page 类的主要属性、方法和事件	83
3.1.3	应用 Page 类	85
3.2	Request 对象	87
3.2.1	Request 对象的属性和方法	87
3.2.2	应用 Request 对象	88
3.3	Response 对象	89
3.3.1	Response 对象的属性	90
3.3.2	Response 对象的方法	90
3.3.3	应用 Response 对象	91
3.4	Server 对象	92
3.4.1	Server 对象的属性和方法	93
3.4.2	应用 Server 对象	94
3.5	Cookie 对象	95
3.5.1	Cookie 简介	95
3.5.2	Cookie 对象的属性和方法	96
3.5.3	应用 Cookie 对象	97
3.6	Session 对象	99



ASP.NET 4.0 常用内置对象 1.wmv/32 分钟
 ASP.NET 4.0 常用内置对象 2.wmv/34 分钟
 ASP.NET 4.0 常用内置对象 3.wmv/33 分钟

3.6.1	Session 简介	99
3.6.2	对 Session 的跟踪	100
3.6.3	Session 对象的属性和方法	100
3.6.4	Session 对象的储存	101
3.6.5	应用 Session 对象	103
3.7	Application 对象	105
3.7.1	Application 对象的属性和方法	106
3.7.2	应用 Application 对象	107
3.8	ViewState 对象	109
3.8.1	ViewState 中的键值对	109
3.8.2	ViewState 的安全机制	109
3.8.3	存储自定义对象	110
3.8.4	应用 ViewState 对象	111
3.9	上机题	113
第 4 章	ASP.NET 4.0 服务器控件	114
4.1	服务器控件类	114
4.1.1	服务器控件基本属性	115
4.1.2	服务器控件的事件	118
4.2	文本服务器控件	120
4.2.1	标签 (Label) 控件	120
4.2.2	静态文本 (Literal) 控件	120
4.2.3	文本框 (TextBox) 控件	120
4.2.4	超链接文本 (HyperLink) 控件	121
4.3	按钮服务器控件	123
4.3.1	普通按钮 (Button) 控件	123
4.3.2	超链接按钮 (LinkButton) 控件	123
4.3.3	图片按钮 (ImageButton) 控件	123
4.4	图像服务器控件	125
4.4.1	图像 (Image) 控件	125
4.4.2	图像地图 (ImageMap) 控件	125
4.5	选择服务器控件	127
4.5.1	复选框 (CheckBox) 控件	128
4.5.2	复选框列表 (CheckBoxList) 控件	128
4.5.3	单选按钮 (RadioButton) 控件	130
4.5.4	单选按钮列表 (RadioButtonList) 控件	131
4.6	列表服务器控件	133
4.6.1	列表框 (ListBox) 控件	133
4.6.2	下拉列表框 (DropDownList) 控件	135
4.6.3	项目列表 (BulletedList) 控件	137



ASP.NET 4.0 服务器控件 1.wmv/50 分钟
 ASP.NET 4.0 服务器控件 2.wmv/34 分钟
 ASP.NET 4.0 服务器控件 3.wmv/45 分钟

第 1 章 ASP.NET 4.0 开发入门

学习目标

ASP.NET (Active Server Page.NET) 是微软公司推出的基于 .NET 4.0 框架的新一代网络编程语言,也是目前最新的 Web 技术之一。作为之前各个 ASP.NET 版本的集大成者,ASP.NET 4.0 开创了公共语言运行库和动态语言运行库相结合的编程框架,可用于在服务器上生成功能强大的 Web 应用程序。本章将介绍 ASP.NET 4.0 的相关基础知识以及如何创建其开发环境,帮助读者对这一强大的 Web 编程武器有一个初步的基本认识。

本章重点

- 熟悉 Visual Studio 2010 开发环境
- 掌握 IIS 服务器的安装和配置
- 了解 Web.config 文件的结构
- 理解 ASP.NET 4.0 的基本框架

1.1 网页基础知识

互联网的出现是在上个世纪 60 年代末,早期的互联网用户大多局限于政府机关和军事领域。但是随着网络信息技术的不断提高,互联网逐渐向普通民众敞开了大门,逐渐在全球范围内实现了信息共享。作为互联网的组成部分——Web 网站在这一过程中获得了同步的发展,而网页开发技术也逐渐走向了成熟。

1.1.1 网页基础理论

在人们通过互联网浏览网页时,用户会自动与网页服务器建立连接。用户提交信息资源的过程称为向服务器“发出请求”。通过服务器解释信息资源来定位对应的页面,并传送回代码来创建页面,这个过程称为对浏览器的响应。浏览器接受来自于网页服务器的代码,并将它编译成可视页面。在这样的交互过程中,浏览器称为“客户机”或者“客户端”,整个交互的过程则称为“客户机/服务器”的通信过程。

“客户机/服务器”这一术语通过概括任务的分布来描述网页的工作方式。服务器(Web 服务器)存储数据、解释数据、分布数据。客户机(浏览器)访问服务器以得到数据。为了更详细地理解这一交互过程,需要了解客户机和服务器如何使用 HTTP 协议通过 Internet 进行交互。

HTTP 协议又叫做超文本传输协议,是一个客户机和服务器端请求和应答的标准。我们在浏览

网页时，浏览器通过 HTTP 协议与服务器交换信息。

HTTP 协议具有以下的特点。

(1) HTTP 按客户机/服务器模式工作。HTTP 支持客户与服务器的通信，相互传输数据。HTTP 定义的事务由以下 4 步组成。

- 客户与服务器建立连接。
- 客户向服务器提出请求。
- 如果请求被接受，则服务器送回响应，在响应中包括状态码和所需的文件。
- 客户与服务器断开连接。

(2) HTTP 是无状态的。也就是说，浏览器和服务器每进行一次 HTTP 操作，就建立一次连接，但任务结束就中断连接。

(3) HTTP 使用元信息作为头标。HTTP 对所有的事务都加了头标。它使服务器能够提供正在传送数据的有关信息。比如传送对象是哪种类型，是用哪种语言编写的等等。

(4) HTTP 支持两种请求和响应的格式。HTTP 由不同的两部分组成，一种是从浏览器发往服务器的请求，另一种是服务器对客户的响应。HTTP 支持两种请求和响应，一种是简单请求和响应，另一种是完全请求和响应。

(5) HTTP 是基于文本的简单协议。每个 Web 主机都有一个服务器进程来监听 TCP 端口 80，以便同前来建立连接的客户取得联系。连接建立后，客户发送一个请求，服务器返回一个响应，然后就释放连接。除建立和释放连接外，HTTP 事务处理的主要内容是客户机的请求与服务器端的响应，HTTP 常用的请求方法如表 1-1 所示。

表 1-1 HTTP 常用的请求方法

方法名称	描述
GET	请求读取一个 Web 页面
HEAD	请求读取一个 Web 页面的头标
PUT	请求存储一个 Web 页面
POST	附加到命名资源中
DELETE	删除 Web 页面
LINK	连接两个已有资源
UNLINK	取消两个资源之间的已有连接

1.1.2 静态页面

早期网站发布的是静态网页，主要由 HTML 语言组成，没有其他的可以执行的程序代码。静态页面一经制成，内容就不会再改变，不管何时何人访问，显示的都是一样的内容，如果要修改有关内容，就必须修改源代码，然后重新上传到服务器上。静态页面虽然包含文字和图片，但这些内容却需要在服务器端以手工的方式来修改，因此很难把他们描述为 Web 程序。下面是一个简单的 HTML 语言组成的静态网页代码：

```
<html>
<head>
    <title>静态网页</title>
</head>
<body>
    <h1>静态网页</h1>
    <p>由 HTML 语言编写</p>
</body>
</html>
```

代码说明：程序包含一个标题和一句文字。其中标题包含在标记<h1>和</h1>之间，文字包含在标记<p>和</p>之间。图 1-1 显示了该静态网页文件被浏览器解析后的效果。

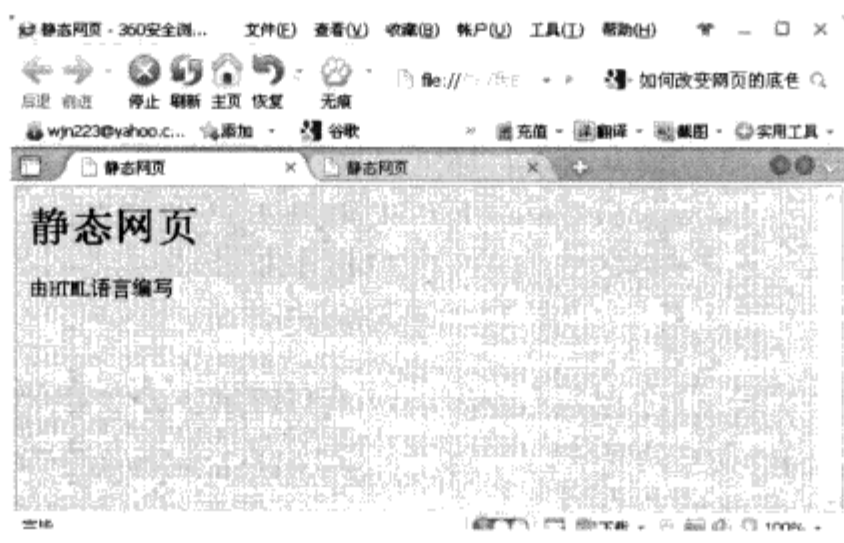


图 1-1 静态网页

HTML 是互联网的描述语言，基本的 HTML 包含由 HTML 标记格式化的文本和图像内容。文本是 HTML 要显示的内容，标记则告诉浏览器如何显示这些内容，它定义了不同层次的标题，段落、链接、斜体格式化和横向线等。HTML 文件的后缀可以是.htm，也可以是.html。

1.1.3 动态页面

动态页面不仅含有 HTML 标记，而且含有可以执行的程序代码，动态页面能够根据不同的输入和请求动态生成返回的页面，例如常见的 BBS、留言板、聊天室等就是用动态网页来实现的。动态网页的使用非常灵活，功能强大。

一直到 HTML 2.0 版本，引入 HTML 表单，这时才有了真正意义的包含动态页面的 Web 程序：在一个 HTML 表单中，所有的控制都放置在<form>和</form>中。当读者在客户端单击“提交”按钮后，网页上的所有内容就以字符串的形式发送到服务器端，服务器端的处理程序根据事先设置好的标准来响应客户的请求。下面所示的就是一个由 HTML 表单构成的动态页面代码。

```
<html>
<head>
    <title>动态网页</title>
</head>
<body>
    <form>
        <h3>请选择你的爱好？</h3>
        <p>请作出选择：</p>
        <input type="checkbox" />阅读<br/>
        <input type="checkbox" />音乐<br/>
    </form>
</body>
</html>
```

```
<input type="checkbox" />体育<br/>
<input type="checkbox" />旅游<br/>
<input type="submit" value="提交">
</form>
</body>
</html>
```

代码说明：该程序由 HTML 表单的组成，包括一个标题、四个复选框和提交按钮，这些内容和标记均被包含在表单标记之间。该网页运行效果如图 1-2 所示。

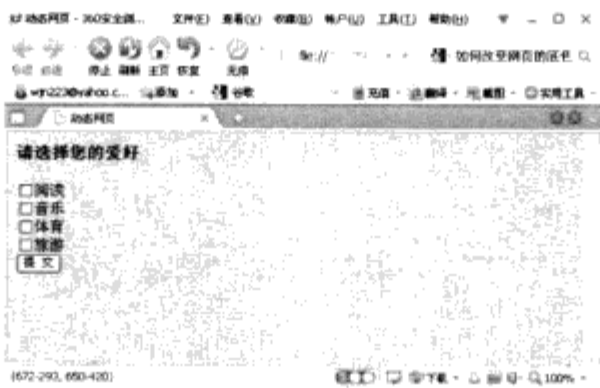


图 1-2 动态页面

今天，尽管动态 ASP.NET 页面已经比较流行，但 HTML 表单仍然是这些页面的基本组成元素，所不同的是构成 ASP.NET 页面的 HTML 表单控件运行在服务器端。所以读者必须掌握最基本的 HTML 表单以便能够更好地使用 ASP.NET 平台进行程序开发。

1.1.4 CGI 接口

CGI 是 Common Gateway Interface 的缩写，代表服务器端的一种通用（网关）接口。CGI 开启了动态网页的先河。它的运行原理是每当服务器接到客户更新数据的要求以后，利用这个接口去启动外部应用程序（利用 C，C++，Perl，Java 或其他语言编写）来完成各类计算、处理或访问数据库的工作，处理完后将结果返回 Web 服务器，再返回浏览器。后来又出现了技术有所改进的 ISAPI 和 NSAPI 技术，提高了动态网页的运行效率，但由于 CGI 在多用户访问时会占用很多的系统资源，并且执行起来速度相对比较慢，因此目前已经很少使用了。

1.1.5 脚本语言

在 CGI 技术之后出现了很多优秀脚本语言，如 ASP、JSP、PHP 等。脚本语言简化 Web 程序的开发，一时间成为 Web 开发人员的最爱。但脚本语言使用起来也并不是那么简单，首先其代码组织混乱，和 HTML 标记杂乱堆砌在一起，开发维护都非常不方便，以至当 ASP.NET 的代码隐藏模式出现后，使用这些脚本语言的 Web 程序开发人员都有一种耳目一新的感觉；另外脚本语言的编程思想不符合当前流行的面向对象编程思想。因此脚本语言必将会被其它更高级语言（ASP.NET、Java 等）所代替。



提示

实际上 Web 页面的后缀定义，都是约定俗成的，文件命名并没有强制的规定，如果把静态网页的后缀修改为 .asp 或 .jsp 也不会出现错误。不过这里还是建议读者遵守默认的命名规则。

1.2 ASP.NET 4.0 框架

2002 年, 微软公司宣布了自己的 .NET 框架。 .NET 框架的基本思想是: 把原有的重点从连接到互联网的单一网站或设备转移到计算机、设备和服务群组上, 而将互联网本身作为新一代操作系统的基础。这样, 用户能够控制信息的传递方式、时间和内容, 从而得到更多的服务。

经过了 8 年的发展, .NET 技术已经深入互联网。基于 HTML 的通信技术通过可编程的基于 XML 的信息得到增强, Web 服务的概念被引入, 普通用户成为最大的受益者, 实现了通过手写、语音以及图像增强技术与个人数据进行交互。如今在经历 .NET 3.5 的短暂过度之后, .NET 4.0 以正式版本的形式出现开发人员的视野中, .NET 4.0 框架代表着一系列的技术, 这些技术可以用来帮助我们建立丰富的应用程序。

1.2.1 .NET 框架的发展历程

.NET 框架的发展, 从最新的 1.0 版本到目前的 4.0 版本, 有整整 8 个年头了。平均每两年更新一次。下面就让我们来回顾一下这一段不平凡的历程。

2002 年, .NET 框架 1.0 版本 (完整版本号是 1.0.3705) 发布, 它是 .NET 框架的最初版本, 同时也是 Visual Studio .NET 2002 的一部分。它给软件开发带来了 many 激动人心的特性。

- 统一的类型系统, 基础类库, 垃圾回收和多语言支持。
- ADO.NET 1.0 开启了微软全新的数据访问技术。
- ASP.NET 1.0 变革了 ASP, 提供一种全新的方式来开发 Web 应用程序。
- Windows Forms 1.0 把微软开发 Windows 桌面系统的界面统一在一起。

2003 年, .NET 框架 1.1 版本 (完整版本号是 1.1.4322) 发布, 它是 .NET 框架的首个主要升级版本, 同时也是 Visual Studio .NET 2003 的一部分, 它也是首个 Windows Server 2003 内置的 .NET 框架版本。1.1 版本发布后, 程序界开始追捧这个平台。

2005 年, .NET 框架 2.0 版本 (完整版本号是 2.0.50727.42) 发布, 这次的变化是革命性的, 它同时是 Visual Studio .NET 2005 的一部分。2.0 版本带来的新变化如下。

- ADO.NET 2.0 加强了很多功能, 提升了性能, 能够更好地进行数据层的开发。
- Web 服务的性能得到提升, 并且在安全性等方面都得以提高。
- 泛型和内置泛型集合的支持, 和其他基础类库的扩展, 可以让内部的公共类库开发更加简化。
- 全新事物机制 (System.Transactions) 的引入, 让整个系统的事务处理更加方便。

2006 年, .NET 3.0 发布, 这个版本比较特殊, 它需要 .NET 2.0 安装后才能运行, 因此软件界普遍不把 .NET 3.0 当着正式的 .NET 的版本。它提供了如下组件。

- WindowsCommunicationFoundation (WCF), 支持面向服务的应用程序。
- WindowsWorkflowFoundation (WF), 支持基于工作流的应用程序。
- WindowsPresentationFoundation (WPF), 适用于不同用户界面的统一方法。
- WindowsCardSpace (WCS), 是一致的数字标识用户控件。

.NET3.0 提供的这些组件为开发企业应用程序提供了一致的基础框架,这样业务开发人员就只需要关注于业务问题的解决。

2007 年 11 月 19 日, .NET 3.5 发布, 它同时是 Visual Studio .NET 2008 的一部分。 .NET 3.5 带来的新特性如下:

- ASP.NET AJAX, AJAX 扩展包内置到 .NET 3.5 里面。
- 语言改进和 LINQ, 具体改进包括自动属性、对象初始化器、集合初始化器、扩展方法、Lambda 表达式、查询句法、匿名类型。
- LINQtoSQL 实现的数据访问改进。
- 在 ASP.NET 3.5 扩展版本中推出了 MVC 编程框架。

2010 年, .NET 4.0 发布, 它同时是 Visual Studio .NET 2010 的一部分。本书就是基于目前这一最新版本进行 ASP.NET 4.0 网站开发的介绍。

通过 .NET 框架的发展历程可以看出, 微软的 .NET 战略就是要进一步解放程序员, 让项目开发变得更加高效率, 而且更加简单容易操作。沿着这个方向发展, 可以预见在未来的一段时间内, 程序开发将不再是专业程序人员的事情, 而真正懂业务逻辑的专业人才将会成为项目开发的主力。

1.2.2 .NET 4.0 语言

ASP.NET 4.0 框架支持多种语言, 包括 C#、VB、J#、C++ 等, 而本书使用的语言主要是 C#。

C# 是一个是在 .NET 1.0 中开始出现的一种新语言, 在语法上, 它与 Java 和 C++ 比较相似。实际上 C# 是微软整合了 Java 和 C++ 的优点而开发出来的一种语言, 是微软对抗 Java 平台的一个王牌。

.NET 框架还支持其它语言, 比如 J# 等, 甚至还可以使用第三方提供的语言, 比如 Eiffel 或 COBOL 的 .NET 版本。这样就增加了程序员开发应用程序时可供选择的范围。尽管如此, 在开发 ASP.NET 应用程序时 VB 和 C# 还是首选。

其实, 在被执行之前, 所有 .NET 语言都会被编译成为一种低级别的语言, 这种语言就是中间语言 (Intermediate Language, IL)。CLR 之所以支持很多种语言, 就是因为这些语言在运行之前被编译成了中间语言。正是因为所有的 .NET 语言都是建立在中间语言之上, 所以 VB 和 C# 具有相同的特性和行为。因此一个利用 C# 编写的 Web 页面可以使用一个使用 VB 编写的组件, 同样使用 VB 编写的 Web 页面也可以使用 C# 编写的组件。

.NET 框架提供了一个公共语言规范 (Common Language Specification, CLS) 以保证这些语言之间的兼容性。只要遵循 CLS, 任何利用某一种 .NET 语言编写的组件都可以被其他语言所引用。CLS 的一个重要部分是公共类型系统 (Common Type System, CTS), CTS 定义了诸如数字、字符串和数组等数据类型的规则, 这样它们就能为所有的 .NET 语言所共享。CLS 还定义了诸如类、方法、实践等对象成分。然而事实上, 基于 .NET 进行程序开发的程序员却没有必要考虑 CLS 是如何工作的, 因为这一切都有 .NET 平台自动来完成。其实 CLR 只执行中间语言代码, 然后把它们进一步编译成为机器语言代码以使当前平台能执行。

1.2.3 公共语言运行时

公共语言运行时 (Common Language Runtime, 简称 CLR) 是用 .NET 语言编写的代码公共运

行环境,是.NET 框架的基础,也是实现.NET 跨平台、跨语言、代码安全等核心特性的关键。它是一个在执行时管理代码的代理,以跨语言集成、自描述组件、简单配制和版本化及集成安全服务为特点,提供核心服务(如内存管理、线程管理和远程处理)。

公共语言运行时管理了.NET 中的代码,这些代码称为受托管代码。它们包含了有关代码的信息,例如代码中定义的类、方法和变量。受托管代码中所包含的信息称为元数据。公共语言运行时使用元数据来安全地执行代码程序。除了安全地执行程序以外,受托管代码的目的在于 CLR 服务。这些服务包括查找和加载类以及与现有的 DLL (Dynamic Link Library, 动态链接库) 代码和组件对象之间的相互操作。

公共语言运行时遵循公共语言架构的标准,可以使 C++、C#、Visual Basic 以及 JScript 等多种语言能够深度集成。

1.2.4 动态语言运行时

.NET 4.0 框架中最令人激动的新特性是动态语言运行时 (Dynamic Language Runtime, 简称 DLR)。就像公共语言运行时 (CLR) 为静态型语言如 C# 和 VB.NET 提供了通用平台一样,动态语言运行时 (DLR) 为像 JavaScript、Ruby、Python 甚至 COM 组件等动态语言提供了通用平台。这代表.NET4.0 框架在互操作性方面向前迈进了一大步。

动态语言运行时 (DLR) 是一种运行时环境,它将一组适用于动态语言的服务添加到公共语言运行时。借助于动态语言运行时,可以更轻松地开发要在 .NET 4.0 框架上运行的动态语言,而且向静态类型化语言添加动态功能也会更容易。

动态语言可以在运行时标识对象的类型,而在类似 C# 和 Visual Basic 的静态类型化语言中,你必须在设计时指定对象类型。动态语言在开发中体现的优点如下。

- 可以使用快速反馈循环。在输入几条语句之后立即执行它们以查看结果。
- 同时支持自上而下的开发和更传统的自下而上的开发。例如,当你使用自上而下的方法时,可以调用尚未实现的函数,然后在需要时添加基础实现。
- 更易于进行重构和代码修改操作,原因是不必在代码中四处更改静态类型声明。
- 利用动态语言可以生成优秀的脚本语言。
- 利用新的命令和功能,客户可以轻松地扩展使用动态语言创建的应用程序。
- 动态语言还经常用于创建网站和测试工具、维护服务器程序、开发各种实用工具以及执行数据转换。

动态语言运行时的目的是允许动态语言系统在.NET 框架上运行,并为动态语言提供 .NET 互操作性。在 Visual Studio 2010 中,动态语言运行时将动态对象引入到 C# 和 Visual Basic 中,以便这些语言能够支持动态行为,并且可以与动态语言进行互操作,同时动态语言运行时还可帮助你创建支持动态操作的库。

与公共语言运行时类似,动态语言运行时是 .NET 4.0 框架的一部分,并随.NET Framework 4.0 和 Visual Studio 2010 安装包一起提供。

动态语言运行时与公共语言运行时相比具有以下优点。

(1) 简化动态语言到 .NET 4.0 框架的移植

借助于动态语言运行时，语言实施者不必再按传统的方式来创建词法分析器、语法分析器、语义分析器、代码生成器以及其他工具。动态语言运行时和 .NET4.0 框架可以自动执行许多代码分析和代码生成任务。这样，语言实施者就可以将精力集中在独有的语言功能上。

(2) 允许在静态类型化语言中使用动态功能

现有的 .NET 4.0 语言（如 C# 和 Visual Basic）可以创建动态对象，并将动态对象与静态类型化对象一起使用。例如，C# 和 Visual Basic 可以将动态对象用于 HTML、文档对象模型（DOM）和 .NET 反射。

(3) 在 .NET 框架版本升级后同步获益

通过使用动态语言运行时实现的语言可以从将来的动态语言运行时和 .NET 框架改进中获益。例如，如果 .NET 框架发布的新版本改进了垃圾回收器或加快了程序集加载时间，则通过使用动态语言运行时实现的语言会立即获得相同的益处。如果动态语言运行时优化了某些方面（如编译功能得到改进），则通过使用动态语言运行时实现的所有语言的性能也会随之提高。

(4) 允许共享库和对象

动态语言运行时在让使用一种语言实现的对象和类库可供其他语言使用的同时，还允许在静态类型化语言和动态语言之间进行相互操作。例如，C# 可以声明一个动态对象，而此对象可以使用动态语言编写的类库。同时，动态语言也可以使用 .NET 框架中的类库。

(5) 提供快速的动态调度和调用

动态语言运行时通过支持高级多态缓存，能够快速执行动态操作。

动态语言运行时首先会创建一些规则以将使用对象的操作绑定到必需的运行时实现，然后缓存这些规则，以避免在对同一类型的对象连续执行相同代码期间，出现将耗尽资源的绑定计算。

图 1-3 显示了动态语言运行时的体系结构。

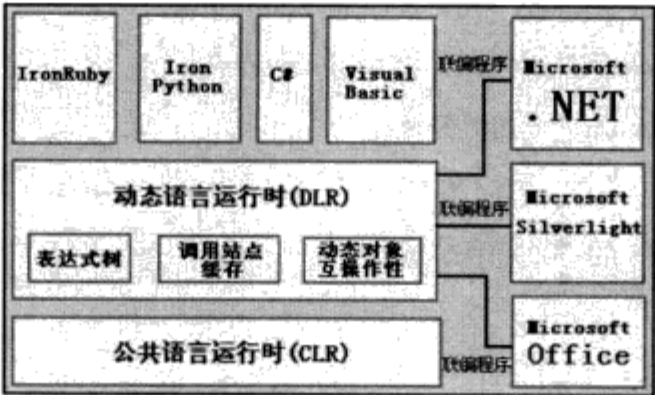


图 1-3 DLR 体系结构

上图中动态语言运行时向公共语言运行时中添加了一组服务，以便更好地支持动态语言。这些服务包括：

(1) 表达式树

动态语言运行时使用表达式树来表示语言语义。为此，动态语言运行时对 LINQ 表达式树进行了扩展，以便包括控制流、工作分配以及其他语言建模节点。

(2) 调用站点缓存

动态调用站点是代码中用于对动态对象执行类似 $a+b$ 或 $a.b()$ 的操作的位置。动态语言运行时将缓存 a 和 b 的特性（通常是这些对象的类型）以及有关操作的信息。如果之前已执行过此类操作，则动态语言运行时将从缓存中检索所有必需的信息，以实现快速调度。

(3) 动态对象互操作性

动态语言运行时提供一组表示动态对象和操作的类和接口，可供语言实施者和动态库的作者使用。这些类和接口包括 `IDynamicMetaObjectProvider`、`DynamicMetaObject`、`DynamicObject` 和 `ExpandoObject`。

动态语言运行时通过在调用站点中使用联编程序，不仅可以与 .NET Framework 通信，还可以与其他基础结构和服务（包括 Silverlight 和 COM）通信。联编程序将封装语言的语义，并指定如何使用表达式树在调用站点中执行操作。这样，使用动态语言运行时的动态和静态类型化语言就能够共享类库，并获得对动态语言运行时支持的所有技术的访问权。

1.2.5 .NET 类库

.NET 4.0 框架的另一个主要组件是类库，它是一个综合性的面向对象的可重用类型集合，例如 ADO.NET、ASP.NET 等。.NET 基类库位于公共语言运行库的上层，与 .NET Framework 紧密集成在一起，可被 .NET 支持的任何语言所使用，这也就是为什么 ASP.NET 中可以使用 C#、VB.NET、VC.NET 等语言进行开发的原因。.NET 类库非常丰富，提供数据库访问、XML、网络通信、线程、图形图像、安全、加密等多种功能服务。类库中的基类提供了标准的功能，如输入输出、字符串操作、安全管理、网络通信、线程管理、文本管理和用户界面设计功能。这些类库使得开发人员更容易地建立应用程序和网络服务，从而提高开发效率。



提示

.NET 4.0 类库非常庞大，读者不可能熟悉其提供的每一个类和集合，读者只需要熟悉一些常用的类和命名空间即可，比如：Request 类、Response 类和 System.Web 命名空间等。

1.3 ASP.NET 应用程序

ASP.NET 应用程序是一系列资源和配置的组合，这些资源和配置只在同一个应用程序内共享，而其他应用程序则不能享用这些资源和配置，有时尽管它们发布在同一台服务器上。就技术而言，每个 ASP.NET 应用程序都运行在一个单独的应用程序域，应用程序域是内存中的独立区域，这样可以确保在同一台服务器上的应用程序不会相互干扰，不至于因为其中一个应用程序发生错误就影响到其他应用程序的正常进行。同样，应用程序域限制一个应用程序中的 Web 页面访问其他的应用程序的存储信息。每个应用程序单独运行，具有自己的存储、应用和会话数据。

ASP.NET 应用程序的标准定义是：文件、页面、处理器、模块和可执行代码的组合，并且它们能够从服务器上的一个虚拟目录中被引用。换句话说，虚拟目录是界定应用程序的基本组织结构。

1.3.1 ASP.NET 页面与服务器交互

ASP.NET 页面作为代码在服务器上运行。因此，要得到处理，页面必须在用户单击按钮（或者当用户选中复选框或与页面中的其他控件交互）时提交到服务器。每次页面都会提交回自身，以便它可以再次运行其服务器代码，然后向用户呈现其自身的新版本。传递 Web 页面的过程如下。

- 用户请求页面。使用 HTTP GET 方法请求页面，页面第一次运行，执行初步处理（如果已通过编程让它执行初步处理）。
- 页面将标记动态呈现到浏览器中。
- 用户键入信息或从可用选项中进行选择，然后单击按钮。如果用户单击链接而不是按钮，页面可能仅仅定位到另一页，而第一页不会被进一步处理。
- 页面发送到 Web 服务器。浏览器执行 HTTP POST 方法，该方法在 ASP.NET 中称为“回发”。更明确地说，页面发送回其自身。例如，如果用户正在使用 Default.aspx 页面，则单击该页上的某个按钮可以将该页发送回服务器，发送的目标则是 Default.aspx。
- 在 Web 服务器上，该页再次运行。并且可在页面上使用用户键入或选择的信息。
- 页面执行通过编程所要实行的操作。
- 页面将其自身呈现回浏览器。

只要用户在该页面中工作，此过程就会循环继续。用户每次单击按钮时，页面中的信息会发送到 Web 服务器，然后该页面再次运行。每个循环称为一次“往返行程”。由于页面处理发生在 Web 服务器上，因此页面可以执行的每个操作都需要一次到服务器的往返行程。

1.3.2 ASP.NET Web 窗体

在 ASP.NET 中，发送到客户端浏览器中的网页是经过 .NET 框架中的基类动态生成的。这个基类就是 Web 页面框架中的 Page 类，而实例化的 Page 类就是一个 Web 窗体，也就是 Web Forms。因此说，一个 ASP.NET 页面就是一个 Web 窗体。而作为窗体对象，就具有属性、方法和事件，可以作为容器容纳其他控件。

Web 窗体是一个保存为后缀名为.aspx 的文本文件，可以使用任何文本编辑器打开和编写它，我们知道 ASP.NET 是编译的运行机制，为了简化开发人员的工作，一个.aspx 页面不需要手工编译，而是在页面被调用时，有公共语言运行时自行决定是否要被编译。

在 Web 窗体可以使用一般的 HTML 窗体控件，但 ASP.NET 也提供了自己的可以在服务器上运行的 Web 窗体控件。

1.3.3 ASP.NET 4.0 的新特性

要涵盖 ASP.NET 4.0 的所有新功能，本节的篇幅远远不够，因为整个 .NET 4.0 框架以及基础运行时中的改进数不胜数。所以，下面把 ASP.NET 4.0 在 ASP.NET 3.5 基础上增加的重要特性做一简要的介绍。

1. ASP.NET MVC 2.0

微软公司在 2007 年 12 月发布了第一个 ASP.NET 3.5 MVC 预览版后，接着就分别发布了 8 个后续的测试版本，终于在 2009 年 3 月正式发布了 ASP.NET 3.5 MVC 1.0 版，但是并没有来得及集成到 Visual Studio 2008 开发环境中。而这次的 ASP.NET MVC 2.0 版本被集成到了 Visual Studio 2010 开发环境中作为了一个项目模板出现。

MVC (Model View Controller) 模式是一种结构设计模式，一般根据应用程序功能的不同，分为三个主要部分：模型、视图和控制器。ASP.NET MVC 2.0 基于 MVC 设计模式，它提供了一种 ASP.NET Web 窗体模式之外的另一种开发模式，从而让 .NET 开发者有了不同的选择。ASP.NET MVC 框架是一个可测试性非常高的轻型框架，与基于 Web 窗体的应用程序一样，它集成了现有的 ASP.NET 功能，如母版页和基于成员资格的身份验证。

MVC 模式可以帮助开发者创建应用程序的各层（输入逻辑、业务逻辑和 UI 逻辑）分离的应用程序，同时可在这些元素之间提供松散耦合。该模式指定每种逻辑在应用程序中应处的位置。UI 逻辑位于视图中，输入逻辑位于控制器中，业务逻辑位于模型中，在生成应用程序时，通过使用这种分离方式，可以帮助你化繁为简，因为它可以使你侧重于一次实现应用程序的一个方面，例如，可以侧重于独立于业务逻辑的视图。MVC 应用程序的这三个主要组件之间的松散耦合也可促进并行开发。例如，一个开发人员可以从事视图方面的工作，第二个开发人员可以从事控制器逻辑方面的工作，第三个开发人员可以侧重于模型中的业务逻辑。

ASP.NET MVC 2.0 框架具有以下优点。

- 通过将应用程序分为模型、视图和控制器，化繁为简，使编程工作更加轻松。
- 它不使用视图状态或基于服务器的窗体。这使得 MVC 框架特别适合想要完全控制应用程序行为的开发人员。
- 它使用一种通过单一控制器处理 Web 应用程序请求的前端控制器模式。这使你可以设计一个支持丰富路由基础结构的应用程序。它为测试驱动的开发 (TDD) 提供了更好的支持。
- 它非常适合大型开发团队开发的 Web 应用程序，以及需要对应用程序行为进行极度控制的 Web 设计人员。

对于 ASP.NET MVC 2.0 的具体内容请参考本书第 16 章 ASP.NET MVC 程序开发。

2. ASP.NET AJAX 4.0

在 ASP.NET 4.0 中，ASP.NET AJAX 4.0 的出现让 ASP.NET 在 AJAX 的运用上得到了很大的提高。使用 ASP.NET AJAX 4.0 功能，可以快速创建既提供丰富的用户体验又提供响应迅速的常见用户界面 UI 元素的网页。ASP.NET AJAX 4.0 包含客户端脚本库 (Microsoft Ajax Library)，这些库融合了跨浏览器的 ECMAScript (JavaScript) 技术和动态 HTML (DHTML) 技术。可以将客户端脚本库用作生成 AJAX 的应用程序框架。但是使用前必须下载并安装客户端脚本库，该库是独立于 .NET Framework 4.0 和 Visual Studio 2010 发行的，可以通过访问 Microsoft AJAX 网站来下载最新版本。使用客户端脚本库，可以生成一个完全在浏览器中运行的、数据库驱动的 Web 应用程序。该库中包括一些可以完成下列任务的功能：

- 通过与 Web 服务交互，检索和编辑数据库记录。
- 创建客户端主/详细信息窗体。
- 使用 Microsoft AJAX 客户端控件来创建交互功能丰富的网页。
- 使用 JSONP 从其他域中检索数据。

ASP.NET AJAX 4.0 中包括全新的 DataView 的控件允许开发人员将模板扩展到整个数据列表；标记扩展允许方便快捷地操作模板引擎的行为；自定义行为可用于处理复杂逻辑，而不用在过度复杂的条件属性中实现该逻辑。通过使用 ASP.NET AJAX 4.0 可以改进用户体验并大大提高 Web 应用程序的效率。本书在后面第 14 章 ASP.NET AJAX 技术中，对 ASP.NET AJAX 4.0 作一个简单的介绍。

3. ASP.NET WebForms 4

截止到 ASP.NET 发布版本 4 时，ASP.NET Web Form 已经走过了八个年头。在此期间，开发团队坚持不懈地优化着该框架，进行着各种各样的改进。这次 Web 窗体中的重大变化解决了该框架的一些主要缺点。

(1) 加强对视图状态 (ViewState) 的控制

视图状态数据包含在页面的 HTML 中，在将页面发送至客户端和发回时会延长所用时间。存储多余的视图状态可能会导致性能显著降低。在 ASP.NET 的早期版本中，可通过禁用特定控件的视图状态来减轻视图状态对页面性能的影响。所以在 ASP.NET 4.0 中为 Control 类增加了一个新属性：ViewStateMode。使用该属性，可以针对页面上未显式启用视图状态的所有控件禁用视图状态。

(2) 支持最近引入的浏览器和设备

ASP.NET 包含一项名为“浏览器功能”的功能，可用于确定用户使用的浏览器的功能。有关特定浏览器功能的信息由浏览器定义文件定义。在 ASP.NET 4.0 中，这些浏览器定义文件已更新为包含有关最近引入的浏览器和设备（如 Google Chrome, Research in Motion BlackBerry 智能电话和 Apple iPhone）的信息。现有的浏览器定义文件也已更新。

(3) 支持对 Web 窗体使用 ASP.NET 路由

ASP.NET 4.0 中增加了对使用 Web 窗体进行路由的内置支持。通过此功能可将应用程序配置为使用对用户和搜索引擎有意义的 URL，这样无需指定物理文件名。使用这项功能，可以提高站点的用户友好度，并增加站点内容被搜索引擎发现的概率。

(4) 加强对生成的 ID 的控制

通常情况下，可以使用 document.getElementById 方法在客户端脚本中获得对 HTML 元素的引用，需要将引用的 HTML 元素的 Id 特性值传递到该方法。对于 ASP.NET 服务器控件呈现的元素，ASP.NET 的早期版本可能难以甚至无法实现上述操作。因为并非总能预测 ASP.NET 将生成的 Id 值，而 ASP.NET 也可能生成很长的 Id 值。对于将针对标记中控件的单个实例生成多个行的数据控件，该问题特别难以解决。如今，在 ASP.NET 4.0 中新的 ClientIDMode 属性可以方便用户编写引用服务器控件呈现的 HTML 元素的客户端脚本。

(5) 支持数据源控件的筛选

对于创建数据驱动的网页开发人员来说，一项十分常见的任务就是筛选数据。该任务的传统执行方法是在数据源控件中生成 Where 子句。这种方法可能十分复杂，而且在某些情况下，通过 Where 语法无法充分利用基础数据库的全部功能。为简化筛选操作，ASP.NET 4.0 中增加了一个新的 QueryExtender 控件。可以将此控件添加到 EntityDataSource 或 LinqDataSource 控件以筛选这些控件返回的数据。QueryExtender 控件依赖于 LINQ，但我们无需了解如何编写 LINQ 查询即可使用这个查询扩展程序。

4. ASP.NET Web Deployment

Visual Studio 2010 开发环境中的网页设计器已经做了改进，提高了 CSS 的兼容性，增加了对 HTML 和 ASP.NET 标记代码段的支持，并提供了重新设计的 JScript 智能感知功能。

- 提高了 CSS 的兼容性。与 Visual Studio 的早期版本相比，该 CSS 设计器可以更好地与 CSS 2.1 标准相容，能够完好保留 HTML 源代码。
- 支持 HTML 和 JScript 代码段标记。在 Visual Studio 2010 的 HTML 编辑器中，智能感知 (IntelliSense) 可以自动完成标记名称的完整显示；它完善了 Visual Studio 早期版本中对 JScript、C# 和 Visual Basic 代码段的支持功能。其所包含的 200 多个代码段，能够自动完成常见的 ASP.NET 和 HTML 标记。更可贵的是，我们不仅可以下载其他代码，也可以编写自己的代码段，封装成常见任务标记块。
- 增强了 JScript 的智能感知功能。在 Visual Studio 2010 中，JScript 的智能感知经过了重新设计，已经可以识别各种动态生成的对象，几乎不存在处理延迟的情况，带给我们更好的编程体验。同时兼容性显著提高，几乎支持所有的第三方类库。

1.4 Visual Studio 2010 开发环境

每一个正式版本的 .NET 框架都会有一个与之对应的高度集成的开发环境，微软称之为 Visual Studio，中文的意思是可视化工作室。随同 ASP.NET 4.0 一起发布的开发工具是 Visual Studio 2010，它对基于 ASP.NET 4.0 的项目开发有很大帮助，使用 Visual Studio 2010 可以很方便地进行各种项目的创建、具体程序的设计、程序调试和跟踪以及项目发布等等。

1.4.1 安装 Visual Studio 2010

Visual Studio 2010 目前有三个版本：Visual Studio 2010 Professional 版本、Visual Studio 2010 Premium 版本和 Visual Studio 2010 Ultimate 版本，其中前两种用于个人和小型开发团队采用最新技术开发应用程序和实现有效的业务目标，第三种为体系结构、设计、开发、数据库开发以及应用程序测试等多任务的团队提供集成的工具集，在应用程序生命周期的每个步骤，团队成员都可以继续协作并利用一个完整的工具集与指南。

下面我们一起来学习如何将 Visual Studio 2010 Professional 安装到电脑中。具体步骤如下：

01 可以到 <http://www.microsoft.com/visualstudio/zh-tw/products/2010-editions/professional> 下

载到 Visual Studio 2010 的试用版，也可以去购买正版安装程序。

- 02 打开安装程序后，首先进入图 1-4 所示的“安装向导界面”。
- 03 选择“安装 Microsoft Visual Studio 2010”即可进入如图 1-5 所示的“资源复制过程”。



图 1-4 安装向导界面



图 1-5 资源复制过程

- 04 在资源复制完毕后，进入如图 1-6 所示的“加载组件的过程”。
- 05 组件加载完毕后，“下一步”按钮被激活，如图 1-7 所示。

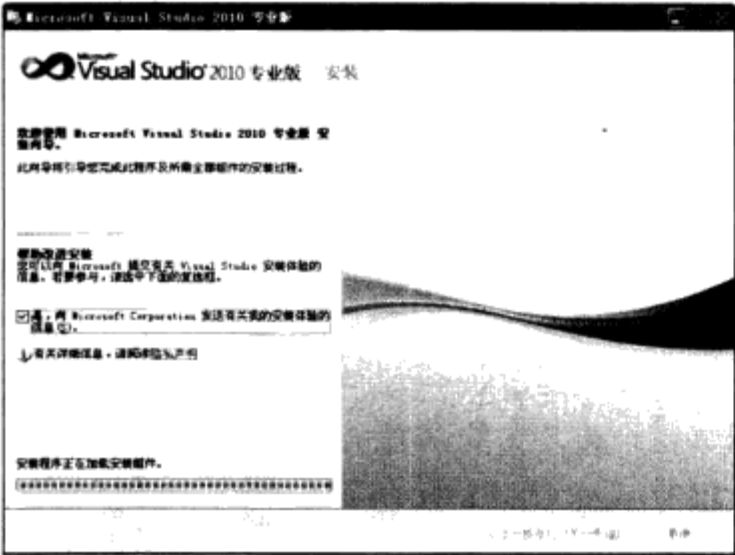


图 1-6 加载组件的过程

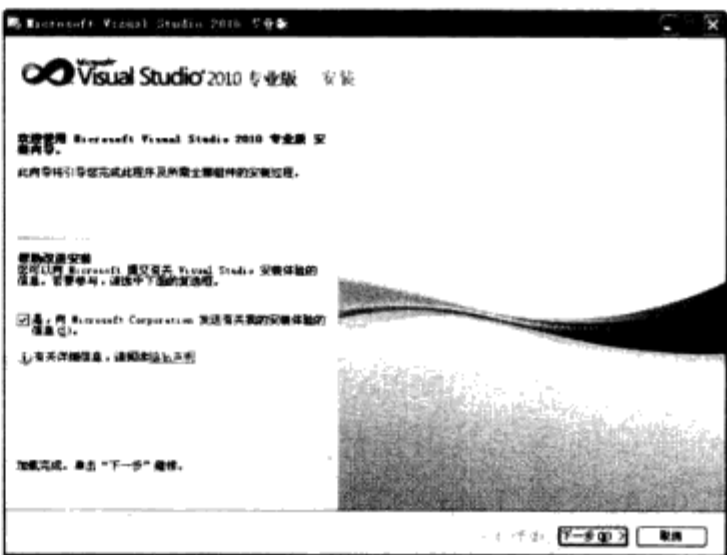


图 1-7 “下一步”按钮被激活

- 06 单击“下一步”按钮，进入如图 1-8 所示的“软件安装许可认证界面”。
- 07 选择“我已阅读并接受许可条款”单选按钮，并输入产品密钥和用户名称，“下一步”按钮会被激活，单击“下一步”按钮，进入如图 1-9 所示的“安装功能和路径界面”。选择“完全”单选按钮，并选择相应的安装路径，然后单击“安装”按钮。

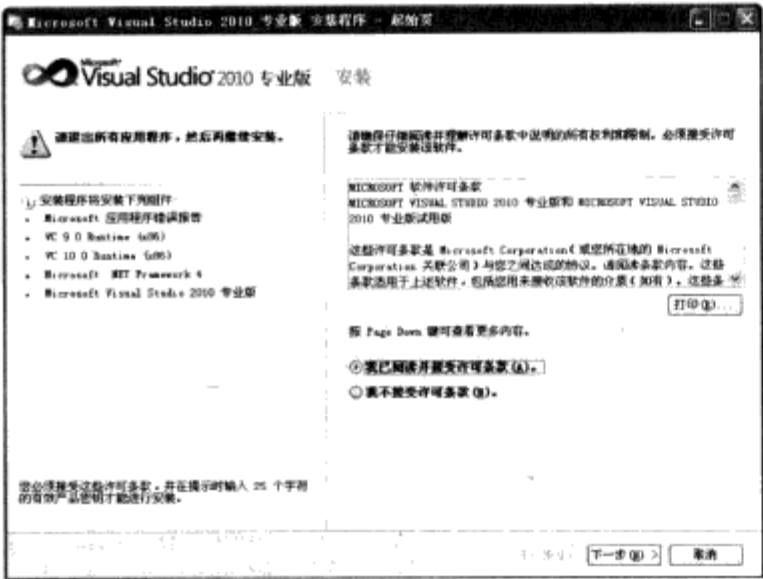


图 1-8 软件安装许可认证界面

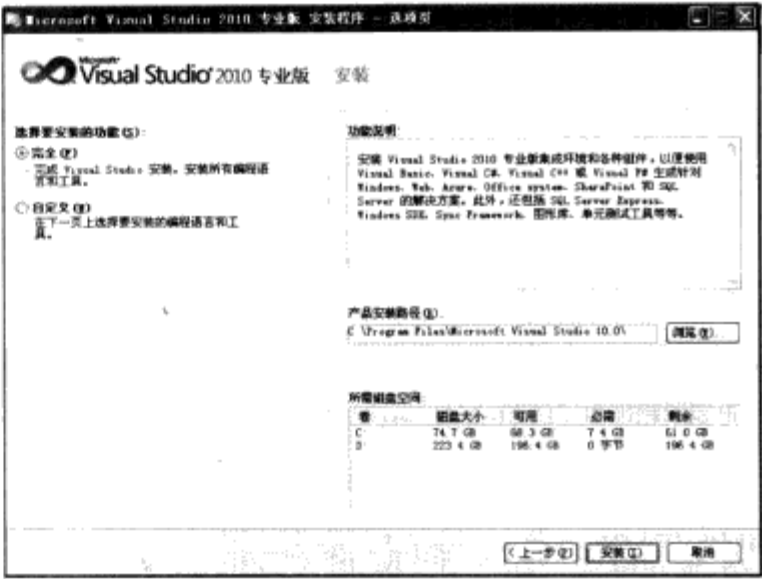


图 1-9 安装功能和路径界面

- 08** 进入如图 1-10 所示的安装过程显示界面，单击“下一步”按钮，开始安装并显示当前安装的组件。
- 09** 当所有组件安装成功后，进入图 1-11 所示的界面，显示已经成功地安装 Visual Studio 2010，最后单击“完成”按钮，结束安装过程。



图 1-10 安装过程显示界面



图 1-11 安装完成界面

至此 Visual Studio 2010 成功地被安装到本地的机器上。

1.4.2 创建 Web 项目

安装完成 Visual Studio 2010 开发环境之后，我们使用这一强大的工具来创建一个 ASP.NET 4.0 项目，使大家对 Visual Studio 2010 有一个初步的熟悉，具体步骤如下：

- 01** 选择“开始”|“所有程序”|“Microsoft Visual Studio 2010”|“Microsoft Visual Studio 2010”命令，打开 Visual Studio 2010，进入主界面，如图 1-12 所示。



图 1-12 Visual Studio 2010 界面

Visual Studio 2010 的主界面各组成部分如下：

- 标题栏、菜单栏、工具栏和状态栏。用于实现软件所有的功能和功能导航。
- 起始页。包括连接到团队、新建项目和打开项目的快捷按钮；最近使用的项目列表和 Visual Studio 2010 入门、指南和新闻列表的选项卡。
- 工具箱。提供了设计页面时常用的各种控件，只要简单的将控件拖动到设计页面即可方便的使用。
- 解决方案资源管理器。用于对解决方案和项目进行统一的管理，其主要组成是各种类型的文件。
- 团队资源管理器。是一个简化的 Visual Studio Team System 2010 环境，专用于访问 Team Foundation Server 服务。
- 服务器资源管理器。用于打开数据连接，登录服务器，浏览它们的数据库和系统服务。

02 单击主界面起始页上的“新建项目”快捷按钮或选择“文件”|“新建项目”命令，打开如图 1-13 所示的“新建项目”对话框。“新建项目”对话框左边窗口中显示“已安装模板”的树状列表，中间窗口显示与选定模板相对应的项目类型列表，右边窗口是对模板的描述。这里我们需要打开“Visual C#”类型节点，选择“Web”子节点这个模板，同时在右边窗口显示了可以创建的“Web”项目类型列表。选择“ASP.NET 空 Web 应用程序”，在“名称”文本框中输入项目名称，并在“位置”文本框中输入相应的存储路径，在“解决方案名称”文本框中输入解决方案名称。最后，单击“确定”按钮即可创建一个新的 Web 项目。



图 1-13 新建项目对话框

1.4.3 Web 项目管理

当创建一个新的网站项目之后，就可以利用资源管理器对网站项目进行管理，通过资源管理器，可以浏览当前项目所包含的所有的资源（.aspx 文件、.aspx.cs 文件、图片等），也可以向项目中添加新的资源，并且可以修改、复制和删除已经存在的资源。解决方案资源管理器如图 2-12 所示。

1. 添加新资源

在上节所创建网站的“解决方案资源管理器”中右键单击项目名称“WebApplication”，会弹出如图 1-15 所示的菜单。

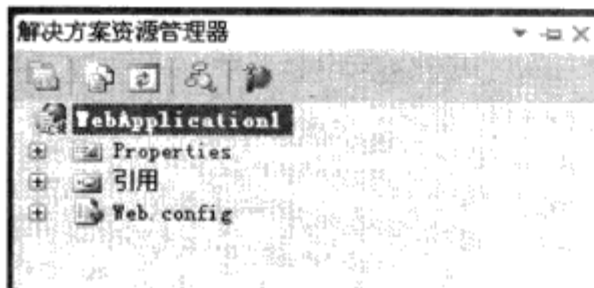


图 1-14 解决方案资源管理器

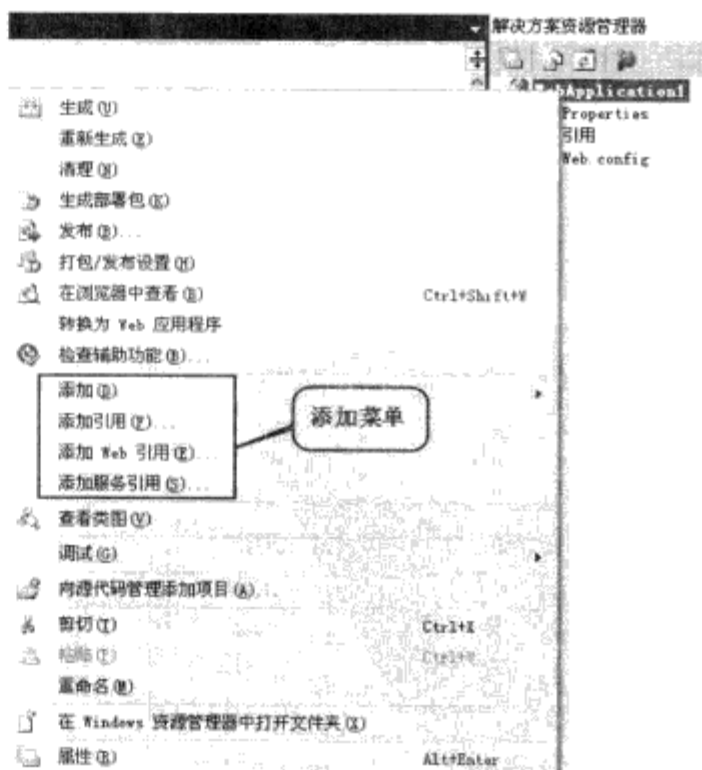


图 1-15 添加菜单

上图菜单中有多个添加项，分别是“添加”、“添加引用”、“添加 Web 引用”、“添加服务引用”。其中“添加引用”命令用来添加对类的引用，“添加 Web 引用”命令用来添加对存在于 Web 上的公开类的引用，“添加服务引用”命令用来添加对服务的引用。

选择“添加”命令，弹出如图 1-16 所示的下一级子菜单，包括“新建项”、“现有项”、“新建文件夹”和“添加 ASP.NET 文件夹”四个命令。其中“新建项”命令用来添加 ASP.NET 4.5 支持的所有文件资源；“现有项”命令用来把已经存在的文件资源添加到当前项目中去；“新建文件夹”命令用来向网站项目中添加一个文件夹；“添加 ASP.NET 文件夹”命令用来向网站项目中添加一个 ASP.NET 独有的文件夹。

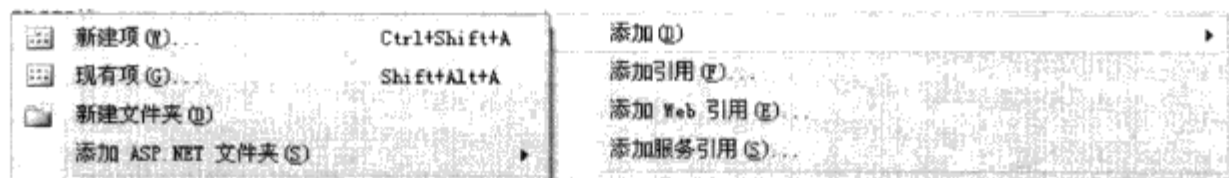


图 1-16 添加子菜单

选择“新建项”命令，打开如图 1-17 所示的“添加新项”对话框，在该对话框中选择“已安装模板”下的“Web”模板，并在模板文件列表中选“Web 窗体”，然后在“名称”文本框输入该文件的名称，最后单击“添加”按钮即可向网站项目中添加一个新的文件。

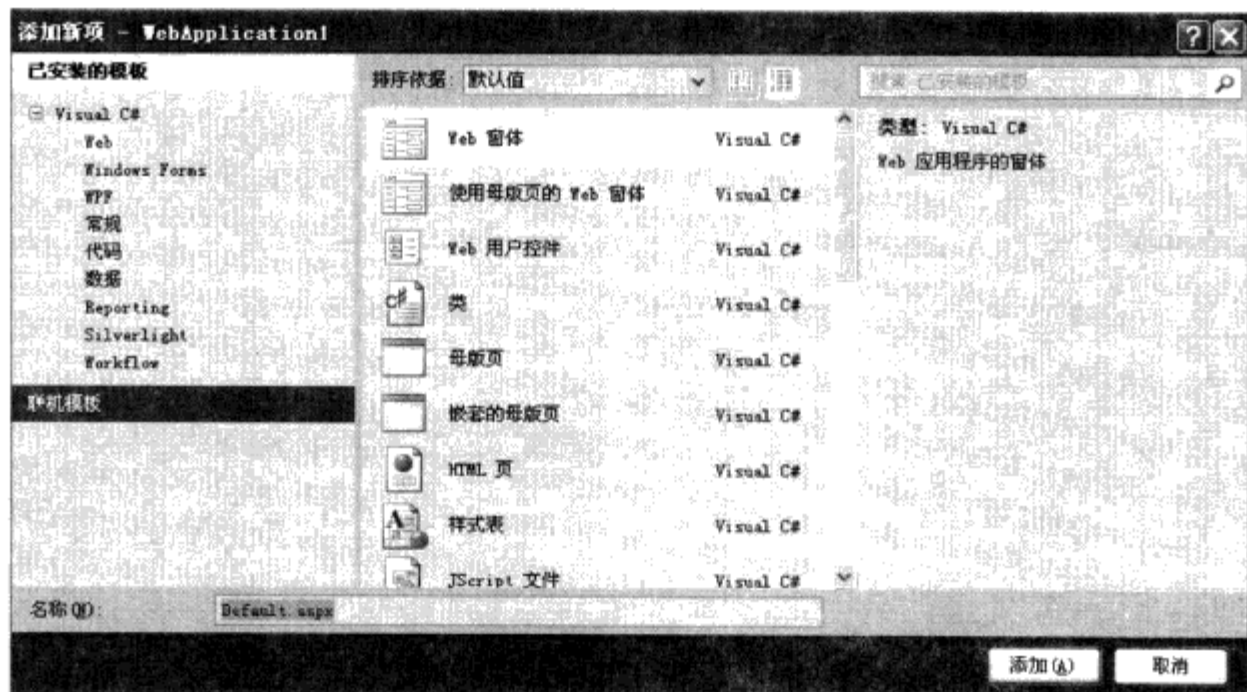


图 1-17 添加新项对话框

2. 编辑 Web 页面

在添加一个 Web 页面后，可以使用 Visual Studio 对它进行编辑，在资源管理器中双击某个要编辑 Web 页面文件，该页面文件就会在如图 1-18 所示左边的窗口中打开。

图 1-18 的页面文件编辑窗口可以编辑三种视图：设计视图、拆分视图和源视图。其中，设计视图用来显示设计的效果，并且可以从“工具箱”中直接把控件放置在设计视图中，“工具箱”是放置控件的容器，如图 1-19 所示；拆分视图同时显示设计视图和源视图；源视图显示设计源码，可以在该视图中直接通过编写代码来设计页面。

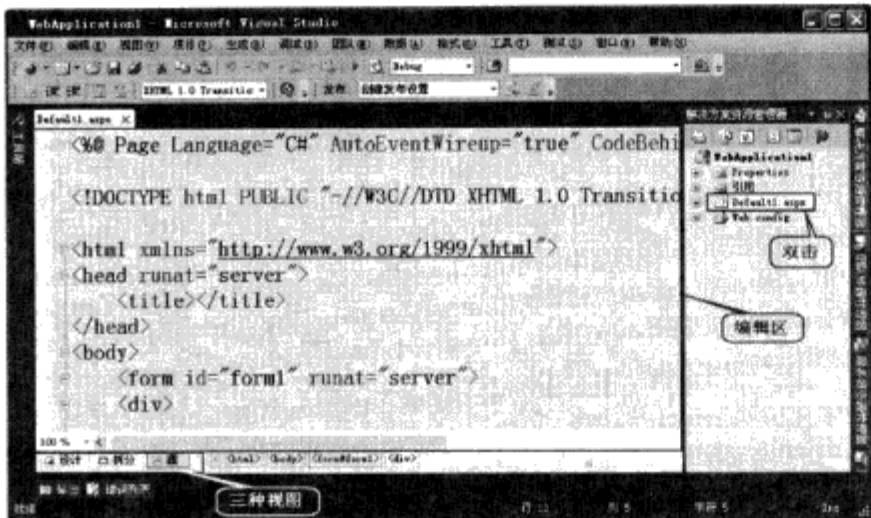


图 1-18 视图设计器

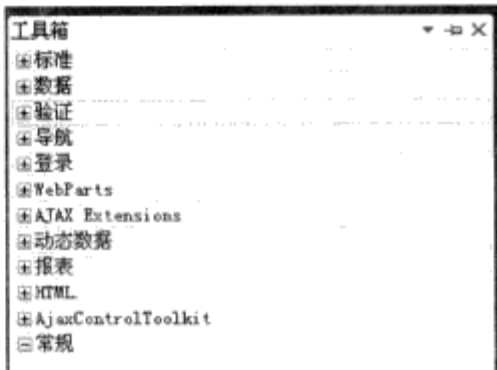


图 1-19 工具箱

3. 属性查看器

在 Web 页面设计视图下，右键单击某一个控件或页面的任何地方，在弹出的菜单中选择“属性”命令或者选择如图 1-20 所示的菜单栏上“视图”|“属性窗口”命令。

随即就会弹出如图 1-21 所示的控件“属性查看器”。在属性查看器中，可以编辑想要修改的各种控件属性，比如修改背景色，可以在 BgColor 后面的文本框中输入对应的颜色值，或者单击 BgColor 后面的按钮弹出颜色选择器，在颜色选择器中可以选择对应的颜色。

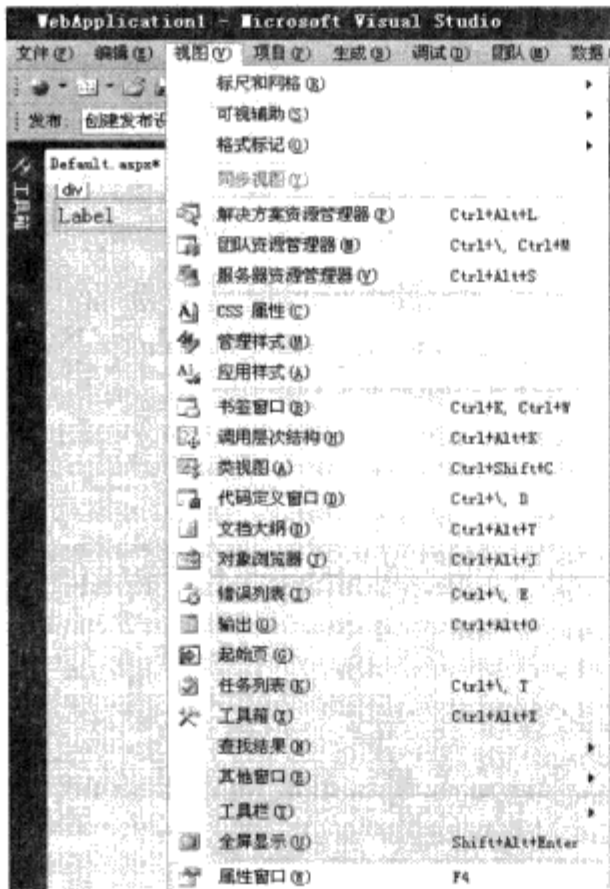


图 1-20 属性窗口菜单

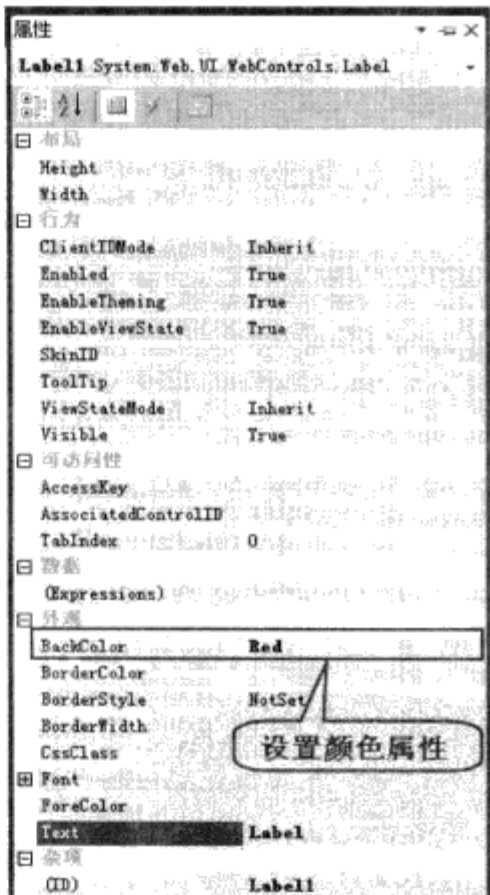


图 1-21 属性窗口

4. 编辑后台代码

在 Web 页面的设计视图下，双击页面的任何地方即可打开隐藏的后台代码文件，在此界面中，开发者可以编写与页面对应的后台逻辑代码。或者通过双击网站目录下的文件名，进入如图 1-22 所示的后台代码编辑区。

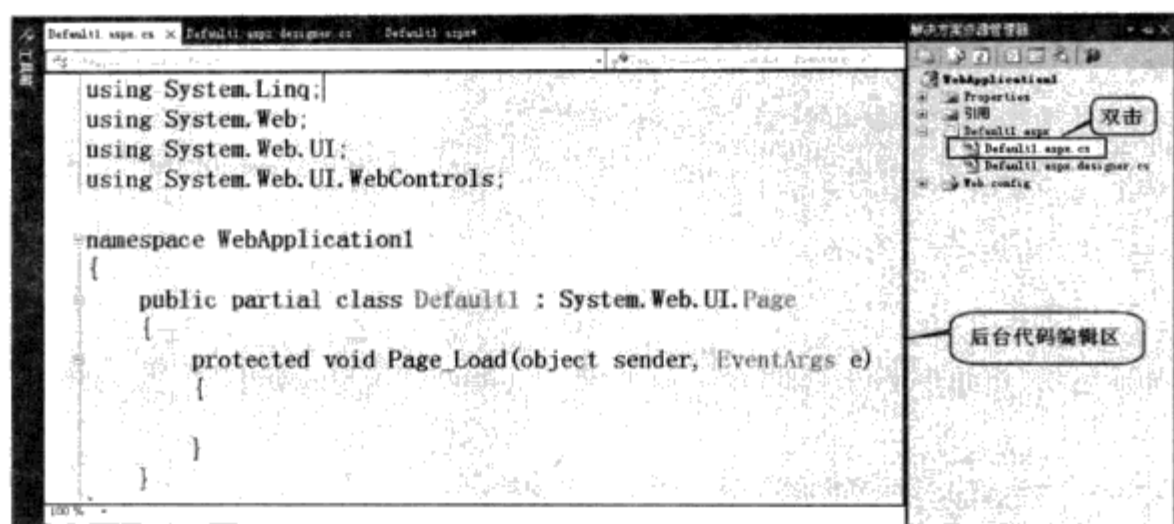


图 1-22 后台代码编辑区

5. 编译和运行应用程序

选择菜单栏上的“生成”|“生成网站”命令，如果生成成功，则屏幕下方的“输出”窗体中的内容如图 1-23 所示。

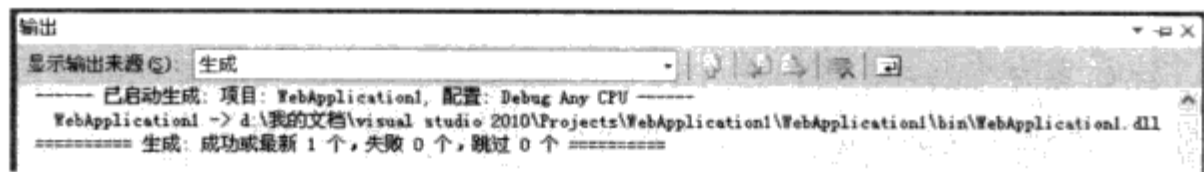



图 1-23 输出窗体

单击工具栏上的“启动调试”按钮  运行程序，浏览器就会显示程序的运行效果。



提示

如果已经安装了 Visual Studio 2010，可以选择安装 MSDN 的帮助文档，如果不安装此文档，也可以到 MSDN 网站在线查看相关文档。这个 MSDN 是每个 ASP.NET 开发人员最为重要的参考工具。

1.4.4 Visual Studio 2010 的新特性

与上一个版本 Visual Studio 2008 相比，Visual Studio 2010 集成开发环境新增的主要特性说明如下。

1. 窗口移动

文档窗口不再受限于集成开发环境（IDE）的编辑框架。现在可以将文档窗口停靠在 IDE 的边缘，或者将它们移动到桌面（包括辅助监视器）上的任意位置。如果打开并显示两个相关的文档窗口，则在一个窗口中所做的更改将立即反映在另一个窗口中。

工具窗口也可以进行自由移动，使它们停靠在 IDE 的边缘、浮动在 IDE 的外部或者填充部分或全部文档框架。这些窗口始终保持可停靠的状态。

2. 调用层次结构

调用层次结构可以帮助我们分析代码，并实现导航定位功能。在方法、属性、字段、索引器或者构造函数上单击右键，选择弹出菜单中的“查看调用层次结构”命令。在如图 1-24 所示的调用层次结构窗口能看到被调用的方法的层次结构，双击方法名称，马上可以定位到方法定义的地方。

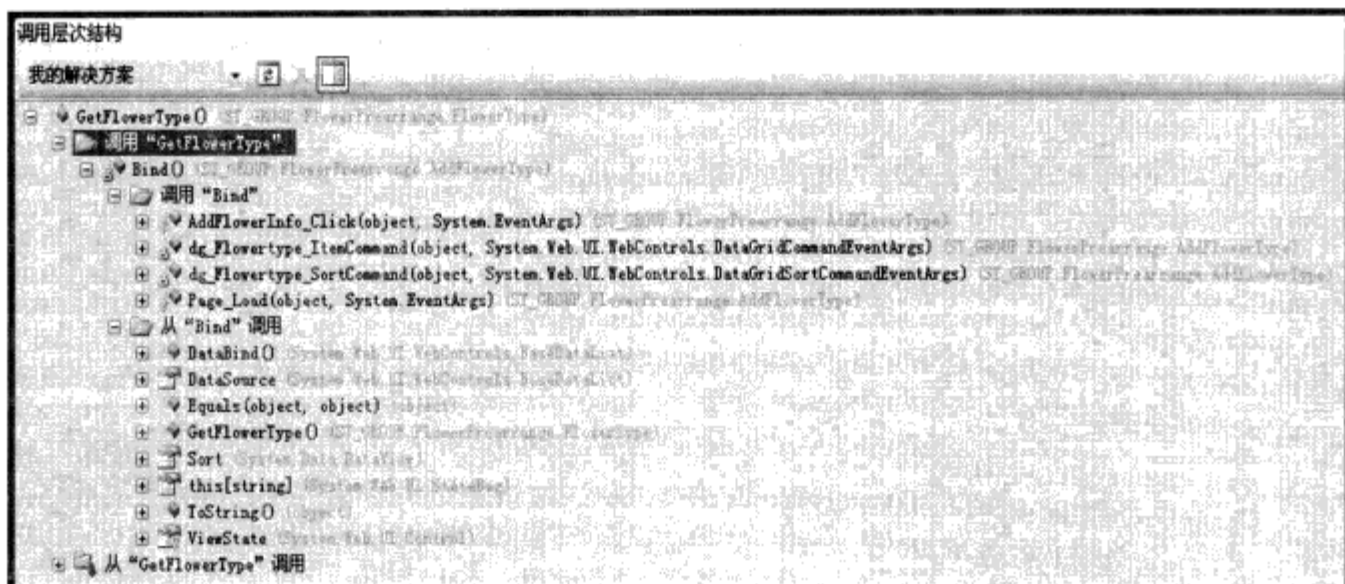


图 1-24 调用层次结构窗口

3. 定位搜索

这是一个使用字符进行快速搜索定位的工具。可以快速搜索源代码中的类型、成员、符号和文件。选择菜单栏上的“编辑”|“定位到”命令，打开如图 1-25 所示的定位搜索窗口。在搜索栏中输入查询内容（支持模糊查询功能）后，将列出相关结果信息。双击搜索结果可以直接转到代码所在位置。

4. 突出显示引用

将鼠标选中任何一个符号如方法、属性、变量等，在如图 1-26 所示的代码编辑器中将自动突出显示此符号的所有实例。你还可以通过快捷键“CTRL+SHIFT+向上/向下键”来从一个加亮的符号跳转到下一个加亮的符号。

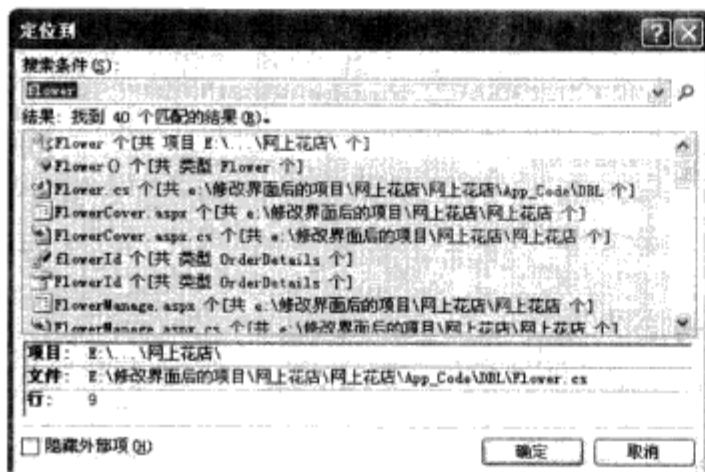


图 1-25 定位搜索窗口

```
FlowerType flowerType = new FlowerType();
DataView dv = flowerType.GetFlowerType();
this.dropType.DataSource = dv;
this.dropType.DataTextField = "Name";
this.dropType.DataValueField = "Id";
this.dropType.DataBind();
dropType.Items.Add("所有类型");
dropType.Items[dropType.Items.Count-1].Value = "-1";
dropType.SelectedIndex = dropType.Items.Count-1;
```

图 1-26 突出显示引用

5. 智能感知

在 Visual Studio 2010 中，智能感知（IntelliSense）功能进一步得到完善和加强。在我们输入一些关键字时，其搜索过滤功能并不只是将关键字作为查询项的开头，而是包含查询项所有位置。有时我们需要使用 switch、foreach、for 等类似语法结构，只需输入语法关键字，并单击两下“Tab 键”，Visual Studio 2010 就会自动完成相应的语法结构。这一功能大大地提高开发人员的编程效率。

1.5 配置 Web 服务器

开发 ASP.NET 4.0 应用程序之前，需要安装并配置 IIS。IIS 是 Internet Information Server 的缩写，是微软公司主推的 Web 服务器，通过 IIS，开发人员可以方便地调试程序和发布网站。

接下来我们就在电脑中安装和配置 IIS 服务器，具体步骤如下。

01 选择“开始”|“控制面板”|“添加或删除程序”命令，显示如图 1-27 所示的“添加或删除程序”对话框，该对话框显示当前已经安装的程序。在对话框的左侧选择“添加/删除 Windows 组件”图标。

02 在弹出如图 1-28 所示的“Windows 组件向导”对话框中找到“Internet 信息服务 (IIS)”，如果尚未安装，则其左侧的复选框不会被选中；如果复选框是不可选状态，说明 IIS 的组件没有全部安装。否则说明 IIS 已经全部安装，退出安装过程。

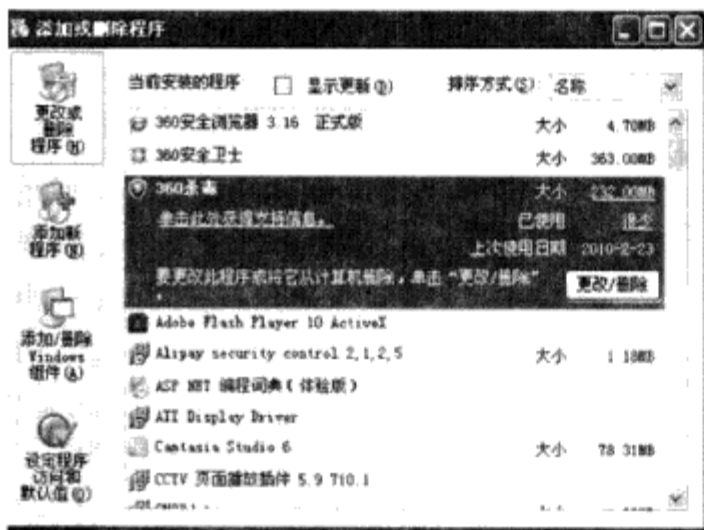


图 1-27 添加或删除程序对话框

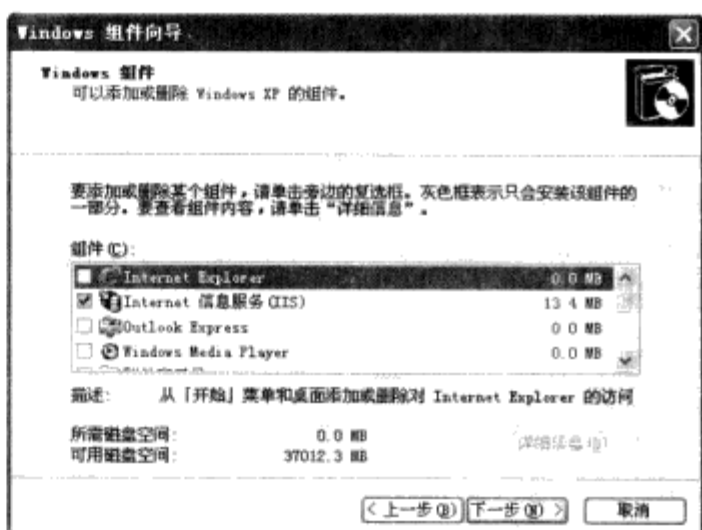


图 1-28 Windows 组件向导对话框

03 如果复选框没有被选中，则选中该复选框。如果复选框是不可选状态，则选中该项，单击“详细信息”按钮。

04 在弹出如图 1-29 所示“Internet 信息服务 (IIS) 对话框”中选择要安装的选项，对于本书来说，“公用文件”是一定要选中的。选择要安装的项后，单击“确定”按钮，返回到“Windows 组件向导”对话框。单击“下一步”按钮安装 IIS，此时可能会提示用户将 Windows XP 系统盘放入光驱。

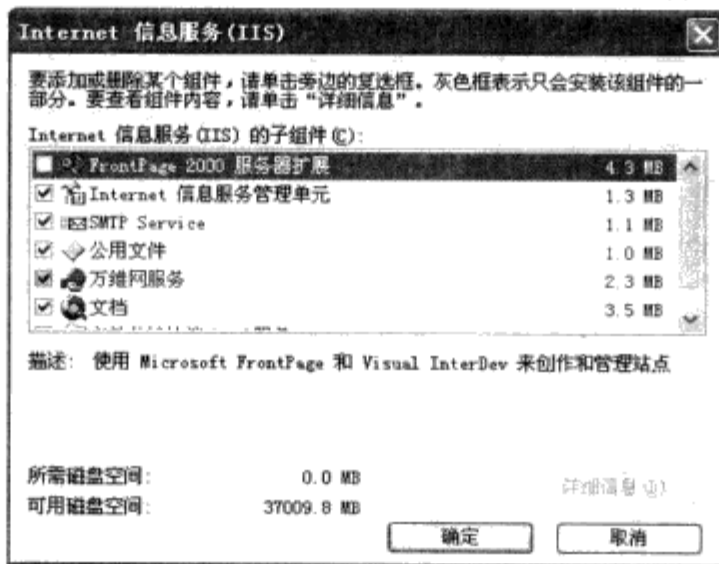


图 1-29 Internet 信息服务 (IIS) 对话框

05 单击“完成”按钮，完成 IIS 的安装。

- 06 选择“开始”|“控制面板”|“管理工具”|“Internet 信息服务”命令，弹出如图 1-30 所示的“Internet 信息服务”对话框，依次展开“根”节点、“网站”节点、“默认网站”节点。
- 07 右键单击“默认网站”节点，弹出如图 1-31 所示的菜单。用户可以选择“停止”关闭 IIS 服务，也可以选择“暂停”暂停 IIS 服务。



图 1-30 Internet 信息服务对话框



图 1-31 选择菜单

当用户通过 HTTP 浏览位于 Web 服务器上的一些 Web 页面时,Web 服务器需要确定与该页面对应的文件位于服务器硬盘上的什么位置。事实上，在由 URL 给出的信息与包含页面文件的物理位置（在 Web 服务器的文件系统中）之间有着重要的关系。这个关系是通过虚拟目录来实现。

虚拟目录相当于物理目录在 Web 服务器机器上的别名，它不仅使用户避免了冗长的 URL，也是一种很好的安全措施，因为虚拟目录对所有浏览者隐藏了物理目录结构。下面介绍创建虚拟目录的步骤。

- 01 在硬盘上创建一个物理目录，这里在 C 盘的根目录下创建一个目录，命名为“Sample”。
- 02 启动“Internet 信息服务”，右键单击“默认网站”节点，在图 1-31 所示的菜单中选择“新建”|“虚拟目录”命令，启动“虚拟目录创建向导”。
- 03 弹出如图 1-32 所示的“虚拟目录创建向导”对话框中，单击“下一步”按钮。

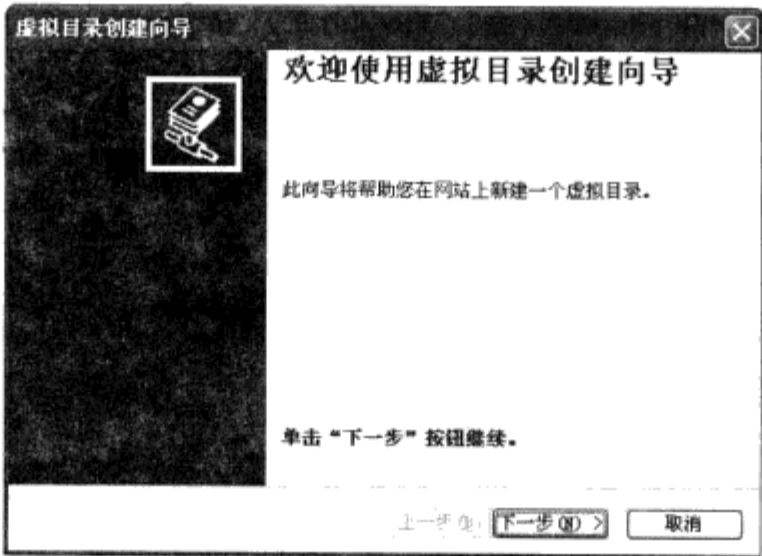


图 1-32 虚拟目录创建向导对话框

04 弹出如图 1-33 所示，在“别名”文本框中输入虚拟目录的名字，这里命名为“Sample”，和它的物理目录的名字相同。然后单击“下一步”按钮。

05 弹出如图 1-34 所示的“网站内容目录”对话框，选择刚才创建的物理目录“C:\Sample”，单击“下一步”按钮。

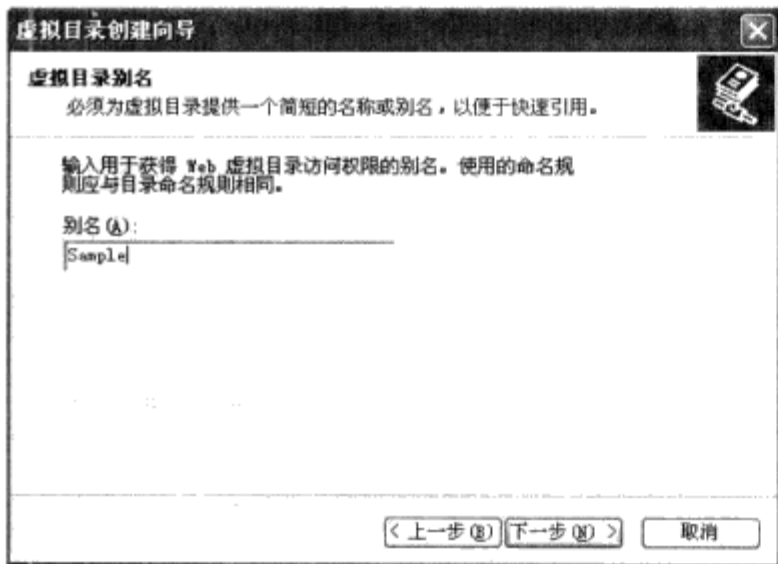


图 1-33 虚拟目录别名对话框

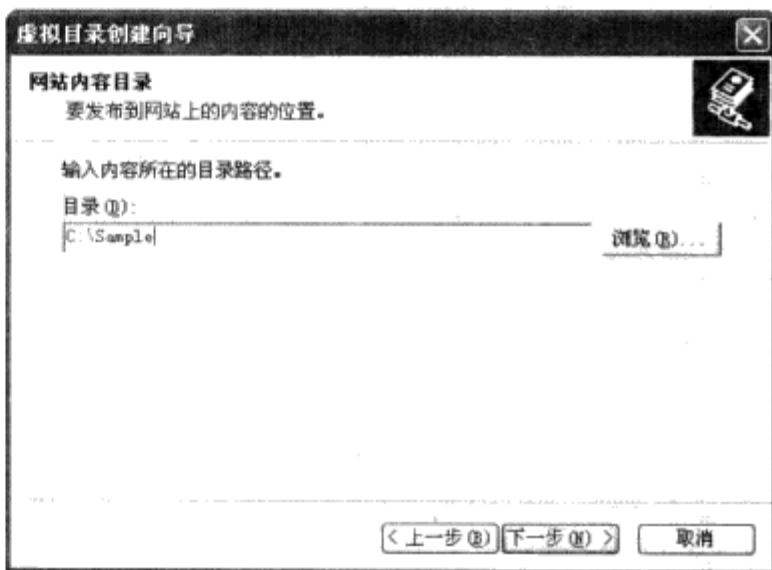


图 1-34 网站内容目录对话框

06 在弹出如图 1-35 所示的“访问权限”对话框中设置虚拟目录的访问权限，除非读者明白自己需要什么样的权限，否则不要改变创建时默认的权限。单击“下一步”按钮。

07 弹出如图 1-36 所示的，单击“完成”按钮，完成虚拟目录的创建。

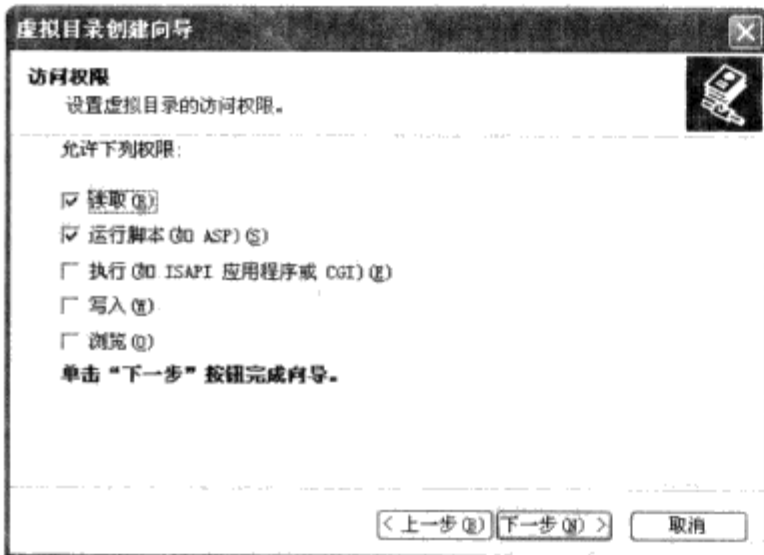


图 1-35 访问权限对话框

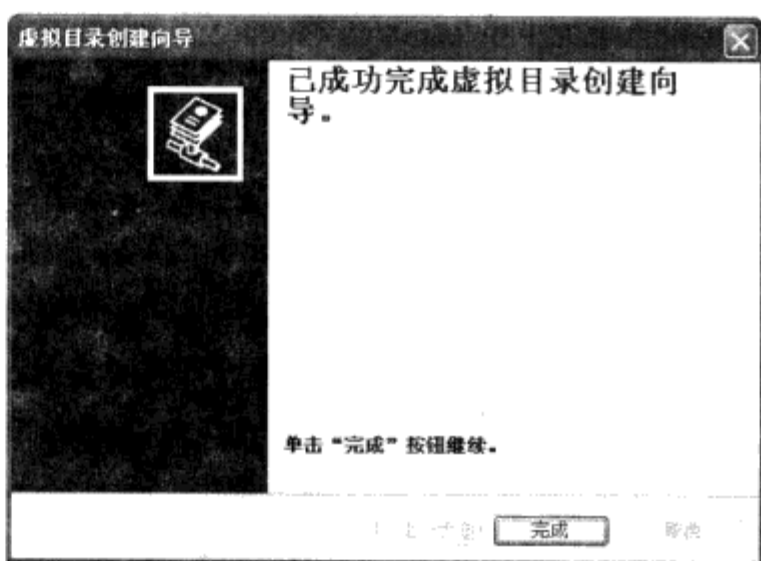


图 1-36 完成虚拟目录对话框

08 此时，在“Internet 信息服务”窗体的目录树中将显示该“Sample”虚拟目录，如图 1-37 所示。

09 在如图 1-38 所示“Internet 信息服务”窗体中右键单击“Sample”虚拟目录，从弹出的菜单中选择“属性”命令。

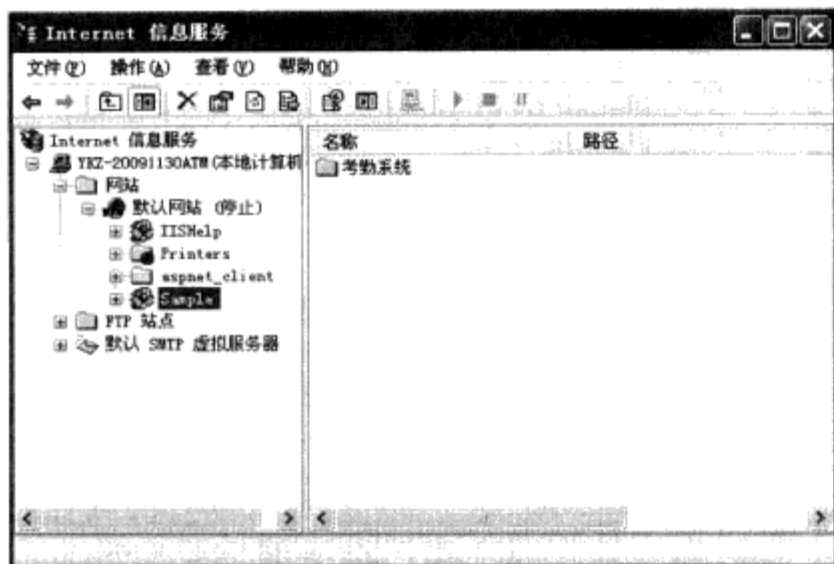


图 1-37 Internet 信息服务

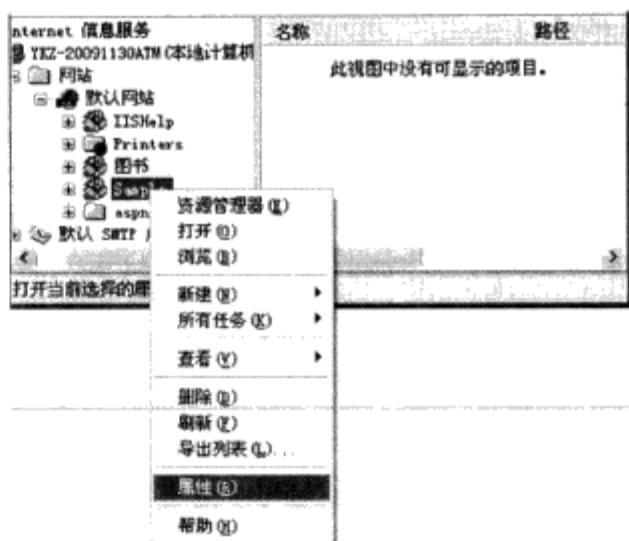


图 1-38 选择“属性”命令

10 弹出如图 1-39 的“Sample 属性”对话框，选择“虚拟目录”选项卡，并设置连接到 Web 站点的内容来源，默认为“此计算机上的目录”，对此目录的默认权限只有“读取”、“记录访问”和“索引资源”3 项，如果没有特殊的要求，此选项卡中的内容不需要改动。

11 选择“文档”选项卡，并选中“启动默认文档”复选框，这样当运行 Web 程序后，不需要在地址栏中填写此文件名，系统会默认读取默认文档文本框中的文件。用户可以添加和删除默认文档，如图 1-40 所示。

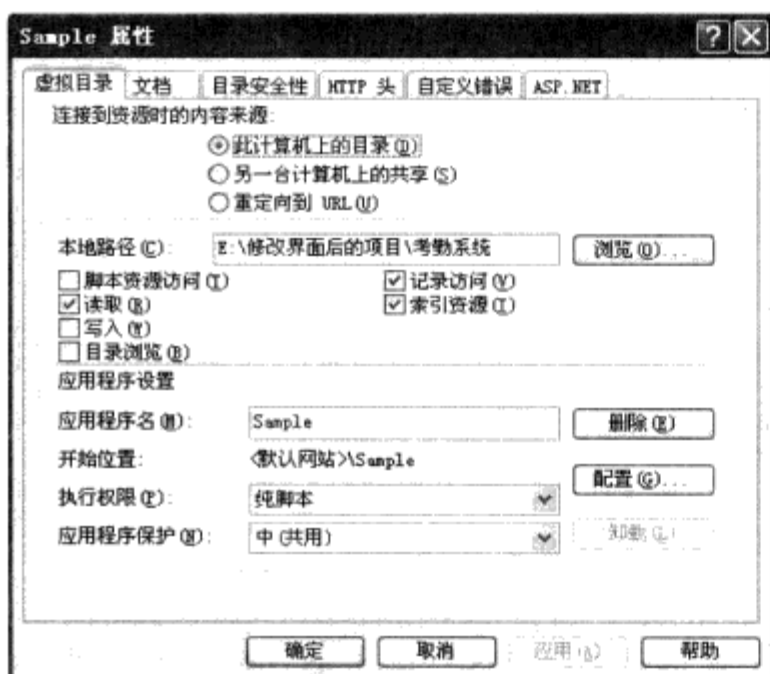


图 1-39 “虚拟目录”选项卡

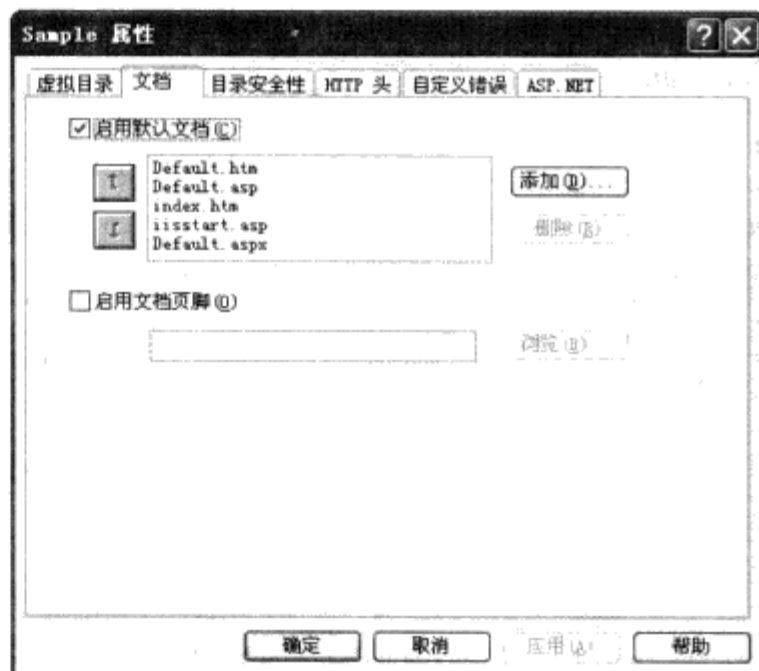


图 1-40 “文档”选项卡

12 选择如图 1-41 所示的“目录安全性”选项卡，设置目录安全性。共有 3 种方法可以控制目录的安全性，分别为“身份验证和访问控制”、“IP 地址和域名限制”以及“安全通信”，通过这 3 种方法可以有效地控制目录的安全性。

13 选择“ASP.NET”选项卡，如图 1-42 所示，设置用户使用的 ASP.NET 版本，这里设置为“2.0.50727”，最后单击“确定”按钮，完成所有 Web 服务器的设置。

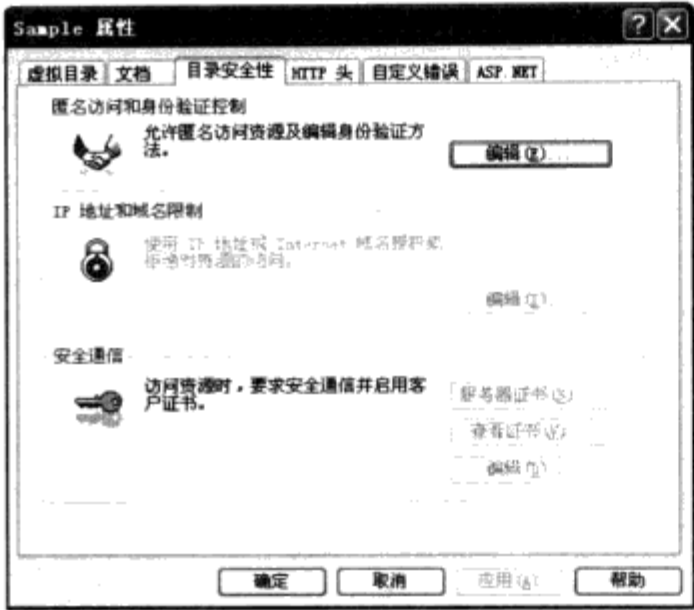


图 1-41 “目录安全性”选项卡



图 1-42 “ASP.NET”选项卡



提示

一旦安装完成，系统会自动启动 IIS，而且在此之后，无论何时启动 Windows，系统都会自动启动 IIS。因此，用户不需要运行启动程序，也不需要像启动 Word 等程序那样通过单击快捷方式启动。

1.6 配置 ASP.NET 4.0 应用程序

在 ASP.NET 4.0 应用程序中，可以在系统提供的配置文件 Web.config 中对该应用程序进行配置，可以配置的信息包括错误信息显示方式、会话存储方式和安全设置等。Web.config 文件是一个 XML 文本文件，它用来储存 ASP.NET Web 应用程序的配置信息（如最常用的设置 ASP.NET Web 应用程序的身份验证方式等），它可以出现在应用程序的每一个目录中。

当读者通过 ASP.NET 4.0 新建一个 Web 应用程序后，默认情况下会在根目录下自动创建一个默认的 Web.config 文件。由于 ASP.NET 4.0 的 Machine.config 文件自动注册所有的 ASP.NET 标识、处理器和模块，所以在 Visual Studio 2010 中创建新的空白 ASP.NET 应用项目时，会发现默认的 Web.config 文件既干净又简洁而不像以前的版本有 100 多行代码。如果想修改配置的设置，可以在 Web.config 文件下的 Web.Release.config 文件中进行重新配置。它可以提供重写或修改 Web.config 文件中定义的设置。在运行时对 Web.config 文件的修改不需要重启服务就可以生效（注：<processModel>节例外）。当然 Web.config 文件是可以扩展的。我们可以自定义新配置参数并编写配置节处理程序以对它们进行处理。Web.config 配置文件的所有代码都应该位于 <configuration><system.web>和</system.web></configuration>之间。下面介绍一下常用的配置节。

1. <authentication>节

<authentication>节通常用来配置 ASP.NET 身份验证支持（参数可以使用 Windows、Forms、PassPort、None 四种）。该元素只能在计算机、站点或应用程序级别声明。<authentication>元素必需与<authorization>节配合使用。

例如：基于窗体的身份验证站点的配置，代码如下。

```

1. <authentication mode="Forms">
2.   <forms loginUrl="Login.aspx" name=".ASPXAUTH"/>
3. </authentication>

```

代码说明：第 1 行和第 3 行定义<authentication>节，把 Mode 属性设置为 Forms，表示这个站点将执行基于窗体的身份验证，第 2 行定义当没有登陆身份的用户访问页面时自动跳转到的页面，其中元素 loginUrl 表示登陆网页的名称，name 表示 Cookie 名称。

2. <authorization> 节

<authorization>节通常用来控制对 URL 资源的客户端访问，如允许匿名用户访问。此元素可以在任何级别（计算机、站点、应用程序、子目录或页）上声明。必需与<authentication> 节配合使用。可以使用 user.identity.name 来获取已经过验证的当前用户名；也可以使用 web.Security.FormsAuthentication.RedirectFromLoginPage 方法将已验证的用户重定向到用户刚才请求的页面。

例如：禁止匿名用户访问的站点的配置，代码如下。

```

1. <authorization>
2.   <deny users="?" />
3. </authorization>

```

代码说明：第 1 行和第 3 行定义<authorization>节，第 2 行通过设置<deny users="?" />来实现任何来访的用户都需要身份认证。

3. <compilation> 节

<compilation>节通常用来配置 ASP.NET 使用的所有编译设置。默认的 debug 属性为“True”。在程序编译完成交付使用之后应将其设为“false”。

4. <customErrors> 节

<customErrors>节通常用来为 ASP.NET 应用程序提供有关自定义错误信息，但它不适用于 XML Web services 中发生的错误。

例如：当发生错误时，将网页跳转到自定义的错误页面的配置，代码如下。

```

1. <customErrors defaultRedirect="ErrorPage.aspx" mode="RemoteOnly">
2. </customErrors>

```

代码说明：第 1 行和第 2 行定义<customErrors>节，并通过属性 defaultRedirect 来定义发生错误时跳转的页面是 ErrorPage.aspx。

5. <httpRuntime> 节

<httpRuntime>节通常用来配置 ASP.NET HTTP 运行库设置。该节可以在计算机、站点、应用程序和子目录级别声明。

例如：ASP.NET HTTP 运行库设置代码如下。

```
<httpRuntime maxRequestLength="2048" executionTimeout="100" appRequestQueueLimit="50"/>
```

代码说明：这段代码的含义是控制读者上传文件最大为 2M，最长时间为 100 秒，最多请求数为 50。

6. <pages> 节

<pages>节通常用来标识特定于页的配置设置（如是否启用会话状态、视图状态，是否检测读者的输入等）。<pages>节可以在计算机、站点、应用程序和子目录级别声明。

例如：检测用户在浏览器输入的内容中是否存在潜在的危险数据的代码如下。

```
<pages buffer="true" enableViewStateMac="true" validateRequest="false"/>
```

代码说明：buffer="true"定义了页面发送前先缓冲输出。enableViewStateMac="true"表示在从客户端回发页时将检查加密的视图状态，以验证视图状态是否已在客户端被篡改。validateRequest="false"表示 ASP.NET 检查从浏览器输入的所有数据，以找出潜在的危险数据。

7. <sessionState> 节

<sessionState>节通常用来为当前应用程序配置会话状态设置（如设置是否启用会话状态，会话状态保存位置）。

例如：设置会话状态代码如下。

```
1. <sessionState mode="InProc" cookieless="true" timeout="60"/>
2. </sessionState>
```

代码说明：第 1 行和第 2 行用来设置会话状态，其中 mode="InProc"表示在本地储存会话状态用户也可以选择储存在远程服务器或 SAL 服务器中或不启用会话状态。cookieless="true"表示如果读者浏览器不支持 Cookie 时启用会话状态（默认为 False）timeout="60"表示会话可以处于空闲状态的分钟数。

8. <trace> 节

<trace>节通常用来配置 ASP.NET 跟踪服务，主要用来判断程序哪里出错。

例如：Web.config 中的对跟踪服务的默认配置，代码如下。

```
<trace enabled="false" requestLimit="15" pageOutput="false" traceMode="SortByTime" localOnly="true" />
```

代码说明：这行代码用来设置跟踪服务的，其中 enabled="false"表示不启用跟踪；requestLimit="15"表示指定在服务器上存储的跟踪请求的数目；pageOutput="false"表示只能通过跟踪实用工具访问跟踪输出；traceMode="SortByTime"表示以处理跟踪的顺序来显示跟踪信息；localOnly="true"表示跟踪查看器只用于宿主 Web 服务器。



提示

ASP.NET 4.0 配置系统有两类配置文件：应用程序配置和服务器配置。服务器配置信息存储在 machine.config 文件中，这个文件描述了所有 ASP.NET 4.0 应用程序所用的默认配置。应用程序配置信息存储在 web.config 文件中，该配置文件描述了一个 ASP.NET 4.0 应用程序的设置信息。

1.7 上机练习

下面通过本书的第一个 Web 应用程序来介绍创建 ASP.NET 4.0 应用程序的过程，本练习将实现在页面显示“我的第一个 ASP.NET 4.0 网页”，练习步骤如下。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，弹出如图 1-43 所示的新建项目对话框，打开“Visual C#”类型节点，选择“Web”子节点这个模板，同时在右边窗口选择“ASP.NET 空 Web 应用程序”。在“名称”文本框中输入“上机练习”，并在“位置”文本框中输入相应的存储路径，在“解决方案名称”文本框中输入“上机练习”。最后，单击“确定”按钮。

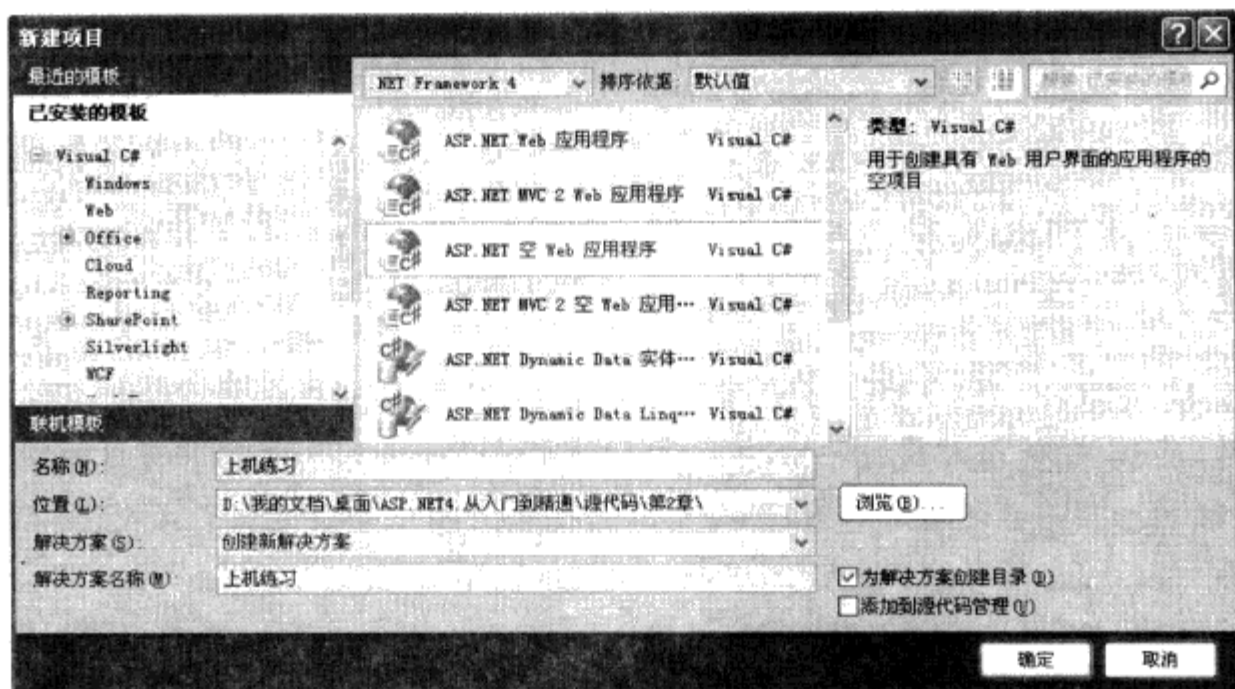


图 1-43 新建项目对话框

02 这时，在解决方案管理器中的网站根目录下会生成一个“上机练习”的 Web 项目。右击单击项目名称“上机练习”，在弹出的菜单中选择“添加”|“新建项”命令。在弹出的“添加新项”对话框中选择“已安装模板”下的“Web”模板，并在模板文件列表中选中“Web 窗体”，然后在“名称”文本框输入该文件的名称“Default.aspx”，最后单击“添加”按钮。

此时上机练习下面会生成一个如图 1-44 所示的“Default.aspx”页面，它包括两个文件，一个是“Default.aspx.cs”文件用于编写后台代码。另一个是“Default.aspx.designer.cs”存放的是一些页面控件中控件的配置信息。

03 用鼠标双击网站的根目录下的“Default.aspx”文件，进入如图 1-45 所示的“视图设计器”。从“工具箱”拖动一个“Label 控件”到设计视图中。

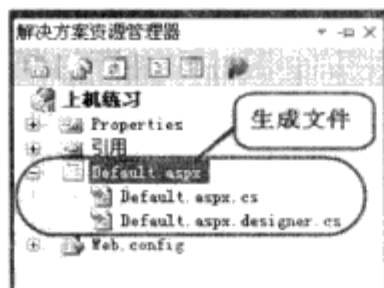


图 1-44 生成 Default.aspx 页面

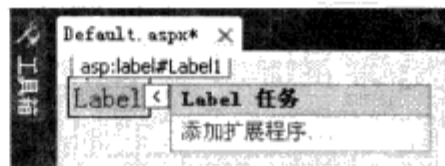


图 1-45 设计视图

04 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```
1.  protected void Page_Load (object sender, EventArgs e){  
2.      Label1.Text="我的第一个 ASP.NET 4.0 网页！";  
3.  }
```

代码说明：第 1 行处理页面 Page 的加载事件 Load，第 2 行设置 Label1 控件的文本显示“我的第一个 ASP.NET 4.0 网页！”。

05 按下“Ctrl+F5”，运行程序的效果如图 1-46 所示。

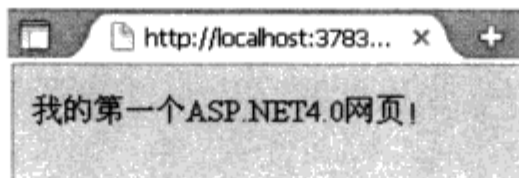


图 1-46 运行结果

1.8 上机题

1. 使用记事本编写一个 HTML 网页，运行后在网页中显示“我的第一个静态网页”。
2. 在本地电脑中安装和配置 IIS (Internet Information Server) 服务器并进行相应的配置。
3. 为了给本书后面的学习做好准备，参考本章的相关内容在本地电脑中安装 Visual Studio 2010 开发环境。
4. 在安装好的 Visual Studio 2010 开发环境中创建一个 Web 站点，并通过浏览器访问，运行后，在页面中显示“欢迎来到 ASP.NET 4.0 的世界”。

第 2 章 C# 语言基础

学习目标

C# 语言是微软公司设计的一种编程语言，它继承了 C/C++ 优良传统，又借鉴了 Java 的很多特点，是专门为 .NET 平台而创建的。目前，随 ASP.NET 4.0 一起发布的是 C# 4.0 的版本。本章从 C# 语言的基础入手，着重介绍了 C# 语言的各种语法知识和用途，力求从整体上引导读者有对该门编程语言有一个总体的掌握。特别是对以前从没有接触过编程语言的读者，建议一定要认真阅读和学习本章内容，以便能够顺利地学习后面的章节内容。

本章重点

- 熟悉 C# 中常用的数据类型
- 学会在程序中定义变量和常量
- 熟练使用 C# 中控制语句
- 重点掌握面向对象编程的知识，包括类的封装、继承和事件
- 了解 C# 4.0 的新特性

2.1 C# 语言概述

C# 是一种源于 C 和 C++ 语言之上的、简单的、现代的、面向对象和类型安全的编程语言。其设计目标是要把 Visual Basic 高速开发应用程序的能力和 C++ 本身的强大功能结合起来。C# 作为一种优秀的编程语言，可以用来开发控制台应用程序、.NET Windows 应用程序、ASP.NET 应用程序以及 Web 服务等各种类型的应用程序。在实际应用中，我们可以使用像记事本那样的编辑器来编写代码，同样也可以使用开发工具如 Visual Studio 2010 来开发 C# 代码。

C# 语言具有以下一些主要优点。

1. 语法简单

由于源于 C 和 C++，因此这三者在语法风格上基本一致。同时它又抛弃了 C 和 C++ 中一些晦涩不清的表达。在默认情况下，C# 的代码在 .NET 框架提供的可操作环境中运行，不允许直接操作内存。它的最大特色是没有 C 和 C++ 的指针操作。另外，使用 C# 创建应用程序，不必记住复杂的基于不同处理器架构的隐含类型，包括各种类型的变化范围，这样大大的降低了 C# 语言的复杂性。

2. 完全的面向对象

C# 语言具有面向对象语言所应有的一切特性，包括封装、继承和多态。同时，在 C# 类型系

统中，每种类型都可以看作一个对象。因此，任何值类型、引用类型和 `Object` 类型之间都可以进行相互转换。

3. 消除了大量的程序错误

C# 的现代化设计能够消除很多常见的 C++ 编译错误。例如：C# 的资源回收功能减轻了内存管理的负担，变量由环境自动初始化，变量是类型安全的等等。这样，使用 C# 语言编写和维护那些复杂的应用程序就变得很方便。

4. 与 Web 开发紧密结合

C# 可以在 .NET 平台上轻松地构造 Web 应用程序的扩展框架。C# 语言包含了内置的特性，使任何组件可以转换为 XML 网络服务，从而通过 Internet 被任何操作系统上运行的组件调用。更为重要的是，XML 网络服务框架可以使处理现有的 XML 网络服务就像处理 C# 对象一样的简单。此外，为了提高性能，C# 还允许将 XML 数据直接映射为 Struct 数据类型。

2.1.1 第一个 C# 程序

由于 C# 语言的各种优点，在代码的编写过程中，不需要花费太大的力气就可以编出可读性很强的代码。下面通过一个入门实例开始 C# 语言的学习。

1. 编写 C# 源代码

我们可以选择在 Visual Studio 2010 中编写应用程序，也可以在记事本中输入代码。为了让大家更深入地了解 C# 语言，这里采用记事本编写控制台应用程序。

【例 2-1】使用记事本编写“Hello World”控制台应用程序。

01 选择“开始”|“所有程序”|“附件”|“记事本”命令，打开记事本编辑界面。

02 在记事本中输入以下代码：

```
1. using System;
2. class HelloWorld{
3.     public static void Main (){
4.         Console.WriteLine("Hello World! "); //打印出 Hello World
5.     }
6. }
```

03 将文件保存为 Sample1.cs。保存文件的路径是 D:/Sample/。

2. 配置 C# 控制台编译环境

C# 源程序需要 .NET Framework SDK 安装程序提供的 C# 编译器 `csc.exe` 来编译。为了能够编译 C# 程序，需要设置系统环境变量，步骤如下。

01 右键单击桌面上“我的电脑”图标，在弹出的快捷菜单中选择“属性”命令，打开如图 2-1 所示的“系统属性”对话框，并选择“高级”选项卡，然后单击“环境变量”按钮。

02 在弹出的如图 2-2 所示的“环境变量”对话框中的“系统变量”列表框中选择“Path”项，单击“编辑”按钮。

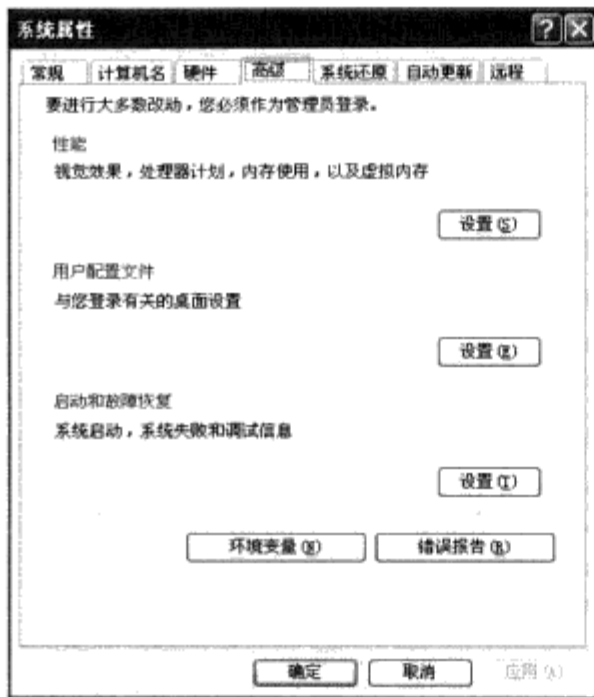


图 2-1 系统属性对话框

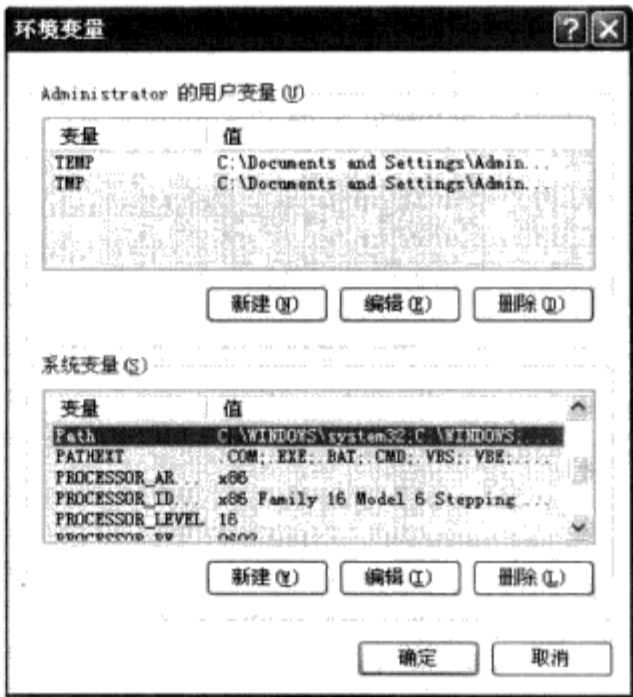


图 2-2 环境变量对话框

03 在如图 2-3 所示的“编辑系统变量”对话框中，将 .NET Framework SDK 安装程序路径添加到“变量值”后面的文本框中。这里输入笔者存放程序的路径位置：“C:\WINDOWS\Microsoft.NET\Framework\v4.0.50727”。最后单击“确定”按钮，退出“环境变量”对话框。

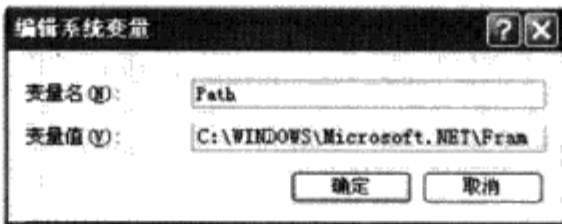


图 2-3 “编辑系统变量”对话框

3. 编译执行程序

在设置好 C# 控制台应用程序的编译环境之后，就可以对刚才编写的“Hello World”程序进行编译了，步骤如下。

- 01** 执行“开始”|“运行”命令，在如图 2-4 所示的“运行”对话框中的“打开”文本框中输入“cmd”后单击“确定”按钮。
- 02** 在弹出如图 2-5 所示的命令行窗口中输入“D:/Sample/”，进入 Sample.cs 文件所在的路径。然后再输入“csc Sample.cs”，最后按“Enter”键。
- 03** 此时会在 cs 文件的同一目录下生成一个后缀为 .exe 同名的可执行文件“Sample.exe”。在如图 2-6 所示的命令行窗口中输入“Sample.exe”，按“Enter”键就会显示输出结果“Hello World”。



图 2-4 运行对话框

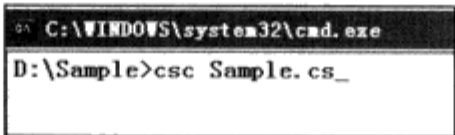


图 2-5 编译程序

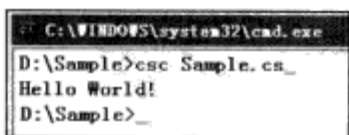


图 2-6 运行结果

至此，我们就完成了第一个 C# 控制台程序。

2.1.2 C# 代码结构

C# 的代码和其他语言一样有其固有的代码结构，在编写 C# 代码时必须遵循这些规则。下面通过分析前文的第一个控制台程序 `Sample` 来了解这些结构。

1. 命名空间和类

.NET 框架提供了许多的类，以便让 .NET 程序语言使用这些类的功能。这些类根据功能划分了许多的命名空间。.NET 框架有一个 `System` 命名空间，常用的类都在这个命名空间下。比如 `Sample` 程序的第一行是通知 C# 编译器使用 `System` 命名空间中的类，代码如下。

```
using System;
```

代码说明：通过使用关键字“`using`”来引用 `System` 命名空间，以便在下面的程序代码中能够直接使用各种类，这个例子使用了 `Console` 类来操作控制台程序的输入和输出。

每个 C# 程序都是由很多的类、结构和数据类型组成的集合。我们还可以使用“`namespace`”关键字来声明自己的命名空间，声明命名空间的语法如下：

```
Namespace 命名空间名称  
{  
    //命名空间的声明  
}
```

关键字“`class`”用来声明类，在 `Samples` 的例子中，声明了一个名为“`Hello World`”的类，声明类的语法如下：

```
Class 类名  
{  
    //类的声明  
}
```

在 C# 中，所有的应用程序都必须包装在一个类中，类中包含程序所需的变量与方法的定义。

2. Main 方法

每个应用程序都有一个且只有一个 `Main` 方法，该方法定义了这个类的行为或者是该类的功能，它是程序的入口。`Main` 方法定义的语法如下：

```
public static void Main()  
{  
    // Main 方法中的代码  
}
```

代码说明：“`public`”关键字表示所有的程序都可以访问 `Main` 方法。“`static`”关键字代表 `Main` 方法为整个程序运行期间都有效的方法，而且在调用这个方法之前不必对该类进行实例化。`Main` 方法的返回值除了 `void` 之外也可以是 `int` 类型。`Main` 方法也可以带参数，比如下面的代码。

```
public static void Main(String[] args)  
{
```

```

    Console.WriteLine("Hello World! ");
}

```

以上的代码和 Sample 例子中的 Main 方法唯一的区别就是带了字符串数组 String[] 类型的参数 args。这里特别要强调 Main 方法的第一个字母 M 必须大写。

3. 语句块

在 C# 程序中,把使用符号“{”和“}”包含起来的程序称为语句块。语句块在条件和循环语句中经常会用到,主要是把重复使用的程序语句放在一起以方便使用,这样有助于程序的结构化。上面 Sample 例子中的 Main 方法的代码就是一个语句块。以下这段代码是求 100 以内所有偶数的和,我们来看一下它的语句块结构,代码如下:

```

1.  int sum = 0;
2.  for(int i = 1; i <= 100; i++){
3.      if(i % 2 == 0){
4.          sum = sum + i;
5.      }
6.  }

```

代码说明:第 2~6 行使用了二组“{”和“}”符号形成的不同语句块,也就是实现了语句块的嵌套。

4. 语句终止符

每一句 C# 程序都要以语句终止符来结束,C# 的语句终止符是“;”分号,例如代码:

```
string name;
```

代码说明:使用了语句终止符“;”,结束变量的定义。

在 C# 程序中,可以在一行中写多个语句,但每个语句都要以“;”结束,也可以在多行中写一个语句,但要在最后一行中以“;”结束,例如代码:

```

1.  int number; string name; float shuzi;
2.  int number = 1, number1 = 2, number2 = 3, number3 = 4, sum;
3.  sum = number + number1 +
4.  number2 + number3;

```

代码说明:第 1 行中包含有多个语句,语句之间使用终止符是“;”进行分割。第 3 和 4 行将一句代码写在多个行之中。

5. 注释

注释在一个开发语言中也是非常重要的。C# 提供了两种注释的类型。

第一种是单行注释,注释符号是“//”,例如代码:

```
int one; //一个整型变量,存储整数
```

代码说明:使用了单行注释符号“//”,符号后面是注释的具体内容。

第二种是多行注释,注释符号是“/*”和“*/”,任何在符号“/*”和“*/”之间的内容都会被编译器忽略,例如代码:

```

1.  /*一个整型变量
2.  存储整数*/
3.  int one;

```

代码说明：第 1 和第 2 行使用了多行注释“/*”和“*/”符号，符号之间的是注释内容。此外 XML 注释符号“///”也可以用来对 C# 程序进行注释，例如代码：

```

1.  ///一个整型变量
2.  ///存储整数
3.  int one;

```

代码说明：第 1 和第 2 行使用了 XML 注释符号“///”，符号后面的是具体的注释内容。

6. 大小写的区别

C# 是一种对大小写敏感的语言。在 C# 程序中，同名的大写和小写代表不同的对象，因此在输入关键字、变量和函数时必须使用适当的字符。

此外，C# 对小写比较偏好，它的关键字基本上都采用小写，例如 if、for、while 等。

在定义变量时，C# 程序员一般都遵守这样的规范：对于私有变量的定义一般都以小写字母开头、而公共变量的定义则以大写字母开头，例如：以 name 来定义一个私有变量、而以 Name 来定义一个公共变量。

2.2 基本语法

C# 语言自问世以来就得到了业界的高度关注，它的许多优点使编程人员越来越多地在开发中选择使用它。要熟练掌握 C# 的运用必须从最基本的语法学起。

2.2.1 数据类型

C# 中数据类型可以分为值类型和引用类型，值类型也称为数值类型，其中包含枚举类型（Enum Types）和结构类型（Struct Types）；引用类型包含类类型（Class Types）、对象类型（Object Types）、字符串类型（String Types）、数组类型（Array Types）、接口类型（Interface Types）和代理类型（Delegate Types）等。

1. 值类型

值类型主要由结构和枚举组成，其中结构又可以分为：数值类型、布尔类型和用户定义的结构。下面具体来介绍值类型。

（1）数值类型

数值类型主要包括整数、浮点数和小数，这些均属于简单类型。简单类型都是 .NET Framework 中定义的标准类的别名，都隐式地从类 object 继承而来。所有类型都隐含地声明了一个公共的无参数的构造函数，称为默认构造函数。默认构造函数返回一个初始值为零的实例。

① 整数类型

整数类型可以分为无符号型、有符号型和 char，其中无符号型包括：byte, ushort, uint 和 ulong；

有符号型包括：sbyte, short, int 和 long。char 在 C# 中表示 16 位 Unicode 字符。

无符号型说明如下。

- byte 类型：对应于 .NET Framework 中定义的 System.Byte 类，其大小为 1 个字节，取值范围从 0 到 255。
- ushort 类型：对应于 .NET Framework 中定义的 System.UInt16 类，其大小为 2 个字节，取值范围从 0 到 65,535。
- uint 类型：对应于 .NET Framework 中定义的 System.UInt32 类，其大小为 4 个字节，取值范围从 0 到 4,294,967,295。
- ulong 类型：对应于 .NET Framework 中定义的 System.UInt64 类，其大小为 8 个字节，取值范围从 0 到 18,446,744,073,709,551,615。

有符号型说明如下。

- sbyte 类型：对应于 .NET Framework 中定义的 System.SByte 类，其大小为 1 个字节，取值范围从 -128 到 127。
- short 类型：对应于 .NET Framework 中定义的 System.Int16 类，其大小为 2 个字节，取值范围从 -32,768 到 32,767。
- int 类型：对应于 .NET Framework 中定义的 System.Int32 类，其大小为 4 个字节，取值范围从 -2,147,483,648 到 2,147,483,647。
- long 类型：对应于 .NET Framework 中定义的 System.Int64 类，其大小为 8 个字节，取值范围从 -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807。

② 浮点数类型

在 C# 中有两种浮点类型：单精度浮点（float）类型和双精度浮点（double）类型。

- 单精度浮点类型对应于 .NET Framework 中定义的 System.Single 类，其大小为 4 个字节，取值范围为 1.5×10^{-45} 到 3.4×10^{38} ，有 7 位数字位精度。
- 双精度浮点类型对应于 .NET Framework 中定义的 System.Double 类，其大小为 8 个字节，取值范围为 5.0×10^{-324} 到 1.7×10^{308} ，有 15 到 16 位数字位精度。

浮点类型支持以下几种数值。

- 正零和负零：在大多数情况下，正零和负零与简单的零值相同，但是它们的使用中间有一些区别。如果浮点数操作的结果对于目标形式来说太小，操作的结果就会转换为正零或负零。
- 正无穷大：表示正的无穷大数，例如， $5.0 / 0.0$ 会产生正无穷大值。如果浮点数操作的正数结果对于目标形式来说太大，操作的结果就会转换为正无穷大。
- 负无穷大：表示负的无穷大数，例如， $-5.0 / 0.0$ 会产生负无穷大值。如果浮点数操作的负数结果对于目标形式来说太大，操作的结果就会转换为负无穷大。
- NaN (Not a Number)：NaN 是非数字数据，表示无效的浮点数操作，例如零除以零就会得到 NaN 值。如果浮点数的操作是无效的，操作的结果就会转换为 NaN。如果一个或所有

浮点操作的操作数都是 NaN，那么操作的结果就变为 NaN。

如果二元运算符的一个操作数是浮点类型，那么其它操作数必须是整数类型或者是浮点数类型，并且操作按下面规则进行。

- 如果一个操作数是 double 类型，那么其他操作数就要转换为 double，操作就要按照 double 类型的范围和精度来进行，而且计算的结果也是 double 类型。
- 在没有 double 类型时，如果一个操作数是 float 类型，那么其他操作数就要转换为 float，操作就要按照 float 类型的范围和精度来进行，而且计算的结果也是 float 类型。

对于 float 类型的数值，末尾需要使用 f 说明该数值为 float 类型；对于 double 类型的数值，末尾需要使用 d 说明该数值为 double 类型。如果没有这些说明，系统会把这些小数作为 double 类型处理。下面是浮点类型的使用示例。

```
1. float a = 5.0f;
2. double b = 5.0d;
3. double c = a + b;
```

代码说明：第 1 行定义了一个 float 类型的变量 a，第 2 行定义了一个 double 类型的变量 b，第 3 行把 a 和 b 的值相加，把结果赋给 c，在进行加法运算时，a 自动转换为 double 类型。

③ 小数类型

小数 (decimal) 类型在所有数值类型中精度是最高的，它有 128 位，一般做精度要求高的金融和货币的计算。decimal 类型对应于 .NET Framework 中定义的 System.Decimal 类。取值范围大约为 1.0×10^{-28} 到 7.9×10^{28} ，有 28~29 位的有效数字。decimal 类型的赋值和定义如下所示。

```
decimal d = 8.8m;
```

代码说明：末尾 m 代表该数值为 decimal 类型，如果没有 m 将被编译器默认为 double 类型的 8.8。

decimal 类型不支持有符号零、无穷大和 NaN。一个十进制数由 96 位整数和十位幂表示。



提示

一般不要把 decimal 类型和浮点类型之间进行类型转换（无论是隐式的或者是显式的），因为 decimal 类型比浮点类型有更高的精度但是有更小的范围，从浮点数类型转换到 decimal 类型也许会产生溢出的异常，并且从 decimal 类型转换到浮点数类型也许会有精度损失。

(2) 布尔类型

布尔类型 (bool) 表示布尔逻辑值，对应于 .NET Framework 中定义的 System.Boolean 类。布尔类型的可能值为 true 和 false（仅有 true 和 false 两个布尔值），其中 true 表示逻辑真，false 表示逻辑假。可以直接将 true 或 false 值赋给一个布尔变量，或将一个逻辑判断语句的结果赋给布尔类型的变量，如下面代码所示。

```
1. bool b = false;
```

```
2. bool isSmall = 500>700;
```

代码说明：第2行的语句中，首先计算逻辑判断语句 $500>700$ 的值，其值为 `false`，然后再把该值赋值给变量 `isSmall`。

与 C/C++ 不同，布尔类型不能和其他类型进行转换，布尔数据不能用于使用整数类型的地方，反之亦然。这是因为零整数值或空指针不可以直接被转换为布尔数值 `false`，而非零整数值或非空指针也可以直接转换为布尔数值 `true`。在 C# 中，布尔类型的变量不能由其他类型的变量代替，但是可以通过一个转换变为布尔类型。

在 C/C++ 中，我们经常使用不等于 0 的整数表示 `true`，使用 0 表示 `false`。

(3) 自定义类型

在不会引起歧义的情况下，用户自定义结构常常简称为结构。结构类型通常是一组相关的信息组合成的单一实体。其中的每个信息称为它的一个成员。结构类型可以用来声明构造函数、常数、字段、方法、属性、索引、操作符和嵌套类型。结构类型通常用于表示较为简单或者较少的数据，其实际应用的意义在于使用结构类型可以节省使用类的内存的占用，因为结构类型没有如同类对象所需的大量额外的引用。下面代码定义了一个简单数据结构 `Animal`。

```
1. struct Animal{
2.     public uint id
3.     public string name;
4.     public string gender;
5.     public uint age;
6. }
```

代码说明：第1行使用关键字 `struct` 指明这里将要定义一个用户自定义结构，对于一个记录动物信息的结构，编号、名称、性别、年龄是必不可少的，这里在第2~5行定义了这些信息。在使用这个结构时，可以根据需要增加相关的信息。

(4) 枚举类型

枚举类型 (`enum`) 是由一组特定的常量构成一种数据结构，系统把相同类型、表达固定含义的一组数据作为一个集合放到一起形成新的数据类型，比如一年分为十二个月可以放到一起作为新的数据类型来描述月份，代码如下所示。

```
1. enum Month {
2.     January,           //一月份
3.     February,          //二月份
4.     March,             //三月份
5.     April,             //四月份
6.     May,               //五月份
7.     June,              //六月份
8.     July,              //七月份
9.     August,            //八月份
10.    September,         //九月份
11.    October,           //十月份
12.    November,          //十一月份
13.    December           //十二月份
14. };
```

代码说明：第1行中 `enum` 是定义枚举类型的关键字，`Month` 的枚举类型的名字，`{}` 中的是枚

举元素，第 2~13 行定义了枚举元素，用逗号分隔。这样一年十二个月份的集合就构成了一个枚举类型，它们都是枚举类型的组成元素。

枚举元素实际上都是整数类型，缺省时第一个枚举元素值为 0，以后每个元素递增 1。开发者也可以自定义首元素的值，甚至每个元素的值，例如下面的代码就自定义了元素的值。

```
1.  enum Month {  
2.      January=1,           //一月份  
3.      February=2,          //二月份  
4.      March=3,             //三月份  
5.      April=4,             //四月份  
6.      May=5,               //五月份  
7.      June=6,              //六月份  
8.      July=7,              //七月份  
9.      August=8,            //八月份  
10.     September=9,         //九月份  
11.     October=10,          //十月份  
12.     November=11,         //十一月份  
13.     December=12          //十二月份  
14. }
```

代码说明：这段代码对月份分别进行了赋值，更符合中国人的文字习惯。

2. 引用类型

引用类型的变量又称为对象，可存储对实际数据的引用。如前所述，引用类型包括字符串、数组、类和对象、接口、代理等，本节介绍字符串和数组，其余类型在后面章节介绍。

(1) 字符串

字符串实际上是 Unicode 字符的连续集合，通常用于表示文本，而 `string` 是表示字符串的 `System.Char` 对象的连续集合。在 C# 中提供了对字符串（`string`）类型的强大支持，可以对字符串进行各种的操作。`string` 类型对应于 .NET Framework 中定义的 `System.String` 类，`System.String` 类是直接从 `object` 派生的，并且是 `final` 类，不能从它再派生其他类。

字符串值使用双引号表示，例如“Welcome!”、“欢迎!”等，而字符型使用单引号表示，这点读者需要注意区分。下面是几个关于字符串操作的例子代码。

```
1.  string str1="Welcome";  
2.  string str2="to ShangHai! ";  
3.  string str3=str1+str2;  
4.  char number = str[3];
```

代码说明：第 1 行和第 2 行直接把字符串赋值给字符串变量 `str1` 和 `str2`，第 3 行语句把两个字符串进行了合并，字符串变量 `str3` 的值最后为“Welcome to ShangHai!”。第 4 行用于获得字符串中某个字符值，字符串中第一个字符的位置是 0，第二个字符的位置是 1，依此类推。这里 `number` 的值是第 4 个字符“c”。

(2) 数组

数组是包含若干个相同类型数据的集合，数组的数据类型可以是任何类型。数组可以是一维的，也可以是多维的，常用的是二维和三维数组。

① 一维数组

数组的维数决定了相关数组元素的下标数，一维数组只有一个下标。一维数组的声明方式如下：

```
数组类型[] 数组名;
```

其中，“数组类型”是数组的基本类型，一个数组只能有一种数据类型。数组的数据类型可以是任何类型，包括前面介绍的枚举和结构类型。[]是必须的，否则将成为定义变量了。“数组名”定义的数组名字，相当于变量名。

数组声明以后，就可以对数组进行初始化了，数组必须在访问之前初始化。数组是引用类型，所以声明一个数组变量只是为对此数组的引用设置了空间。数组实例的实际创建是通过数组初始化程序实现的，数组的初始化有两种方式：第一种是在声明数组的时候进行初始化；第二种是使用 new 关键字进行初始化。

使用第一种方法初始化是在声明数组的时候，提供一个用逗号分隔开的元素值列表，该列表放在花括号中，例如：

```
int[] Array = {50, 60, 80, 90};
```

以上代码声明了一个整数类型的数组 Array，其中共有 4 个元素，每个元素都是整数值。

第二种是使用关键字“new”为数组申请一块内存空间，然后直接初始化数组的所有元素。例如：

```
int[] Array = new int[4]{ 50, 60, 80, 90};
```

以上代码通过“new”关键字声明了一个整数类型的数组 Array，其中有 4 个元素，每个元素都是整数值。



使用 new 关键字初始化数组时，数组大小必须与元素个数相匹配，如果定义的元素数和初始化的元素数不同会出现编译错误。

数组中的所有元素值都可以通过数组名和下标来访问，数组名后面的方括号中指定下标（指定要访问的第几个元素），就可以访问该数组中的各个成员。数组的第一个元素的下标是 0，第二个元素的下标是 1，依此类推。下面通过一个例子来做进一步的说明。

```
1. int[] Array = {50, 60, 80, 90};
2. Array [3] = 90;
```

上面的代码中，第 1 行定义并初始化了一个有 4 个元素的数组 Array，第 2 行使用 Array [2] 访问该数组的第 4 个元素。

② 多维数组

多维数组和一维数组有很多相似的地方，下面介绍多维数组的声明、初始化和访问方法。多维数组有多个下标，例如二维数组和三维数组声明的语法分别为：

```
1. 数组类型[,] 数组名;
2. 数组类型[,,] 数组名;
```

以上代码第 1 行声明了一个二维数组，第 2 行声明了一个三维数组，区别在于“[]”中逗号的数量。

更多维数的数组声明则需要更多的逗号。多维数组的初始化方法也和一维数组的相似，可以在声明的时候初始化，也可以使用 new 关键字进行初始化。下面的代码声明并初始化了一个 3×2 的二维数组，相当于一个三行两列的矩阵。

```
int[,] Array = { {1, 3}, {2, 4}, {3, 5}};
```

以上代码初始化时，数组的每一行值都使用 {} 括号包括起来，行与行间用逗号分隔。Array 数组的元素安排如表 2-1 所示。

表 2-1 Array 数组的元素

	第 1 列	第 2 列
第 1 行	1	3
第 2 行	2	4
第 3 行	3	5

要访问多维数组中的每个元素，只需指定它们的下标，并用逗号分隔开，例如访问 Array 数组第一行中的第 2 列数组元素（其值为 3）的代码如下所示。

```
mypoint[0,1]
```

另外，C# 中还支持“不规则”的数组，或者称为“数组的数组”，下面的代码就演示了一个不规则数值的声明和初始化过程。

```
1. int[][] Array= new int[8][];  
2. Array [0] = new int[] {2, 4, 6};  
3. Array [1] = new int[] {2, 4, 6, 8,10, 12};  
4. Array [2] = new int[] {2, 4, 6,8, 10, 12,14, 16, 18};
```

在上面代码中，第 1 行定义了一个名为 Array 的数组，它是一个由 int 数组组成的数组，或者说是一个一维 int[] 类型的数组。这些 int[] 变量中的每一个都可以独自被初始化，同时允许数组有一个不规则形状。第 2~4 行代码中每个 int[] 数组定义了不同的长度，其中，第 2 行定义数组的第一个元素，该元素是一个一维数组，长度为 3。第 3 行定义的一维数组的长度为 6，第 4 行定义的一维数组的长度为 9。

【例 2-1】本例通过控制台接受输入的数字放入一维数组，并实现数组进行反转数据处理，然后将反转后的结果显示在控制台屏幕上，实现步骤如下。

01 启动 Visual Studio 2010，执行“文件” | “新建项目”命令，在弹出的图 2-7 所示的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-1”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-1”项目，如图 2-8 所示。

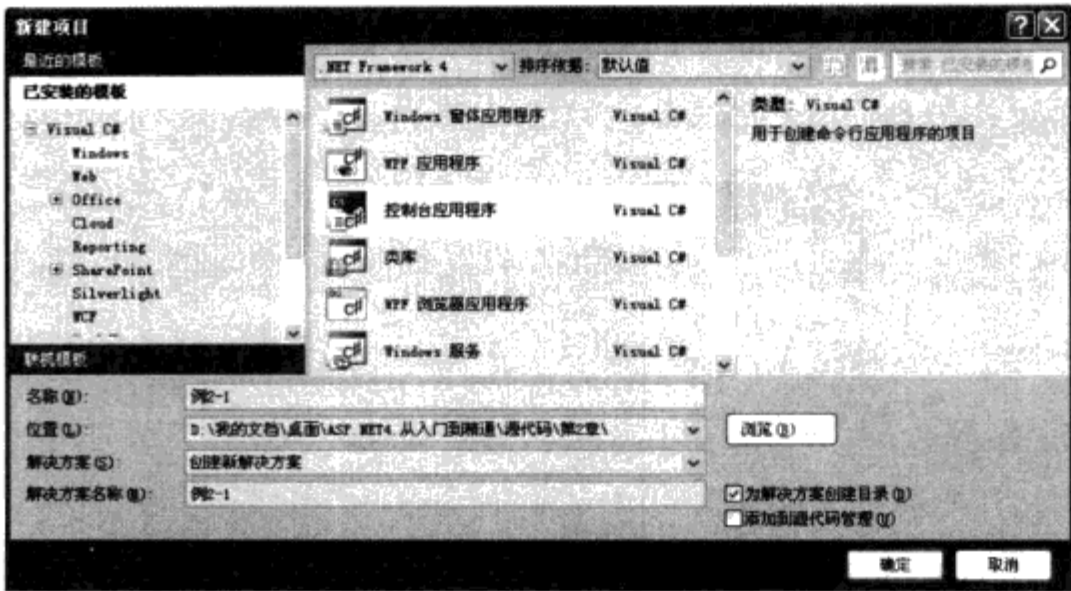


图 2-7 新建项目对话框

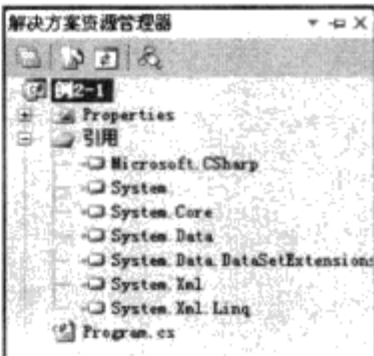


图 2-8 生成项目

03 用鼠标双击网站目录下的“Program.cs”文件，在该文件中加入如下逻辑代码。

```
1.      class Program{
2.          static void Main(string[] args){
3.              int[] oldArray = new int[5] ;
4.              for (int i=1; i < 6; i++){
5.                  Console.WriteLine("请输入第 {0} 个数字", i);
6.                  int number=int.Parse (Console.ReadLine());
7.                  oldArray[i-1]=number;
8.              }
9.              Console.WriteLine("反转前数组元素: ");
10.             foreach (int element in oldArray){
11.                 Console.Write(element + "," );
12.             }
13.             ReverseArray(oldArray);
14.             Console.WriteLine();
15.             Console.WriteLine("反转后数组元素:");
16.             foreach (int element in oldArray){
17.                 Console.Write(element + "," );
18.             }
19.             Console.ReadLine();
20.         }
21.         private static void ReverseArray(int[] array){
22.             int temp;
23.             int count = array.Length;
24.             for (int i = 0; i < count / 2; i++){
25.                 temp = array[count - 1 - i];
26.                 array[count - 1 - i] = array[i];
27.                 array[i] = temp;
28.             }
29.         }
30.     }
```

代码说明：第 1 行定义了一个 Program 类。第 2 行定义了一个 Main 方法，这是程序执行的入口。第 3 行声明了一个包含 5 个元素的 int 类型数组 oldArray。第 4~8 行通过 for 循环接受用户输入的 5 个数字并放入 oldArray 数组中。第 10~12 行通过 foreach 循环在控制台输出数组中元素的值。第 13 行调用 ReverseArray 方法来实现反转数组，将 oldArray 数组作为参数传递到该方法中。第 15~18 行通过 foreach 循环在控制台输出数组反转后元素的值。第 16~29 行定义了一个 ReverseArray

方法来使用 for 循环反转数组，它的参数是一个 int 类型的数组。

04 按下“Ctrl+F5”，程序运行结果如图 2-9 所示。

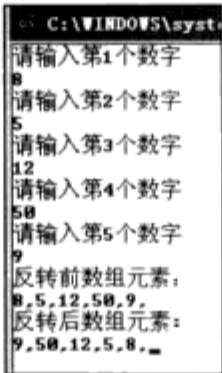


图 2-9 运行结果

3. 装箱和拆箱

装箱和拆箱使值类型能够作为对象使用。对值类型装箱将把该值类型打包到 Object 引用类型的一个实例中，这使得值类型可以存储于垃圾回收堆中。拆箱将从对象中提取值类型，下面的代码分别演示了装箱和拆箱操作。

```

1.  int i = 100;
2.  object o = (object) i; // 装箱
3.  o = 100;
4.  i = (int) o; //拆箱

```

代码说明：第 1 行定义了一个整型变量，第 2 行把该变量打包到 Object 引用类型的一个实例中，也就是进行装箱操作。第 4 行进行拆箱操作。

相对于简单的赋值而言，装箱和取消装箱过程需要进行大量的计算。对值类型进行装箱时，必须分配并构造一个全新的对象。另外拆箱所需的强制转换也需要进行大量的计算。因此，读者在进行装箱和拆箱操作时应该考虑到这两种操作对性能的影响。

2.2.2 变量和常量

在进行程序设计时，我们经常需要保存程序运行的信息，因此在 C# 中引入了“变量”的概念。而在程序中某些值是不能被改变的，这就是所谓的“常量”。

1. 变量

所谓变量，就是在程序运行过程中其值可以被改变的量，变量的类型可以是任何一种 C# 的数据类型。所有值类型的变量具有实际存在于内存中的值，也就是说当将一个值赋给变量执行的是值拷贝操作。变量的定义格式为：

```

变量数据类型 变量名 (标识符);

或者

变量数据类型 变量名 (标识符) = 变量值;

```

上面的第一个定义只是声明了一个变量，并没有对变量进行赋值，此时变量使用默认值。第

一个声明定义变量的同时对变量进行了初始化，变量值应该和变量数据类型一致。下面代码是变量使用的例子。

```
1. int one = 100;
2. double two, three;
3. int four=50, five=20;
4. double six = one + two + three + four + five;
```

代码说明：第1行的代码声明了一个整数类型的变量 `one`，并对其赋值为 100。第2行的代码定义了两个 `double` 类型的变量，当定义多个同类型的变量时，可以把在一行声明，各个变量间使用逗号分隔。第3行代码定义了两个整数类型的变量，并对变量进行了赋值。当定义并初始化多个同类型的变量时，也可以在一行进行，使用逗号分隔。第4行把前面定义的变量相加，然后赋给一个 `double` 类型的变量，在进行求和计算时，`int` 类型的变量会自动转换为 `double` 类型的变量。



提示

变量的使用要注意，有些错误编译器能够帮助程序员找出来，有些错误编译器却不能找出来。所以在开发程序时一定要遵守最基本的规则，从最基本的程序写起，才能做出高质量的程序。

2. 常量

所谓常量，就是在程序的运行过程中其值不能被改变的量。常量的类型也可以是任何一种 C# 的数据类型。常量的定义格式为：

```
const 常量数据类型 常量名(标识符) = 常量值;
```

上面的“`const`”关键字表示声明一个常量，“常量名”就是标识符，用于唯一地标识该常量。常量名要有代表意义，不能过于简练或者复杂。常量和变量的声明都要使用标识符，其命名规则如下。

- 标识符必须以字母开头或者@符号开始。
- 标识符只能由字母、数字、下划线组成，不能包括空格、标点符号、运算符等特殊符号。
- 标识符不能与 C# 中的关键字同名。
- 标识符不能与 C# 中的库函数名相同。

常量值的类型要和常量数据类型一致，如果定义的是字符串型，常量值就应该是字符串类型，否则会发生错误。例如以下代码：

```
1. const double yuanzoulu = 3.1415926;
2. const string word = "Visual Studio 2010";
```

第1行定义了一个 `double` 类型的常量，第2行定义了一个字符串型的常量。一旦用户在后面的代码中试图改变这两个常量的值，则编译器会发现这个错误，导致代码无法编译通过。

3. 隐形局部变量

在前面讲过 C# 是强类型的，在声明变量时必须指明变量的类型，而 JavaScript、VB 等语言则是弱类型的，就是在声明变量时不必指明变量类型，而是通过初始化这个变量的表达式来推导这

个变量的类型，这就是隐型局部变量，这种声明变量的方式比较自由，方便程序开发。C# 3.5 引入了关键字 `var`，也使得在 C# 中也可以在声明变量时不必声明变量类型。下面代码演示了使用 `var` 声明局部变量的各种方式。

```
1. var one = 8;
2. var two = "Number";
3. var three = new[] { 0, 1, 2 };
4. var four = new { Name = "Mary", Age = 28 };
5. var ArrayList = new ArrayList<string>();
```

第 1 行定义了变量 `one` 是一个整数，第 2 行定义的变量 `two` 被当做字符串，第 3 行定义了一个变量 `three`，因为它右边是一个数组，所以该变量为数组变量。第 4 行定义了一个匿名变量，第五行定义了一个 `ArrayList` 类型的变量，`ArrayList` 类型为 .NET Framework 所包含的类型。

在 C# 2.0 及其以前版本中，若定义一个可以向其赋任何值的变量，那么需要先以 `object` 类型来定义该变量，这种方式对值类型的操作方式是装箱和拆箱的过程，这种过程要耗费很多资源。而使用关键字 `var` 来声明变量可以很轻易实现这个功能。

C# 3.5 通过本地类型推断功能，根据表达式对变量的赋值来判断变量的类型，这样可以保护类型安全，而且也可以实现更为自由的编码。依据这个本地类型推断功能，使用 `var` 定义的变量在编译期间就推断出变量的类型，而在编译后的 IL 代码中就会包含推断出的类型，这样可以保证类型安全。

使用隐式类型的变量声明时，需要注意以下限制。

- 只有在同一语句中声明和初始化局部变量时，才能使用 `var`，不能将该变量初始化为 `null`。
- 不能将 `var` 用于类范围的域。
- 由 `var` 声明的变量不能用在初始化表达式中。换句话说，`var v = v++;` 会产生编译时错误。
- 不能在同一语句中初始化多个隐式类型的变量。
- 如果一个名为 `var` 的类型位于范围中，则当读者尝试用 `var` 关键字初始化局部变量时，将收到编译时错误。

2.2.3 运算符

前面介绍了 C# 中的基本数据类型，本小节将介绍如何通过运算符操作变量和常量，例如前面多次用到的赋值运算符“`=`”。运算符是表示各种不同运算的符号，C# 中的运算符非常多，从操作数上划分运算符大致分为 3 类：

- 一元运算符：处理一个操作数，只有少数几个一元运算符。
- 二元运算符：处理两个操作数，大多数运算符都是二元运算符。
- 三元运算符：处理三个操作数，只有一个三元运算符。

从功能上划分，运算符主要分为：算术运算符、赋值运算符、关系运算符、条件运算符、位运算符和逻辑运算符。下面分别进行介绍。

1. 算术运算符

算术运算符主要用于数学计算，主要有加法运算符(+)、减法运算符(-)、乘法运算符(*)、除法运算符(/)、求模运算符(%)、自加运算符(++)和自减运算符(--)。各运算符说明如表 2-2 所示。

表 2-2 算术运算符

运算符	符号	描述
加法运算符	+	加法运算符也可称为正值运算符，其形式为：x+y，如 8+8 等
减法运算符	-	减法运算符也可称为负值运算符，其形式为：x-y，如 8-3 等
乘法运算符	*	其形式为：x*y，如 8*8
除法运算符	/	其形式为：x/y，如 5/8
求模运算符	%	也可称为求余运算符，“%”运算符两边操作数的数据类型必须是整型，如“8%3”的结果为 2
自加运算符	++	其作用是使变量的值自动增加 1，例如 a++，++a
自减运算符	--	其作用是使变量的值自动减少 1，例如 a--，--a

加法运算符、减法运算符、乘法运算符、除法运算符以及模运算符又称为基本的算术运算符，它们都是二元运算符，而自增运算符和自减运算符则是一元运算符。算术运算符通常用于整数类型和浮点类型的计算，比如下面代码。

```
1. int one = 8;
2. int two = 8.08;
3. int three = one + two;
```

第 1 行定义了一个值为 8 的整型变量 one，第 2 行定义了一个整型变量 two，然后把一个小数 8.08 赋给 two，因为 b 的类型为整数，赋值操作会对小数 8.08 进行自动转换，舍去小数部分。第 3 行把 one 和 two 的值相加，然后赋给整型变量 three。

当一个或两个操作数为 string 类型时，二元“+”运算符进行字符串连接运算。如果字符串连接的一个操作数为 null，则用一个空字符串代替。另外，通过调用从基类型 object 继承来的虚方法 ToString()，任何非字符串参数将被转换成字符串表示法。如果 ToString()返回 null，则用一个空字符串代替。字符串连接运算符的结果是一个字符串，由左操作数的字符后面连接右操作数的字符组成。字符串连接运算符不返回 null 值。如果没有足够的内存分配给结果字符串，将会产生 OutOfMemoryException 异常。



提示

对于除法运算符来说，整数相除的结果也应该为整数，比如 7/5 或 8/5 的结果都为 1，而不是 1.6 及 1.8，计算结果会舍弃小数部分。
a++和++a 都相当于 a=a+1，其不同之处在于：a++是先使用 a 的值，再进行 a+1 的运算；++a 则是先进行 a+1 的运算，再使用 a 的值。--a 和 a--类似于 a++和++a。初学者一定仔细分辨其中的区别。

2. 赋值运算符

赋值运算符用于将一个数据赋予一个变量、属性或者引用，数据可以是常量，也可以是表达式。前面已经多次使用了简单的等号“=”赋值运算符，例如 `int one=8`，或者 `int three=one+two`。其实，除了等号运算符，还有一些其他的赋值运算符，它们都非常有用。这些赋值运算符都是在“=”之前加上其他运算符，这样就构成了复合的赋值运算符。复合赋值运算符的运算非常简单，例如“`b+=8`”就等价于“`b=b+8`”，它相当于对变量进行一次自加操作。表 2-3 列出了复合赋值运算符的定义和含义。

表 2-3 复合赋值运算符

复合赋值运算符	类别	描述
<code>+=</code>	二元	<code>one += two</code> 等价于 <code>one = one + two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 的和
<code>-=</code>	二元	<code>one -= two</code> 等价于 <code>one = one - two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 的差
<code>*=</code>	二元	<code>one *= two</code> 等价于 <code>one = one * two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 的乘积
<code>/=</code>	二元	<code>one /= two</code> 等价于 <code>one = one / two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 相除所得的结果
<code>%=</code>	二元	<code>one %= two</code> 等价于 <code>one = one % two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 相除所得的余数
<code>&=</code>	二元	<code>one &= two</code> 等价于 <code>one = one & two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 进行“与”操作的结果
<code> =</code>	二元	<code>one = two</code> 等价于 <code>one = one two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 进行“或”操作的结果
<code>^=</code>	二元	<code>one ^= two</code> 等价于 <code>one = one ^ two</code> ， <code>one</code> 被赋予 <code>one</code> 与 <code>two</code> 进行“异或”操作的结果
<code>>>=</code>	一元	<code>one >>= two</code> 等价于 <code>one = one >> two</code> ，把 <code>one</code> 的二进制值向右移动 <code>two</code> 位，就得到 <code>one</code> 的值
<code><<=</code>	一元	<code>one <<= two</code> 等价于 <code>one = one << two</code> ，把 <code>one</code> 的二进制值向左移动 <code>two</code> 位，就得到 <code>one</code> 的值

3. 关系运算符

关系运算符表示了对操作数的比较运算，有关系运算符组成的表达式就是关系表达式。关系表达式的结果只可能有两种：“true”或“false”。常用的关系运算符有 6 种，如表 2-4 所示。

表 2-4 关系运算符

关系运算符	类别	描述
<code>></code>	二元	大于关系比较，例如 <code>10 > 1</code> 的结果为 <code>true</code> ， <code>1 > 10</code> 的结果为 <code>false</code>
<code><</code>	二元	小于关系比较，例如 <code>10 < 1</code> 的结果为 <code>false</code> ， <code>1 < 10</code> 的结果为 <code>true</code>
<code>==</code>	二元	等于关系比较，例如 <code>10 == 1</code> 的结果为 <code>false</code> 。 <code>int a = 100; 100 == a;</code> 的结果为 <code>true</code>
<code>>=</code>	二元	大于等于关系比较，例如 <code>10 >= 1</code> 的结果为 <code>true</code> ， <code>1 >= 10</code> 的结果为 <code>false</code>
<code><=</code>	二元	小于等于关系比较，例如 <code>10 <= 1</code> 的结果为 <code>false</code> ， <code>1 <= 10</code> 的结果为 <code>true</code>
<code>!=</code>	二元	不等于关系比较，例如 <code>10 != 1</code> 的结果为 <code>true</code> 。 <code>int a = 100; 100 != a;</code> 的结果为 <code>false</code>

4. 逻辑运算符

逻辑运算符主要用于逻辑判断，包括逻辑与逻辑或和逻辑非。其中，逻辑与和逻辑或属于二元运算符，它要求运算符两边有两个操作数，这两个操作数的值必须为逻辑值。逻辑非运算符是一元运算符，它只要求有一个操作数，操作数的值也必须为逻辑值。由逻辑运算符组成的表达式是逻辑表达式，其值只可能有两种，即“true”或“false”。表 2-5 是关于逻辑运算符的说明。

表 2-5 逻辑运算符

逻辑运算符	类别	描述
&&	二元	逻辑与运算时，如果有任何一个运算元为假，则运算结果也为假，只有两个运算元都为真时运算结果才为真。
	二元	逻辑或运算同“逻辑与”运算正好相反，如果有任何一个运算元为真，则运算结果也为真，只有两个运算元都为假时运算结果才为假。
!	一元	逻辑非运算是针对操作数的逻辑值取反，即如果操作数的逻辑值为真，则运算结果为假；反之，如果操作数的逻辑值为假，则运算结果为真。

下面通过一段程序来说明如何使用逻辑运算符，代码如下。

```
1.  int one= 8;
2.  int two = 80;
3.  bool three = (one>0) && (two>0);
4.  bool four = (one>8) && (two>8);
5.  bool five=(one <0) || (two<0);
6.  bool six = (one <=8) || (two<8);
7.  bool seven= !(80>0);
```

在上面代码中，第 3 行定义的 bool 类型变量 three 的值为 true，因为 one>0 的值为 true，并且 two>0 的值也为 true。第 4 行的 bool 变量 four 的值为 false，因为 one>8 的值为 false，只要有一个值为 false，最后的结果也就是 false。第 5 行的 bool 变量 five 的值为 false，因为 one <0 的值为 false 并且 two<0 的值也为 false。第 6 行的 bool 变量 six 的值为 ture，因为 one <=8 的值为 true，只要有一个值为 true，最后的结果也就是 true。第 7 行的 bool 变量 seven 的值为 false，因为 80>0 为 true，对它取反后得到的值为 false。

5. 条件运算符

C# 中唯一的一个三元操作符就是条件运算符(?:)，由条件运算符组成的表达式就是条件表达式，条件表达式的一般格式为：

```
操作数 1?操作数 2:操作数 3。
```

其中，“操作数 1”的值必须为逻辑值，否则将出现编译错误。进行条件运算时，首先判断问号前面的“操作数 1”的逻辑值是真还是假，如果逻辑值为真，则条件运算表达式的值等于“操作数 2”的执行结果值；如果为假，则条件运算表达式的值等于“操作数 3”的执行结果值。例如下面的代码。

```
1.  int one =8
```

```
2. int two = 10
3. int three = one > two ? 80 : -8;
```

代码说明：第 1 和第 2 行分别定义了两个整型变量。第 3 行最后变量 `three` 的值为 -8，因为因为 `one > two` 的值为 `false`。

条件表达式的类型由“操作数 2”和“操作数 3”控制，如果“操作数 2”和“操作数 3”是同一类型，那么这一类型就是条件表达式的类型。否则，如果存在“操作数 2”到“操作数 3”（而不是“操作数 3”到“操作数 2”）的隐式转换，那么“操作数 3”的类型就是条件表达式的类型。反之，“操作数 2”的类型就是条件表达式的类型。

6. 位运算符

位运算符是以二进制的方式操作数据，并且操作数和结果都是整数类型的数据。位运算符主要包括按位与、按位或、按位异或、按位取反、左移和右移操作。在这些运算符中，除按位取反运算符是一元运算符外，其他的都是二元运算符。位运算符的详细信息和使用方法如表 2-6 所示。

表 2-6 位运算符

位运算符	类别	描述
&	按位与	按位与运算符是把两个操作数的各个对应的二进制位进行“与”操作，如果两个操作数相应的二进制位都为 1，那么运算结果对应的位也为 1，否则结果位为 0。例如：10010001&11110000=10010000
	按位或	按位或运算符是把两个操作数的各个对应的二进制位进行“或”操作，如果两个操作数相应的二进制位有一个为 1，那么运算结果对应的位就为 1，否则结果位为 0。例如：10010001 11110000=11110001
^	按位异或	按位异或是把两个操作数的各个对应的二进制位进行“异或”操作，如果两个操作数相应的二进制位相同，那么结果位的值就为 0，否则就为 1，也就是相异为 1，相同为 0。例如：10010001^11110000=01100001
~	按位取反	按位取反是一元运算符，它对二进制数进行按位取反，即将二进制数的 0 转换为 1，1 转换为 0。例如：~0000 0001=0000 1100
<<	左移	左移运算符是二元运算符，它将数的二进制位全部向左移动指定的位数，并在后面移入的空位添 0，而前面的高位移出后会被舍弃。例如：70<<2，即 01000110<<2=00011000。
>>	右移	右移运算符是二元运算符，它将一个数的二进制位全部向右移动指定的位数，并在前面移入的空位填 0，而后面的低位移出后会被舍弃。例如：70>>2，即 01000110>>2=00010001

7. 运算符的优先级

在 C# 中为上面所述的多种运算符定义了不同的优先级，相同优先级的运算符，除了赋值运算符按照从右至左的顺序执行之外，其余运算符按照从左至右的顺序执行。括号是优先级最高的，可以任意改变符号的计算顺序。在 C# 中运算符的优先级定义如表 2-7 所示，其中 1 级表示最高优先级，12 级表示最低优先级。

表 2-7 运算符的优先级

级别	符号	说明
1	++	在操作符前面
	--	在操作符后面
	+	正号
	-	负号
	!	逻辑非
	~	按位取反
2	*	算术乘号
	/	算术除号
	%	算术求余
3	+	算术加法
	-	算术减法
4	<<	左移
	>>	右移
5	<	大于
	>	小于
	<=	小于等于
	>=	大于等于
6	==	关系等于
	!=	关系不等于
7	&	按位与
8	^	按位异或
9		按位或
10	&&	逻辑与
11		逻辑或
12	=	赋值等于
	*=, /=, %=, +=, -=, <<=, >>, &=, ^=, =	复合赋值符



提示

仅仅依靠优先级来安排数据的运算顺序是可靠的，大部分情况下考虑使用括号来进行强制优先级，凡是用括号括起来的比其他运算符都有高的优先级。

【例 2-2】本例通过控制台输入长方形的两条边长，求出长方形的面积并将结果显示在控制台屏幕上，实现步骤如下。

01 启动 Visual Studio 2010，执行“文件” | “新建项目”命令，在弹出的“新建项目”对

话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-2”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-2”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中加入如下逻辑代码。

```
1. static void Main(string[] args)
2.     {
3.         Console.WriteLine("请输入长方形的长: ");
4.         int a = int.Parse(Console.ReadLine());
5.         Console.WriteLine("请输入长方形的宽: ");
6.         int b = int.Parse(Console.ReadLine());
7.         var area = a * b;
8.         Console.WriteLine("长方形的面积为{0}",area);
9.     }
```

代码说明：第 1 行定义了一个 Main 函数。第 2~6 行获取输入的长方形的长和宽。第 7 行定义了隐形局部变量 area 计算长方形长和宽的乘积。第 8 行向控制台屏幕打印输出结果。

03 按下“Ctrl+F5”，程序运行结果如图 2-10 所示。

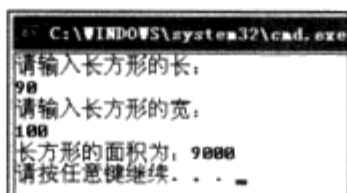


图 2-10 运行结果

2.2.4 转义字符

在 C# 中，通常用字符“\”加上另外一种字符组成的字符组合来表示一种含义，这种方式称为转义，把字符“\”称为转义字符，例如：

- \"表示双引号，在 C# 中，字符串以双引号来封闭的，因此在字符串里需要包含双引号的时候，就需要利用这种转义的形式。
- \n 表示换行。
- \t 表示制表符。
- \\表示单斜杠“\”，由于字符“\”被定义为转义字符，所以需要“\\”表示单斜杠“\”。

C# 中有很多转义字符，这里不再一一解说，读者可以参考 MSDN。

2.2.5 C# 中的控制语句

一般来说，程序代码除了顺序执行之外，对于复杂的工作，为了达到预期的执行结果，还需要使用流程控制语句来控制程序的执行。

流程控制语句使用条件表达式来进行判断，以便执行不同的代码段，或是重复执行指定的代码段。

1. 选择语句

选择语句就是分支控制语句，决定哪个流程分支被执行。要跳转到的代码分支由某个条件语句来控制，条件语句使用布尔逻辑。C# 中选择语句主要有 if 语句和 switch 语句，三元运算符(?:) 也有分支的功能。三元运算符前面已经介绍过，下面主要介绍 if 语句和 switch 语句。

(1) if 语句

if 语句是最常用的分支语句，使用该语句可以有条件地执行其他语句。if 语句最基本的使用格式为：

```
if(布尔表达式)
    布尔表达式为 true 时的代码或者代码块
```

程序执行时首先检测布尔表达式的值，如果布尔表达式的值是 true，就执行 if 语句中的代码，代码执行完毕后，将继续执行 if 语句下面的代码。如果布尔表达式的值是 false，则直接跳转到 if 语句后面的代码执行。如果 if 语句中为代码块（即有多于 1 行代码），则需要使用大括号“{}”把代码包括起来。当只有一行代码时可以省略大括号。

if 语句可以和 else 关键字合并使用，使用格式如下：

```
if(布尔表达式)
    语句 1
else
    语句 2
```

如果布尔表达式的值为真，首先执行语句 1 的内容，然后再执行 if 语句后的代码。如果布尔表达式的值为假，将首先执行语句 2，然后再执行 if 语句后的代码。同样地，如果 if 语句中为代码块（即有多于 1 行代码），则需要使用大括号“{}”把代码包括起来。

如果有多个条件需要判断，也可以通过添加 else if 语句。if 语句允许使用嵌套来实现复杂的选择流程。

【例 2-3】接受用户输入的三个数字，通过选择语句判断其中的最大值和最小值数并输到控制台显示。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-3”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-3”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```
1. static void Main(string[] args){
2.     var max = 0;
3.     var min = 0;
4.     Console.WriteLine("请输入第 1 个数");
5.     var a = Convert.ToInt32(Console.ReadLine());
6.     Console.WriteLine("请输入第 2 个数");
7.     var b = Convert.ToInt32(Console.ReadLine());
8.     Console.WriteLine("请输入第 3 个数");
```

```

9.      var c = Convert.ToInt32(Console.ReadLine());
10.     if (a > b){
11.         max = a;
12.         min = b;
13.     }
14.     else{
15.         max = b;
16.         min = a;
17.     }
18.     if (max < c)
19.         max = c;
20.     else if (min > c)
21.         min = c;
22.     Console.WriteLine("最大的数={0}, 最小的数={1}", max, min);
23. }

```

代码说明：第 1 行定义了一个 Main 函数。第 2~3 行声明两个整型变量。第 4~9 行获得输入的三个数字。第 10~17 行使用 if.....else.....语句判断第 1 个和第 2 个数字中的最大值和最小值。第 18~21 行使用 if.....else if.....语句将第一次判断的结果和第三个数值进行比较，判断最大值和最小值。第 22 行将判断的结果输出。

03 按下“Ctrl+F5”，程序运行结果如图 2-11 所示。

(2) switch 语句

switch 语句非常类似于 if 语句，它也是根据测试的值来有条件地执行代码，实际上 switch 语句完全可以使用 if 语句代替。一般情况下，如果有简单的几个分支就需要使用 if 语句，否则建议使用 switch 语句。与 if 语句不同的是，switch 语句可以一次将测试变量与多个值进行比较，而不是仅仅测试一个条件，这样可以使代码的执行效率比较高。switch 语句的基本语法定义如下：

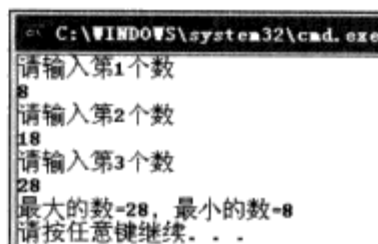


图 2-11 运行结果

```

switch (控制表达式)
{
    case 测试值 1:
        当控制表达式的值等于测试值 1 时要执行的代码
        break;
    case 测试值 2:
        当控制表达式的值等于测试值 2 时要执行的代码
        break;
    ...
    case 测试值 n:
        当控制表达式的值等于测试值 n 时要执行的代码
        break;
    default:
        当控制表达式的值不等于以上各个测试值时要执行的代码
        break;
}

```

在 switch 语句的开始，首先计算控制表达式的值，如果该值符合某个 case 语句中定义的“测试值”就跳转到该 case 语句执行，当控制表达式的值没有任何匹配的“测试值”时就执行 default 块中的代码。执行完代码块后退出 switch 语句，继续执行下面的代码。其中，测试值只能是整数类型或者是字符类型并且各个测试值要互不相同。default 语句是可选成分，没有 default 语句时，

如果控制表达式的值没有任何匹配的“测试值”，程序将会退出 switch 语句转而执行后面的代码

【例 2-4】本例使用 switch 语句完成一个简单的计算器程序，用户输入运算数和四则运算符，输入计算结果。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-4”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-4”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```

1. static void Main(string[] args){
2.     Console.WriteLine("请输入第 1 个数");
3.     var a = Convert.ToInt32(Console.ReadLine());
4.     Console.WriteLine("请输入运算类型");
5.     var b = char.Parse(Console.ReadLine());
6.     Console.WriteLine("请输入第 2 个数");
7.     var c = Convert.ToInt32(Console.ReadLine());
8.     switch (b){
9.         case '+': Console.WriteLine("计算结果为:{0}", a+c);
10.            break;
11.        case '-': Console.WriteLine("计算结果为:{0}", a-c);
12.            break;
13.        case '*': Console.WriteLine("计算结果为:{0}", a*c);
14.            break;
15.        case '/': Console.WriteLine("计算结果为:{0}", a/c);
16.            break;
17.        default: Console.WriteLine("计算符号输入错误");
18.            break;
19.    }
20. }
```

代码说明：第 1 行定义一个 Main 函数。第 2~7 行接受用输入的数字和运算符。第 8~19 行使用 switch 语句，判断用户输入的运算符号，在控制台显示相应相应的四则运算结果。其中第 17 行判断如果运算符号输入有误，输出提示信息。

03 按下“Ctrl+F5”，程序运行结果如图 2-12 所示。

2. 步循环语句

当需要反复执行某些相似的语句时，就可以使用循环语句了，这对于大量的重复操作（上千次，甚至百万次）时尤其有意义。C#中的循环语句有四种：do-while 循环，while 循环，for 循环和 foreach 循环，下面分别进行介绍。

（1）do-while 语句

do-while 语句根据其布尔表达式的值有条件的执行它的循环语句一次或者多次，其语法定义如下：

```

do
    循环代码
while (布尔表达式);
```

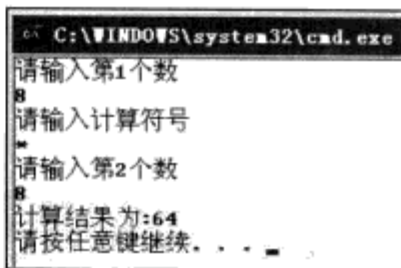


图 2-12 运行结果

do-while 循环以下述方式执行：程序首先执行一次循环代码，然后判断布尔表达式的值，如果值为 true 就从 do 语句位置开始重新执行循环代码，一直到布尔表达式的值 false。所以，无论布尔表达式的值是 true 还是 false，循环代码会至少执行一次。当循环代码要执行多条语句时，要用大括号“{}”把所要执行的语句括起来。

(2) while 语句

while 循环非常类似于 do-while 循环，其语法定义如下：

```
while (布尔表达式)
    循环代码
```

while 语句和 do-while 语句有一个重要的区别：while 循环中的布尔测试是在循环开始时进行，而 do-while 循环是在最后检测。如果测试布尔表达式的结果为 false 就不会执行循环代码，程序直接跳转到 while 循环后面的代码执行，而 do-while 语句则会至少执行一次循环代码。当循环代码要执行多条语句时，要用大括号“{}”把所要执行的语句括起来。

【例 2-5】本例接受用户从键盘输入的数值，当数值小于 10 时，将打印出从输入数值到 10 之间的所有数字之和，分别使用 while 和 do-while 语句实现。通过此例观察两种语句的区别。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-5”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-5”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```
1.      static void Main(string[] args){
2.          var sum = 0;
3.          Console.WriteLine("请输入数字?");
4.          var i=int.Parse (Console.ReadLine());
5.          while (i <= 10){
6.              sum = sum + i;
7.              i++;
8.          }
9.          Console.WriteLine("使用 while 语句获得的结果等于{0}",sum);
10.         var sun = 0;
11.         Console.WriteLine("请输入数字");
12.         var a = int.Parse(Console.ReadLine());
13.         do{
14.             sun = sun + a;
15.             a++;
16.         } while (a <= 10);
17.         Console.WriteLine("使用 do-while 语句获得的结果等于{0}", sun);
18.     }
```

代码说明：第 1 行定义一个 Main 函数。第 2、3 行获得用户的输入。第 5~8 行使用 while 语句获得输入数值到 10 之间所有的数字之和。第 13~16 行使用 do-while 语句实现上面 while 语句同样的功能。第 9、17 行打印结果。

03 按下“Ctrl+F5”，程序运行后，在控制台命令行输入10，运行结果如图2-13所示。

04 如果在控制台命令行输入11，运行结果如图2-14所示。

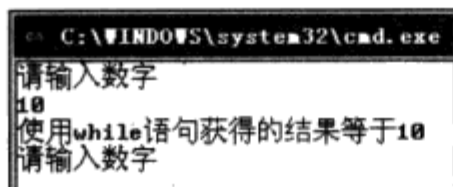


图 2-13 运行结果 1

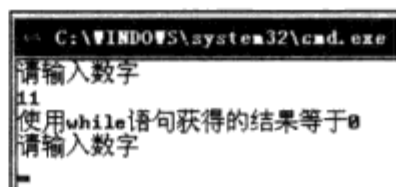


图 2-14 运行结果 2

出现上面不同结果的原因是因为 while 循环是先判断后执行，因此当输入数值 11 时，循环控制条件返回 false，不执行循环代码，直接执行循环代码后的输出语句打印 sum 的值；而 do-while 循环是先执行后判断的循环，因此输入数值 11 时，先执行循环代码，使变量 sum 的值在原值基础上加上从控制台接收的 11，然后再判断 a 的值，虽然循环条件返回 false 退出循环，但此时循环代码已经执行了一次，因此，输出的值是 11。

(3) for 语句

for 循环是最常用的一种循环语句，这类循环可以执行指定的次数，并维护它自己的计数器。for 语句首先计算一系列初始表达式的值，接下来当条件成立时，执行其循环语句，之后计算重复表达式的值，并根据其值决定下一步的操作。for 循环的语法定义如下：

```
for (循环变量初始化; 循环条件; 循环操作)
    循环代码
```

循环变量初始化可以存在，也可以不存在。如果该部分存在，则可能是一个局部变量声明和初始化的语句（循环计数变量）或者是一系列用逗号分割的表达式。此局部变量的有效区间从它被声明开始到循环语句结束为止。有效区间包括 for 语句执行条件部分和 for 语句重复条件部分。

循环条件部分可以存在也可以不存在。如果没有循环停止条件，则循环可能为死循环，除非 for 循环语句中有其他的跳出语句。循环条件部分用于检测循环的执行条件，如果符合条件就执行循环代码，否则就执行 for 循环后面的代码。

循环操作部分也是可以存在或者不存在的。在每一个循环结束或执行循环操作部分，因此通常会在这个部分修改循环计数器的值，使之最终逼近循环结束的条件。当然这并不是必须的，完全可以在循环代码中修改循环计数器的值。

下面的代码通过 for 循环在标准输出设备上打印输出从 1 到 100。

```
1.  for (int i = 1; i <= 100; i++){
2.  Console.WriteLine("{0}", i);
3.  }
```

在上面第 1 行代码中，程序首先执行 `int i=1`，声明并初始化了循环计数器。然后执行 `i <= 10`，判断 `i` 的值是否小于等于 10。这里 `i` 的值为 1，满足循环条件，因此会执行循环代码在标准输出设备上打印输出 1。最后执行 `i++` 语句，使得循环计数器的值变为 2。

第一个循环完毕后开始执行第二个循环，首先检测 `i` 的值是否符合循环条件，如果满足就继续执行循环代码，并在最后更新 `i` 的值。如此循环一直到 `i` 的值变为 101 后，循环条件不再满足了，此时跳转到 for 循环的下一条语句执行。

【例 2-6】本例接受用户从键盘输入一个正整数，然后程序会计算这个数的阶乘并输出。接着用户可以再输入另一个正整数计算它的阶乘，直到输入一个负数为止。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-6”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-6”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```
1. static void Main(string[] args){
2.     int a = 0;
3.     do{
4.         Console.WriteLine("请输入一个正整数计算它的阶乘，如果想结束输入请输入一个负数!");
5.         Console.Write("请输入: ");
6.         a = int.Parse(Console.ReadLine());
7.         long b = 1;
8.         for (int i = 1; i <= a; i++){
9.             b *= i;
10.        }
11.        Console.WriteLine(a.ToString() + "的阶乘=" + b.ToString());
12.    } while (a >= 0);
13. }
```

代码说明：第 3~13 行使用 do-while 循环控制输入的次数。其中，第 6 行获得用户输入的数值。第 8~10 行嵌套了一个 for 循环计算用户输入值的阶乘。第 12 行设置输入的结束的条件。

03 按下“Ctrl+F5”，程序运行结果如图 2-15 所示。

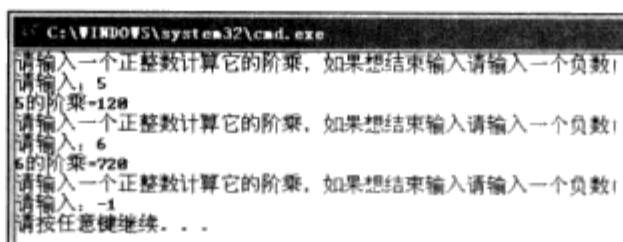


图 2-15 运行结果

(4) foreach 语句

foreach 语句列举出一个集合（collection）中的所有元素，并执行关于集合中每个元素的嵌套语句。foreach 语句的语法定义如下：

```
foreach (类型 标识符 in 表达式)
    循环代码
```

foreach 语句括号中的类型和标识符用来声明该语句的循环变量，标识符即循环变量的名称。循环变量相当于一个只读的局部变量，它的有效区间为整个循环语句内。在 foreach 语句执行过程中，重复变量代表着当前操作针对的集合中相关元素。

并非所有的类型都可以用 foreach 来遍历，可以遍历的类型必须包含公有非静态方法 GetEnumerator()，并且由 GetEnumerator()返回的结构、类、接口等必须包含一个 MoveNext()的方法。

法，返回值为布尔型。

【例 2-7】通过 foreach 循环将找出字符数组中 0 和 1 的数量。

01 一步启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-7”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成例“例 2-7”的项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码：

```

1. static void Main(string[] args){
2.     int a = 0, b = 0;
3.     char[] c = new char[] { '0', '0', '1', '1', '1', '1', '0', '0', '1' };
4.     foreach (char e in c){
5.         if (e == '0')
6.             a++;
7.         else
8.             b++;
9.     }
10.    Console.WriteLine("0 的个数{0}, 1 的个数{1}",a,b);
11. }
```

代码说明：第 2 行定义了 2 个变量 a 和 b，分别表示字符 0 和 1 的个数。第 3 行声明了一个字符数值。第 4~9 行使用 foreach 循环判断字符格式。如果遍历到的字符为 0，将 a 的值加 1，否则就将 b 的值加 1。第 10 行输出打印结果显示到控制台。

03 按下“Ctrl+F5”，程序运行结果如图 2-16 所示。

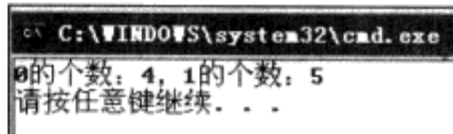


图 2-16 运行结果

3. 跳转语句

跳转语句可以更改流程，进行无条件跳转，在 C# 中共提供了如下 5 种跳转语句。

- break 语句：终止并跳出循环
- continue 语句：终止当前的循环，重新开始一个新的循环
- goto 语句：跳转到指定的位置
- return 语句：跳出循环及其包含的函数
- throw 语句：抛出一个异常

在方法或函数中会使用到 return 语句，用于退出方法（当然也就退出了循环了），如果需要抛出一个异常，则需要使用 throw 语句。goto 语句并不常用，建议读者不要使用 goto 语句，因为该语句可能会破坏程序的结构性。

break 语句用于跳出包含它的 switch、while、do、for 或者 foreach 语句。break 语句的目标地址为包含它的 switch、while、do、for 或 foreach 语句的结尾。假如 break 不是在 switch、while、do、for 或者 foreach 语句的块中，将会发生编译错误。

下面的代码使用 break 语句查询 1~100 中符合条件的数值，如果查找成功就输出数值，然后跳出循环。

```

1.  for(int i=1;i<100;i++){
2.      if(i==5)
3.          break;
4.      Console.WriteLine(i);
5.  }

```

代码说明：第 1~5 行通过 for 循环遍历 1~100。其中，第 2 行判断如果遍历到数值 5，第 3 行使用 break 语句跳出循环，所有最后输出的结果为：1,2,3,4。

continue 语句用于终止当前的循环，并重新开始新一次包含它的 while、do、for 或者 foreach 语句的执行。假如 continue 语句不被 while、do、for 或者 foreach 语句包含，将产生编译错误。以下代码用于打印 100 以内的偶数。

```

1.  for(int i=1;i<100;i++){
2.      if(i%2!=0)
3.          continue;
4.      Console.WriteLine(i);
5.  }

```

代码说明：第 1~5 行通过 for 循环遍历 1~100。其中，第 2 行判断如果遍历到数值是奇数。第 3 行使用 continue 语句终止当前的循环，重新开始一个新的循环，所以最后输出的数字为 100 以内的所有的偶数。



提示

当有 switch、while、do、for 或 foreach 语句相互嵌套的时候，break 语句只是跳出直接包含它的那个语句块。如果要在多处嵌套语句中完成转移，必须使用 return 或者 goto 语句。

4. 异常处理

程序“异常”（Exception），是指程序运行中的一种“例外”情况，也就是正常情况以外的一种状态。异常对程序可能碰到的错误进行了概括，是错误的集合。如果对异常置之不理，程序会因为它而崩溃。往往一个微小的异常错误也会使终止代码的继续执行。

我们希望在程序出现异常的时候，开发人员能够通过有针对性的代码编写来加以处理，在一定的程度上限制异常产生的影响，使程序输出异常信息的同时能得以继续运行。

在一般情况下，我们会考虑在容易出现异常情况的场合中使用异常处理，例如：

- 算术错误，如以零作除数；
- 方法接收的参数错误；
- 数组大小与实际不符；
- 数字转化格式异常；
- 空指针引用；
- 输入输出错误；
- 找不到文件；
- 不能加载所需的类。

以上列举的仅是一些常用的场合。在程序中产生异常的情况是非常普遍的，下面我们通过代

码看看程序中异常的出现。

```

1. class Program{
2.     static void Main(string[] args) {
3.         int a=20/int.Parse("0");
4.         Console.WriteLine ("计算结果是: "+a);
5.         Console.WriteLine ("程序结束");
6.     }
7. }
```

以上代码运行结果显示：未处理的异常，试图除以零。程序在第 3 行被迫中止了，以后的代码都未执行。这肯定不是我们所期望的结果。那么，如何通过编码来解决这一情况呢？

在 C# 中，可以用异常和异常处理程序很容易地将实现程序主逻辑的代码与错误处理代码区分开来，所有的异常都是从 System.Exception 继承而来，此类是所有异常的基类。当发生错误时，系统或当前正在执行的应用程序，通过引发包含关于该错误的信息的异常来报告错误。异常发生后，将由该程序或默认异常处理程序处理。

当在一个函数或方法中遇到异常处理的时候，就会创建一个异常处理的对象，并在函数中被抛出（throw）。当然，也可以在此函数中处理该异常。为了在函数中实现监视和处理异常的代码，C# 提供了 3 个关键字：try、catch 和 finally。try 关键字后面的代码块称为一个 try 块，那么 catch 后的代码称为 catch 块，finally 形成一个 finally 块。标准的异常处理语法定义如下：

```

try{
    程序代码块;
}
catch (Exception e) {
    异常处理代码块;
}
catch(Exception e1){
    异常处理代码块;
}
finally{
    无论是否发生异常，均要执行的代码块;
}
```

以上代码处理异常的步骤如下。

01 代码要放在一个 try 块中。代码运行时，它会尝试执行 try 块中所有语句。如果没有任何语句产生一个异常，那么所有语句都会运行。这些语句将一个接一个运行，直到全部完成。然而，一旦出现异常，就会跳出 try 块，进入一个 catch 块处理程序中执行。

02 在 try 块之后紧接着写一个或多个 catch 处理程序，用于处理可能发生的错误。在 try 块中抛出的所有的异常对象与下面的每个 catch 块进行比较，判断其中的 catch 块是否可以捕捉此异常。

03 如果没有找到匹配的 catch 块，catch 块就不会被执行。非捕获的异常对象由 CLR 的省缺异常处理器处理，在此情况下，程序会突然终止。

04 finally 块中的代码总是被执行。

注意，一个 try 块可以：

- 有一个或多个相关的 catch 块，无 finally 块；

- 有一个 finally 块，无 catch;
- 包含一个或多个 catch 块，同时有 finally 块。

【例 2-8】通过使用 try、catch、finally 语句块进行上面例子中的异常处理。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-8”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-8”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```
1. static void Main(string[] args){
2.     try {
3.         int a = 20 / int.Parse("0");
4.         Console.WriteLine("计算结果为: ", a);
5.     }
6.     catch (ArithmeticException e){
7.         Console.WriteLine(e);
8.     }
9.     catch (Exception e1){
10.        Console.WriteLine(e1);
11.    }
12.    finally {
13.        Console.WriteLine("程序结束! ");
14.    }
15. }
```

代码说明：第 2~4 行使用 try 块将可能发生异常的代码写在其中。如果发生异常会将此异常抛出，中止了以下代码的继续执行。在第 6~11 行的两个 catch 块中捕获可能产生的异常，两相判断如果是同一类型的异常后，第 7、10 行显示异常的各种详细信息和提示语句。接着保证执行 finally 语句块中第 13 行程序结束的语句。

03 按下“Ctrl+F5”，程序运行结果如图 2-17 所示。

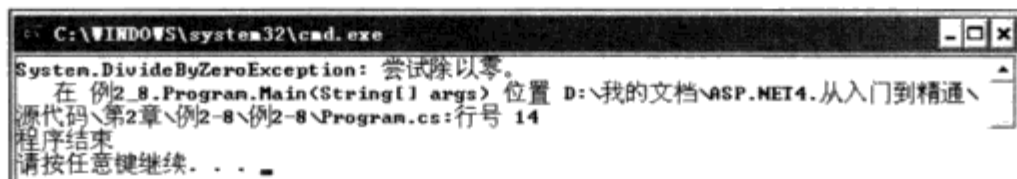


图 2-17 运行结果

2.3 面向对象编程

面向对象的程序设计（Object-Oriented Programming，OOP）是一种基于结构分析的、以数据为中心的程序设计方法。它是程序设计的一次大的进步，程序员跳出了结构化程序设计的传统方法，在程序设计过程中，过多的考虑事务的处理和现实世界的自然描述。与传统的面向过程的设计方法相比，采用面向对象的设计方法设计的程序可维护性较好，源程序易于阅读理解和修改，降低了复杂度。

其主要思想是将数据及处理这些数据的操作都封装（Encapsulation）到一个称为类（Class）的数据结构中，使用这个类时，只需要定义一个类的变量即可，这个变量叫做对象（Object）。

2.3.1 类

在 C# 中，类是一种功能强大的数据类型，而且是面向对象的基础。类定义属性和行为，我们可以声明类的实例，从而可以利用这些属性和行为。类中包含数据成员（常数、域和事件）、功能成员（方法、属性、索引、操作符、构造函数和析构造函数）和嵌套类型。类支持继承，派生的类可以对基类进行扩展和特殊化，使得程序代码可以复用，子类中可以继承祖先类中的部分代码。由于类封装了数据和操作，从类外面看，只能看到公开的数据和操作，而这些操作都在类设计时进行安全性考虑，因而外界操作不会对类造成破坏。

C# 中提供了很多标准的类，用户在开发过程中可以使用这些类，这样大大节省了程序的开发时间。C# 中也可以自己定义类，类的定义方法为：

```
[类修饰符] class 类名[:父类名]
{
    [成员修饰符] 类的成员变量或者成员函数;
};
```

在上面定义中，“类名”是自定义类的名字，该名字要符合标识符的要求。“父类名”表示从哪个类继承。“:父类名”可以省略，如果没有父类名，则默认从 Object 类继承而来。Object 类是每个类的祖先类，C# 中所有的类都是从 Object 类派生出来的。“类修饰符”用于对类进行修饰，说明类的特性，类的每个成员都需要设定访问修饰符，不同的修饰符会造成对成员访问能力不一样。如果没有显式指定类成员访问修饰符，默认类型为私有类型修饰符。C# 中类成员修饰符的定义和使用方法如表 2-8 所示。

表 2-8 成员修饰符的定义和使用方法

修饰符	含义	说明
new	新建的类或者类成员	当 new 用于修饰类成员时，new 修饰符用来指出派生成员要隐藏基类成员。对于一个类，可以用与继承成员相同的名称或签名来声明一个成员。当这发生时，派生类成员被称作隐藏了基类成员。隐藏一个继承成员并不被认为是错误的，但是会造成编译器给出警告。为了禁止这个警告，派生类成员的声明可以包括一个 new 修饰符
public	公有的	公有的成员对于任何人都是可见的，外界可以不受限制地访问。这是限制最少的一种访问方式，它的优点是使用灵活，缺点是外界可能会破坏对象成员值的合理性
protected	受保护的	当用 protected 修饰类成员时，表示该成员对于外界来说是隐藏的，但对于这个类的派生类则可以访问
internal	内部成员	表示该成员是内部成员，只有本类成员才能访问
private	私有成员	私有的成员是隐藏的，外界不能直接访问该成员变量或成员函数。对该成员变量或成员函数的访问只能由该类中其他函数访问，其派生类也不能访问

(续表)

修饰符	含义	说明
abstract	抽象函数	使用 abstract 修饰符可以定义抽象函数
const	常量	const 修饰符用于修饰常量，如果是常量表达式，则在编译时被求值
virtual	虚函数	virtual 用于修饰虚函数，对于虚函数，它的执行方式可以被派生类改变，这种改变是通过重载实现的
event	事件	event 修饰符定义一个事件
extern	外部实现	extern 修饰符告诉编译器函数将在外部实现
override	重载	override 修饰符用于修饰重载基类中的虚函数的函数
readonly	只读成员	修饰类的只读成员。一个使用 readonly 修饰符的域成员只能在它的声明或者在构造函数中被更改
static	静态成员	声明为 static 的成员属于类，而不属于类的实例，所有此类的实例都共用一个成员。访问静态成员时，也是通过类名访问的



在一个类声明中，同一类修饰符不能多次出现，否则会出错。不同的类修饰符可以组合使用，例如 protected internal、public sealed 等。有些修饰符不能放在一起使用，例如 public、private、protected、internal 等。

2.3.2 属性、方法和事件

在 C# 中，按照类的成员是否为函数将其分为两大类，一种不以函数形式体现，称为成员变量，主要有以下几个类型。

- 常量：代表与类相关的常量值。
- 变量：类中的变量。
- 事件：由类产生的通知，用于说明发生了什么事情。
- 类型：属于类的局部类型。

另一种是以函数形式体现，一般包含可执行代码，执行时完成一定的操作，被称为成员函数，主要有以下几个类型。

- 方法：完成类中各种计算或功能的操作，不能和类同名，也不能在前面加“~”波浪线符号。方法名不能和类中其他成员同名，既包括其他非方法成员，又包括其他方法成员。
- 属性：定义类的值，并对它们提供读、写操作。
- 索引指示器：允许编程人员在访问数组时，通过索引指示器访问类的多个实例，又称下标指示器。
- 运算符：定义类对象能使用的操作符。
- 构造函数：在类被实例化时首先执行的函数，主要是完成对象初始化操作。构造函数必须和类名相同。

- 析构函数：在类被删除之前最后执行的函数，主要是完成对象结束时的收尾操作。构造函数必须和类名相同，并前加一个波浪线符号“~”。

2.3.3 构造函数

当创建一个对象时，系统首先给对象分配合适的内存空间，随后系统就自动调用对象的构造函数。因此构造函数是对象执行的入口函数，非常重要。在定义类时，可以给出构造函数也可以不定义构造函数。如果类中没有构造函数，系统会默认执行 `System.Object` 提供的构造函数。如果要定义构造函数，那么构造函数的函数名必须和类名一样。构造函数的类型修饰符总是公有类型 `public`，如果是私有类型 `private` 的，表示这个类不能被实例化，这通常用于只含有静态成员类中。构造函数由于不需要显式调用，因而不用声明返回类型。构造函数可以带参数也可以不带参数。具体实例化时，对于带参数的构造函数，需要实例化的对象也带参数，并且参数个数要相等，类型要一一对应。如果是不带参数的构造函数，因而在实例化时对象不需要参数。下面是一个类中构造函数的代码例子。

```
1.  class Person{
2.      string _name;
3.      int _age;
4.  }
5.  public Person(string name,int age){
6.      string _name=name;
7.      int _age=age;
8.  }
9.  public Person(){
10.     string _name="John"
11.     int _age=20;
12. }
```

以上代码中，第 1~4 行定义了一个 `Person` 类。它有两个成员变量 `_name` 和 `_age` 分别表示人的姓名和年龄。第 5~8 行是 `Person` 类带两个参数的构造函数，通过参数给两个成员变量赋值。第 9~12 行是 `Person` 类带参数的构造函数，在函数中直接给两个成员变量赋值，这种在函数或方法中定义的参数，称为“形式参数”，简称“形参”。

上面的例子定义了两个不同的构造函数，像这样在一个类中如果有两个函数（包括构造函数）或者方法名称相同，但参数个数不同或者参数的类型不同，我们称为“方法的重载”。实现方法时，系统会自动选择合适的类型和调用的函数相匹配。

在使用定义的类时，可以使用任何一个构造函数创建实例化对象。对象是类的实例化，只有对象才能包含数据、执行行为、触发事件，而类只不过就像 `int` 一样是数据类型，只有实例化才能真正发挥作用。对象具有以下特点。

- C# 中使用的全都是对象。
- 对象是实例化的，对象是从类和结构所定义的模板中创建的。
- 对象使用属性获取和更改它们所包含的信息。
- 对象通常具有允许它们执行操作的方法和事件。
- 所有 C# 对象都继承自 `Object`。
- 对象具有多态性，对象可以实现派生类和基类的数据和行为。

对象的声明就是类的实例化，类实例化的方式很简单，通过使用 `new` 来实现，例如：

```
1. Person p1 = new Person ();
2. Person p2 = new Person("Mary",28);
```

代码说明：第 1 行使用前面定义的 `Person` 类默认构造函数实例化对象 `p1`。第 2 行使用前面定义的 `Person` 类带两个参数的构造函数实例化对象 `p2`，这种在调用函数或方法时提供的参数值，称为“实际参数”，简称“实参”。

2.3.4 继承和多态

为了提高代码复用性，C# 支持从父类中派生子类也就是类可以继承。同时，为了区分父类和子类的同名操作，C# 引入了“多态”的概念。

1. 继承

继承性是面向对象的一个重要特性，C# 中支持类的单继承，即只能从一个类继承。继承是传递的，如果 `C` 继承了 `B`，并且 `B` 继承了 `A`，那么 `C` 继承在 `B` 中声明的 `public` 和 `protected` 成员，同时也继承了在 `A` 中声明的 `public` 和 `protected` 成员。继承性使得软件模块可以最大限度地复用，并且编程人员还可以对前人或自己以前编写的模块进行扩充，而不需要修改原来的源代码，大大提高了软件的开发效率。

在定义类的时候，可以指定要继承的类，语法如下：

```
[类修饰符] class 类名[:父类名]
{
    [成员修饰符] 类的成员变量或者成员函数;
};
```

例如，类 `B` 从类 `A` 中继承，类 `A` 被称着基类，类 `B` 被称着派生类，代码如下。

```
1. public class A {
2.     public A() {}
3. }
4. public class B : A{
5.     public B() {}
6. }
```

代码说明：第 1 行定义了一个类 `A`，第 4 行定义了继承自类 `A` 的类 `B`。

派生类是对基类的扩展，派生类可以增加自己新的成员，但不能对已继承的成员进行删除，只能不予使用。基类可以定义自身成员的访问方式，从而决定派生类的访问权限。且可以通过定义虚方法、虚属性，使它的派生类可以重载这些成员，从而实现类的多态性。

一个派生类自动包含来自基类的所有字段。创建一个对象时，这些字段需要初始化。因此，有时候，需要通过调用基类的构造函数来对基类的字段进行初始化。在定义了构造函数的基础上，我们可以使用“`base`”关键字来调用基类的构造函数。

```
1. class Person{
2.     string _name;
3.     int _age;
4. }
```

```

5.     public Person(string name,int age){
6.         string _name=name;
7.         int _age=age;
8.     }
9.     class Man:Person{
10.         public Main(string str,ingt time):base(str, time)
11.     }

```

在上面代码中,第9~11行定义了一个继承与 Person 类的派生类 Man。其中,第10行使用“base”关键字调用了 Person 类的带参数的构造函数。

2. 多态

类的另外一个特性是多态,也称为多态性,所谓多态是指同一操作作用于不同类的实例,这些类进行不同的解释,从而产生不同的执行结果的现象。比如马戏团的杂技师进行动物训练,对于不同的动物有不同的训练方式,以下代码定义了两个不同的训练动物的方法。

```

1.     public void train(Dog dog){
2.         //训练小狗站立、排队、做算术的代码
3.     }
4.     public void train(Monkey monkey){
5.         //训练小猴敬礼、翻跟头、骑自行车的代码
6.     }

```

显然当调用代码中不同的对象小狗和小猴的实例时,会产生不同的结果。

在 C# 中有两种多态性,一种是编译时的多态性,这种多态性是通过函数的重载实现的,由于重载函数的参数或者是数量不同,或者是类型不同,所以编译系统在编译期间就可以确定用户所调用的函数是哪一个重载函数。另外一种多态性是运行时的多态性,这种多态性是通过虚成员方式实现的。运行时的多态性是指系统在编译时不确定选用哪个重载函数,而是在系统运行时,才根据实际情况决定采用哪个重载函数。

在定义类成员时,可以使用 virtual 关键字,virtual 关键字用于修改方法或属性的声明。被 virtual 关键字修饰的方法或属性称为虚拟成员,虚拟成员的实现可由派生类中的重写成员更改。

不能将 virtual 修饰符与 static、abstract、override 等修饰符一起使用,此外在静态属性上使用 virtual 修饰符是错误的。通过包括使用 override 修饰符的属性声明,可以在派生类中重写虚拟继承属性,这种重写的方法称为重写基方法。C# 中关于 override 重写的要求如下。

- 不能重写非虚方法或静态方法。重写基方法必须是虚拟的、抽象的或重写的。
- 重写基方法必须与重写方法具有相同的名字。
- 重写声明不能更改虚方法的可访问性,重写方法和虚方法必须具有相同的访问级修饰符。
- 不能使用 new、static、virtual、abstract 等修饰符修改重写方法。
- 返回值类型必须与基类中的虚拟方法一致。
- 参数列表中的参数顺序、数量和类型必须一致。

下面是一个子类重写基类中 train 方法的代码。

```

1.     class Animal{
2.         public virtual void train(){
3.             //训练站立、排队、做算术的代码

```

```

4.    }
5.    class Dog:Animal{
6.    public override void train(){
7.        //训练敬礼、翻跟头、骑自行车的代码
8.    }

```

在上面代码中，第2行在 Animal 类中使用关键字“virtual”声明 train 方法可以被派生类所重写。第6行在 Animal 类的派生类 Dog 类中使用关键字 override 重写父类的 train 方法。

【例 2-9】通过基类的引用，调用方法时根据实际对象的类型来动态绑定方法。某公司召开员工大会，会议主持人可以是员工代表，也可以是部门经理。现在要求在召开大会时制定一个会议主持人，该主持人作自我介绍。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-9”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-9”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```

1.    class Employee{
2.        protected String name;
3.        public Employee(string name){
4.            this.name = name;
5.        }
6.        public virtual void introduce(){
7.            Console.WriteLine("大家好！我是普通员工。");
8.            Console.WriteLine("我的名字是：" + name);
9.        }
10.    }
11.    class DepartmentManager : Employee{
12.        public DepartmentManager(string name) : base(name) {
13.            Console.WriteLine(name);
14.        }
15.        public override void introduce(){
16.            Console.WriteLine("大家好！我是部门经理。");
17.            Console.WriteLine("我的名字是：" + name);
18.        }
19.    }
20.    class Meeting{
21.        Employee emcee;
22.        public Meeting(Employee emcee){
23.            this.emcee = emcee;
24.        }
25.        public void begin(){
26.            emcee.introduce();
27.        }
28.    }
29.    static void Main(string[] args){
30.        Employee emp = new DepartmentManager("王冬");
31.        //Employee emp = new Employee("张琴");
32.        Meeting meeting = new Meeting(emp);
33.        meeting.begin();
34.    }

```

代码说明：第 1~10 行定义了一个 `Employee` 员工类。其中第 2 行定义员工的名字 `name`；第 3~5 行定义一个带名字参数的构造函数。第 6~9 行定义了一个 `introduce` 方法向控制台输出自我介绍。这个方法使用了关键字“`virtual`”声明可以被派生类所重写。第 11~19 行定义了一个 `DepartmentManager` 类继承与员工类。其中，第 12 行的构造函数使用了 `base` 关键字显式的调用基类的构造函数；第 15~17 行使用关键字 `override` 重写了基类的同名方法 `introduce`，输出部门经理的自我介绍。

第 20~28 行定义了一个 `Meeting` 会议类，其中，第 21 行声明了一个属性 `emcee`，代表会议的主持人，因为我们并不知道将来指定的主持人是员工还是部门经理，所以我们把它声明为一个基类的引用，而基类的引用可以指向派生类的对象，这就是实现多态绑定的关键。第 25~27 行定义了一个会议开始的方法 `begin`，在方法中调用 `introduce` 方法由主持人做自我介绍。

第 29~34 行的 `Main` 方法中，其中，第 30 行先实例化一名员工，这名员工可以是普通员工对象或者是部门经理对象。这里我们实例化一个叫王冬的部门经理对象。然后第 32 行再实例化一个会议类对象。最后 33 行调用会议开始的方法。

03 按下“`Ctrl+F5`”，程序运行结果如图 2-18 所示。

04 修改代码中第 30 行为：“`Employee emp = new Employee("张琴")`”，再运行程序，结果如图 1-19 所示。程序实现了根据实例化对象的不同动态绑定了不同方法。

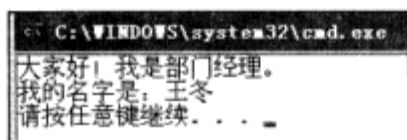


图 2-18 运行结果 1



图 2-19 运行结果 2

2.3.5 委托

委托其实也是一种引用方法的类型，创建了委托，就可以声明委托变量，也就是委托实例化。实例化的委托就是委托的对象，可以为委托对象分配方法，也就是把方法名赋予委托对象。一旦为委托对象分配了方法，委托对象将与该方法具有完全相同的行为。委托对象的使用可以像其他任何方法一样，具有参数和返回值，如下代码所示。

```
1. public delegate int JiSuan(int one, int two);
2. JiSuan js;
```

上面的代码第 1 行定义了一个名为 `JiSuan` 的委托，该委托封装了包含两个整型参数，且返回值为整型的方法。第 2 行声明 `js` 为委托 `d` 的对象，就可以把方法名赋给该对象。

方法的分配比较自由，任何与委托的签名（由返回类型和参数组成）匹配的方法都可以分配给该委托的对象。这样就可以通过编程方式来更改方法调用，还可以向现有类中插入新代码。只要知道委托的签名，便可以分配自己的委托方法。

将方法作为参数进行引用的能力，使委托成为定义回调方法的理想选择。例如，可以向排序算法传递对比较两个对象的方法的引用。分离比较代码使得可以采用更通用的方式编写算法。

委托具有以下特点。

- 委托类似于 C++ 函数指针，但它是类型安全的。

- 委托允许将方法作为参数进行传递。
- 委托可用于定义回调方法。
- 委托可以链接在一起；例如，可以对一个事件调用多个方法。
- 方法不需要与委托签名精确匹配。

构造委托对象时，通常提供委托将包装的方法的名称或使用匿名方法。实例化委托后，委托将把对它进行的方法调用传递给方法。调用方传递给委托的参数被传递给方法，来自方法的返回值（如果有）由委托返回给调用方，这称为调用委托。可以将一个实例化的委托视为被包装的方法本身来调用该委托。

例如，为上面的委托定义一个方法，代码如下：

```
public int JiaFa(int one, int two){
    return one * two;
}
```

定义一个委托的实例，把上面的方法赋给该委托实例，代码如下：

```
JiSuan js = JiaFa; //实例化委托 JiSuan
js(8,8); //调用委托
```

委托类型派生自 .NET Framework 中的 Delegate 类。委托类型是密封的，不能从 Delegate 中派生委托类型，也不可能从中派生自定义类。由于实例化委托是一个对象，所以可以将其作为参数进行传递，也可以将其赋值给属性。这样，方法便可以将一个委托作为参数来接受，并且以后可以调用该委托，这称为异步回调，是在较长的进程完成后用来通知调用方的常用方法。以这种方式使用委托时，使用委托的代码无需了解所用方法的实现方面的任何信息。此功能类似于接口所提供的封装。

回调的另一个常见用法是定义自定义的比较方法，并将该委托传递给排序方法。它允许调用方的代码成为排序算法的一部分。

例如，定义一个方法，该方法包含一个参数为前面定义的委托，代码如下：

```
public int JiSuanCallBack(int one, int two, JiSuan js){
    return js(one,two);
}
```

调用以上方法，并把前面定义的 JiSuan 委托的实例 js 传递给该方法，代码如下：

```
JiSuanCallBack(8,8,js);
```

将委托构造为包装实例方法时，该委托将同时引用实例和方法。除了它所包装的方法外，委托不了解实例类型，所以只要任意类型的对象中具有与委托签名相匹配的方法，委托就可以引用该对象。将委托构造为包装静态方法时，它只引用方法。

例如，声明一个类 JiSuanClass，该类包含两个方法，代码如下：

```
public class JiSuanClass{
    public int He(int one,int two) { //求两个整数的和
        return one+two;
    }
    public int Shang(int one,int two) { //求两个整数的商
        return one / two;
    }
}
```

```

    }
}

```

类 `JiSuanClass` 定义了两个方法，加上前面定义方法 `JiaFa`，可以把这三个方法都按前面委托的实例进行封装，代码如下：

```

JiSuanClass jsc = new JiSuanClass jsc ();
JiSuan js 1= jsc. He;
JiSuan js 2 = jsc. Shang;
JiSuan js 3 = JiaFa;

```

调用委托时，它可以调用多个方法，这称为多路广播。若要向委托的方法列表（调用列表）中添加额外的方法，只需使用加法运算符或加法赋值运算符（“+”或“+=”）添加委托。例如：

```

JiSuan js = js 1 + js 12;
js += js 3;

```

此时，`js` 在其调用列表中包含三个方法：`JiaFa`、`He` 和 `Shang`，原来的三个委托 `js 1`、`js 2` 和 `js 3` 保持不变。调用 `js` 时，将按顺序调用所有这三个方法。如果委托使用引用参数，则引用将依次传递给三个方法中的每个方法，由一个方法引起的更改对下一个方法是可见的。如果任一方法引发了异常，而在该方法内未捕获该异常，则该异常将传递给委托的调用方，并且不再对调用列表中后面的方法进行调用。如果委托具有返回值和/或输出参数，它将返回最后调用的方法的返回值和参数。若要从调用列表中移除方法，请使用减法运算符或减法赋值运算符（“-”或“-=”）。例如：

```

js -= js 1;
js 4 = js - js 2;

```

由于委托类型派生自 `System.Delegate`，所以可在委托上调用该类定义的方法和属性。例如，为了找出委托的调用列表中的方法数，可以编写下面的代码：

```

int Number = js.GetInvocationList().GetLength(0);

```

多路广播委托广泛用于事件处理中。事件源对象向已注册接收该事件的接收方对象发送事件通知。为了为事件注册，接收方创建了旨在处理事件的方法，然后为该方法创建委托并将该委托传递给事件源。事件发生时，源将调用委托。然后，委托调用接收方的事件处理方法并传送事件数据。给定事件的委托类型由事件源定义。

2.3.6 事件

事件是类在发生其关注的事情时用来提供通知的一种方式。例如，封装用户界面控件的类可以定义一个在用户单击该控件时发生的事件。控件类不关心单击按钮时发生了什么，但它需要告知派生类单击事件已发生，然后，派生类可选择如何响应。

事件使用委托来为触发时将调用的方法提供类型安全的封装。委托可以封装命名方法和匿名方法。

事件具有以下特点。

- 事件是类用来通知对象需要执行某种操作的方式。
- 尽管事件在其他时候（如信号状态更改）也很有用，事件通常还是用在图形用户界面中。

- 事件通常使用委托事件处理程序进行声明。
- 事件可以调用匿名方法来替代委托。

【例 2-10】设计一个员工类，成员包括员工的姓名和工资，使用委托的方法对员工发放的工资进行排序输出。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-10”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-10”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```

1.  delegate bool CompareOp(object lhs, object rhs);
2.  class BubbleSorter{
3.      public static void Sort(object[] sortArray, CompareOp getCompareMethod) {
4.          for (int i = 0; i < sortArray.Length; i++)
5.              for (int j = i + 1; j < sortArray.Length; j++)
6.                  if (getCompareMethod(sortArray[i], sortArray[j])){
7.                      object temp = sortArray[i];
8.                      sortArray[i] = sortArray[j];
9.                      sortArray[j] = temp;
10.                 }
11.             }
12.     }
13.     class Employee {
14.         private string _Name;
15.         private decimal _Salary;
16.         public string Name{
17.             get { return _Name; }
18.             set { _Name = value; }
19.         }
20.         public decimal Salary{
21.             get { return _Salary; }
22.             set { _Salary = value; }
23.         }
24.         public Employee(string name, decimal salary) {
25.             this._Salary = salary;
26.             this._Name = name;
27.         }
28.         public override string ToString(){
29.             return string.Format(_Name + ",{0}元 a", _Salary);
30.         }
31.         public static bool Compare(object lhs, object rhs){
32.             Employee empLhs = (Employee)lhs;
33.             Employee empRhs = (Employee)rhs;
34.             return (empLhs._Salary > empRhs._Salary) ? true : false;
35.         }
36.     }
37.     static void Main(string[] args){
38.         Employee[] employees = {
39.             new Employee("王海",2000),
40.             new Employee("张琴",2500),
41.             new Employee("李飞",2300),

```

```

42.         new Employee("赵宇",2800),
43.         new Employee("孙琪",1900),
44.     };
45.     CompareOp compareOp = new CompareOp(Employee.Compare);
46.     BubbleSorter.Sort(employees, compareOp);
47.     for (int i = 0; i < employees.Length; i++)
48.         Console.WriteLine(employees[i].ToString());
49.     Console.ReadLine();
50. }

```

代码说明：第 1 行一个委托 CompareOp，包含两个 object 类型的参数。第 2~12 行定义了一个 BubbleSorter 类。其中，第 13 行定义了一个排序的方法 Sort，它有一个 object 类型的数组参数和一个委托类型的参数。第 4~10 行使用两个 for 的嵌套循环给传递进来的数组元素进行从小到大的排序。

第 13~36 行封装了一个员工类。其中第 14~23 行定义员工类的属性成员。第 24~27 行定义员工类的带参构造函数。第 28 行重写了 ToString 方法，返回员工姓名和工资的信息。第 31 行定义了一个比较的方法对两个员工的工资进行比较。

第 37~50 行定义一个 Main 函数。第 38~44 行实例化了 5 个员工对象并赋值。第 45 行实例化委托对象 compareOp。第 46 行调用 BubbleSorter 类的 Sortd 对员工实例的工作进行排序。第 47~49 行使用 for 循环打印输出结果。

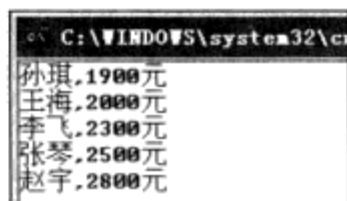


图 2-20 运行结果

03 按下“Ctrl+F5”，程序运行结果如图 2-20 所示。

2.4 泛型

C# 中的泛型类似 C++ 的模板，它在一定程度上能够提高应用程序的效率。使用泛型可以定义类型安全的数据结构，而无需使用具体的数据类型。通过使用泛型，能够将数据类型参数化，以此完成代码重用的目标。

2.4.1 使用系统的泛型类

通常情况下，泛型常见于集合应用中。在 System.Collections.Generic 名称空间中，包含了一些基于泛型的容器类，例如 System.Collections.Generic.Stack、System.Collections.Generic.Dictionary、System.Collections.Generic.List 和 System.Collections.Generic.Queue 等，这些类库可以在集合中实现泛型。

创建泛型的格式如下：

1. 类名 <Type> 变量名 = new 类名 <Type> ();
2. Stack <String> list = new Stack <String> ();

第 1 行是泛型创建的格式，使用泛型类必须指定实际的类型，并在在尖括号 <> 中指定实际的类型。第 2 行根据格式创建了 String 类型的泛型类的集合类型 list，这意味着 list 只能存储 String 类型的数据。

泛型的使用如下代码所示。

```

1. Stack<String> list=new Stack<String>();
2. list.push("ASP");
3. list.push(".NET");
4. int number=list.count;

```

第 1 行使用泛型 `Stack<String> list=new Stack<String>` 创建 `Stack` 的对象 `list`，然后第 2~3 行使用 `Stack` 类的方法 `push` 传入 `String` 类型的参数，这里只能传泛型中定义的类型，否则报错。所以第 4 行使用 `Stack` 类的方法 `count` 取得元素个数。

C# 泛型类在编译时，先生成中间代码 IL，通用类型 `T` 只是一个占位符。在实例化类时，根据用户指定的数据类型代替 `T` 并由即时编译器（JIT）生成本地代码，这个本地代码中已经使用了实际的数据类型，等同于用实际类型写的类。我们把为所有类型参数提供参数的泛型类型称为封闭构造泛型类型，简称封闭类。不同封闭类的本地代码是不一样的，按照这个规则，可以这样认为：泛型类的不同封闭类是不同的数据类型。

2.4.2 创建泛型

除了使用系统的泛型类之外，读者可以编写自己的泛型类。下面介绍一下泛型类和普通类的区别。

1. 静态构造函数

静态构造函数的规则：只能有一个，且不能有参数，它只能被 .NET 运行时自动调用，而不能人工调用。

泛型中的静态构造函数的原理和非泛型类是一样的，只需把泛型中的不同的封闭类理解为不同的类即可。以下两种情况可激发静态的构造函数：

- 特定的封闭类第一次被实例化。
- 特定封闭类中任一静态成员变量被调用。

2. 静态成员变量

在 C# 1.0 中，类的静态成员变量在不同的类实例间是共享的，并且它是通过类名访问的。C# 2.0 中由于引进了泛型，导致静态成员变量的机制出现了一些变化：静态成员变量在相同封闭类间共享，不同的封闭类间不共享。

这也非常容易理解，因为不同的封闭类虽然有相同的类名称，但由于分别传入了不同的数据类型，他们是完全不同的类，比如：

```

List<int> a = new ArrayList<int>();
List<int> b = new ArrayList<int>();
List<long> c = new ArrayList<long>();

```

类实例 `a` 和 `b` 是同一类型，它们之间共享静态成员变量，但类实例 `c` 却是和 `a`、`b` 完全不同的类型，所以不能和 `a`、`b` 共享静态成员变量。

3. 数据类型的约束

在编写泛型类时，总是会对通用数据类型 `T` 进行有意或无意地假想，也就是说 `T` 一般来说是

不能适应所有类型，但怎样限制调用者传入的数据类型呢？这就需要对传入的数据类型进行约束，约束的方式是指定 T 的祖先，即继承的接口或类。因为 C# 的单根继承性，所以约束可以有多个接口，但最多只能有一个类，并且类必须在接口之前。

由于通用类型 T 是从 Object 继承来的，所以他在类 Node 的编写中只能调用 Object 类的方法，这给程序的编写造成了困难。比如你的类设计只需要支持两种数据类型 int 和 string，并且在类中需要对 T 类型的变量比较大小，但这些却无法实现，因为 Object 是没有比较大小的方法的。为了解决这个问题，只需对 T 进行 IComparable 约束，这时在类 Node 里就可以对 T 的实例执行 CompareTo 方法了。这个问题可以扩展到其他用户自定义的数据类型。

如果在类 Node 里需要对 T 重新进行实例化该怎么办呢？因为类 Node 中不知道类 T 到底有哪些构造函数。为了解决这个问题，需要用到 new 约束，需要注意的是，new 约束只能是无参数的，所以也要求相应的类 Stack 必须有一个无参构造函数，否则编译失败。

上面了解了泛型的基本概念和创建泛型的方法，下面通过一个例子来演示如何使用泛型。

【例 2-11】本实例使用泛型类 Stack 实现员工管理系统中录入员工信息的功能，在控制台根据提示输入员工的工号、姓名、年龄和地址，确认录入信息后，显示该员工的信息。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-11”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-11”项目，用鼠标双击网站目录下的“Program.cs”文件，在该文件中编写如下逻辑代码。

```

1.      public class Employee{
2.          private int id;
3.          private string name;
4.          private int age;
5.          private string address;
6.          public Employee() { }
7.          public Employee(int id, String name, int age,string address){
8.              this.id = id;
9.              this.name = name;
10.             this.age = age;
11.             this.address = address;
12.         }
13.         public void print(Stack<Employee> empl){
14.             foreach (Employee s in empl){
15.                 Console.WriteLine("员工的工号是: " + s.id);
16.                 Console.WriteLine("员工的姓名是: " + s.name);
17.                 Console.WriteLine("员工的年龄是:" + s.age);
18.                 Console.WriteLine("员工的地址龄是:" + s.address);
19.             }
20.         }
21.     }
22.     static void Main(string[] args){
23.         Stack<Employee> empl = new Stack<Employee>();
24.         do{
25.             Console.WriteLine("请输入员工的工号: ");
26.             int id = Convert.ToInt32(Console.ReadLine());

```

```

27.         Console.WriteLine("请输入员工的姓名: ");
28.         string name = Console.ReadLine();
29.         Console.WriteLine("请输入员工的年龄: ");
30.         int age = Convert.ToInt32(Console.ReadLine());
31.         Console.WriteLine("请输入员工的家庭住址: ");
32.         string address = Console.ReadLine();
33.         Employee e = new Employee(id, name, age, address);
34.         empl.Push(e);
35.         Console.WriteLine("是否继续输入员工的信息? Y/N");
36.     } while (Console.ReadLine().ToLower() == "y");
37.     Employee a = new Employee();
38.     a.print(empl);
39. }
40. }

```

代码说明：第 1~12 行自定义封装了学生类。其中，第 2~5 行定义了 4 个私有字段表示员工的工号、姓名、年龄和地址。第 6 行定义不带参数的构造函数，第 7~12 行在构造函数内对四个字段进行初始化。第 13~21 行编写一个方法 print 通过 foreach 语句循环打印员工的信息。

第 23 行使用泛型初始化泛型集合对象 empl，对象类型规定为我们定义的 Employee 类型。第 25~32 行获得输入的数据。第 34 行调用 empl 对象的 Push 方法将员工信息添加到泛型集合中。第 38 行调用 Employee 对象的 print 方法打印员工信息。

03 按下“Ctrl+F5”，程序运行结果如图 2-21 所示。



图 2-21 运行结果

2.5 C# 4.0 的新特性

C# 经历几个版本的变革，虽然在大的编程方向和设计理念上没有引起太多的变化，但每次版本更新都带来一些新的特性，这些新特性使程序开发更加方便。本节将介绍几个比较常用的新特性，使读者对这些新特性能有一个大致的印象，具体如何使用还需要读者在实践中不断加深理解。

2.5.1 大整数类型 BigInteger

在 C# 4.0 中增加了一个数据类型 BigInteger，即大整数类型，它位于 System.Numerics 命名空间下。BigInteger 类型是不可变类型，代表一个任意大的整数，它不同于 .NET Framework 中的其他整型，其值在理论上已没有上部或下部的界限。BigInteger 类型的成员与其他整数类型的成员近乎相同。

可通过多种方法实例化 BigInteger 对象。

(1) 用 `new` 关键字并提供任何整数或浮点值以作为 `BigInteger` 构造函数的一个参数。下面的示例阐释如何使用 `new` 关键字实例化 `BigInteger` 值。

```
BigInteger big = new BigInteger(179032.6541);
BigInteger bigInt = new BigInteger(934157136952);
```

以上代码，第 1 行声明了 `BigInteger` 类型的对象 `big`，参数是浮点值，但仅保留小数点之前的整数值。第 2 行声明了 `BigInteger` 类型的对象 `bigInt`，参数是一个大整数。

(2) 声明 `BigInteger` 变量并向其分配一个值，分配的值可以是任何数值，只要该值为整型即可。下面的示例利用赋值从 `Int64` 创建 `BigInteger` 值。

```
long value = 6315489358112;
BigInteger big = longValue;
```

以上代码，第 1 行先声明了一个 `long` 类型的变量 `value` 并赋值。第二行将该值再分配给 `BigInteger` 类型的变量 `big`。

(3) 通过强制类型转换实例化一个 `BigInteger` 对象，使其值可以超出现有数值类型的范围。

```
1. BigInteger big = (BigInteger)179032.6541;
2. BigInteger bigInt = (BigInteger)64312.65m;
```

上面代码中，直接使用强制类型转换的方式声明 `BigInteger` 的对象，仅保留整数部分的值。

可以像使用其他任何整数类型一样使用 `BigInteger` 实例。`BigInteger` 重载标准数值运算符，能够执行基本数学运算，如加法、减法、除法、乘法、减法、求反和一元求反。还可以使用标准数值运算符对两个 `BigInteger` 值进行比较。与其他该整型类型类似，`BigInteger` 还支持按位运算符。对于不支持自定义运算符的语言，`BigInteger` 结构还提供了用于执行数学运算的等效方法。其中包括 `Add`、`Divide`、`Multiply`、`Negate`、`Subtract` 和多种其他内容。

`BigInteger` 结构的许多成员直接对应于 `Math` 类（该类提供处理基元数值类型的功能）的成员。此外，`BigInteger` 增加了自己特有的成员，包括：

- `Sign`: 可以返回表示 `BigInteger` 值符号的值。
- `Abs`: 可以返回 `BigInteger` 值的绝对值。
- `DivRem`: 可以返回除法运算的商和余数。
- `GreatestCommonDivisor`: 可以返回两个 `BigInteger` 值的最大公约数。

【例 2-12】 接受用户输入的数字，计算斐波纳契数。通过本例学习 `BigInteger` 类型的使用以及为何要增加这个新的数据类型。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，在弹出的“新建项目”对话框中先选择“Visual C#”节点下的“Windows”模板，然后选择“控制台应用程序”。在“名称”文本框和“解决方案名称”文本框中输入“例 2-12”，然后在“位置”文本框中输入文件路径，最后单击“确定”按钮。

02 在解决方案资源管理器中生成“例 2-12”项目，用鼠标双击网站目录下的“Program.cs”

文件，在该文件中编写如下逻辑代码。

```

1.  static void Main(string[] args){
2.      do{
3.          Console.WriteLine("请输入数字");
4.          var number = int.Parse(Console.ReadLine());
5.          var value = Fibonacci(number);
6.          Console.WriteLine(value);
7.          BigInteger value1 = Fibonacci1(number);
8.          Console.WriteLine(value1);
9.          Console.WriteLine("是否录继续数字? Y/N");
10.     } while (Console.ReadLine().ToLower() == "y");
11. }
12. public static int Fibonacci(int x){
13.     var previousValue = -1;
14.     var currentResult = 1;
15.     for (var i = 0; i < x; ++i){
16.         var sum = currentResult + previousValue;
17.         previousValue = currentResult;
18.         currentResult = sum;
19.     }
20.     return currentResult;
21. }
22. public static BigInteger Fibonacci1(int x){
23.     var previousValue = new BigInteger(-1);
24.     var currentResult = new BigInteger(1);
25.     for (var i = 0; i < x; ++i){
26.         var sum = currentResult + previousValue;
27.         previousValue = currentResult;
28.         currentResult = sum;
29.     }
30.     return currentResult;
31. }

```

代码说明：第 2~10 行，循环接受用户的输入。其中，第 5、7 行分别调用计算斐波纳契数的两个方法获得普通类型的数值和 `BigInteger` 类型的数值。第 12~21 行定义方法 `Fibonacci` 和第 22~32 行定义的 `Fibonacci1` 方法。二者仅一个地方有所不同，即 13、14 行将方法中需要使用的现值和前值声明为普通类型，而第 23、24 行将方法中使用的前值和现值声明为 `BigInteger` 类型。

03 按下“Ctrl+F5”，程序运行结果如图 2-22 所示。我们发现一个有趣的问题，当输入 47 以下的数字时，两种数据类型的计算结果是相同的。但是当输入 47 以上的数字时，两种数据类型计算的结果却不同，其中，普通类型是错的，而 `BigInteger` 类型是正确。这也就是 `BigInteger` 存在的意义，因为当普通类型数据长度超出它的上限后会出现错误。

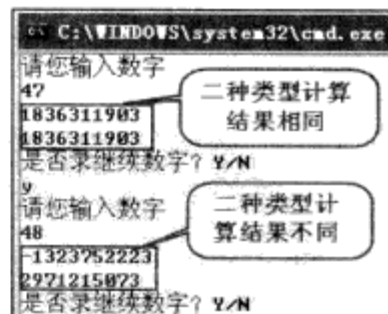


图 2-22 运行结果

2.5.2 动态数据类型 `dynamic`

`dynamic` 是 C# 4.0 引入了一个新的静态类型，它会告诉编译器，在编译期不去检查 `dynamic` 类型，而是在运行时才决定。这表示了不再需要在程序中去声明一个固定的数据类型，而是由 C# 框架自动在执行期间获得数值的类型即可。

在大多数情况下, `dynamic` 类型与 `object` 类型的行为是一样的。但是, 不会用编译器对包含 `dynamic` 类型表达式的操作进行解析或类型检查。编译器将有关该操作信息打包在一起, 并且该信息以后用于计算运行时操作。在此过程中, 类型 `dynamic` 的变量会编译到类型 `object` 的变量中。因此, 类型 `dynamic` 只在编译时存在, 在运行时则不存在。例如下列的代码:

```
1. dynamic v = 124;
2. Console.WriteLine(v.GetType());
```

以上代码, 声明了一个 `dynamic` 类型的对象 `v` 并赋值。第 2 行通过 `GetType` 方法, 输出对象 `v` 的类型。最后显示的结果是 “`System.Int32`”。并且输入对象 `v` 时, Visual Studio 2010 不会出现 `Intellisense` 智能提示, 因为 Visual Studio 2010 不知道 `v` 的数据类型什么, 所以也无法自动提示可用的成员。要使用方法也需要手动输入。同时 `typeof` 方法在 `dynamic` 类型上也无法使用。

`dynamic` 类型和其他数据类型之间, 可以直接做隐式的数据转换, 不论左边是 `dynamic` 或右边是 `dynamic` 都一样, 例如:

```
1. dynamic d1 = 7;
2. dynamic d2 = "a string";
3. dynamic d3 = System.DateTime.Today;
4. int i = d1;
5. string str = d2;
6. DateTime dt = d3;
```

以上代码, 第 1~3 行定义了 3 个 `dynamic` 类型的数据, 分别是整型、字符串类型和时间类型。第 4~6 行再将这些 `dynamic` 类型的数据分别赋给对应的数据类型。



提示

将 `dynamic` 类型和其他数据类型进行转换时。如果二者不兼容, 必须进行显式的强制类型转换, 否则会出现异常

`dynamic` 类型和 `var` 隐形局部变量粗看有些类似, 实则它们有许多不同, 最本质的是 `var` 虽然可以不指定具体的数据类型, 但是它却会在编译时期检查数据类型, 所以使用 `var` 声明的数据不存在时, 编译器会指出编译错误。而且使用 `var` 来声明数据, 其成员也会由智能提示来提供。

2.5.3 命名参数和可选参数

在 C# 3.5 时代, 当我们希望用类似于 C++ 的可选参数为参数指定默认值时, 会得到一个编译器错误, 指示 “不允许参数的默认值”。这个限制是因为在 C# 中, 任何地方都引入面向对象思想, 所以尽量使用重载而不是可选参数。但在 C# 4.0 中这一点得到了改变。

开放命名参数和可选参数是出于动态语言运行时兼容性的要求。动态语言中存在动态绑定的参数列表, 有时候并不是所有的参数值都需要指定; 另外, 在一些 COM 组件互操作时, 往往 COM `Invoke` 的方法参数列表非常长, 例如 `ExcelApplication.Save` 方法可能需要 12 个参数, 但 COM 暴露的参数的实际值往往为 `null`, 只有很少一部分参数需要指定值或者仅仅一个值。这就需要 C# 的编译器能够实现开放命名参数和可选参数。

1. 可选参数

方法、构造函数、索引器或委托的定义可以指定其参数为必选参数还是可选参数。任何调用都必须为所有必选的参数提供参数值，但可以为可选的参数省略参数值。每个可选参数都具有默认值作为其定义的一部分。如果没有为该参数发送参数值，则使用默认值。

可选参数在参数列表的末尾定义，位于任何必选的参数之后。如果调用方为一系列可选参数中的任意一个参数提供了参数值，则它必须为前面的所有可选参数提供参数值。参数值列表不支持使用逗号分隔，例如以下代码。

```
public void ExampleMethod(int required, string optionalstr = "default string", int optionalint = 10)
```

以上代码中，使用一个必选参数和两个可选参数定义实例方法 `ExampleMethod`。其中，`int required` 是必选参数，而由于 `string optionalstr` 和 `int optionalint` 都设置了默认值，所以是可选参数。接下来看一下如何正确调用 `ExampleMethod` 方法。

```
1. ExampleMethod(18, "Hello", "28");
2. ExampleMethod(18);
3. ExampleMethod(optionalstr: "Hello");
4. ExampleMethod(18, , 28);
5. ExampleMethod(18, optionalint: "Hello");
```

以上代码，第 1 行的调用方法正确地对每一个参数都提供了参数值。第 2 行的调用方法仅对必选参数指定参数值，也是正确的。第 3 行的调用方法错误，因为没有给必选参数指定参数值。第 4 行的调用方法错误，参数值列表中不支持使用逗号分隔且为第二个可选参数而不是第一个可选参数提供参数值。第 5 行的调用方法正确，给必选参数和第二个可选参数提供了参数值，其中可选参数指定参数值使用了“参数名：参数值”的正确格式。

2. 命名参数

命名参数让我们可以在调用方法时指定参数名字来给参数赋值，这种情况下可以忽略参数的顺序。利用命名参数，可以为特定参数指定参数值，方法是将参数值与该参数的名称关联，而不是与参数在参数列表中的位置关联。

命名参数的语法为：

```
参数名称 1: 参数值 1, 参数名称 2: 参数值 2...
```

有了命名参数，我们将不再需要记住或查找参数在所调用方法的参数列表中的顺序。可以按参数名称指定参数值，例如：

```
public int MyFunction(int ArgA, int ArgB, int ArgC)
```

上面的代码定义了一个方法，有三个 `int` 类型的参数列表。根据命名参数的规则，我们可以按以下的方法去调用。

```
MyFunction(ArgA: 8, ArgB: 18, ArgC:28);
MyFunction(ArgB: 18,ArgA: 8, ArgC:28);
MyFunction(ArgC:28, ArgB: 18, ArgA: 8);
```

以上代码中三种调用方法突破了以前需要按照参数列表中顺序进行指定实参的限制，如果不

记得参数的顺序，但却知道其名称，我们可以按任意顺序发送参数值。

不过要记住的是命名参数和可选参数虽然非常的好用，但是绝对不要滥用，否则会对程序的可读性造成相当大的伤害。

2.6 上机题

1. 编写一个求质数的控制台应用程序。要求定义一个类来求取指定范围内的所有质数，然后在主程序中创建一个该类的实例，调用该类的方法输出所有的质数。

2. 定义一个形状类，然后派生两个子类：圆形、矩形。再定义一个长方形继承于矩形。在程序中使用委托的方法来显示这三种几何形状的类型和面积。

3. 对用户输入的文本进行分析。程序在控制台统计元音、辅音、字母、数字和单词的个数。

4. 设计一个二维数组，从控制台接受输入，存放五位学生的三门课的成绩，最后求得每位学生三门课的平均成绩。

5. 设计一个程序，在控制台输出三角形状的九九乘法表。

6. 定义一个 `Person` 的类，其中含有四个私有的数据成员：`name`、`age`、`id`。两个公有函数方法 `display` 和 `person`，前者用于打印 `Employee` 对象的基本信息；后者为构造函数。最后在 `Main` 函数中测试程序。

7. 在控制台接受用户输入的数字，然后将数字使用冒泡排序的方法进行排列，并将排序后的结果输出到控制台显示。

8. 根据用户输入的数字（1~12 之间），输出相应的月份所具有的天数。要求使用 `switch` 结构，并使用 2、3、10 和 -1 作为测试数据。

9. 企业发放奖金的根据是利润的提成。用 i 表示利润，利润低于或等于 10 万元 ($i \leq 100000$)，奖金可提成 10%；利润高于 10 万元，低于 20 万元 ($100000 < i \leq 200000$) 时，可以提 7.5%。200000 $< i \leq 400000$ 时，按 5% 提成；400000 $< i \leq 600000$ 时，高于 40 万的部分按 3% 提成；600000 $< i \leq 1000000$ 时，高于 60 万按 1.5% 提成； $i > 1000000$ 时，超过 100 万元的部分按 1% 提成。从键盘输入企业利润 i ，求应发的奖金总数。

第 3 章 ASP.NET 4.0 常用内置对象

学习目标

ASP.NET 4.0 中包含了大量的对象类库，我们在 Web 开发中完成的许多工作都要用到由这些类定义的对象，它提供了大量现成的功能供开发人员使用。同时，在应用程序中还经常会用到这些对象来维护程序的相关信息。本章所介绍的 Page 类、Request 对象、Response 对象和 Server 对象就是主要用来连接服务器和客户端浏览器之间的联系，而 Cookie 对象、Session 对象和 Application 对象则主要用于网站状态管理。掌握这些对象有利于读者站在系统的角度来构建 Web 应用程序。

本章重点

- 理解页面的生命周期
- Session 对象的常用属性和方法
- 掌握 Cookie 的应用
- 在页面中使用 Request 对象传递值

3.1 Page 类

在 ASP.NET Framework 中，Page 类为 ASP.NET 应用程序文件所构建的对象提供基本行为。该类在命名空间 System.Web.UI 中定义，从 TemplateControl 中派生出来，而 TemplateControl 类继承自 System.Web.UI.Control，它也是一种特殊的 Control 并实现了 IHttpHandler 接口。

3.1.1 页面的生命周期

在我们的项目中，所有的 Web 页面都继承与 System.Web.UI.Page 类，要了解 Page 类，必须知道 ASP.NET 页面的工作过程。

01 客户端浏览器向 Web 应用程序进行一个页面的请求。

02 服务器端 Web 应用程序接收到这个请求，先查看这个页面是否被编译过，如果没有被编译过，就编译这个 Web 页面，然后对这个页面进行实例化产生一个 Page 对象。

03 Page 对象根据客户请求，把信息返回给 IIS，然后信息由 IIS 返回给客户端浏览器。

在这个过程中，每个页面都被编译成一个类，当有请求的时候就对这个类进行实例化。对于页面生命周期，一共要关心如下 5 个阶段。

① **页面初始化**：在这个阶段，页面及其控件被初始化。页面判断这是一个新的请求还是一个回传请求。页面事件处理器 Page_PreInit 和 Page_Init 被调用。另外，任何服务器控件的 PreInit 和 Init 被调用。

② **载入**：如果请求是一个回传请求，控件属性使用从视图状态和控件状态的特殊页面状态容器中恢复的信息来载入。页面的 `Page_Load` 方法以及服务器控件的 `Page_Load` 方法事件被调用。

③ **回送事件处理**：如果请求是一个回传请求，任何控件的回发事件处理器被调用。

④ **呈现**：在页面呈现状态中，视图状态保存到页面，然后每个控件及页面都是把自己呈现给输出相应流。页面和控件的 `PreRender` 和 `Render` 方法先后被调用。最后，呈现的结果通过 HTTP 响应发送回客户机。

⑤ **卸载**：对页面使用过的资源进行最后的清除处理。控件或页面的 `Unload` 方法被调用。

3.1.2 Page 类的主要属性、方法和事件

`Page` 类与扩展名为 `.aspx` 的文件相关联，这些文件在运行时被编译为 `Page` 对象，并缓存在服务器内存中。如果要使用代码隐藏技术创建 Web 窗体页，需要从该类派生。应用程序快速开发（RAD）设计器（如 Microsoft Visual Studio）自动使用此模型创建 Web 窗体页。`Page` 对象充当页中所有服务器控件的容器。

在单文件页中，标记、服务器端元素以及事件处理代码全都位于同一个 `.aspx` 文件中。在对该页进行编译时，如果存在使用 `@Page` 指令的 `Inherits` 属性定义的自定义基类，编译器将生成和编译一个从该基类派生的新类，否则编译器将生成和编译一个从 `Page` 基类派生的新类。例如，如果在应用程序的根目录中创建一个名为 `SamplePage1` 的新 ASP.NET 网页，则随后将从 `Page` 类派生一个名为 `ASP.SamplePage1.aspx` 的新类。对于应用程序子文件夹中的页，将使用子文件夹名称作为生成的类的一部分。生成的类中包含 `.aspx` 页中的控件的声明以及用户添加的事件处理程序和其他自定义代码。

在生成页之后，生成的类将编译成程序集，并将该程序集加载到应用程序域，然后对该类进行实例化并执行该类，将输出呈现到浏览器。如果对生成类的页进行更改（无论是添加控件还是修改代码），则已编译的类代码将失效，并生成新的类。

在代码隐藏模型中，页的标记和服务端元素（包括控件声明）位于 `.aspx` 文件中，而用户定义的页代码则位于单独的代码文件中。该代码文件包含一个分部类，即具有关键字 `partial` 的类声明，以表示该代码文件只包含构成该页的完整类的全体代码的一部分。在分部类中，添加应用程序要求该页所具有的代码。此代码通常由事件处理程序构成，但是也可以包括用户需要的任何方法或属性。

代码隐藏页的继承模型比单文件页的继承模型要稍微复杂一些，模型说明如下：

- 代码隐藏文件包含一个继承自基页类的分部类。基页类可以是 `Page` 类，也可以是从 `Page` 派生的其他类。
- `.aspx` 文件在 `@Page` 指令中包含一个指向代码隐藏分部类的 `Inherits` 属性。
- 在对该页进行编译时，ASP.NET 基于 `.aspx` 文件生成一个分部类；此类是代码隐藏类文件的分部类。生成的分部类文件包含页控件的声明。使用此分部类，用户可以将代码隐藏文件用作完整类的一部分，而无需显式声明控件。
- 最后，ASP.NET 生成另外一个类，该类从上一步骤中生成的类继承而来。它包含生成该页所需的代码。该类和代码隐藏类将编译成程序集，运行该程序集可以将输出呈现到浏览器。

下面用表格介绍 `Page` 类的常见属性和方法，如表 3-1 所示。

表 3-1 Page 类的重要属性和方法

属性和方法	说明
Application	为当前 Web 请求获取 <code>HttpApplicationState</code> 对象
IsPostBack	指示该页是否正为响应客户端回发而加载，或者它是否正被首次加载和访问
IsValid	指示页验证是否成功
Request	获取请求的页的 <code>HttpRequest</code> 对象
Response	获取与该 Page 对象关联的 <code>HttpResponse</code> 对象
Server	获取 <code>Server</code> 对象，它是 <code>HttpServerUtility</code> 类的实例
Session	获取 ASP.NET 提供的当前 <code>Session</code> 对象
Validators	获取请求的页上包含的全部验证控件的集合
ViewState	获取状态信息的字典，这些信息使用户可以在同一页的多个请求间保存和还原服务器控件的视图状态
MapPath(virtualPath)	将 <code>virtualPath</code> 指定的虚拟路径转换成实际路径
ResolveUrl(relativeUrl)	将 <code>virtualPath</code> 指定的虚拟路径转换成实际路径
DataBind()	将数据源连接到网页上的服务器控件
Dispose()	将数据源连接到网页上的服务器控件
FindControl(id)	在页面上搜索标识名称为 <code>id</code> 的控件
Validate()	执行网页上的所有验证控件
HasControls()	判断 Page 对象是否包含控件

Page 类中有很多属性是对象的引用，比如上表中的 `Request`、`Response`、`Application` 和 `Session` 等，这样在页面中可以直接对这些对象进行访问，而无需通过 Page 对象。比如下面两行代码的作用是一样的。

```
1. Page.Response.Redirect("Default.aspx");
2. Response.Redirect("Default.aspx");
```

第 1 行代码是通过 Page 对象的 `Response` 属性得到 `Response` 对象的引用，第二行直接通过 `Response` 对象名对 `Response` 对象进行引用。

Page 类除了属性和方法外，还有八个常见的事件如表 3-2 所示。

表 3-2 Page 类的主要事件

事件名称	说明
PreInit	在页初始化开始前发生，是网页执行时第一个被触发的事件
PreLoad	在信息被写入到客户端前会触发此事件
Load	当网页被加载时会触发此事件
Init	在网页初始化开始时发生
PreRender	在信息被写入到客户端前会触发此事件
Unload	网页完成处理并且信息被写入到客户端后触发此事件
InitComplete	在页面初始化完成时发生
LoadComplete	在页面生命周期的加载阶段结束时发生

上表中 Page 对象的事件贯穿于网页执行的整个过程。在每个阶段，ASP.NET 都触发了可以在代码中处理的事件，对于大多数情况，我们只需要关心 `Page_Load` 事件。该事件的两个参数是由 ASP.NET 定义的，第一个参数定义了产生事件的对象，第二个是传递给事件的详细信息。每次触发服务器控件的时候，页面都会去执行一次 `Page_Load` 事件，说明页面被加载了一次。这个技术称

为回传（或者称为回送）技术，这个技术是 ASP.NET 最为重要的特性之一。这样，Web 页面就好像一个 Windows 窗体一样。在 ASP.NET 中，当客户端触发了一个事件，它不是在客户端浏览器上对事件进行处理，而是把该事件的信息传送回服务器进行处理。服务器在接收到这些信息后，会重新加载 Page 对象，然后处理该事件，所以 Page_Load 事件被再次触发。

由于 Page_Load 在每次页面加载时运行，因此其中的代码即使在回传的情况下也会运行，在这个时候 Page 的 IsPostBack 属性就可以用来解决这个问题，因为这个属性是用来识别 Page 对象是否处于一个回送的状态下，也就弄清楚是请求页面的第一个实例，还是请求回送原来的页面。可以在 Page 类的 Page_Load 事件中使用该属性，以便数据访问代码只在首次加载页面时运行，具体代码如下所示。

```
1. protected void Page_Load (object sender,EventAge e) {
2.     if(!IsPostBack){
3.         // 需要执行的代码
4.     }
5. }
```

代码说明：第 1 行处理 Page 页面的加载事件 Load，第 2 行使用 Page 的 IsPostBack 属性判断当前加载的页面是否是回送页面。

3.1.3 应用 Page 类

以上两个小节讲述了 Page 类的概念及其属性、方法和事件，本小节将通过一个例子来讲述 Page 类的使用。

【例 3-1】使用 Page 对象的 Init 事件，根据用户输入的爱好，将内容添加到网页的列表控件中显示。Init 事件主要用来设置网页或者控件的初始值。同一个网页只会触发一次 Init 事件。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，打开如图 3-1 所示的“新建项目”对话框。打开“Visual C#”类型节点，选择“Web”子节点这个模板，同时在右边窗口中选择“ASP.NET 空 Web 应用程序”，在“名称”文本框中输入“例 3-1”，并在“位置”文本框中输入相应的存储路径。最后，单击“确定”按钮。

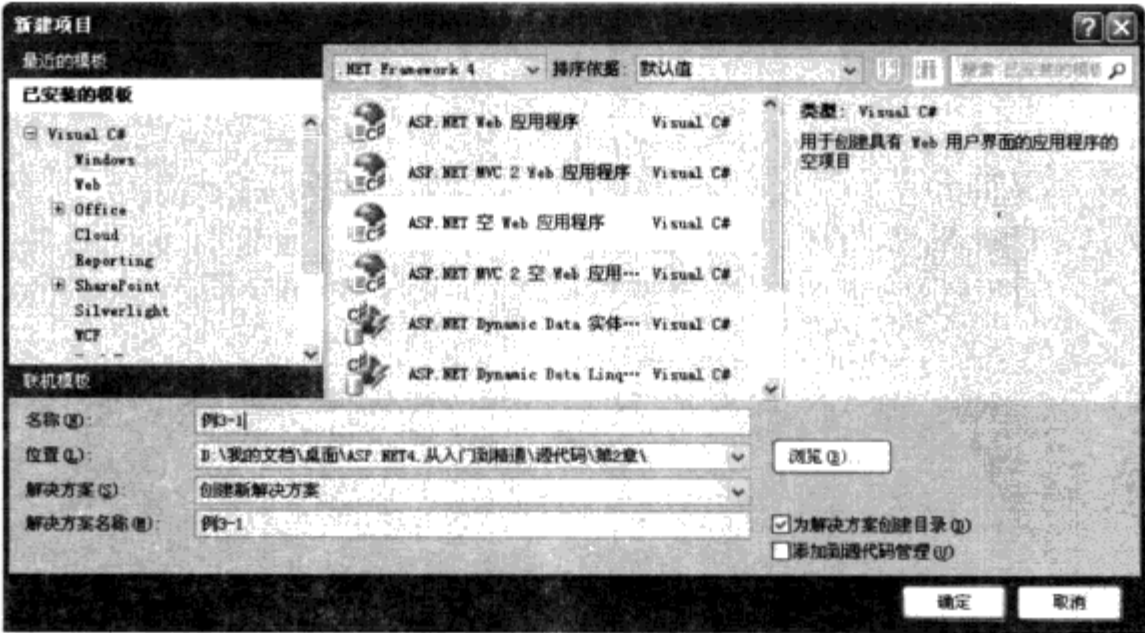


图 3-1 新建项目对话框

02 在解决方案资源管理器中的项目名称上右键单击，在弹出的快捷菜单中选择“添加”|“新建项”命令，打开如图 3-2 所示的“添加新项”对话框，在该对话框中选择“已安装模板”下的“Web”模板，并在模板文件列表中选中“Web 窗体”，然后在“名称”文本框输入该文件的名称“Default.aspx”，最后单击“添加”按钮即可向网站项目中添加一个新的文件。

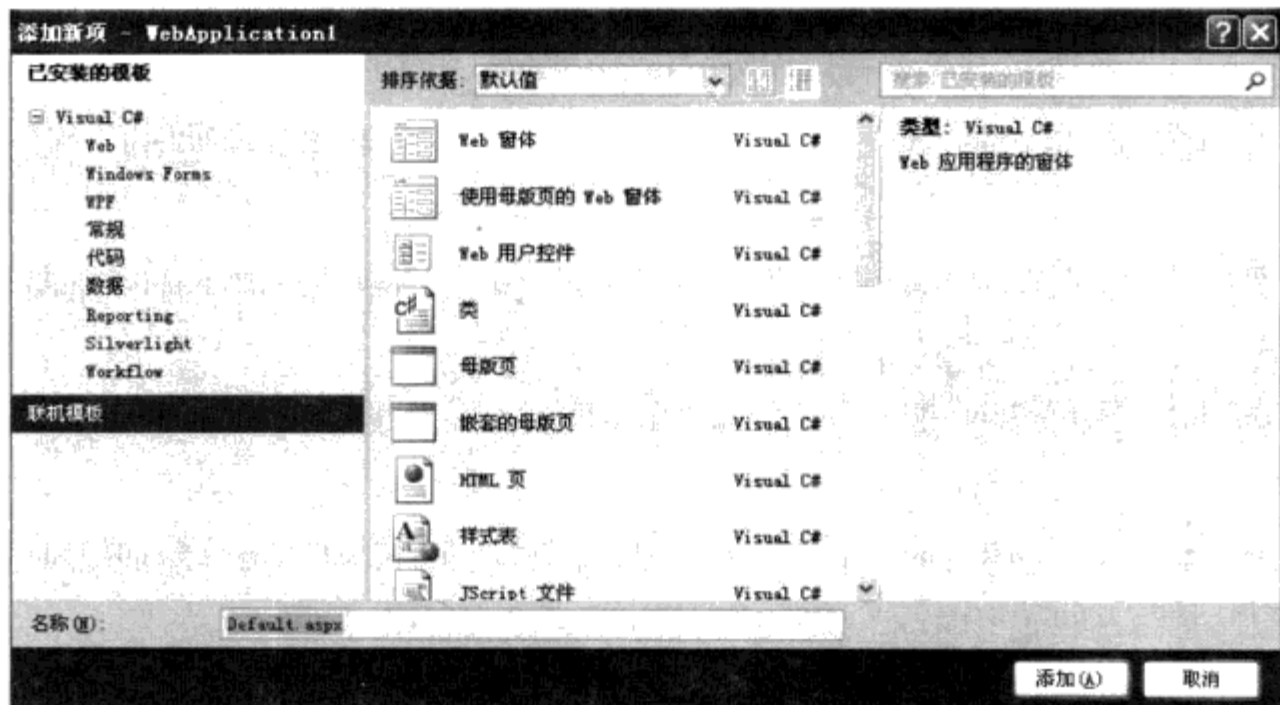


图 3-2 添加新项对话框

03 用鼠标双击网站的根目录下刚才添加的“Default.aspx”文件，进入“视图编辑界面”打开“源视图”编辑区，在<html></html>标记之间编写如下代码。

```

1.  <head runat="server"><title></title>
2.      <script language="c#" runat="server">
3.          void Page_Init(Object sender, EventArgs e){
4.              Interest.Items.Add("文学");
5.              Interest.Items.Add("旅游");
6.              Interest.Items.Add("音乐");
7.          }
8.          void AddToList_Click(Object sender, EventArgs e){
9.              Interest.Items.Add(Text1.Value);
10.         }
11.     </script>
12. </head>
13. <body>
14.     <form id="form1" runat="server">
15.         <div>
16.             请选择你的爱好?<br />
17.             <select id="Interest" runat="server"></select>
18.             <p>向列表中添加爱好</p>
19.             <input type="text" id="Text1" runat="server" />
20.             <input type="button" runat="server" value="添加" onserverclick="AddToList_Click" />
21.         </div>
22.     </form>
23. </body>

```

代码说明：第 2~11 行添加了一段 JavaScript 代码。其中，第 3~7 行处理 Page 页面 Init 事件的代码；第 4~7 行通过下拉列表控件 Interest 的 Items.Add 方法添加列表项内容。第 8~10 行处理按钮

控件 AddToList 的事件 Click 的代码。其中，第 9 行使用下拉列表控件 Interest 的 Items.Add 方法将用户输入在文本框中的内容添加到列表项。第 17 行在表单中添加一个下拉类别列表控件 Interest。第 19 行添加一个文本框控件 Text1。第 20 行添加一个按钮控件并设置其单击事件为“AddToList_Click”。

- 04
- 程序运行结果如图 3-3 所示，展开下拉列表可以看到有三个列表项内容。
- 05
- 用户在文本框输入“体育”，单击“添加”按钮后，列表中多出了如图 3-4 所示的“体育”项。



图 3-3 运行结果 1



图 3-4 运行结果 2



在 ASP.NET 程序中可以编写事件处理程序来响应 Page 事件，其事件处理顺序依次是 Page_PreInit、Page_Init、Page_PreLoad、Page_Load、Page_PreRender 和 Page_Unload。

3.2 Request 对象

Request 对象是 System.Web.HttpRequest 类的实例。当用户在客户端使用 Web 浏览器向 Web 应用程序发出请求时，就会将客户端的信息发送到 Web 服务器。Web 服务器就接收到一个 HTTP 请求，它包含了所有查询字符串参数或表单参数、Cookie 数据以及浏览器的信息。在 ASP.NET 中运行时把这些客户端的请求信息封装成 Request 对象。

3.2.1 Request 对象的属性和方法

要掌握 Request 对象的使用，必须了解它的常用属性和方法。Request 对象的常用属性和方法如表 3-3 所示。

表 3-3 Request 对象的常用属性和方法

属性和方法	说明
AcceptTypes	获取客户端支持的 MIME 接受类型的字符串数组
ApplicationPath	获取服务器上 ASP.NET 应用程序的虚拟应用程序根路径
Browser	获取有关正在请求的客户端的浏览器功能的信息
Cookies	获取客户端发送的 cookie 的集合
CurrentExceptionFilePath	获取或设置输出流的 HTTP 字符集
FilePath	获取当前请求的虚拟路径
Files	获取客户端上载的文件（多部件 MIME 格式）集合

(续表)

属性和方法	说明
Form	获取窗体变量集合
Headers	获取 HTTP 头集合
InputStream	获取传入的 HTTP 实体主体的内容
Item	获取 Cookies、Form、QueryString、ServerVariables 集合中指定的对象。在 C# 中，该属性为 HttpRequest 类的索引器
Path	获取当前请求的虚拟路径
PathInfo	获取具有 URL 扩展名的资源的附加路径信息
PhysicalPath	获取与请求的 URL 相对应的物理文件系统路径
QueryString	获取 HTTP 查询字符串变量集合
RawUrl	获取当前请求的原始 URL
ServerVariables	获取 Web 服务器变量的集合
BinaryRead	执行对当前输入流进行指定字节数的二进制读取
MapImageCoordinates	将传入图象字段窗体参数影射为适当的 x/y 坐标值
MapPath	为当前请求将请求的 URL 中的虚拟路径映射到服务器上的物理路径
SaveAs	将 HTTP 请求保存到磁盘
ValidateInput	验证由客户端浏览器提交的数据，如果存在具有潜在危险的数据，则引发一个异常
Url	获取有关当前请求的 URL 的信息

3.2.2 应用 Request 对象

上一小节我们介绍了 Request 对象的概念和主要的属性以及它的常用方法。为了加深理解本小节通过一个例子来介绍 Request 对象中的事件和方法的具体使用。

【例 3-2】本例是一个简单的用户登录界面，要求用户输入用户名和密码信息，然后跳转到另一个页面，并在该页面中显示刚才所输入的用户名和密码。

01 启动 Visual Studio 2010，执行“文件”|“新建项目”命令，打开如图 3-1 所示的“新建项目”对话框。打开“Visual C#”类型节点，选择“Web”子节点这个模板，同时在右边窗口中选择“ASP.NET 空 Web 应用程序”，在“名称”文本框中输入“例 3-2”，并在“位置”文本框中输入相应的存储路径。最后，单击“确定”按钮。

02 在解决方案资源管理器中的项目名称上右键单击，在弹出的快捷菜单中选择“添加”|“新建项”命令，打开“添加新项”对话框，在该对话框中选择“已安装模板”下的“Web”模板，并在模板文件列表中选“Web 窗体”，然后在“名称”文本框输入该文件的名称“Default.aspx”，最后单击“添加”按钮。

03 使用相同的方法在例 3-2 中添加一个名为“New.aspx”的页面。

04 用鼠标双击网站的根目录下刚才添加的“Default.aspx”文件，进入“视图编辑界面”打开“源视图”编辑区，在<form></form>标记之间编写如下代码。

```

1. 用户名: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
2. <br />
3. 密 码: <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
4. <br />
5. <asp:Button ID="Button1" runat="server" Text="登录" onclick="Button1_Click" />

```

代码说明: 第 1、3 行分别添加两个文本框服务器控件 TextBox1 和 TextBox2。第 5 行添加一个按钮服务器控件 Button1 并设置其单击事件为 Click。

05 用鼠标双击网站目录下的“Default.aspx.cs”文件, 编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2.     Response.Redirect("New.aspx?username="+TextBox1.Text+"&password="+TextBox2.Text);
3. }

```

代码说明: 第 1 行定义登陆按钮 Button1 的单击事件 Click。第 2 行将用户输入的用户名和密码通过 URL 地址的形式传递到显示的页面 New.aspx。

06 用鼠标双击网站目录下的“New.aspx.cs”文件, 编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2.     Response.Write("用户名: "+Request.QueryString["username"]+"<br>");
3.     Response.Write("密 码: "+Request.QueryString["password"]+"<br>");
4. }

```

代码说明: 第 1 行定义 New 页面对象的加载事件 Load。第 2、3 行分别使用 Request 对象的 QueryString 的属性来获得 URL 地址中传递的用户名和密码。

07 按下“Ctrl+F5”, 运行结果如图 3-5 所示, 在文本框中输入“用户名”和“密码”内容, 单击“登录”按钮。

08 在如图 3-6 所示的 New 页面中显示用户名和密码的具体内容。

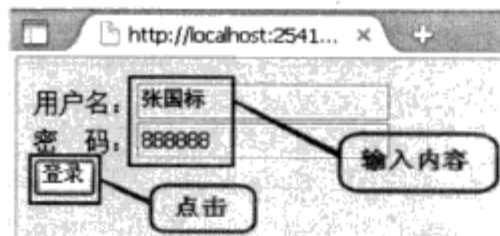


图 3-5 运行结果 1

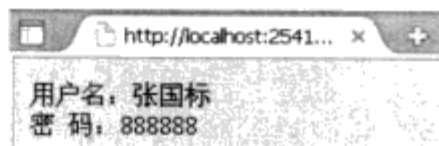


图 3-6 运行结果 2



提示

在 Request 对象的调用方法中, QueryString 集合主要用于收集 HTTP 协议中的 Get 请求发送的数据, 如果一个请求事件中被请求的程序 Url 中出现“?”号后的数据, 则表示此次请求方式为 Get。而 Form 集合用于收集 Post 方法发送的请求数据, Post 请求必须由 Form 来发送。

3.3 Response 对象

Response 对象是 System.Web.HttpResponse 类的实例, Response 对象封装了 Web 服务器对客户

端请求的响应，它用来操作 HTTP 相应的信息，用于将结果返回给请求者。虽然 ASP.NET 中控件的输出不需要我们去写 HTML 代码，但是在很多时候我们希望能自己手动控制输出流，比如文件的下载、重定向、脚本输出等。

3.3.1 Response 对象的属性

要想掌握好 Response 对象的使用，必须先熟悉它的常用属性，Response 的主要属性如表 3-4 所示。

表 3-4 Response 对象的常用属性

属性	说明
Buffer	获取或设置一个值，该值指示是否缓冲输出，并在完成处理整个响应之后将其发送
BufferOutput	获取或设置一个值，该值指示是否缓冲输出，并在完成处理整个页之后将其发送
Cache	获取 Web 页的缓存策略（过期时间、保密性、变化子句）
CacheControl	将 Cache-Control HTTP 头设置为 Public 或 Private
Charset	获取或设置输出流的 HTTP 字符集
ContentEncoding	获取或设置输出流的 HTTP 字符集
ContentType	获取或设置输出流的 HTTP MIME 类型
Cookies	获取响应 Cookie 集合
Expires	获取或设置在浏览器上缓存的页过期之前的分钟数。如果用户在页过期之前返回同一页，则显示缓存的版本
ExpiresAbsolute	获取或设置将缓存信息从缓存中移除时的绝对日期和时间
Filter	获取或设置一个包装筛选器对象，该对象用于在传输之前修改 HTTP 实体主体
IsClientConnected	获取一个值，通过该值指示客户端是否仍连接在服务器上
Output	启用到输出 HTTP 响应流的文本输出
OutputStream	启用到输出 HTTP 内容主体的二进制输出
RedirectLocation	获取或设置 HTTP “位置” 标头的值
Status	设置返回到客户端的 Status 栏
StatusCode	获取或设置返回给客户端的输出的 HTTP 状态代码
StatusDescription	获取或设置返回给客户端的输出的 HTTP 状态字符串
SuppressContent	获取或设置一个值，该值指示是否将 HTTP 内容发送到客户端

3.3.2 Response 对象的方法

仅仅是了解 Response 对象的属性还远远不够，Response 对象还提供了一些非常实用的方法供我们在编写程序中使用。

（1）Response 对象的 Redirect 方法可以将客户端重定向到新的 URL，其语法定义如下所示。

```
1. public void Redirect(string url);
2. public void Redirect( string url, bool endResponse);
```

代码说明：方法中 url 参数为要重新定向的目标网址，参数 endResponse 指示当前页的执行是否应终止。

(2) Write 方法用于将信息写入 HTTP 响应输出流，输出到客户端显示，其语法定义如下所示。

```
1. public void Write(char[], int, int);
2. public void Write(string);
3. public void Write(object);
4. public void Write(char);
```

代码说明：从上面的四个方法的参数可以看出，通过 Write 方法可以把字符数组，字符串，对象，或者一个字符输出显示。

如果把指定的文件直接写入 HTTP 响应输出流，需要调用 WriteFile 方法，其语法定义如下所示。

```
1. public void WriteFile(string filename);
2. public void WriteFile(string filename, long offset, long size);
3. public void WriteFile(IntPtr fileHandle, long offset, long size);
4. public void WriteFile(string filename, bool readIntoMemory);
```

代码说明：参数 filename 为要写入 HTTP 输出流的文件名；参数 offset 为文件中将开始进行写入的字节位置；参数 size 为要写入输出流的字节数（从开始位置计算）。参数 fileHandle 是要写入 HTTP 输出流的文件的文件句柄；参数 readIntoMemory 指示是否将把文件写入内存块。

(3) 下面是其他几个 Response 对象的方法定义。

- BinaryWrite: 将一个二进制字符串写入 HTTP 输出流。
- Clear: 清除缓冲区流中的所有内容输出。
- ClearContent: 清除缓冲区流中的所有内容。
- ClearHeaders: 清除缓冲区流中的所有头信息。
- Close: 关闭到客户端的套接字连接。
- End: 将当前所有缓冲的输出发送到客户端，停止该页的执行，并引发 Application_EndRequest 事件。
- Flush: 向客户端发送当前所有缓冲的输出。Flush 方法和 End 方法都可以将缓冲的内容发送到客户端显示，但是 Flush 与 End 的不同之处在于，Flush 不停止页面的执行。

3.3.3 应用 Response 对象

上面介绍了 Response 对象的概念以及它的常用方法和属性。本小节将结合一个例子来讲解 Response 对象在实际中的使用，以便使读者能够快速入门。

【例 3-3】实现 Response 对象的重定向跳转页面功能，在页面的下拉列表中选择需要访问网站的地址，单击确定按钮后，浏览器显示该网站的页面。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 3-3”。

02 用鼠标双击网站的根目录下“Default.aspx”文件，进入到“视图设计器”。选择“源视图”，在<form></form>之间编写代码如下。

```
1. 请选择你要登录的网站
2. <asp:DropDownList ID="DropDownList1" runat="server">
```

```

3. <asp:ListItem Value="0">谷歌</asp:ListItem>
4. <asp:ListItem Value="1">百度</asp:ListItem>
5. <asp:ListItem Value="2">搜狗</asp:ListItem>
6. </asp:DropDownList>
7. <br /><br />
8. <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="确定" />

```

代码说明：第 2 行添加了一个下拉列表服务器控件 DropDownList1。第 3~5 行给 DropDownList1 添加三个列表项。第 8 行添加了一个按钮服务器控件 Button1 并设置单击事件为 Click。

03 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2.     if (DropDownList1.SelectedIndex == 0)
3.         Response.Redirect("http://www.google.com");
4.     if (DropDownList1.SelectedIndex == 1)
5.         Response.Redirect("http://www.baidu.com");
6.     if (DropDownList1.SelectedIndex == 2)
7.         Response.Redirect("http://www.sougou.com");
8. }

```

代码说明：第 1 行处理按钮 Button1 的单击事件 Click。第 2、4 和 6 行分别判断如果选择的列表项索引是 0、1 和 2。在第 3、5 和 7 行分别使用 Response 对象的 Redirect 方法跳转到相应网站的 URL 地址。

04 按下“Ctrl+F5”，运行结果如图 3-7 所示，展开下拉列表选择“谷歌”，然后单击“确定”按钮。

05 浏览器显示如图 3-8 所示的谷歌网站首页。

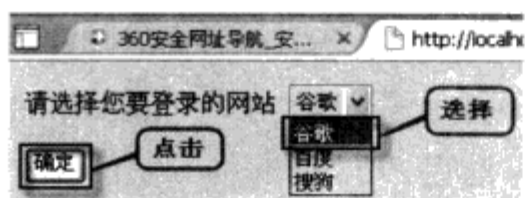


图 3-7 运行结果 1



图 3-8 运行结果



提示

使用 Response.Redirect 方法时重定向操作发生在客户端，总共涉及到两次与服务器的通信（两个来回）：第一次是对原始页面的请求，得到一个 302 应答，第二次是请求 302 应答中声明的新页面，得到重定向之后的页面。

3.4 Server 对象

Server 对象是 System.Web.HttpServerUtility 类的实例，它包含了一些与服务器相关的信息。使用它可以获得有关最新的错误信息、对 HTML 文本进行编码和解码、访问和读写服务器端的文件等功能。

3.4.1 Server 对象的属性和方法

Server 对象提供许多访问的方法和属性帮助程序有序地执行，Server 对象常用属性定义如表 3-5 所示。

表 3-5 Server 对象的常用属性和方法

方法和属性	说明
Execute	在当前请求的上下文中执行指定虚拟路径的处理程序
GetLastError	返回前一个异常
HtmlDecode	对 HTML 编码的字符串进行解码，并将解码出发送到 System.IO.TextWriter 输出流
HtmlEncode	对字符串进行 HTML 编码，并将解码输出发送到 System.IO.TextWriter 输出流
MapPath	返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径
Transfer	终止当前页的执行，并为当前请求开始执行新页
UrlDecode	对字符串进行解码，该字符串为了进行 HTTP 传输而进行编码并在 URL 中发送到服务器
ScriptTimeout	获取和设置请求超时（以秒计）
MachineName	获取服务器的计算机名称
UrlEncode	编码字符串，以便通过 URL 从 Web 服务器到客户端进行可靠的 HTTP 传输
UrlPathEncode	对 URL 字符串的路径部分进行 URL 编码，并返回已编码的字符串

Server 对象的 GetLastError 方法可以获得前一个异常，当发生错误时可以通过该方法访问错误信息。例如：

```
Exception LastError = Server.GetLastError();
```

Server 对象的 Transfer 方法用于终止当前页的执行，并为当前请求开始执行新页，其语法定义如下所示。

```
1. public void Transfer( string path);
2. public void Transfer(string path, bool preserveForm);
```

代码说明：参数 path 是服务器上要执行的新页的 URL 路径。参数 preserveForm 如果为 true，则保存 QueryString 和 Form 集合，否则就清除它们（默认为 false）。

Server 对象 MapPath 方法应用返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径，其语法定义如下所示。

```
public string MapPath( string path);
```

代码说明：参数 path 是 Web 服务器上的虚拟路径，返回值是与 path 相对应的物理文件路径。MapPath 是一个非常有用的方法。

Server 对象的 HtmlEncode 方法用于对要在浏览器中显示的字符串进行编码，其语法定义如下所示。

```
1. public string HtmlEncode(string s);
2. public void HtmlEncode(string s, TextWriter output);
```

代码说明：参数 *s* 是要编码的字符串。Output 是 TextWriter 输出流，包含已编码的字符串。例如希望在页面上输出 “<p></p> 标签用于分段”，通过代码 `Response.Write("<p></p> 标签用于分段")` 输出后，则结果并非是这个字符串，其中的 <P> 和 </P> 并没有当做 HTML 元素来解析，为了能够输出自己希望的结果，这里可以使用 `HtmlEncode` 方法对字符串进行编码，然后再通过 `Response.Write` 方法输出。

Server 对象的 `HtmlDecode` 方法用于对已进行 HTML 编码的字符串进行解码，是 `HtmlEncode` 方法的反操作，其语法定义如下所示。

```
1. public string HtmlDecode(string s);
2. public void HtmlDecode( string s, TextWriter output);
```

代码说明：参数 *s* 是要解码的字符串。Output 是 TextWriter 输出流，包含已解码的字符串。下面的代码可以把已经过 HTML 编码的字符串进行还原。

Server 对象的 `UrlEncode` 方法用于编码字符串，以便通过 URL 从 Web 服务器到客户端进行可靠的 HTTP 传输。`UrlEncode` 方法的语法定义如下所示。

```
1. public string UrlEncode( string s);
2. public void UrlEncode(string s, TextWriter output);
```

代码说明：参数 *s* 是要编码的字符串。Output 是 TextWriter 输出流，包含已编码的字符串。

Server 对象的 `UrlDecode` 方法用于对字符串进行解码，该字符串为了进行 HTTP 传输而进行编码，并在 URL 中发送到服务器。`UrlDecode` 方法的语法定义如下所示。

```
1. public string UrlDecode(string s);
2. public void UrlDecode(string s, TextWriter output);
```

代码说明：参数 *s* 是要解码的字符串。Output 是 TextWriter 输出流，包含已解码的字符串。

`UrlDecode` 方法是 `UrlEncode` 方法的逆操作，可以还原被编码的字符串。

3.4.2 应用 Server 对象

上面介绍了 Server 对象的概念以及它的常用方法和属性。本小节用一个例子来介绍 Server 对象的属性和方法在实际中的使用。

【例 3-4】 通过 Server 对象的 `GetLastError` 方法获得最近遇到错误产生的异常对象。这个方法在应用事件处理中应用很广泛，其中，使用到了多个 Sever 对象的方法。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 3-4”。

02 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Page_Error(object sender, EventArgs e){
2.     StringBuilder sb = new StringBuilder();
3.     sb.Append("导致错误的 URL: <br/>");
4.     sb.Append(Server.HtmlEncode(Request.Url.ToString()));
5.     sb.Append("<br/><br/>");
6.     sb.Append("错误信息:<br/>");
7.     sb.Append(Server.GetLastError().ToString());
8.     Response.Write(sb.ToString());
9.     Server.ClearError();
```

```

10.    }
11.    protected void Page_Load(object sender, EventArgs e) {
12.        int i = int.Parse("12de");
13.    }

```

代码说明:第1行处理 Page 页面对象的错误事件 Error。第2行使用实例化了一个 StringBuilder 类对象 sb,用于字符串的动态拼接。第4行使用 Server 对象 HtmlEncode 对字符串进行 HTML 编码,获得当前请求的地址并拼接到字符串对象 sb。第7行使用 Server 对象的 GetLastError 获得最近的一个异常具体信息并拼接到字符串对象 sb。第9行使用 Server 对象的 ClearError 方法清除异常。

第11行处理 Page 页面对象的加载事件 Load。为了演示运行结果,在第12行人人为地在程序中设置了一个异常代码,把一个字符串转换成 int 类型,将会抛出一个 FormatException 的异常。

03 按下“Ctrl+F5”,运行结果如图3-9所示。在浏览器中显示了最近一个异常的具体信息。

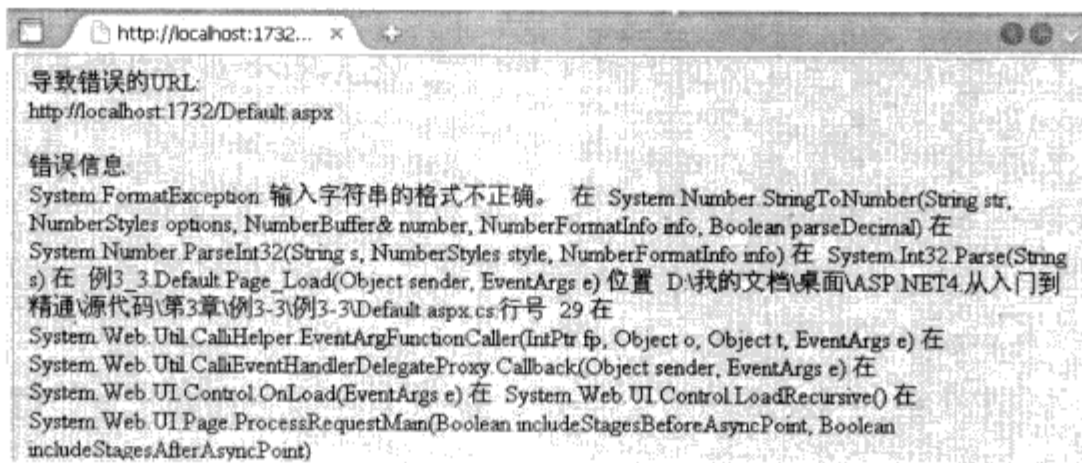


图 3-9 运行结果



提示

Server.MapPath 获得的路径都是服务器上的物理路径,也就是常说的绝对路径。

<%=server.mappath ("database/cnbruce.mdb") %>获得所在页面的当前目录,等价于 Server.MapPath (".")。

<%=server.mappath ("/database/cnbruce.mdb") %>获得应用程序根目录所在的位置。

<%=server.mappath ("../database/cnbruce.mdb") %>获得所在页面的上级目录。

3.5 Cookie 对象

Cookie 对象是 System.Web 命名空间中 HttpCookie 类的对象。Cookie 对象为 Web 应用程序保存用户相关信息提供了一种有效的方法。当用户访问某个站点时,该站点可以利用 Cookie 保存用户首选项或其他信息,这样当用户下次再访问该站点时,应用程序就可以检索以前保存的信息。

3.5.1 Cookie 简介

Cookies 是一种能够让网站服务器把少量数据储存在客户端的硬盘或内存中,或是从客户端的硬盘读取数据的一种技术。Cookies 是当用户浏览某网站时,由 Web 服务器置于用户硬盘上的一个非常小的文本文件,它可以记录用户的 ID、密码、浏览过的网页、停留的时间等信息。当用户再次来到该网站时,网站通过读取 Cookies,得知用户的相关信息,就可以做出相应的动作,如在页

面显示欢迎用户的问候语，或者让用户不用输入 ID、密码就直接登录等等。

保存的信息片断以“键/值”对的形式储存，一个“键/值”对仅仅是一条命名的数据。一个网站只能取得它放在用户电脑中的信息，它无法从其它的 Cookies 文件中取得信息，也无法得到用户电脑上的其它任何东西。Cookies 中的内容大多数经过了加密处理，因此一般用户看来只是一些毫无意义的字母数字组合，只有服务器的处理程序才知道它们真正的含义。

使用 Cookies 的优点可以归纳为如下几点。

- 可配置到期规则。Cookies 可以在浏览器会话结束时到期，或者可以在客户端计算机上无限期存在，这取决于客户端的到期规则。
- 不需要任何服务器资源。Cookies 存储在客户端并在发送后由服务器读取。
- 简单性。Cookies 是一种基于文本的轻量结构，包含简单的键值对。
- 数据持久性。虽然客户端计算机上 Cookies 的持续时间取决于客户端上的 Cookies 过期处理和用户干预，Cookies 通常是客户端上持续时间最长的数据保留形式。

但使用 Cookie 也会有一些缺点：一些用户可能在他们的浏览器中禁止 Cookies，这就会导致那些需要 Cookies 的 Web 应用程序出现问题。大部分情况下，Cookies 得到广泛使用，很多网站都在使用 Cookies。然而，Cookies 可能会限制网站的潜在用户，Cookies 不适合移动装置的嵌入式浏览器，同时，用户可以手动删除存放在自己的硬盘内的 Cookies。

在 ASP.NET 4.0 中，Cookies 是一个内置对象的，但该对象并不是 Page 类的子类，这一点和下一节将要讲述到的 Session 是不同的。

3.5.2 Cookie 对象的属性和方法

使用 Cookie 对象之前，需要了解它的常用属性和方法，Cookie 对象的常用属性和方法如表 3-6 所示。

表 3-6 Cookie 对象的常用属性和方法

属性和方法	说明
Expires	获取或设置此 Cookie 的过期日期和时间
Item	HttpCookie.Values 的快捷方式。此属性是为了与以前的 ASP 版本兼容而提供的。在 C#中，该属性为 HttpCookie 类的索引器
Name	获取或设置 Cookies 的名称
Path	获取或设置输出流的 HTTP 字符集
Add	添加一个 Cookies 变量
Clear	清除 Cookies 集合中的变量
Get	通过索引或变量名得到 Cookies 变量值
GetKey	以索引值获取 Cookies 变量名称
Remove	通过 Cookies 变量名称来删除 Cookies 变量
Value	获取或设置单个 Cookies 值
Values	获取在单个 Cookies 对象中包含的键值对的集合

3.5.3 应用 Cookie 对象

Cookie 使用起来非常容易, 在使用 Cookie 之前, 程序员需要在自己的程序里引用 System.Web 命名空间, 代码如下:

```
using System.Web;
```

对象 Request 和 Response 都提供了一个 Cookies 集合。可以利用 Response 对象设置 Cookie 的信息, 而使用 Request 对象获取 Cookie 的信息。

为了设置一个 Cookie, 只需要创建一个 System.Web.HttpCookie 的实例, 把信息赋予该实例, 然后把它添加到当前页面的 Response 对象里面, 创建 HttpCookie 实例的代码如下:

```
HttpCookie cookie = new HttpCookie("Login"); //创建一个 cookie 实例
cookie.Values.Add("Name", "John");          //添加要存储的信息, 采用键/值结合的方式
Response.Cookies.Add(cookie);                //把 cookie 加如当前的页面的 Response 对象里面
```

采用以上方式, 一个 Cookie 被添加, 它将被发送至每一请求, 该 Cookie 会保持到用户关闭浏览器。为了创建一个生命周期比较长的 Cookie, 程序可以为 Cookie 设置一个生命期限, 示例代码如下:

```
cookie.Expires = DateTime.Now.AddYears(2); //为 cookie 设置 2 年的生命期限
```

开发人员可以利用 Cookie 的名字从 Request.Cookies 集合取得信息, 示例代码如下:

```
HttpCookie cookie1 = Request.Cookies["Login"]; //声明一变量用来存储从 Cookie 里取出的信息
string name;
//判断 cookie1 是否为空, 因为用户有可能禁止 Cookies, 也可能用户把 Cookies 给删除掉
if (cookie1 != null) {
    name = cookie1.Values["Name"];
}
```

有时, 可能需要修改某个 Cookie, 更改其值或延长其有效期 (注意: 由于浏览器不会把有效期信息传递到服务器, 所以程序无法读取 Cookie 的过期日期), 实际上并不是直接更改 Cookie。尽管可以从 Request.Cookies 集合中获取 Cookie 并对其进行操作, 但 Cookie 本身仍然存在于用户硬盘上的某个地方。因此, 修改某个 Cookie 实际上是指用新的值创建新的 Cookie, 并把该 Cookie 发送到浏览器, 覆盖客户机上旧的 Cookie。以下示例说明了如何更改用于储存站点访问次数的 Cookie 值:

```
1. int counter;
2. if (Request.Cookies("counter") == null) {
3.     counter = 0;
4. }
5. else {
6.     counter = counter + 1;
7. }
8. Response.Cookies("counter").Value = counter.ToString;
9. Response.Cookies("counter").Expires = DateTime.Now.AddDays(1);
```

以上代码, 第 2 行判断 Cookie 如果为空, 将站点访问次数设置为 0。第 6 行如果 Cookie 不为空, 将次数加 1。第 8 行将访问次数保存到 Cookie。第 9 行为 Cookie 设置 1 年的生命期限。

删除 Cookie 是修改 Cookie 的另一种形式。由于 Cookie 位于用户的计算机中，所以无法直接将其删除，但可以让浏览器来删除 Cookie。修改 Cookie 的方法前面已经介绍过（即用相同的名称创建一个新的 Cookie），不同的是将其有效期设置为过去的某个日期。当浏览器检查 Cookie 的有效期时，就会删除这个已过期的 Cookie。

删除一个 Cookie 的方式就是利用一个过期的 Cookie 来代替它，示例代码如下：

```
1. HttpCookie cookie = new HttpCookie("Login");
2. cookie.Expires = DateTime.Now.AddDays(-1);
3. Response.Cookies.Add(cookie);
```

以上代码，第 2 行将 Cookie 的年生命期限设置为-1 来表示 Cookie 的过期。

【例 3-5】使用 Cookie 保存用户登录网站的信息。在首次登录后，将登录信息写入到用户计算机的 Cookie 中；当再次登录时，将用户计算机中的 Cookie 信息读出并直接登录到网站，而不需要再次进入登录页面输入用户信息。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 3-5”。

02 用鼠标双击网站的根目录下的“Default.aspx”文件，进入“视图编辑界面”打开“源视图”编辑区，在<form></form>标记之间编写如下代码。

```
1. 用户名: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
2. <asp:CheckBox ID="CheckBox1" runat="server" Text="记住我" />
3. <br />
4. 密码: <asp:TextBox ID="TextBox2" runat="server" TextMode="Password"></asp:TextBox>
5. <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="登录" />
```

代码说明：第 1、4 行分别添加两个文本框服务器控件 TextBox1 和 TextBox2。第 2 行添加了一个单选按钮服务器控件 CheckBox1。第 5 行添加一个按钮服务器控件 Button1，并设置其单击事件为 Click。

03 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Page_Load(object sender, EventArgs e) {
2.     if (Request.Cookies["ID"]!=null &&Request.Cookies["PWD"]!=null){
3.         string id = Request.Cookies["ID"].Value.ToString();
4.         string pwd = Request.Cookies["PWD"].Value.ToString();
5.         Response.Redirect("New.aspx?ID="+id+"&PWD="+pwd);
6.     }
7. }
8. protected void Button1_Click(object sender, EventArgs e) {
9.     if (CheckBox1.Checked){
10.         Response.Cookies["ID"].Expires = new DateTime(2010, 12, 30);
11.         Response.Cookies["PWD"].Expires = new DateTime(2010, 12, 30);
12.         Response.Cookies["ID"].Value = TextBox1.Text;
13.         Response.Cookies["PWD"].Value = TextBox2.Text;
14.     }
15.     Response.Redirect("New.aspx?ID="+TextBox1.Text+"&PWD="+TextBox2.Text);
16. }
```

代码说明：第 1 行处理 Page 页面的加载事件 Load。第 2 行判断如果用户计算机中 Cookie 中 ID 和 PWD 存在的话。第 3、4 行通过 Request 对象的 Cookie 的 value 属性获取用户名和密码的值。第 5 行通

过 Response 对象的 Redirect 方法跳转到 New.aspx 页面, 并将用户名和密码的值同时传递过去。第 8 行处理按钮控件 Button1 的单击事件 Click。第 9 行判断如果用户选择单选按钮。第 10、11 行设置用户名和密码 Cookie 的生命周期。第 12、13 行通过 Request 对象的 Cookie 的 value 属性将两个文本框中的值保存到 Cookies 中。第 15 行跳转到 New.aspx 页面并将两个文本框中的值同时传递过去。

03 在应用程序中添加一个 New.aspx 页面, 然后在 New.aspx.cs 文件中添加如下代码。

```
1. protected void Page_Load(object sender, EventArgs e){
2.     if (Request.QueryString["ID"] != null && Request.QueryString["PWD"] != null){
3.         Response.Write("'" + Request.QueryString["ID"] + "欢迎光临本网站");
4.     }
5. }
```

代码说明: 第 1 行处理 Page 页面的加载事件 Load。第 2 行判断如果 Request 对象的 QueryString 属性获得的用户名和密码不为空。第 3 行在页面显示用户名加“欢迎光临本网站”的欢迎辞。

04 按下“Ctrl+F5”, 运行结果如图 3-10 所示。在显示的登录页面中输入用户名和密码, 选中复选框, 单击“登录”按钮。

05 页面跳转至如图 3-11 所示的 New.aspx。

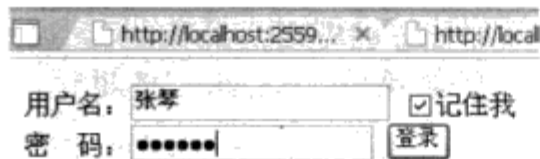


图 3-10 运行结果 1



图 3-11 运行结果 2

06 此后再运行程序, 将发现直接进入 New.aspx 页面而不再进入登录页面, 这是因为用户名和密码已经保存到了 Cookie 的效果。请读者运行验证。

3.6 Session 对象

Session 对象实际上操作 System.Web 命名空间中的 HttpSessionState 类。Session 对象可以为每个用户的会话存储信息。Session 对象中的信息只能被用户自己使用, 而不能被网站的其他用户访问, 因此可以在不同的页面间共享数据, 但是不能在用户间共享数据。

3.6.1 Session 简介

在 ASP.NET 4.0 中 Session 对象是 HttpSessionState 的一个实例。该类为当前用户会话提供信息, 还提供对可用于存储信息的会话范围的缓存的访问, 以及控制如何管理会话的方法。

可以使用 Session 对象存储特定用户会话所需的信息。这样, 当用户在应用程序的 Web 页之间跳转时, 存储在 Session 对象中的变量将不会丢失, 而是在整个用户会话中一直存在下去。当会话过期或被放弃后, 服务器将中止该会话。

利用 Session 进行状态管理是一个 ASP.NET 4.0 的显著特点。它允许程序员把任何类型的数据存储在服务器上。数据信息是受到保护的, 因为它是永远不会传送给客户端, 它捆绑到一个特定的 Session。每一个向应用程序发出请求的客户端则有不同的 Session 和一个独特的信息集合来管理。

当用户请求来自应用程序的 Web 页时，如果该用户还没有会话，则 Web 服务器将自动创建一个 Session 对象。Session 是理想的信息存储器，比如当用户从一个页面跳转另一个页面时，可以在它里面存储购物篮的内容。

在 ASP.NET 4.0 中，Session 是一个内置对象，该对象是 Page 类的子类。

3.6.2 对 Session 的跟踪

ASP.NET 采用一个具有 120 位的标识符来跟踪每一个 Session。ASP.NET 中利用专有算法来生成这个标识符的值，从而保证（统计上的）这个值是独一无二的，它有足够的随机性，能够保证恶意用户不能利用逆向工程或“猜测”获得某个客户端的标识符的值。这个特殊的标识符称为 SessionID。

对于每个用户的每次访问 Session 对象是唯一的，具体包含两个含义：

- 对于某个用户的某次访问，Session 对象在访问期间是唯一，可以通过 Session 对象在页面间共享信息。只要 Session 没有超时，或者 Abandon 方法没有被调用，Session 中的信息就不会丢失。
- 对于用户的每次访问而言，每次产生的 Session 都不同，所以不能共享数据，而且 Session 对象是有时间限制的，通过 Timeout 属性可以设置 Session 对象的超时时间，单位为分钟。如果在规定的时间内，用户没有对网站进行任何操作，Session 将超时。

为使系统能够正常工作，客户端必须为每个请求保存相应的 SessionID，获取某个请求的 SessionID 的方式有两种：

- 使用 Cookies。在这种情况下，当 Session 集合被使用时，SessionID 被 ASP.NET 自动转化一个特定的 Cookie（被命名为 ASP.NET_SessionID）。
- 使用改装的 URL。在这种情况下，SessionID 被转化一个特定的改装的 URL。ASP.NET 的这个特性可以让程序员在客户端禁用 Cookies 时创建 Session。

但是使用 Session 并不是免费，虽然它解决了许多相关的问题，同其他形式的状态管理相比，它迫使服务器存储额外的信息。这笔额外的存储要求，即使是很小，随着数百或数千名客户进入网站，也能快速积累到可以破坏服务器正常运行的水平。

3.6.3 Session 对象的属性和方法

在使用 Session 对象之前必须熟悉它的各种属性和方法，Session 对象的常用属性和方法如表 3-7 所示。

表 3-7 Session 对象的常用属性和方法

属性和方法	说明
Count	获取会话状态下 Session 对象的个数
Timeout	Session 对象的生存周期
SessionID	用于标识会话的唯一编号

(续表)

属性和方法	说明
Abandon	取消当前会话
Add	向当前会话状态集合中添加一个新项
Clear	清空当前会话状态集合中所有键和值
Remove	删除会话状态集合中的项
RemoveAll	删除所有会话状态值
RemoveAt	删除指定索引处的项

Session 对象具有两个事件：Session_OnStart 事件和 Session_OnEnd 事件。Session_OnStart 事件在创建一个 Session 时被触发，Session_OnEnd 事件在用户 Session 结束时（可能是因为超时或者调用了 Abandon 方法）被调用。可以在 Global.asax 文件中为这两个事件增加处理代码。

3.6.4 Session 对象的储存

Session 存储在两个位置，分别是客户端和服务端。客户端只负责保存相应网站的 SessionID，而其他的 Session 信息则保存在服务器端。这就是为什么说 Session 是安全的原因。在客户端只存储 SessionID，这个 SessionID 只能被当前请求网站的客户所使用，对其他人则是不可见的；而 Session 的其它信息则是保存在服务器端，而且永远也不发送到客户端，这些信息对客户都是不可见的，服务器只会按照请求程序取出相应的 Session 信息发送到客户端。所以只要服务器不被攻破，想要利用 Session 搞破坏还是比较困难的。下面按存储位置来分别讲述 Session 的存储。

1. 在客户端的存储

在 ASP.NET 中客户端的 Session 信息存储方式有两种，分别是：使用 Cookie 存储和不使用 Cookie 存储。默认状态下，在客户端是使用 Cookie 存储 Session 信息的。有时为了防止用户禁用 Cookie 造成程序混乱，就会不使用 Cookie 存储 Session 信息。在客户端不使用 Cookie 存储 Session 信息的设置说明如下：

找到当前 Web 应用程序的根目录，打开 Web.Config 文件，找到如下代码段：

```
1. <sessionState
2.     mode="InProc"
3.     stateConnectionString="tcpip=220.115.249.99:42424"
4.     sqlConnectionString="data source=220.115.249.991;Trusted_Connection=yes"
5.     cookieless="false"
6.     timeout="50"
7. />
```

以上代码段中，第 1 行的 sessionState 节点用于设置网站 Session 的状态。其中第 5 行 cookieless="false"表示客户端使用 Cookie 保存 Session 信息。如果改为 cookieless="true"，客户端就不再使用 Cookie 存储 Session 信息，而是将其通过 URL 存储。运行已创建的 Web 应用程序，就会在运行出来的页面的地址栏里看到如图 3-12 所示 SessionID:e3tgb5wzqyx23wlnyamg23kb。这段信息是由 IIS 自动加上的，不会影响以前正常的连接。

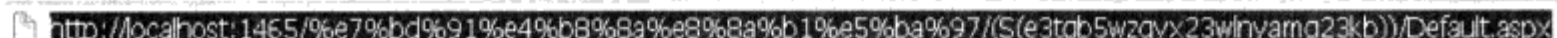


图 3-12 URL 中存储 SessionID

2. 在服务器端的存储

在服务器端存储的 Session 信息可以有三种存储方式：

(1) 存储在进程内

在 Web.Config 文件中找到如下段落：

```

1. <sessionState
2.   mode="InProc"
3.   stateConnectionString="tcpip=220.115.249.99:42424"
4.   sqlConnectionString="data source=220.115.249.99;Trusted_Connection=yes"
5.   cookieless="false"
6.   timeout="50"
/>

```

以上代码，第 2 行设置 `mode="InProc"`，表示采用在服务器端将 Session 信息存储在 IIS 进程中的存储模式。当 IIS 关闭、重起后，这些信息都会丢失。但是这种模式的性能最高。因为所有的 Session 信息都存储在了 IIS 的进程中，所以 IIS 能够很快访问到这些信息，这种模式性能比进程外存储 Session 信息或是在 SQL Server 中存储 Session 信息都要快上很多。这种模式是 ASP.NET 的默认方式。

(2) 存储在进程外

首先，要在“控制面板”里来打开“管理工具”下的“服务”，找到名为：ASP.NET State Service 的服务，启动它。这个服务启动一个可以保存 Session 信息的进程。启动这个服务后，可以从“Windows 任务管理器”的“进程”面板中看到一个名为 `aspnet_state.exe` 的进程，这个就是保存 Session 信息的进程。

然后，回到 Web.config 文件中的 Session 设置代码段中，将 `mode` 的值改为 `StateServer`。这种存储模式就是进程外存储，当 IIS 关闭、重起后，这些信息都不会丢失。

实际上，这种将 Session 信息存储在进程外的方式，不仅可以将信息存储在本机的进程外，还可以将 Session 信息存储在其他服务器的进程中。这时，需要将 `mode` 的值改为 `StateServer`，还需要在 `stateConnectionString` 中配置相应的参数。例如，读者的计算机 IP 是 220.115.249.99，想把 Session 存储在 IP 为 220.115.249.99 的计算机的进程中，就需要设置成这样：`stateConnectionString="tcpip=1220.115.249.99:42424"`。当然，不要忘记在 220.115.249.99 的计算机中装上 .NET Framework，并且启动 ASP.NET State Services 服务。

(3) 存储在 SQL Server 中

启动 SQL Server 和 SQL Server 代理服务。在 SQL Server 中执行一个叫做 `InstallSqlState.sql` 的脚本文件。这个脚本文件将在 SQL Server 中创建一个用来专门存储 Session 信息的数据库，及一个维护 Session 信息数据库的 SQL Server 代理作业。读者可以在以下路径中找到这个脚本文件：

[System Drive]\Winnt\Microsoft.net\Framework\v2.0

然后打开查询分析器，连接到 SQL Server 服务器，打开这个文件脚本并且执行。稍等片刻，

数据库及作业就建立好了。这时，读者可以打开企业管理器，看到新增了一个叫 ASPState 的数据库。但是这个数据库中只是些存储过程，没有用户表。实际上 Session 信息存储在 tempdb 数据库的 ASPStateTempSessions 表中，另外一个 ASPStateTempApplications 表存储了 ASP 中 Application 对象信息。这两个表也是用上面的脚本建立的。另外查看“管理”|“SQLServer 代理”|“作业”，发现也多了一个叫做 ASPState_Job_DeleteExpiredSessions 的作业，这个作业实际上就是每分钟去 ASPStateTempSessions 表中删除过期的 Session 信息的。

接着，返回到 Web.config 文件，修改 mode 的值为 SQL Server。注意，还要同时修改 sqlConnectionString 的值，格式为：

```
sqlConnectionString="data source=localhost; Integrated Security=SSPI;"
```

其中 data source 是指 SQL Server 服务器的 IP 地址，如果 SQL Server 与 IIS 是一台机子，写 127.0.0.1 就行了。Integrated Security=SSPI 的意思是使用 Windows 集成身份验证，这样，访问数据库将以 ASP.NET 的身份进行，通过如此配置，能够获得比使用 userid=sa;password=口令的 SQL Server 验证方式更好的安全性。当然，如果 SQL Server 运行于另一台计算机上，读者可能需要通过 Active Directory 域的方式来维护两边验证的一致性。

如何选择 Session 的存储方式需要根据具体的情况分析。在客户端是否使用 Cookie 来存储 SessionID 需要程序员做一个判断，判断客户是否禁用 Cookie，不过一般情况下都会采用默认情况。在服务器端如何就三种形式做出选择，需要根据实际需求：追求效率的话肯定采用默认的形式存储 Session 信息；要想持久保存 Session 就可以选择其它两种方式。其实，究竟采用哪种方式，最重要的是由程序员根据实际需求来分析，有时可能需要做很多测试才能决定下来采用哪种方式。

3.6.5 应用 Session 对象

Session 对象的使用和 ViewState 使用方法一样，在 Session 里存储一个 Login 的示例代码如下：

```
Session["Login"] = login; // login 为 Login 的一个实例
```

可以通过如下代码从 Session 里取得该 Login：

```
login = (Login) Session["login "];
```

对当前用户来说，Session 对象是整个应用程序的一个全局变量，程序员在任何页面代码里都可以访问该 Session 对象。但某些情况下，Session 对象有可能会丢失：

- 用户关闭浏览器或重启浏览器。
- 如果用户通过另一个浏览器窗口进入同样的页面，尽管当前 Session 依然存在，但在新开的浏览器窗口中将找不到原来的 Session。这和 Session 的机制有关。
- Session 过期。
- 程序员利用代码结束当前 Session。

在前两种情况下，Session 实际上仍然在内存中，因为服务器有不知道客户端已关闭浏览器或改变窗口，本次 Session 将保留在内存中，直到该 Session 过期。但是程序员却无法再找到 Session，因为 SessionID 此时已经丢失，失去了 SessionID 就无法从 Session 集合里检索到该 Session。

【例 3-6】在网站中存在着不同的用户，他们各自有不同的权限，登录后可以进入页面也就不同。本来在用户登录时利用 Session 对象记录登录用户的类别，然后根据不同的类别登录实现自动的导航功能。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 3-6”。

02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”打开“源视图”编辑区，在<form></form>标记之间编写如下代码。

```
1.  用户类型: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
2.  <br />
3.  密码: <asp:TextBox ID="TextBox2" runat="server" TextMode="Password"></asp:TextBox>
4.  <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="登录" />
```

代码说明：第 1、3 行分别添加两个文本框服务器控件 TextBox1 和 TextBox2。第 4 行添加一个按钮服务器控件 Button1，设置其单击事件为 Click。

03 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1.  private static readonly string[] users = new string[] { "admin", "user" };
2.  private int usertype(string userid){
3.      if (userid == users[0])
4.          return 1;
5.      if (userid == users[1])
6.          return 2;
7.      else
8.          return 0;
9.  }
10. protected void Button1_Click(object sender, EventArgs e){
11.     string userid = TextBox1.Text.ToString();
12.     string pwd = TextBox2.Text.ToString();
13.     Session["UserType"] = usertype(userid);
14.     switch (Session["UserType"].ToString()){
15.         case "1": Response.Redirect("Admin.aspx?userid="+userid);
16.             break;
17.         case "2": Response.Redirect("User.aspx?userid="+userid);
18.             break;
19.         default: Response.Write("<script>alert('对不起，你不是合法用户！')</script>");
20.             break;
21.     }
22. }
```

代码说明：第 1 行定义了一个只读的静态变量的字符串数组，保存系统中已经注册的用户两种类型 admin 和 user。第 2 行定义了一个 usertype 方法，参数是用户类型。第 3~9 行判断用户类型如果存在于注册的用户中，则返回一个给定的整数。第 10 行处理按钮控件的单击事件 Click。第 11、12 行获得用户输入的值。第 13 行将用户类型保存到 Session 中。第 14~21 行使用 switch-case 语句判断 Session 中的值，并根据不同的值，跳转到不同用户类型的页面。如果没有存在该种类型，给出错误的提示。

04 在应用程序中分别添加一个 Admin.aspx 页面和一个 User.aspx 页面。然后在 Admin.aspx.cs 和 User.cs 文件中添加如下代码。

```

1. protected void Page_Load(object sender, EventArgs e){
2.     string user = Request.QueryString["userid"].ToString();
3.     Response.Write("'" + user + ",欢迎你! ");
4. }

```

代码说明：第 1 行处理 Page 页面的加载事件 Load。第 2 行通过 Request 对象的 QueryString 属性获得传递进来的用户类型。第 3 行在页面显示“欢迎你”的信息。

05 按下“Ctrl+F5”，运行结果如图 3-13 所示。在显示的登录页面中输入用户类型“admin”和密码，单击“登录”按钮。

06 页面跳转至如图 3-14 所示的 Admin.aspx。

07 在登录页面中输入用户类型“user”和密码，单击“登录”按钮。页面会跳转至 User.aspx，显示“User,欢迎你！”。如果输入不存在的用户类型，单击“登录”按钮，会出现如图 3-15 所示的提示对话框。

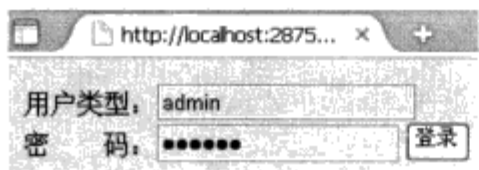


图 3-13 运行结果 1

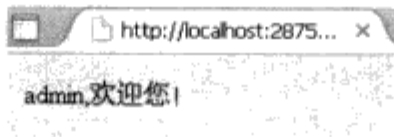


图 3-14 运行结果 2



图 3-15 运行结果 3

3.7 Application 对象

Application 对象是 HttpSessionState 类的一个实例，定义 ASP.NET 应用程序中的所有应用程序对象通用的方法、属性和事件。HttpSessionState 类是由用户在 global.asax 文件中定义的应用程序的基类。此类的实例 Application 对象是在 ASP.NET 基础结构中创建的，而不是由用户直接创建的。一个实例在其生存期内用于处理多个请求，但它一次只能处理一个请求。这样，成员变量才可用于存储针对每个请求的数据。

Application 的原理是在服务器端建立一个状态变量，来存储所需的信息。要注意的是，首先，这个状态变量是建立在内存中的，其次是这个状态变量是可以被网站的所有页面访问的。

Application 对象具有如下特点。

- 数据可以在 Application 对象内部共享。
- 一个 Application 对象包含事件，可以触发某些 Application 对象脚本。
- 个别 Application 对象可以用 Internet Service Manager 来设置而获得不同属性。
- 单独的 Application 对象可以隔离出来在它们自己的内存中运行。
- 可以停止一个 Application 对象(将其所有组件从内存中驱除)而不会影响到其他应用程序。
- 一个网站可以有不止一个 Application 对象。典型情况下，可以针对个别任务的一些文件创建个别的 Application 对象。例如，可以建立一个 Application 对象来适用于全部公用用户，而再创建另外一个只适用于网络管理员的 Application 对象。
- Application 对象成员在服务器运行期间持久地保存数据。Application 对象成员的生命周期止于关闭 IIS 或使用 Clear 方法清除。

- 因为多个用户可以共享一个 Application 对象，所以必须要有 Lock 和 Unlock 方法，以确保多个用户无法同时改变某一属性。

3.7.1 Application 对象的属性和方法

Application 的原理是在服务器端建立一个状态变量，来存储所需的信息。要注意的是，首先，这个状态变量是建立在内存中的，其次是这个状态变量是可以被网站的所有页面访问的。

Application 对象用来存储变量或对象，以便在网页中再次被访问（不管是不是同一个连接者或访问者），所存储的变量或对象的内容还可以重新调出来使用，也就是说 Application 对于同一网站来说是公用的，可以在各个用户间共享。访问 Application 对象变量的方法如下所示。

```
Application["变量名"]=变量值
变量=Application["变量名"]
```

以上代码，第 1 行给 Application 对象设置一个名称并赋值，第 2 行获取该 Application 对象的值并赋给某个变量。

为了简便，我们还可以把 Application["变量名"]直接当作变量来使用。在 Web 页面中可以通过语句<%=Application["变量名"]%>直接使用这个值。如果通过 ASP.NET 内置的服务器对象使用应用程序变量，则代码为：Label1.Text = (String) Application["变量名"]。

利用 Application 对象存取变量时需要注意以下几点。

- Application 对象变量应该是经常使用的数据，如果只是偶尔使用，可以把信息存储在磁盘的文件中或者数据库中。
- Application 对象是一个集合对象，它除了包含文本信息外，也可以存储对象。
- 如果站点开始就有很大的通信量，则建议使用 Web.config 文件进行处理，不要用 Application 对象变量。

Application 对象的常用属性和方法如表 3-8 所示。

表 3-8 Application 对象的常用属性和方法

方法和属性	说明
AllKeys	获取 HttpApplicationState 集合中的访问键
Count	获取 HttpApplicationState 集合中的对象数
Add	新加一个 Application 对象的变量
Clear	清除全部 Application 对象的变量
Get	使用索引或者变量名称获取变量值
GetKey	使用索引获取变量名称
Lock	锁定全部变量
Remove	使用变量名删除一个 Application 对象的变量
RemoveAll	删除 Application 对象的所有变量
Set	使用变量名更新 Application 对象变量的内容
UnLock	解锁 Application 对象的变量

Application 对象是一个集合对象，并在整个 ASP.NET 网站内可用，不同的用户在不同的时间都有可能访问 Application 对象的变量，因此 Application 对象提供了 Lock 方法用于锁定对 HttpSessionState 变量的访问以避免同步访问造成的问题。在对 Application 对象的变量访问完成后，需要调用 Application 的 Unlock 方法取消对 HttpSessionState 变量的锁定。下面的代码通过 Lock 和 Unlock 方法实现了对 Application 变量的修改操作。

```
1. Application.Lock();
2. Application["Online"] = 21;
3. Application["AllAccount"] = Convert.ToInt32(Application["AllAccount"]) + 1;
4. Application.Unlock();
```

第 1 行在更改变量前执行 Lock() 方法，避免其他用户存取 Online 和 AllAccount 变量，如果是读取变量而不是更改变量，就不需要 Lock() 方法。如第 4 行所示，在更改完成后，要及时调用 Unlock() 函数，以便让其他用户可以更改这些变量。

Application 对象还有两个比较重要的事件：Application_OnStart 和 Application_OnEnd，其中 Application_OnStart 在 ASP.NET 应用程序执行时触发，Application_OnEnd 事件在 ASP.NET 应用程序结束执行时触发。一般在 Global.asax 文件对这两个事件进行处理，添加用户自定义代码。

3.7.2 应用 Application 对象

上一小节介绍了 ViewState 对象的概念以及它的常用方法和属性。本小节通过一个示例来演示 Application 对象的属性和方法在实际中的使用。

【例 3-7】在网站中常常需要记录用户的访问量，这样有利于网站管理员对网站访问情况进行统计。本例使用 2 个 Application 来存储 2 种用户类型的访问量，管理员通过登录进入显示访问量的页面。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 3-7”。

02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”打开“源视图”编辑区，在<form></form>标记之间编写代码和“例 3-5”中“Default.aspx”文件同样的代码。

03 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. private static readonly string[] users = new string[] { "admin", "user" };
2. private int usertype(string userid){
3.     //此处的代码与“例 3-6”中相同，故省略。
4. }
5. protected void Button1_Click(object sender, EventArgs e) {
6.     string userid = TextBox1.Text.ToString();
7.     string pwd = TextBox2.Text.ToString();
8.     if (usertype(userid) == 1 || usertype(userid) == 2)
9.         Application.Lock();
10.    switch (usertype(userid).ToString()){
11.        case "1": Application["admin"] = Convert.ToInt32(Application["admin"]) + 1;
12.            break;
13.        case "2": Application["user"] = Convert.ToInt32(Application["user"]) + 1;;
14.            break;
15.        default: Response.Write("<script>alert('对不起，你不?是合法用户!')</script>");
16.            break;
17.    }
18.    Application.Unlock();
```

```

19.         Response.Redirect("Total.aspx?userid=" + userid);
20.     }

```

代码说明：第 1 行定义了一个只读的静态变量的字符串数组，保存系统中已经注册的用户的两类型 admin 和 user。第 2 行定义了一个和“例 3-6”完全相同的 usertype 方法。第 5 行处理按钮控件的单击事件 Click。第 6~7 行获得用户输入的值。第 8 行判断是否用户类型存在，第 9 行使用 Application 对象的 Lock 方法锁定该对象，不允许进行对数据的修改。第 10~17 行使用 switch-case 语句判断用户的类型，如果是 admin 或 user，将相应的 Application 对象的访问量加 1。否则，显示错误提示信息。第 18 行调用 Application 对象的 UnLock 方法开锁，允许进行对数据的修改。第 19 行跳转到 Total.aspx 页面并传递用户输入的值。

04 在应用程序中分别添加一个 Total.aspx 页面，然后在 Total.aspxcs 文件中添加如下代码。

```

1.     protected void Page_Load(object sender, EventArgs e){
2.         string userid = Request.QueryString["userid"].ToString();
3.         Response.Write(userid+"!欢迎你光临<br>");
4.         if (Application["admin"] != null)
5.             Response.Write("<br>管理员的访问量是: " + Application["admin"].ToString ());
6.         else
7.             Response .Write ("<br>管理员的访问量是: 0" );
8.         if (Application["user"] != null)
9.             Response.Write("<br>用户的访问量是: " + Application["user"].ToString());
10.        else
11.            Response.Write("<br>用户的访问量是: 0");
12.    }

```

代码说明：第 1 行处理 Page 页面的加载事件 Load。第 4~7 行判断 Application 中保存的管理员值如果存在，则显示管理员访问页面的次数，否则，显示次数为 0。第 8~11 行断 Application 中保存的用户值，如果存在，则显示用户访问页面的次数，否则，显示次数为 0。

05 按下“Ctrl+F5”，运行程序。在显示的登录页面中输入用户名“admin”或“user”以及密码，再单击“登录”按钮，如此反复操作数次，分别记住 admin 和 user 登录的次数。

06 每次跳转到 Total.asp 时，会在如图 3-16 的页面中显示用户名、管理员和用户访问页面的次数。

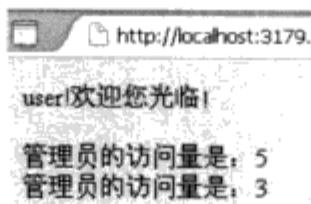


图 3-16 运行结果



提示

Application 对象变量应该是经常使用的数据，如果只是偶尔使用，可以把信息存储在磁盘的文件中或者数据库中。如果站点开始就有很大的通信量，则建议使用 Web.config 文件进行处理，不要用 Application 对象变量。

3.8 ViewState 对象

ViewState（视图状态）是 ASP.NET 中重要的一种机制，使用这种机制可以用来跟踪服务器控件的状态值，否则这些值将不作为 HTTP 窗体的一部分而回传。例如，由 Label 控件显示的文本默认情况下就保存在 ViewState 中。然而 ViewState 的功能远不只这些，编程人员可以直接把信息（比如当前用户的信息）存储在 ViewState 之中，在页面回传之后访问存储在其中的信息。

3.8.1 ViewState 中的键值对

ViewState 是由 ASP.NET 框架管理的一个隐藏的窗体字段。当 ASP.NET 执行某个页面时，该页面上的 ViewState 值和所有控件将被收集并格式化成一个编码字符串，然后被分配给隐藏窗体字段的值属性（即）。由于隐藏窗体字段是发送到客户端的页面的一部分，所以 ViewState 值被临时存储在客户端的浏览器中。如果客户端选择将该页面回传给服务器，则 ViewState 字符串也将被回传。

ViewState 提供了一个 ViewState 集合（Collection）属性。该集合是集合（Collection）类的一个实例，集合类是一个键值集合，开发人员可以通过键来为 ViewState 增加或者去除项。例如下面的代码：

```
ViewState["Count"] = 8;
```

这句代码含义是把一个整数 2 赋值给 ViewState 集合，而且给它一个键名 Count 来标识。如果当前 ViewState 集合里没有键名 Count，那么一个新项就自动添加到 ViewState 集合里；如果存在键名 Count，则与该键名 Count 对应的值就会被替换。

在 ViewState 集合里，利用键名可以访问到与键名对应的值，这是键值集合的特性。ViewState 集合里存储的是对象（Objects），因此它可以用来处理各种数据类型。下面代码就是从 ViewState 集合里取得整型数据的示例代码：

```
int count = (int)ViewState["Count"];
```

3.8.2 ViewState 的安全机制

我们查看一个 Web 页面的源文件，就会看到存储的 ViewState，其内容可能如下：

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wDILYBHyO4EPDwUKMTQ2OTkzNDMyMQ8WAh4HQ291bnRlcgICZGRbrmBI9jHiwh0Z9Jg=" />
```

以上代码就是 ViewState 的窗体字段，ViewState 的信息是存储在属性 value 中的。由于属性 value 的值是不可读的，大多数人可能就认为存储在 ViewState 里的信息是被加密过的，然而事实上并非如此，在以上代码中所看到的 value 的值只是一个经过 Base64（一种内容传送编码技术）编码过的字符串。别人可以利用反向工程技术在几秒钟内就把经过 Base64 编码的字符串变为可读的字符串，从而查看到 ViewState 中存储的信息。

如果想要使 ViewState 变得更加安全的话，可以有两种选择。

(1) 采用哈希编码技术

哈希编码技术被认为是一种强大的编码技术。其算法思想是让 ASP.NET 检查 ViewState 中的所有数据，然后通过散列算法（在密钥值的帮助下）把这些数据编码。该散列算法产生一段很短的数据信息，即哈希代码。然后把这段代码加在 ViewState 信息后面。

页面被回传后，ASP.NET 检查 ViewState 的数据信息，使用同样步骤重新计算哈希代码，核查计算出来的编码信息是否与存储在 ViewState 里哈希代码相匹配。如果有用户更改了 ViewState 里存储的信息，ASP.NET 就会产生一段不能相匹配的新的哈希代码，此时 ASP.NET 就会拒绝页面完全回传。

哈希代码的功能实际上是默认的，所以如果编程人员希望有这种功能，不需要采取额外的步骤。但有时开发商选择禁用此项功能，以防止出现这样的问题：在一个网站系统中不同的服务器（一个大型的网站通常有很多台服务器）有不同的密钥。为了禁用哈希代码，我们可以使用在 Web.config 文件中的 <Pages> 元素的 enableViewStateMac 属性，示例代码如下：

```
<pages enableViewStateMac="false"></pages>
```

(2) ViewState 加密

尽管使用了哈希代码，ViewState 信息依然能够被用户阅读到，很多情况下这是完全可以接受的。但是如果 ViewState 里包含了需要保密的信息，就需要采用 ViewState 加密。可以设置单独的某一页面采用 ViewState 加密，代码如下：

```
<%Page ViewStateEncryptionMode="Always"%>
```

也可以在 Web.Config 文件为整个网站设置采用 ViewState 加密，代码如下：

```
<pages viewStateEncryptionMode="Always"></pages>
```

当在 ViewState 里添加很多信息时，属性 value 的值就会变得很长，这样就会影响页面打开的速度，所以有时希望禁用个别控件的视图状态。这时就可以通过修改控件的 EnableViewStated 的属性来改变视图状态。如果不想逐个控件进行修改，也可以禁用整个页面的视图状态。通过修改 Page 指令的 EnableViewState 属性来达到该效果。

3.8.3 存储自定义对象

在 ViewState 里可以存储自定义的对象，就像在 ViewState 里存储数值和字符串类型一样容易。然而，为了在 ViewState 里贮存该对象，ASP.NET 技术必须能够把该对象转化成一种字节流，使它可以添加到页面的隐藏输入字段的后面，这一过程被称为序列化。如果对象是不能序列化的，当程序员试图把这样的对象放在 ViewState 里，就会出现错误。

为了使对象序列化，编程人员需要在类定义之前加一个 Serializable 属性。例如一个简单学生类的定义代码如下：

```
1. [Serializable] //序列化标识
2. public class Student{
3.     public string name;
4.     public int age;
5.     public Student (string name, int age){
```

```

6.         this.name = name;
7.         this.age = age;
8.     }
9. }

```

以上代码,第1行对学生类进行序列化后就可以存储在 ViewState 里。第2~8行封装了一个自定义的学生类。

从 ViewState 里取回数据时的代码如下:

```

1.     Student s;
2.     s = (Student)ViewState["Student"];

```

以上代码第2行从 ViewState 里取回一个学生对象。

3.8.4 应用 ViewState 对象

上一小节介绍了 ViewState 对象的概念以及它的常用方法和属性,本小节通过示例来演示 ViewState 对象在实际中的使用。

【例 3-8】通过使用 ViewState 存放排序方式和排序的列标题,实现显示信息列表的升序和降序排列功能。

01 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 3-8”。

02 在 App_Data 文件夹中创建一个数据库文件,文件会给到大家,不需要自行创建。

03 用鼠标双击网站的目录下的“Default.aspx”文件,进入“视图编辑界面”打开“源视图”编辑区,在<form></form>标记之间编写代码如下。

```

1.     <strong><span class="style1">鲜花信息</span> </strong>
2.     <asp:GridView ID="GridView1" runat="server" onsorting="GridView1_Sorting1" AllowSorting="True">
3.     </asp:GridView>

```

代码说明:第2行添加一个服务器列表控件 GridView1 并设置允许排序。设置控件排序的事件 GridView1_Sorting1。

04 用鼠标双击网站目录下的“Default.aspx.cs”文件,编写代码如下。

```

1.     SqlConnection sqlcon;
2.     string strCon = "Data Source=(local);Database=E:\\ASP.NET 实例书案例\\网上花店\\DATABASE\\
ST_FLOWERPREARRANGE.MDF;Uid=sa;Pwd=585858";
3.     private int sum = 0; //取指定列的数据和
4.     protected void Page_Load(object sender, EventArgs e){
5.         if (!IsPostBack){
6.             ViewState["SortOrder"] = "Name";
7.             ViewState["OrderDire"] = "ASC";
8.             bind();
9.         }
10.    public void bind(){
11.        string sqlstr = "select * from Flower";
12.        sqlcon = new SqlConnection(strCon);
13.        SqlDataAdapter myda = new SqlDataAdapter(sqlstr, sqlcon);
14.        DataSet myds = new DataSet();
15.        myda.Fill(myds, "Flower");
16.        DataView view = myds.Tables["Flower"].DefaultView;

```

```
17.         string sort = (string)ViewState["SortOrder"] + " " + (string)ViewState["OrderDire"];
18.         view.Sort = sort;
19.         GridView1.DataSource = view;
20.         GridView1.DataBind();
21.         GridView1.ShowFooter = false;
22.     }
23.     protected void GridView1_Sorting1(object sender, GridViewSortEventArgs e){
24.         string sPage = e.SortExpression;
25.         if (ViewState["SortOrder"].ToString() == sPage){
26.             if (ViewState["OrderDire"].ToString() == "Desc")
27.                 ViewState["OrderDire"] = "ASC";
28.             else
29.                 ViewState["OrderDire"] = "Desc";
30.         }
31.         else{
32.             ViewState["SortOrder"] = e.SortExpression;
33.         }
34.         bind();
35.     }
```

代码说明：第 2 行定义数据库连接字符串，这个字符串需要根据大家放置数据库文件的路径做相应改变。第 4 行处理页面对象的加载事件。第 5 行判断如果加载的页面不是回传，则第 6 行将要排序的列标题存放到 ViweState 对象中。第 7 行将排序的方式也存放到 ViweState 对象中。第 8 行调用 bind 方法。第 10~22 行定义了 bind 方法。其中，第 11~15 行连接数据库查询鲜花表数据。第 16 行将查询的结果存放到 DataView 对象 view 中。第 17 行将存放在 ViweState 对象中排序的列标题和排序方式取出拼接起来并赋给一个字符串对象 sort。第 18 行把 sort 赋给 DataView 对象 view 的 sort 属性。第 19 行将 view 做为服务器列表控件的数据源。第 20 行调用控件的 DataBind 方法将查询结果绑定控件显示出来。

第 23~35 行处理服务器列表控件的排序事件 Sorting1。其中第 24 行获取数据源的排序表达式。第 25 行判断排序表达式如果与 ViweState 对象中排序的列标题相同的话，则第 26 行在判断 ViweState 对象中如果是降序方式，就将升序方式存放到 ViweState 对象中，否则不去改动。第 31 行判断如果排序表达式与 ViweState 对象中排序的列标题不相同的话，就将排序表达式存放到 ViweState 对象中。第 34 行调用 bind 方法。

05 按下“Ctrl+F5”，运行结果如图 3-17 所示。单击列标题可以进行升序和降序的排序操作。

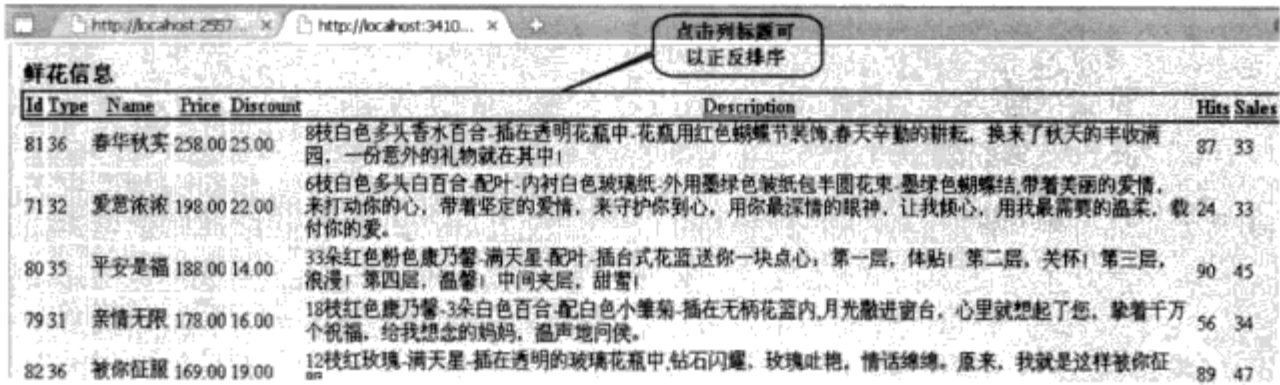


图 3-17 运行结果

3.9 上机题

1. 实现从程序中读取某张已存在的图片,运行程序后将图片显示在浏览器中。提示:使用 `Server` 对象的 `MapPath` 方法获得文件名,然后使用 `Response` 对象的 `WriteFile` 方法读取图片内容。
2. 创建一个网页,设计一个登录界面,当用户输入姓名、籍贯和性别,单击“确定”按钮时,将用户输入的信息显示在页面上。要求使用 `Request` 对象来获取表单的数据,程序运行结果如图 3-18 所示。
3. 编写一个程序,创建两个网页,每个网页中有一个 `TextBox`,使用 `Session` 判断在第二个 `TextBox` 中输入的数字是否等于第一个 `TextBox` 中输入的数字,运行结果如图 3-19 所示。

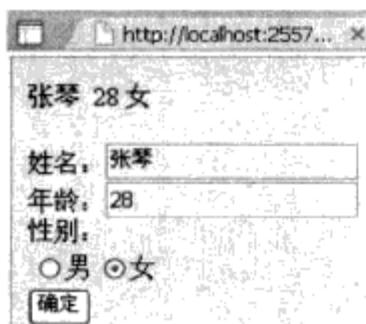


图 3-18 运行结果

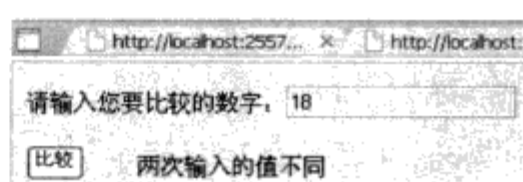
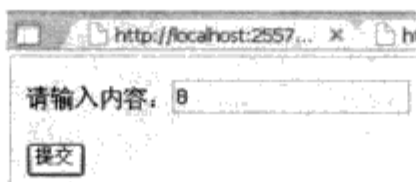


图 3-19 运行结果

4. 大多数网站都有统计网站访问量的功能,通过统计网站的访问量,可以清楚的反映网站的人气。本题要求利用 `Application` 对象来实现统计网站的总访问量,运行结果如图 3-20 所示。
5. 记录一个用户的上一次访问网站的时间,对管理网站非常重要,因为可以由此分析获取该用户的访问规律。要求利用 `Cookie` 对象来实现这一功能,程序运行结果如图 3-21 所示

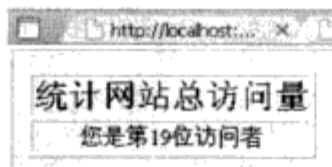


图 3-20 运行结果

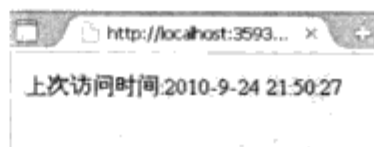


图 3-21 运行结果

6. 实现检查本地机器 IE 浏览器版本的功能,如果版本低于指定的版本号,则建议升级至最新的版本;如果还没有使用 IE 浏览器,则给出相应的提示信息。
7. 在网络世界中,服务器之间的识别依赖于 IP 地址,而域名和 IP 地址之间是一一对应的,确定对方的 IP 地址显得非常重要。本题就要求使用页面内置对象获取服务器端和客户端的 IP 地址并显示。
8. 在实际应用中,经常需要将 Web 服务器上指定的虚拟路径转换为相对应的物理路径。本题要求先在 Visual Studio 2010 中创建一个 Web 项目,然后获取所该项目的当前工作目录和上一级父目录的物理路径。

第 4 章 ASP.NET 4.0 服务器控件

学习目标

ASP.NET 4.0 服务器控件是在服务器端运行的，它与代码和标记一起被包含在页面中。在初始化时，服务器控件会根据用户浏览器的版本生成适合浏览器的 HTML 代码。服务器控件参与页面的执行过程，并在客户端生成自己的标记呈现内容。虽然这些控件类似于常见的 HTML 元素，但是它包括了一些相对复杂的行为。在创建 ASP.NET 页面时会大量使用这些控件，因此掌握 Web 控件的相关知识非常重要。

本章重点

- 掌握服务器控件的基本属性
- 在页面中使用各种按钮服务器控件
- 掌握列表控件的常用属性和事件
- 熟练应用图像服务器控件

4.1 服务器控件类

HTML 控件在过去的页面开发中基本可以满足用户的需求，但是没有办法利用程序直接来控制它们的属性、方法和事件。而在交互性要求比较高的动态页面（需要同用户交互的页面）中需要使用到 ASP.NET 4.0 提供的 Web 服务器控件，这些 Web 控件提供了丰富的功能。在熟悉了这些控件后，开发人员就可以将主要精力放在程序逻辑业务的开发上。

大多数的 Web 服务器控件类都派生于 `System.Web.UI.WebControl`，而 `WebControl` 类又从 `System.Web.UI.Control` 类派生，都包含在 `System.Web.UI.WebControls` 命名空间下面。

在 `System.Web.UI.WebControls` 以下，服务器控件可分为两部分：

- Web 控件。这种控件用来组成与用户进行交互的页面，比如最常见的用户提交表单。这类控件包括最常用的按钮控件、文本框控件、标签控件等，还有验证用户输入的控件，以及自定义的用户控件等。使用这些控件可以组成与用户交互的接口。
- 数据绑定控件，在 Web 应用程序中，我们往往需要在页面中呈现一些来自于数据库、XML 文件等的信息。这时我们就要用到数据绑定控件来实现数据的绑定和显示。这类控件包括广告控件、表格控件等，还有用于导航的菜单控件和树型控件。



提示

ASP.NET 把几乎所有的 HTML 控件都转化成了服务器控件,然而这些控件的功能有限,ASP.NET 提供的 Web 控件则提供了丰富功能,可以使程序的开发变得更加简单和丰富。ASP.NET 服务器控件在服务器端运行,它们在初始化时,根据客户的浏览器版本,自动生成适合浏览器的 HTML 代码。

4.1.1 服务器控件基本属性

服务器控件的基类 WebControl 定义了一些可以应用于几乎所有的服务器控件的基本属性,涵盖了控件的外观、行为、布局和可访问性等方面。

1. 外观属性

ASP.NET 服务器控件的外观属性主要包括前景色、背景色、边框和字体等,这些属性一般在设计时设置,如有必要,也可以在运行时动态设置。

(1) BackColor 和 ForeColor 属性

BackColor 属性:用于设置对象的背景色,其属性的设定值为颜色名称或是#RRGGBB 的格式。

ForeColor 属性:用于设置对象的前景色,其属性的设定值和 BackColor 的要求一样,为颜色名称或是#RRGGBB 的格式。

例如设置控件 Button1 的颜色属性的代码如下:

```
1. int alpha = 150,red = 0,green = 150,blue = 0;
2. Button1.BackColor = Color.FromArgb(alpha,red,green,blue);
3. Button1.BackColor = Color.Red;
4. Button1.BackColor = ColorTranslator.FromHtml("Blue");
```

代码说明:第 1~2 行利用 ARGB 值设置控件 Button1 的背景色,第 3 行使用颜色枚举值设置控件 Button1 的背景色,第 4 行使用 HTML 颜色名创建颜色来设置控件 Button1 的背景色。

(2) Border 属性

边框属性包括 BorderWidth、BorderColor、BorderStyle 等几个属性。其中,BorderWidth 属性可以设定 Web 控件的边框宽度,单位是像素。下面的代码把 Label 控件的边框宽度设置为 10 个像素。

```
<ASP:Label Id="Label1" Text="Label" BorderWidth=10 Runat="Server"/>
```

BorderColor 属性用于设定边框的颜色,其属性的设定值为颜色名称或是 #RRGGBB 的格式。

BorderStyle 属性用来设定对象的边框样式,总共有以下几种设定值。

- Notset: 默认值。
- None: 没有边框。
- Dotted: 边框为虚线,点较小。
- Dashed: 边框为虚线,点较大。
- Solid: 边框为实线。
- Double: 边框为实线,但厚度是 Solid 的两倍。
- Groove: 在对象四周出现 3D 凹陷式的边框。

- Ridge: 在对象四周出现 3D 突起式的边框。
- Inset: 控件呈陷入状。
- Outset: 控件成突起状。

(3) Font 属性

Font 属性有以下几个子属性，分别表现不同的字体特性。

- Font-Bold: 如果属性值设定为 True，则会变成粗体显示。
- Font-Italic: 如果属性值设定为 True，则会变成斜体显示。
- Font-Names: 设置字体的名字。
- Font-Size: 设置字体大小，共有九种大小可供选择 Smaller、Larger、XX-Small、X-Small、Small、Medium、Large、X-Large 或者 XX-Large。
- Font-Strikeout: 如果属性值设定为 True，则文字中间显示一条删除线。
- Font-Underline: 如果属性值设定为 True，则文字下面显示一条底线。

例如设置按钮 Button1 的字体属性代码如下：

```
1. Button1.Font.Name = "Verdana";  
2. Button1.Font.Bold = true;  
3. Button1.Font.Size = FontUnit.Small;  
4. Button1.Font.Size = FontUnit.Point(14);
```

代码说明：第 1 行设置按钮控件 Button1 上文字的字体为宋体。第 2 行设置字体为加粗。第 3 行设置字体的相对大小。第 4 行设置字体的实际大小为 14 个像素。

2. 行为属性

服务器空间的行为属性主要包括是否可见、是否可用以及控件的提示信息。除了提示信息之外，其余的行为属性多在运行时动态设置。

(1) Enabled 属性

Enabled 属性用于设置禁止控件还是使能控件。当该属性值为 False 时，控件为禁止状态。当该属性值为 True 时控件为使能状态，对于有输入焦点的控件，用户可以对控件执行一定的操作，例如单击 Button 控件、在文本框中输入文字等。默认情况下，控件都是使能状态。

(2) ToolTip 属性

ToolTip 属性用于设置控件的提示信息。在设置了该属性值后，当鼠标停留在 Web 控件上一小段时间后就会出现 ToolTip 属性中设置的文字。通常设置 ToolTip 属性为一些提示操作的文字。

(3) Visible 属性

Visible 属性决定了控件是否会被显示，如果属性值为 True 将显示该控件，否则将隐藏该控件（该控件存在，只是不可见），默认情况下，该属性为 True。

3. 可访问性

为了方便用户使用键盘访问网页，设计网页时需要支持快捷键和 Tab 键，只有这样，我们设计的网页才能方便用户访问。

(1) AccessKey 属性

AccessKey 属性用来为控件指定键盘的快速键，这个属性的内容为数字或是英文字母。例如设置为“A”，那么使用时用户按下 Alt+A 组合键就会自动将焦点移动到这个控件上面。只有 Internet Explorer 4.0 或者更高的版本才支持这个特性。

(2) TabIndex 属性

TabIndex 属性用来设置 Tab 按钮的顺序。当用户按下 Tab 键时，输入焦点将从当前控件跳转到下一个可以获得焦点的控件，TabIndex 键就是用于定义这种跳转顺序的。合理使用 TabIndex 属性，可以使用户使用程序更加容易，使得程序更加人性化。如果没有设置 TabIndex 属性，那么该属性值默认为零。如果 Web 控件的 TabIndex 属性值一样，就会以 Web 控件在 ASP.NET 网页中被配置的顺序来决定。

下面代码设置了三个 Button 控件的 TabIndex 属性，由于 B3 的 TabIndex 值最小，所以当用户按下 Tab 键时，输入焦点首先停留在 B3 上，当再按下 Tab 键后，焦点跳转到 B2 上，再次按下 Tab 键焦点将跳转到 B1 上。

```
1. <ASP:Button Id="Button1" Text="B1" TabIndex="3" Runat="Server"/>
2. <ASP:Button Id=" Button2" Text="B2" TabIndex="2" Runat="Server"/>
3. <ASP:Button Id=" Button3" Text="B3" TabIndex="1" Runat="Server"/>
```

第 1 行定义 Button 1 的 TabIndex 属性为 3，第 2 行定义 Button 2 的 TableIndex 属性为 2，第 3 行定义 Button3 的 TableIndex 属性为 1。程序运行时，用户可以通过 Tab 键在这三个控件之间切换，切换的顺序为 Button3→Button2→Button1。

4. 布局属性

服务器控件提供了 Width 和 Hight 属性来控制控件显示的大小，可以使用一个数值加一个度量单位设置这些属性，这些度量单位包括像素（pixels）、百分比等。在设置这些属性时，必须添加单位符号 px（表示像素）或%（百分比）以指明使用的单位类型。

- Height 属性。Height 属性获取或设置 Web 服务器控件的高度。
- Width 属性。Width 属性获取或设置 Web 服务器控件的宽度。

例如：定义一个 Button 控件，并设置属性 BorderWidth、Hight 和 Width 的值来定义 TextBox 控件的边框大小、高度和宽度，代码如下：

```
<asp:Button ID="Button1" runat="server" BorderWidth="2px" Width="200px" Height="50px"></asp: Button >
```

代码说明：这段代码设置了 Button11 控件的属性 BorderWidth 为 2px，表示边框的宽度为 2px，Hight 为 50px，表示高度为 50px；Width 为 200px，表示宽度为 200px。

【例 4-1】本例将学习如何设置控件的外观，创建 12 个 TextBox 控件，前 10 个用于演示不同

的 BorderStyle 属性, 另外两个分别用于演示 BorderColor 属性和 BorderWidth 属性。最后这两个按钮的前景色和背景色都设置为红色背景和蓝色前景。

01 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 4-1”。

02 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 12 个 TextBox 控件到编辑区中。然后切换到“源视图”, 在编辑区中 <form></form> 标记之间编写如下代码。

```
1. <asp:TextBox ID="T6" Text="双实线边框" BorderStyle="Double" BorderWidth="3" runat="Server"/>
2. <asp:TextBox ID="T7" Text="凹槽状边框" BorderStyle="Groove" BorderWidth="3" runat="Server" />
3. <asp:TextBox ID="T8" Text="突起边框" BorderStyle="Ridge" BorderWidth="3" runat="Server" />
4. <asp:TextBox ID="T9" Text="内嵌边框" BorderStyle="Inset" BorderWidth="3" runat="Server" />
5. <asp:TextBox ID="T10" Text="外嵌边框" BorderStyle="Outset" BorderWidth="3" runat="Server" />
6. <asp:TextBox ID="T11" Text="未设置边框样式" runat="Server" />
7. <asp:TextBox ID="T2" Text="无边框" BorderStyle="None" BorderWidth="3" runat="Server" />
8. <asp:TextBox ID="T3" Text="虚线边框" BorderStyle="Dotted" BorderWidth="3" runat="Server" />
9. <asp:TextBox ID="T4" Text="点划线边框" BorderStyle="Dashed" BorderWidth="3" runat="Server" />
10. <asp:TextBox ID="T5" Text="实线边框" BorderStyle="Solid" BorderWidth="3" runat="Server" />
11. <asp:TextBox ID="T11" Text="边框颜色" BorderColor="Blue" BorderWidth="3" BackColor="#ff0000"
    ForeColor="#0000ff" runat="Server" />
12. <asp:TextBox ID="T12" Text="边框宽度" BorderWidth="6" BackColor="Red" ForeColor="Blue" runat="Server" />
```

代码说明: 第 1~10 行分别为文本框 TextBox 设置了不同类型的边框样式的属性, 并将每个样式的名称作为文本显示在文本框内, 同时设置边框的宽度都为 3px。第 11 行设置 TextBox 的边框宽度 BorderWidth 属性为 3px, 边框颜色 BorderColor 属性为蓝色, 同时指定了文本框的前景色和背景色。第 12 行设置了文本框的边框宽度 BorderWidth 属性以及前景色和背景色。对比这两行, 我们可以发现, 直接使用颜色的英文名称或是使用 #RRGGBB 格式设置颜色效果是一样的。

03 按下“Ctrl+F5”, 运行结果如图 4-1 所示。我们看到了各种不同的控件外观和样式。

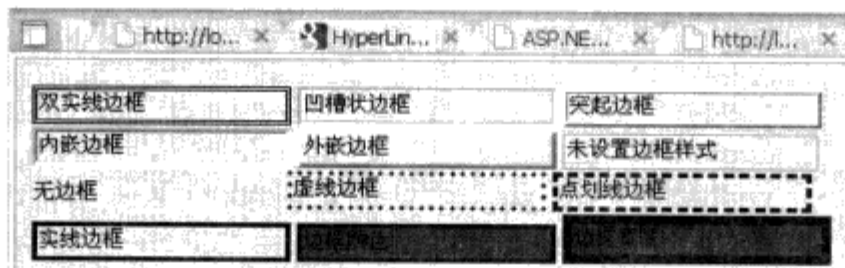


图 4-1 运行结果

4.1.2 服务器控件的事件

在 ASP.NET 页面中, 用户与服务器的交互是通过 Web 控件的事件来完成的, 比如, 当单击一个按钮控件时, 就会触发该按钮的单击事件, 如果程序员在该按钮的单击事件处理函数中编写相应的代码的话, 服务器就会按照这些代码来对用户的单击行为做出响应。

1. 服务器控件的事件模型

Web 控件事件的工作方式与传统 HTML 标记的客户端事件工作方式有所不同, 这是因为 HTML 标记的客户端事件是在客户端引发和处理的, 而 ASP.NET 页面中的 Web 控件的事件是在客户端引发, 在服务器端处理的。

Web 控件的事件模型是：客户端捕捉到事件信息，然后通过 HTTP POST 将事件信息传输到服务器，而且页框架必须解释该 POST 以确定所发生的事件，然后在要处理该事件的服务器上调用代码中的相应方法。

基于以上的事件模型，Web 控件事件可能会影响到页面的性能，因此，Web 控件仅提供有限的一组事件，如表 4-1 所示。

表 4-1 Web 控件事件

事件	支持的控件	功能
Click	Button、ImageButton	单击事件
TextChanged	TextBox	输入焦点变化
SelectedIndexChanged	DropDownList、ListBox、CheckBoxList、RadioButtonList	选择项变化

Web 控件通常不再支持经常发生的事件，如 onmouseover 事件等，因为这些事件如果在服务器端处理的话，就会浪费大量的资源。但 Web 控件仍然可以为这些事件调用客户端处理程序。此外，控件和页面本身在每个处理步骤都会引发生命周期事件，如 Init、Load 和 PreRender 事件，在应用程序中可以利用这些生命周期事件。

所有的 Web 事件处理函数都包括两个参数：第 1 个参数表示引发事件的对象，第 2 个参数表示包含该事件特定信息的事件对象，通常是 EventArgs 类型，或 EventArgs 类型的继承类型。例如按钮的单击事件处理函数，代码如下：

```
1. public void Button1_Click(Object Sender, EventArgs e) { //单击事件处理程序
2.           //在此处添加处理程序
3. }
```

代码说明：第 1 行定义的函数包含两个参数，第 1 个参数 Sender 为引发事件的对象，这里引发该事件的对象就是一个 Button 对象；第 2 个参数 e 为 EventArgs 类型，该类型继承它表示该事件本身。

2. 服务器控件事件的绑定

在处理 Web 控件时，需要把事件绑定到事件处理程序。事件绑定到事件处理程序的方法有两种：

- 在 ASP.NET 页面中，在声明控件时指定该控件的事件对应的事件处理程序，例如把一个 Button 控件的 Click 事件绑定到名为 ButtonClick 的方法，代码如下：

```
<asp:button id="Button1" runat="server" text="按钮" onclick="ButtonClick"/>
```

- 如果控件是被动态创建的，则需要使用代码动态地绑定事件到方法，例如以下代码：

```
1. Button btn= new Button;
2. btn.Text = "提交";
3. btn.Click += new System.EventHandler(ButtonClick);
```

代码说明：这段代码声明了一个按钮控件，并把名为 ButtonClick 的方法绑定到该控件的 Click 事件上。其中，第 1 行定义了一个按钮控件 btn，第 3 行为该控件添加了一个名为 ButtonClick 的单击事件处理程序。



提示

在以声明的方式把事件绑定到事件处理程序时，事件在控件定义标记中都以 on + 事件名称的形式出现，这里对大小写没有要求。

4.2 文本服务器控件

本节主要介绍文本服务器控件，包括标签控件、静态文本控件、文本框控件和超链接文本控件。

4.2.1 标签（Label）控件

Label 服务器控件为开发人员提供了一种以编程方式设置 Web 窗体页中文本的方法。通常当希望在运行时更改页面中的文本时就可以使用 Label 控件。当希望显示的内容不可以被用户编辑时，也可以使用 Label 控件。

Label 控件最常用的 Text 属性用于设置要显示的文本内容，声明 Label 控件的语法定义如下：

1. `<asp:Label id="Label1" Text="要显示的文本内容" runat="server"/>`
2. `<asp:Label id="Label1" Text="要显示的文本内容" runat="server"/></asp:Label>`

代码说明：以上代码是定义 Label 标记的两种方式，属性 ID 定义该控件的标识为 Label，Text 属性表示控件要显示的文字，属性 runat 表示该控件是一个服务器控件。

4.2.2 静态文本（Literal）控件

当要以编程方式设置文本而不添加额外的 HTML 标记时，可以向页面添加 Literal 控件。

Literal 控件声明的语法定义如下：

1. `<asp:Literal id="Literal1" Text="要显示的文本内容" runat="server"/>`
2. `<asp:Literal id="Literal1" Text="要显示的文本内容" runat="server"/></asp:Literal>`

代码说明：以上代码是定义 Literal 标记的两种方式，属性 ID 定义该控件的标识为 Label，Text 属性表示控件要显示的文字，属性 runat 表示该控件是一个服务器控件。

除了前文介绍的基本属性外，Literal 控件还有以下几个重要的属性。

- Text: 获取或设置在 Literal 控件中显示的文本。
- Mode: 设置 Literal 控件文本的显示方式。共有三个选项：Transform 选项不修改 Literal 控件的空文本；PassThrough 选项仅移除文本中不受支持的标记语言元素；Encode 选项对 Literal 控件的文本进行 HTML 编码。

4.2.3 文本框（TextBox）控件

TextBox 控件为用户提供了一种向 Web 窗体页面中输入信息，包括文本、数字和日期的方法。TextBox 控件声明的语法定义有两种，代码如下：

- ```
<asp:TextBox id="TextBox1" runat="server"/>
<asp:TextBox id="TextBox1" runat="server"/></asp:TextBox>
```

TextBox 控件除了所有控件都具有的基本属性之外,还有以下几个重要的属性。

- AutoPostBack: 用于设置在文本修改后,是否自动回传到服务器。它有两个选项, true 表示回传, false 表示不回传,默认为 false。
- Columns: 获取或设置文本框的宽度(以字符为单位)。
- MaxLength: 获取或设置文本框中最多允许的字符数。
- ReadOnly: 获取或设置一个值,用于指示是否可以更改 TextBox 控件的内容。它有两个选项, true 表示只读,不能修改; false 表示可以修改。
- TextMode: 用于设置文本的显示模式。它有三个选项, SingleLine 表示创建只包含一行的文本框。Password 创建用于输入秘密的文本框,用户输入的密码被其他字符替换。MultiLine: 创建包含多个行的文本框。
- Text: 设置和读取 TextBox 中的文字。
- Row: 属性用于获取或设置多行文本框中显示的行数,默认值为 0,表示单行文本框。该属性当 TextMode 属性为 MultiLine(多行文本框模式下)才有效。

TextBox 控件有一个常用 TextChanged 事件,当文本框的内容在向服务器发送时,如果内容和上次发送的不同,就会触发该事件。



提示

一定要将文本框的 AutoPostBack 属性设置为 True,在文本修改后,自动回发到服务器,才能激发 TextChange 事件。

#### 4.2.4 超链接文本(HyperLink)控件

HyperLink 控件用于创建超链接,相当于 HTML 元素的<a>标记。HyperLink 控件声明的语法定义有两种,代码如下:

```
1. <asp:HyperLink ID="HyperLink1" runat="server">HyperLink</asp:HyperLink>
2. <asp:HyperLink ID="HyperLink1" runat="server">HyperLink/>
```

HyperLink 控件除了基本属性之外,还有以下几个重要的属性。

- Text: 用于设置或获取 HyperLink 控件的文本内容。
- NavigateURL: 用设置或获取单击 HyperLink 控件时链接到的 URL。
- Target: 用于设置或获取目标链接要显示的位置,有如下的值可选, \_blank 表示在新窗口中显示目标链接的页面, \_parent 表示将目标链接的页面显示在上一个框架集父级中, \_self 表示将目标链接的页面显示在当前的框架中, \_top 表示将内容显示在没有框架的全窗口中;页面可以是自定义的 HTML 框架的名称。
- ImageUrl: 用于设置或获取显示为超链接图像的 URL。

**【例 4-2】**通过本例学习文本服务器控件的使用。用户在文本框中输入的网址,通过 TextBox 控件的 TextChanged 事件将文本框接受的值传递到 HyperLink 控件,该控件在获得值之前进行隐藏,获得值之后才将网址显示在页面,用户单击该网址链接可进入该网站浏览。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-2”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 Label 控件、1 个 TextBox 控件和一个到 HyperLink 控件到编辑区中。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. <asp:Label ID="Label1" runat="server" Text="请输入网址：" /></asp:Label>
2. <asp:TextBox ID="TextBox1" runat="server" BorderStyle="Inset" BorderWidth="4" AutoPostBack="true"
 ontextchanged="TextBox1_TextChanged" /></asp:TextBox>

3. <asp:HyperLink ID="HyperLink1" runat="server" Visible="false" />HyperLink</asp:HyperLink>
```

代码说明：第 1 行添加一个服务器标签控件 Label1 并设置显示的文本属性 Text。第 2 行添加一个服务器文本框控件 TextBox1，设置边框样式 BorderStyle 和宽度 BorderWidth 属性；设置 AutoPostBack 属性为 true，表示文本修改后自动回发到服务器；设置文本框中的文本被修改后触发事件 TextBox1\_TextChanged。第 3 行添加一个服务器超链接文本控件 HyperLink1，并设置 Visible 属性为不可见。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void TextBox1_TextChanged(object sender, EventArgs e){
2. HyperLink1.Visible = true;
3. HyperLink1.Text = TextBox1.Text;
4. HyperLink1.NavigateUrl = TextBox1.Text;
5. }
```

代码说明：第 1 行定义处理文本框控件 TextBox1 的文本内容改变事件的方法 TextBox1\_TextChanged。第 2 行设置超链接文本控件 HyperLink1 的 Visible 属性为可见。第 3 行将用户输入在文本框中的内容在显示在超链接文本控件上。第 4 行将 HyperLink1 控件的被单击链接的 URL 设置为用户输入的网址。

**04** 按下“Ctrl+F5”，运行结果如图 4-2 所示，注意此时页面没有出现超链接。用户在文本框中输入网址后按“Enter”键。

**05** 页面出现了出现如图 4-3 所示的网址超链接，单击该链接。

**06** 页面跳转如图 4-4 所示的开心网首页。



图 4-2 运行结果 1

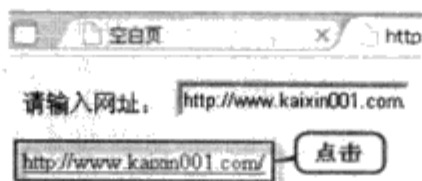


图 4-3 运行结果 2



图 4-4 运行结果 3



提示

虽然可以利用 Visual Studio 2010 的拖拽功能改变控件的大小，但有时为了精确定义控件的大小，需要利用单位来为控件的高度、宽度等属性设置大小。

## 4.3 按钮服务器控件

用户在访问网页时常常需要在特定的时候激发某个动作来完成一系列的操作,使用按钮控件就可以实现这个功能。在服务器控件中包括三种按钮控件: Button、LinkButton 和 ImageButton。

### 4.3.1 普通按钮 (Button) 控件

Button 按钮控件是一种常见的单击按钮传递信息的方式,能够把页面信息返回到服务器。Button 控件的声明语法有两种,代码如下。

1. `<asp:Button ID="Button1" runat="Server" Text="按钮"></asp:Button>`
2. `<asp:Button ID="Button1" runat="Server" Text="按钮"/>`

Button 控件除了基本属性之外,还有以下几个重要的属性和事件。

- Text: 设置或获取在 Button 控件上显示的文本内容,用来提示用户进行何种操作。
- CommandName: 用于设置和获取 Button 按钮将要触发事件的名称。当有多个按钮共享一个事件处理函数时,通过该属性来区分要执行哪个 Button 事件。
- CommandArgument: 用于指示命令传递的参数,提供有关要执行命令的附加信息,以便在事件中进行判断。
- OnClick 事件: 当用户单击按钮时要执行的事件处理方法。

### 4.3.2 超链接按钮 (LinkButton) 控件

LinkButton 控件是一个超链接按钮控件,它是一种特殊的按钮,其功能和普通按钮控件 Button 类似,但是该控件是以超链接的形式显示的。LinkButton 控件外观和 HyperLink 相似,功能和 Button 相同。HyperLink 控件声明的语法定义有两种,代码如下:

1. `<asp:LinkButton ID="LinkButton1" runat="Server" Text="按钮"></asp:LinkButton>`
2. `<asp:LinkButton ID="LinkButton1" runat="Server" Text="按钮"/>`

LinkButton 控件的 Text 属性用于设置控件上的文字按钮,OnClick 事件是当用户单击按钮时的事件处理函数。

### 4.3.3 图片按钮 (ImageButton) 控件

ImageButton 控件是一个显示图片的按钮,其功能和普通按钮 Button 类似,但是 ImageButton 控件是以图片形式显示的,其外观与 Image 控件相似,但功能与 Button 相同。ImageButton 控件声明的语法定义有两种,代码如下:

1. `<asp:ImageButton ID="ImageButton1" runat="Server" Text="按钮"></asp:ImageButton>`
2. `<asp:ImageButton ID=" " runat="Server" Text="按钮"/>`

ImageButton 控件除了基本的属性之外,其他重要的常用方法和事件如下:

- ImageUrl: 用于设置和获取在 ImageButton 控件中显示的图片位置。

- OnClick 事件：用户单击按钮后的事件处理函数。

**【例 4-3】**通过本例学习 ImageButton 控件的使用。在网站中一般都有一个友情链接的功能，通过单击这些链接可以方便地到和本网站密切相关的网站浏览。我们在友情链接中使用 ImageButton 控件显示三个搜索网站的图片，用户单击图片可进入该网站浏览。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-3”。
- 02** 在程序中创建一个“Image”文件夹，并放置三张网站的图片文件。
- 03** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 3 个 ImageButton 控件。然后切换到“源视图”，在编辑区中 `<form></form>` 标记之间编写如下代码。

```

1. <table style="width: 13%; text-align:center" border="1">
2. <tr><td class="style2">友情链接</td></tr>
3. <tr><td class="style4">
4. <asp:ImageButton ID="ImageButton1" runat="server" ImageUrl="~/Image/百度.JPG" onclick="ImageButton1_Click"
 /></td>
5. </tr>
6. <tr><td class="style4">
7. <asp:ImageButton ID="ImageButton3" runat="server" ImageUrl="~/Image/搜狗.JPG" Width="194px" onclick=
 "ImageButton3_Click"/></td>
8. </tr>
9. <tr><td class="style4">
10. <asp:ImageButton ID="ImageButton2" runat="server" ImageUrl="~/Image/谷歌.JPG" Width="194px"
 onclick="ImageButton2_Click"/></td>
11. </tr>
12. </table>

```

代码说明：第 1~12 行设计了一个 1 行 4 列的表格。第 4、7 和 10 行分别添加了三个图片按钮控件放到下面的三个单元格中，同时分别设置其 ImageUrl 属性获取图片路径以及触发单击的事件 Click。

- 04** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void ImageButton1_Click(object sender, ImageClickEventArgs e){
2. Response.Redirect("http://www.baidu.com");
3. }
4. protected void ImageButton3_Click(object sender, ImageClickEventArgs e){
5. Response.Redirect("http://www.sogou.com/");
6. }
7. protected void ImageButton2_Click(object sender, ImageClickEventArgs e){
8. Response.Redirect("http://www.google.com");
9. }

```

代码说明：第 1、4 和 7 行分别定义处理 3 个 ImageButton 单击事件 Click 的方法。第 2、5 和 8 行分别使用 Response 对象的 Redirect 方法，将网址作为参数以实现跳转页面到相应网站的功能。

**05** 按下“Ctrl+F5”，运行结果如图 4-5 所示。单击所选网站图片可以进入相应的网站浏览。



图 4-5 运行结果

## 4.4 图像服务器控件

在网页设计中，图片元素是不可或缺的，在服务器控件中提供了有关图像操作的两个控件：Image 控件和 ImageMap 控件，本节将对这两个控件进行介绍。

### 4.4.1 图像（Image）控件

Image 控件是用于显示图像的，相当于 HTML 标记语言中的<img>标记，Image 控件的声明方法有两种，代码如下：

```
1. <asp: Image ID= "Image1" runat="Server" ></asp: Image>
2. <asp: Image ID= "Image1" runat="Server"/>
```

Image 控件除了一些基本的属性外，还有如下几个重要的属性。

- ImageUrl: 用于设置和获取在 Image 控件中显示图片的路径。
- AlternateText: 获取和设置当图像不可用时，在 Image 控件中显示替换的文本。
- ImageAlign: 用于获取和设置 Image 控件相对于网页中其他元素的对齐方式。共有以下九种值可供选择。
  - Left: 图像沿网页的左边缘对齐，文字在图像右边换行。
  - Right: 图像沿网页的右边缘对齐，文字在图像左边换行。
  - BaseLine: 图像的下边缘与第一行文本的下边缘对齐。
  - Top: 图像的上边缘与同一行上最高元素的上边缘对齐。
  - Middle: 图像的中间于第一行文本的下边缘对齐。
  - Bottom: 图像的下边缘与第一行文本的下边缘对齐。
  - AbsBottom: 图像的下边缘与同一行中最大元素的下边缘对齐。
  - AbsMiddle: 图像的中间与同一行中最大元素的中间对齐。
  - TextTop: 图像的上边缘与同一行上最高文本的上边缘对齐。

### 4.4.2 图像地图（ImageMap）控件

ImageMap 控件是一个可以在地图上定义热点（HotSpot）区域的服务器控件，用户可以通过单击这些热点区域进行回传操作或者定向到某个 URL 位置。该控件通常用在需要对某张图片的局部范围进行互动操作的时候。Image 控件的声明方法有两种，代码如下：

1. `<asp: ImageMap ID= "ImageMap1" runat="Server" ></asp: ImageMap>`
2. `<asp: ImageMap ID= "ImageMap" runat="Server"/>`

ImageMap 控件除了一些基本的属性外，还有如下几个重要的属性。

- ImageUrl: 用于设置和获取在 ImageMap 控件中显示的图像的路径。
- AlternateText: 获取和设置当图像不可用时，在 Image 控件中显示替换的文本。
- ImageAlign: 用于获取和设置图像上热点区域位置和链接文件。

在 ImageMap 控件中设置热点区域的方法如下。

**01** 在如图 4-6 所示的“属性”窗口中单击 HotSpots 属性右侧的按钮。

**02** 打开如图 4-7 所示的“HotSpot 集合编辑器”对话框。单击“添加”按钮右侧的向下箭头，会弹出热区形状选择列表，包括圆形热区（CircleHotSpot）、矩形热区（RectangleHotSpot）和多边形热区（PolygonHotSpot），默认的是圆形热区。然后，单击“添加”按钮可向成员列表中添加热区。在右边的熟悉列表中，可以设置热区的形状、文件链接的路径。

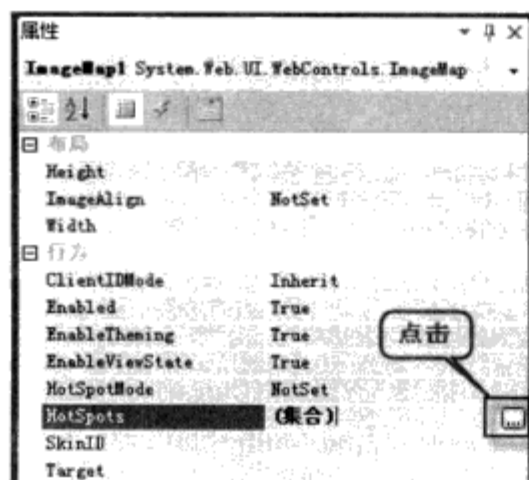


图 4-6 属性窗口



图 4-7 HotSpot 集合编辑器

**03** 在属性中有一个 HotSpotMode 用于设置图像上的热区的类型，对应的枚举类型是 System.Web.UI.WebControls.HotSpotMode，它有三种枚举值。

- NotSet: 默认值，会执行定向操作，定向到用户指定的 URL 地址去。如果用户未指定 URL 位置，那么将定向到其 Web 应用程序根目录。
- PostBack: 回传操作。单击热区后，将执行后部的 Click 事件。
- Inactive: 无任何操作，即此时形同一张没有热区的普通图片。

**04** 最后单击“确定”按钮即可完成热区的设置。

**【例 4-4】**所有的网站都具备图片浏览的功能。本实例就通过使用 LinkButton 控件和 Image 图像控件来实现多个图像的浏览功能。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-4”。

**02** 在程序中创建一个“Image”文件夹，并放置 4 张风景图片文件。

**03** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 Image 控件和 4 个 LinkButton 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. <asp:Image ID="Image1" runat="server" ImageUrl = "~/Image/桂林.jpg"/>

2. <asp:LinkButton ID="HyperLink1" runat="server" onclick="HyperLink1_Click">1</asp:LinkButton>
3. <asp:LinkButton ID="HyperLink2" runat="server" onclick="HyperLink2_Click">2</asp:LinkButton>
4. <asp:LinkButton ID="HyperLink3" runat="server" onclick="HyperLink3_Click">3</asp:LinkButton>
5. <asp:LinkButton ID="HyperLink4" runat="server" onclick="HyperLink4_Click">4</asp:LinkButton>
```

代码说明：第 1 行添加了一个服务器图像控件 Image1 并设置了显示控件上图像路径的属性 ImageUrl。第 2~5 行分别添加了 4 个服务器超链接按钮控件 LinkButton，并设置各自触发单击事件的方法 Click。

**04** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void HyperLink1_Click(object sender, EventArgs e){
2. Image1.ImageUrl = "~/Image/桂林.jpg";
3. }
4. protected void HyperLink2_Click(object sender, EventArgs e){
5. Image1.ImageUrl = "~/Image/故宫.jpg";
6. }
7. protected void HyperLink3_Click(object sender, EventArgs e){
8. Image1.ImageUrl = "~/Image/拉萨.jpg";
9. }
10. protected void HyperLink4_Click(object sender, EventArgs e){
11. Image1.ImageUrl = "~/Image/神农架.jpg";
12. }
```

代码说明：第 1、第 4、第 7 和第 10 行分别定义了四个处理超链接按钮单击事件 Click 的方法。第 2、5、8 和 11 行设置单击超链接按钮时 Image 控件显示不同的图片，通过使用 ImageUrl 属性关联要显示的图片的路径来实现。

**05** 按下“Ctrl+F5”，运行结果如图 4-8 所示。

**06** 单击如图 4-9 所示的图片下面的不同的数字链接，显示不同的图片。



图 4-8 运行结果 1



图 4-9 运行结果 2

## 4.5 选择服务器控件

在 Web 页面中，经常需要从多个信息中选择其中一个或几个需要的数据，如选择爱好和性别等。服

务器控件中的 CheckBox、CheckBoxList、RadioButton、RadioButtonList 四种控件正是用于这些场合的。

### 4.5.1 复选框 (CheckBox) 控件

CheckBox 控件用于在 Web 窗体中创建复选框，该复选框允许用户在 True 和 False 之间切换，提供用户从选项中进行多项选择的功能。CheckBox 控件声明方法有两种，代码如下：

```
1. <asp:CheckBox ID="CheckBox1" runat="Server" ></asp:CheckBox>
2. <asp:CheckBox ID="CheckBox1" runat="Server"/>
```

CheckBox 控件除了一些基本的属性外，其他常用的属性和事件如下。

- AutoPostBack: 设置或获取一个值布尔，该值表示在单击 CheckBox 控件时状态是否回传到服务器。默认值是 false。
- Checked: 获取或设置一个值，该值指示是否已选中 CheckBox 控件。该值只能是 True (选中) 或 False (取消选中)。
- Text: 获取或设置与 CheckBox 关联的文本标签。
- TextAlign: 获取或设置与 CheckBox 控件关联的文本标签的对齐方式。该值只有 Left 和 Right，指定文本标签是显示在复选框的右边还是左边，默认为 Right。
- CheckedChanged 事件: 当 Checked 属性的值在向服务器进行发送期间更改时发生，即当从选择状态变为取消选择或从未选中状态到选中状态时发生。



提示

默认情况下，单击 CheckBox 控件时不会自动向服务器发送窗体。如要启用自动发送，应将 AutoPostBack 设置为 true。

### 4.5.2 复选框列表 (CheckBoxList) 控件

CheckBoxList 控件用于在 Web 窗体中创建复选框组，它是一个 CheckBox 的集合。CheckBoxList 控件声明方法有两种，代码如下：

```
1. <asp:CheckBoxList ID="CheckBoxList1" runat="Server" ></asp:CheckBoxList>
2. <asp:CheckBoxList ID="CheckBoxList1" runat="Server"/>
```

设置 CheckBoxList 控件中的 CheckBox 成员项的方法如下。

(1) 将鼠标移到 CheckBoxList 控件上，其上方会出现如图 4-10 所示一个向右的黑色小三角。单击它，弹出“CheckBoxList 任务”列表，选择“编辑项”。

(2) 打开如图 4-11 所示的“ListItem 集合编辑器”对话框。单击“添加”按钮可向“成员”列表中添加选项，并在“属性”列表中设置选项的 Text 属性，然后单击“确定”按钮。如果要将选项设置为选中状态，可以将 Selected 属性设置为“true”。

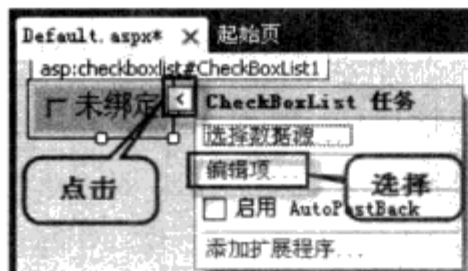


图 4-10 “CheckBoxList 任务”列表

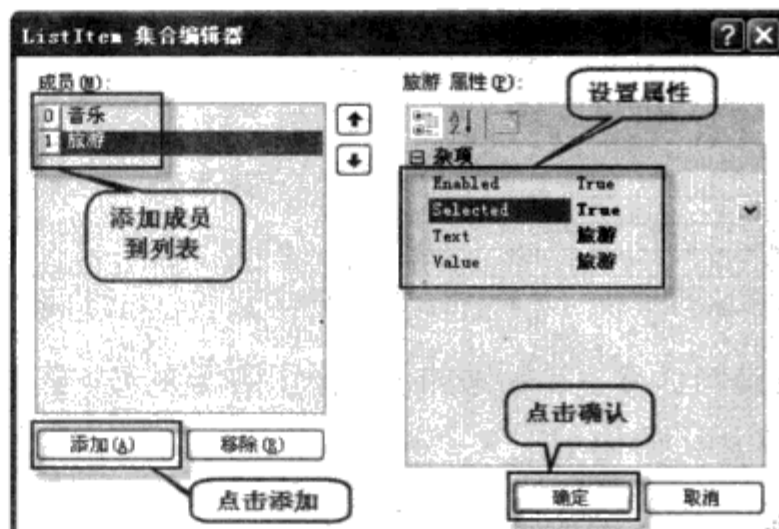


图 4-11 “ListItem 集合编辑器”对话框

CheckBoxList 控件除了一些基本的属性外，其他常用的属性和事件如下。

- AutoPostBack: 获取或设置一个值，该值指示当用户更改列表中的选定内容时是否自动产生向服务器的回发。
- CellPadding: 获取或设置表单元格的边框和内容之间的距离（以像素为单位）。
- DataSource: 获取或设置对象，数据绑定控件从该对象中检索其数据项列表。
- DataTextField: 获取或设置为列表项提供文本内容的数据源字段。
- DataValueField: 获取或设置为各列表项提供值的数据源字段。
- Items: 获取列表控件项的集合。
- RepeatColumns: 获取或设置要在 CheckBoxList 控件中显示的列数。
- RepeatDirection: 获取或设置一个值，该值指示 CheckBoxList 控件是垂直显示还是水平显示。
- RepeatLayout: 获取或设置 CheckBoxList 控件的 ListItem 排列方式是 Table 排列还是直接排列。
- SelectedIndex: 获取或设置列表中选定项的最低序号索引。
- SelectedItem: 获取列表控件中索引最小的选定项。
- SelectedValue: 获取列表控件中选定项的值，或选择列表控件中包含指定值的项。
- TextAlign: 获取或设置组内复选框的文本对齐方式。

**【例 4-6】** 在网络考试系统中一般有多选题，在实现多选题回答时通常使用 CheckBoxList 控件。本例就通过使用 CheckBoxList 控件和 Button 控件完成对多选题的回答，并将回答的结果显示在页面上。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-6”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 CheckBoxList 控件和 1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

1. 多选题<br /><br />
2. 面向编程对象设计的三大特点是：<br />

```

3. <asp:CheckBoxList ID="CheckBoxList1" runat="server" >
4. <asp:ListItem Value="0">封装</asp:ListItem>
5. <asp:ListItem Value="1">继承</asp:ListItem>
6. <asp:ListItem Value="2">多态</asp:ListItem>
7. <asp:ListItem Value="3">重载</asp:ListItem>
8. </asp:CheckBoxList>
9. <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="提交" />
10. <asp:Label ID="Label1" runat="server" Text="Label" Visible="False"></asp:Label>

```

代码说明：第3~8行添加了一个服务器复选框列表控件 CheckBoxList1。其中，第4~7行分别向 CheckBoxList1 添加了4个列表成员并添加显示的文本和设置了值。第9行添加了一个服务器按钮控件 Button1 并设置其触发的单击事件 Click 以及所显示的文本的属性 Text。第10行添加了一个服务器标签控件 Label1 并设置其 Visible 属性为不可见。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. Label1.Visible = true;
3. if (CheckBoxList1.Items[0].Selected && CheckBoxList1.Items[1].Selected && CheckBoxList1.Items[2].Selected)
4. Label1.Text = "恭喜你，答对了！";
5. else
6. Label1.Text = "对不起，你答错了！";
7. }

```

代码说明：第1~7行定义处理提交按钮单击事件 Click 的方法。其中，第2行将标签控件 Visible 属性设置为可见。第3行判断如果复选框列表控件 CheckBoxList1 的第1、第2和第3项内容同时被选中时，在 Label 控件上显示回答正确的提示。否则，在页面显示回答错误的提示。

**04** 按下“Ctrl+F5”，运行结果如图4-12所示。选择答案后，单击“提交”按钮。

**05** 页面显示如图4-13所示的答题结果正确与否。

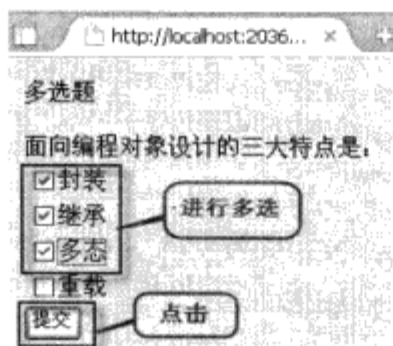


图 4-12 运行结果 1

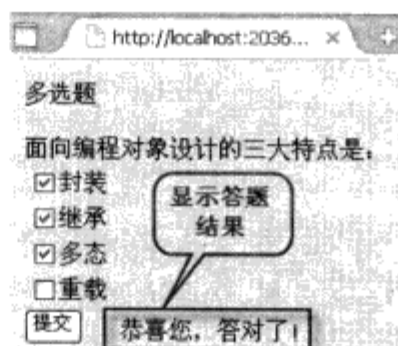


图 4-13 运行结果 2



提示

如果想用数据库中的数据创建一系列的复选框，则 CheckBoxList 控件是较好的选择。但是，如果想更好地控制页面上复选框的布局时，则选择 CheckBox 控件较为适宜。

### 4.5.3 单选按钮（RadioButton）控件

有时候需要提供一组互相排斥的选项，例如最高学历只能选择一个。服务器控件为我们提供了 RadioButton 控件用于在 Web 窗体中创建一个单选按钮，可以将多个单选按钮分为一组以提供一组互相排斥的选项，用户一次只能选中一个。RadioButton 控件声明方法有两种，代码如下：

1. `<asp: RadioButton ID= "RadioButton1" runat="Server" ></asp: RadioButton>`
2. `<asp: RadioButton ID= "RadioButton1 runat="Server" />`

RadioButton 控件除了一些基本的属性外, 其他常用的属性和事件如下。

- AutoPostBack: 获取或设置一个值, 该值指示在单击 RadioButton 控件时状态是否自动回发到服务器。
- Checked: 获取或设置一个值, 该值指示是否已选中 CheckBox 控件。该值只能是 True(选中)或 False(取消选中)。
- GroupName: 获取或设置单选按钮所属的组名。
- TextAlign: 获取或设置与 RadioButton 控件关联的文本标签的对齐方式。该值只有 Left 和 Right, 指定文本标签是显示在单选框的右边还是左边。默认为 Right。
- Text: 获取或设置与 RadioButton 控件关联的文本标签。
- CheckedChanged 事件: 当 Checked 属性的值在向服务器进行发送期间更改时发生。

#### 4.5.4 单选按钮列表 (RadioButtonList) 控件

RadioButtonList 控件是一个单选按钮列表框控件, 也就是一个 RadioButton 控件的集合。该控件与 CheckBoxList 控件类似, 只是该控件用于单项选择, 而 CheckBoxList 控件用于多项选择。RadioButtonList 控件可以直接添加选项或者通过绑定数据来添加选项。当希望单独设置 RadioButton 的布局 and 外观时, 可以使用 RadioButton 控件。但要使用多个 RadioButton 时, 就最好使用 RadioButtonList 控件。RadioButtonList 控件声明方法有两种, 代码如下:

1. `<asp: RadioButtonList ID="RadioButtonList1" runat="Server" ></asp: RadioButtonList>`
2. `<asp: RadioButtonList ID= "RadioButtonList1 runat="Server" />`

向 RadioButtonList 控件中添加 RadioButton 成员的操作和 CheckBoxList 控件类似, 这里不再重复。RadioButtonList 控件的常用属性和方法如下。

- RepeatColumns: 获取或设置要在 RadioButtonList 控件中显示的列数。
- RepeatDirection: 获取或设置一个值, 该值指示 RadioButtonList 控件是垂直显示还是水平显示。
- RepeatLayout: 获取或设置组内单选按钮的布局。
- SelectedIndex: 获取或设置列表中选定项的最低序号索引。
- SelectedItem: 获取列表控件中索引最小的选定项。
- SelectedValue: 获取列表控件中选定项的值, 或选择列表控件中包含指定值的项。
- SelectedIndexChanged 事件: 当列表控件的选定项在信息发往服务器之间变化时发生。
- DataBinding: 当服务器控件绑定到数据源时发生。

**【例 4-5】**在网络考试系统中除了有多选题, 一般还会有单选题, 在实现单选题回答时通常使用 RadioButtonList 控件。本例就通过使用 RadioButtonList 控件和 Label 控件完成对单选题的回答, 并将回答的结果显示在页面上。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 4-5”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 RadioButtonList 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. 单选题

2. 下面那一本书不属于中国古代四大名著?

3. <asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True" onselectedindexchanged=
 "RadioButtonList1_SelectedIndexChanged">
4. <asp:ListItem Value="0">A 三国演义</asp:ListItem>
5. <asp:ListItem Value="1">B 西游记</asp:ListItem>
6. <asp:ListItem Value="2">C 红楼梦</asp:ListItem>
7. <asp:ListItem Value="3">隋唐演义</asp:ListItem>
8. </asp:RadioButtonList>
9.

10. <asp:Label ID="Label1" runat="server" Text="Label" Visible="False"></asp:Label>

```

代码说明：第 3~8 行添加了一个服务器单框列表控件 RadioButtonList1 并设置 AutoPostBack 属性的值为 true，表示需要自动回传到服务器。其中，第 4~7 行分别向 RadioButtonList1 添加了 4 个列表成员并设置 Value 属性的值同时显示的文本。第 10 行添加了一个服务器标签控件 Label1 并设置其 Visible 属性为不可见。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e){
2. Label1.Visible = true;
3. if (int.Parse (RadioButtonList1.SelectedValue) == 3)
4. Label1.Text = "恭喜你，选择正确！";
5. else
6. Label1.Text = "对不起，你的选择错误！";
7. }

```

代码说明：第 1~7 行定义处理服务器单框列表控件 RadioButtonList1 选项改变事件 SelectedIndexChanged 的方法。其中，第 2 行将标签控件 Visible 属性设置为可见。第 3 行判断如果单选框列表控件中选中项的值是等于 3 时，第 4 行在页面 Label 控件上显示回答正确的提示。否则，显示回答错误的提示。

**04** 按下“Ctrl+F5”，运行结果如图 4-14 所示，选择答案后，页面显示答题结果正确与否。

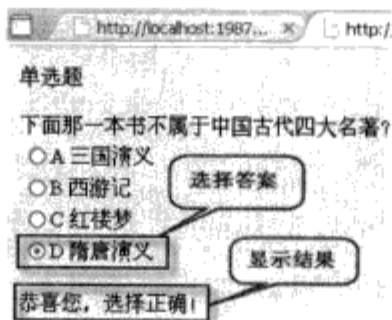


图 4-14 运行结果 1

## 4.6 列表服务器控件

上一节介绍的几个用于选择的控件，都是在选择项目比较少的时候使用的。比如购物网站的商品成千上万，如果仍然使用 RadioButton 和 CheckBox 这种类型的控件，页面布局会变得非常困难，可以使用这一节将要介绍的 DropDownList、BulletedList 和 ListBox 这三种列表控件。

### 4.6.1 列表框 (ListBox) 控件

ListBox 控件是一个静态的列表框，用户可以在该控件中添加一组内容列表，以供访问网页的用户选择其中的一项和或多项。ListBox 控件声明方法有两种，代码如下：

```
1. <asp:ListBox ID="ListBox1" runat="Server" ></asp:ListBox>
2. <asp:ListBox ID="ListBox1" runat="Server"/>
```

ListBox 控件中的可选项目是通过 ListItem 元素定义的，该控件支持数据绑定。该控件添加到页面后，设置列表项的方法和 CheckBoxList 控件相同。

ListBox 控件除了基本属性之外，还有以下几个重要的属性和事件。

- AutoPostBack: 获取或设置一个值，该值指示当用户更改列表中的选定内容时是否自动产生向服务器的回发。
- DataSource: 获取或设置对象，数据绑定控件从该对象中检索其数据项列表。
- DataTextField: 获取或设置为列表项提供文本内容的数据源字段。
- DataValueField: 获取或设置为各列表项提供值的数据源字段。
- Items: 获取列表控件项的集合，每一个项的类型都是 ListItem。
- Rows: 获取或设置 ListBox 控件中显示的行数。
- SelectedIndex: 获取或设置列表选定项的最低序号索引。
- SelectedItem: 获取列表控件中索引最小的选定项。
- SelectedValue: 获取列表控件中选定项的值，或选择列表控件中包含指定值的项。
- SelectionMode: 使用 SelectionMode 属性指定 ListBox 控件的模式行为。将该属性设置为“Single”表示只能从 ListBox 控件中选择一项，而“Multiple”表示可选择多项。
- Count: 表示列表项中条目的总数。
- Selected: 表示某个项被选中。
- ClearSelected: 取消选择是 ListBox 中的所有项。
- GetSelected: 返回一个值，该值指示是否选定了指定的项。
- Sort: 对 ListBox 中项进行排序。

**【例 4-7】** 本例实现两个列表框的级联变化过程。第一个 ListBox 显示网站类型，另一个显示具体的网站，显示网站的列表框会根据显示网站类型的列表框中所选的值的不同而显示不同的网站，并且网站可多选。选择完毕后，单击“确定”按钮，就可以在页面显示所选的网站类型和具体网站。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-7”。

02

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.
- 11.
- 12.

代码

03

- 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.

```

31. string temp="->";
32. for (int i = 0; i < ListBox2.Items.Count; i++){
33. if (ListBox2.Items[i].Selected){
34. Label1.Text += temp + ListBox2.Items[i].Text;
35. temp = ", ";
36. }
37. }
38. }

```

代码说明：第 1 行定义处理 Page 对象加载事件 Load 的方法。第 2 行判断当前页面如果不是回传页面，则第 3~5 行在列表框控件 ListBox2 中使用 Items.Add 方法添加三个列表项。第 8 行定义处理列表框控件 ListBox1 的选中项改变事件的方法 SelectedIndexChanged。第 9 行将 ListBox2 的列表项清空。第 10~25 行使用 switch-case 语句判断 ListBox1 中当前选中项的值，根据不同的值，在 ListBox2 中添加不同列表项。第 27 行定义处理按钮控件 Button1 的单击事件 Click 的方法。第 28 行将标签控件 Visible 属性设置为可见。第 29~36 行将用户选中项的信息拼接成字符串显示在标签上；其中，第 32~36 行使用 for 循环语句遍历 ListBox2 列表项集合，只要有列表项被选中，就将其拼接到字符串中。

**04** 按下“Ctrl+F5”，运行结果如图 4-15 所示。

**05** 在图 4-15 中的网站类别列表框中选择相应类别，同时在选择网站列表框中显示类别包含的具体网站。可以选择一个和多个，然后单击“确定”按钮。页面显示如图 4-16 所示的用户选择的具体内容。

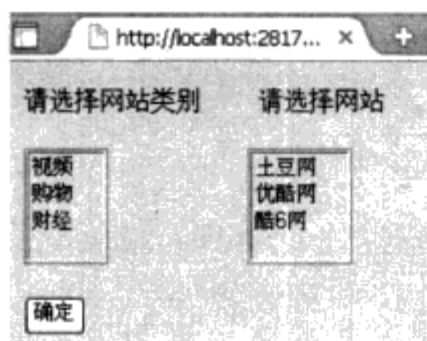


图 4-15 运行结果 1

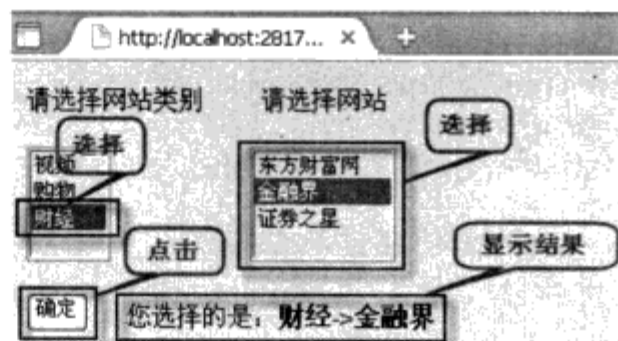


图 4-16 运行结果 2

## 4.6.2 下拉列表框 (DropDownList) 控件

DropDownList 控件是一个下拉列表框控件，该控件与 ListBox 控件类似，也可以选择一项或多项内容，只是它们的外观不同。DropDownList 控件有一个下拉列表框，而 ListBox 控件是在静态列表中显示内容。

DropDownList 控件可以直接设置选项，也可以通过绑定数据来设置选项，其设置选项的绑定数据的方法与 ListBox 控件相识。DropDownList 控件声明方法有两种，代码如下：

```

1. <asp: DropDownList ID= "DropDownList1" runat="Server" ></asp: DropDownList>
2. <asp: DropDownList ID= "DropDownList" runat="Server"/>

```

DropDownList 控件除了基本属性之外，还有以下几个重要的属性和事件。

- **AutoPostBack**: 获取或设置一个值，该值指示当用户更改列表中的选定内容时是否自动产生向服务器的回发。

- **DataSource**: 获取或设置对象，数据绑定控件从该对象中检索其数据项列表。
- **DataSourceField**: 获取或设置为列表项提供文本内容的数据源字段。
- **DataValueField**: 获取或设置为各列表项提供值的数据源字段。
- **Items**: 获取列表控件项的集合，每一个项的类型都是 `ListItem`。在“属性”窗口中单击该属性的按钮，可以打开“`ListItem` 集合编辑器”对话框来设置列表项。
- **SelectedIndex**: 返回被选取到的 `ListItem` 的索引项。
- **SelectedItem**: 获取列表控件中索引最小的选定项。
- **SelectedValue**: 获得列表框中被选中的值。

**【例 4-8】** 在一些旅游网站，都具有了预定酒店的功能。本例就通过 DropDownList 控件实现酒店预定功能中用户选择到店和离店日期的操作。当用户提交选择后，将日期显示在页面上。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-8”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 6 个 DropDownList 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 请选择预定的时间：

2. 到店时间：<asp:DropDownList ID="DropDownList1" runat="server"> </asp:DropDownList>年
3. <asp:DropDownList ID="DropDownList2" runat="server"> </asp:DropDownList>月
4. <asp:DropDownList ID="DropDownList3" runat="server"> </asp:DropDownList>日

5. 离店时间：<asp:DropDownList ID="DropDownList4" runat="server"> </asp:DropDownList>年
6. <asp:DropDownList ID="DropDownList5" runat="server"></asp:DropDownList>月
7. <asp:DropDownList ID="DropDownList6" runat="server"></asp:DropDownList>日

8. <asp:Button ID="Button1" runat="server" Text="确定 onclick="Button1_Click" />

9. <asp:Label ID="Label1" runat="server" Text="Label" Visible="False"></asp:Label>
```

代码说明：第 2、3 和 4 行分别添加了 3 个服务器下拉列表控件 `DropDownList`。显示到店的年、月和日的列表项内容。第 5、6 和 7 行分别添加了 3 个服务器下拉列表控件 `DropDownList` 显示离店日期的年、月和日列表项内容。第 8 行添加一个服务器按钮控件 `Button1`，并设置其单击事件 `Click` 和显示的文本。第 9 行添加一个服务器标签控件 `Label1` 并设置 `Visible` 属性为不可见。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. for (int i = 2010; i <= 2020; i++){
4. DropDownList1.Items.Add(i.ToString());
5. DropDownList4.Items.Add(i.ToString());
6. }
7. for (int i = 1; i <= 12; i++){
8. DropDownList2.Items.Add(i.ToString());
9. DropDownList5.Items.Add(i.ToString());
10. }
11. for (int i = 1; i <= 31; i++){
12. DropDownList3.Items.Add(i.ToString());
13. DropDownList6.Items.Add(i.ToString());
14. }
15. }
16. }

```

```

17. protected void Button1_Click(object sender, EventArgs e){
18. Label1.Visible = true;
19. Label1.Text = "你到店的时间是: " + DropDownList1.Text + "年" + DropDownList2.Text + "月" +
 DropDownList3.Text + "日";
20. Label1.Text += "
你离店的时间是: " + DropDownList4.Text + "年" + DropDownList5.Text + "月" +
 DropDownList6.Text + "日";
21. }

```

代码说明：第1行定义处理 Page 对象加载事件 Load 的方法。第2行判断当前页面如果不是回传页面，则第3~6行使用 for 循环在表示到店和离店年份的两个服务器下拉列表控件 DropDownList 中使用 Items.Add 方法添加 2010 到 2020 的年份。第7~9行使用 for 循环在表示到店和离店月份的两个服务器下拉列表控件 DropDownList 中使用 Items.Add 方法添加 1 到 12 的月份。第11~14行使用 for 循环在表示到店和离店日期的两个服务器下拉列表控件 DropDownList 中使用 Items.Add 方法添加 1 到 31 的天数。第17行定义处理按钮控件 Button1 的单击事件 Click 的方法。第18行将标签控件设置为可见。第19~20行通过 DropDownList 控件的 Text 属性获得每个下拉列表框中用户所选中项的值，进行字符串的拼接将结果显示在 Label1 控件上。

**04** 按下“Ctrl+F5”，运行结果如图 4-17 所示。选择 6 个下拉列表中到店和离店的具体日期，单击“确定”按钮。

**05** 在页面中显示如图 4-18 所示的用户选择结果。



图 4-17 运行结果 1



图 4-18 运行结果 2

### 4.6.3 项目列表 (BulletedList) 控件

BulletedList 控件是一个显示项目列表的控件，用于创建以项目符号格式化的列表项，而且还可以显示为超链接列表。该控件可以直接添加到列表项，也可以通过绑定数据设置列表项。BulletedList 控件声明方法有两种，代码如下：

```

1. <asp: BulletedList ID= "BulletedList1" runat="Server" ></asp: BulletedList>
2. <asp: BulletedList ID= "BulletedList" runat="Server"/>

```

BulletedList 控件除了基本属性之外，还有以下几个重要的属性和事件。

- DataSource: 获取或设置对象，数据绑定控件从该对象中检索其数据项列表。
- DataTextFieldID: 获取或设置为列表项提供文本内容的数据源字段。
- DataValueField: 获取或设置为各列表项提供值的数据源字段。
- Items: 返回 BulletedList 控件中的 ListItem 的参数，每一个项的类型都是 ListItem。在“属性”窗口中单击该属性的按钮，可以打开“ListItem 集合编辑器”对话框来设置列表项。

- **BulletStyle**: 获取或设置 BulletedList 控件的项目符号样式。
- **DisplayMode**: 获取或设置 BulletedList 控件中的列表内容的显示模式。该属性有三个取值: Text 显示为文本; HypeLink 显示为超链接; LinkButton 显示为链接按钮。
- **Click 事件**: 当单击 BulletedList 控件中的链接按钮时发生。
- **SelectedIndexChanged 事件**: 当列表控件的选定项在信息发往服务器之间变化时发生。

BulletedList 控件中可用的项目符号样式有以下数种。

- **NotSet**: 未设置符号样式。
- **Numbered**: 数字符号样式。
- **LowerAlpha**: 小写字母符号样式。
- **UpperAlpha**: 大写字母符号样式。
- **LowerRoman**: 小写罗马数字符号样式。
- **UpperRoman**: 大写罗马数字符号样式。
- **DisC**: 实心圆符号样式。
- **Circle**: 圆圈符号样式。
- **Square**: 实心正方形符号样式。
- **CustomImage**: 自定义图象符号样式。

**【例 4-9】** 本例使用 BulletedList 项目列表控件显示选项节目的列表。用户可以从列表中选中自己最喜欢的选秀节目，程序将用户的选择显示在列表下。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-9”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 BulletedList 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. 请选择你最喜欢的选秀节目
2. <asp:BulletedList runat="server" ID="BulletedList1" BulletStyle="Numbered" DisplayMode="LinkButton"
 onclick="BulletedList1_Click">
3. <asp:ListItem>中国达人秀</asp:ListItem>
4. <asp:ListItem>超级女声</asp:ListItem>
5. <asp:ListItem>我型我秀</asp:ListItem>
6. <asp:ListItem>舞林争霸</asp:ListItem>
7. </asp:BulletedList>
8. <asp:Label ID="Label1" runat="server" Text="Label" Visible="false"></asp:Label>

```

代码说明：第 2 行添加一个项目列表控件 BulletedList1，设置类别项目符号样式为数字，设置项目列表项格式为链接按钮，设置处理 BulletedList1 控件的单击事件为 Click。第 3~6 行添加 4 个列表项和显示的文本。第 8 行添加一个服务器标签控件 Label1，并设置 Visible 属性为不可见。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void BulletedList1_Click(object sender, BulletedListEventArgs e){
2. Label1.Visible = true;
3. switch (e.Index) {

```

```

4. case 0: Label1.Text = "你最喜欢的选秀节目是中国达人秀!";
5. break;
6. case 1: Label1.Text = "你最喜欢的选秀节目是超级女声!";
7. break;
8. case 2: Label1.Text = "你最喜欢的选秀节目是我型我秀!";
9. break;
10. case 3: Label1.Text = "你最喜欢的选秀节目是舞林争霸!";
11. break;
12. default: Label1.Text = "你的选择错误!";
13. break;
14. }
15. }

```

代码说明：第 1 行定义处理项目列表控件 BulletedList1 的单击事件 Click 的方法。第 2 行将标签控件的 Visible 属性设置为可见。第 3~14 行使用 switch-case 语句判断事件源对象 e 的 index 属性（即用户在项目列表中实际选定项的索引值），根据不同的索引值 0~3，分别在 Label1 上显示用户选择的列表项内容。比如选择的列表项索引是 0，就将列表项内容“中国达人秀”显示在页面的标签控件上，其余类同。

**04** 按下“Ctrl+F5”，运行结果如图 4-19 所示。用户选择项目列表中的某个项，在下面显示项的内容。

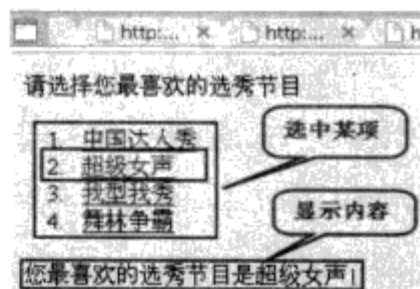


图 4-19 运行结果

## 4.7 容器服务器控件

在服务器控件中提供了一些容器类型的控件，所谓容器类控件是指可以容纳其他控件放置的一类控件，它们的出现方便了开发人员进行页面设计。可以作为容器控件的有 Panel 控件、MultiView 控件、View 控件和 PlaceHold 控件。

### 4.7.1 面板（Panel）控件

Panel 控件是一种用来对其他控件进行分组的容器控件，这样可以使得用户界面更加清晰、友好，同时也方便在运行中将多个控件作为一个单元来处理。因此在编程过程中，如果用户打算控制一组控件的集体行为，比如隐藏、显示多个控件或者禁止使用一组控件时，就可以使用 Panel 控件。把一组控件添加到同一个 Panel 控件中就能实现这一目的。Panel 控件声明方法有两种，代码如下：

```

1. <asp: Panel ID= "Panel1" Height="1000px Weight="1000px" runat="Server" ></asp: Panel>
2. <asp: Panel ID= "Panel" Height="1000px Weight="1000px" runat="Server"/>

```

Panel 控件除了基本属性之外，还有以下几个重要的属性和事件。

- BackImageUrl: 获取或设置面板控件背景图像的 URL。
- DefaultButton: 规定 Panel 中默认按钮的 ID。
- Direction: 规定 Panel 的内容显示方向。
- GroupingText: 规定 Panel 中控件组的标题。

- ScrollBars: 规定 Panel 中滚动栏的位置和可见性。
- BackColor: 获取或设置 Web 服务器控件的背景色。
- HorizontalAlign: 获取或设置面板内容的水平对齐方式。
- Wrap: 获取或设置一个指示面板中的内容是否换行的值。
- Visible: 获取或设置 Web 服务器控件的宽度。
- Attributes: 获取与控件的属性不对应的任意特性（只用于呈现）的集合。
- Controls: 获取 ControlCollection 对象，该对象表示 UI 层次结构中指定服务器控件的子控件。
- DataBinding 事件: 当服务器控件绑定到数据源时发生。

**【例 4-10】** 本例将利用 Panel 控件实现隐藏或显示控件。当用户单击新用户注册按钮时，用于输入登录信息的 Panel 控件将隐藏，用于输入注册信息的 Panel 控件将显示。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-10”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 2 个 BulletedList 控件、4 个 TextBox 控件、2 个 Button 控件和 2 个 LinkButton 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. <asp:LinkButton ID="LinkButton2" runat="server" onclick="LinkButton2_Click">会员请登录</asp:LinkButton>

2. <asp:LinkButton ID="LinkButton1" runat="server" onclick="LinkButton1_Click">新会员注册</asp:LinkButton>
3. <asp:Panel ID="Panel1" runat="server" Width="254px">
4. 登录

5. 用户名: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

6. 密 码: <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

7. <asp:Button ID="Button1" runat="server" Text="登录" />
8. </asp:Panel>
9. <asp:Panel ID="Panel2" runat="server">
10. 新用户注册

11. 用户名: <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

12. 密 码: <asp:TextBox ID="TextBox4" runat="server"></asp:TextBox>

13. <asp:Button ID="Button2" runat="server" Text="注册" />
14. </asp:Panel>

```

代码说明：第 1、2 行各自添加了一个服务器超链接按钮控件 LinkButton 并设置其单击事件。第 3~8 行添加了一个服务器面板控件 Panel1 显示登录的控件组合；其中，第 5、6 行添加了一个服务器文本框控件 TextBox；第 7 行添加一个服务器按钮控件 Button1 并设置显示的文本。第 9~14 行添加了一个服务器面板控件 Panel1，显示注册的控件组合，其中，第 11、12 行添加了一个服务器文本框控件 TextBox，第 13 行添加一个服务器按钮控件 Button2 并设置显示的文本。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. Panel1.Visible = false;
4. Panel2.Visible = false;
5. }
6. }
7. protected void LinkButton2_Click(object sender, EventArgs e){

```

```

8. Panel1.Visible = true;
9. Panel2.Visible = false;
10. }
11. protected void LinkButton1_Click(object sender, EventArgs e){
12. Panel1.Visible = false;
13. Panel2.Visible = true;
14. }

```

代码说明：第 1 行定义处理 Page 对象加载事件 Load 的方法。第 2 行判断当前页面如果不是回传页面，则第 3 和第 4 行将两个面板控件 Panel 设置为隐藏不可见。第 7 行定义处理 LinkButton2 控件的单击事件 Click 的方法。第 8 行将 Panel1 设置为显示可见。第 9 行将 Panel2 设置为隐藏不可见。第 11 行定义处理 LinkButton1 控件的单击事件 Click 方法。第 12 行将 Panel2 设置隐藏为不可见。第 13 行将 Panel2 设置显示可见。

**04** 按下“Ctrl+F5”，运行结果如图 4-20 所示。

**05** 当单击“会员请登录”按钮，出现如图 4-21 所示的登录面板，而注册面板被隐藏。

**06** 当单击“新会员注册”按钮，出现如图 4-22 所示的注册面板，而登录面板被隐藏。



图 4-20 运行结果 1

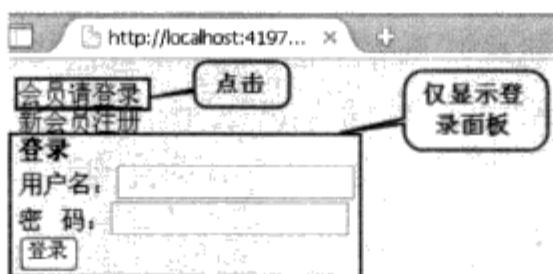


图 4-21 运行结果 2



图 4-22 运行结果 3

## 4.7.2 多视图（MultiView）控件

MultiView 控件用于定义 View（视图）控件组，使用它可以定义一组 View 控件。View 控件只有添加到 MultiView 控件中才能使用，其中每个 View 控件都包含其他控件，如标签、下拉列表等。

MultiView 控件可以作为一个或多个 View 控件的外部容器，View 控件则是标记和控件的容器。MultiView 控件一次可以显示一个 View 控件，并公开该 View 控件内的标记和控件，从而实现多视图窗口。MultiView 控件有点像在 C/S 开发中很常见的 TabControl 控件，可以在一个页面中，放置多个 View。这样就可以让用户在同一页面中，通过切换，从而看到要看的内容，而不用每次都重新打开一个新的窗口。MultiView 控件声明方法有两种，代码如下：

```

1. <asp:MultiView ID="MultiView1" ActiveViewIndex="0" runat="Server" ></asp:MultiView>
2. <asp:MultiView ID="MultiView" ActiveViewIndex="0" runat="Server"/>

```

MultiView 控件除了基本属性之外，还有以下几个重要的属性和方法。

- **ActiveViewIndex**: 获取或设置活动 View 控件的索引。MultiView 控件按 View 控件页面上出现的顺序进行从 0 到 n-1 的编号，n 表当前 MultiView 控件中的 View 控件数量。如果显示添加到 MultiView 控件中的第 1 个 View 控件，该属性设置为“0”。
- **EnableTheming**: 获取或设置一个值，该值指示是否向 MultiView 控件应用主题。
- **Views**: 获取 MultiView 控件的 View 控件集合。
- **Visible**: 用于设置 MultiView 控件在默认状态下是否可见。

- AddParsedSubObject: 通知 MultiView 控件已分析了一个 XML 或 HTML 元素, 并将该元素添加到 MultiView 控件的 ViewCollection 集合中。
- CreatedControlCollection: 创建 ControlCollection 以保存 MultiView 控件的子控件。
- GetActiveView: 返回 MultiView 控件的当前活动的 View 控件。
- LoadControlState: 加载 MultiView 控件的当前状态。
- SetActiveView: 将指定的 View 控件设置为 MultiView 控件的活动视图。
- SaveControlState: 保存 MultiView 控件的当前状态。
- RemovedControl: 在将 View 控件从 MultiView 控件的 Controls 集合中移除后调用。
- OnBubbleEvent: 确定 MultiView 控件的事件是否传递给页的用户界面服务器控件层次结构。
- OnActiveViewChanged 事件: 引发 MultiView 控件的 ActiveViewChanged 事件。

【例 4-11】本例使用 MultiView 和 View 控件实现在网络考试系统中用户可以选择不同的考试题型的切换。用户可以使用 RadioButtonList 来选择两种题型之间的转换。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 4-11”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 1 个 BulletedList 控件和 1 个 Label 控件。然后切换到“源视图”, 在编辑区中<form></form>标记之间编写如下代码。

```

1. 请选择考试的题型

2. <asp:RadioButtonList ID="RadioButtonList1" runat="server" RepeatDirection="Horizontal" onselectedindexchanged=
 "RadioButtonList1_SelectedIndexChanged" AutoPostBack="True">
3. <asp:ListItem Value="0">单选题</asp:ListItem>
4. <asp:ListItem Value="1">多选题</asp:ListItem>
5. </asp:RadioButtonList>

6. <asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex = "0">
7. <asp:View ID="View1" runat="server">单选题:

8. 下面那一本书不属于中国古代四大名著?

9. <asp:RadioButtonList ID="RadioButtonList2" runat="server" AutoPostBack="True" >
10. <asp:ListItem Value="0">A 三国演义</asp:ListItem>
11. <asp:ListItem Value="1">B 西游记</asp:ListItem>
12. <asp:ListItem Value="2">C 红楼梦</asp:ListItem>
13. <asp:ListItem Value="3">D 隋唐演义</asp:ListItem>
14. </asp:RadioButtonList>
15. </asp:View>
16. <asp:View ID="View2" runat="server">多选题:

17. 面向编程对象设计的三大特点是:

18. <asp:CheckBoxList ID="CheckBoxList1" runat="server" >
19. <asp:ListItem Value="0">封装</asp:ListItem>
20. <asp:ListItem Value="1">继承</asp:ListItem>
21. <asp:ListItem Value="2">多态</asp:ListItem>
22. <asp:ListItem Value="3">重载</asp:ListItem>
23. </asp:CheckBoxList>
24. </asp:View>
25. </asp:MultiView>

```

代码说明: 第 2~5 行添加一个服务器单选按钮列表控件 RadioButtonList1, 使用 RepeatDirection 属性设置布局方向为垂直方式, 同时设置单选按钮的选项改变事件为 SelectedIndexChanged。其中, 第 3、4 行为单选控件添加了单选题和多选题两个选项; 第 6 行添加了一个服务器多视图控件 MultiView1, 并通过 ActiveViewIndex 属性设置默认显示为添加到该控件中的第一个 View 视图。第 7~15 行添加了一个服务器视图控件 View1 显示了单选题界面, 在其中又添加了一个单选按钮列

表控件 RadioButtonList2 并添加了 4 个选择项。第 16~24 行添加了一个服务器视图控件 View1 显示多选图界面。在其中又添加了一个服务器复选框列表控件 CheckBoxList1 并添加了 4 个选择项。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e){
2. MultiView1.ActiveViewIndex = RadioButtonList1.SelectedIndex;
3. }
```

代码说明：第 1 行定义处理单选按钮列表控件 RadioButtonList1 的选中项索引改变事件 SelectedIndexChanged 的方法。第 2 行通过控件的 SelectedIndex 属性，将 RadioButtonList1 的选中项索引作为多视图控件 MultiView1 当前显示的 ActiveViewIndex 索引值。

**04** 按下“Ctrl+F5”，运行结果如图 4-23 所示。

**05** 用户选择多选题，显示如图 4-24 所示的 View2 内容，View1 不予显示；如果用户选择单选题，则显示 View2 的内容，View1 不予显示。

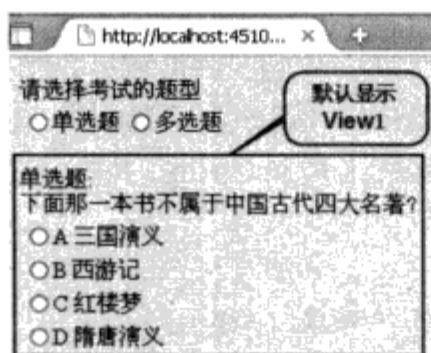


图 4-23 运行结果 1

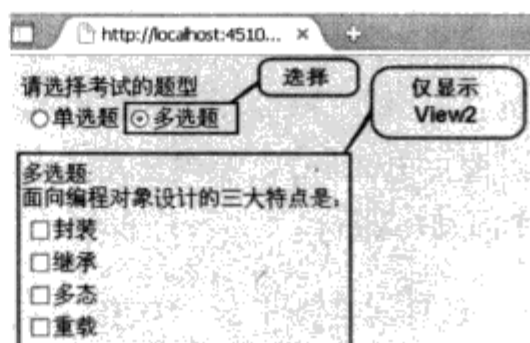


图 4-24 运行结果 2

### 4.7.3 动态容器（Placeholder）控件

Placeholder 控件一般用于在页面中动态加载其他控件，该控件没有任何基于 HTML 的输出，并且仅用于在页面执行期间向该控件的 Controls 集合中添加其他控件。Placeholder 控件声明方法有两种，代码如下：

```
1. <asp: Placeholder ID= "Placeholder1" runat="Server"></asp: Placeholder>
2. <asp: Placeholder ID= "Placeholder1" runat="Server"/>
```

MultiView 控件除了基本属性之外，还有以下几个重要的属性和方法。

- Controls: 获取 ControlCollection 对象，该对象表示 UI 层次结构中指定服务器控件的子控件。
- Visible: 获取或设置一个值，该值指示服务器控件是否作为 UI 呈现在页上。
- ViewState: 获取状态信息的字典，这些信息使你可以在同一页的多个请求间保存和还原服务器控件的视图状态。
- Site: 获取容器信息，该容器在呈现于设计图面上时承载当前控件。
- DesignMode: 获取一个值，该值指示是否正在使用设计图面上的一个控件。
- ClientID: 获取由 ASP.NET 生成的服务器控件标识符。
- AddedControl: 在子控件添加到 Control 对象的 Controls 集合后调用。

- **CreateChildControls**: 由 ASP.NET 页面框架调用, 以通知使用基于合成实现的服务器控件创建它们包含的任何子控件, 以便为回发或呈现做准备。
- **HasControls**: 确定服务器控件是否包含任何子控件。

**【例 4-12】** 本例使用 **PlaceHolder** 控件实现向页面中动态添加控件。用户在文本框中输入添加控件的个数, 单击按钮后, 可以在 **PlaceHolder** 控件中添加相应数量的标签控件、文本框控件和按钮控件。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 4-12”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 1 个 **PlaceHolder** 控件和 1 个 **Button** 控件。然后切换到“源视图”, 在编辑区中 `<form></form>` 标记之间编写如下代码。

```
1. 请输入添加控件组的数量

2. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. <asp:Button ID="Button1" runat="server" Text="确定" onclick="Button1_Click" />

4. <asp:PlaceHolder ID="PlaceHolder1" runat="server"></asp:PlaceHolder>
```

代码说明: 第 2 行添加了一个服务器文本框控件 **TextBox1** 接受用户输入的数据。第 3 行添加了一个服务器按钮控件 **Button1**, 并设置文本及处理控件的单击事件 **Click**。第 4 行添加了一个服务器动态容器控件 **PlaceHolder1**。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件, 编写代码如下。

```
1. protected void Button1_Click(object sender, EventArgs e){
2. try{
3. int num = int.Parse(TextBox1.Text);
4. for (int i = 1; i <= num; i++){
5. Label l = new Label();
6. l.ID = "Label" + i.ToString();
7. l.Text = "Label" + i.ToString();
8. TextBox t = new TextBox();
9. t.ID = "TextBox" + i.ToString();
10. t.Text = "TextBox" + i.ToString();
11. Button b = new Button();
12. b.ID = "Button" + i.ToString();
13. b.Text = "Button" + i.ToString();
14. PlaceHolder1.Controls.Add(l);
15. PlaceHolder1.Controls.Add(t);
16. PlaceHolder1.Controls.Add(b);
17. PlaceHolder1.Controls.Add(new LiteralControl("
"));
18. }
19. }
20. catch (Exception e1) {
21. PlaceHolder1.Controls.Add(new LiteralControl(e1.Message));
22. }
23. }
```

代码说明: 第 1 行定义处理 **Button1** 按钮控件的单击事件 **Click** 的方法。第 3 行获得用户输入的数值。第 4~18 行使用一个 **for** 循环向页面动态添加控件。其中, 第 5~7 行实例化一个标签控件并设置其 **ID** 和 **Text** 属性。第 8~10 行实例化一个文本框控件并设置其 **ID** 和 **Text** 属性。第 11~13 行实例化一个按钮控件并设置其 **ID** 和 **Text** 属性。第 14~16 行, 使用动态容器控件 **PlaceHolder1** 的 **Controls.Add** 方法将上面创建的三个控件添加到 **PlaceHolder1** 中显示。第 17 行在控件最后添加一

个换行符。在这段代码中使用了 try-catch 的异常处理语句块捕捉异常。

**04** 按下“Ctrl+F5”，运行结果如图 4-25 所示。用户在文本框输入添加控件组的数量，单击“确定”按钮。

**05** 在页面下面出现如图 4-26 所示的已动态添加的三组由标签控件、文本框控件和按钮控件组成的控件组。

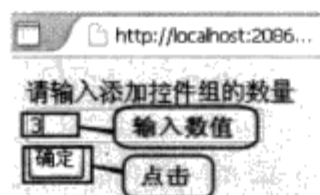


图 4-25 运行结果 1



图 4-26 运行结果 2

## 4.8 高级服务器控件

ASP.NET 4.0 提供除了前面所介绍的各种服务器控件以外，还提供了一些比较复杂的高级控件。使用这些控件可以创建丰富的页面效果，并创建强大的用户交互功能。本节将主要介绍 Calendar 控件和 AdRotator 控件。

### 4.8.1 日历（Calendar）控件

Calendar 控件可以在 Web 窗体中显示日历，以便于用户选择年、月或者日期。Calendar 控件为用户选择日期提供了丰富的可视界面，通过该控件用户可以选择日期并移动到上一个或下一个月。Calendar 控件必须放在 Form 或 Panel 控件内，或者是控件的模板内。在添加了一个 Calendar 控件之后，它一次显示一个月的日期。另外，它还显示该月之前的一周和之后的一周。所以说，一共显示六周。Calendar 控件声明方法有两种，代码如下：

1. `<asp: Calendar ID= "Calendar1" runat="Server" ></asp: Calendar>`
2. `<asp: Calendar ID= "Calendar1" runat="Server"/>`

由于 Calendar 控件比较复杂，所以除基本属性之外，它还有以下的常用属性。

- DayNameFormat: 获取或设置周中各天的名称格式，其值是一个名为 DayNameFormat 的枚举类型，该枚举类型的值包括 FirstLetter、FirstTowLetter、Short、Full、Shortest。
- FirstDayOfWeek: 获取或设置要在 Calendar 控件的第一天列中显示的一周中的某天。可以设置为 Default、Sunday、Monday、Tuesday、Wednesday、Thursday、Friday 和 Saturday。
- NextMonthText: 获取或设置为下一月导航控件显示的文本。ShowNextPreMonth 属性必须设置为 true，并且 NextPreMonth 属性设置为 CustomText 时才有效。
- NextPrevFormat: 获取或设置 Calendar 控件的标题部分中下个月和上个月导航元素的格式。可以设置为 ShortMonth、FullMonth 及默认值 CustomText。
- SelectedDate: 获取或设置选定的日期。默认为程序执行的日期。

- SelectedDates: 获取 System.DateTime 对象的集合, 这些对象表示 Calendar 控件上的选定日期。
- SelectionMode: 获取或设置 Calendar 控件上的日期选择模式。可设置为 Day, 用户只可以选中某一天为默认值; None, 不能选取日期, 只能显示日期; DayWeek, 用户可以一次选取整个星期或者某一天; DayWeekMonth, 用户可以一次选取这个月、整个星期或者某一天。
- SelectionMonthText: 获取或设置为选择器列中月份选择元素显示的文本。要将 SelectionMode 属性设置为 DayWeekMonth 才有效。
- SelectWeekText: 获取或设置为选择器列中周选择元素显示的文本。要将 SelectionMode 属性设置为 DayWeekMonth 才有效。
- ShowDayHeader: 获取或设置一个值, 该值指示是否显示一周中各天的标头。有 true 和 false 两个选项。
- ShowGridLines: 获取或设置一个值, 该值指示是否用网格线分割 Calendar 控件上的日期。
- ShowNextPrevMonth: 获取或设置一个值, 该值指示 Calendar 控件是否在标题部分显示下个月和上个月导航元素。
- ShowTitle: 获取或设置一个值, 该值指示是否显示标题部分。
- TitleFormat: 获取或设置标题部分的格式。可以设置默认值 MonthYear 或 Month。
- TodaysDate: 获取或设置今天的日期的值。
- VisibleDate: 获取或设置指定要在 Calendar 控件上显示的月份的日期。

Calendar 控件除了各种属性之外, 它还有以下的常用方法和事件。

- AddDays: 返回与指定的 DateTime 相距指定天数的 DateTime。
- AddMonths: 返回与指定 DateTime 相距指定月数的 DateTime。
- AddWeeks: 返回与指定 DateTime 相距指定周数的 DateTime。
- AddYears: 返回与指定 DateTime 相距指定年数的 DateTime。
- GetDayOfMonth: 返回指定 DateTime 中的日期是该月的几号。
- GetDayOfWeek: 返回指定 DateTime 中的日期是星期几。
- GetDayOfYear: 返回指定 DateTime 中的日期是该年中的第几天。
- GetDaysInMonth: 返回指定月份中的天数。
- GetDaysInYear: 返回指定年份中的天数。
- GetLeapMonth: 计算指定年份或指定纪元年份的闰月。
- GetMonth: 返回指定的 DateTime 中的月份。
- GetMonthsInYear: 返回指定年份中的月数。
- GetWeekOfYear: 返回年中包括指定 DateTime 中日期的星期。
- GetYear: 将返回指定的 DateTime 中的年份。
- IsLeapMonth: 确定某月是否为闰月。
- IsLeapYear: 确定某年是否为闰年。
- ToDateTime: 返回设置为指定日期和时间的 DateTime。
- SelectionChanged 事件: 当用户选取日期时, 会驱动 SelectionChanged 指定的事件。
- DayRender 事件: Calendar 控件每产生一个日期都会触发该事件。

- VisibleMonthChanged 事件：当用户单击日历控件标题上的“上个月”或“下个月”按钮时触发。

**【例 4-13】** 在本章前面的例 4-8 中通过 DropDownList 控件实现酒店预订功能时，用户选择到店和离店的日期操作。但是用 6 个列表控件让用户一个个选择会让用户的感受体验不是最好，所以这里使用两个 Calendar 控件实现相同的功能，这也是目前各大旅游网站如携程网、艺龙网主页上采用的人性化交互的方法。进入页面时将日历隐藏，当单击按钮时显示日历，选择日期完成后，再将日历隐藏，同时将用户选择的日期呈现在按钮中。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-13”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 2 个 Calendar 控件、2 个 Button 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 酒店预订日期选择：

2. 入住日期
<asp:Button ID="Button1" runat="server" BorderStyle="Inset" BorderWidth="1px" Height="21px"
 Width="102px" onclick="Button1_Click" />

3. <asp:Calendar ID="Calendar1" runat="server" Visible="false" onselectionchanged="Calendar1_SelectionChanged">
 </asp:Calendar>

4. 离店日期

5. <asp:Button ID="Button2" runat="server" BorderStyle="Inset" BorderWidth="1px" Height="21px" Width="102px"
 onclick="Button2_Click" />

6. <asp:Calendar ID="Calendar2" runat="server" Visible="false"
 onselectionchanged="Calendar2_SelectionChanged"></asp:Calendar>
```

代码说明：第 2 行添加一个服务器按钮控件 Button1，设置控件的宽度、高度、边框样式以及处理控件单击事件 Click。第 3 行添加一个服务器日历控件 Calendar1，并设置为不可见。同时设置控件的选中项改变事件 SelectionChanged。第 5 行添加一个服务器按钮控件 Button2，设置控件的宽度、高度、边框样式以及处理控件单击事件 Click。第 6 行添加一个服务器日历控件 Calendar2 并设置为不可见。同时设置控件的选中项改变事件 SelectionChanged。

**03** 此时在“设计视图”中显示如图 4-27 所示的 Calendar1 和 Calendar2 是系统默认的，其外观样式比较单调。我们可以让日历的外观变得更美观一些。在控件右上方有一个向右的黑色小三角，单击这个小按钮打开“Calendar 任务”列表，选择“自动套用格式”。

**04** 弹出如图 4-28 所示的自动套用格式对话框。在左边的选择架构列表中有 6 种外观格式供我们使用，只要选中某一格式，在右边的预览窗口中会看到该格式的效果。最后，单击“确定”按钮，即可在页面中使用这一外观格式。



图 4-27 “Calendar 任务”列表

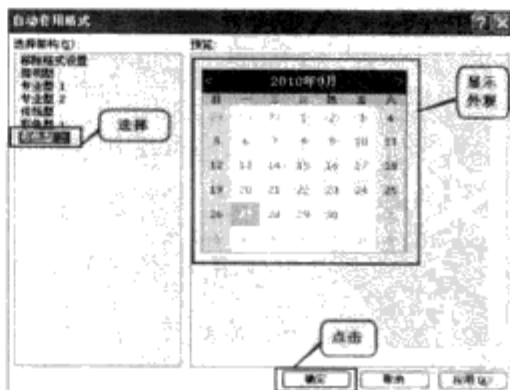


图 4-28 自动套用格式对话框

**05** 此时，在“设计视图”中看到的 Calendar1 和 Calendar2 控件的外观与图 4-27 中默认的外观已经大不相同了。切换到“源视图”，在代码编辑区中发现在原来的 Calendar1 和 Calendar2 控件的声明中自动添加了如下一些代码。

```

1. <asp:Calendar ID="Calendar1" runat="server" Visible="false"
 onselectionchanged="Calendar1_SelectionChanged"
2. BackColor="White" BorderColor="#3366CC" BorderWidth="1px" CellPadding="1"
3. DayNameFormat="Shortest" Font-Names="Verdana" Font-Size="8pt"
4. ForeColor="#003399" Height="200px" Width="220px">
5. <DayHeaderStyle BackColor="#99CCCC" ForeColor="#336666" Height="1px" />
6. <NextPrevStyle Font-Size="8pt" ForeColor="#CCCCFF" />
7. <OtherMonthDayStyle ForeColor="#999999" />
8. <SelectedDayStyle BackColor="#009999" Font-Bold="True" ForeColor="#CCFF99" />
9. <SelectorStyle BackColor="#99CCCC" ForeColor="#336666" />
10. <TitleStyle BackColor="#003399" BorderColor="#3366CC" BorderWidth="1px" Font-Bold="True" Font-Size="10pt"
 ForeColor="#CCCCFF" Height="25px" />
11. <TodayDayStyle BackColor="#99CCCC" ForeColor="White" />
12. <WeekendDayStyle BackColor="#CCCCFF" />
13. </asp:Calendar>

```

代码说明：第 1~14 行声明了一个使用自动套用格式中“彩色型 2”格式的 Calendar 控件。其中，第 2 行设置控件的背景色、边框颜色、边框宽度和文字到边框之间的距离。第 3 行设置各天的名称的格式、文字格式和大小。第 4 行设置控件的前景色和大小尺寸。第 5 行到设置日历上方显示日名称的背景色、前景色和高度。第 6 行设置标题栏左端和右端放置月份的颜色和文字大小。第 7 行设置上个月和下个月中的天的颜色。第 8 行设置用户选中日期的背景色、字体和前景色。第 9 行设置位于左侧列的背景色和前景色。第 10 行设置控件顶端标题栏的背景色、前景色、边框颜色宽度、字体和高度。第 11 行设置当前日期的背景色、前景色。第 12 行设置周末日期的颜色。

**06** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写关键代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. Calendar1.Visible = true;
3. }
4. protected void Button2_Click(object sender, EventArgs e){
5. Calendar2.Visible = true;
6. }
7. protected void Calendar1_SelectionChanged(object sender, EventArgs e){
8. Button1.Text = Calendar1.SelectedDate.Year.ToString() + "-" + Calendar1.SelectedDate.Month.ToString() + "-" +
Calendar1.SelectedDate.Day.ToString();
9. Calendar1.Visible = false;
10. }
11. protected void Calendar2_SelectionChanged(object sender, EventArgs e) {
12. Button2.Text = Calendar2.SelectedDate.Year.ToString() + "-" + Calendar2.SelectedDate.Month.ToString() + "-" +
Calendar2.SelectedDate.Day.ToString();
13. Calendar2.Visible = false;
14. }

```

代码说明：第 1 行定义处理按钮控件 Button1 单击事件的方法。第 2 行将第一个日历控件 Calendar1 的可见性设置为显示。第 4 行定义处理按钮控件 Button2 单击事件的方法。第 5 行将第二个日历控件 Calendar2 的可见性设置为显示。第 7 行定义处理 Calendar1 选中项改变事件 SelectionChanged 的方法。第 8 行分别使用 Calendar1 控件的 SelectedDate.Year、SelectedDate.Month 和 SelectedDate.Day 获得用户选中的到店日期的年月日，并将它们以字符串拼接的形式显示在按钮上。第 9 行将 Calendar1 控件设置为不可见。第 11 行定义处理 Calendar2 选中项改变事件

SelectionChanged 的方法。第 12 行分别使用 Calendar2 控件的 SelectedDate.Year、SelectedDate.Month 和 SelectedDate.Day 获得用户选中离店日期的年月日,并将它们以字符串拼接的形式显示在按钮上。第 13 行将 Calendar2 控件设置为不可见。

- 07 按下 “Ctrl+F5”, 运行结果如图 4-29 所示。
- 08 分别单击两个按钮, 出现如图 4-30 所示的隐藏日历, 并选择预定的入住和离店日期。
- 09 此时出现如图 4-31 所示的页面。日历控件消失隐藏, 用户选择的日期显示在按钮上。

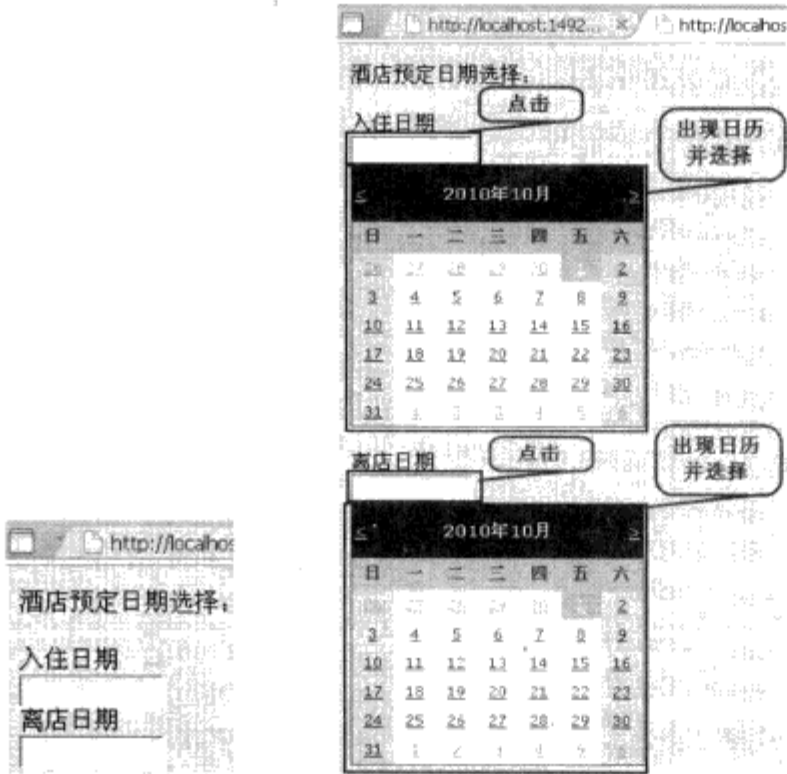


图 4-29 运行结果 1



图 4-30 运行结果 2

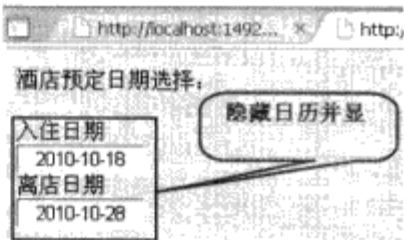


图 4-31 运行结果 3

4.8.2 动态广告 (AdRotator) 控件

AdRotator 服务器控件用于在 Web 窗体中显示公布的标志。该控件经常被用来显示一些广告的内容。AdRotator 控件使用一组在一个特定 XML 文件中定义好的信息以轮流的方式播出广告, 每条广告的各种信息都在 XML 文件中加以指定。AdRotator 控件声明方法有两种, 代码如下:

```
1. <asp: AdRotator ID= "AdRotator1" runat="Server" ></asp: AdRotator>
2. <asp: AdRotator ID= "AdRotator1" runat="Server"/>
```

AdRotator 控件除了基本属性之外, 还有以下一些重要的属性和方法。

- AdvertisementFile: 获取或设置包含广告信息的 XML 文件的路径。
- AlternateTextFieldID: 获取或设置一个自定义数据字段, 使用它代替广告的 AlternateText 属性。
- Font: 获取与广告横幅控件关联的字体属性。
- ImageUrlFieldID: 获取或设置一个自定义数据字段, 使用它代替广告的 ImageUrl 属性。
- KeywordFilter: 获取或设置类别关键字以筛选出 XML 公布文件中特定类型的公布。
- NavigateUrlFieldID: 获取或设置一个自定义数据字段, 使用它代替广告的 NavigateUrl 属性。
- TagKey: 获取 AdRotator 控件的 HTML 标记, 该属性是受保护的。
- Target: 获取或设置当单击 AdRotator 控件时, 显示所链接到的页面内容的浏览器窗口或框

架的名称。

- UniqueID: 获取 AdRotator 控件在层次结构中的唯一限定标识符。
- PerformDataBinding: 将指定数据源绑定到 AdRotator 控件。
- PerformSelect: 将关联数据源检索广告数据。
- Render: 在客户端上显示 AdRotator 控件。
- OnAdCreated 事件: 每次要产生新的广告内容便触发该事件。
- OnPreRender 事件: 通过查找文件数据或调用用户事件获取要呈现的广告信息。

AdRotator 控件可以从 XML 文件中读取广告信息,也可以从数据库中读取广告信息。它需要通过自己的属性来定义一个广告体所需要信息,但这些信息都是可选的,因此无论在 XML 文件中还是在数据库定义广告体,可以选用如下属性来作为广告体的信息。

- ImageUrl, 要显示的图像的 URL。
- NavigateUrl, 单击 AdRotator 控件将要转到的页面的 URL。
- AlternateText, 图像不可用时显示的文本。
- Keyword, 可用语筛选特定广告的广告类别。
- Impressions, 一个指示广告的可能显示频率的数值。
- Height, 广告的高度。
- Width, 广告的宽度。

创建 XML 文件时,开始标记<Advertisements>和结束标记</Advertisements>分别标记该文件开头和结尾。标记<Ad>和</Ad>用于划定一个广告的界限。所以的广告都嵌套在开始和结束<Advertisements>标记之间。尽管某些数据元素是预定义的,如 ImageUrl 和 NavigateUrl,但仍然可以在<Ad>标记之间放置自定义元素。AdRotator 控件在分析该文件时将读取这些元素。然后将该信息传递给 AdCreated 事件。下面一段代码是 XML 文件的示例。

```

1. <Advertisements>
2. <Ad>
3. <ImageUrl> 广告的图像路径</ImageUrl>
4. <NavigateUrl> 单击广告时跳转的网址</NavigateUrl>
5. <AlternateText>图像不能显示时的替代文字信息/提示信息</AlternateText>
6. <Keyword>过滤广告的关键字</Keyword>
7. <Impressions>广告出现频度</Impressions>
8. </Ad>
9. <Ad>
10.
11. </Ad>
12. <Ad>
13.
14. </Ad>
15. </Advertisements>

```

代码说明: 第 1 行和第 15 行是一对<Advertisements>标记,所有的内容必须都包括在其中。其中第 2~8 行通过一对<Ad>标记定义一个广告的具体信息,包括广告的图形路径、单击时跳转的网址、替代图片不显示时的文字、过滤广告关键字和广告出现的频率,其中这些属性标记都是可选的。第 9~11 行以及第 12~14 行分别定义了两个其他的广告信息,代码说明这里从略。

**【例 4-14】**广告是当今网站不可或缺的一个组成部分。本例利用 AdRotator 控件实现在页面中显示广告的功能。广告图片形式呈现，如果用户单击该图片会进入显示广告具体信息的页面。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 4-14”。
- 02** 在程序中创建一个“Image”文件夹，并放置三张网站的图片文件。
- 03** 在程序中创建一个 XML 文件 XMLFile1.xml，在文件中编写如下代码：

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <Advertisements>
3. <Ad>
4. <ImageUrl>~/Image/机票.bmp</ImageUrl>
5. <NavigateUrl>http://life.sina.com.cn/jipiao/resultsfee.htm</NavigateUrl>
6. <AlternateText>机票预订</AlternateText>
7. <Impressions>50</Impressions>
8. <Keyword>Category1</Keyword>
9. </Ad>
10. <Ad>
11. <ImageUrl>~/Image/贺卡.bmp</ImageUrl>
12. <NavigateUrl>http://diy.sina.com.cn/cardshow.php?%20from=430</NavigateUrl>
13. <AlternateText>精美贺卡</AlternateText>
14. <Impressions>50</Impressions>
15. <Keyword>Category1</Keyword>
16. </Ad>
17. </Advertisements>

```

代码说明：第 2~17 行使用开始和结束标记定义两个广告信息文件内容。其中第 3~9 行是第一个广告，定义了广告的图片路径 ImageUrl、链接的网址 NavigateUrl、代替图片显示的文字 AlternateText、显示的频率 Impressions 和筛选依据 Keyword。第 10~16 行定义了第二个广告的信息，方法与第一个广告相同。

**04** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 Calendar 控件、1 个 Button 控件和一个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. 请单击广告

2. <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile="~/XMLFile1.xml" Target="_blank"
 OnAdCreated="AdCreated_Event" />

```

代码说明：第 2 行添加了一个服务器动态广告控件 AdRotator1，设置其关联的 XML 文件和处理创建广告事件 AdCreated\_Event。

**05** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

protected void AdCreated_Event(object sender, AdCreatedEventArgs e){
}

```

代码说明：第 1~2 行定义处理动态广告控件 AdCreated1 事件的方法。比较特别的地方是这个方法中不需要写任何代码。

**06** 按下“Ctrl+F5”，运行结果如图 4-32 所示。随机显示一个广告，如果刷新页面会出现另一个广告，随机出现的频率由 XML 文件中的属性设置决定。

**07** 当单击广告，会跳转到如图 4-33 所示的显示广告具体信息的页面。



图 4-32 运行结果

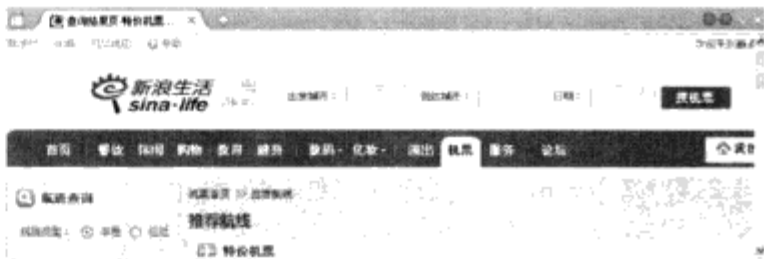


图 4-33 运行结果 2

## 4.9 上机题

1. 设计三种不同类型的 TextBox 控件，分别为单行、密码和多行文本框。如果用户在密码文本框中输入的密码少于 8 位或者大于 15 位，就弹出错误提示对话框，程序运行结果如图 4-34 所示。
2. 编写一个程序，用户可以在网页上通过 DropDownList 控件选择他的出生日期，并使用 Label 控件显示该用户的出生日期，程序运行结果如图 4-35 所示。

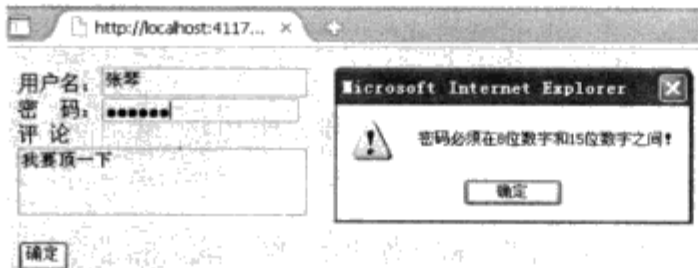


图 4-34 运行结果

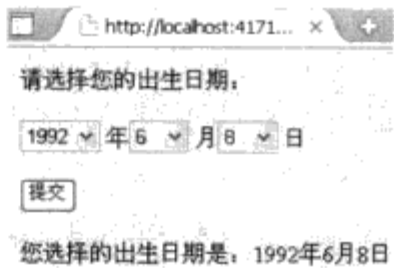


图 4-35 运行结果

3. 使用 Image 控件和 DropDownList 控件，在下拉列表中有若干列表项表示图片的名称，当选中某个图片的名称，将该图片显示在图像控件上。用户可以自由选择并切换，显示相应的不同的图片，程序运行结果如图 4-36 所示。
4. 分别使用 CheckBox 控件和 CheckBoxList 控件完成相同的功能，比较这两个控件的异同，程序运行结果如图 4-37 所示。

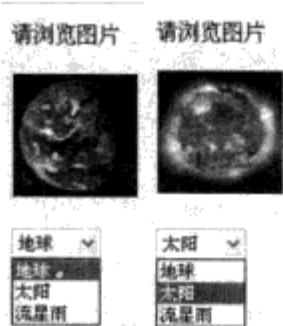


图 4-36 运行结果

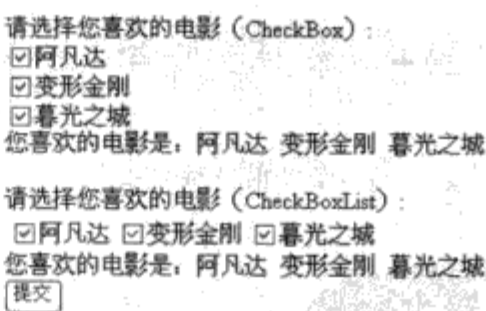


图 4-37 运行结果

5. 使用单选按钮列表控件 RadioButtonList 实现单选功能，并可以设置不同的布局 and 显示的方式，一种是水平显示，一种是纵向显示，程序运行结果如图 4-38 所示。
6. 利用 Calendar 控件实现用户选择日历控件中的日期，页面显示所选日期的具体年月日和星期。进入页面将日历隐藏，当单击选择按钮时显示日历，选择日期完成后再将日历隐藏，程序运行结果如图 4-39 所示。

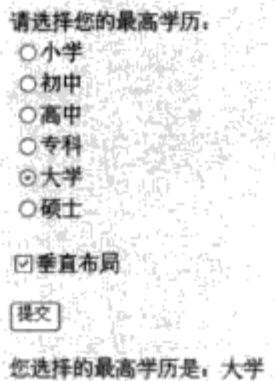


图 4-38 运行结果

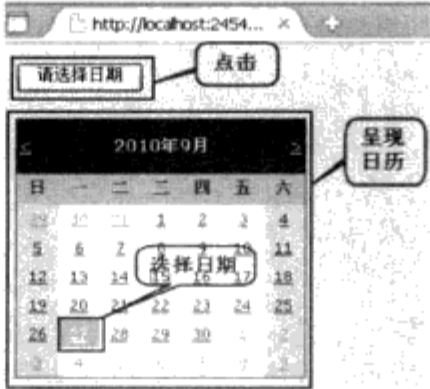


图 4-39 运行结果

7. 参考本章“例 4-7”，使用两个下拉列表框实现级联变化过程。第一个 DropDownList 显示网站类型，另一个 DropDownList 显示具体的网站，显示网站的下拉列表框会根据显示网站类型的下拉列表框中所选的值的不同而显示不同的网站。选择完毕后，单击“确定”按钮，就可以在页面显示所选的网站类型和具体网站，程序运行结果如图 4-40 所示。
8. 使用 BulletedList 控件显示网站列表，用户可以选择自己喜欢的网站，选择的结果显示在标签 Label 控件上，程序运行结果如图 4-41 所示。

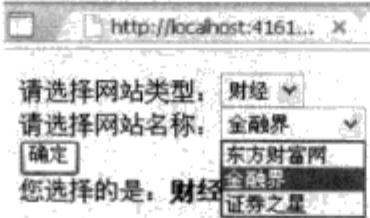


图 4-40 运行结果



图 4-41 运行结果

# 第 5 章 验证控件和用户控件

## 学习目标

ASP.NET 4.0 除了标准的内部服务器控件以外，还提供了功能强大的一组验证控件用于在服务器端对用户输入的信息进行验证，包括针对特定模式、范围或值进行验证，并且可以指定验证出现错误时显示的提示信息。这样开发人员就可以将大部分精力放在程序的业务逻辑功能上。当然，在实际编程工作中，这些控件并不能完全满足我们的业务需求，这就要求我们能够自定义一些用户控件来完成特殊的功能。所以本章在介绍了六种验证控件后，对用户控件的开发也一并做了介绍。

## 本章重点

- 掌握页面数据验证的原理
- 熟练使用六种服务器验证控件
- 在窗体中应用用户控件

## 5.1 数据验证的两种方式

当用户向服务器提交页面之后，为了保证用户输入的数据是有价值的，需要对用户提交的数据进行验证。在 Web 应用程序中，用户提交的数据经客户端浏览器发送到服务器端，我们可以选择在窗体发送到 Web 服务器之前使用 JavaScript 脚本验证输入到窗体上的数据，这称为客户端数据验证；也可以在 Web 服务器端验证用户提交的数据，这称为服务器端数据验证。本章所介绍的验证控件都是在服务器端使用的验证控件，也就是说属于服务器端数据验证的范畴。

### 5.1.1 服务器端数据验证

服务器端进行数据验证的控件在服务器代码中执行输入检查。服务器将逐个调用验证控件来检查用户输入。如果在任意输入控件中检测到验证错误，则该页面将自行设置为无效状态，以便在代码运行之前测试其有效性。验证发生的时间是：已对页面进行了初始化，即处理了视图状态和回发数据，但尚未调用任何更改或单击事件处理程序。

通过像添加其他服务器控件那样向页面添加验证控件，即可启用对用户输入的验证。有各种类型的验证控件，如范围检查或模式匹配验证控件。每个验证控件都引用页面上其他地方的输入服务器控件。处理用户输入时，验证控件会对用户输入进行测试，并设置属性以指示该输入是否通过测试。调用了所有验证控件后，会在页面上设置一个属性以指示是否出现验证检查失败。

可将验证控件关联到验证组中，使得属于同一组的验证控件可以一起进行验证。可以使用验证组有选择地启用或禁用页面上相关控件的验证。关于验证组，我们在这里不作介绍，感兴趣的读

者可以参考相关资料。

我们可以编写自己的代码来测试页和单个控件的状态。例如，可以在使用用户输入的信息更新数据记录之前来测试验证控件的状态。如果检测到状态无效，将会略过更新。通常，如果任何验证检查失败，都将跳过所有处理过程并将页面返回给用户。检测到错误的验证控件，随后将生成显示在页上的错误信息。可以使用 `ValidationSummary` 控件在一个位置显示所有验证错误。

每个验证控件通常只执行一次测试。但我们可能需要检查多个条件。例如，可能需要指定必需的用户输入，同时将该用户输入限制为只接受特定范围内的日期。此时，可以将多个验证控件附加到页面上的一个输入控件。通过使用逻辑 AND 运算符来解析控件执行的测试，这意味着用户输入的数据必须通过所有测试才能视为有效。

验证控件通常在呈现的页面中不可见。但是，如果控件检测到错误，则它将显示指定的错误信息文本。错误信息可以以各种方式显示，如下所示。

- 内联方式：每一验证控件可以单独就地（通常在发生错误的控件旁边）显示一条错误信息。
- 摘要方式：验证错误可以收集并显示在一个位置，例如页面的顶部。这一策略通常与在发生错误的输入字段旁显示消息的方法结合使用。如果用户使用 Internet Explorer 4.0 或更高版本，则摘要可以显示在消息框中。
- 就地和摘要方式：同一错误信息的摘要显示和就地显示可能会有所不同。可使用此选项就地显示简短错误信息，而在摘要中显示更为详细的信息。
- 自定义方式：通过捕获错误信息并设计自己的输出来自定义错误信息的显示。

服务器端数据验证相对而很安全，因为这种验证是基于服务器端的验证，不容易被绕过，而且也可以不考虑客户端的浏览器是否支持客户端脚本语言，一旦提交的数据无效，页面就会回送到客户端上。由于页面必须提交到一个远程位置进行检验，这使得服务器端的验证过程比较慢。

### 5.1.2 客户端数据验证

客户端数据验证通常是对客户端浏览器窗体上的数据进行验证，是通过向客户端浏览器传送的页面提供的一个脚本，通常采用 JavaScript 形式，在窗体回送到服务器之前，对数据进行验证。

客户端数据验证的突出优点是，能够快速向用户提供验证结果的反馈，当用户的信息输入有错误的时候，可以立即显示一条错误的信息，而不需要将这些数据传输到服务器，减少了服务器处理压力的负担。但是客户端的验证没有直接访问数据库的功能，无法实现用户合法性的验证。用户可以很容易地查看到页面的代码，而且有可能伪造提交的数据，如果浏览器版本过低或浏览器禁用了客户端脚本，客户端验证就无效了。对于一些黑客而言可以很方便地绕过客户端的验证，所以，仅仅依靠客户端的验证是不安全的。

默认情况下，在执行客户端验证时，如果页面上出现错误，则用户无法将页面发送到服务器。但有时需要允许用户即使在出错时也可以发送。例如，页上可能有一个取消按钮或一个导航按钮，即使在部分控件未通过验证的情况下，我们需要该按钮也能提交页面。此时，需要对 ASP.NET 4.0 服务器控件禁止验证。因本书不涉及对 JavaScript 脚本语言的介绍，所以关于客户端数据验证的内容在此仅一笔带过。



提示

即使验证控件已在客户端执行验证，ASP.NET 仍会在服务器上执行验证，这样可以在基于服务器的事件处理程序中测试有效性。此外，在服务器上重新测试有助于防止用户通过禁用或更改客户端脚本检查来逃避验证。

## 5.2 服务器验证控件

为用户输入创建 ASP.NET 网页的一个重要目的是检查用户输入的信息是否有效。ASP.NET 4.0 提供了一组服务器验证控件，用于提供一种易用且功能强大的检错方式，并在必要时向用户显示错误信息。

### 5.2.1 验证控件的分类

ASP.NET 4.0 的服务器验证控件共有五种，分别用于检查用户输入信息的不同方面，各种控件的类型和作用如表 5-1 所示。

表 5-1 验证控件分类

| 验证类型   | 使用的控件                      | 控件的作用                      |
|--------|----------------------------|----------------------------|
| 必需项    | RequiredFieldValidator     | 验证某个控件的内容是否被改变             |
| 与某值的比较 | CompareValidator           | 用于对两个值进行比较验证               |
| 范围检查   | RangeValidator 控件          | 用于验证某个值是否在要求的范围内           |
| 模式匹配   | RegularExpressionValidator | 用于验证相关输入控件的值是否匹配正则表达式指定的模式 |
| 自定义    | CustomValidator            | 调用在服务器端编写的自定义验证函数          |

对于一个输入控件，我们可以附加多个验证控件。既可以验证某个控件是必需的，也可以验证该控件必须包含特定范围的值。

除了以上验证控件之外，还有一个相关控件，即 ValidationSummary 控件。该控件不执行验证，但经常与其他验证控件一起用于显示来自页上所有验证控件的错误信息。

### 5.2.2 RequiredFieldValidator 控件

RequiredFieldValidator 控件通常用于在用户输入信息时，对必选字段进行验证。在页中添加 RequiredFieldValidator 控件并将其链接到必选字段控件（通常是 TextBox 控件）。在控件失去焦点时，如果其初始属性值没有被改变，将会触发 RequiredFieldValidator 控件。RequiredFieldValidator 控件的使用语法定义如下所示。

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server">
</asp:RequiredFieldValidator>
```

对于 RequiredFieldValidator 控件的使用一般是通过对其属性设置来完成的，该控件常用的属性如表 5-2 所示。

表 5-2 RequiredFieldValidator 控件的常用属性

| 属性                | 说明                                                                                                                                                                                                             |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ControlToValidate | 通过设置该属性为某控件的 ID 来把验证控件绑定到需要验证的控件                                                                                                                                                                               |
| ErrorMessage      | 通过该属性来设置当验证控件无效时需要显示的信息                                                                                                                                                                                        |
| ValidationGroup   | 绑定到验证程序所属的组                                                                                                                                                                                                    |
| Text              | 当验证控件无效时显示的验证程序的文本                                                                                                                                                                                             |
| Display           | 通过该属性来设置验证控件的显示模式，该属性有三个值： <ul style="list-style-type: none"><li>● None，表示验证控件无效时不显示信息；</li><li>● Static，表示验证控件在页面上占位是静态的，不能为其它空间所占；</li><li>● Dynamic，表示验证控件在页面上占位是动态的，可以为其它空间所占，当验证失效时验证控件才占据页面位置。</li></ul> |

可被验证的标准控件包括 TextBox、ListBox、DropDownList、RadioButtonList 以及一些 HTML 服务器控件。

**【例 5-1】**使用 RequiredFieldValidator 控件最常用的场合是对一个用户登录的页面，进行验证用户名和密码的输入是否为空的操作，本例通过这个操作来学习该控件的应用。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-1”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 2 个 TextBox 控件、2 个 RequiredFieldValidator 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 用户登录

2. 用户名: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
3. <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="TextBox1"
 Display="Dynamic"
 ErrorMessage="用户名不能为空" ForeColor="Red"></asp:RequiredFieldValidator>

4. 密 码: <asp:TextBox ID="TextBox2" runat="server" TextMode="Password"></asp:TextBox>
5. <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ErrorMessage="密码不能为空" ForeColor="Red"
 ControlToValidate="TextBox2" Display="Dynamic"></asp:RequiredFieldValidator>

6. <asp:Button ID="Button1" runat="server" Text="登录" onclick="Button1_Click" />

7. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

代码说明：第 2 行添加一个服务器文本框控件 TextBox1 接受用户的输入。第 3 行添加了一个服务器 RequiredFieldValidator 控件，分别设置需验证的关联控件、显示方式和错误提示的文字和颜色。第 4 行添加一个服务器文本框控件 TextBox2 接受用户输入密码，并设置文本框的模式显示密码格式。第 5 添加了一个服务器 RequiredFieldValidator 控件，分别设置需验证的关联控件 TextBox2、显示方式和错误提示的文字和颜色。第 6 行添加一个服务器按钮控件 Button1 并设置其单击事件 Click。第 7 行添加一个服务器标签控件 Label1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Button1_Click(object sender, EventArgs e){
2. if (TextBox1.Text == "admin" && TextBox2.Text == "123456"){
3. Label1.Text = "欢迎你，登录成功！";
```

```
4. }
5. else{
6. Label1.Text = "你输入的用户名或密码错误，请重新输入！";
7. }
8. }
```

代码说明：第 1 行定义处理按钮控件 Button 单击事件 Click 的方法。第 2 行判断如果用户输入的用户名是“admin”，同时密码输入是“123456”，则第 3 行在 Label1 控件上显示登录成功的提示文字。否则第 6 行在 Label1 控件上显示登录失败的提示文字。

**04** 按下“Ctrl+F5”，运行结果如图 5-1 所示。如果用户没有输入任何内容就单击“登录”按钮，出现 RequiredFieldValidator 控件的红色验证错误提示的文字。

**05** 用户在两个文本框中输入正确的用户名和密码，单击“登录”按钮，显示如图 5-2 所示的登录成功提示。

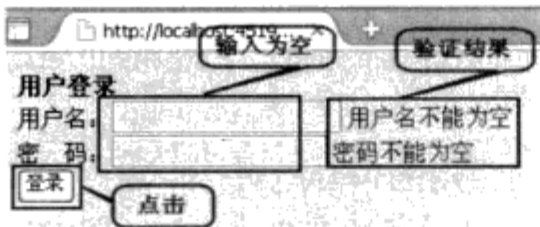


图 5-1 运行结果 1

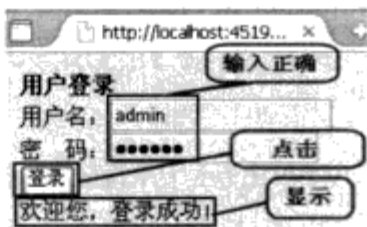


图 5-2 运行结果 2



在使用 RequiredFieldValidator 控件执行验证之前需清除输入控件中输入值前后的多余空格，这样可防止在输入控件中输入的空格通过验证。

### 5.2.3 CompareValidator 控件

CompareValidator 控件用于将用户输入的值和其他控件的值或者常数进行比较，以确定这两个值是否与由比较运算符（小于、等于、大于等等）指定的关系相匹配。还可以使用 CompareValidator 控件来指示输入到输入控件中的值是否可以转换为 BaseCompareValidator.Type 属性所指定的数据类型。CompareValidator 控件的使用语法定义如下所示。

```
<asp:CompareValidator ID=" CompareValidator1" runat="server" >
</asp:CompareValidator>
```

对于 CompareValidator 控件的使用一般也是通过对其属性的设置来完成的，该控件常用的属性如表 5-3 所示。

表 5-3 CompareValidator 控件的常用属性

| 属性                | 说明                                |
|-------------------|-----------------------------------|
| ControlToValidate | 通过设置该属性为某控件的 ID 来把验证控件绑定到需要验证的控件上 |
| ErrorMessage      | 通过该属性来设置当验证控件无效时需要显示的信息           |
| ValidationGroup   | 绑定到验证程序所属的组                       |
| Text              | 当验证控件无效时显示的验证程序的文本                |

(续表)

| 属性               | 说明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Display          | 通过该属性来设置验证控件的显示模式，该属性有三个值： <ul style="list-style-type: none"><li>● None，表示验证控件无效时不显示信息；</li><li>● Static，表示验证控件在页面上占位是静态的，不能为其它空间所占；</li><li>● Dynamic，表示验证控件在页面上占位是动态的，可以为其它空间所占，当验证失效时验证控件才占据页面位置。</li></ul>                                                                                                                                                                                                                                                                      |
| Operator         | 通过该属性来设置比较时所用到的运算符，运算符有以下几种： <ul style="list-style-type: none"><li>● Equal，所验证的输入控件的值与其他控件的值或常数值之间的相等比较；</li><li>● NotEqual，所验证的输入控件的值与其他控件的值或常数值之间的不相等比较；</li><li>● GreaterThan，所验证的输入控件的值与其他控件的值或常数值之间的大于比较；</li><li>● GreaterThanEqual，所验证的输入控件的值与其他控件的值或常数值之间的大于或等于比较；</li><li>● LessThan，所验证的输入控件的值与其他控件的值或常数值之间的小于比较；</li><li>● LessThanEqual，所验证的输入控件的值与其他控件的值或常数值之间的小于或等于比较；</li><li>● DataTypeCheck，输入到所验证的输入控件的值与 BaseCompareValidator.Type 属性指定的数据类型之间的数据类型比较。</li></ul> |
| Type             | 通过该属性来设置按照哪种数据类型来进行比较，常用的数据类型包括： <ul style="list-style-type: none"><li>● String，字符串数据类型；</li><li>● Integer，32 位有符号整数数据类型；</li><li>● Double，双精度浮点数数据类型；</li><li>● Date，日期数据类型；</li><li>● Currency，一种可以包含货币符号的十进制数据类型。</li></ul>                                                                                                                                                                                                                                                      |
| ValueToCompare   | 设置用来做比较的数据                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| ControlToCompare | 设置用来做比较的控件，有时需要让验证控件控制的控件和其它控件里的数据做比较就会用到这个属性                                                                                                                                                                                                                                                                                                                                                                                                                                       |

【例 5-2】在网站注册中，有一个最常用的比较，就是对用户输入密码的二次验证，这就需要用到上面的 CompareValidator 控件，本例将实现这一常用功能。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-2”。

02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 2 个 TextBox 控件、1 个 CompareValidator 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 验证二次密码输入是否一致

2. 密码: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. 确认密码: <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
4. <asp:CompareValidator ID="CompareValidator1" runat="server" ControlToCompare="TextBox1"
 ControlToValidate="TextBox2" Display="Dynamic" ErrorMessage="二次密码输入不一致"
```

```

ForeColor="Red"></asp:CompareValidator>

5. <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="提交" />

6. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>

```

代码说明：第 1 和第 2 行各种添加一个服务器文本框控件 TextBox 接受用户输入的二次密码。第 3 行添加了一个服务器验证控件 CompareValidator1，分别设置需验证的关联控件、需比较的控件、控件显示方式和错误提示的文字和颜色。第 5 行添加一个服务器按钮控件 Button1 并设置其单击事件 Click。第 7 行添加一个服务器标签控件 Label1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. if (TextBox1.Text == TextBox2.Text){
3. Label1.Text = "二次密码输入相同!";
4. }
5. }

```

代码说明：第 1 行定义处理按钮控件 Button 单击事件 Click 的方法。第 2 行判断如果两个密码文本框中输入的内容相同，则第 4 行在 Label1 控件上显示密码相同的提示文字。

**04** 按下“Ctrl+F5”，运行结果如图 5-3 所示。用户输入的二次密码不同，单击“提交按钮”后，显示 CompareValidator 控件的红色错误提示文字。

**05** 用户输入的二次密码一致，则显示如图 5-4 所示的密码相同的提示。

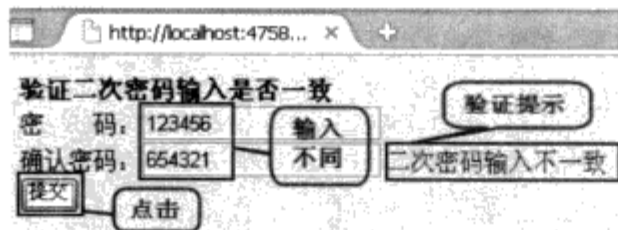


图 5-3 运行结果 1



图 5-4 运行结果 2



提示

如果输入控件为空，CompareValidator 控件将显示验证成功，即 CompareValidator 控件不起作用，因此需要先使用 RequiredFieldValidator 控件验证用户必须输入内容，然后在利用 CompareValidator 控件比较用户输入的内容。

#### 5.2.4 RangeValidator 控件

RangeValidator 控件用于测试输入控件的值是否在指定范围内。在实际应用中，有时需要用户在一定范围内输入某个值，例如用户输入的年龄应该大于 1 小于 200，这时就需要使用 RangeValidator 控件。RangeValidator 控件的使用语法定义如下所示。

```

<asp:RangeValidator ID="RangeValidator1" runat="server">
</asp:CompareValidator>

```

对于 RangeValidator 控件的使用，一般也是通过对其属性设置来完成的，该控件常用的属性如表 5-4 所示。

表 5-4 RangeValidator 控件的常用属性

| 属性                | 说明                                                                                                                                                                                                                                            |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ControlToValidate | 通过设置该属性为某控件的 ID 来把验证控件绑定到需要验证的控件上                                                                                                                                                                                                             |
| ErrorMessage      | 通过该属性来设置当验证控件无效时需要显示的信息                                                                                                                                                                                                                       |
| ValidationGroup   | 绑定到验证程序所属的组                                                                                                                                                                                                                                   |
| Text              | 当验证控件无效时显示的验证程序的文本                                                                                                                                                                                                                            |
| Display           | 通过该属性来设置验证控件的显示模式，该属性有三个值： <ul style="list-style-type: none"><li>● None，表示验证控件无效时不显示信息；</li><li>● Static，表示验证控件在页面上占位是静态的，不能为其它空间所占；</li><li>● Dynamic，表示验证控件在页面上占位是动态的，可以为其它空间所占，当验证失效时验证控件才占据页面位置。</li></ul>                                |
| Type              | 通过该属性来设置按照哪种数据类型来进行比较，常用的数据类型包括： <ul style="list-style-type: none"><li>● String，表示字符串数据类型；</li><li>● Integer，表示 32 位有符号整数数据类型；</li><li>● Double，表示双精度浮点数数据类型；</li><li>● DateTime，表示日期数据类型；</li><li>● Currency，表示一种可以包含货币符号的十进制数据类型。</li></ul> |
| MaximumValue      | 设置用来做比较的数据范围上限                                                                                                                                                                                                                                |
| MinimumValue      | 设置用来做比较的数据范围下限                                                                                                                                                                                                                                |

**【例 5-3】** 在用户注册模块中，经常需要用户输入年龄，事实上用户的年龄不可能无限大，因此就需要对用户输入的年龄值进行验证。本例使用 RangeValidator 控件对用户的年龄进行控制。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-2”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 TextBox 控件、1 个 RangeValidator 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 请输入你的年龄：

2. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. <asp:Button ID="Button1" runat="server" Text="提交" onclick="Button1_Click" />

4. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>

5. <asp:RangeValidator ID="RangeValidator1" runat="server" ControlToValidate="TextBox1" Display="Dynamic"
 ErrorMessage="你输入的值不是一个 1 到 150 之间的整数" ForeColor="Red" MaximumValue="150" MinimumValue="1"
 Type="Integer"></asp:RangeValidator>
```

代码说明：第 2 行添加一个服务器文本框控件 TextBox1 接受用户输入的年龄。第 2 行添加一个服务器按钮控件 Button1 并设置其单击事件 Click。第 4 行添加一个服务器标签控件 Label1。第 5 行添加了一个服务器验证控件 RangeValidator1，分别设置需验证的关联控件、控件显示方式、错误提示的文字和颜色、验证值的数据类型以及值的最大最小范围。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. if (RangeValidator1.IsValid){
3. Label1.Text = "恭喜你，通过了验证！";
4. }
5. else{
6. Label1.Text = "";
7. }
8. }

```

代码说明：第 1 行定义处理按钮控件 Button 单击事件 Click 的方法。第 2 行使用验证控件对象 RangeValidator1 的 IsValid 属性获得文本框中的值，判断如果该值已经通过验证，则第 3 行标签控件上显示通过验证的信息。否则，不在控件中显示任何内容。

**04** 按下“Ctrl+F5”，运行结果如图 5-5 所示。用户输入的值超过验证的范围，单击“提交”按钮后，显示 RangeValidator1 控件的红色错误提示文字。

**05** 如果用户输入的年龄值范围正确。单击“提交”按钮后，显示如图 5-6 所示的成功通过验证的提示。

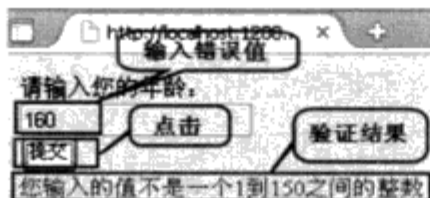


图 5-5 运行结果 1

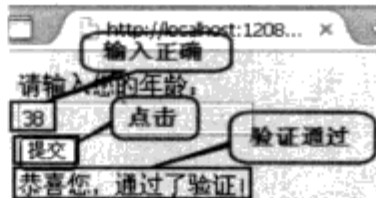


图 5-6 运行结果 2

### 5.2.5 RegularExpressionValidator 控件

RegularExpressionValidator 控件在所有的验证控件中是比较复杂的，并不是控件本身的复杂，而是该控件使用过程中会涉及另一个内容——正则表达式。为了能让大家更好地掌握 RegularExpressionValidator 控件的使用，本章将花一定的篇幅介绍这一编程中常用的知识。

#### 1. RegularExpressionValidator 控件简介

RegularExpressionValidator 控件用于验证相关输入控件的值是否匹配正则表达式指定的模式。在实际应用中，经常需要用户输入一些固定格式的信息，例如电话号码、邮政编码、网址等内容。为了保证用户输入符合规定的要求，例如电话号码，美国、欧洲和中国的表示方法都各不相同，此时就需要使用 RegularExpressionValidator 控件进行验证。RegularExpressionValidator 控件的使用语法定义如下所示。

```

<asp: RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" >
</asp: RegularExpressionValidator >

```

对于 RegularExpressionValidator 控件的使用一般也是通过对其属性设置来完成的，该控件常用的属性如表 5-5 所示。

表 5-5 RegularExpressionValidator 控件的常用属性

| 属性                   | 说明                                                                                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ControlToValidate    | 通过设置该属性为某控件的 ID 来把验证控件绑定到需要验证的控件                                                                                                                                                                               |
| ErrorMessage         | 通过该属性来设置当验证控件无效时需要显示的信息                                                                                                                                                                                        |
| ValidationGroup      | 绑定到验证程序所属的组                                                                                                                                                                                                    |
| Text                 | 当验证控件无效时显示的验证程序的文本                                                                                                                                                                                             |
| Display              | 通过该属性来设置验证控件的显示模式，该属性有三个值： <ul style="list-style-type: none"><li>● None，表示验证控件无效时不显示信息。</li><li>● Static，表示验证控件在页面上占位是静态的，不能为其它空间所占。</li><li>● Dynamic，表示验证控件在页面上占位是动态的，可以为其它空间所占，当验证失效时验证控件才占据页面位置。</li></ul> |
| ValidationExpression | 通过该属性来设置利用正则表达式描述的预定义格式                                                                                                                                                                                        |

在所有的属性中 ValidationExpression 涉及到了正则表达式，该表达式的定义非常复杂。在 Visual Studio 2010 中提供了一个正则表达式的编辑器，这个编辑器中提供了一些常用的正则表达式。要使用这个编辑器，首先在窗体中选中 RegularExpressionValidator 控件，然后在如图 5-7 所示的属性窗口中单击 ValidationExpression 属性旁的省略号按钮即可。

打开如图 5-8 所示的“正则表达式编辑器”对话框。在上面的标准表达式列表中列出了常用的正则表达式名称。选择其中某一个，该正则表达式的具体内容将显示在下面的文本框中。只要单击“确定”按钮，就表示 RegularExpressionValidator 控件将使用这个正则表达式进行验证。

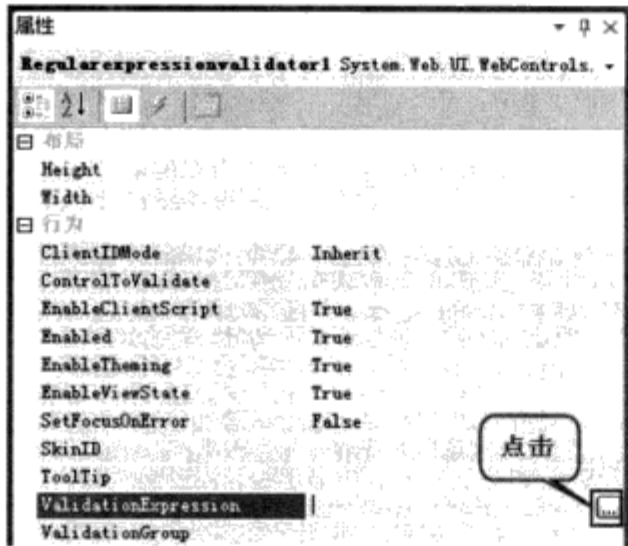


图 5-7 属性窗口

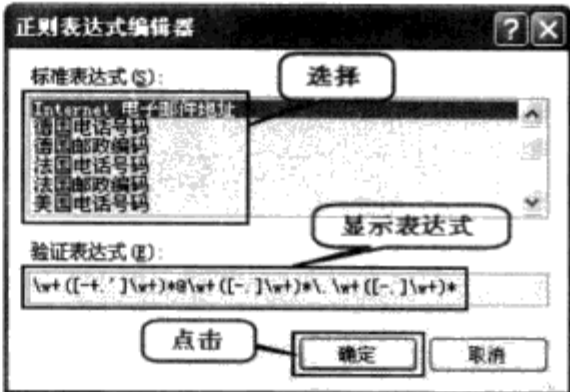


图 5-8 “正则表达式编辑器”对话框

2. 正则表达式

在 Visual Studio 2010 的“正则表达式编辑器”中提供的正则表达式只是很少的一部分，这些正则表达式可能无法满足用户的实际要求，此时就需要开发人员自己来编写正则表达式。为了使读者能对正则表达式有一个大概的了解，下面简单介绍一些正则表达式的基础知识。

正则表达式 (Regular Expression) 描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。简单地说就是用某

种模式去匹配一类字符串的公式。它是由普通字符（例如字符 A 到 Z）以及特殊字符（称为元字符）组成的文字模式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

（1）普通字符

普通字符分为打印字符和非打印字符两种。打印字符包含所有的大小写字母、0~9 的数字以及所有的标点字符。非打印字符如表 5-6 所示。

表 5-6 非打印字符

| 字符 | 说明                      |
|----|-------------------------|
| \n | 匹配一个换行符                 |
| \r | 匹配一个回车符                 |
| \f | 匹配一个换页符                 |
| \t | 匹配一个制表符                 |
| \v | 匹配一个垂直制表符               |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等 |
| \S | 匹配任何非空白字符               |
| \w | 匹配任何单词字符,包括字母和下划线       |
| \W | 匹配任何非单词字符               |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置  |
| \B | 匹配单词的开头                 |
| \d | 匹配一个数字字符，等价于 [0-9]      |
| \D | 匹配任何一个非数字字符             |
| \\ | 匹配 “\”                  |

（2）特殊字符

所谓特殊字符，就是一些有特殊含义的字符，比如 Windows 笔记本文件命名的默认格式“\*.txt”中的“\*”就是一个特殊字符，它表示任何字符串的意思。表 5-7 列举了正则表达式中特殊字符的含义。

表 5-7 特殊字符

| 特殊字符 | 说明                                      |
|------|-----------------------------------------|
| ^    | 匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字符集合 |
| \$   | 匹配输入字符串的结尾位置                            |
| *    | 匹配前面的子表达式零次或多次                          |
| +    | 匹配前面的子表达式一次或多次                          |
| .    | 匹配除换行符 \n 之外的任何单字符                      |
| ?    | 匹配 0 个或 1 个前面的字符                        |
| {    | 标记限定符表达式的开始                             |
|      | 指明两项之间的一个选择                             |
| {n}  | n 是一个非负整数，匹配确定的 n 次                     |

(续表)

| 特殊字符   | 说明                          |
|--------|-----------------------------|
| {n,}   | 匹配至少出现 n 次前面的字符             |
| {m,n}  | 匹配至少 m 个，至多 n 个前面的字符        |
| [xyz]  | 表示一个字符集合，匹配括号中所包含的任意一个字符    |
| [^xyz] | 表示一个否定的字符集合，匹配不在此括号中的任意一个字符 |
| [^m-n] | 表示某个范围之外的字符，匹配不在指定范围内的字符    |
| [a-z]  | 表示一个字符范围，匹配指定范围内的任意字符       |



提示

如果要匹配某个特殊字符本身，就需要用转义字符。比如，如果要匹配某个字符串中是否含有“\*”，就需要使用“\\*”。

(3) 各种操作符的优先级

相同优先级的从左到右进行运算，不同优先级的运算先高后低。各种操作符的优先级从高到低如表 5-8 所示。

表 5-8 操作符的的优先等级

| 优先级 | 操作符                       | 说明      |
|-----|---------------------------|---------|
| 1   | \                         | 转义符     |
| 2   | ()、(?:)、(?:=)、[]          | 圆括号和方括号 |
| 3   | *, +, ?, {n}, {n,}, {n,m} | 限定符     |
| 4   | ^, \$, \anymetacharacter  | 位置和顺序   |
| 5   |                           | 或操作符    |

(4) 应用举例

通过分析匹配中华人民共和国的电话号码的正则表达式来看如何使用正则表达式进行字符匹配，表达式如下所示。

```
(\d{3}\)|\d{3}-)?\d{8}
```

正则表达式说明：我们知道电话号码是由 3 位区号和 8 位号码组成，类似 (021) 12345678 或者是 021-12345678 的两种字符串都被认为是合法的电话号码。可以看到正则表达式中的“(\d{3}\)”部分，“\ (”和“\ )”使用转义字符“\”分别匹配左、右括号，而中间的“\d{3}”则表示由 3 个数字组成的字符串。所以这一部分匹配的是“3 位数字的区号”。上面正则表达式中的“\d{3}-”匹配的就是“3 位数字的区号加一个横杠”。这两个部分之间使用一个“|”来连接，表示这二者取其一。所以“(\d{3}\)|\d{3}-)”整个匹配的是电话号码的区位部分。

正则表达式中的“?”表示区号出现 1 次或 0 次。后面部分中的“\d”匹配一个字符串；“{8}”表示出现 8 次非负数的正数，所以“\d{8}”整个部分表示由 8 位数字组成的字符串，匹配后面的 8 位电话号码。

上面的这个例子比较简单，实际操作中，有些正则表达式的定义非常复杂。由于本书内容重点不在此，所以仅做一个简单的说明，目的是让大家对正则表达式有一个初步的认识。如果读者想

深入学习正则表达式的话，请参考相关专门的书籍。



正则表达式作为一门专门的知识有学习的必要，同时也可以通过网络，在谷歌和百度搜索引擎上搜索相关的正则表达式，这些现成的、数量众多的正则表达式可供我们直接拿来使用。

**【例 5-4】**本例使用 `RegularExpressionValidator` 控件验证用户输入是否匹配指定的模式。添加一个文本框供用户输入身份证号码，由于身份证号码是有固定格式的，为了用户输入非法格式，我们使用一个 `RegularExpressionValidator` 控件来验证用户输入的内容。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-2”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 `RegularExpressionValidator` 控件、1 个 `TextBox` 控件、1 个 `Button` 控件和 1 个 `Label` 控件。然后切换到“源视图”，在编辑区中 `<form></form>` 标记之间编写如下代码。

```
1. 请输入你的身份证号码:

2. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. <asp:Button ID="Button1" runat="server" Text="提交" onclick="Button1_Click" />
4.

5. <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" ControlToValidate="TextBox1"
 Display="Dynamic" ErrorMessage="你输入的身份证号码输入格式错误" ForeColor="Red"
 ValidationExpression="\d{17}[\d|X]\d{15}"></asp:RegularExpressionValidator>
6.

7. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

代码说明：第 2 行添加一个服务器文本框控件 `TextBox1` 接受用户输入的身份证号码。第 2 行添加一个服务器按钮控件 `Button1` 并设置其单击事件 `Click`。第 5 行添加了一个服务器验证控件 `RegularExpressionValidator`，分别设置需验证的关联控件 `TextBox1`、控件显示方式为动态、错误提示的文字和颜色、最关键的是设置 `ValidationExpression` 属性即验证身份证号码格式为 `\d{17}[\d|X]\d{15}`，这个正则表达式是“正则表达式编辑器”的列表中提供的。第 7 行添加一个服务器标签控件 `Label1` 显示提示信息。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Button1_Click(object sender, EventArgs e){
2. if (RegularExpressionValidator1.IsValid) {
3. Label1.Text = "恭喜你，输入的身份证号码正确！";
4. }
5. else{
6. Label1.Text = "";
7. }
8. }
```

代码说明：第 1 行定义处理按钮控件 `Button` 单击事件 `Click` 的方法。第 2 行使用验证控件 `RegularExpressionValidator1` 对象的 `RangeValidator1` 的 `IsValid` 属性获得文本框中的值，判断如果该值已经通过验证，则第 3 行标签控件上显示通过验证的信息。否则，不在控件中显示任何内容。

**04** 按下“Ctrl+F5”，运行结果如图 5-9 所示。用户输入不正确格式的身份证号码，单击“提

交”按钮后，显示 RegularExpressionValidator1 控件的红色错误提示文字。

**05** 如果用户输入身份证的格式正确，单击“提交”按钮后，显示如图 5-10 所示的成功通过验证的提示。

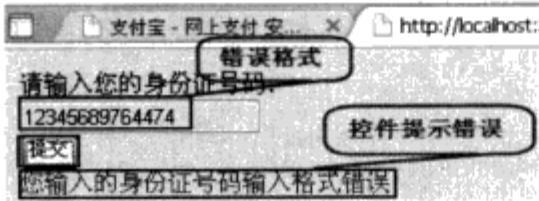


图 5-9 运行结果 1

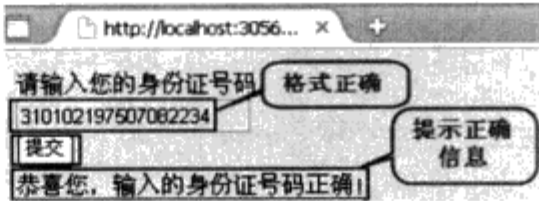


图 5-10 运行结果 2

### 5.2.6 CustomValidator 控件

有时使用现有的验证控件可能满足不了开发人员的需求，因此需要开发人员自己来编写验证函数，而通过 CustomValidator 控件的服务器端事件可以把验证函数绑定到相应的控件。CustomValidator 控件的使用语法定义如下所示。

```
1. <asp: CustomValidator ID="CustomValidator1" runat="server">
2. </asp: RegularExpressionValidator >
```

CustomValidator 控件的使用一般也是通过对其属性设置来完成的，该控件常用的属性如表 5-9 所示。

表 5-9 CustomValidator 控件的常用属性

| 属性                   | 说明                                                                                                                                                                                                             |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ControlToValidate    | 通过设置该属性为某控件的 ID 来把验证控件绑定到需要验证的控件                                                                                                                                                                               |
| ErrorMessage         | 通过该属性来设置当验证控件无效时需要显示的信息                                                                                                                                                                                        |
| ValidationGroup      | 绑定到验证程序所属的组                                                                                                                                                                                                    |
| Text                 | 当验证控件无效时显示的验证程序的文本                                                                                                                                                                                             |
| Display              | 通过该属性来设置验证控件的显示模式，该属性有三个值： <ul style="list-style-type: none"><li>● None，表示验证控件无效时不显示信息；</li><li>● Static，表示验证控件在页面上占位是静态的，不能为其它空间所占；</li><li>● Dynamic，表示验证控件在页面上占位是动态的，可以为其它空间所占，当验证失效时验证控件才占据页面位置。</li></ul> |
| IsValid              | 获取一个值来判断是否通过验证，true 表示通过验证，而 false 表示不通过验证                                                                                                                                                                     |
| ValidationExpression | 通过该属性来判断绑定的控件为空时是否执行验证，该属性为 true 含义是绑定的控件为空时执行验证，为 false 含义则是绑定的控件为空时不执行验证                                                                                                                                     |

除了以上常用的属性，CustomValidator 控件还有一个重要的事件 ServerValidate，启用该控件的 ServerValidate 事件才能把开发人员自定义的函数绑定到相应的控件上。

**【例 5-5】**日期和时间混合格式的验证对开发人员来说，一直是一个比较复杂的工作，但在 ASP.NET 4.0 中，就变得比较简单。本例使用 CustomValidator 控件来验证会员注册时输入的日期和时间格式是否正确。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-5”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 TextBox 控件、1 个 RangeValidator 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 请输入注册的时间:

2. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. <asp:Button ID="Button1" runat="server" Text="提交"/>

4. <asp:CustomValidator ID="CustomValidator1" runat="server" ControlToValidate="TextBox1" Display="Dynamic"
 ForeColor="Red" onservvalidate="CustomValidator1_ServerValidate"></asp:CustomValidator>

5. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

代码说明：第 2 行添加一个服务器文本框控件 TextBox1 接受用户输入的注册时间。第 3 行添加一个服务器按钮控件 Button1 并设置显示的文本。第 4 行添加了一个服务器验证控件 CustomValidator1，分别设置需验证的关联控件 TextBox1、控件显示方式为动态、错误提示的颜色为红色。最关键的是设置验证控件 CustomValidator1 的服务器端验证事件 ServerValidate。第 5 行添加一个服务器标签控件 Label1 显示提示信息。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args){
2. string str = args.Value;
3. try{
4. DateTime date = Convert.ToDateTime(str);
5. Label1.Text = "你输入的时间格式正确！";
6. args.IsValid=true;
7. }
8. catch{
9. CustomValidator1.ErrorMessage = "你输入的时间格式错误！";
10. args.IsValid = false;
11. }
12. }
```

代码说明：第 1 行定义自定义验证控件 CustomValidator1 服务器验证事件 ServerValidate 的方法。第 2 行使用事件源对象 args 的 Value 属性获得传递的文本框用户输入的值。第 4 行如果文本框中的值能够通过转换数据类 Convert 的 ToDateTime 方法转换成时间日期格式，则第 5 行在标签控件上显示通过输入格式正确的信息。第 6 行同时将控件的 IsValid 属性设置为 true，表示验证通过。如果文本框的值无法转换成功，则程序终止 try 语句块中的流程，进入 catch 捕捉异常的语句块中。第 8 行设置验证控件 CustomValidator1 的错误信息属性，在页面显示错误提示。第 9 行同时将控件的 IsValid 属性设置为 false，表示验证不能通过。

**04** 按下“Ctrl+F5”，运行结果如图 5-11 所示。用户输入不正确的时间日期格式，单击“提交”按钮后，显示 CustomValidator1 控件的红色错误提示文字。

**05** 如果用户输入时间日期格式正确，单击“提交”按钮后，显示如图 5-12 所示的成功通过验证的提示。

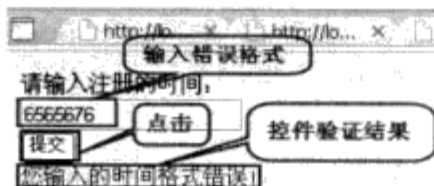


图 5-11 运行结果 1

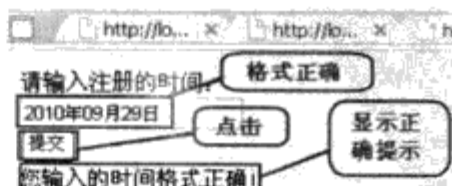


图 5-12 运行结果 2

5.2.7 ValidationSummary 控件

ValidationSummary 控件用于显示页面中的所有验证错误的摘要。当页面上有很多验证控件时，可以使用一个 ValidationSummary 控件在一个位置总结来自 Web 页上所有验证程序的错误信息。ValidationSummary 控件的使用语法定义如下所示。

```
<asp: ValidationSummary ID="ValidationSummary" runat="server" >
</asp: ValidationSummary >
```

ValidationSummary 控件的使用一般也是通过对其属性设置来完成的，该控件常用的属性如表 5-10 所示。

表 5-10 ValidationSummary 控件的常用属性

| 属性             | 说明                                                                                                                                                                              |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HeaderText     | 验证摘要页的标题部分显示的文本                                                                                                                                                                 |
| ShowMessage    | 指定是显示还是隐藏 ValidationSummary 控件，如果属性值为 true，则显示 ShowSummary 控件，否则不显示该控件                                                                                                          |
| ShowMessageBox | 用于指定是否显示一个消息对话框显示验证的摘要信息，如果属性值为 true，则显示消息对话框，否则不显示                                                                                                                             |
| ValidationGrop | 用于指定验证控件所属的验证组的名称                                                                                                                                                               |
| DisplayMode    | 通过该属性来设置验证摘要的显示模式，该属性有三个值： <ul style="list-style-type: none"><li>BulletList，默认的显示模式，每个消息都显示为单独的项；</li><li>List，每个消息显示在单独的行中；</li><li>SingleParagraph，每个消息显示为段落中的一个句子。</li></ul> |

【例 5-6】本例通过使用 ValidationSummary 控件来收集用户注册页面中其他验证控件的错误提示信息并进行统一显示处理。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-6”。
- 02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 ValidationSummary 控件、1 个 RegularExpressionValidator 控件、1 个 CompareValidator 控件、1 个 RequiredFieldValidator 控件、3 个 TextBox 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. 注册

2. 用户名: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
3. <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="TextBox1" ErrorMessage=
"用户名必填"Display="None"></asp:RequiredFieldValidator>

4. 密 码: <asp:TextBox ID="TextBox2" runat="server" TextMode="Password" Height="18px" Width="153px"></asp:TextBox>
5. <asp:CompareValidator ID="CompareValidator1" runat="server" ControlToValidate="TextBox2" ErrorMessage="密码输入
错误"ValueToCompare="123456" Display="None"></asp:CompareValidator>

6. E-mail: <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
7. <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" ControlToValidate="TextBox3"
ErrorMessage="格式错误" ValidationExpression="\w+([-.\']\w+)*@\w+([-.\']\w+)*\.\\w+([-.\']\w+)*"
Display="None"></asp:RegularExpressionValidator>

8. <asp:Button ID="Button1" runat="server" Text="注册" onclick="Button1_Click" />

9. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

```
10. <asp:ValidationSummary ID="ValidationSummary1" runat="server" BorderColor="Red" BorderStyle="Solid"
 BorderWidth="1px" ForeColor="#404040" HeaderText="所有的错误信息提示" style="margin-top: 0px" Width="196px" />
```

代码说明：第 2 行添加一个服务器文本框控件 TextBox1 用于接受用户名的输入。第 3 行添加了一个服务器 RequiredFieldValidator1 验证控件，分别设置需验证的关联控件 TextBox1、显示方式 Display 属性为不显示以及显示错误信息文字内容。第 4 行添加一个服务器文本框控件 TextBox1 用于接受密码的输入。第 5 行添加一个服务器 CompareValidator1 验证控件，分别设置关联的控件 TextBox2、显示方式 Display 属性为不显示、ValueToCompare 属性即输入密码值的比较对象为“123456”、显示错误信息文字内容。第 6 行添加一个服务器文本框控件 TextBox3 用于接受用户名的 E-Mail 地址。第 7 行添加了一个服务器 RegularExpressionValidator1 验证控件，分别设置需关联的控件为 TextBox13、错误显示文字、验证的 E-mail 地址的正则表达式、显示方式 Display 属性为不显示。第 8 行添加服务器按钮控件 Button1 用于提交注册，并设置控件上显示的文字和按钮的单击事件。第 9 行添加一个服务器标签控件 Label1。第 10 行添加一个服务器 ValidationSummary1 验证摘要控件，分别设置边框的颜色、样式和宽带；设置文字的颜色、控件摘要的标题、样式和宽度。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
protected void Button1_Click(object sender, EventArgs e){
 Label1.Text = "恭喜你，注册成功！";
}
```

代码说明：第 1 行定义处理注册按钮的事件 Click 的方法。第 2 行在标签控件上显示注册成功的提示。

**04** 按下“Ctrl+F5”，运行结果如图 5-13 所示。如果用户未输入用户名、密码输入错误或 E-mail 地址输入不正确的情况。显示 ValidationSummary1 验证错误的摘要列表。

**05** 如果用户输入都符合要求，单击“注册”按钮后，显示如图 5-14 所示的成功通过注册的提示。

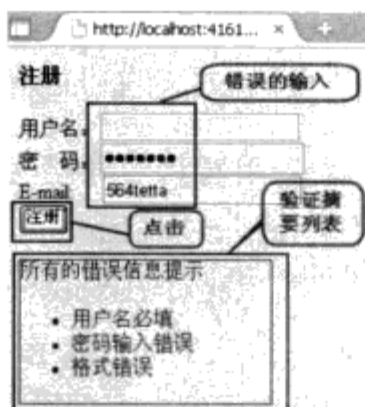


图 5-13 运行结果 1



图 5-14 运行结果 2



提示

RequiredFieldValidator 控件、CompareValidator 控件、RegularExpressionValidator 控件和 RangeValidator 控件必须设置 ControlToValidate 属性，否则就出现编译错误；而不设置 ControlToValidate 属性的 CustomValidator 控件却是可以使用的。

## 5.3 用户控件

在开发网站的时候，有时会发现具有同样功能控件的组合会经常重复出现在页面中，比如具有查询数据功能的控件，这时我们可能会试图采用一种方法来定义一个可重复利用的控件，并且希望这种控件能够像 ASP.NET 系统提供的标准控件那样可以很方便地拖放到网页中，从而减少重复代码的编写工作，以提高开发效率。所以 ASP.NET 4.0 中提供了一种让我们可以根据自己的需要来开发的自定义控件——用户控件。

### 5.3.1 简述

一个用户控件就是一个简单的 ASP.NET 页面，不过它可以被另外一个 ASP.NET 页面包含进去。用户控件存放在文件扩展名为 .ascx 的文件中，典型的 .ascx 文件中的代码如下：

```
1. <%@ Control Language="C#" AutoEventWireup="true" CodeFile="WebUserControl.ascx.cs"
 Inherits="WebUserControl" %>
2. <asp:Label ID="Label1" runat="server" Text="Hello World"></asp:Label>
```

代码说明：第 1 行代码和 .aspx 文件一样，没有什么区别，只是把 Page 指令换成了 Control 指令，第 2 行添加了一个服务器标签控件，显示文本“Hello World”。

从以上 .ascx 文件中的代码可以看出，用户控件代码格式和 .aspx 文件中的代码格式非常相似，.ascx 文件中没有 <html> 标记，也没有 <body> 标记和 <form> 标记，因为用户控件是要被 .aspx 文件所包含，而这些标记在一个 .aspx 文件都只能包含一个。一般说来，用户控件和 ASP.NET 网页有如下区别。

- 用户控件的文件扩展名为 .ascx。
- 用户控件中没有 @ Page 指令，而是包含 @ Control 指令，该指令对配置及其他属性进行定义。
- 用户控件不能作为独立文件运行。而必须像处理任何控件一样，将它们添加到 ASP.NET 页中。
- 用户控件中没有 html、body 或 form 元素。这些元素必须位于宿主页中。

用户控件提供了这样一种机制，它使得程序员可以建立能够非常容易被 ASP.NET 页面使用或者重新利用的代码部件。在 ASP.NET 应用程序当中，使用用户控件的一个主要优点是用户控件支持一个完全面向对象的模式，使得程序员有能力去捕获事件。而且，用户控件支持程序员使用一种语言编写 ASP.NET 页面中的一部分代码，而使用另外一种语言编写 ASP.NET 页面的另外一部分代码，因为每一个用户控件可以使用和主页面不同的语言来编写。



提示

用户控件参与每个请求的整个执行生存期，并且可以处理自己的事件，封装来自包装含 Web 窗体页面的一些代码逻辑。

### 5.3.2 用户控件的创建和使用

如果想要在程序中实现用户控件的功能，首先要做的是创建一个后缀名为 .ascx 的用户控件，这一过程与创建普通的 aspx 窗体页面并没有多大的不同。但是，当用户访问页面时，该用户控件是不能被用户直接访问的，所以必须在 Web 窗体中通过注册的方式调用创建成功的用户控件。

下面，我们创建一个用户控件并在窗体中调用它，具体步骤如下所示。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序。

**02** 右键单击“网站项目”名称, 在弹出如图 5-15 所示的菜单中选择“添加新项”|“新建项”命令。

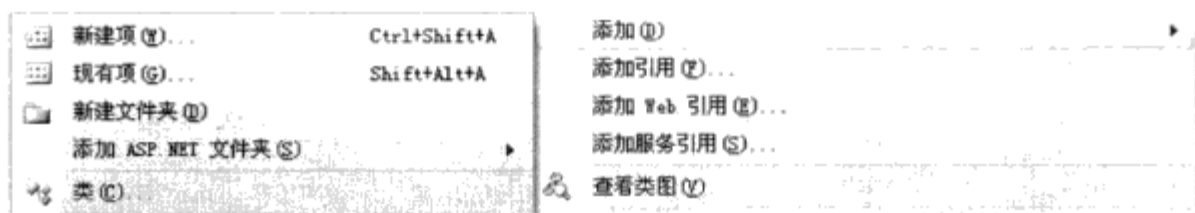


图 5-15 选择“新建项”命令

**03** 弹出如图 5-16 所示的“添加新项”对话框。在该对话框中选择“已安装模板”下的“Web”模板, 并在模板文件列表中选中“Web 用户控件”, 然后在“名称”文本框输入该文件的名称“WebUserControl1.ascx”, 最后单击“添加”按钮。

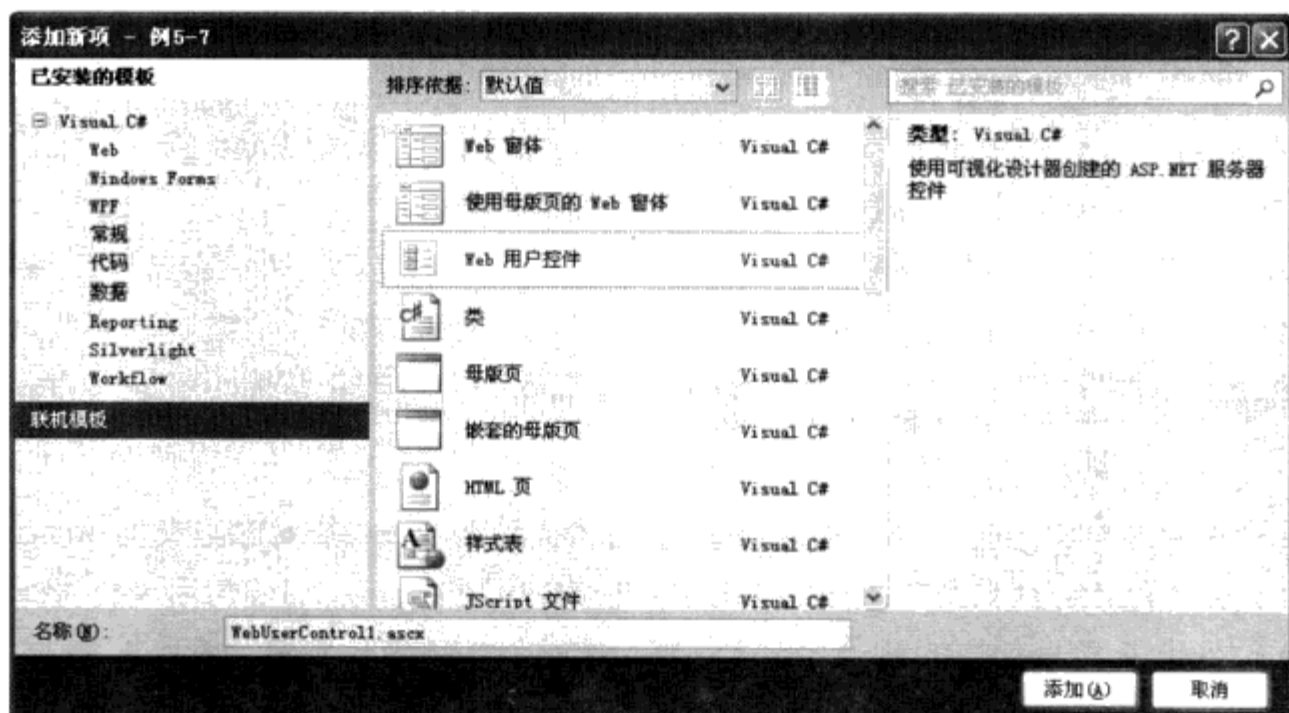


图 5-16 “添加新项”对话框

**04** 此时解决方案资源管理中的项目下会生成一个如图 5-17 所示的“WebUserControl1.ascx”的页面, 它包括两个文件, 一个是“WebUserControl1.ascx.cs”, 用于编写后台代码。另一个是“WebUserControl1.ascx.designer.cs”, 存放的是一些用户控件中使用控件的配置信息。

**05** 用鼠标双击“WebUserControl1.ascx”文件, 在文件中生成的初始代码如下:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="WebUserControl1.ascx.cs" Inherits="WebUserControl1" %>
```

代码说明: 以上是用户控件的界面定义代码, @ Control 指令说明这是一个用户控件的文件。CodeFile 属性指明了用户控件后台代码文件是“WebUserControl1.ascx.cs”, AutoEventWireup 属性设置控件的事件自动匹配, Inherits 属性说明该控件的名称为 WebUserControl。

**06** 接下来就可以设计用户控件的外观。切换到“视图设计器”, 从“工具箱”拖动 1 个 Label 控件、1 个 TextBox 控件、1 个 Button 控件到如图 5-18 所示的“设计视图”中。如果需要添加用户控件的事件, 可以在“WebUserControl1.ascx.cs”文件的后台代码中进行编写。



图 5-17 生成.ascx 文件



图 5-18 用户控件外观

**07** 创建完毕用户控件，我们就可以在页面中调用注册该用户控件。用鼠标双击打开“Default.aspx”文件并切换到“视图设计器”，然后在解决方案资源管理中的项目下用鼠标单击“WebUserControl1.ascx”文件不放开，直接拖动到“设计视图”中的想要放置的地方。此时在设计视图中就呈现出如图 5-19 所示的用户控件。

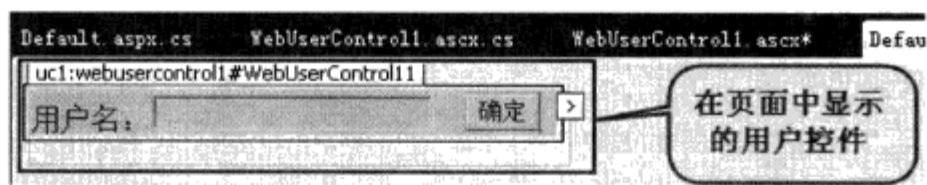


图 5-19 页面中的用户控件

**08** 切换到 Default.aspx 的“源视图”，会看到关键的注册和声明用户控件的代码。

1. `<%@ Register src="WebUserControl1.ascx" tagname="WebUserControl1" tagprefix="uc1" %>`
2. `<uc1:WebUserControl1 ID="WebUserControl11" runat="server" />`

代码说明：第 1 行是注册用户控件到页面的代码。其中 `@Register` 指令提供了 ASP.NET 在运行期间检索控件所需要的所有信息。Src 属性是用户控件的虚拟路径，如果用户控件与包含它的页面在相同的目录中，那只需要提供文件名，如果用户控件在另一个目录中，那么需要提供相对或绝对路径。Tagname 属性表示当前页面中关联到用户控件的名称，可以使用任意的名称，在页面上创建用户控件的实例时要使用这个名称。Tagprefix 属性表示当前页面中关联到用户控件的命名空间（以便多个同名的用户控件可以相互区分），可以使用任意字符串。如果使用相同的 Tagname 向页面添加另一个用户控件，仍然可以使用 Tagprefix 来区分这两个控件。

第 2 行是当在页面注册了用户控件后，Web 页面会出现把用户控件添加到页面的标记。可以在一个页面中多次使用相同的用户控件，唯一的要求就是每个实例具有唯一的 ID。

至此，我们就可以在程序中使用该用户控件了。

### 5.3.3 用户控件的示例

用户控件在实际的编程中使用比较广泛，我们可以把 Web 程序中经常需要使用到的功能制作成一个用户控件，以后在其他 Web 程序中可以重复使用，比如常见的网站的登录或者是注册功能。下面通过一个实例来学习一个用户控件的开发。

**【例 5-7】**本例将开发一个酒店管理系统中使用的用户控件，其功能是完成对酒店房间信息的修改。当用户修改过程中发生错误，在页面给出提示错误信息，如果修改成功，同样在页面显示修改成功的提示信息。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 5-7”。

**02** 右键单击“例 5-7”名称，在弹出的菜单中选择“添加新项”|“新建项”命令。

**03** 弹出“添加新项”对话框。在该对话框中选择“已安装模板”下的“Web”模板，并在模板文件列表中选中“Web 用户控件”，然后在“名称”文本框输入该文件的名称“MyControl.ascx”，最后单击“添加”按钮。

**04** 用鼠标双击“MyControl.ascx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 Table 控件、2 个 TextBox 控件、2 个 DropDownList 控件、3 个 RadioButtonList 控件、1 个 Button 控件和 1 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. <table style="width:100%; text-align:center" border="0">
2. <tr>
3. <td style="text-align: left;"></td>
4. <td style="text-align: left;">
5.
6. 修改房间信息</td>
7. </tr>
8. <tr>
9. <td style="text-align: right;">房号: </td>
10. <td colspan="2" style="width: 70%; text-align: left;">
11. <asp:TextBox ID="txtNum" runat="server"></asp:TextBox>
12. </td>
13. </tr>
14. <tr>
15. <td style="text-align: right;">房型: </td>
16. <td colspan="2" style="width: 70%; text-align: left;">
17. <asp:DropDownList ID="drop" runat="server" Width="158px">
18. <asp:ListItem Selected="True" Value="标准房">标准房</asp:ListItem>
19. <asp:ListItem>单人房</asp:ListItem>
20. <asp:ListItem>商务房</asp:ListItem>
21. <asp:ListItem>套房</asp:ListItem>
22. </asp:DropDownList></td>
23. </tr>
24. <tr>
25. <td style="text-align: right;">房态: </td>
26. <td colspan="2" style="width: 70%; text-align: left;">
27. <asp:DropDownList ID="dropStatu" runat="server" Width="158px">
28. <asp:ListItem Selected="True" Value="住客房">住客房</asp:ListItem>
29. <asp:ListItem Value="空房">空房</asp:ListItem>
30. <asp:ListItem Value="退房">退房</asp:ListItem>
31. <asp:ListItem Value="维修房">维修房</asp:ListItem>
32. </asp:DropDownList></td>
33. </tr>
34. <tr>
35. <td style="text-align: right;">预定状况: </td>
36. <td colspan="2" style="width: 70%; text-align: left;">
37. <asp:RadioButtonList ID="rablBook" runat="server" RepeatDirection="Horizontal">
38. <asp:ListItem Value="有">有</asp:ListItem>
39. <asp:ListItem Value="无" Selected="True">无</asp:ListItem>
40. </asp:RadioButtonList></td>
41. </tr>
42. <tr>
43. <td style="text-align: right;">床位: </td>
44. <td colspan="2" style="width: 70%; text-align: left;">
45. <asp:TextBox ID="txtBed" runat="server"></asp:TextBox></td>
46. </tr>
47. <tr>
48. <td style="text-align: right;" class="style2">电视: </td>

```

```

49. <td colspan="2" style="width:70%; text-align: left;">
50. <asp:RadioButtonList ID="radlTv" runat="server"
51. RepeatDirection="Horizontal" >
52. <asp:ListItem Selected="True">有</asp:ListItem>
53. <asp:ListItem>无</asp:ListItem>
54. </asp:RadioButtonList></td>
55. </tr>
56. <tr>
57. <td style="text-align: right" class="style2">空调: </td>
58. <td colspan="2" style="width:70%; text-align: left;">
59. <asp:RadioButtonList ID="rablAir" runat="server" RepeatDirection="Horizontal" >
60. <asp:ListItem Selected="True">有</asp:ListItem>
61. <asp:ListItem>无</asp:ListItem>
62. </asp:RadioButtonList></td>
63. </tr>
64. <tr>
65. <td ></td>
66. <td style="text-align: left;">
67. <asp:Button ID="btnUpdate" runat="server" Text="修改信息" OnClick="btnUpdate_Click" />
68. <asp:Label ID="Label1" runat="server" Text="" ForeColor="Red"></asp:Label></td>
69. </tr>
70. </table>

```

代码说明：以上代码使用了一个 9 行 2 列的表格显示修改房间信息的界面。第 5 行在表格的第 1 行第 2 列中显示注册的标题并设置了文字的字体和大小。第 10 行在表格的第 2 行第 2 列中添加了一个服务器文本框控件 txtNum 获取用户输入要修改的房号。第 17~22 行在表格第 3 行第 2 列中添加一个服务器下拉列表控件 drop 显示房型，其中，第 19~21 行添加了三个列表项。第 27~32 行在表格第 4 行第 2 列中添加一个服务器下拉列表控件 dropStatu 显示房态，其中，第 29~31 行添加了三个列表项。

第 37~40 行在表格的第 5 行第 2 列中添加一个服务器单选按钮列表控件 rablBook，供用户选择预定状况并设置控件水平方向的布局，其中，第 38 和 39 行添加了两个不同的按钮选项。第 25 行在表格的第 6 行第 2 列中添加了一个服务器文本框控件 txtBed 获取用户输入要修改的床位数。第 50~54 行在表格的第 7 行第 2 列中一个服务器单选按钮列表控件 radlTv 供用户选择房间是否有电视机并设置控件水平方向的布局，其中，第 52~53 行添加了两个不同的按钮选项。

第 59~62 行在表格的第 8 行第 2 列中添加一个服务器单选按钮列表控件 rablAir，供用户选择房间是否有空调设备并设置控件水平方向的布局，其中，第 60~61 行添加了两个不同的按钮选项。第 67 行在表格的第 9 行第 2 列中添加一个服务器按钮控件 btnUpdate 用户提交房间的修改信息，同时设置了控件的显示文本和要处理的单击事件 Click。第 68 行添加一个服务器标标签控件 Label1 并设置其显示文字的颜色。

#### 05 用鼠标双击“MyControl.ascx.cs”文件，编写以下代码：

```

protected void btnUpdate_Click(object sender, EventArgs e){
 if (txtNum.Text == "" && txtBed.Text == "" || txtNum.Text == "" || txtBed.Text == ""){
 Label1.Text = "修改房间信息失败！";
 }
 else{
 string num = txtNum.Text;
 string type = drop.SelectedValue;
 string statu = dropStatu.SelectedValue;
 string book = rablBook.SelectedValue;
 string bed = txtBed.Text;
 }
}

```

```

string tv = radlTv.SelectedValue;
string air = radlAir.SelectedValue;
Label1.Text = "修改信息成功: 房号:" + num + ", 房型: " + type + ", 房态: " + statu + ",";
Label1.Text += "床位数: " + bed + ", 电视: " + tv + ", 空调: " + air + "。";
}
}

```

代码说明: 第 1 行定义处理提交注册按钮 btnUpdate 单击事件 Click 的方法。第 2 行判断如果用户没有输入房号文本框和床位数文本框, 则第 3 行标签控件显示注册失败的文字。否则, 第 6 行获得用户输入房号。第 7 行获取房型信息。第 8 行获取房态信息。第 9 行获取是否有预定。第 10 行获取床位数。第 11 行获取房间是否有电视。第 12 行获取房间是否有空调。第 13~14 行在标签控件上显示修改房间信息成功的提示并将修改的具体信息呈现在页面。

**06** 将创建完毕的用户控件直接拖动到 Default.aspx 文件的“设计视图”中。

**07** 按下“Ctrl+F5”, 运行结果如图 5-20 所示。用户修改房间信息出现错误后单击“修改信息按钮”, 出现红色的提示修改失败的信息。

**08** 用户修改房间信息正确后, 单击“修改信息按钮”, 出现如图 5-21 红色的提示修改成功的信息。

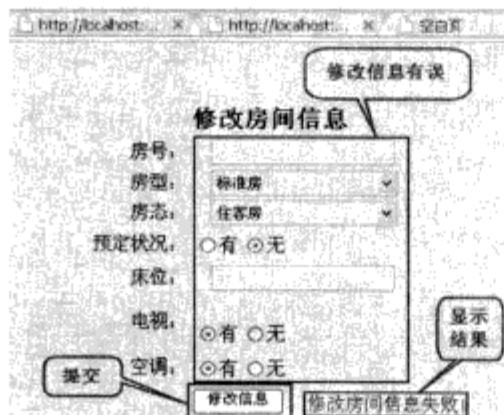


图 5-20 运行结果 1

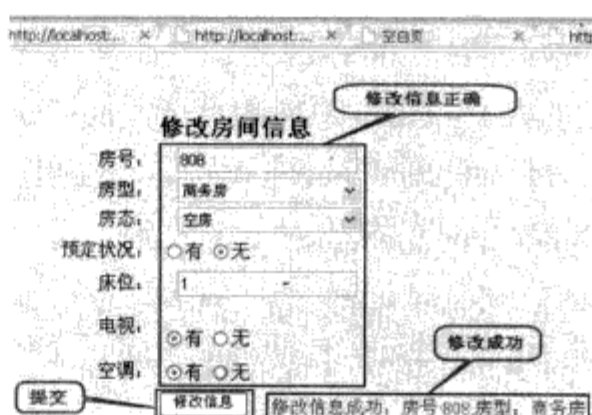


图 5-21 运行结果 2

## 5.4 上机题

1. 提供一个文本框供用户输入, 然后使用自定义控件来控制文本框的数量是否为一个 0~100 之间的奇数。输入不同的值, 会出现不同的提示。运行结果如图 5-22 所示。

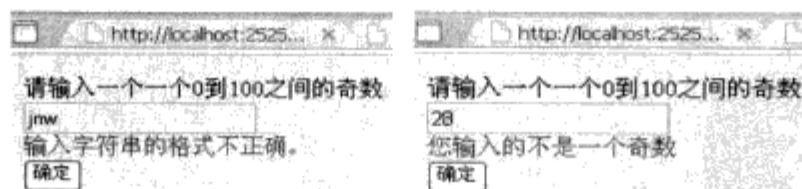


图 5-22 程序运行结果

2. 创建一个用户酒店预订的页面, 该页面提示用户输入预订的日期、人数和电子邮件地址, 使用本章的各种验证控件对页面输入进行验证, 提示验证错误的信息。运行结果如图 5-23 所示。

3. 创建一个网页, 该网页包含两个 TextBox 控件, 使用比较验证控件, 验证在第二个文本框用户输入的数字小于第一个文本框中输入的数字, 运行结果如图 5-24 所示。

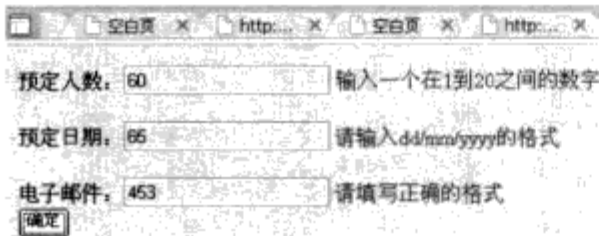


图 5-23 程序运行结果



图 5-24 程序运行结果

4. 创建一个自定义的用户控件，实现通过验证控件在网站注册用户信息的功能，通过不同的验证控件对用户名、密码、重复密码、年龄、电话、E-Mail 进行验证，运行效果如图 5-25 所示。
5. 在上机题 4 的基础上，不使用单个验证控件的错误信息提示，而是使用验证摘要控件 ValidationSummary 实现验证错误的列表集中显示，运行结果如图 5-26 所示。



图 5-25 程序运行结果

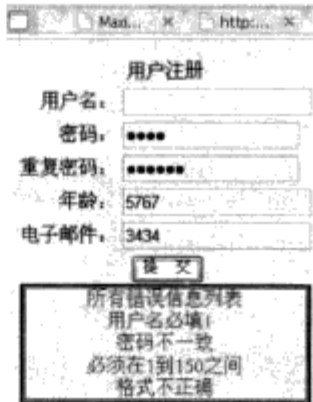


图 5-26 程序运行结果

6. 在一些论坛和博客的网站都有让用户对帖子进行评论的功能。本题要求实现用户输入评论内容时，使用服务器验证控件检查用户输入的字符数不能少于 10 个。如果没有达到要求，由验证控件显示错误的提示。程序运行结果如图 3-27 所示。
7. 在动态网站的数据录入页面中，经常需要对用户输入的登录名进行判断。本题要求用户输入的真实姓名为汉字，如果用户输入的不是汉字，将被视为不合法。程序运行结果如图 5-28 所示。
8. 在一些涉及到办理出国护照的系统中，经常需要验证护照号码是否合法。本题要求通过服务器验证控件来验证注册会员输入的护照号码是否合法。程序运行结果如图 5-29 所示。

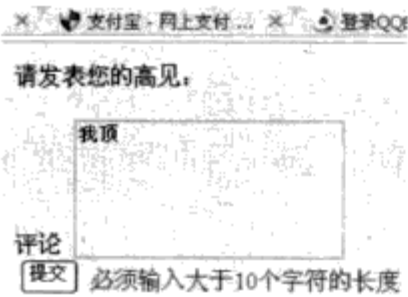


图 5-27 程序运行结果



图 5-28 程序运行结果



图 5-29 程序运行结果

# 第 6 章 ADO.NET 数据库编程

## 学习目标

ASP.NET 4.0 提供了 ADO.NET 技术，它是 ASP.NET 4.0 应用程序与数据库进行交互的一种技术。应用程序可以通过 ADO.NET 连接到各种数据源，并检索、操作和更新数据。ADO.NET 技术把对数据库的操作分成几个步骤，并为每个步骤提供对象来封装操作过程，利用 ADO.NET 开发人员可以简单而快速地访问数据库。通过本章的学习能够使读者掌握对数据库访问的基本技术。

## 本章重点

- 在 SQL Server 2005 中创建数据库
- 使用 DataReader 读取数据
- DataSet 对象和 DataAdapter 对象的配合应用
- 使用 DataTable 对象创建数据表

### 6.1 创建数据库

在介绍 ADO.NET 数据库编程之前，首先要学会创建最基本的数据库。虽然 ADO.NET 可以用来访问任何类型的数据库，但由于 SQL Server 数据库也是微软公司的主打产品，与 Visual Studio 开发环境有着先天上的契合性，所以本书使用了 SQL Server 2005 数据库管理系统。

#### 6.1.1 使用 SQL Server 2005 创建数据库

本小节将详细为大家介绍如何在 SQL Server 2005 中创建数据库“BookStore”和数据表“BookInfo”的过程，本章中所有涉及数据库操作都将使用该数据库，具体创建步骤如下：

**01** 打开 Microsoft SQL Server Management Studio，弹出“连接到服务器”对话框，如图 6-1 所示。



图 6-1 连接到服务器

(2) 读者选择合适的服务器名称和身份验证方式后, 在“连接到服务器”对话框单击“连接”按钮, 连接到 SQL Server 服务器。连接成功后, 进入程序的主界面, 如图 6-2 所示。

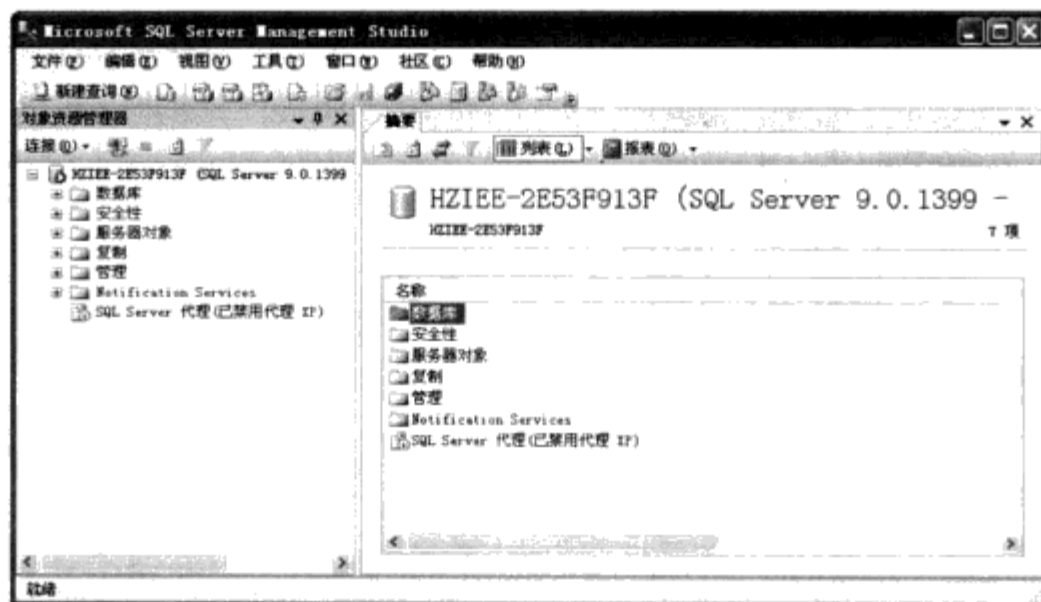


图 6-2 SQL Server Management Studio

**03** 在“对象资源管理器”中右键单击“数据库”, 从弹出的上下文菜单中选择“新建数据库”命令, 弹出如图 6-3 所示的对话框。



图 6-3 新建数据库

**04** 在“数据库名称”中输入读者想要创建的数据库, 这里输入的名称为 BookStor, 单击“确定”按钮创建 BookStor 数据库。此时读者会发现在“对象资源管理器”的“数据库”节点中增加了一个名为 BookStor 的数据库, 如图 6-4 所示。

**05** 展开 BookStor 节点, 右键单击“表”节点, 开始进行表编辑操作, 如图 6-5 所示。



图 6-4 对象资源管理器



图 6-5 编辑表

**06** 在右侧的属性窗体中把表的名称改为 BookInfo，然后在编辑表的窗体中加入 4 列，最终结果如图 6-6 所示。

**08** 右键单击“编号”列，在弹出的上下文菜单中选择“设置主键”命令，“编号”成为该表的主键，如图 6-7 所示。

| 列名     | 数据类型          | 允许空                      |
|--------|---------------|--------------------------|
| ID     | nvarchar(100) | <input type="checkbox"/> |
| Name   | nvarchar(100) | <input type="checkbox"/> |
| Author | nvarchar(50)  | <input type="checkbox"/> |
| Press  | nvarchar(100) | <input type="checkbox"/> |

图 6-6 BookInfo 表

| 列名     | 数据类型          | 允许空                      |
|--------|---------------|--------------------------|
| ID     | nvarchar(100) | <input type="checkbox"/> |
| Name   | nvarchar(100) | <input type="checkbox"/> |
| Author | nvarchar(50)  | <input type="checkbox"/> |
| Press  | nvarchar(100) | <input type="checkbox"/> |

图 6-7 设置主键之后的 BookInfo 表

**08** 在“对象资源管理器”中右键单击 BookStore 数据库的 BookInfo 表，从弹出的上下文菜单中选择“打开表”命令，向表中输入记录，如图 6-8 所示。

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 西游记   | 吴承恩    | 人民文学出版社 |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蘅塘退士   | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |
| NULL   | NULL  | NULL   | NULL    |

图 6-8 向 BookInfo 表中添加记录

至此，就完成了数据库和数据表的基本创建过程。

### 6.1.2 在 Visual Studio 2010 中管理数据库

如果手头没有 SQL Server 2005 的话,还可以直接通过使用 Visual Studio 2010 中的服务器资源管理器来管理已经创建的数据库。具体的操作步骤如下所示。

**01** 启动 Visual Studio 2010,选择“菜单栏”中的“视图”|“服务器资源管理”命令,打开如图 6-9 所示的“服务器资源管理器窗口”,使用鼠标右键单击的“数据连接”,从弹出的上下文菜单中选择“添加连接”命令。

**02** 在弹出的“添加连接”对话框中单击“浏览”按钮,选择已创建好的“BookInfo”的数据库文件。单击“测试连接”按钮,如果连接成功,会弹出提示连接成功的对话框,然后单击该对话框中的“确定”按钮。最后单击“添加连接”对话框中“确定”按钮,如图 6-10 所示。

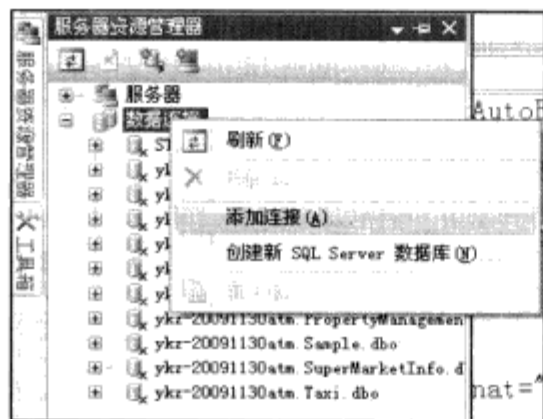


图 6-9 服务器资源管理器窗口



图 6-10 添加连接对话框

**03** 这时在图 6-11 所示的“服务器资源管理器窗口”中的“数据连接”节点下会出现刚才添加好的“BookStor.mdf”数据库。展开“BookStor.mdf”节点|“表”节点,可以看到上例创建的“BookInfo 表”。

**04** 用鼠标双击“BookInfo”,弹出如图 6-12 所示的“表编辑窗口”可以浏览和修改数据表的结构。



图 6-11 BookStor.mdf 数据库

| 列名     | 数据类型           | 允许为 null                 |
|--------|----------------|--------------------------|
| ID     | nvarchar (100) | <input type="checkbox"/> |
| Name   | nvarchar (100) | <input type="checkbox"/> |
| Author | nvarchar (50)  | <input type="checkbox"/> |
| Press  | nvarchar (100) | <input type="checkbox"/> |

图 6-12 表编辑窗口

**05** 使用鼠标右键单击的“BookInfo”,从弹出如图 6-13 所示的上下文菜单中选择“显示表数据”命令。

06 在弹出如图 6-14 所示的“表编辑窗口”中，可以浏览和修改“BookInfo”数据表的数据内容。



图 6-13 显示表示数据

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 西游记   | 吴承恩    | 人民文学出版社 |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蒋驷士    | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |
| *      | NULL  | NULL   | NULL    |

图 6-14 表编辑窗口

使用 Visual Studio 2010 中的服务器资源管理器来管理已经创建的数据库，虽然没有 SQL Server 2005 提供的图形化管理工具来得功能强大，但也可以满足基本的操作需要。

## 6.2 ADO.NET 概述

随着应用程序开发模式的发展和演变，新的应用程序模型对访问数据库的要求越来越高。而 ADO.NET 正是对 Microsoft ActiveX Data Objects (ADO) 的一个跨时代改进，它提供了平台互操作性和可伸缩的数据访问功能。在 .NET 4.0 框架中，传送的数据采用可扩展标记语言 (Extensible Markup Language, XML) 格式，因此任何能够读取 XML 格式的应用程序都可以进行数据处理。事实上，接受数据的组件不一定要是 ADO.NET 组件，它可以是一个基于 Microsoft Visual Studio 的解决方案，也可以是运行在其它平台上的任何应用程序。

### 6.2.1 ADO.NET 简介

在 Web 系统开发中，数据的操作占据了大量的工作，要操作的数据包括：存储在数据库中的数据、存储在文件中的数据以及 XML 数据，其中操作存储在数据库中的数据最为普遍。ASP.NET 提供了 ADO.NET 技术，它是一组向 .NET 编程人员公开数据访问服务的类。ADO.NET 提供了对关系数据、XML 和应用程序数据的访问，所以是 .NET Framework 不可缺少的一部分。ADO.NET 支持多种开发需求，包括创建由应用程序、工具、语言或 Internet 浏览器使用的前端数据库客户端和中间层业务对象。

ADO.NET 组件将数据访问与数据处理分离。它通过两个主要的组件：.NET 数据提供程序 (data provider) 和 Dataset 来完成这一操作的。图 6-15 说明了数据访问与数据处理分离的概念。

图 6-15 是 ADO.NET 的组件结构图，在这里除了可以清楚地看到其内部组成，还可以看出数据访问一般有两种方式：一是通过 DataReader 对象来直接访问，另一种则是通过 DataSet 和 DataAdapter 来访问。

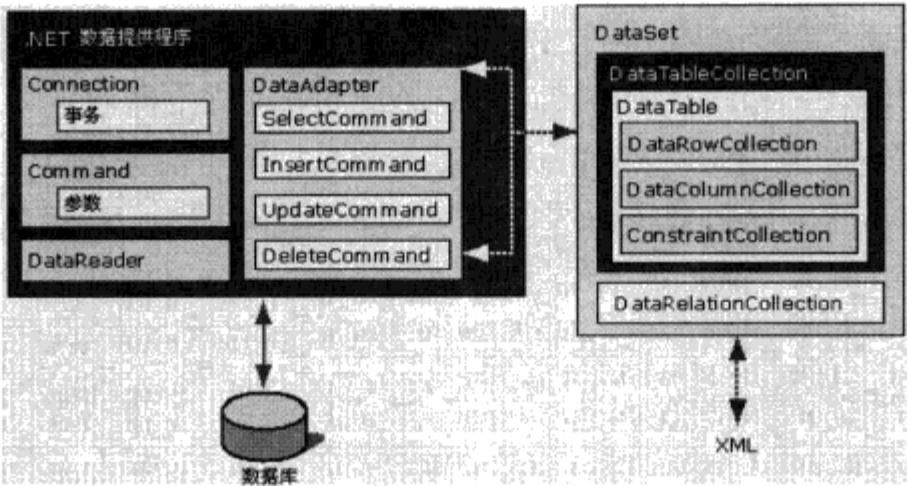


图 6-15 ADO.NET 组件结构图

ADO.NET 体系结构的一个核心元素是 .NET 数据提供程序，它是专门为数据处理以及快速地只进、只读访问数据而设计的组件，包括 Connection、Command、DataReader 和 DataAdapter 对象的组件，具体如表 6-1 所示。

表 6-1 数据提供者的对象

| 对象名称        | 描述                                                                                               |
|-------------|--------------------------------------------------------------------------------------------------|
| Connection  | 提供与数据源的连接                                                                                        |
| Command     | 用于返回数据、修改数据、运行存储过程以及发送或检索参数信息的数据库命令                                                              |
| DataReader  | 从数据源中提供高性能的数据流                                                                                   |
| DataAdapter | 提供连接 DataSet 对象和数据源的桥梁,使用 Command 对象在数据源中执行 SQL 命令,以便将数据加载到 DataSet 中,并使对 DataSet 中数据的更改与数据源保持一致 |

DataSet 是 ADO.NET 体系结构中另一个核心组件，它专门为各种数据源的数据访问独立性而设计的，所以它可以用于多个不同的数据源、XML 数据或管理应用程序的本地数据，如内存中的数据高速缓存。DataSet 包含一个或多个 DataTable 对象的集合，这些对象由数据行、数据列以及有关 DataTable 对象中数据的主键、外键、约束和关系信息组成。它本质上是一个内存中的数据库，但从不关心它的数据是从数据库中、XML 文件中、还是从这两者中或是从其他什么地方获得。

6.2.2 ADO.NET 命名空间

针对不同的数据源，ADO.NET 提供了不同数据提供程序，但连接数据源的过程具有着类似的方式，可以使用几乎同样的代码来完成数据源连接。数据提供器类都继承自相同的基类，实现同样的接口和包含相同的方法和属性。尽管某个针对特殊数据源的提供器可能具有自己独有的特性，例如 SQL Server 的提供器能够执行 XML 查询，但用来获取和修改数据的成员是基本相同的。

.NET 主要包含如下四个数据提供程序：

- SQL Server 提供程序：用来访问 SQL Server 数据库。
- OLE DB 提供程序：用来访问所有拥有 OLE DB 驱动器的数据源。
- Oracle 提供程序：用来访问 Oracle 数据库。
- ODBC 提供程序：用来访问所有拥有 ODBC 驱动器的数据源。

此外，第三方开发者和数据库提供商也发布了他们自己的 ADO.NET 提供程序，按照与 .NET 提供数据源提供器同样的公约和同样的方式，这些 ADO.NET 提供器同样可以很方便地使用。

选择的数据源提供程序应当是针对你所选的数据源。如果不能够找到合适的数据源提供程序，可以使用 OLE DB 提供程序，这时数据源需要拥有 OLE DB 驱动器。OLE DB 技术在 ADO 中已经使用很久了，因此很多数据源（SQL Server、Oracle、Access、MySQL 等）都拥有 OLE DB 驱动器。如果数据源不包含自己的提供程序，也不拥有 OLE DB 驱动的话，就可以使用 ODBC 提供程序，这时只要拥有 ODBC 驱动即可。

ADO.NET 组件包含在 .NET 类库中的几个不同的命名空间里，表 6-2 列举了 ADO.NET 组件所在的命名空间。

表 6-2 ADO.NET 命名空间

| 命名空间                     | 描述                                                                                                          |
|--------------------------|-------------------------------------------------------------------------------------------------------------|
| System.Data              | 该命名空间提供对表示 ADO.NET 结构的类的访问                                                                                  |
| System.Data.Common       | 该命名空间包含由各种 .NET 数据提供器共享的类                                                                                   |
| System.Data.OLE DB       | 该命名空间用于 OLE DB 的 .NET 数据提供器                                                                                 |
| System.Data.SqlClient    | 该命名空间用于 SQL Server 的 .NET 数据提供器                                                                             |
| System.Data.SqlTypes     | 该命名空间为 SQL Server 2005 中的本机数据类型提供类，这些类为 .NET 公共语言运行库所提供的数据类型提供了一种更为安全和快速的替代项。使用此命名空间中的类有助于防止出现精度损失造成的类型转换错误 |
| System.Data.OracleClient | 该命名空间用于 Oracle 的 .NET 数据提供器                                                                                 |
| System.Data.Odbc         | 该命名空间用于 ODBC 的 .NET 数据提供器                                                                                   |



提示

对于不同的数据库，.NET 框架提供了不同的数据提供程序，比如对于 SQL Server 数据库，Connection 对象就是 SqlConnection，对于 Access 数据库，Connection 对象就是 OleDbConnection。尽管实际应用中数据提供程序的名称可能有所不同，但它们都具有相同的用法，只是操作的数据库不同而已。

### 6.3 连接数据库

在对数据库中的数据进行操作前，首先要建立数据库连接。在 ADO.NET 中，数据库的连接借助于 Connection 对象来完成。对于不同的数据源需要使用不同的类建立连接，Connection 对象根据不同的数据源可以分为以下几类。

- OleDbConnection: 用于对支持 OLE DB 的数据库执行连接。
- SqlConnection: 用于对 SQL Server 数据库执行连接。
- OdbcConnection: 用于支持 ODBC 的数据库执行连接。
- OracleConnection: 用于对 Oracle 数据库执行连接。

本节主要讲解 SqlConnection，其他连接与此类似。SqlConnection 对象是通过 connectionString

属性的设置来连接数据库的，连接字符串的基本格式包括一系列由分号分隔的字符串参数列表构成。SqlConnection 连接字符串常用参数如表 6-3 所示。

表 6-3 SqlConnection 连接字符串参数表

| 参数                  | 说明                                                 |
|---------------------|----------------------------------------------------|
| Data Source Server  | SQL Server 数据库服务器的名称，可以是 local、localhost，也可以是具体的名字 |
| Initial Catalog     | 数据库的名称                                             |
| Integrated Security | 决定连接是否是安全的，取值可以是 True、False 或 SSPI                 |
| User ID             | SQL Server 登录帐户                                    |
| Password            | SQL Server 帐户的登录密码                                 |

在使用 SqlConnection 来连接数据库前，要使用构造函数来初始化 SqlConnection 对象。同时，在创建连接时，需要引用 System.Data 和 System.Data.SqlClient 命名空间。创建一个数据库连接的步骤如下。

(1) 声明一个 Connection 对象。

使用构造函数来初始化 SqlConnection 对象，在创建连接时，需要引用 System.Data 和 System.Data.SqlClient 命名空间。SqlConnection 的构造函数定义如下所示。

```
1. public SqlConnection(
2. string connectionString
3.);
```

代码说明：第 1 行定义 SqlConnection 的构造函数的名称，第 2 行的参数 connectionString 指定了用于打开 SQL Server 数据库的连接。

(2) 为该对象的属性 ConnectionString 设定一个值。

一般情况下，在一个网站项目中，所有创建数据库连接的代码都使用相同的数据库连接字符串，因此，可以使用一个类的成员存储这个字符串。代码如下：

```
1. public class ConString{
2. public static string ConnectionString = "Data Source=.;Initial Catalog=WebShopping;User ID=sa;Password=585858";
3. }
```

代码说明：第 1 行声明一个类 ConString。第 2 行声明一个静态的公开成员属性 ConnectionString，用于存储公用的数据库连接字符串。其中 Data Source 设置登录 SQL Server 数据库服务器为本地机器，Initial Catalog 设置数据库的名称为 WebShopping。User ID 设置 SQL Server 登录帐户的账号，Password 设置 SQL Server 登录帐户的密码。

(3) 然后，创建数据库连接对象时候就可以采用如下代码：

```
SqlConnection connection = new SqlConnection (ConString.ConnectionString);
```

代码说明：使用 SqlConnection 的构造函数实例化一个数据库连接对象 connection。

除了在程序中声明数据库连接字符串，还可以在配置文件 Web.config 中的<connectionStrings>节点利用“键/值”对来存储数据库连接字符串，代码如下：

```

1. <configuration>
2. <connectionStrings>
3. <add name="Con" connectionString=
4. "Data Source=.;Initial Catalog=Hotel;User ID=sa;Password=585858"/>
5. </connectionStrings>
6. </configuration>

```

代码说明：在第 2 和第 5 行的<connectionStrings></connectionStrings>节点必须包含在第 1 行和第 6 行的父节点<configuration></configuration>中。第 3 行添加了一个名为“Con”的数据连接字符串对象 connectionString 来保存连接 SQL Server 2005 中 WebShopping 数据库的数据库连接字符串。

为了获取配置文件中存储的数据库连接信息，需要使用 System.Web.Configuration 命名空间下包含的静态类 WebConfigurationManager，代码如下：

```

SqlConnection connection = new SqlConnection (System.Web.Configuration.WebConfigurationManager.ConnectionStrings["Con"].
ConnectionString.ToString());

```

(4) 利用上面的方法创建一个数据库连接后，在执行任何数据库操作之前，需要打开数据库连接，示例代码如下：

```
connection.Open();
```

代码说明：调用 SqlConnection 的 Open 方法打开数据库。到这一步为止，已经成功连接到了 SQL Server 数据库。

除了使用 SqlConnection 对象连接 SQL Server 数据库之外，这里给出几个连接其他常用数据源的字符串连接代码，供读者参考。

① 连接 OLE DB 数据源的代码如下：

```

1. OLE DBConnection con=new OLE DBConnection("Provider=SQLOLEDB; Data Source=localhost;
 Integrated Security=SSPI; Initial Catalog=BookStor");
2. con.Open();

```

以上代码，第 1 行通过 OLE DBConnection 类带一个参数的构造函数实例化一个 OLE DBConnection 对象 con，其参数列表中第 1 个参数 Provider 表示程序提供的对象是 SQLOLE DB。其他的参数与连接到 SQL Server 数据源的代码相同。第 2 行调用 con 的 Open 方法打开数据库连接。

② 连接到 ODBC 数据源，代码如下：

```

1. OdbcConnection con=new OdbcConnection("Driver={SQL Server};Server={local};
 Trusted_Connection=Yes;Database=BookStor");
2. con.Open();

```

以上代码，第 1 行通过 OdbcConnection 类带一个参数的构造函数实例化一个 OdbcConnection 对象 con，其参数列表中 Driver={SQL Server}表示数据库驱动程序是 SQL Server；Server={local}表示登录数据库服务器为本地机器；Trusted\_Connection=Yes 与连接 SQL Server 数据源中的参数 Integrated Security=SSPI 是相同的含义；Database 设置数据库的名称为 BookStor。第 2 行调用 con 的 Open 方法打开数据库连接。

③ 连接 Oracle 数据源，代码如下：

```

1. OracleConnection con=new OracleConnection("Data Source=localhost; Integrated Security=yes");
2. con.Open();

```

以上代码,第 1 行通过 OracleConnection 类带一个参数的构造函数实例化一个 OracleConnection 对象 con,其参数列表设置了登录数据库服务器为本地机器;Integrated Security=yes 与 Integrated Security=SSPI 含义相同。第 2 行调用 con 的 Open 方法打开数据库连接。



提示

在设置数据连接字符串时,如果没有采用 Windows 组帐号登录 SQL Server 数据库服务器,这时需要在连接字符串中指定 User ID(uid)和 Password(pwd)。登录时 SQL Server 会将此用户 ID 和口令进行验证。

server=localhost; database=SuperMarket;User ID=sa;Password=123;

## 6.4 获取数据

成功连接到数据库后,就可以读取数据库中的表数据了。在 ADO.NET 中实现获取的功能,需要使用到两个对象:Command 和 DataReader。

### 6.4.1 Command 对象

Command 对象主要可以来对数据库发出一些命令,比如对数据库下达查询、更新和删除数据等命令,以及调用存在于数据库中的预存的程序等。Command 对象是架构于 Connection 对象之上的,所以 Command 对象是通过连接到数据源的 Connection 对象来下达命令的。常用的 SELECT、INSERT、UPDATE、DELETE 等 SQL 命令都可以在 Command 对象中创建。根据不同的数据源,Command 对象可以分以下四类。

- SqlCommand: 用于对 SQL Server 数据库执行命令。
- OLE DBCommand: 用于对支持 OLE DB 的数据库执行命令。
- OdbcCommand: 用于支持 Odbc 的数据库执行命令。
- OracleComand: 用于对 Oracle 数据库执行命令。

本小节主要讲解 SqlCommand 对象,其他 3 个对象与此类似。SqlCommand 的常用属性如表 6-4 所示。

表 6-4 SqlCommand 属性表

| 属性                     | 说明                                                                                        |
|------------------------|-------------------------------------------------------------------------------------------|
| CommandText            | 类型为 string, 命令对象包含的 SQL 语句、存储过程或表                                                         |
| CommandTimeOut         | 类型为 int, 终止执行命令并生成错误之前的等待时间                                                               |
| CommandType            | 类型为枚举类型, 有三个值: Text 值表示采用 SQL 语句、StoredProcedure 值表示使用存储过程、TableDirect 值表示要读取的表。默认值为 Text |
| Connection             | 获取 SqlConnection 实例, 使用该对象对数据库通信                                                          |
| SqlParameterCollection | 提供给命令的参数                                                                                  |

SqlCommand 的常用方法如表 6-5 所示。

表 6-5 SqlCommand 方法表

| 方法               | 说明                                                                                         |
|------------------|--------------------------------------------------------------------------------------------|
| Cancle           | 类型为 void，取消命令的执行                                                                           |
| CreateParameter  | 创建 SqlParameter 对象的实例                                                                      |
| ExecuteNonQuery  | 类型为 int，执行不返回结果的 SQL 语句，包括 INSERT、UPDATE、DEIETE、CREATE TABLE、CREATE PROCEDURE 以及不返回结果的存储过程 |
| ExecuteReader    | 类型为 SqlDataReader，执行 SELECT、TableDirect 命令或有返回结果的存储过程                                      |
| ExecuteScalar    | 类型为 Object，执行返回单个值的 SQL 语句，如 Count(*)、Sum()、Avg()等聚合函数                                     |
| ExecuteXmlReader | 类型为 XmlReader，执行返回 Xml 语句的 SELECT 语句                                                       |

可以使用构造函数生成 SqlCommand 对象，也可以使用 SqlConnection 对象的 CreateCommand() 函数生成。

SqlCommand 的构造函数如表 6-6 所示。

表 6-6 SqlCommand 构造函数表

| 构造函数                                                                    | 说明                                   |
|-------------------------------------------------------------------------|--------------------------------------|
| SqlCommand()                                                            | 不用参数创建 SqlCommand 对象                 |
| SqlCommand(string CommandText)                                          | 根据 Sql 语句创建 SqlCommand 对象            |
| SqlCommand(string CommandText, SqlConnection conn)                      | 根据 Sql 语句和数据源连接创建 SqlCommand 对象      |
| SqlCommand(string CommandText, SqlConnection conn, SqlTransaction tran) | 根据 sql 语句、数据源连接和事务对象创建 SqlCommand 对象 |

创建 SqlCommand 对象有两个方式：

① 创建一个 Command 对象，指定 SQL 命令，并设置可以利用的数据库连接，示例代码如下：

```
1. SqlCommand myCommand = new SqlCommand();
2. myCommand.Connection = connection;
3. myCommand.CommandText = "Select * from DataTable";
```

代码说明：第 1 行使用不带参数的构造函数创建 SqlCommand 对象 myCommand。第 2 行使用 myCommand 对象的 Connection 属性设置可以利用的数据库连接。第 3 行通过 myCommand 对象的 CommandText 属性设置命令类型为 Sql 的查询语句。

③ 第三种方法是在创建 Command 对象的时候，直接指定 SQL 命令和数据库连接，示例代码如下：

```
SqlCommand myCommand = new SqlCommand("Select * from DataTable", connection);
```

代码说明：第 1 行通过使用一个带两个参数的 SqlCommand 构造函数直接创建一个 SqlCommand 对象 myCommand。其中，第一个参数是 Sql 查询语句，第二个参数是数据库连接对象 connection。



SqlCommand 的 CommandText 属性除了可以设置为 SQL 语句外，还可以设置为存储过程。此时需要将其 CommandType 属性设置为 CommandType.StoredProcedure，这表示要执行的是一个存储过程。

6.4.2 DataReader 对象

DataReader 对象的作用是从数据库中检索只读、只进的数据流。所谓“只读”，是指在数据阅读器 DataReader 上不可更新、删除、增加记录，所谓“只进”是指记录的接收是顺序进行且不可后退的，数据阅读器 DataReader 接收到的数据是以数据库的记录为单位的。查询结果在查询执行时返回，并存储在客户端的网络缓冲区中，直到用户使用 DataReader 的 Read 方法对它们发出请求。使用 DataReader 可以提高应用程序的性能，原因是它只要数据可用就立即检索数据，并且（默认情况下）一次只在内存中存储一行，减少了系统开销。根据不同的数据源，可以分为以下四类。

- SqlDataReader: 用于对 SQL Server 数据库读取行的只进流的方式。
- OLE DBDataReader: 用于对支持 OLE DB 的数据库读取数据行的只进流的方法。
- OdbcDataReader: 用于支持 Odbc 的数据库读取数据行的只进流的方法。
- OracleDataReader: 用于支持 Oracle 数据库读取数据行的只进流的方法。

本小节主要讲解 SqlDataReader 对象，其他 3 个对象与此类似。SqlDataReader 的常用属性如表 6-7 所示。

表 6-7 SqlDataReader 的常用属性

| 属性              | 说明                                    |
|-----------------|---------------------------------------|
| HasMoreResult   | 表示是否有多个结果                             |
| FieldCount      | 获取当前行中的列数                             |
| HasRows         | 获取一个值，该值指示 SqlDataReader 是否包含一行或多行    |
| IsClosed        | 检索一个布尔值，该值指示是否已关闭指定的 SqlDataReader 实例 |
| Item            | 获取以本机格式表示的列的值                         |
| RecordsAffected | 获取执行 Transact-SQL 语句所更改、插入或删除的行数      |
| Connection      | 获取与 SqlDataReader 关联的 SqlConnection   |

SqlDataReader 的常用方法如表 6-8 所示。

表 6-8 SqlDataReader 常用方法

| 方法              | 说明                                            |
|-----------------|-----------------------------------------------|
| Close           | 关闭 SqlDataReader 对象                           |
| GetDataTypeName | 获取源数据类型的名称                                    |
| GetName         | 获取指定列的名称                                      |
| GetSqlValue     | 获取一个表示基础 SqlDbType 变量的 Object                 |
| GetSqlValues    | 获取当前行中的所有属性列                                  |
| IsDBNull        | 已重写。获取一个值，该值指示列中是否包含不存在的或已丢失的值                |
| NextResult      | 已重写。当读取批处理 Transact-SQL 语句的结果时，使数据读取器前进到下一个结果 |
| Read            | 已重写。使 SqlDataReader 前进到下一条记录                  |

在创建 Command 对象的一个实例之后，用户可以通过对命令调用 ExecuteReader 方法来创建

DataReader, 该方法从 Command 对象中指定的数据源检索一些行, 这时, DataReader 就会被来自数据库的记录所填充。

以 SqlDataReader 对象为例, 数据阅读器 DataReader 的定义和创建格式为:

```
SqlDataReader 数据阅读器变量名=Command 变量名.ExecuteReader();
```

以上代码中的 ExecuteReader 是命令对象 Command 的一个方法。通过这一方法可以创建一个 SqlDataReader 对象的实例。

使用 DataReader 对象的 Read 方法可从查询结果中获取行。通过向 DataReader 传递列的名称或序号引用, 可以访问返回行的每一列。不过, 为了实现最佳性能, DataReader 提供了一系列方法, 使用户能够访问其本机数据类型 (GetDateTime、GetDouble、GetGuid、GetInt32 等) 的列值。DataReader 提供未缓冲的数据流, 该数据流使过程逻辑可以有效地按顺序处理从数据源中返回的结果。由于数据不在内存中缓存, 所以在检索大量数据时, DataReader 是一种适合的选择。

如果返回的是多个结果集, DataReader 会提供 NextResult 方法来按顺序循环访问这些结果集。当 DataReader 打开时, 可以使用 GetSchemaTable 方法检索有关当前结果集的架构信息。GetSchemaTable 返回一个填充了行和列的 DataTable 对象, 这些行和列包含当前结果集的架构信息。对于结果集的每一列, DataTable 都包含一行。架构表行的每一列都映射到在结果集中返回的列的属性, 其中 ColumnName 是属性的名称, 而列的值为属性的值。

由于 DataReader 允许对数据库进行直接、高性能的访问, 它只提供对数据的只读和只向前的访问, 它返回的结果不会驻留在内存中, 并且它一次只能访问一条记录, 对服务器的内存要求较小, 而且, 只使用 DataReader 就可以显示数据。所以, 只需要显示数据的应用程序中, 如学历的录入、职称的录入、所属部门的录入、职务的录入等, 为了根据职称表、部门表等形成下拉框, 以保证录入的安全、数据有效及录入的便捷, 可以尽量使用 DataReader, 因为它将提供最佳的性能。

**【例 6-2】**本例中使用 DataReader 对象获取 BookStor 数据库 BookInfo 表的内容, 并把得到的结果显示在网页上。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 6-2”。

**02** 用鼠标双击网站根目录下的“Default.aspx.cs”文件, 编写代码如下。

```
1. String sqlconn = "Server=.; DataBase=BookStor; user id=sa;password=585858 ";
2. SqlConnection myConnection = new SqlConnection(sqlconn);
3. myConnection.Open();
4. SqlCommand myCommand = new SqlCommand("select * from BookInfo", myConnection);
5. SqlDataReader myReader;
6. myReader = myCommand.ExecuteReader();
7. Response.Write("<h3>使用 SqlReader 对象查询数据库</h3>");
8. Response.Write("<table border=1 cellspacing=0 cellpadding=2>");
9. Response.Write("<tr bgcolor=yellow>");
10. for (int i = 0; i < myReader.FieldCount; i++){
11. Response.Write("<td>" + myReader.GetName(i) + "</td>");
12. }
13. Response.Write("</tr>");
14. while (myReader.Read()){
15. Response.Write("<tr>");
16. for (int i = 0; i < myReader.FieldCount; i++){
17. Response.Write("<td>" + myReader[i].ToString() + "</td>");
```

```

18. }
19. Response.Write("</tr>");
20. }
21. Response.Write("</table>");
22. myReader.Close();
23. myConnection.Close();
24. }

```

代码说明：第 1 行设置连接字符串，服务器为本地机器，数据库为 BookStor，用户名为 sa，密码是 5858585。第 2 行创建一个 SqlConnection 对象 myConnection 并传递参数为连接字符串。第 3 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 4 行创建一个 SqlCommand 的实例 myCommand，并在参数中指定 Sql 查询语句，获得 BookInfo 表的所有数据信息。

第 5 行声明一个 SqlDataReader 对象 myReader。第 6 行通过对命令调用 ExecuteReader 方法来给 myReader 填充 BookInfo 表的内容。第 10~12 行通过 for 循环遍历 myReader 对象中所有的行，通过 GetName 方法获得 BookInfo 表各列的名称。第 14 行调用了 SqlDataReader 对象的 Read 方法，获取数据没有结束前，必须不断的调用 Read 方法，它负责前进到下一条记录。第 16~18 行同样通过 for 循环遍历 myReader 对象中所有的行，使用 SqlDataReader 对象的下标 BookInfo 表各行的数据值并显示出来。第 22 行关闭 SqlDataReader 对象，第 23 行关闭与数据库的连接，释放使用的资源。

**03** 按下“Ctrl+F5”，运行结果如图 6-16 所示。在浏览器中显示数据表“BookInfo”的全部数据。



使用SqlReader对象读取数据

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 西游记   | 吴承恩    | 人民文学出版社 |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蘅塘退士   | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |

图 6-16 运行结果



提示

DataReader 被填充时，它将先被定位到 Null 记录，直至第一次调用它的 Read 方法。这种方法与传统 ADO 逻辑中默认情况下指向记录集的第一条记录是不同的。

## 6.5 填充数据集

数据集 DataSet 是 ADO.NET 数据库组件中非常重要的一个控件，通过这个控件可以实现大多数的数据库访问和操纵功能。DataSet 做为一个实体而单独存在，并可以被视为始终断开的记录集，这点是 ADO.NET 与以前数据结构之间的最大区别。DataSet 对象常和 DataAdapter 对象配合使用，通过 DataAdapter 对象向 DataSet 中填充数据。

### 6.5.1 DataSet 对象

DataSet 对象是支持 ADO.NET 的断开式、分布式数据方案的核心对象。DataSet 对象在 ADO.NET 实现从数据库抽取数据中起到关键作用，在从数据库完成数据抽取后，DataSet 对象就是数据的存放地，它是各种数据源中的数据在计算机内存中映射成的缓存，所以有时说 DataSet 对象可以看成是一个数据容器。也有人把 DataSet 对象称为内存中的数据库，因为在 DataSet 对象可以包含很多数据表以及这些数据表之间的关系。此外，DataSet 对象在客户端实现读取、更新数据库等过程中起到了中间部件的作用。

DataSet 对象从数据源中获取数据以后就断开了与数据源之间的连接。允许在 DataSet 对象中定义数据约束和表关系，增加、删除和编辑记录，还可以对 DataSet 中的数据进行查询、统计等。当完成了各项操作以后还可以把 DataSet 对象中的数据送回数据源。

DataSet 对象的产生满足了多层分布式程序的需要，它能够在断开数据源的情况下对存放在内存中的数据进行操作，这样可以提高系统整体性能，而且有利于扩展。

创建 DataSet 对象的方式有两种，第一种方式如下代码所示。

```
DataSet dataSet = new DataSet();
```

以上这种方式是使用 DataSet 不带参的构造函数 DataSet 先建立一个空的数据集 dataSet，然后再把建立的数据表放到该数据集里。

另外一种方式则采用以下的声明形式，如下代码所示。

```
DataSet dataSet = new DataSet("表名");
```

这种方式是使用 DataSet 带参的构造函数 DataSet("表名") 先建立数据表，然后在建立包含数据表的数据集。

DataSet 对象里包含了几种类以用于数据操作，一个 DataSet 的对象可用如图 6-17 所示的模型来描述。

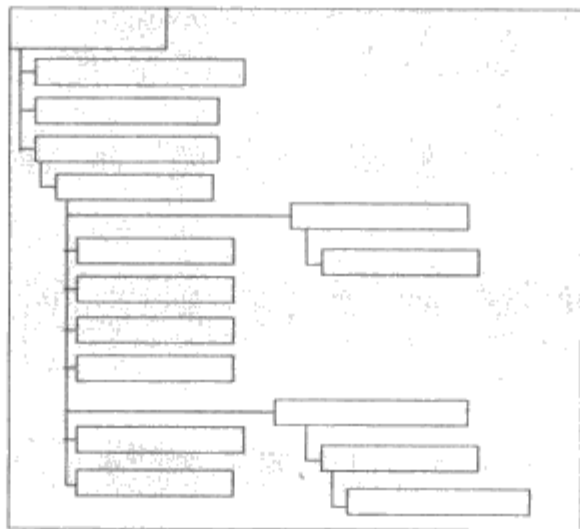


图 6-17 DataSet 对象的数据模型



提示

DataSet 中的数据必须至少存在一个主键列或唯一列。如果不存在主键列或唯一列，调用 Update 方法时会产生 InvalidOperationException 异常，而且不会生成自动更新数据库的 INSERT、UPDATE 或 DELETE 命令。

为了方便对 DataSet 对象的操作，DataSet 还提供了一系列的属性和方法，表 6-9 列举了 DataSet 的常用属性。

表 6-9 DataSet 对象常用的属性

| 属性                 | 说明                                                           |
|--------------------|--------------------------------------------------------------|
| CaseSensitive      | 获取或设置一个值，该值指示 DataTable 对象中的字符串比较是否区分大小写                     |
| DataSetName        | 获取或设置当前 DataSet 的名称                                          |
| DefaultViewManager | 获取 DataSet 所包含的数据的自定义视图，以允许使用自定义的 DataViewManager 进行筛选、搜索和导航 |
| EnforceConstraints | 获取或设置一个值，该值指示在尝试执行任何更新操作时是否遵循约束规则                            |
| ExtendedProperties | 获取与 DataSet 相关的自定义用户信息的集合                                    |
| HasErrors          | 获取一个值，指示在此 DataSet 中的任何 DataTable 对象中是否存在错误                  |
| Prefix             | 获取或设置一个 XML 前缀，该前缀是 DataSet 的命名空间的别名                         |
| Relations          | 获取用于将表链接起来并允许从父表浏览到子表的关系的集合                                  |
| Tables             | 获取包含在 DataSet 中的表的集合                                         |

DataSet 对象的常用方法如表 6-10 所示。

表 6-10 DataSet 对象常用的方法

| 方法             | 说明                                                                 |
|----------------|--------------------------------------------------------------------|
| Clear          | 通过移除所有表中的所有行来清除任何数据的 DataSet                                       |
| Copy           | 复制该 DataSet 的结构和数据                                                 |
| GetXml         | 返回存储在 DataSet 中的数据的数据的 XML 表示形式                                    |
| GetXmlSchema   | 返回存储在 DataSet 中的数据的数据的 XML 表示形式的 XML 架构                            |
| HasChanges     | 获取一个值，该值指示 DataSet 是否有更改，包括新增行、已删除的行或已修改的行                         |
| Merge          | 将指定的 DataSet、DataTable 或 DataRow 对象的数组合并到当前的 DataSet 或 DataTable 中 |
| ReadXml        | 将 XML 架构和数据读入 DataSet                                              |
| ReadXmlSchema  | 将 XML 架构读入 DataSet                                                 |
| WriteXml       | 从 DataSet 写 XML 数据，还可以选择写架构                                        |
| WriteXmlSchema | 写 XML 架构形式的 DataSet 结构                                             |

DataSet 对象中经常使用的对象包括 DataTable、DataRow、DataColumn 和 DataRelation，下面分别进行介绍。

1. DataTable 对象

DataTable 对象用于表示内存中的数据库表，可以独立地创建和使用，也可以由其他对象创建和使用。通常情况下，DataTable 对象都作为 DataSet 对象的成员存在，一个 DataSet 对象可以包含多个 DataTable。每个 DataTable 包含多个行（DataRow）和列（DataColumn）。可以通过 DataSet 对象的 Tables 属性来访问 DataSet 对象中的 DataTable 对象。

DataTable 对象也提供了很多属性和方法，表 6-11 列举了 DataTable 的常用属性，

表 6-11 DataTable 常用的属性

| 属性                 | 说明                                       |
|--------------------|------------------------------------------|
| CaseSensitive      | 获取或设置一个值，该值指示 DataTable 对象中的字符串比较是否区分大小写 |
| ChildRelations     | 获取此 DataTable 的子关系的集合                    |
| Columns            | 获取属于该表的列的集合                              |
| Constraints        | 获取或设置一个值，该值指示获取由该表维护的约束的集合               |
| DataSet            | 获取此表所属的 DataSet                          |
| DefaultView        | 获取可能包括筛选视图或游标位置的表的自定义视图                  |
| DisplayExpression  | 获取或设置一个 XML 前缀，该前缀是 DataSet 的命名空间的别名     |
| ExtendedProperties | 获取自定义用户信息的集合                             |
| HasErrors          | 获取一个值，该值指示该表所属的 DataSet 的任何表的任何行中是否有错误   |
| ParentRelations    | 获取该 DataTable 的父关系的集合                    |
| PrimaryKey         | 获取或设置充当数据表主键的列的数组                        |
| Rows               | 获取属于该表的行的集合                              |
| TableName          | 获取或设置 DataTable 的名称                      |

DataTable 的常用方法如表 6-12 所示。

表 6-12 DataTable 常用的方法

| 方法             | 说明                                          |
|----------------|---------------------------------------------|
| Clear          | 清除所有数据的 DataTable                           |
| Compute        | 计算用来传递筛选条件的当前行上的给定表达式                       |
| Copy           | 复制该 DataTable 的结构和数据                        |
| ImportRow      | 将 DataRow 复制到 DataTable 中，保留任何属性设置以及初始值和当前值 |
| LoadDataRow    | 查找和更新特定行。如果找不到任何匹配行，则使用给定值创建新行              |
| Merge          | 将指定的 DataTable 与当前的 DataTable 合并            |
| NewRow         | 创建与该表具有相同架构的新 DataRow                       |
| ReadXml        | 将 XML 架构和数据读入 DataTable                     |
| Select         | 获取 DataRow 对象的数组                            |
| WriteXml       | 将 DataTable 的当前内容以 XML 格式写入                 |
| WriteXmlSchema | 将 DataTable 的当前数据结构以 XML 架构形式写入             |

将 DataTable 对象添加到 DataSet 中的方法如下：

(1) 通过 DataTable 类的构造函数来创建，代码如下。

```
DataTable managerTable = new DataTable("Manager");
```

以上代码使用 DataTable 类带参的构造函数 DataTabel ("数据表名") 来创建一个 DataTable 类的对象 managerTable，参数就是该数据表名称 "Manager"。

(2) 将 DataTable 对象 managerTable 添加到 DataSet 对象的 Tables 集合中去，代码如下。

```
1. DataSet myds=new DataSet();
2. myds.Tables.Add(managerTable);
```

以上代码中，第 1 行使用 DataSet 类的构造函数声明一个 DataSet 对象 myds。第 2 行将创建好的 managerTable 通过 DataSet 对象 myds 属性 Tables 的 Add 方法添加到 DataSet 对象中。

(3) 创建 DataTable 后，就可以从 DataSet 中提取 DataTable 了，示例代码如下。

```
DataTable dataTable = myds.数据表名;
```

2. DataRow 对象

DataRow 对象表示数据表里的行，是给定 DataTable 中的一行数据，或者说是一条记录。DataRow 对象提供了很多属性和方法，表 6-13 列举了 DataRow 的常用属性和方法。

表 6-13 DataRow 常用的属性和方法

| 属性和方法        | 说明                   |
|--------------|----------------------|
| Item         | 获取或设置存储在指定列中的数据      |
| ItemArray    | 通过一个数组来获取或设置此行的所有值   |
| Table        | 获取该行拥有其架构的 DataTable |
| GetChildRows | 获取 DataRow 的子行       |
| IsNull       | 判断该行是否包含一个 null 值    |
| Delete       | 清除所有数据的 DataTable    |

为表添加新行，即创建 DataRow 对象，可以调用 DataTable 对象的 NewRow 方法来实现。创建的 DataRow 对象与表具有相同的结构。之后，使用 Add 方法可以将新的 DataRow 对象添加到表的 DataRow 对象集合中，示例代码如下。

```
1. DataTable table=new DataTable("student");
2. DataRow row=table.NewRow();
3. row["Name"]="Wang";
4. row["Age"]=27;
5. table.Rows.Add(row);
```

代码说明：第 1 行通过 DataTable 类的构造函数实例化一个 DataTable 的对象。表名为“student”。第 2 行使用 table 对象的 NewRow 方法给表添加了一个新的行。第 3 行给列名为“Name”的行添加一条数据。第 4 行给列名为“Age”的行添加一条数据。第 5 行使用 table 对象行集合 Rows 的 Add 方法将数据行 row 对象添加到数据表中。

3. DataColumn 对象

DataColumn 对象表示数据表里的行，是给定 DataTable 中的一列数据，每个 DataColumn 对象都有一个 DataType 属性，定义了数据表中该行的数据类型。

DataColumn 类也提供了很多属性和方法，表 6-14 列举了 DataColumn 的常用属性和方法。

表 6-14 DataColumn 常用的属性和方法

| 属性和方法        | 说明                                       |
|--------------|------------------------------------------|
| AllowDBNull  | 获取或设置一个值，该值指示 DataTable 对象中的字符串比较是否区分大小写 |
| Caption      | 获取此 DataTable 的子关系的集合                    |
| ColumnName   | 获取属于该表的列的集合                              |
| DefaultValue | 获取或设置一个值，该值指示获取由该表维护的约束的集合               |
| Table        | 获取此表所属的 DataSet                          |
| SetOrdinal   | 将 DataColumn 的序号或位置更改为指定的序号或位置           |

要添加一个列，就需要创建一个 DataColumn 对象。可以使用 DataColumn 类的构造函数来创建一个 DataColumn 对象；也可以通过调用 DataTable 的 Columns 属性的 Add 方法实现在 DataTable 内创建 DataColumn 对象。通常，Add 方法带有两个参数，分别是列名（ColumnName）和列的数据类型（DataType），示例代码如下：

```
1. DataTable dt=new DataTable(student);
2. dt.Columns.Add(new DataColumn("Name",typeof(string)));
3. dt.Columns.Add(new DataColumn("Age",typeof(int)));
```

以上代码，第 1 行通过 DataTable 类的构造函数实例化一个 DataTable 的对象，表名为“student”。第 2 行使用 table 对象数据列集合 Columns 的 Add 方法添加一个新的列，名为“Name”，该列的数据类型设置为字符串类型。第 3 行使用 table 对象数据列集合 Columns 的 Add 方法添加一个新的列，名为“Name”，该列的数据类型设置为整型。

4. DataRelation 对象

DataRelation 对象通过 DataColumn 对象将两个 DataTable 对象相互关联，表示它们之间具有约束关系。例如在数据库中消费者和订单的约束关系是：一个订单属于一个消费者。这种约束关系是在两个表的匹配列之间创建的，前提是两个列的数据类型必须相同。

DataRelation 类也提供了很多属性，表 6-15 列举了 DataRelation 的常用属性。

表 6-15 DataRelation 常用的属性

| 属性                  | 说明                                                   |
|---------------------|------------------------------------------------------|
| ChildColumns        | 获取此关系的子 DataColumn 对象                                |
| ChildKeyConstraint  | 获取关系的外键约束                                            |
| ChildTable          | 获取此关系的子表                                             |
| DataSet             | 获取 DataRelation 所属的 DataSet                          |
| ExtendedProperties  | 获取存储自定义属性的集合                                         |
| ParentColumns       | 获取作为此 DataRelation 的父列的 DataColumn 对象的数组             |
| ParentKeyConstraint | 获取聚集约束，它确保 DataRelation 的父列中的值是唯一的                   |
| ParentTable         | 获取此 DataRelation 的父级 DataTable                       |
| RelationName        | 获取或设置用于从 DataRelationCollection 中检索 DataRelation 的名称 |

在创建 `DataRelation` 时,它首先验证是否可以建立关系。在将它添加到 `DataRelationCollection` 之后,通过禁止会使关系无效的任何更改来维持此关系。在创建 `DataRelation` 并将其添加到 `DataRelationCollection` 之间的这段时间,可以对父行或子行进行其他更改。如果这样会使关系不再有效,则会生成异常。建立表之间关系的示例代码如下:

```
1. DataColumn parent= DataSet1.Tables["Customers"].Columns["CustID"];
2. DataColumn child= DataSet1.Tables["Orders"].Columns["CustID"];
3. DataRelation roc=new DataRelation("CustomersOrders",parent,child);
4. DataSet1.Relations.Add(roc);
```

以上代码,第1行通过数据集 `DataSet1` 中的 `Customers` 客户表的列 `CustID` (表示客户编号)定义父数据行对象 `parent`。第2行通过数据集 `DataSet1` 中的 `Orders` 客户表的列 `CustID` (表示客户编号)定义子数据行对象 `child`。第3行根据 `DataRelation` 类的带参构造函数实例化一个 `DataRelation` 对象 `roc`,其中第一个参数 `CustomersOrders` 是给这一表关系起的名称。第二和第三个参数分别表示父数据行和子数据行。第4行将关系通过数据集 `DataSet1` 属性 `Relations` 的 `Add` 方法添加到数据集中。

**【例 6-1】**使用 `DataColumn` 对象和 `DataRow` 对象生成一个 `DataTable` 对象数据表“Book”,并将该表的数据内容显示在页面上。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 6-1”。

**02** 用鼠标双击网站根目录下的“Default.aspx.cs”文件,编写代码如下。

```
1. DataTable dt = new DataTable("Book");
2. DataColumn dc = new DataColumn("图书编号", typeof(string));
3. DataColumn dc1 = new DataColumn("图书名称", typeof(string));
4. DataColumn dc2 = new DataColumn("作者", typeof(string));
5. DataColumn dc3 = new DataColumn("出版社", typeof(string));
6. dt.Columns.Add(dc);
7. dt.Columns.Add(dc1);
8. dt.Columns.Add(dc2);
9. dt.Columns.Add(dc3);
10. DataRow dr = dt.NewRow();
11. dr["图书编号"] = "100001";
12. dr["图书名称"] = "西游记";
13. dr["作者"] = "吴承恩";
14. dr["出版社"] = "人民文学出版社";
15. dt.Rows.Add(dr);
16. DataRow dr1 = dt.NewRow();
17. dr1["图书编号"] = "100002";
18. dr1["图书名称"] = "三国演义";
19. dr1["作者"] = "罗贯中";
20. dr1["出版社"] = "人民文学出版社";
21. dt.Rows.Add(dr1);
22. DataRow dr2 = dt.NewRow();
23. dr2["图书编号"] = "100003";
24. dr2["图书名称"] = "红楼梦";
25. dr2["作者"] = "曹雪芹";
26. dr2["出版社"] = "人民文学出版社";
27. dt.Rows.Add(dr2);
28. DataRow dr3 = dt.NewRow();
29. dr3["图书编号"] = "100004";
30. dr3["图书名称"] = "水浒传";
31. dr3["作者"] = "施耐庵";
32. dr3["出版社"] = "人民文学出版社";
```

```

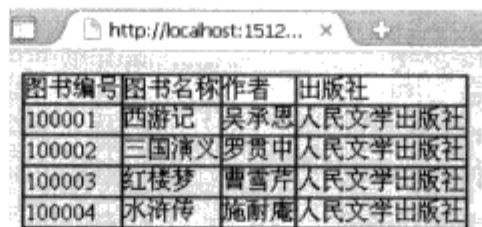
33. dt.Rows.Add(dr3);
34. Response.Write("<table bordercolor=green border=1 cellpadding=0>");
35. foreach (DataColumn col in dt.Columns){
36. Response.Write("<td bgcolor=yellow>" + col.ColumnName + "</td>");
37. }
38. foreach (DataRow myrow in dt.Rows){
39. Response.Write("<tr>");
40. foreach (DataColumn col in dt.Columns){
41. Response.Write("<td>" + myrow[col] + "</td>");
42. }
43. Response.Write("</tr>");
44. }
45. Response.Write("</table>");
46. }

```

代码说明：第 1 行实例化一个 DataTable 对象 dt 并为此数据表命名为“Boook”。第 2 行创建一个数据列 DataColumn 对象 dc，列名为“图书编号”，列的数据类型为字符串类型。第 3 行创建一个数据列 DataColumn 对象 dc1，列名为“图书名称”，列的数据类型为字符串类型。第 4 行创建一个数据列 DataColumn 对象 dc2，列名为“图书作者”，列的数据类型为字符串类型。第 5 行创建一个数据列 DataColumn 对象 dc3，列名为“出版社”，列的数据类型为字符串类型。第 6~9 行通过 DataTable 对象 dt 属性 DataColumnns 的 Add 方法将上面的 4 个列添加到数据列集合中。第 10 行创建一个数据行对象 dr，第 11~14 行给数据行添加 4 条数据。第 15 行通过 DataTable 对象 dt 属性 DataRowns 的 Add 方法将上面的数据行对象 dr 添加到数据行集合中。第 16~33 行重复第 10~15 行的操作，再向数据表中的数据行集合添加了 3 个新的数据行对象。

第 34 行为了向页面输出表格，添加一个 table 的 HTML 开始标记并设置表格的边框颜色和大小。第 35 行使用 foreach 循环遍历数据表 dt 中数据列集合的每一个数据列。第 36 行使用 DataColumn 对象 col 的属性 ColumnName 向页面输出数据列的名称并显示在表格的单元格中。第 38~44 行使用了两个 foreach 循环遍历每一个行和列中的数据内容。其中，第 1 个 foreach 循环首先遍历每一行；第 2 个 foreach 循环遍历的是每一行中的每一个单元格的数据，通过 myrow[col] 的方法输出。第 44 行添加一个表格的结束标记。

**03** 按下“Ctrl+F5”，运行结果如图 6-18 所示，在浏览器中显示数据表“Book”的全部内容。



| 图书编号   | 图书名称 | 作者  | 出版社     |
|--------|------|-----|---------|
| 100001 | 西游记  | 吴承恩 | 人民文学出版社 |
| 100002 | 三国演义 | 罗贯中 | 人民文学出版社 |
| 100003 | 红楼梦  | 曹雪芹 | 人民文学出版社 |
| 100004 | 水浒传  | 施耐庵 | 人民文学出版社 |

图 6-18 运行结果



提示

不连接的数据访问方式并不是意味着不需要连接到数据库，而连接数据库后，把数据从数据库中取出并把这些数据放入 DataSet，然后断开数据库连接，这时虽然数据库连接断开了，但仍然可以对这些数据进行操作。不过，由于数据库连接已经断开，因此对这些数据的操作将不会影响到数据库中数据的状态。

6.5.2 DataAdapter 对象

DataAdapter 对象充当数据库和 ADO.NET 对象模型中非连接对象之间的桥梁，能够用来保存和检索数据。DataAdapter 对象类的 Fill 方法用于将查询结果引入 DataSet 或 DataTable 中，以便能够脱机处理数据。

根据不同的数据源 DataAdapter 对象，可以分为以下四类。

- SqlDataAdapter: 用于对 SQL Server 数据库执行命令。
- OleDbDataAdapter: 用于对支持 OLE DB 的数据库执行命令。
- OdbcDataAdapter: 用于支持 Odbc 的数据库执行命令。
- OracleDataAdapter: 用于对 Oracle 数据库执行命令。

SqlDataAdapter 对象的常用属性如表 6-16 所示。

表 6-16 SqlDataAdapter 常用属性

| 属性            | 说明                     |
|---------------|------------------------|
| SelectCommand | 从数据源中检索记录              |
| InsertCommand | 从 DataSet 中把插入的记录写入数据源 |
| UpdateCommand | 从 DataSet 中把修改的记录写入数据源 |
| DeleteCommand | 从数据源中删除记录              |

SqlDataAdapter 对象的常用方法如表 6-17 所示。

表 6-17 SqlDataAdapter 常用方法

| 方法                                     | 说明                                                             |
|----------------------------------------|----------------------------------------------------------------|
| Fill(DataSet dataset)                  | 类型为 int, 通过添加或更新 DataSet 中的行填充一个 DataTable 对象。返回值是成功添加或更新的行的数量 |
| Fill(DataSet dataset,string datatable) | 根据 dataTable 名填充 DataSet                                       |
| Update(DataSet dataset)                | 类型为 int, 更新 DataSet 中指定表的所有已修改行。返回成功更新的行的数量                    |

可以使用构造函数生成 SqlDataAdapter 对象，SqlDataAdapter 的构造函数如表 6-18 所示。

表 6-18 SqlDataAdapter 构造函数

| 构造函数                                                     | 说明                                                      |
|----------------------------------------------------------|---------------------------------------------------------|
| SqlDataAdapter ()                                        | 不用参数创建 SqlDataAdapter 对象                                |
| SqlDataAdapter(SqlCommand cmd)                           | 根据 SqlCommand 语句创建 SqlDataAdapter 对象                    |
| SqlDataAdapter(string sqlCommandText,SqlConnection conn) | 根据 SqlCommand 语句和数据源连接创建 SqlDataAdapter 对象              |
| SqlCommand(string sqlCommandText,string sqlConnection)   | 根据 SqlCommand 语句和 sqlConnection 字符串创建 SqlDataAdapter 对象 |

使用 SqlDataAdapter 的具体步骤如下所示。

**01** 创建一个 SqlDataAdapter 对象，示例代码如下。

```
SqlDataAdapter dataAdapter = new SqlDataAdapter ();
```

以上代码，使用上表中 SqlDataAdapter 类的第一种不带参数构造函数创建了一个 SqlDataAdapter 对象 dataAdapter。

**02** 把 Command 对象定义的操作赋给以上定义的对象 dataAdapter，代码如下。

```
dataAdapter.SelectCommand = "Select * from BookInfo";
```

以上代码，通过 dataAdapter 对象的属性 SelectCommand 设置 Sql 查询语句“Select \* from BookInfo”。

**03** DataAdapter 对象将数据填入数据集时调用方法 Fill()，代码如下。

```
dataAdapter.Fill(dataset.BookInfo);
```

或者

```
dataAdapter.Fill(dataset,"BookInfo");
```

以上代码，dataAdapter 是 SqlDataAdapter 的实例，dataset 是数据集 DataSet 的实例，BookInfo 则是数据库中的数据表名。当 dataAdapter 调用 Fill() 方法时，将使用与之相关的命令组建所指定的 Select 语句从数据源中检索数据行。然后将行中的数据添加到 DataSet 对象的数据表中，如果数据表不存在，则自动创建该对象。

当执行 Select 语句时，与数据库的连接必须有效，但连接对象没有必要是打开的，在调用 Fill() 方法时会自动打开关闭的数据连接，使用完毕后再自动关闭。如果调用前该连接就处在打开状态，则操作完毕后连接仍然保持原状。

一个数据集中可以放置多张数据表，但是每个 DataAdapter 对象只能够对应于一张数据表。

**【例 6-3】** 本例中使用 DataSet 和 DataAdapter 对象填充数据的方法来访问 BookStor 数据库 BookInfo 表的内容，并把得到的结果显示在网页上。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 6-3”。

**02** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，在 Page\_Load 事件中编写代码如下。

```
1. string str = "Data Source=\\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
 Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True; User Instance=True";
2. SqlConnection myConnection = new SqlConnection(str);
3. myConnection.Open();
4. SqlCommand myCommand = new SqlCommand("select * from BookInfo", myConnection);
5. SqlDataAdapter Adapter = new SqlDataAdapter();
6. Adapter.SelectCommand = myCommand;
7. DataSet myDs = new DataSet();
8. Adapter.Fill(myDs);
9. DataTable myTable = myDs.Tables[0];
10. Response.Write("<h3>使用 DataSet 和 DataAdapter 查询数据库</h3>");
11. Response.Write("<table border=1 cellpadding=2>");
```

```

12. Response.Write("<tr bgcolor=yellow>");
13. foreach (DataColumn myColumn in myTable.Columns){
14. Response.Write("<td>" + myColumn.ColumnName + "</td>");
15. }
16. Response.Write("</tr>");
17. foreach (DataRow myRow in myTable.Rows){
18. Response.Write("<tr>");
19. foreach (DataColumn myColumn in myTable.Columns){
20. Response.Write("<td>" + myRow[myColumn] + "</td>");
21. }
22. Response.Write("</tr>");
23. }
24. Response.Write("</table>");
25. myConnection.Close();

```

代码说明：第 1 行设置连接字符串 `str`。设置连接数据库的服务器为本地机器，数据库名为 `BookStor`，连接数据库文件 `BookStor.mdf` 的路径，使用当前的 Windows 账户进行身份验证。第 2 行创建一个 `SqlConnection` 对象 `myConnection` 并传递参数为连接字符串 `str`。第 3 行通过 `SqlConnection` 对象的 `open` 方法打开数据库连接。第 4 行创建一个 `SqlCommand` 的实例 `myCommand`，并在参数中指定 `Sql` 查询语句，获得 `BookInfo` 表的所有数据信息。第 5 行实例化了一个 `SqlDataAdapter` 类型的对象 `Adapter`。第 6 行调用 `Adapter` 的属性 `SelectCommand` 获取 `Sql` 命令对象 `myCommand`。第 7 行实例化一个 `DataSet` 类型的对象 `myDs`。第 8 行调用 `Adapter` 的填充数据集的方法 `Fill`，将查询结果保存到数据集中。第 9 行通过数据集对象实例化一个 `DataTable` 对象 `dt` 来获取数据集对象 `Adapter` 表集合中第一个数据表。

第 11 行为了向页面输出表格，添加一个 `table` 的 HTML 的开始标记并设置表格的边框颜色和大小。第 13 行使用 `foreach` 循环遍历数据表 `dt` 中数据列集合中的每一个数据列。第 14 行使用 `DataColumn` 对象 `col` 的属性 `ColumnName` 向页面输出数据列的名称，并显示在表格的单元格中。第 17~23 行使用了两个 `foreach` 循环遍历每一个行和列中的数据内容。其中，第 1 个 `foreach` 循环首先遍历每一行；第 2 个 `foreach` 循环遍历的是每一行的每一个单元格中的数据，通过 `myRow[myColumn]` 的方法输出。第 24 行添加一个表格的结束标记。

第 25 行关闭数据库连接。

**03** 按下“Ctrl+F5”，运行结果如图 6-19 所示，在浏览器中显示数据表“`BookInfo`”的全部内容。

使用 `DataSet` 和 `DataAdapter` 查询数据库

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 西游记   | 吴承恩    | 人民文学出版社 |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蘅塘退士   | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |

图 6-19 运行结果



提示

使用 `DataAdapter` 对象可以实现数据的批量更新：更新完 `DataSet` 中的数据，调用 `DataAdapter` 对象的 `Update` 方法，该方法循环访问指定的 `DataTable` 中的行，检查每个 `DataRow` 是否已修改，如果修改，则调用相应的命令以更新数据库。

## 6.6 添加数据

获得了数据库的表数据后，就能进行各种访问数据库的操作。其中，向数据库添加记录的关键在于 SqlCommand 中命令对象的 Sql 语句使用的 Insert into 添加语句而不是 Select 语句。在最后还要调用 SqlCommand 对象的 ExecuteNonQuery 方法完成添加操作。

**【例 6-4】**本例向 BookInfo 表添加一条图书信息记录，该图书编号为“1000011”，图书名称为“诛仙”，作者为“萧鼎”，出版社为“花山文艺出版社”。运行程序可以看到 BookInfo 表中多了一条这样的数据。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 6-4”。
- 02 用鼠标双击网站根目录下的“Default.aspx.cs”文件，在 Page\_Load 事件中编写代码如下。

```
1. string str = "Data Source=\\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
 Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True; User Instance=True";
2. SqlConnection myConnection = new SqlConnection(str);
3. myConnection.Open();
4. string sqlstr = "insert into BookInfo values('100011','诛仙','萧鼎','花山文艺出版社)";
5. SqlCommand com = new SqlCommand(sqlstr,myConnection);
6. com.ExecuteNonQuery();
7. //以下代码和例“6-3”中第 4~25 行相同
8. //这里不再重复
```

代码说明：第 1 行设置连接字符串 str。设置连接数据库的服务器为本地机器，数据库名为 BookStor，连接数据库文件 BookStor.mdf 的路径，使用当前的 Windows 账户进行身份验证。第 2 行创建一个 SqlConnection 对象 myConnection 并传递参数为连接字符串 str。第 3 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 4 行创建一个字符串使用 insert-into 语句添加数据库数据。第 5 行创建一个 SqlCommand 的实例 com，并在参数中指定 Sql 查询语句 sqlstr 和数据库连接对象 myConnection。第 6 行调用 com 的 ExecuteNonQuery 方法完成数据库添加操作。第 7 行以下进行查询数据库的操作。

03 按下“Ctrl+F5”，运行结果如图 6-20 所示，在浏览器中显示数据表“BookInfo”添加的数据内容。

添加数据库记录

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 西游记   | 吴承恩    | 人民文学出版社 |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蘅塘退士   | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |
| 100011 | 诛仙    | 萧鼎     | 花山文艺出版社 |

添加的数据

图 6-20 运行结果

## 6.7 更新数据

更新数据库数据与添加数据库数据操作的区别是 Sql 语句不同,使用 Update-set 语句更新数据。

**【例 6-5】**本例向 BookInfo 表中更新编号为“100001”的记录,把该记录的图书名称修改为“饮马流花河”,作者修改为“萧逸”,出版社修改为“商务出版社”。运行程序可以看到 BookInfo 表中数据被修改成功。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 6-5”。

**02** 用鼠标双击网站根目录下的“Default.aspx.cs”文件,在 Page\_Load 事件中编写代码如下。

```
1. string str = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True;Connect Timeout=30;
User Instance=True";
2. SqlConnection myConnection = new SqlConnection(str);
3. myConnection.Open();
4. string sqlstr = "update BookInfo set Name='饮马流花河',Author='萧逸',Press='商务出版社'where ID='100001'";
5. SqlCommand com = new SqlCommand(sqlstr,myConnection);
6. com.ExecuteNonQuery();
7. //以下代码和例“6-3”中第 4~25 行相同
8. //这里不再重复
```

代码说明:第 1 行设置连接字符串 str。设置连接数据库的服务器为本地机器,数据库名为 BookStor,连接数据库文件 BookStor.mdf 的路径,使用当前的 Windows 账户进行身份验证。第 2 行创建一个 SqlConnection 对象 myConnection 并传递参数为连接字符串 str。第 3 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 4 行创建一个字符串使用 Update-set 语句修改数据库数据。第 5 行创建一个 SqlCommand 的实例 com,并在参数中指定 Sql 查询语句 sqlstr 和数据库连接对象 myConnection。第 6 行调用 com 的 ExecuteNonQuery 方法完成数据库添加操作。第 7 行以下进行查询数据库的操作。

**03** 按下“Ctrl+F5”,运行结果如图 6-21 所示。在浏览器中显示数据表“BookInfo”修改的数据内容。

更新数据库数据

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 饮马流花河 | 萧逸     | 商务出版社   |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蘅塘退士   | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |
| 100011 | 诛仙    | 萧鼎     | 花山文艺出版社 |

修改的数据

图 6-21 运行结果

## 6.8 删除数据

删除数据库数据比添加和更新数据库数据操作要简单,使用 Delete 语句进行删除。

**【例 6-6】**本例在 BookInfo 表中删除编号为“100011”的记录。运行程序可以看到 BookInfo 表中的这一条数据被删除。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 6-6”。

**02** 用鼠标双击网站根目录下的“Default.aspx.cs”文件,在 Page\_Load 事件中编写代码如下。

```
1. string str = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True;Connect
```

```
Timeout=30;User Instance=True";
2. SqlConnection myConnection = new SqlConnection(str);
3. myConnection.Open();
4. string sqlstr = "Delete from BookInfo where ID='100011'";
5. SqlCommand com = new SqlCommand(sqlstr, myConnection);
6. com.ExecuteNonQuery();
7. //以下代码和例“6-3”中第4~25行相同
8. //这里不再重复
```

第1行设置连接字符串 str。设置连接数据库的服务器为本地机器，数据库名为 BookStor,连接数据库文件 BookStor.mdf 的路径，使用当前的 Windows 账户进行身份验证。第2行创建一个 SqlConnection 对象 myConnection 并传递参数为连接字符串 str。第3行通过 SqlConnection 对象的 open 方法打开数据库连接。第4行创建一个字符串使用 Delete 语句删除数据库数据。第5行创建一个 SqlCommand 的实例 com，并在参数中指定 Sql 查询语句 sqlstr 和数据库连接对象 myConnection。第6行调用 com 的 ExecuteNonQuery 方法完成数据库添加操作。第7行以下的代码进行查询数据库的操作。

**03** 按下“Ctrl+F5”，运行结果如图 6-22 所示，在浏览器中显示数据表“BookInfo”删除数据后的内容。

删除数据库数据

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 饮马流花河 | 萧逸     | 商务出版社   |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 蘅塘退士   | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |

最后一条数据被删除

图 6-22 运行结果

6.9 上机题

1. 在 SQL Server 2005 中创建数据库 School，然后在该数据库中创建一个名为 Students 表，详细字段定义如表 6-19 所示。

表 6-19 表 Students 的字段

| 字段名     | 类型      | 大小  | 说明  |
|---------|---------|-----|-----|
| ID      | int     |     | 主键  |
| StuName | varchar | 50  | 学生名 |
| Phone   | varchar | 20  | 电话  |
| Address | varchar | 200 | 地址  |
| City    | varchar | 50  | 城市  |
| State   | varchar | 50  | 国家  |

2. 使用 ADO.NET 对象向上题创建的“Students”数据表中插入 6 条数据, 然后使用 SqlDataReader 对象查询该表的信息并在页面显示, 程序运行结果如图 6-23 所示。

3. 使用 DataSet 对象和 SqlDataAdapter 对象查询数据库“Students”数据表的详细信息, 程序运行结果如图 6-24 所示。

使用SqlReader对象查询Students表

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
| 2  | 李非      | 13698562314 | 海淀区清河镇  | 北京   |
| 3  | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |
| 4  | 黄韵琳     | 13965669856 | 浦东新区    | 上海   |
| 5  | 赵芳      | 13658784596 | 海淀区     | 北京   |
| 6  | 钱云      | 15986563693 | 汉阳区     | 武汉   |

图 6-23 运行结果

使用DataSet和DataAdapter查询Students表

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
| 2  | 李非      | 13698562314 | 海淀区清河镇  | 北京   |
| 3  | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |
| 4  | 黄韵琳     | 13965669856 | 浦东新区    | 上海   |
| 5  | 赵芳      | 13658784596 | 海淀区     | 北京   |
| 6  | 钱云      | 15986563693 | 汉阳区     | 武汉   |

图 6-24 运行结果

4. 使用 SqlCommand 对象把“Students”数据表中“StuName”为“李非”的记录修改为“邵飞”, 并显示修改后的记录如图 6-25 所示。

5. 删除“Students”数据表中“StuName”为“钱云”的记录, 最终数据如图 6-26 所示。

更新Students表数据

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 2  | 邵飞      | 13698562314 | 海淀区清河镇  | 北京   |

图 6-25 运行结果

删除Students表数据

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
| 2  | 邵飞      | 13698562314 | 海淀区清河镇  | 北京   |
| 3  | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |
| 4  | 黄韵琳     | 13965669856 | 浦东新区    | 上海   |
| 5  | 赵芳      | 13658784596 | 海淀区     | 北京   |

图 6-26 运行结果

6. 使用 SqlCommand 对象, 查询数据库“School”里“Students”数据表中 StuName 为“张琴”的用户信息, 运行结果如图 6-27 所示。

7. 使用 DataTable 对象中的 DataRow 对象和 DataColumn 对象以代码的方法生成数据表“Students”, 并往表中添加 2 条数据信息, 运行结果如图 6-28 所示。

查询Students表单条数据

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |

图 6-27 运行结果

Students表

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 侯华      | 13928765432 | 下关区     | 天津   |
| 2  | 周进      | 13924655432 | 汉口区     | 武汉   |

图 6-28 运行结果

8. 使用 Visual Studio 2010 中的服务器资源管理器连接第 1 题创建的“School”数据库, 浏览和操作“Students”数据表的数据, 运行结果如图 6-29 所示。

| Students: 查询(y...文件\SCHOOL.MDF) × db: Students...库文件\SCHOOL.MDF Default.aspx |      |         |             |         |      |
|------------------------------------------------------------------------------|------|---------|-------------|---------|------|
|                                                                              | ID   | StuName | Phone       | Address | City |
| ▶                                                                            | 1    | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
|                                                                              | 2    | 邵飞      | 13698562314 | 海淀区清河镇  | 北京   |
|                                                                              | 3    | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |
|                                                                              | 4    | 黄韵琳     | 13965669856 | 浦东新区    | 上海   |
|                                                                              | 5    | 赵芳      | 13658784596 | 海淀区     | 北京   |
| *                                                                            | NULL | NULL    | NULL        | NULL    | NULL |

图 6-29 服务器资源管理器

# 第 7 章 数据绑定和数据源控件

## 学习目标

数据绑定是 ASP.NET 4.0 除 ADO.NET 之外的又一种访问数据库的方法，它不仅允许开发人员可以绑定数据源，还可以绑定到简单属性、集合、表达式等，使数据的显示更加方便和高效。而各种数据源控件与数据绑定技术配合使用更是相得益彰、事半功倍，大大地提高了我们的开发效率。通过本章的学习使读者能够熟练掌握数据绑定和数据源控件的应用以实现高效率开发。

## 本章重点

- 数据绑定的基本概念和类型
- 掌握复杂数据绑定
- 熟练使用常用控件的数据绑定
- 掌握 SqlDataSource 数据源控件

## 7.1 数据绑定简介

数据绑定是 ASP.NET 4.0 提供的另外一种访问数据库的方法。与 ADO.NET 数据库访问技术不同的是：数据绑定技术可以让编程人员不必太关注数据库的连接、数据库的命令以及如何格式化等技术环节，而直接把数据绑定到服务器控件或 HTML 元素。这种读取数据的方式效率非常高，而且基本上不用写多少代码就可以实现。

数据绑定的原理是：首先要设置控件的数据源和数据的显示格式，把这些设置完毕以后，控件就会自动处理剩余的工作，然后把数据按照预定的格式显示在页面上。

ASP.NET 4.0 的数据绑定具有两种类型：简单绑定和复杂绑定。简单数据绑定将一个控件绑定到单个数据元素（如标签控件显示的值），这是用于诸如 TextBox 或 Label 之类控件（通常是只显示单个值的控件）的典型绑定类型。复杂数据绑定将一个控件绑定到多个数据元素（通常是数据库中的多个记录），复杂绑定又称为基于列表的绑定。

在 ASP.NET 4.0 中，引入了数据绑定的语法，使用该语法可以轻松地将 Web 控件的属性绑定到数据源，其语法如下：

`<%#数据源%>`

这种非常灵活的语法允许开发人员绑定到不同的数据源，可以是变量、属性、表达式、列表、数据集和视图等。

在指定了绑定数据源之后，通过调用控件的 DataBind 方法或者该控件所属父控件的 DataBind

方法来实现页面所有控件的数据绑定，从而在页面中显示出相应的绑定数据。DataBind 方法将控件及其所有的子控件绑定到 DataSource 属性指定的数据源。当在父控件上调用 DataBind 方法时，该控件及其所有的子控件都会调用 DataBind 方法。

DataBind 方法是 ASP.NET 4.0 的 Page 对象和所有 Web 控件的成员方法。由于 Page 对象是该页面上所有控件的父控件，所有在该页面上调用 DataBind 方法将会使页面中所有的数据绑定都被处理。通常情况下，Page 对象的 DataBind 方法都在 Page\_Load 事件响应函数中调用。调用方法如下。

```
1. Protected void Page_Load(object sender,EventArgs e){
2. Page.DataBind();
3. }
```

以上代码中，第 2 行调用 Page 对象的 DataBind 方法。DataBind 方法主要用于同步数据源和数据控件中数据，使得数据源中任何更改都可以在数据控件中反映出来。通常是在数据源中数据更新后才被调用。



单值绑定存在两个缺点：①数据绑定的代码和定义用户界面的代码混合在一起；②代码过于分散。正是由于这两个缺点，就很不方便对页面和代码进行管理。因此，在程序开发中尽可能少采用单值数据绑定方式绑定数据。

### 7.1.1 简单绑定

简单绑定的数据源包括变量、表达式、集合、属性等，下面结合示例进行介绍。

#### 1. 绑定到变量

绑定到变量是最为简单的数据绑定方式。它的基本语法如下：

```
<%#简单变量%>
```

**【例 7-1】**本例将介绍如何将变量设置为控件的属性，运行程序将用户的登录名和系统时间显示出来。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-1”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 2 个 Label 控件。然后切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. <h3>变量的绑定</h3>
2. 你好: <asp:Label ID="Label1" runat="server" Text="<%#Name%>"></asp:Label>

3. 登录时间: <asp:Label ID="Label2" runat="server" Text="<%#LoginTime%>"></asp:Label>
```

代码说明：第 1 行显示标题文字。第 2 行添加一个服务器标签控件 Label1。使用绑定变量的语法<%#Name%>将变量 Name 绑定到控件的文本属性 Text。第 3 行添加一个服务器标签控件 Label2。使用绑定变量的语法<% LoginTime %>将变量 LoginTime 绑定到控件的文本属性 Text。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. public string Name = "管理员";
2. public DateTime LoginTime = DateTime.Now;
3. protected void Page_Load(object sender, EventArgs e)
4. {
5. Page.DataBind();
6. }

```

代码说明：第 1 行声明字符串变量 Name 并赋值。第 2 行声明一个 DateTime 类型的变量 LoginTime 获取系统当前的时间。第 3 行定义处理页面 Page 加载事件的方法 Load。第 5 行调用页面对象 Page 的 DataBind 方法在页面中显示出绑定的数据。



图 7-1 运行结果

**04** 按下“Ctrl+F5”，运行结果如图 7-1 所示。



提示

使用变量绑定数据,在后台代码声明变量的时候,要将其设置为 public 或 protected 类型,否则将出现变量受保护级别限制的错误。

## 2. 绑定到表达式

绑定到表达式类似于绑定到变量,只是把变量替换成表达式,基本语法如下:

```
<%=表达式%>
```

**【例 7-2】** 本例将介绍如何将数据绑定至表达式。运行程序求整数 1~100 的总和及平均数。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 7-2”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 2 个 Label 控件。然后切换到“源视图”, 在编辑区中<form></form>标记之间编写如下代码。

```

1. <h3>绑定表达式</h3>
2. 求整数 1 到 100 的总和及平均数

3. 总和=<asp:Label ID="Label1" runat="server" Text="<%=sum %>"></asp:Label>

4. 平均数=<asp:Label ID="Label2" runat="server" Text="<%=sum/10 %>"></asp:Label>

```

代码说明：第 1 行显示标题文字。第 3 行添加一个服务器标签控件 Label1。使用绑定变量的语法<%=sum %>将变量 Name 绑定到控件的文本属性 Text。第 3 行添加一个服务器标签控件 Label2。使用绑定表达式的语法<%=sum/10 %>将表达式 sum/10 绑定到控件的文本属性 Text。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件, 编写代码如下。

```

1. protected int sum = 0;
2. protected void Page_Load(object sender, EventArgs e){
3. if (!IsPostBack){
4. sum = GetSum();
5. }
6. Page.DataBind();
7. }
8. protected int GetSum(){
9. for (int i = 0; i <= 100; i++){

```

```

10. sum += i;
11. }
12. return sum;
13. }

```

代码说明：第1行声明 `int` 类型的变量 `sum` 并初始化值为0。第2行定义处理页面 `Page` 加载事件的方法 `Load`。第3行判断当前加载的页面如果不是回传的页面，则第4行调用自定义方法 `GetSum` 获得计算结果。第6行调用页面对象 `Page` 的 `DataBind` 方法在页面中显示出绑定的数据。第8行自定义一个返回 `int` 类型值的 `GetSum` 方法。第9~11行通过 `for` 循环获得1到100的总和。第12行返回计算结果 `sum`。

**04** 按下“Ctrl+F5”，运行结果如图7-2所示。

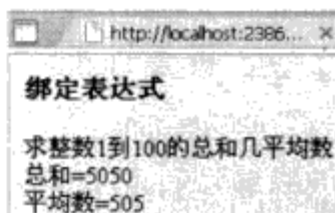


图7-2 运行结果

### 3. 绑定到集合

如果绑定的数据源是一个集合，如数组、`DataTable` 对象等，那么就要把这些数据绑定到支持多值绑定的 Web 服务器控件上。绑定到集合的基本语法如下：

```
<%#集合%>
```

**【例 7-3】** 本例将介绍如何将利用集合作为数据源绑定数据到服务器控件。运行程序求在 `DataGrid` 控件上显示一年的四季。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-3”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“源视图”，在编辑区中 `<form>` `</form>` 标记之间编写如下代码。

```
<asp:DataGrid ID="DataGrid1" runat="server" DataSource="<%#myArray %>" ></asp:DataGrid>
```

代码说明：添加一个服务器列表控件 `DataGrid1`，设置数据源属性 `DataSource` 绑定到集合的语法 `<%#myArray %>`，将 `myArray` 集合对象的数据输出在列表控件。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected ArrayList myArray = new ArrayList();
2. protected void Page_Load(object sender, EventArgs e){
3. if (!IsPostBack){
4. myArray.Add("春天: Spring");
5. myArray.Add("夏天: Summer");
6. myArray.Add("秋天: autumn");
7. myArray.Add("冬天: Winter");
8. DataGrid1.DataBind();
9. }
10. }

```

代码说明：声明一个集合类 `ArrayList` 的对象 `myArray`。第2行定义处理页面 `Page` 加载事件的方法 `Load`。第3行判断当前加载的页面如果不是回传的页面，则第4~7行分别调用 `myArray` 对象的 `Add` 方法将一年的四个季节添加到集合中。第8行调用列表控件 `DataGrid1` 的 `DataBind` 方法在页面中显示出绑定的集合中的数据。

**04** 按下“Ctrl+F5”，运行结果如图 7-3 所示。



提示

由于 ArrayList 类包含 System.Collections 命名空间之中，所以在使用 ArrayList 对象时，必须引入命名空间 System.Collections，否则会使程序出现编译错误。



图 7-3 运行结果

#### 4. 绑定到方法

有时在控件上显示数据之前需要经过复杂的逻辑处理。这时，可以通过定义方法对数据进行处理，然后把控件绑定到返回处理结果的方法上。同时根据需要可以定义无参数或带有参数的方法。绑定到方法的基本语法如下：

```
<%#方法([参数]) %>
```

**【例 7-4】**本例将介绍如何将控件的属性绑定到方法的返回值中。运行程序求一个数的绝对值。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-4”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. <h3>绑定到方法</h3>
2. <asp:DataList ID="DataList1" runat="server">
3. <ItemTemplate>
4. 数字: <%#Container.DataItem %>

5. 绝对值为: <%#AbsoluteValue((int)Container.DataItem) %>
6. </ItemTemplate>
7. </asp:DataList>
```

代码说明：第 1 行显示标题文字。第 2 行添加一个服务器列表控件 DataList1。第 3~6 行设置控件的项模板。其中，第 4 行使用绑定表达式<%#Container.DataItem %>获取控件关联的数据项。第 5 行使用绑定表达式<%#AbsoluteValue((int)Container.DataItem) %>绑定 AbsoluteValue 方法的返回值到控件。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. ArrayList myArray = new ArrayList();
4. myArray.Add(9);
5. myArray.Add(-9);
6. myArray.Add(10);
7. myArray.Add(-10);
8. DataList1.DataSource = myArray;
9. DataList1.DataBind();
10. }
11. }
12. public int AbsoluteValue(int number){
13. if (number > 0){
14. return number;
15. }
```

```

16. else if(number<0){
17. return -number;
18. }
19. else
20. return 0;
21. }

```

代码说明：第1行定义处理页面 Page 加载事件的方法 Load。第2行判断当前加载的页面如果不是回传的页面，则第3行声明一个集合类 ArrayList 的对象 myArray。第4~7行分别调用 myArray 对象的 Add 方法将数据添加到集合中。第8行使用列表控件 DataList1 的 DataSource 属性将集合对象 myArray 作为数据源。第9行调用列表控件 DataList1 的 DataBind 方法在页面中显示出绑定的集合中的数据。

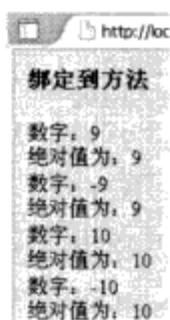


图 7-4 运行结果

**04** 按下“Ctrl+F5”，运行结果如图 7-4 所示。



提示

如果打算在表达式中使用双引号，则应当用单引号把表达式引起来。如果将控件的属性绑定到从方法调用返回的结果，该方法必须声明为 public 方法或 protected 方法。

### 7.1.2 复杂的绑定

前面我们将数据绑定到一些简单的数据源，本小节将介绍把数据绑定到复杂的数据源上的方法。在开发中常用的复杂数据源主要有 DataSet 和 DataTable。

#### 1. 绑定到 DataSet

DataSet 是 ADO.NET 的主要组件，是应用程序将从数据源中检索到的数据缓存在内存中。其包含的数据可以来自多种数据源，如数据库、XML 文档和界面输入等。

**【例 7-5】**本例将介绍如何将控件的绑定到 DataSet 对象，其中需要使用到上一章中创建的 BookStor 数据库中的 BookInfo 数据表，运行程序后将该表中数据绑定到 DataGrid 控件。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-5”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

<h3>绑定到 DataSet</h3>
<asp:DataGrid ID="DataGrid1" runat="server"></asp:DataGrid>

```

代码说明：第1行显示标题文字。第2行添加一个服务器列表控件 DataList1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. string constr = "Data Source=.;SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True; User Instance=True";
4. string str = "select * from BookInfo";

```

```

5. SqlConnection con = new SqlConnection(constr);
6. con.Open();
7. SqlDataAdapter sda = new SqlDataAdapter(str,constr);
8. DataSet ds = new DataSet();
9. sda.Fill(ds,"BookInfo");
10. DataGrid1.DataSource = ds;
11. DataGrid1.DataBind();
12. con.Close();
13. }
14. }

```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断当前加载的页面如果不是回传的页面，第 3 行设置连接字符串 constr。设置连接数据库的服务器为本地机器，数据库名为 BookStor,连接数据库文件 BookStor.mdf 的路径，使用当前的 Windows 账户进行身份验证。第 4 行 创建 Sql 语句查询的字符串 str。第 5 行创建一个 SqlConnection 对象 con 并传递参数为连接字符串 constr。第 6 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 7 行实例化了一个 SqlDataAdapter 类型的对象 sda 并将 constr 和 str 作为参数传递。第 8 行实例化一个 DataSet 类型的对象 ds。第 9 行调用 sda 的填充数据集的方法 Fill，将查询结果保存到数据集的“BookInfo”表中。第 10 行使用列表控件 DataGrid1 的 DataSource 属性将数据集对象 ds 作为数据源。第 11 行调用列表控件 DataGrid1 的 DataBind 方法在页面中显示出绑定的数据。

**04** 按下“Ctrl+F5”，运行结果如图 7-5 所示。

## 2. 绑定到 DataTable

DataTable 对象表示的是一个保存在内存中的关系型数据表格，它既可以独立创建和使用，也可以由页面上的控件将其作为数据源使用。

**【例 7-6】**本例将介绍如何利用 DataTable 将数据绑定到列表控件 DataGrid 中，显示数据表中的信息。运行程序，假设银行的利息为 3%，每月固定存款 1000 元，求 12 个月内的本息和。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-6”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. <h3>绑定到 DataSet</h3>
2. <asp:DataGrid ID="DataGrid1" runat="server" ></asp:DataGrid>

```

代码说明：第 1 行显示标题文字。第 2 行添加一个服务器列表控件 DataList1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. double interest;
2. double total;
3. protected void Page_Load(object sender, EventArgs e){
4. if (!IsPostBack){
5. DataTable dt = new DataTable();
6. DataRow dr;
7. dt.Columns.Add(new DataColumn("月份", typeof(int)));
8. dt.Columns.Add(new DataColumn("现存款数", typeof(float)));

```

绑定到DataSet

| ID     | Name  | Author | Press   |
|--------|-------|--------|---------|
| 100001 | 饮马流花河 | 萧逸     | 商务出版社   |
| 100002 | 三国演义  | 罗贯中    | 人民文学出版社 |
| 100003 | 红楼梦   | 曹雪芹    | 人民文学出版社 |
| 100004 | 水浒传   | 施耐庵    | 人民文学出版社 |
| 100005 | 唐诗三百首 | 唐虞士    | 古籍出版社   |
| 100006 | 鬼吹灯   | 天下霸唱   | 百花文艺出版社 |
| 100007 | 天机    | 蔡骏     | 上海文艺出版社 |
| 100008 | 藏地密码  | 何马     | 世纪出版社   |
| 100009 | 射雕英雄传 | 金庸     | 上海三联出版社 |
| 100010 | 倚天屠龙记 | 金庸     | 上海三联出版社 |

图 7-5 运行结果

```

9. dt.Columns.Add(new DataColumn("利息", typeof(float)));
10. dt.Columns.Add(new DataColumn("本息和", typeof(float)));
11. for (int i = 0; i <= 11; i++){
12. if (!i > 0){
13. interest = 0;
14. total = 1000;
15. }
16. else{
17. interest = (total * 0.03);
18. total = (1000 + interest + total);
19. }
20. dr = dt.NewRow();
21. dr[0] = i + 1;
22. dr[1] = 1000;
23. dr[2] = interest;
24. dr[3] = total;
25. dt.Rows.Add(dr);
26. }
27. DataGrid1.DataSource = dt;
28. DataGrid1.DataBind();
29. }
30. }

```

代码说明：第1和第2行分别声明了两个 double 类型的变量 interest 和 total。第3行定义处理页面 Page 加载事件的方法 Load。第4行判断当前加载的页面如果不是回传的页面，第5行实例化一个 DataTable 数据表类型的对象 dt。第6行声明一个 DataRow 数据行类型的对象 dr。第7~10行通过 DataTable 对象 dt 属性 DataColumn 的 Add 方法将添加四个数据列到数据表中，四个列分别是月份、现存款数、利息和本息和。第11行通过 for 循环生成 12 个月的数据。第12行判断如果月份数小于 0，第13、14行将 interest 和 total 赋值。否则第17、18行计算每月的利息和本息和。第20行通过数据表 dt 对象的 NewRow 方法实例化 dr 数据行对象。第21~24行给每个数据行的每一列进行赋值。第25行通过 DataTable 对象 dt 属性 DataRow 的 Add 方法将上面的数据行对象 dr 添加到数据行集合中。第27行将数据表对象 dt 作为列表控件 DataGrid1 的数据源。第28行调用列表控件 DataGrid1 的 DataBind 方法在页面中显示出绑定的数据。

**04** 按下“Ctrl+F5”，运行结果如图 7-6 所示。



**提示**

不使用 IsPostBack 属性会导致两个不良的后果：首先，它会影响性能，因为每次打开页面都要从数据库中获取数据。其次，如果用户从列表控件中选择了某一项，而再次将 DropDownList 重新绑定的到数据库时，那么用户之前所做的选择将无法保存下来。

绑定到数据表

| 月份 | 现存款数 | 利息       | 本息和      |
|----|------|----------|----------|
| 1  | 1000 | 0        | 1000     |
| 2  | 1000 | 30       | 2030     |
| 3  | 1000 | 60.9     | 3090.9   |
| 4  | 1000 | 92.727   | 4183.627 |
| 5  | 1000 | 125.5088 | 5309.136 |
| 6  | 1000 | 159.2741 | 6468.41  |
| 7  | 1000 | 194.0523 | 7662.462 |
| 8  | 1000 | 229.8739 | 8892.336 |
| 9  | 1000 | 266.7701 | 10159.11 |
| 10 | 1000 | 304.7732 | 11463.88 |
| 11 | 1000 | 343.9164 | 12807.8  |
| 12 | 1000 | 384.2339 | 14192.03 |

图 7-6 运行结果

### 7.1.3 常用控件的数据绑定

本小节通过几个常用 Web 服务器控件的数据绑定来学习显示多个数据值的方法。使用这些控件来显示数据的具体步骤可以概括为以下三步。

- 将用于显示数据的 Web 服务器控件添加到 ASP.NET 页面中。

- 将数据源对象赋给控件的 DataSource 属性。
- 执行控件的 DataBind 方法。

### 1. DropDownList 控件的数据绑定

DropDownList 控件是一个下拉式的菜单，其功能是可以让用户在提供的一组选项中选择单一的值。DropDownList 控件实际上是列表项的容器，这些列表项都属于 ListItem 类型。因此在编程处理列表项时，可以使用 Item 集合。当将数据源绑定到 DropDownList 控件上时，下拉列表框的事件被触发，数据就在 DropDownList 控件的下拉列表中显示出来。

**【例 7-7】** 本例实现将颜色列表通过集合的方式绑定到 DropDownList 的下拉列表中，用户可以根据自己的喜好加以选择。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-7”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 DropDownList 控件和 1 个 Label 控件。切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. <h3>DropDownList 控件的数据绑定</h3>
2. 请选择你喜欢的颜色

3. <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
4. onselectedindexchanged="DropDownList1_SelectedIndexChanged">
5. </asp:DropDownList>
6.

<asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

代码说明：第 1 行显示标题文字。第 3 行添加了一个服务器下拉列表控件 DropDownList1 并将 AutoPostBack 属性设置为 true 自动回传到服务器；同时设置控件选项改变事件 SelectedIndexChanged。第 6 行添加一个服务器标签控件 Label1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. ArrayList array = new ArrayList();
4. array.Add("红色");
5. array.Add("绿色");
6. array.Add("黄色");
7. DropDownList1.DataSource = array;
8. DropDownList1.DataBind();
9. }
10. }
11. protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e){
12. Label1.Text = "你喜欢的颜色是:" + DropDownList1.SelectedValue;
13. }
```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断当前加载的页面如果不是回传的页面，第 3 行声明一个集合类 ArrayList 的对象 array。第 4~6 行分别调用 myArray 对象的 Add 方法将数据添加到集合中。第 7 行使用下拉列表控件 DropDownList1 的 DataSource 属性将集合对象 array 作为数据源。第 8 行调用下拉列表控件 DropDownList1 的 DataBind 方法在页面中显示出绑定的集合中的数据。第 11 行定义处理下拉列表控件选中项改变事件 SelectedIndexChanged 的

方法。第12行调用下拉列表控件 DropDownList1 的 SelectedValue 属性将选中项的值显示在标签控件的文本上。

**04** 按下“Ctrl+F5”，运行结果如图7-7所示。



图 7-7 运行结果



提示

当用户选择 DropDownList 控件下拉列表中的一项时，该控件将引发一个 SelectedIndexChanged 事件。默认情况下，此事件不会导致将页发送到服务器，但可以通过设置 AutoPostBack 为 true 强制控件立即发送。因此，在使用时应注意 DropDownList 控件的 AutoPostBack 属性设置。

## 2. ListBox 控件的数据绑定

ListBox 控件允许用户从预定义的列表中选择一项或多项。它与 DropDownList 控件类似，不同之处在于它可以允许用户一次选择多项。ListBox 控件的数据绑定与 DropDownList 一样，都是通过将数据源赋给 DataSource 属性，然后再执行 DataBind 方法。

**【例 7-8】** 本例将使用上一章中创建的 BookStor 数据库里的 BookInfo 数据表，实现将该表中图书的名称绑定到 ListBox 的列表控件中。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-8”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 ListBox。切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

1. <h3>绑定到 ListBox</h3>
2. <asp:ListBox ID="ListBox1" runat="server" Height="88px"></asp:ListBox>

代码说明：第 1 行显示标题文字。第 2 行添加了一个服务器拉列表控件 ListBox1 并设置它的高度。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (! IsPostBack) {
3. string constr = "Data Source=\\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True; User Instance=True";
4. string str = "select * from BookInfo";
5. SqlConnection con = new SqlConnection(constr);
6. con.Open();
7. SqlDataAdapter sda = new SqlDataAdapter(str,constr);
8. DataSet ds = new DataSet();
9. sda.Fill(ds,"BookInfo");
10. ListBox1.DataSource =ds.Tables ["BookInfo"];
11. ListBox1.DataTextField="Name";
12. ListBox1.DataValueField ="ID";
13. ListBox1.DataBind ();
```

```

14. con.Close();
15. }
16. }

```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断当前加载的页面如果不是回传的页面，第 3 行设置连接字符串 constr。设置连接数据库的服务器为本地机器，数据库名为 BookStor，连接数据库文件 BookStor.mdf 的路径，使用当前的 Windows 账户进行身份验证。第 4 行创建 Sql 语句查询的字符串 str。第 5 行创建一个 SqlConnection 对象 con 并传递参数为连接字符串 constr。第 6 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 7 行实例化了一个 SqlDataAdapter 类型的对象 sda 并将 constr 和 str 作为参数传递。第 8 行实例化一个 DataSet 类型的对象 ds。第 9 行调用 sda 的填充数据集的方法 Fill，将查询结果保存到数据集中的“BookInfo”表中。第 10 行使用列表控件 ListBox1 的 DataSource 属性将数据集对象 ds 作为数据源。第 11 行调用列表控件 ListBox1 的 DataTextField 属性设置显示在控件中文字为数据表中的 Name 字段的值。第 11 行调用列表控件 ListBox1 的 DataValueField 属性设置对应显示在控件中文字项的值为数据表中的 ID 字段的值。第 13 行调用列表控件 ListBox1 的 DataBind 方法在页面中显示出绑定的数据。

**04** 按下“Ctrl+F5”，运行结果如图 7-8 所示。

### 3. RadioButtonList 控件的数据绑定

RadioButtonList 控件是一个单选按钮列表框控件，也即是一组单选按钮控件的集合。通过将数据绑定到 RadioButtonList 控件后，用户可以选择按钮集合中的某一个值。



图 7-8 运行结果

**【例 7-9】**本例将使用上一章中创建的 BookStor 数据库中的 BookInfo 数据表，实现选择 RadioButtonList 控件中的某一本书的名称，并且在 DataGrid 控件动态显示该书的相应信息。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-9”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 RadioButtonList 控件和 1 个 DataGrid 控件。切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```

1. <h3>绑定数据到 RadioButtonList</h3>
2. <asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True" RepeatDirection="Horizontal"
 onselectedindexchanged="RadioButtonList1_SelectedIndexChanged">
3. <asp:ListItem>三国演义</asp:ListItem>
4. <asp:ListItem>唐诗三百首</asp:ListItem>
5. <asp:ListItem>红楼梦</asp:ListItem>
6. <asp:ListItem>水浒传</asp:ListItem>
7. <asp:ListItem>射雕英雄传</asp:ListItem>
8. <asp:ListItem>鬼吹灯</asp:ListItem>
9. </asp:RadioButtonList>

10. <asp:DataGrid ID="DataGrid1" runat="server"></asp:DataGrid>

```

代码说明：第 1 行显示标题文字。第 2 行添加一个服务器单选按钮列表控件 RadioButtonList1，设置自动回传服务器，水平布局和处理控件选择项改变事件 SelectedIndexChanged。第 3~8 行给控件添加 6 个单选按钮选项。第 10 行添加一个服务器列表控件 DataGrid1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e){
2. string constr = "Data Source=\\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
 Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True; User Instance=True";
3. string str = RadioButtonList1.SelectedValue;
4. string sqlstr = "Select * from BookInfo where Name='" + str + "'";
5. SqlConnection con = new SqlConnection(constr);
6. con.Open();
7. SqlDataAdapter sda = new SqlDataAdapter(sqlstr, con);
8. DataSet ds = new DataSet();
9. sda.Fill(ds, "BookInfo");
10. DataGrid1.DataSource = ds;
11. DataGrid1.DataBind();
12. con.Close();
13. }
```

代码说明：第 1 行定义处理单选按钮列表控件 RadioButtonList1 选中项改变事件 SelectedIndexChanged 的方法。第 2 行设置连接字符串 constr。设置连接数据库的服务器为本机，数据库名为 BookStor，连接数据库文件 BookStor.mdf 的路径，使用当前的 Windows 账户进行身份验证。第 3 行获取单选按钮列表控件选中项的值。第 4 行创建 Sql 语句查询的字符串 str，查找数据表中书名为选中项值的数据信息。第 5 行创建一个 SqlConnection 对象 con 并传递参数为连接字符串 constr。第 6 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 7 行实例化了一个 SqlDataAdapter 类型的对象 sda 并将 sqlstr 和 con 作为参数传递。第 8 行实例化一个 DataSet 类型的对象 ds。第 9 行调用 sda 的填充数据集的方法 Fill，将查询结果保存到数据集中的“BookInfo”表中。第 10 行使用列表控件 DataGrid1 的 DataSource 属性将数据集对象 ds 作为数据源。第 11 行调用列表控件 DataGrid1 的 DataBind 方法在页面中显示出绑定的数据。

**04** 按下“Ctrl+F5”，运行结果如图 7-9 所示。用户选择书名，列表显示相应的图书信息。

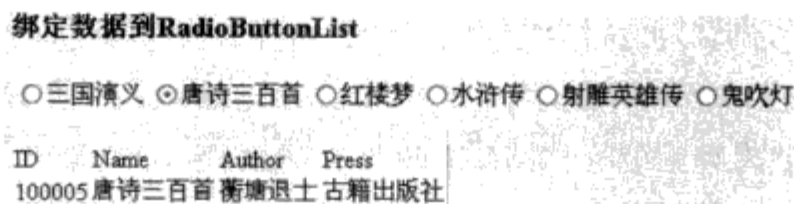


图 7-9 运行结果

## 7.2 数据源控件

数据源控件是一个为了与数据绑定控件交互而设计的服务器控件，它隐藏了人工数据绑定的复杂性。数据源控件不仅为控件提供数据，而且还支持数据绑定控件执行常见的数据操作。数据源控件不呈现任何用户界面，充当了特定数据源（如数据库、业务对象或 XML 文件）与网页上的其他数据绑定控件之间的桥梁。数据源控件实现了丰富的数据检索和修改功能，其中包括查询、排序、分页、筛选、更新、删除以及插入等功能。

ASP.NET 4.0 中包含支持不同数据绑定方案的数据源控件，这些控件可以使用不同的数据源。此外，数据源控件模型是可扩展的，因此用户还可以创建自己的数据源控件或者为现有的数据源提供附加功能，从而实现与不同数据源的交互。

数据源控件主要可以用来执行以下两种任务。

- 让数据绑定控件从数据源控件中获取数据，并把数据填充到要显示的控件中。在这种情况下，数据的获取或绑定都是自动完成的，并不需要调用方法 `DataBind` 来完成绑定。
- 利用数据源控件更新数据源。此时，数据源控件需要同复杂数据绑定控件（诸如 `GridView` 或 `DetailsView` 这类控件）一起使用。

ASP.NET 4.0 的内置数据源控件有以下数种。

- `ObjectDataSource`，用于向数据绑定控件表示数据识别中间层对象或数据接口对象。它允许绑定到一个返回数据的自定义业务对象或数据访问对象，可以在 N 层结构中存取中间层的数据。
- `SqlDataSource`，用来访问在关系型数据源，这些数据源包括 Microsoft SQL Server 和 OLE DB 以及 ODBC 数据源。它与 SQL Server 一起使用时支持高级缓存功能。当数据作为 `DataSet` 对象返回时，此控件还支持排序、筛选和分页。
- `EntityDataSource`，该控件支持基于实体数据模型（EDM）的数据绑定方案。此数据规范将数据表示为实体和关系集。它支持自动生成更新、插入、删除和选择命令以及排序、筛选和分页。
- `LinqDataSource`，通过该控件，可以在 ASP.NET 网页中使用 LINQ，从数据表或内存数据集合中检索数据。使用声明性标记，可以对数据进行检索、筛选、排序和分组操作。从 SQL Server 数据库表中检索数据时，也可以配置 `LinqDataSource` 控件来处理更新、插入和删除操作。
- `AccessDataSource`，主要用来访问 Microsoft Access 数据库。当数据作为 `DataSet` 对象返回时，支持排序、筛选和分页。
- `XmlDataSource`，主要用来访问 XML 文件，特别适用于分层的服务器控件，如 `TreeView` 或 `Menu` 控件。支持使用 XPath 表达式来实现筛选功能，并允许对数据应用 XSLT 转换。它允许通过保存更改后的整个 XML 文档来更新数据。
- `SiteMapDataSource`，该控件结合页面站点导航使用，为常用的导航控件提供数据源。
- `QueryExtend`，为简化查询中的筛选操作，ASP.NET 4.0 中增加了一个新的 `QueryExtender` 控件。可以将此控件结合 `EntityDataSource` 或 `LinqDataSource` 控件以筛选这些控件返回的数据。`QueryExtender` 控件依赖于 LINQ，但我们无需了解如何编写 LINQ 查询即可使用该查询扩展程序。

以上这些控件的使用方法大同小异，其中，`SqlDataSource` 控件是最常用的数据源控件，所以本节以该控件为例，详细介绍该控件的知识和用法，另外的几个数据源控件会在本书其他的章节中进行介绍。

### 7.2.1 SqlDataSource 控件

在 ASP.NET 页面文件中，`SqlDataSource` 控件定义的标记同其他控件一样，示例如下：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ... ></asp:SqlDataSource>
```

通过 SqlDataSource 控件, 可以使用 Web 控件访问位于某个关系数据库中的数据, 该数据库包括 Microsoft SQL Server、Oracle 数据库、OLE DB 以及 ODBC 数据源。也可以将 SqlDataSource 控件和用于显示数据的其他控件 (如 GridView、FormView 和 DetailsView 控件) 结合使用, 使用很少的代码或不使用代码就可以在 ASP.NET 网页中显示和操作数据。

SqlDataSource 控件使用 ADO.NET 类提供的程序访问数据库, 它们是:

- System.Data.SqlClient 提供程序, 用来访问 Microsoft SQL Server。
- System.Data.OleDb 提供程序, 用来以 OLE DB 的方式访问数据库。
- System.Data.Odbc 提供程序, 用来以 ODBC 的方式访问数据库。
- System.Data.OracleClient 提供程序, 用来访问 Oracle。

## 7.2.2 SqlDataSource 控件的属性

按照 SqlDataSource 控件可以实现的功能, 把其属性分为以下几类:

### 1. 执行数据库操作命令

SelectCommand、UpdateCommand、DeleteCommand 和 InsertCommand 四个属性对应数据库操作的四个命令选择、更新、删除和插入, 只需要把对应的 SQL 语句赋予这四个属性, SqlDataSource 控件即可完成对数据库的操作。

可以把带参数的 SQL 语句赋予这四个属性, 例如:

```
UpdateCommand="UPDATE [BookInfo] SET [Name] = @姓名, [Author] = @作者, [Press] = @出版社 WHERE [Name] = @姓名"
```

代码说明: 以上代码中就是把代码参数的 SQL 语句赋予 UpdateCommand 属性。其中 @姓名、@作者、@出版社为 SQL 语句的参数。

SQL 语句的参数值可以从其他控件、查询字符串中获得, 也可以通过编程方式指定参数值。参数的设置则是由属性 InsertParameters、SelectParameters、UpdateParameters 和 DeleteParameters 来进行设置。

### 2. 返回 DataSet 或 DataReader 对象

SqlDataSource 控件可以返回两种格式的数据: 作为 DataSet 对象或作为 ADO.NET 数据读取器。通过设置数据源控件的 DataSourceMode 属性, 可以指定要返回的格式。

### 3. 进行缓存

默认情况下不启用缓存。将 EnableCaching 属性设置为 true, 便可以启用缓存。

如果要使用 SqlDataSource 控件从数据库中检索数据, 需要设置以下属性:

- ProviderName, 设置为 ADO.NET 提供程序的名称, 该提供程序表示正在使用的数据库。
- ConnectionString, 设置为用于数据库的连接字符串。
- SelectCommand, 设置为从数据库中返回数据的 SQL 查询或存储过程。



提示

使用 SqlDataSource 控件，可以在 ASP.NET 页中访问和操作数据，而无需直接使用 ADO.NET 类。只需提供用于连接到数据库的连接字符串，并定义使用数据的 SQL 语句或存储过程即可。在运行时，SqlDataSource 控件会自动打开数据库连接，执行 SQL 语句或存储过程，返回选定数据，然后关闭连接。

7.2.3 SqlDataSource 控件的应用

前面我们提到 SqlDataSource 控件和用于显示数据的控件（如 GridView、FormView 和 DetailsView 控件）结合使用，能够用很少的代码或不编写代码就可以在 ASP.NET 网页中显示和操作数据。下面我们就来看一个这样的例子。

【例 7-10】本例分别使用两个 SqlDataSource 数据源控件与 1 个 DataGrid 控件和 1 个 DropDownList 控件进行数据绑定，实现用户选择下拉列表中的书名，在列表控件上显示图书信息。本例将不写一行的代码，全部使用可视化图形的操作设置各种控件的属性，体验一下 SqlDataSource 数据源控件的强大功能。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 7-10”。
- 02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 DropDownList、1 个 DataGrid 控件和 2 个 SqlDataSource 数据源控件。
- 03 将鼠标移到 DataGrid 控件上，其上方会出现如图 7-10 所示一个向右的黑色小三角。单击它，弹出“DataGrid 任务”列表。在“选择数据源”下拉列表中选中“SqlDataSource1”。
- 04 将鼠标移到 SqlDataSource1 控件上，其上方会出现如图 7-11 所示一个向右的黑色小三角，单击它，弹出“SqlDataSource 任务”列表，选择“配置数据源”选项。

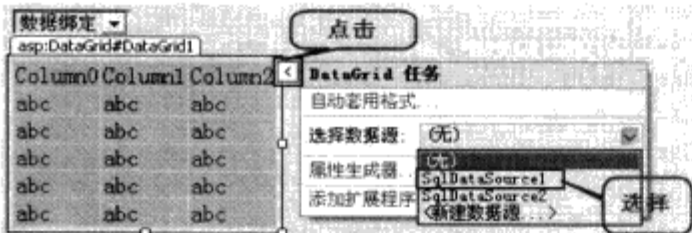


图 7-10 DataGrid 任务列表

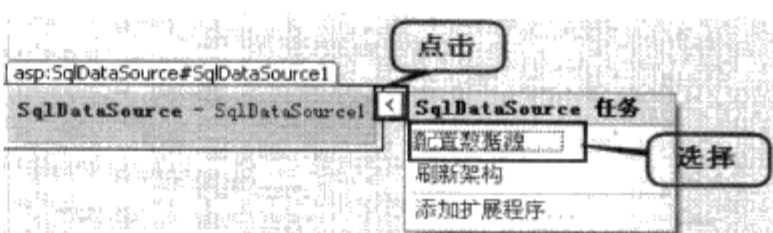


图 7-11 SqlDataSource1 任务列表

- 05 弹出如图 7-12 所示的“选择数据连接”对话框，单击“新建连接”按钮。

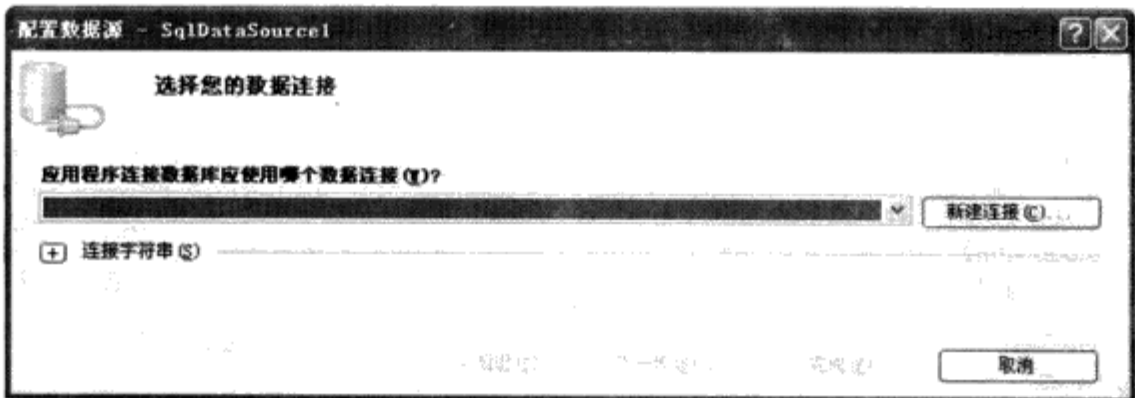


图 7-12 “选择数据连接”对话框

- 06 进入如图 7-13 所示的“添加连接”对话框，单击“浏览”按钮。

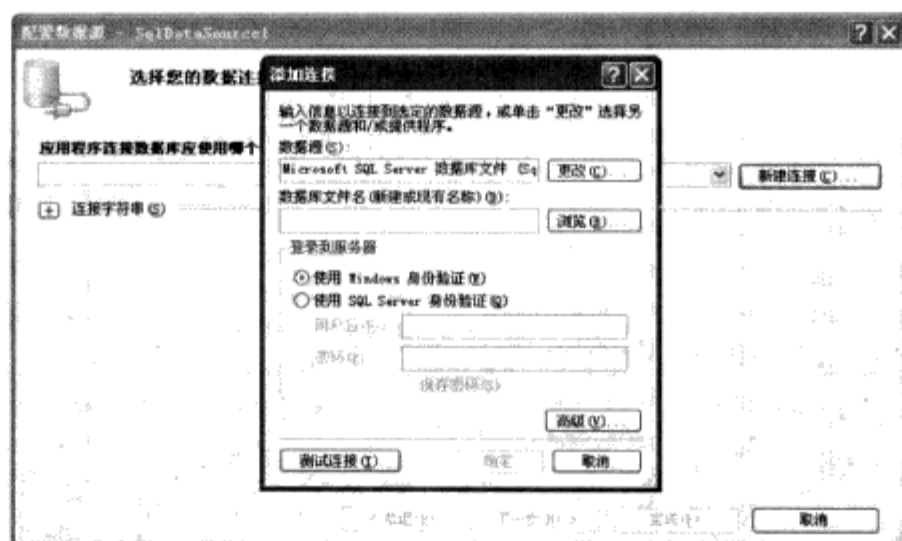


图 7-13 “添加连接”对话框

**07** 进入如图 7-14 所示的“选择 SQL Server 数据库文件”对话框，选择列表中的“BookStor.mdf”数据库文件，单击“打开”按钮。



图 7-14 选择 SQL Server 数据库文件”对话框

**08** 返回到上面图 7-13 所示的“添加连接”对话框，此时“确定”按钮变为可用，单击“确定”按钮。再单击“下一步”按钮，进入如图 7-15 所示的“保存连接字符串”对话框。选中“是，将此连接另存为”复选框，单击“下一步”按钮。

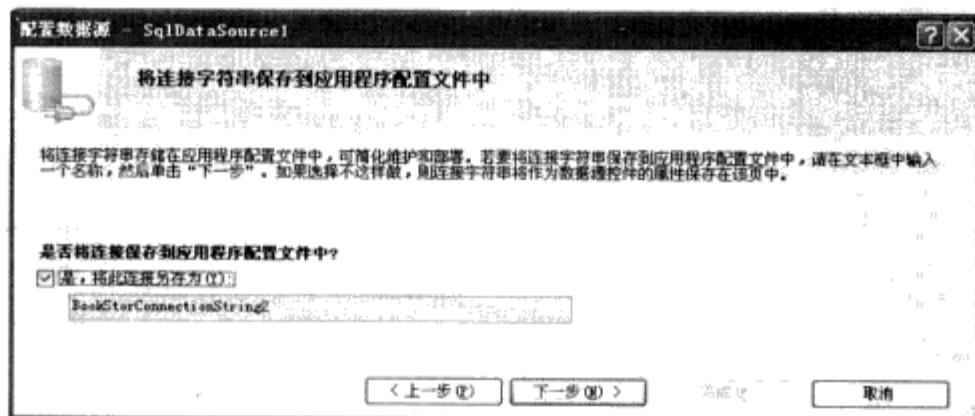


图 7-15 “保存连接字符串”对话框。

**09** 进入如图 7-16 所示的“配置 Select 语句”对话框，单击“WHERE”按钮。



图 7-16 “配置 Select 语句”对话框

**10** 弹出如图 7-17 所示的“添加 WHERE 子句”对话框。在“列”下拉列表中选择“Name”数据字段。在“源”下拉列表中选择“Control”，表示从页面控件中获取查询数据。在“参数属性”列表中的“控件 ID”下拉列表中选择 ID 为“DropDownList1”的下拉列表控件。然后单击“添加”按钮，最后单击“确定”按钮。

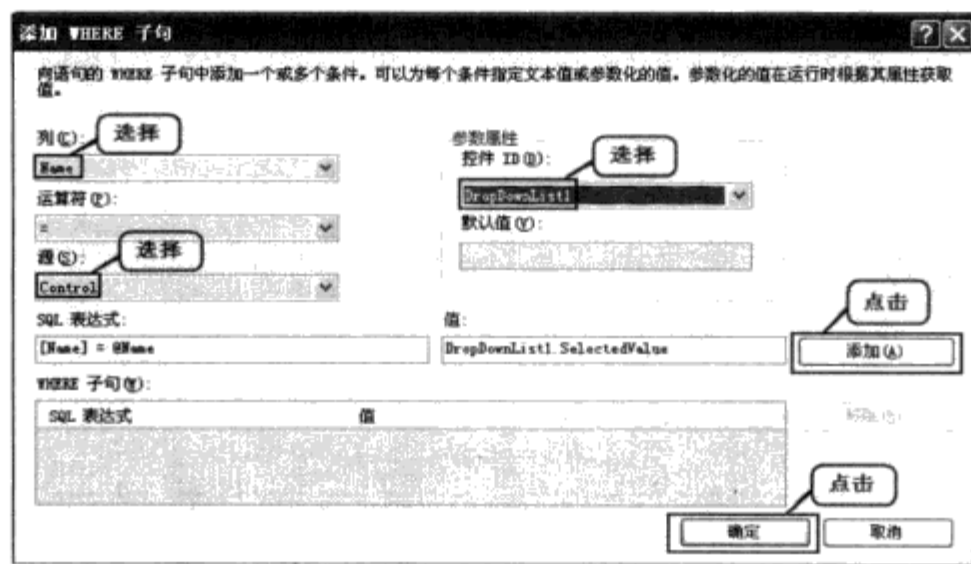


图 7-17 添加 where 子句对话框

**11** 回到图 7-16 所示“配置数据源”对话框，单击“下一步”按钮。进入如图 7-18 所示的“测试查询对话框”，单击“完成”按钮。结束 SqlDataSource2 控件的数据源配置。

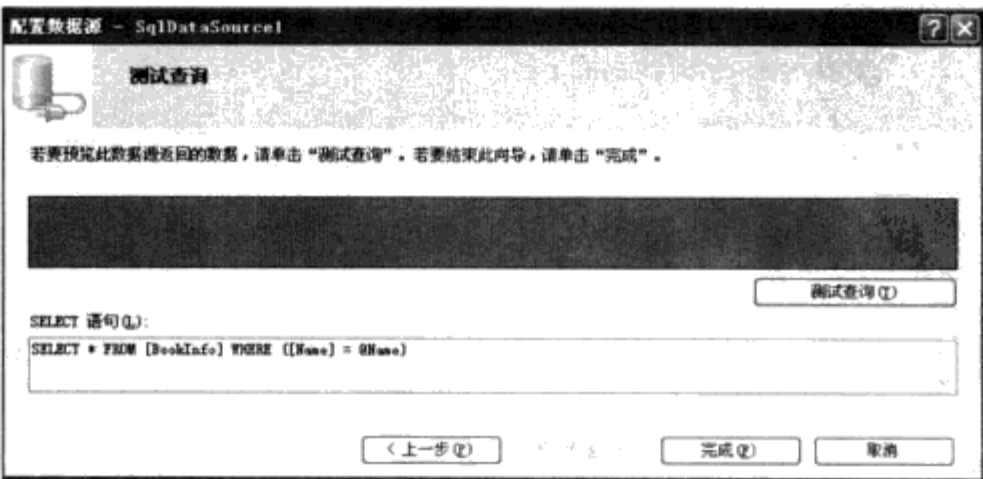


图 7-18 测试查询对话框

12 将鼠标移到 DropDownList 控件上，其上方会出现如图 7-19 所示一个向右的黑色小三角。单击它，弹出“DropDownList 任务”列表。选中“启用 AutoPostBack”，选择“选择数据源”。

13 在弹出如图 7-20 所示的“数据源配置向导”对话框。在“选择数据源”下拉列表中选中“SqlDataSource2”的控件 ID；在“选择要在 DropDownList 中显示的数据字段”的文本框中输入数据表“BookInfo”的字段“Name”，在“为 DropDownList 的值选择数据字段”的文本框中同样输入“Name”，最后单击“确定”按钮。



图 7-19 DropDownList 任务列表

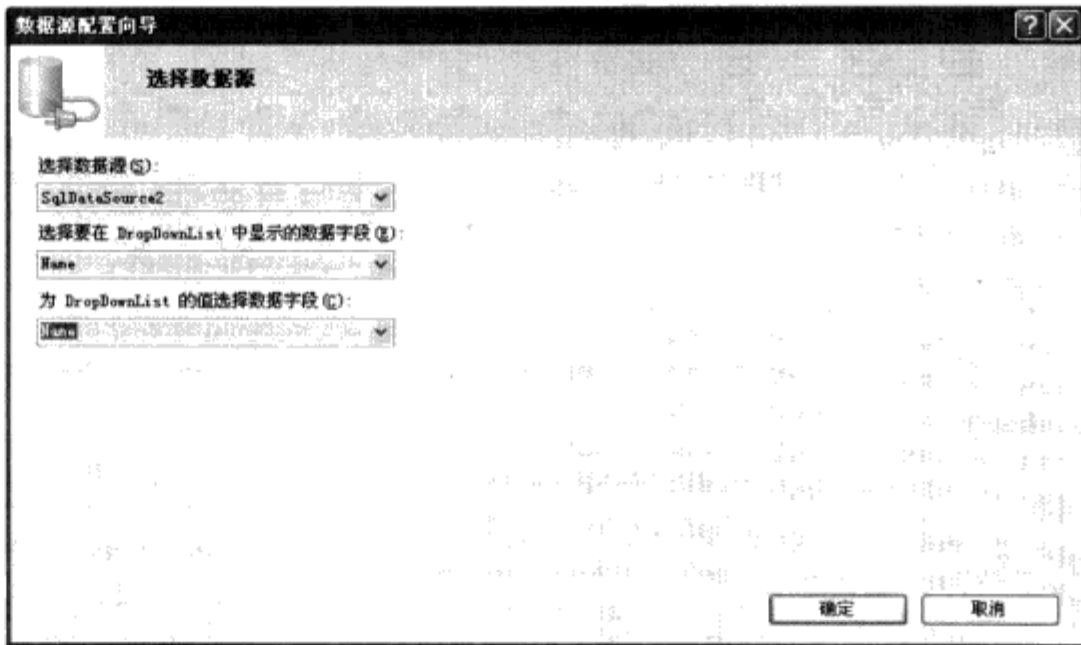


图 7-20 数据源配置向导对话框

14 接着配置开始 SqlDataSource2 控件数据源，配置过程和 SqlDataSource1 控件大致相同，区别仅在于在于查询语句设置不同。当配置过程进入如图 7-21 所示的“配置 Select 语句”对话框时，直接单击“下一步”按钮即可，因为这里在 SQL 查询时不需要使用 Where 子句。其他的配置步骤完全相同，这里不再重复说明。



图 7-21 “配置 Select 语句”对话框

15 按下“Ctrl+F5”，运行结果如图 7-22 所示。

16 当用户选择不同的下拉列表选项，列表控件显示如图 7-23 所示。



图 7-22 运行结果 1



图 7-23 运行结果



如果数据源控件与支持存储过程的数据库相连，则可以在 SQL 语句中指定存储过程的名称。通过编写代码时动态设置或在数据源控件声明时进行静态设置以实现这一功能。

### 7.3 上机题

1. 使用变量绑定的方式，将 QQ 号码、QQ 昵称和 QQ 所在城市信息显示在页面上，程序运行结果如图 7-24 所示。
2. 一个集合可以作为数据源对象，集合中包含了一年中的主要节日信息，要求将 ListBox 控件绑定至 ArrayList 集合类对象，程序运行结果如图 7-25 所示。

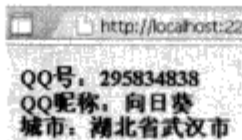


图 7-24 运行结果

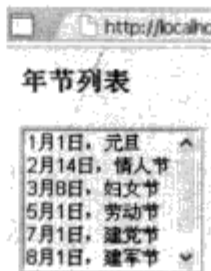


图 7-25 运行结果

3. 编写一个算术表达式，计算二数相乘的积。然后绑定该表达式到页面显示，运行结果如图 7-26 所示。
4. 通过多值绑定的方式将一周的星期数绑定到 DropDownList 控件显示，运行结果如图 7-27 所示
5. 使用上一章中创建的 BookStor 数据库的 BookInfo 数据表，选择 CheckBoxList 控件中的选中的书名，在 DataGrid 控件动态显示相应的信息，运行结果如图 7-28 所示。



图 7-26 运行结果



图 7-27 运行结果



图 7-28 运行结果

6. 利用上一章上机题 1 和 2 中创建的 School 数据库的 Students 数据表，将 DataSet 对象作为数据源获得查询到的全部表信息，最后将数据绑定到 DataGrid 控件，运行结果如图 7-29 所示。

7. 利用 DataTable 对象创建一个数据表，然后将表中的数据绑定到列表控件 DataGrid 中，显示数据表中的全部信息，运行结果如图 7-30 所示。

8. 使用数据源控件 SqlDataSource 作为 DataGrid 控件的数据源，获取 School 数据库 Students 数据表的全部内容并显示在页面，运行结果如图 7-31 所示。

Students表数据

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
| 2  | 邵飞      | 13698562314 | 海淀区清河镇  | 北京   |
| 3  | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |
| 4  | 黄韵琳     | 13965669856 | 浦东新区    | 上海   |
| 5  | 赵芳      | 13658784596 | 海淀区     | 北京   |

图 7-29 运行结果

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
| 2  | 邵飞      | 13698562314 | 海淀区清河镇  | 北京   |
| 3  | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |

图 7-30 运行结果

| ID | StuName | Phone       | Address | City |
|----|---------|-------------|---------|------|
| 1  | 张琴      | 13256789023 | 朝阳门外大街  | 北京   |
| 2  | 邵飞      | 13698562314 | 海淀区清河镇  | 北京   |
| 3  | 王宁      | 15896325689 | 徐家汇美罗城  | 上海   |
| 4  | 黄韵琳     | 13965669856 | 浦东新区    | 上海   |
| 5  | 赵芳      | 13658784596 | 海淀区     | 北京   |

图 7-31 运行结果

# 第 8 章 数据服务器控件

## 学习目标

ASP.NET 4.0 中提供了多种数据服务器控件，用于在 Web 页面中显示数据库中的表数据。与前一章中提到简单的数据绑定控件，如 DropDownList、ListBox 等不同，这些较为复杂的服务器控件具有强大的功能，开发人员只需要简单配置控件的属性，就能够在几乎不编写代码的基础上，快速实现各种布局的数据显示。通过本章的学习，读者将体会到 Visual Studio 2010 中数据服务器控件在 Web 开发中能够发挥的巨大威力。

## 本章重点

- 熟练使用 GridView 控件的常用数据操作
- 掌握 DataList 控件的模板编辑
- 使用 DetailView 控件和 GridView 控件配合实现主从数据显示
- ListView 控件的实战运用
- Chart 控件的基本使用

## 8.1 数据服务器控件简介

数据服务器控件就是能够显示数据的服务器控件，与简单格式的列表控件不同，这些控件属于比较复杂的服务器控件，不但能提供显示数据的丰富界面（可以显示多行多列数据，还可以根据用户定义来显示），还提供了修改、删除和插入数据的接口。

ASP.NET 4.0 提供的复杂数据服务器控件说明如下：

- GridView，是一个全方位的网格控件，能够显示一整张表的数据，它是 ASP.NET 4.0 中最为重要的数据控件。
- DetailsView，是用来一次显示一条记录的数据控件。
- FormView，它也是用来一次显示一条记录的数据控件，与 DetailsView 不同的是，FormView 是基于模板的，可以使布局具有灵活性。
- DataList，可用来自定义显示各行数据库信息的数据控件，显示的格式在创建的模板中定义。
- Repeater，它能生成一系列单个项，可以使用模板定义页面上单个项布局的数据控件，在页面运行时，该控件为数据源中的每个项重复相应的布局。
- ListView，该数据控件可以绑定从数据源返回的数据并显示它们，它会按照使用模板和样

式定义的格式显示数据。

由于 FormView 控件和 Repeater 控件及其相识，它们的属性和方法大多都可以通用，所以本章仅介绍其余的五个数据服务器控件。

## 8.2 GridView 控件

GridView 控件以表格式布局显示数据。默认情况下，GridView 以只读模式显示数据，但是 GridView 也能在运行时完成大部分的数据处理工作，包括添加、删除、修改、选择和排序等功能。GridView 可以以尽可能少的数据实现双向数据绑定。该控件与新的数据源控件系统紧密结合，而且只要底层的数据源对象支持，它还可以直接处理数据源的更新。除了无代码的双向数据绑定，GridView 控件还支持多个主键字段、多种字段类型以及样式和模板选项。GridView 还有一个扩展事件模型，允许我们处理或撤销事件。

### 8.2.1 GridView 控件的属性

GridView 控件支持大量的属性，这些属性包括了行为、样式、状态、模板和可视化设置几大类，表 8-1 详细描述了 GridView 控件的属性。

表 8-1 GridView 控件的属性

| 属性                       | 说明                                                               |
|--------------------------|------------------------------------------------------------------|
| AllowPaging              | 获取或设置一个值，该值指示是否启用分页功能                                            |
| AllowSorting             | 获取或设置一个值，该值指示是否启用排序功能                                            |
| AlternatingRowStyle      | 获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的交替数据行的外观         |
| AutoGenerateColumns      | 获取或设置一个值，该值指示是否为数据源中的每个字段自动创建绑定字段                                |
| AutoGenerateDeleteButton | 获取或设置一个值，该值指示每个数据行都带有“删除”按钮的 CommandField 字段列是否自动添加到 GridView 控件 |
| AutoGenerateSelectButton | 获取或设置一个值，该值指示每个数据行都带有“选择”按钮的 CommandField 字段列是否自动添加到 GridView 控件 |
| BackImageUrl             | 获取或设置要在 GridView 控件的背景中显示的图像的 URL                                |
| Caption                  | 获取或设置要在 GridView 控件的 HTML 标题元素中呈现的文本。提供此属性的目的是使辅助技术设备的用户更易于访问控件  |
| CaptionAlign             | 获取或设置 GridView 控件中的 HTML 标题元素的水平或垂直位置。提供此属性的目的是使辅助技术设备的用户更易于访问控件 |
| CellPadding              | 获取或设置单元格的内容和单元格的边框之间的空间量                                         |
| CellSpacing              | 获取或设置单元格间的空间量                                                    |
| Columns                  | 获取表示 GridView 控件中列字段的 DataControlField 对象的集合                     |

(续表)

| 属性                | 说明                                                                        |
|-------------------|---------------------------------------------------------------------------|
| DataKeyNames      | 获取或设置一个数组，该数组包含了显示在 GridView 控件中的项的主键字段的名称                                |
| DataKeys          | 获取一个 DataKey 对象集合，这些对象表示 GridView 控件中的每一行的数据键值                            |
| DataMember        | 当数据源包含多个不同的数据项列表时，获取或设置数据绑定控件绑定到的数据列表的名称                                  |
| DataSource        | 获取或设置对象，数据绑定控件从该对象中检索其数据项列表                                               |
| DataSourceID      | 获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表                                           |
| EditIndex         | 获取或设置要编辑的行的索引                                                             |
| EditRowStyle      | 获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中为进行编辑而选中的行的外观              |
| EmptyDataRowStyle | 获取对 TableItemStyle 对象的引用，使用该对象可以设置当 GridView 控件绑定到不包含任何记录的数据源时会呈现的空数据行的外观 |
| EmptyDataTemplate | 获取或设置在 GridView 控件绑定到不包含任何记录的数据源时所呈现的空数据行的用户定义内容                          |
| EmptyDataText     | 获取或设置在 GridView 控件绑定到不包含任何记录的数据源时所呈现的空数据行中显示的文本                           |
| FooterRow         | 获取表示 GridView 控件中的脚注行的 GridViewRow 对象                                     |
| FooterStyle       | 获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的脚注行的外观                    |
| GridLines         | 获取或设置 GridView 控件的网格线样式                                                   |
| HeaderRow         | 获取表示 GridView 控件中的标题行的 GridViewRow 对象                                     |
| HeaderStyle       | 获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的标题行的外观                    |
| HorizontalAlign   | 获取或设置 GridView 控件在页面上的水平对齐方式                                              |
| PageCount         | 获取在 GridView 控件中显示数据源记录所需的页数                                              |
| PageIndex         | 获取或设置当前显示页的索引                                                             |
| PagerSettings     | 获取对 PagerSettings 对象的引用，使用该对象可以设置 GridView 控件中的页导航按钮的属性                   |
| PagerStyle        | 获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的页导航行的外观                   |
| PagerTemplate     | 获取或设置 GridView 控件中页导航行的自定义内容                                              |
| PageSize          | 获取或设置 GridView 控件在每页上所显示的记录的数目                                            |
| RowHeaderColumn   | 获取或设置用作 GridView 控件的列标题的列的名称。提供此属性的目的是使辅助技术设备的用户更易于访问控件                   |

(续表)

| 属性               | 说明                                                      |
|------------------|---------------------------------------------------------|
| Rows             | 获取表示 GridView 控件中数据行的 GridViewRow 对象的集合                 |
| RowStyle         | 获取对 TableItemStyle 对象的引用, 使用该对象可以设置 GridView 控件中的数据行的外观 |
| SelectedDataKey  | 获取 DataKey 对象, 该对象包含 GridView 控件中选中行的数据键值               |
| SelectedIndex    | 获取或设置 GridView 控件中的选中行的索引                               |
| SelectedRow      | 获取对 GridViewRow 对象的引用, 该对象表示控件中的选中行                     |
| SelectedRowStyle | 获取对 TableItemStyle 对象的引用, 使用该对象可以设置 GridView 控件中的选中行的外观 |
| SelectedValue    | 获取 GridView 控件中选中行的数据键值                                 |
| ShowFooter       | 获取或设置一个值, 该值指示是否在 GridView 控件中显示脚注行                     |
| ShowHeader       | 获取或设置一个值, 该值指示是否在 GridView 控件中显示标题行                     |
| SortDirection    | 获取正在排序的列的排序方向                                           |
| SortExpression   | 获取与正在排序的列关联的排序表达式                                       |

GridView 控件提供了很多属性, 正是这些属性, 使得程序对它的操作具有了很大的灵活性, 读者现在还没有必要完全记住这些属性, 通过后面一些实例的讲解, 读者会很快明白这些属性的用法, 读者可以把这个当着参考, 需要的时候查询即可。



提示

在使用一个自定义的列集合时, 必须将 GridView 控件的 AutoGenerateColumns 属性设置为 “false”。而在使用一个自动生成的列时, 由于在控件的 Columns 集合中不存任何的数据, 所以不需要修改该属性的默认值 “true”。

### 8.2.2 GridView 控件的方法和事件

如果要使用 GridView 控件完成更高级的效果, 那么在程序中一定要使用 GridView 控件的方法和事件, 通过它们的辅助才能够更加深入地进行事件和属性的设置。GridView 控件的方法和事件比它的属性要少得多, 表 8-2 列出了 GridView 控件的常用方法。

表 8-2 GridView 控件的常用方法

| 方法                  | 说明                  |
|---------------------|---------------------|
| ApplyStyleSheetSkin | 将页样式表中定义的样式属性应用到控件  |
| DataBind            | 将数据源绑定到 GridView 控件 |
| DeleteRow           | 从数据源中删除位于指定索引位置的记录  |
| FindControl         | 在当前的命名容器中搜索指定的服务器控件 |
| Focus               | 为控件设置输入焦点           |
| GetType             | 获得当前实例的类型           |

(续表)

| 方法             | 说明                             |
|----------------|--------------------------------|
| HasControls    | 确定服务器控件是否包含任何子控件               |
| IsBindableType | 确定指定的数据类型是否能绑定到 GridView 控件中的列 |
| Sort           | 根据指定的排序表达式和方向对 GridView 控件进行排序 |
| UpdateRow      | 使用行的字段值更新位于指定行索引位置的记录          |

GridView 控件提供的方法较少，其操作主要是通过属性和在事件处理程序中添加代码来完成的。下面表 8-3 详细描述了 GridView 控件激发的事件。

表 8-3 GridView 控件的事件

| 事件                    | 说明                                           |
|-----------------------|----------------------------------------------|
| PageIndexChanged      | 在单击某一页导航按钮时，但在 GridView 控件处理分页操作之后发生         |
| PageIndexChanging     | 在单击某一页导航按钮时，但在 GridView 控件处理分页操作之前发生         |
| RowCancelingEdit      | 单击编辑模式中某一行的“取消”按钮以后，在该行退出编辑模式之前发生            |
| RowCommand            | 当单击 GridView 控件中的按钮时发生                       |
| RowCreated            | 在 GridView 控件中创建行时发生                         |
| RowDataBound          | 在 GridView 控件中将数据行绑定到数据时发生                   |
| RowDeleted            | 在单击某一行的“删除”按钮时，但在 GridView 控件删除该行之后发生        |
| RowDeleting           | 在单击某一行的“删除”按钮时，但在 GridView 控件删除该行之前发生        |
| RowEditing            | 发生在单击某一行的“编辑”按钮以后，GridView 控件进入编辑模式之前        |
| RowUpdated            | 发生在单击某一行的“更新”按钮，并且 GridView 控件对该行进行更新之后      |
| RowUpdating           | 发生在单击某一行的“更新”按钮以后，GridView 控件对该行进行更新之前       |
| SelectedIndexChanged  | 发生在单击某一行的“选择”按钮，GridView 控件对相应的选择操作进行处理之后    |
| SelectedIndexChanging | 发生在单击某一行的“选择”按钮以后，GridView 控件对相应的选择操作进行处理之前  |
| Sorted                | 在单击用于列排序的超链接时，但在 GridView 控件对相应的排序操作进行处理之后发生 |
| Sorting               | 在单击用于列排序的超链接时，但在 GridView 控件对相应的排序操作进行处理之前发生 |

虽然在大多数情况下实现 GridView 控件的大多数功能只需要编写较少的代码，但可以使用上表中提供的操作事件，极大的增强了我们的编程能力。



提示

在处理 GridView 的更新数据的事件时，在插入信息数据之前，还要检查以确保所需的值非空。这将防止与数据库的字段约束发生意外的冲突。

8.2.3 GridView 控件绑定数据

在 GridView 控件中绑定数据有两种方式：一是使用多值绑定 GridView 控件，二是使用数据源控件绑定 GridView 控件。下面通过两个例子来演示实现的过程。

**【例 8-1】**这个例子介绍如何使用多值绑定方法把数据绑定到 GridView 控件中，要显示的数据是利用 ADO.NET 从数据库“BookStor”的表“BookInfo 中读取的，读取后的数据放在 DataSet 中，然后利用多值绑定方法把数据放在 GridView 中。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 8-1”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入如图所示的“视图编辑界面”，打开“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
<h3>使用 DataSet 对象绑定 GridView</h3>
<asp:GridView ID="GridView1" runat="server"></asp:GridView>
```

代码说明：第 1 行显示标题文字。第 2 行添加一个服务器列表控件 GridView1。

**03** 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮，打开如图 8-1 所示的“GridView 任务”列表，选择“自动套用格式”。

**04** 弹出如图 8-2 所示的自动套用格式对话框。在左边的选择架构列表中有多种外观格式供我们使用，只要选中某一格式，在右边的预览窗口中会看到该格式的效果。最后，单击“确定”按钮，即可在页面中应用这一外观格式。

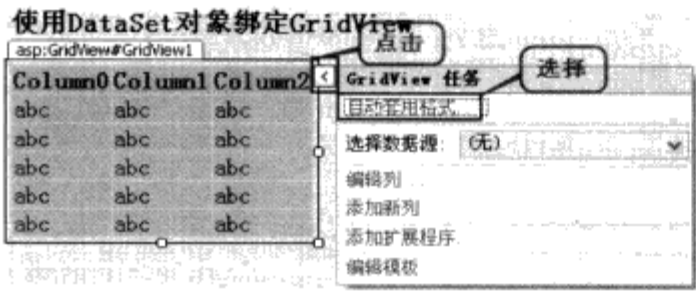


图 8-1 “GridView 任务”列表

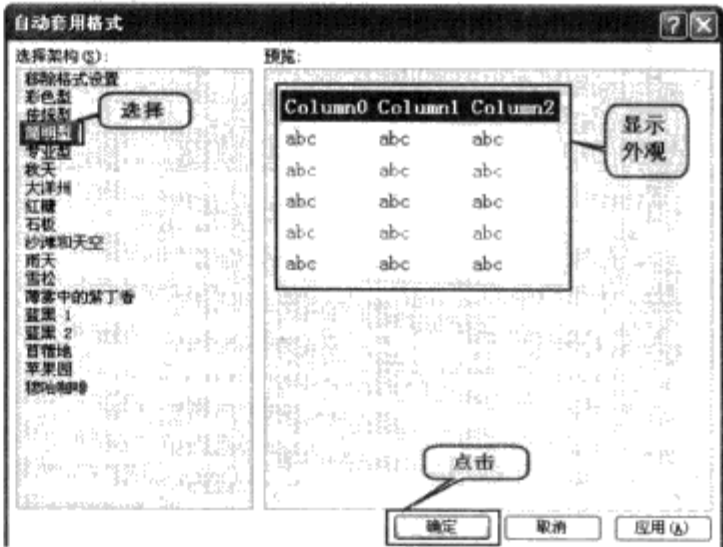


图 8-2 “自动套用格式”对话框

**05** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. string constr = "Data Source=.\SQLEXPRESS;AttachDbFilename=C:\\Program Files\\Microsoft SQL
Server\\MSSQL.1\\MSSQL\\Data\\BookStor.mdf;Integrated Security=True; User Instance=True";
4. string str = "select * from BookInfo";
5. SqlConnection con = new SqlConnection(constr);
6. con.Open();
7. SqlDataAdapter sda = new SqlDataAdapter(str, constr);
8. DataSet ds = new DataSet();
9. sda.Fill(ds, "BookInfo");
```

```
10. GridView1.DataSource = ds;
11. GridView1.DataBind();
12. con.Close();
13. }
14. }
```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断当前加载的页面是否是回传的页面，第 3 行设置连接字符串 constr，设置连接数据库的服务器为本地机器，数据库名为 BookStor，连接数据库文件 BookStor.mdf 的路径，使用当前的 Windows 账户进行身份验证。第 4 行 创建 Sql 语句查询的字符串 str。第 5 行创建一个 SqlConnection 对象 con 并传递参数为连接字符串 constr。第 6 行通过 SqlConnection 对象的 open 方法打开数据库连接。第 7 行实例化了一个 SqlDataAdapter 类型的对象 sda 并将 constr 和 str 作为参数传递。第 8 行实例化一个 DataSet 类型的对象 ds。第 9 行调用 sda 的填充数据集的方法 Fill，将查询结果保存到数据集中的“BookInfo”表中。第 10 行使用列表控件 GridView1 的 DataSource 属性将数据集对象 ds 作为数据源。第 11 行调用列表控件 GridView1 的 DataBind 方法在页面中显示出绑定的数据。

使用DataSet对象绑定GridView

ID	Name	Author	Press
100001	饮马流花河	萧逸	商务出版社
100002	三国演义	罗贯中	人民文学出版社
100003	红楼梦	曹雪芹	人民文学出版社
100004	水浒传	施耐庵	人民文学出版社
100005	唐诗三百首	蘅塘退士	古籍出版社
100006	鬼吹灯	天下霸唱	百花文艺出版社
100007	天机	蔡骏	上海文艺出版社
100008	藏地密码	何马	世纪出版社
100009	射雕英雄传	金庸	上海三联出版社
100010	倚天屠龙记	金庸	上海三联出版社

图 8-3 运行结果

06 按下“Ctrl+F5”，运行结果如图 8-3 所示。

【例 8-2】使用数据源控件 SqlDataSource 把数据库“BookStor”的表“BookInfo 中数据绑定到 GridView 控件。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 8-2”。
- 02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 GridView 控件和 1 个 SqlDataSource 数据源控件。
- 03 将鼠标移到 GridView 控件上，其上方会出现如图 8-4 所示一个向右的黑色小三角，单击它，弹出“GridView 任务”列表。在“选择数据源”下拉列表中选中“SqlDataSource1”。

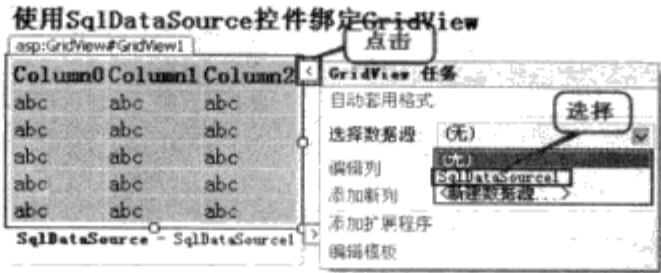


图 8-4 DataGrid 任务列表

- 04 在“GridView 任务”列表，选择“自动套用格式”，弹出自动套用格式对话框。再单击“选择架构”列表中的“彩色型”选项，然后单击“确定”按钮。
- 05 按照本书第 7 章例 7-10 中的步骤配置 SqlDataSource1 控件的数据源。切换到“源”视图，可以看到如下自动生成的 SqlDataSource1 控件的声明代码。

```
1. <asp:SqlDataSource ID="SqlDataSource1" runat="server"
2. ConnectionString="<%%$ ConnectionStrings:BookStorConnectionString %>"
3. SelectCommand="SELECT * FROM [BookInfo]"></asp:SqlDataSource>
```

代码说明：第 1 行添加了 1 个服务器数据源控件 SqlDataSource1。第 2 行设置控件的 ConnectionString 属性连接字符串对象为 BookStorConnectionString，该字符串自动在 Web.config 文件中的<connectionStrings></connectionStrings>节点中生成。第 3 行设置控件的 SelectCommand 属性为查询数据库的 sql 语句。

06 按下“Ctrl+F5”，运行结果如图 8-5 所示。



图 8-5 运行结果



提示

如果 GridView 控件没有设置任何的数据源属性，则该控件不会生成任何东西。如果绑定一个空的数据源并且规定了 EmptyDataTemplate 模板，就会给用户一个友好的显示提示结果的页面。

8.2.4 GridView 控件的列

GridView 控件中显示的列是自动生成的，默认属性 AutoGenerateColumns 为 true。但在很多情况下，GridView 控件的每一列的显示都需要定义。GridView 控件提供了几种类型的列以方便开发人员的操作，如表 8-4 所示。

表 8-4 GridView 控件的列类型

列类型	说明
BoundField	显示数据源中某个字段的值，它是 GridView 控件的默认列类型
ButtonField	为 GridView 控件中的每个项显示一个命令按钮，这样可以创建一系列自定义按钮控件
CheckBoxField	为 GridView 控件中的每一项显示一个复选框，此列类型通常用于显示具有布尔值的字段
CommandField	显示用来执行选择、编辑和删除操作的预定义命令按钮
HyperLinkField	将数据源中某个字段的值显示为超链接，此列字段类型允许将另一个字段绑定到超链接的 URL
ImagField	为 GridView 控件中的每一项显示一个图像
TemplateField	根据指定的模版为 GridView 控件中每一项显示用户定义的内容，此列类型允许创建自定义的列字段。

所有列的编辑都可以通过“字段”对话框来进行，编辑过程如下：

**01** 首先进入“字段”对话框，进入方式有两种，说明如下。

① 选中要编辑的 GridView 控件，单击右上角的小按钮，在弹出如图 8-6 所示的菜单命令中选取“编辑列...”命令。

② 选中要编辑的 GridView 控件，在“属性”窗口中找到 Columns 属性，选中该属性，单击在该属性最右边出现如图 8-7 所示的按钮。

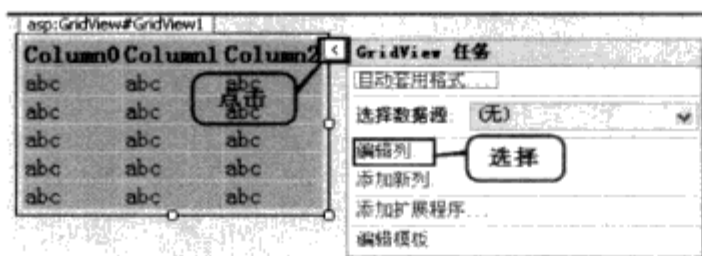


图 8-6 GridView 任务列表

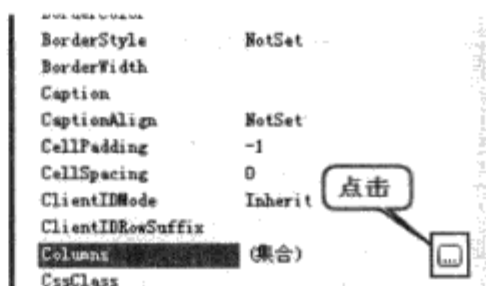


图 8-7 “属性”窗口

**02** 进入如图 8-8 所示的“字段”对话框。在“可用字段”列表中列出了 GridView 控件的列类型。当选择某一列类型后，单击“添加”按钮，即可将该列类型添加到“选定的字段”列表中。同时在右侧相应的列类型的属性列表中设置该字段的属性。

以例 8-2 来说，要在 GridView1 控件中定义四个列字段分别是：编号、书名、作者和出版社。首先选择“可用字段”列表中的“BoundField”类型，单击“添加”按钮。在“选定字段”列表中单击刚才选择的“BoundField”，然后在右侧的“BoundField 属性”列表中设置相关的属性。这里我们设置“BoundField”属性为“ID”，表示绑定的数据来自于数据库中数据表“BookInfo”中 ID 字段上的值。设置“HeaderText”属性为“编号”，表示显示在 GridView1 控件列标题显示的文字。按上面的方法依次设置其余的字段。最后单击“确定”按钮结束 GridView 控件列字段的设置。

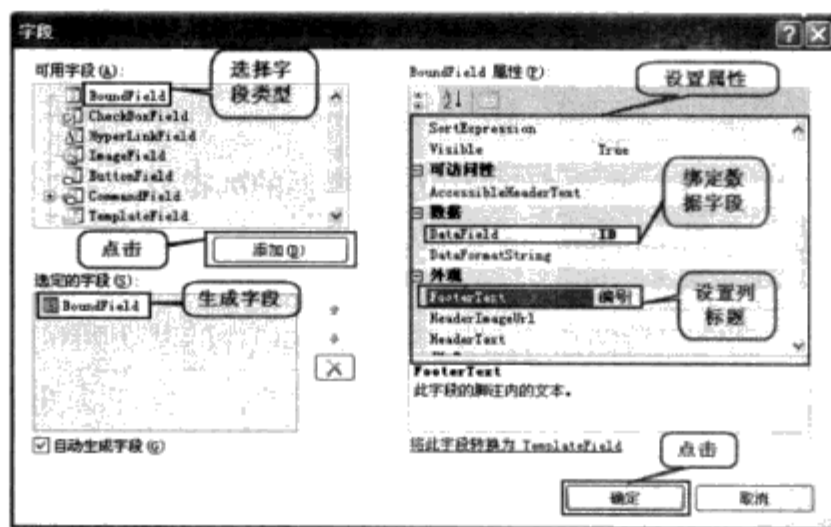


图 8-8 字段对话框

**03** 此时，GridView 控件还无法正常显示我们定义的列字段，因为还有关键的一步没有做，那就是打开如图 8-9 所示的 GridView 控件“属性”窗口，设置“AutoGenerateColumns”属性值为“false”。

至此，GridView 控件的列编辑完毕，再运行例 8-2 应用程序，会发现 GridView 控件的列标题显示了如图 8-10 中设置的中文名称。

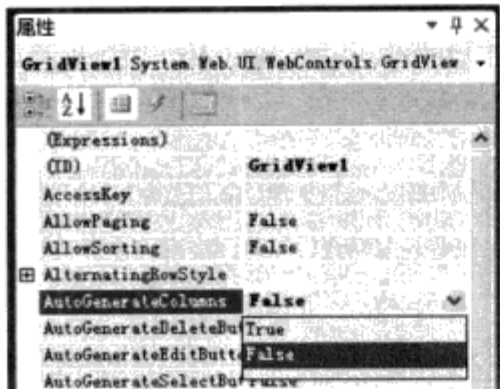


图 8-9 “属性”窗口

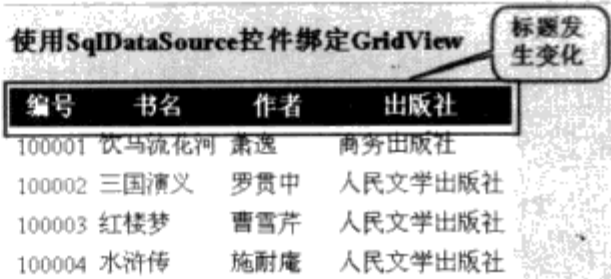


图 8-10 运行结果



BoundField 可以通过 Visible 属性以编程的方式隐藏或呈现出来，而 ReadOnly 属性防止所显示的值在编辑模式中被修改。这两个属性在实际编程中应用颇广。

8.2.5 GridView 控件的分页和排序

GridView 控件支持对所绑定的数据源中的项进行分页，只要把 AllowPaging 属性设置为“true”即可启用 GridView 控件的分页功能。当 AllowPaging 属性设置为“true”时，PagerSettings 属性允许自定义 GridView 控件的分页界面。

PagerSettings 属性对应 PagerSettings 类，它提供一些属性，这些属性支持自定义 GridView 控件自的分页界面，PagerSettings 类的属性如表 8-5 所示。

表 8-5 PagerSettings 类的属性

属性	说明
FirstPageImageUrl	获取或设置为第一页按钮显示的图像的 URL
FirstPageText	获取或设置为第一页按钮显示的文字
LastPageImageUrl	获取或设置为最后一页按钮显示的图像的 URL
LastPageText	获取或设置为最后一页按钮显示的文字
Mode	获取或设置支持分页的控件中的页导航控件的显示模式
NextPageImageUrl	获取或设置为下一页按钮显示的图像的 URL
NextPageText	获取或设置为下一页按钮显示的文字
PageButtonCount	获取或设置在 Mode 属性设置为 Numeric 或 NumericFirstLast 值时页导航中显示的页按钮的数量
Position	获取或设置一个值，该值指定页导航的显示位置
PreviousPageImageUrl	获取或设置为上一页按钮显示的图像的 URL
PreviousPageText	获取或设置为上一页按钮显示的文字
Visible	获取或设置一个值，该值指示是否在支持分页的控件中显示分页控件

在表 8-5 中，通过设置 PagerSettings 类的属性 Mode 可以指定 GridView 控件的分页模式，可用的分页模式有如下几种：

- NextPrevious，上一页按钮和下一页按钮模式。
- NextPreviousFirstLast，上一页按钮、下一页按钮、第一页按钮和最后一页按钮模式。

- Numeric, 可以直接访问页面的带编号的链接按钮模式。
- NumericFirstLast, 带编号的链接按钮、第一页链接按钮和最后一页链接按钮模式。

GridView 控件的 PagerSettings 属性的设置可以在 GridView 控件的属性窗口中进行。此外, 在设置 GridView 控件支持分页功能时, 还需要设置属性 PageSize 的值以指示在每一页中最多显示的数据条数。

当所绑定的数据源控件可以排序数据时, 只要将 GridView 控件的 AllowSorting 属性设置为“true”, 即可启用该控件中的默认排序行为。将此属性设置为“true”会使 GridView 控件将 LinkButton 控件呈现在列标题中。此外, 该控件还将每一列的 SortExpression 属性隐式设置为它所绑定到的数据字段的名称。

在运行时, 用户可以单击某列标题中的 LinkButton 控件按该列排序。单击该链接会使页面执行回发, 并引发 GridView 控件的 Sorting 事件。排序表达式(默认情况下是数据列的名称)作为事件参数的一部分传递。Sorting 事件的默认行为是 GridView 控件将排序表达式传递给数据源控件。数据源控件执行其选择查询或方法, 其中包括由网格传递的排序参数。

执行完查询后, 将引发网格的 Sorted 事件。此事件使可以执行查询后逻辑, 如显示一条状态消息等。最后, 数据源控件将 GridView 控件重新绑定到已重新排序的查询的结果。

GridView 控件不检查数据源控件是否支持排序; 在任何情况下它都会将排序表达式传递给数据源。如果数据源控件不支持排序, 并且由 GridView 控件执行排序操作, 则 GridView 控件会引发 NotSupportedException 异常。可以用 Sorting 事件的处理程序捕获此异常, 并检查数据源以确定数据源是否支持排序, 还是使用自己的排序逻辑进行排序。

在默认情况下, 当把 AllowSorting 属性设置为 true 时, GridView 控件将支持所有列可排序, 但可以通过设置列的属性 SortExpression 为空字符串, 即可禁用对这个列进行排序。

**【例 8-3】**使用数据源控件 SqlDataSource 和 GridView 读取数据库“BookStor”中数据表“BookInfo”的内容。同时实现排序和分页的功能, 并且要求每页显示 5 条数据记录。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 8-3”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 1 个 GridView 控件和 1 个 SqlDataSource 数据源控件。

**03** 将鼠标移到 GridView 控件上, 其上方会出现一个向右的黑色小三角。单击它, 弹出“GridView 任务”列表。在“选择数据源”下拉列表中选中“SqlDataSource1”。

**04** 在“GridView 任务”列表, 选择“自动套用格式”。弹出自动套用格式对话框。我们单击“选择架构”列表中的“秋天”选项, 然后单击“确定”按钮。

**05** 配置 SqlDataSource1 控件的数据源绑定“BookInfo”数据表。

**06** 用鼠标双击网站的目录下的“Default.aspx”文件, “视图编辑界面”, 打开“源视图”, 在编辑区中编写声明 GridView 控件的关键代码如下。

```
1. <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
2. AllowSorting="True" AutoGenerateColumns="False"
3. DataKeyNames="ID" DataSourceID="SqlDataSource1" PageSize="5">
4. <Columns>
5. <asp:BoundField DataField="ID" HeaderText="编号" ReadOnly="True"
```

```

6. SortExpression="ID" />
7. <asp:BoundField DataField="Name" HeaderText="书名" SortExpression="Name" />
8. <asp:BoundField DataField="Author" HeaderText="作者" SortExpression="Author" />
9. <asp:BoundField DataField="Press" HeaderText="出版社" SortExpression="Press" />
10. </Columns>
11. </asp:GridView>

```

代码说明：第 1 行添加了一个服务器列表控件 GridView1 并设置 AllowPaging 属性启用分页功能。第 2 行设置属性 AllowSorting 使得该控件可以以字段标题进行排序，同时设置 AutoGenerateColumns 为“false”禁用自动生成列的功能。第 3 行通过 DataKeyNames 属性将 ID 字段作为主键；设置 DataSourceID 属性将数据源控件 SqlDataSource1 作为 GridView 的数据源；设置 PageSize 属性为 5，表示分页后，每页显示 5 条数据记录。

第 5~8 行分别定义 GridView 控件的四个列 Name、ID、Author 和 Press，每一列的定义包括数据区域指定、列标题和排序表达式的设置。

**07** 按下“Ctrl+F5”，运行结果如图 8-11 所示。

编号	书名	作者	出版社
100010	倚天屠龙记	金庸	上海三联出版社
100009	射雕英雄传	金庸	上海三联出版社
100008	藏地密码	何马	世纪出版社
100007	天机	蔡骏	上海文艺出版社
100006	鬼吹灯	天下霸唱	百花文艺出版社

图 8-11 运行结果



如果使用的是 ObjectDataSource 控件或者不使用数据源控件，我们要实现分页功能就必须编写 GridView 的 PageIndexChanged 事件处理程序。

提示

## 8.2.6 GridView 控件的数据操作

当 GridView 控件把数据显示到页面时，有时候可能需要对这些数据进行修改或删除操作。GridView 控件通过内置的属性来提供这些操作界面，而实际的数据的操作则通过数据源控件或 ADO.NET 来实现。

有如下三种方式来启用 GridView 控件的删除或修改功能。

- 将 AutoGenerateEditButton 属性设置为 true 以启用修改，将 AutoGenerateDeleteButton 属性设置为 true 以启用删除。
- 添加一个 CommandField 列，并将其 ShowEditButton 属性设置为 true 以启用修改，将其 ShowDeleteButton 属性设置为 true 以启用删除。
- 创建一个 TemplateField，其中 ItemTemplate 包含多个命令按钮，要进行更新时可将其 CommandName 设置为“Edit”，要进行删除时可设置为“Delete”。

当启用 GridView 控件的删除或修改功能时, GridView 控件会显示一个能够让用户编辑或删除各行的用户界面: 一般情况下, 会在一系列或多列中显示按钮或链接, 用户通过单击按钮或链接把所在的行置于可编辑模式下或直接把该行删除。

在处理更改和删除的实际操作时, 有如下两种选择。

(1) 使用数据源控件。用户保存更改时, GridView 控件将更改和主键信息传递到由 DataSourceID 属性标识的数据源控件, 从而调用适当的更新操作, 例如, SqlDataSource 控件使用更改后的数据作为参数值来执行 SQL Update 语句。用户删除行时, GridView 控件将主键信息传递到由 DataSourceID 属性标识的数据源控件, 从而调用执行 SQL Delete 语句进行删除操作。

(2) 在事件处理程序中使用 ADO.NET 方法编写自动的更新或删除代码。用户保存更改时将触发事件 RowUpdated, 在该事件处理程序中获得更改后数据, 然后使用 ADO.NET 方法调用 SQL Update 语句把数据更新。用户删除行时将触发事件 RowDeleted, 在事件处理程序中获得要删除行的数据的主键, 然后使用 ADO.NET 方法调用 SQL Delete 语句把数据更新。在事件处理程序中是根据三个属性来获得 GridView 控件传递的数据的, GridView 控件的三个属性分别是 Keys 属性、NewValues 属性和 OldValues 属性。

其中, Keys 属性包含字段的名称和值, 通过它们唯一标识将要更新或删除的记录, 并始终包含键字段的原始值。若要指定哪些字段放置在 Keys 属性中, 可将 DataKeyNames 属性设置为用逗号分隔的、用于表示数据主键的字段名称的列表。DataKeys 属性会用与为 DataKeyNames 属性指定的字段关联的值自动填充。NewValues 属性包含正在编辑的行中的输入控件的当前值。OldValues 属性包含除键字段以外的任何字段的原始值, 键字段包含在 Keys 属性中。

此外, 数据源控件还可以使用 Keys、NewValues 和 OldValues 属性中的值作为更新或删除命令的参数。

**【例 8-4】** 使用 GridView 控件生成的删除列和更新列并与数据源控件一起实现数据的更新和删除操作。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 8-4”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 1 个 GridView 控件和 1 个 SqlDataSource 数据源控件。

**03** 将鼠标移到 GridView 控件上, 其上方会出现一个向右的黑色小三角。单击它, 弹出“GridView 任务”列表。在“选择数据源”下拉列表中选中“SqlDataSource1”。

**04** 在“GridView 任务”列表, 选择“自动套用格式”。弹出自动套用格式对话框。我们单击“选择架构”列表中的“秋天”选项, 然后单击“确定”按钮。

**05** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“源视图”, 在编辑区中编写声明 GridView 控件和 SqlDataSource 的关键代码如下。

```
1. <asp:GridView ID="GridView1" runat="server"
2. AutoGenerateColumns="False"
3. DataKeyNames="ID" DataSourceID="SqlDataSource1">
4. <Columns>
5. <asp:BoundField DataField="ID" HeaderText="编号" ReadOnly="True" SortExpression="ID" />
```

```

6. <asp:BoundField DataField="Name" HeaderText="书名" SortExpression="Name" />
7. <asp:BoundField DataField="Author" HeaderText="作者" SortExpression="Author" />
8. <asp:BoundField DataField="Press" HeaderText="出版社" SortExpression="Press" />
9. <asp:CommandField HeaderText="操作" ShowEditButton="True" />
10. <asp:CommandField HeaderText="操作" ShowDeleteButton="True" />
11. </Columns>
12. </asp:GridView>
13. <asp:SqlDataSource ID="SqlDataSource1" runat="server"
14. ConnectionString="<%$ ConnectionStrings:BookStorConnectionString %>"
15. DeleteCommand="DELETE FROM [BookInfo] WHERE [ID] = @original_ID AND [Name] = @original_Name AND
[Author] = @original_Author AND [Press] = @original_Press"
16. InsertCommand="INSERT INTO [BookInfo] ([ID], [Name], [Author], [Press]) VALUES (@ID, @Name, @Author, @Press)"
17. SelectCommand="SELECT * FROM [BookInfo]"
18. UpdateCommand="UPDATE [BookInfo] SET [Name] = @Name, [Author] = @Author, [Press] = @Press WHERE [ID] =
@original_ID AND [Name] = @original_Name AND [Author] = @original_Author AND [Press] = @original_Press">
19. <DeleteParameters>
20. <asp:Parameter Name="original_ID" Type="String" />
21. <asp:Parameter Name="original_Name" Type="String" />
22. <asp:Parameter Name="original_Author" Type="String" />
23. <asp:Parameter Name="original_Press" Type="String" />
24. </DeleteParameters>
25. <InsertParameters>
26. <asp:Parameter Name="ID" Type="String" />
27. <asp:Parameter Name="Name" Type="String" />
28. <asp:Parameter Name="Author" Type="String" />
29. <asp:Parameter Name="Press" Type="String" />
30. </InsertParameters>
31. <UpdateParameters>
32. <asp:Parameter Name="Name" Type="String" />
33. <asp:Parameter Name="Author" Type="String" />
34. <asp:Parameter Name="Press" Type="String" />
35. <asp:Parameter Name="original_ID" Type="String" />
36. <asp:Parameter Name="original_Name" Type="String" />
37. <asp:Parameter Name="original_Author" Type="String" />
38. <asp:Parameter Name="original_Press" Type="String" />
39. </UpdateParameters>
40. </asp:SqlDataSource>

```

代码说明: 第1行添加了一个服务器列表控件 GridView1 并设置 AllowPaging 属性启用分页功能。第2行设置 AutoGenerateColumns 为“false”禁用自动生成列的功能。第3行通过 DataKeyNames 属性将 ID 字段作为主键,同时设置 DataSourceID 属性将数据源控件 SqlDataSource1 作为 GridView 的数据源。

第5~8行分别定义 GridView 控件的四个列 Name、ID、Author 和 Press, 每一列的定义包括数据区域指定、列标题和排序表达式的设置。第9和第10行分别添加一个操作列 CommandField, 显示编辑按钮和删除按钮。

第13行添加了一个服务器数据源控件 SqlDataSource1。第14行设置控件的 ConnectionString 属性连接字符串对象为 BookStorConnectionString, 该字符串自动在 Web.config 文件中的 <connectionStrings></connectionStrings> 节点中生成。第15行通过 DeleteCommand 属性设置删除数据表数据的 sql 语句。第16行通过 InsertCommand 属性设置插入数据表数据的 sql 语句。第17行

通过 SelectCommand 属性设置查询数据表数据的 sql 语句。第 18 行通过 UpdateCommand 属性设置查询数据表数据的 sql 语句。第 19~24 行定义删除命令中的四个参数编号和类型。第 25~30 行定义插入命令中的四个参数和类型。第 32~39 行定义更新命令中的 4 个新参数、4 个旧参数已及它们的类型。

06 按下“Ctrl+F5”，运行结果如图 8-12 所示。单击表中第一行数据中的“编辑”按钮。

07 进入如图 8-13 所示的编辑操作。每一列可编辑的数据都以文本框的形式出现，这样用户就可以修改其中的数据。输入新的书名“神雕侠侣”、新的作者“金庸”和新的出版社“上海三联出版社”。如果想取消更新操作可以单击“取消”按钮，回到图 8-12 所示的界面。如果确认要进行更新操作，就单击“更新”按钮。

GridView控件更新和删除数据

编号	书名	作者	出版社	操作	操作
100001	饮马流花河	萧逸	商务出版社	编辑	删除
100002	三国演义	罗贯中	人民文学出版社	编辑	删除
100003	红楼梦	曹雪芹	人民文学出版社	编辑	删除
100004	水浒传	施耐庵	人民文学出版社	编辑	删除
100005	唐诗三百首	蘅塘退士	古籍出版社	编辑	删除

图 8-12 运行结果 1

GridView控件更新和删除数据

编号	书名	作者	出版社	操作	操作
100001	神雕侠侣	金庸	上海三联出版社	更新	取消
100002	三国演义	罗贯中	人民文学出版社	编辑	删除
100003	红楼梦	曹雪芹	人民文学出版社	编辑	删除
100004	水浒传	施耐庵	人民文学出版社	编辑	删除
100005	唐诗三百首	蘅塘退士	古籍出版社	编辑	删除

图 8-13 运行结果 2

08 GridView 控件显示如图 8-14 所示的更新数据后的界面。第一行中新的数据显示在列表中，如果要删除此条新数据，单击“删除”按钮即可。

09 删除第一条数据后的 GridView 控件如图 8-15 所示。

GridView控件更新和删除数据

编号	书名	作者	出版社	操作	操作
100001	神雕侠侣	金庸	上海三联出版社	编辑	删除
100002	三国演义	罗贯中	人民文学出版社	编辑	删除
100003	红楼梦	曹雪芹	人民文学出版社	编辑	删除
100004	水浒传	施耐庵	人民文学出版社	编辑	删除
100005	唐诗三百首	蘅塘退士	古籍出版社	编辑	删除

图 8-14 运行结果 3

GridView控件更新和删除数据

编号	书名	作者	出版社	操作	操作
100002	三国演义	罗贯中	人民文学出版社	编辑	删除
100003	红楼梦	曹雪芹	人民文学出版社	编辑	删除
100004	水浒传	施耐庵	人民文学出版社	编辑	删除
100005	唐诗三百首	蘅塘退士	古籍出版社	编辑	删除
100006	鬼吹灯	天下霸唱	百花文艺出版社	编辑	删除

图 8-15 运行结果 4



自动套用格式中为我们提供了 10 多种 GridView 控件的外观格式，不需要编写代码。如果要设计自己的格式可以通过设置格式属性实现。

### 8.3 DetailsView 控件

DetailsView 控件主要用来从与它联系的数据源中一次显示、编辑、插入或删除一条记录。通常，它与 GridView 控件一起使用在主/详细方案中，GridView 控件用来显示主要的数据目录，而 DetailsView 控件显示每条数据的详细信息。

DetailsView 控件提供了如表 8-6 所示的属性。

表 8-6 DetailsView 控件的常用属性

属性	说明
AllowPaging	获取或设置一个值，该值指示是否启用分页功能
CurrentMode	获取 DetailsView 控件的当前数据输入模式
DataItem	获取绑定到 DetailsView 控件的数据项
DataItemCount	获取基础数据源中的项数
DataItemIndex	从基础数据源中获取 DetailsView 控件中正在显示的项的索引
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
PageCount	获取在 DetailsView 控件中显示数据源记录所需的页数
PageIndex	获取或设置当前显示页的索引
PagerSettings	获取对 PagerSettings 对象的引用，使用该对象可以设置 DetailsView 控件中的页导航按钮的属性
Rows	获取表示 DetailsView 控件中数据行的 DetailsViewRow 对象的集合
SelectedValue	获取 DetailsView 控件中选中行的数据键值

默认情况下，在 DetailsView 控件中一次只能显示一行数据，如果有很多行数据的话，就需要使用 GridView 控件一次或分页显示。不过，DetailsView 控件也支持分页显示数据，即，把来自数据源的控件利用分页的方式一次一行地的显示出来，有时一行数据的信息过多的话，利用这种方式显示数据的效果可能会更好。

若要启用 DetailsView 控件的分页行为，则需要把属性 AllowPaging 设置为 true，而其页面大小则是固定，始终都是一行。当启用 DetailsView 控件的分页行为时，则可以通过 PagerSettings 属性来设置控件的分页界面。

DetailsView 控件提供了如表 8-7 所示的常用方法。

表 8-7 DetailsView 控件的方法

方法	说明
ChangeMode	将 DetailsView 控件切换为指定模式
DeleteItem	从数据源中删除当前记录
InsertItem	将当前记录插入到数据源中
IsBindableType	确定指定的数据类型是否可以绑定到 DetailsView 控件中的字段
UpdateItem	更新数据源中的当前记录

DetailsView 控件提供的常用事件如表 8-8 所示。

表 8-8 DetailsView 控件的事件

事件	说明
ItemCommand	当单击 DetailsView 控件中的按钮时发生
ItemCreated	在 DetailsView 控件中创建记录时发生
ItemDeleted	在单击 DetailsView 控件中的“删除”按钮时，但在删除操作之后发生

(续表)

事件	说明
ItemDeleting	在单击 DetailsView 控件中的“删除”按钮时，但在删除操作之前发生
ItemInserted	在单击 DetailsView 控件中的“插入”按钮时，但在插入操作之后发生
ItemInserting	在单击 DetailsView 控件中的“插入”按钮时，但在插入操作之前发生
ItemUpdated	在单击 DetailsView 控件中的“更新”按钮时，但在更新操作之后发生
ItemUpdating	在单击 DetailsView 控件中的“更新”按钮时，但在更新操作之前发生
ModeChanged	在 DetailsView 控件试图在编辑、插入和只读模式之间更改时，但在更新 CurrentMode 属性之后发生
ModeChanging	当 DetailsView 控件试图在编辑、插入和只读模式之间更改时，但在更新 CurrentMode 属性之前发生
PageIndexChanged	当 PageIndex 属性的值在分页操作后更改时发生
PageIndexChanging	当 PageIndex 属性的值在分页操作前更改时发生

DetailsView 控件本身自带了编辑数据的功能，只要把属性 AutoGenerateDeleteButton、AutoGenerateInsertButton 和 AutoGenerateEditButton 设置为 true 就可以启用 DetailsView 控件的编辑数据的功能，当然实际的数据操作过程还是在数据源控件中进行。与 GridView 控件相比，DetailsView 控件中支持进行数据的插入操作。

【例 8-5】通过 SqlDataSource 控件、GridView 控件和 DetailsView 控件的结合使用，实现主从表查询，并在 DetailsView 控件中完成新建、编辑和删除数据的操作。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 8-5”。
- 02 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 DetailsView 控件、1 个 GridView 控件和 1 个 SqlDataSource 数据源控件。
- 03 配置 SqlDataSource1 控件的数据源绑定“BookInfo”数据表。
- 04 在 DetailsView 控件右上方有一个向右的黑色小三角，单击这个小按钮打开如图 8-16 所示的“DetailsView 任务”列表，展开“选择数据源”下拉列表，从中选择“SqlDataSource1”。
- 05 这时 DetailsView1 控件的外观会根据 SqlDataSource1 控件中的设置的属性发生如图 8-17 所示的相应变化。接着选中“DetailsView 任务”列表中的“启用编辑”、“启用分页”、“启用插入”、“启用删除”四个复选按钮。

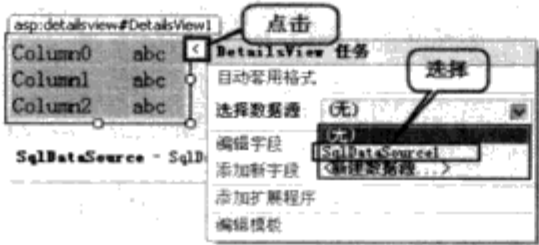


图 8-16 DetailsView 任务列表

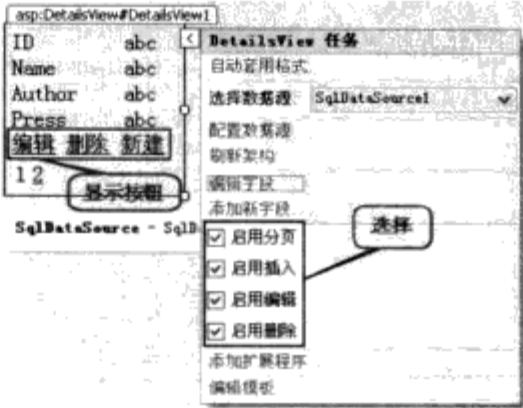


图 8-17 DetailsView 任务列表

**06** 在“DetailsView 任务”列表中选择“自动套用格式”，弹出自动套用格式对话框。我们单击“选择架构”列表中的“专业型”选项，然后单击“确定”按钮。

**07** 在“DetailsView 任务”列表中选择“编辑字段”，进入“字段”对话框，设置 DetailsView 控件中四个列字段：编号、书名、作者和出版社，以及要绑定的数据库表字段的值，最后设计好的 DetailsView 控件界面如图 8-18 所示。

**08** 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮打开“GridView 任务”列表，展开“选择数据源”下拉列表，从中选择“SqlDataSource1”。然后单击“GridView 任务”列表中“编辑列”，进入图 8-19 所示的“字段”对话框。选择“可用字段”列表中的“CommandField”，单击“添加”按钮。在“选定字段”列表中单击刚才我们选择的“CommandField”，右边“CommandField”列表中可以设置相关的属性。这里我们设置“ShowSelectButton”属性为 True，在 GridView 控件添加一个操作选择数据行的列，显示“选择”按钮，最后单击“确定”按钮。

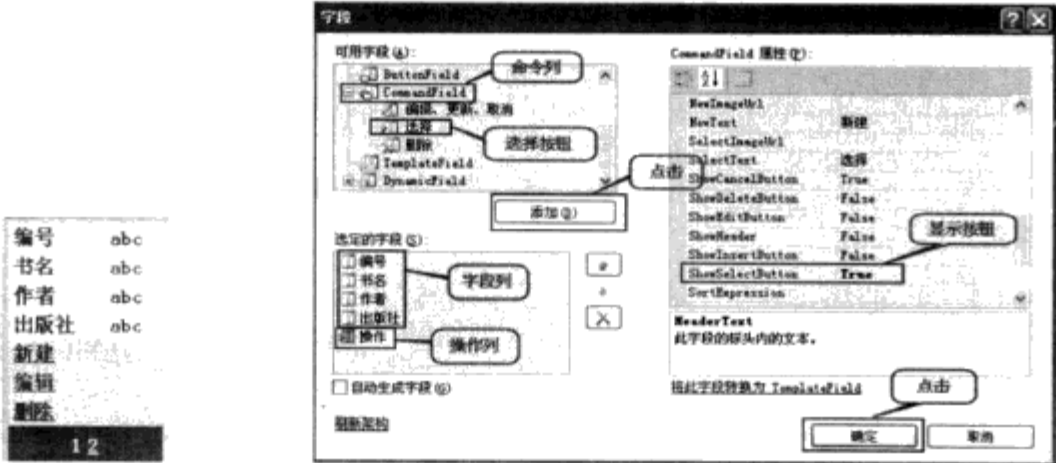


图 8-18 DetailsView 控件界面

图 8-19 字段对话框

**09** 在 GridView 控件的“属性”窗口中设置 AllowPaging 属性为“true”、 PageSize 属性为“3”、AutoGenerateColumns 属性为“false”。

**10** 用鼠标双击网站根目录下的“Default.aspx.cs”，在打开“Default.aspx.cs”文件的中添加如下代码。

```
1. protected void GridView1_SelectedIndexChanged1(object sender, EventArgs e){
2. this.DetailsView1.PageIndex = this.GridView1.SelectedRow.DataItemIndex;
3. }
```

代码说明：第 1 行处理 GridView1 控件的 SelectedIndexChanged1 事件，第 2 行将控件中选择行的数据项的索引作为 DetailsView1 的页面索引。

**11** 按下“Ctrl+F5”，运行结果如图 8-20 所示。页面中上面的主列表是 GridView1 控件，下面的从列表是 DetailsView1 控件。单击上表数据行中的“选择”按钮，下表显示相关数据行详细信息。单击下表中“新建”按钮。

**12** DetailsView 中每一列可编辑的数据都以文本框的形式出现，这样用户就可以新建其中的数据。输入如图 8-21 所示的书名“神雕侠侣”、作者“金庸”和出版社“上海三联出版社”。如果想取消更新操作可以单击“取消”按钮，回到上面图 8-20 的界面。如果确认要进行新建操作，就单击“插入”按钮。

**13** 插入数条数据后的 GridView 控件最后一条显示新插入的数据，单击“选择”按钮，

DetailView 控件同时显示如图 8-22 所示。至于 DetailView 控件编辑和删除的操作与 GridView 控件类似, 这里不再演示。



图 8-20 运行结果

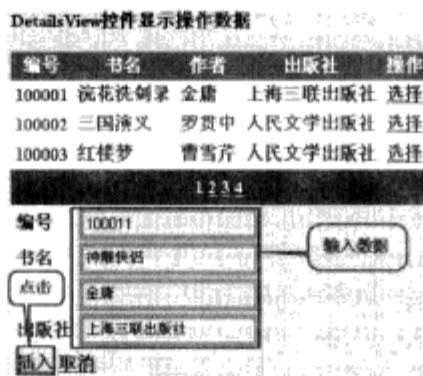


图 8-21 运行结果 2



图 8-22 运行结果 3



DetailView 控件通常用在主/详细信息方案中, 在这种方案中, 主控件(如 GridView 控件)中的所选记录决定了 DetailView 控件显示的记录。

## 8.4 Repeater 控件

Repeater 控件是一个数据绑定列表控件, 允许通过为列表中显示的每一项重复指定模板来定义布局。它没有内置的布局或样式, 因此必须在此控件的模板内显式声明所有的 HTML 布局、格式设置和样式标记。Repeater 的数据显示形式, 完全由用户通过模板来控制, 因此可以有效地弥补 GridView 控件显示效果不丰富的缺点。

Repeater 控件完全是模板驱动的。对于同样的 DataSource, 通过应用不同的模板, 就可以得到不同的外观表现。Repeater 是唯一运行开发人员在模板间拆分 HTML 标记的控件。利用模板创建表, 在 HeaderTemplate 模板中包含表开始标记<table>, 在 ItemTemplate 模板中包含单个表行标记<tr>, 并在 FooterTemplate 模板中包含结束标记</table>。每个 Repeater 必须至少定义一个 ItemTemplate。各种 Template 模板说明如下。

- ItemTemplate: 这是唯一必选的模板, 它用来完成对列表内容和布局的定义。
- AlternatingItemTemplate: 如果该模板定义的话, 则可以用它决定替换项的布局和内容。如果没定义的话, 则使用 ItemTemplate 项的定义。
- SeparatorTemplate: 如果对其进行定义的话, 则在项(交替项)之间将出现分隔符; 如果未定义的话, 则不会出现分隔符。
- HeaderTemplate: 如果对其进行定义的话, 则可以决定列表标头的布局和内容; 如果未定义的话, 则不出现标头。
- FooterTemplate: 如果对其进行定义的话, 则可以决定列表注脚的布局和内容; 如果未定义的话, 则不出现注脚。

**【例 8-6】** 本例使用 BookStor 数据库的 BookInfo 数据表, 通过 Repeater 控件的模板设计自定义的表格, 以 SqlDataSource 控件为数据源显示数据表内容。

- 01 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 8-6”。
- 02 用鼠标双击网站的目录下的“Default.aspx”文件, 进入“视图编辑界面”, 打开“设计视图”, 从工具箱中拖动 1 个 Repeater 控件和 1 个 SqlDataSource 数据源控件。
- 03 配置 SqlDataSource1 控件的数据源绑定“BookInfo”数据表。
- 04 在 Repeater 控件右上方有一个向右的黑色小三角, 单击这个小按钮打开如图 8-23 所示的“Repeater 任务”列表, 展开“选择数据源”下拉列表, 从中选择“SqlDataSource1”。



图 8-23 Repeater 任务列表

- 05 切换到“源视图”, 在编辑区中<form></form>标记之间编写如下代码。

```

1. <h4>Repeater 模板显示数据</h4>
2. <asp:Repeater ID="Repeater1" runat="server" DataSourceID="SqlDataSource1">
3. <ItemTemplate>
4. <tr>
5. <td>
6. <%# DataBinder.Eval(Container.DataItem, "ID")%>
7. </td>
8. <td>
9. <%# DataBinder.Eval(Container.DataItem, "Name") %>
10. </td>
11. <td>
12. <%# DataBinder.Eval(Container.DataItem, "Author") %>
13. </td>
14. <td>
15. <%# DataBinder.Eval(Container.DataItem, "Press") %>
16. </td>
17. </tr>
18. </ItemTemplate>
19. <HeaderTemplate>
20. <table border="1">
21. <tr>
22. <td>编号</td>
23. <td>书名</td>
24. <td>作者</td>
25. <td>出版社</td>
26. </tr>
27. </HeaderTemplate>
28. <FooterTemplate>
29. </table>
30. </FooterTemplate>
31. </asp:Repeater>
32. <asp:SqlDataSource ID="SqlDataSource1" runat="server"
33. ConnectionString="<%= $ConnectionString:BookStorConnectionString %>"
34. SelectCommand="SELECT * FROM [BookInfo]"></asp:SqlDataSource>

```

代码说明: 第 1 行添加显示的标题。第 2 行添加了一个服务器列表控件 Repeater1 并设置控件的数据源为 SqlDataSource1。第 3~18 行定义了 Repeater 控件的 ItemTemplate 模版。其中, 第 4~17 行把数据库表“BookInfo”中的各项和 Repeater 控件中的项绑定在一起。DataBinder 类提供对应用

程序快速开发 (RAD) 设计器的支持, 以便生成和分析数据绑定表达式语法。在 Web 窗体页数据绑定语法中可以使用此类的重载静态 Eval 方法。与标准数据绑定相比, 这提供的语法更容易记忆, 但是因为 DataBinder.Eval 提供自动类型转换, 这会导致服务器响应时间变长。

第 19~27 行定义了 Repeater 控件的 HeaderTemplate 模版。这个模版用来显示 Repeater 控件头部列标题的信息: 编号、书名、作者和出版社。第 28~30 行定义了 Repeater 控件的 FooterTemplate 模版, 在 29 行包含了表格的结束标记 </table>。

第 32 行添加了 1 个服务器数据源控件 SqlDataSource1。第 33 行设置控件的 ConnectionString 属性连接字符串对象为 BookStorConnectionString, 该字符串自动在 Web.config 文件中的 <connectionStrings></connectionStrings> 节点中生成。第 34 行设置控件的 SelectCommand 属性为查询数据库的 sql 语句。

Repeater 模板显示数据

编号	书名	作者	出版社
100001	浣花洗剑录	金庸	上海三联出版社
100002	三国演义	罗贯中	人民文学出版社
100003	红楼梦	曹雪芹	人民文学出版社
100004	水浒传	施耐庵	人民文学出版社
100005	唐诗三百首	蒋清士	古籍出版社
100006	鬼吹灯	天下霸唱	百花文艺出版社
100007	天机	蔡骏	上海文艺出版社
100008	藏地密码	何马	世纪出版社
100009	射雕英雄传	金庸	上海三联出版社
100010	倚天屠龙记	金庸	上海三联出版社
100011	神雕侠侣	金庸	上海三联出版社

**06** 按下 “Ctrl+F5”, 运行结果如图 8-24 所示。

图 8-24 运行结果



在 Repeater 控件的模板中, 有两个模板 HeaderTemplate 和 FooterTemplate 中是不支持数据绑定的。一般情况下使用绑定数据最多的就是 ItemTemplate 模板。

## 8.5 DataList 控件

DataList 控件默认使用表格方式来显示数据, 其使用方法与 Repeater 控件相似, 也是使用模板标记。不过, DataList 控件新增了 SelectItemTemplate 和 EditItemTemplate 模板标记, 可支持选取和编辑功能。DataList 控件可自定义的格式显示数据库行的信息, 显示数据的格式在创建的模板中定义。可以为项、交替项、选定项和编辑项创建模板, 在这些模板中, 除了 ItemTemplate 模板外, 其他都是可选的。各种 Template 模板说明如下。

- AlternatingItemTemplate: 类似于 ItemTemplate 元素, 但在 DataList 控件中隔行 (交替行) 呈现。通过设置 AlternatingItemTemplate 元素的样式属性, 可以为其指定不同的外观。
- EditItemTemplate: 项在设置为编辑模式后的布局。此模板通常包含编辑控件 (如 TextBox 控件)。当 EditItemIndex 设置为 DataList 控件中某一行的序号时, 将为该行调用 EditItemTemplate。
- FooterTemplate: 在 DataList 控件的底部 (脚注) 呈现的文本和控件。FooterTemplate 不能进行数据绑定。
- HeaderTemplate: 在 DataList 控件顶部 (标头) 呈现的文本和控件。HeaderTemplate 不能进行数据绑定。
- ItemTemplate: 为数据源中的每一行都呈现一次的元素。
- SelectedItemTemplate: 当用户选择 DataList 控件中的一项时呈现的元素。通常的用法是增加所显示的数据字段的个数并以可视形式突出标记该行。
- SeparatorTemplate: 在各项之间呈现的元素。SeparatorTemplate 项不能进行数据绑定。

8.5.1 DataList 控件的属性和事件

DataList 控件的相关属性很多，下面介绍常用属性如表所示 8-9 所示。

表 8-9 DataList 控件的常用属性

属性	说明
AlternatingItemStyle	指定 DataList 控件中交替项的样式
HeaderStyle	指定 DataList 控件中页眉的样式
EditItemStyle	指定 DataList 控件中正在编辑的项的样式
ItemStyle	指定 DataList 控件中项的样式
SelectedItemStyle	指定 DataList 控件中选定项的样式
SeparatorStyle	指定 DataList 控件中各项之间的分隔符的样式
RepeatColumns	设置 DataList 控件的数据分成多列显示
RepeatDierction	设置 DataList 控件数据项的呈现方式是 Vertical（垂直，默认值）还是 Horizontal（水平）
RepeatLayout	DataList 控件显示方式的版面配置为 Table 或 Flow

DataList 控件提供了与行行为，以及对行的数据进行选择、编辑、更新和删除等操作相关事件，如表 8-10 所示。

表 8-10 DataList 控件的事件

事件	说明
ItemCommand	当单击 DataList 控件中的任一按钮时发生
ItemCreated	当在 DataList 控件中创建项时在服务器上发生
ItemDataBound	当项被数据绑定到 DataList 控件时发生
EditCommand	对 DataList 控件中的某项单击 Edit 按钮时发生
CancelCommand	对 DataList 控件中的某项单击 Cancel 按钮时发生
UpdateCommand	对 DataList 控件中的某项单击 Update 按钮时发生
DeleteCommand	对 DataList 控件中的某项单击 Delete 按钮时发生
SelectedIndexChanged	在两次服务器发送之间，在数据列表控件中选择了不同的项时发生

8.5.2 编辑 DataList 控件的模板

编辑项模板除了可以以手写代码的方法外，还可以通过项模板编辑器进行可视化操作。在编辑器中提供了项模板、页脚和页眉模板以及分隔符模板，这 3 种是比较常用的项模板。编辑项模板的步骤如下。

- 01
- 在设计视图中选中 DataList 控件后右键单击，在弹出如图 8-25 所示的快捷菜单中，选中“编辑模板”|“项模板”命令。
- 02
- 进入如图 8-26 所示的“项模板”编辑器。项模板有 4 中类型：ItemTemplate（普通项）、AlternatingItemTemplate（交叉项）、SelectedItemTemplate（选中项）、EditItemTemplate（可编辑

项)，可以分别为这些项进行项的编辑。



图 8-25 选择项模板



图 8-26 模板编辑器

ItemTemplate 控制的是 DataList 中每一行的外观；AlternatingItemTemplate 控制的是交替项的外观，当上下两项具有不同外观时，使用该项来设置，奇数项显示由 ItemTemplate 控制的外观，偶数项显示由 AlternatingItemTemplate 控制的外观；SelectedItemTemplate 控制的是被选中项的外观；EditItemTemplate 控制 DataList 控件中为进行编辑而选定的项内容，在需要进行编辑时，将外观从 ItemTemplate 切换到 EditItemTemplate，然后可以修改项中的内容。

**03** 编辑设计 DataList 中 ItemTemplate 模板和 SelectedItemTemplate。分别在 ItemTemplate 和 SelectedItemTemplate 中各自添加一个 LinkButton，并设置不同的颜色背景，这样在单击此按钮时，因为颜色的不同，就很容易与未被选中的状态区别开来。编辑完的模板如图 8-27 所示。



图 8-27 编辑后的项模板

**04** 模板编辑结束后，在模板上右键单击，在弹出的快捷菜单中选择“结束模板编辑”命令，DataList 模板进入不可编辑的状态。

### 8.5.3 使用属性编辑器

除了使用模板来编辑 DataList 控件的外观以外，也可以使用属性编辑器来修改外观，具体步骤如下。

**01** 选中 DataList 控件，在“属性”窗口中单击如图 8-28 所示的图标。

**02** 弹出如图 8-29 所示的“属性生成器”对话框。在该对话框中左侧有三个选项，一个是“常规”选项，主要用于设置 DataList 控件显示表格的布局。一个是“格式”选项，用户设置各种项的

外观属性。还有一个是“边框”选项，用于设计表格的边框属性。我们打开“格式”选项，在“对象”列表框中选择“项”|“选定项”命令，并设置“背景色”为“Gray”。

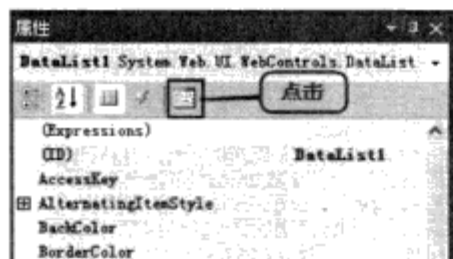


图 8-28 属性窗口

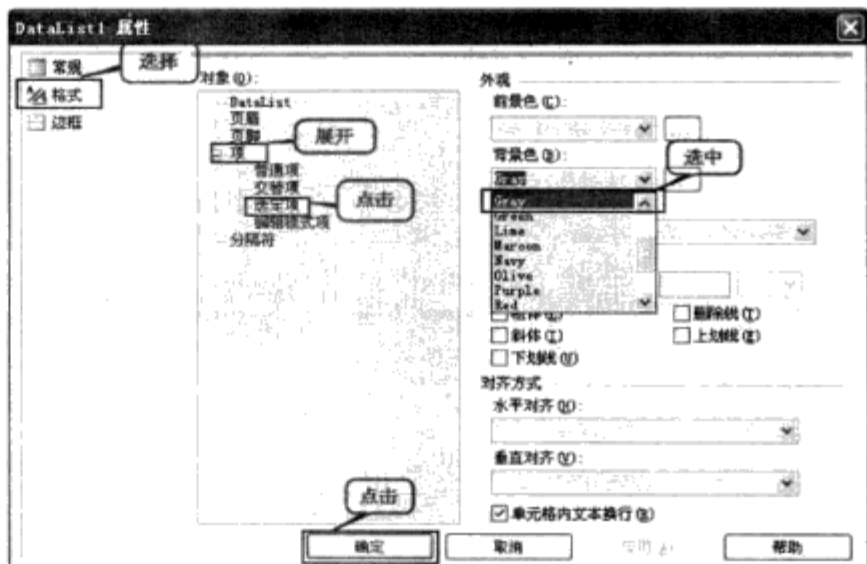


图 8-29 “属性生成器”对话框

**03** 设置完成后，进入 DataList 的项模板编辑器中，可以发现如图 8-30 所示的 SelectedItemTemplate 模板的背景色已经被改变。



图 8-30 选中项模板



如果是对 DataList 进行数据处理，EditCommand、UpdateCommand 和 DeleteCommand 事件很重要；如果是进行一些数据绑定时的动态操作，我们经常使用 ItemDataBound 事件。

**【例 8-7】** 本例通过 DataList 控件的模板，以 SqlDataSource 控件为数据源，实现 BookStor 数据库的 BookInfo 表中当用户选择某条记录后，将展开显示该数据的详细记录。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 8-7”。
- 02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计视图”，从工具箱中拖动 1 个 DataList 控件和 1 个 SqlDataSource 数据源控件。
- 03** 配置 SqlDataSource1 控件的数据源绑定“BookInfo”数据表。
- 04** 在 DataList 控件右上方有一个向右的黑色小三角，单击这个小按钮打开“DataList 控件任务”列表，展开“选择数据源”下拉列表，从中选择“SqlDataSource1”。
- 05** 在“DataList 任务”列表中选择“自动套用格式”，弹出自动套用格式对话框，单击“选择架构”列表中的“专业型”选项，然后单击“确定”按钮。
- 06** 切换到“源视图”，在编辑区中<form></form>标记之间编写如下代码。

```
1. <asp:DataList ID="DataList1" runat="server"
2. DataKeyField="ID" DataSourceID="SqlDataSource1"
3. onitemcommand="DataList1_ItemCommand" RepeatColumns="6">
```

```

4. <ItemTemplate>
5. ID:<asp:Label ID="IDLabel" runat="server" Text='<%# Eval("ID") %>' />
6.

7. Name:<asp:Label ID="NameLabel" runat="server" Text='<%# Eval("Name") %>' />

8. <asp:LinkButton ID="LinkButton1" runat="server" CommandName="Select" Text="查看"></asp:LinkButton>
9. </ItemTemplate>
10. <SelectedItemTemplate>
11. ID:<asp:Label ID="IDLabel" runat="server" Text='<%# Eval("ID", "{0}") %>' />

12. Name:<asp:Label ID="NameLabel" runat="server" Text='<%# Eval("Name", "{0}") %>' />

13. Author:<asp:Label ID="AuthorLabel" runat="server"
14. Text='<%# DataBinder.Eval(Container.DataItem,"Author") %>' />

15. Press:<asp:Label ID="PressLabel" runat="server" Text='<%# DataBinder.Eval(Container.DataItem,"Press") %>' />
16. </SelectedItemTemplate>
17. </asp:DataList>

```

代码说明：第 1 行添加一个服务器 DataList 控件。第 2 行设置控件的数据源为 SqlDataSource1，列表项的主键为 ID。第 3 行定义了单击 DataList 控件项时进入的处理函数，在用户单击 Select 链接的时候会触发 OnItemCommand 事件；同时设置用于表布局中每行中有 6 列数据。第 4~9 行定义了控件 ItemTemplate 模板的内容。其中，第 5、第 7 行各添加一个标签控件，使用绑定表达式绑定数据库表“BookInfo”中的 ID 和 Name 字段值显示在标签控件上。第 8 行添加一个服务器链接控件，设置按钮的命令为“Select”。

第 10~16 行定义了控件 SelectedItemTemplate 模板的内容。其中，第 11~15 行各添加一个标签控件，使用绑定表达式绑定数据库表“BookInfo”中的 ID、Name、Author 和 Press 字段值显示在标签控件上。

**07** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```

1. protected void DataList1_ItemCommand(object source, DataListCommandEventArgs e){
2. DataList1.SelectedIndex = e.Item.ItemIndex;
3. DataList1.DataBind();
4. }

```

代码说明：第 1 行定义处理 DataList1 控件项命令事件 ItemCommand 的方法。第 2 行中的 e.Item.ItemIndex 参数中记录了用户选择的项，通过把这一项赋给 DataList 控件的 SelectedIndex 属性，确定 DataList 控件中需要展开的项。第 3 行调用 DataBind 方法重新进行数据绑定，以便显示展开的项。

**08** 按下“Ctrl+F5”，运行结果如图 8-31 所示。

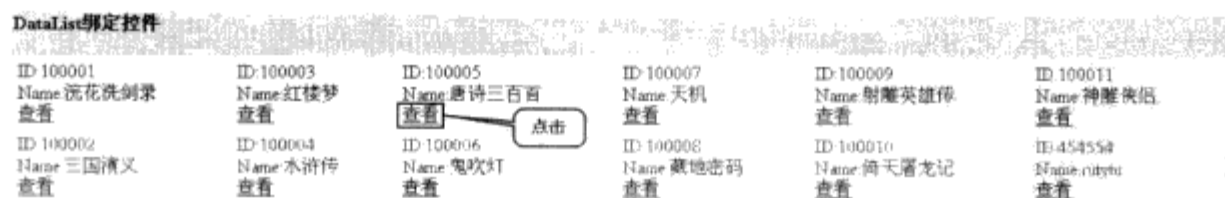


图 8-31 运行结果 1

**09** 用户单击某一条数据中的“查看”链接，显示如图 8-32 所示的数据详情。

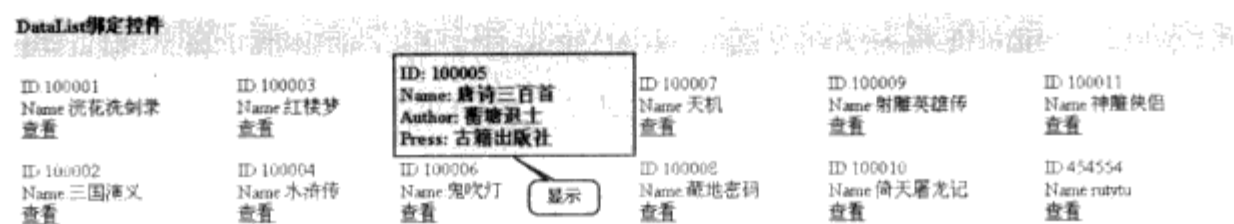


图 8-32 运行结果 2

## 8.6 ListView 控件

ListView 控件使用用户定义的模板显示数据源的值，它类似于 GridView 控件，可以显示使用数据源控件或 ADO.NET 获得的数据，但 ListView 控件和 GridView 的区别在于它可以使用模板和样式定义的格式显示数据。利用 ListView 控件，还能够逐项显示数据或按组显示数据，使控制数据的显示方式更加灵活。该控件同样支持选择、排序、分页、删除、编辑和插入记录。

相比 GridView 控件，ListView 控件基于模板的模式为开发人员提供了可自定义和可扩展的特性，利用这些特性，程序员可以完全控制由数据绑定控件产生的 HTML 标记的外观。ListView 控件使用内置的模板可以指定精确的标记，同时还可以用最少的代码执行数据操作。表 8-11 列举了 ListView 控件可支持的模板。

表 8-11 ListView 控件可支持的模板

模板	说明
LayoutTemplate	标识定义控件的主要布局的根模板。它包含一个占位符对象，例如表行 (tr)、div 或 span 元素。此元素将由 ItemTemplate 模板或 GroupTemplate 模板中定义的内容替换
ItemTemplate	标识要为各个项显示的数据绑定内容
ItemSeparatorTemplate	标识要在各个项之间呈现的内容
GroupTemplate	标识组布局的内容。它包含一个占位符对象，例如表单元格 (td)、div 或 span。该对象将由其他模板（例如 ItemTemplate 和 EmptyItemTemplate 模板）中定义的内容替换
GroupSeparatorTemplate	标识要在项组之间呈现的内容
EmptyItemTemplate	标识在使用 GroupTemplate 模板时空项呈现的内容。例如，如果将 GroupItemCount 属性设置为 5，而从数据源返回的总项数为 8，则 ListView 控件显示的最后一行数据将包含 ItemTemplate 模板指定的 3 个项以及 EmptyItemTemplate 模板指定的 2 个项
EmptyDataTemplate	标识在数据源未返回数据时要呈现的内容
SelectedItemTemplate	标识为区分所选数据项与显示的其他项，而为该所选项呈现的内容
AlternatingItemTemplate	标识为便于区分连续项，而为交替项呈现的内容
EditItemTemplate	标识要在编辑项时呈现的内容。对于正在编辑的数据项，将呈现 EditItemTemplate 模板以替代 ItemTemplate 模板
InsertItemTemplate	标识要在插入项时呈现的内容。将在 ListView 控件显示的项的开始或末尾处呈现 InsertItemTemplate 模板，以替代 ItemTemplate 模板。通过使用 ListView 控件的 InsertItemPosition 属性，可以指定 InsertItemTemplate 模板的呈现位置

通过创建 LayoutTemplate 模板,可以定义 ListView 控件的主要(根)布局。LayoutTemplate 必须包含一个充当数据占位符的控件。这些控件将包含 ItemTemplate 模板所定义的每个项的输出,可以在 GroupTemplate 模板定义的内容中对这些输出进行分组。

在 ItemTemplate 模板中,需要定义各个项的内容。此模板包含的控件通常已绑定到数据列或其他单个数据元素。

使用 GroupTemplate 模板,可以选择对 ListView 控件中的项进行分组。对项分组通常是为了创建平铺的表布局。在平铺的表布局中,各个项将在行中重复 GroupItemCount 属性指定的次数。

使用 EditItemTemplate 模板,可以提供已绑定数据的用户界面,从而使用户可以修改现有的数据项。使用 InsertItemTemplate 模板还可以定义已绑定数据的用户界面,以使用户能够添加新的数据项。

通常需要向模板中添加一些按钮,以允许用户指定要执行的操作。例如,可以向项模板中添加“Delete”(删除)按钮,以允许用户删除该项。通过在模板中添加“Edit”(编辑)按钮,可允许用户切换到编辑模式。在 EditItemTemplate 中,可以添加允许用户保存更改的“Update”(更新)按钮。此外,还可以添加“Cancel”(取消)按钮,以允许用户在不保存更改的情况下切换回显示模式。通过设置按钮的 CommandName 属性,可以定义按钮将执行的操作。下面列出了一些 CommandName 属性值,ListView 控件已内置了针对这些值的行为。

- Select: 显示所选项的 SelectedItemTemplate 模板的内容。
- Insert: 在 InsertItemTemplate 模板中,将数据绑定控件的内容保存在数据源中。
- Edit: 把 ListView 控件切换到编辑模式,并使用 EditItemTemplate 模板显示项。
- Update: 在 EditItemTemplate 模板中,指定应将数据绑定控件的内容保存在数据源中。
- Delete: 从数据源中删除项。
- Cancel: 取消当前操作。显示 EditItemTemplate 模板时,如果该项是当前选定的项,则取消操作会显示 SelectedItemTemplate 模板,否则将显示 ItemTemplate 模板。显示 InsertItemTemplate 模板时,取消操作将显示空的 InsertItemTemplate 模板。
- 自定义值: 默认情况下,不执行任何操作。用户可以为 CommandName 属性提供自定义值。随后在 ItemCommand 事件中测试该值并执行相应的操作。

**【例 8-8】**使用 ListView 控件的模板,以 SqlDataSource 控件为数据源,实现显示 BookStore 数据库的 BookInfo 表的内容并可对数据进行新建、编辑和删除操作。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 8-8”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件,进入“视图编辑界面”,打开“设计视图”,从工具箱中拖动 1 个 ListView 控件和 1 个 SqlDataSource 数据源控件。

**03** 配置 SqlDataSource1 控件的数据源绑定“BookInfo”数据表。

**04** 在 ListView 控件右上方有一个向右的黑色小三角,单击这个小按钮打开“ListView 控件任务”列表,展开“选择数据源”下拉列表,从中选择“SqlDataSource1”。

**05** 在“ListView 控件任务”列表中,选择“配置 ListView”选项,弹出如图 8-33 所示的“配置 ListView”对话框。其中,“选择布局”列表中显示了控件可用的五种布局方式:网格以表格布

局显示数据，平铺是使用组模板的平铺表格布局显示数据，项目符号列表是数据显示在项目符号列表中，流表示数据以使用 div 元素的流布局显示，单行是使数据显示在只有一行的表中。“选择样式”列表中显示了控件可用的四种外观样式。“选项”下的复选按钮提供控件可实现的功能，有编辑、插入、删除和分页，如果选择“启用分页”，必须在下方的下拉列表中选择分页的导航布局方式：一种是“下一页”/“上一页”页文字导航另一种是数字页码导航。我们在这里布局选择“网格”，样式选择“专业性”，功能选择全部四种并在分页功能中使用“下一页”/“上一页”页文字导航，最后单击“确认”按钮。

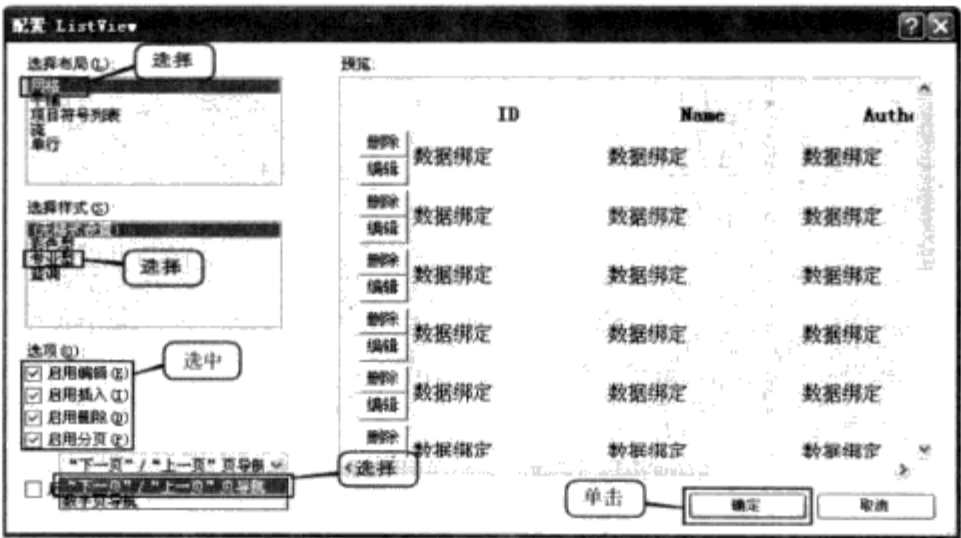


图 8-33 “配置 ListView”对话框

**06** 切换到“源视图”，在编辑区中设置分页的 PageSize 属性，让每页显示 5 条数据，编写如下代码：

```
<asp:DataPager ID="DataPager1" runat="server" PageSize="5"> </asp:DataPager>
```

代码说明：在服务器数据分页控件 DataPager1 的定义中，设置其 PageSize 属性为 5 表示每页显示 5 条数据。

**07** 按下“Ctrl+F5”，运行结果如图 8-34 所示。

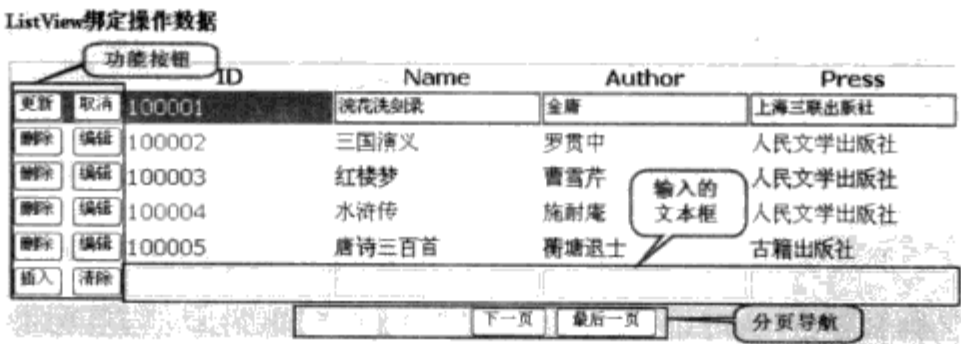


图 8-34 运行结果



ListView 控件中如果选择使用分页功能时，会自动生成一个专门用于分页的服务器控件 DataPager，该控件用于对实现 IPageableItemContainer 接口的控件（如 ListView 控件）所显示的数据进行分页。DataPager 控件支持内置的分页用户界面，让用户可以按照页码来选择页面，也可以让用户在不同页面之间导航，或者可以使用 TemplatePagerField 对象创建自定义的分页用户界面。

## 8.7 新增的 Chart 控件

Chart 控件是 Visual Studio 2010 中新增的一个图表型控件。该控件在 Visual Studio 2008 时代就已经出现，但是需要通过下载然后将它注册配置到 Visual Studio 2008 的工具箱中才能使用。现在，Chart 控件已经内置于 Visual Studio 2010 中了，这意味不用注册或连接任何配置文件项，就可以使用这个控件。所有的配置现在都由 ASP.NET 4.0 预先注册好了。在 Visual Studio 2010 开发环境中，发现在如图 8-35 所示的工具箱“数据”项下，已经存在了一个新的内置 Chart 控件。我们可以像使用其他控件一样将它直接拖到设计视图中就可以使用。

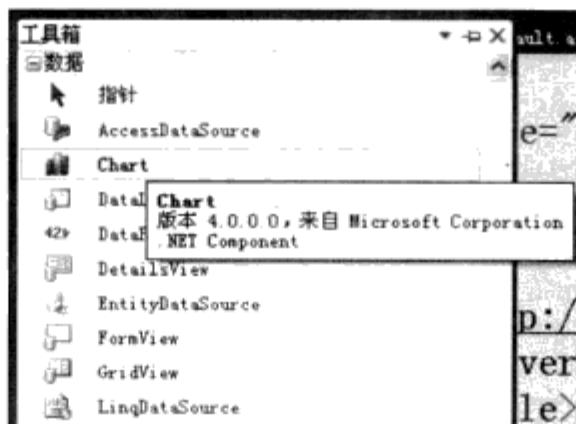


图 8-35 工具箱

Chart 控件功能非常强大，可实现柱状直方图、曲线走势图、饼状比例图等，甚至可以是混合图表，可以是二维或三维图表，可以带或不带坐标系，可以自由配置各条目的颜色、字体等等。声明一个 Chart 控件的代码如下所示。

```

1. <asp:Chart ID="Chart1" runat="server">
2. <Series>
3. <asp:Series Name="Series1"> </asp:Series>
4. </Series>
5. <ChartAreas>
6. <asp:ChartArea Name="ChartArea1"></asp:ChartArea>
7. </ChartAreas>
8. </asp:Chart>

```

以上代码，第 1 行添加了一个服务器图表控件 Chart1 控件。第 2~4 行使用<Series>和</Series>标签定义 Chart 控件的数据显示列的区域，其中第 3 行定义了一个名为 Series1 数据显示列。第 5~7 行使用<ChartAreas>和</ChartAreas>标签定义绘图区域的范围。其中，第 6 行定义了一个名为“ChartArea1”的绘图区域。根据声明可以了解 Chart 控件由 Series（数据列）和 ChartArea（绘图区域）两部分组成。这两部分都是可以有一个或者多个的组成，比如当在一个“图表”中要画多条曲线时，就可能会用到多个“Series”，并且把多个“Series”的 ChartArea 属性设置为指定的“绘图区域”。当我们想在一个“图表”中分区域多形式地显示一种或多种数据的时候，就需要多个 ChartArea 了。

对于简单的图表，只用默认的样式就可以了，不用对 ChartArea 进行太多的修改，只要在<asp:Series>中添加数据点就可以。数据点被包含在<Points>和</Points>标签中，使用<asp:DataPoint/>来定义。数据点有以下几个重要的属性。

- **AxisLabel**: 获取或设置为数据列或空点的 X 轴标签文本。此属性仅在自定义标签尚未就有关 Axis 对象指定时使用。
- **XValue**: 设置或获取一个图表上数据点的 X 坐标值。
- **YValues**: 设置或获取一个图表中数据点的 Y 轴坐标值。

下面是一个在源视图中编写代码实现简单图表的示例代码。

```

1. <asp:Chart ID="Chart1" runat="server">
2. <Series>
3. <asp:Series Name="Series1" YValuesPerPoint="4">
4. <Points>
5. <asp:DataPoint AxisLabel="火箭" YValues="17">
6. <asp:DataPoint AxisLabel="湖人" YValues="15" />
7. <asp:DataPoint AxisLabel="公牛" YValues="6" />
8. <asp:DataPoint AxisLabel="步行者" YValues="4" />
9. <asp:DataPoint AxisLabel="76 人" YValues="3" />
10. <asp:DataPoint AxisLabel="波士顿" YValues="3" />
11. <asp:DataPoint AxisLabel="骑士" YValues="3" />
12. </Points>
13. </asp:Series>
14. </Series>
15. <ChartAreas>
16. <asp:ChartArea Name="ChartArea1"></asp:ChartArea>
17. </ChartAreas>
18. </asp:Chart>

```

代码说明：第 1 行定义了一个服务器图表控件 Chart1 控件。第 2~12 使用<Series>和</Series>标签定义数据列范围。其中第 3 行定义数据列的名称和数据点 Y 轴具有的最大数目；第 4~12 行使用<Points>和</Points>标签包含需要显示的数据点。第 5~11 行定义了 7 个数据点并设置 X 轴上的显示文字和 Y 轴上的数据点的值。第 15~17 行使用<ChartAreas>和</ChartAreas>标签定义副图区域。第 16 行定义一个绘图区域的名称“ChartArea1”。程序运行结果如图 8-36 所示。

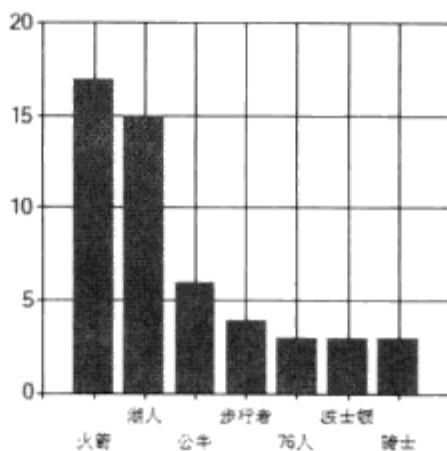


图 8-36 运行结果

**【例 8-9】**在“Northwind”数据库的“Products”和“Categories”数据表中查询获得产品的数量，并使用 Chart 控件的制作成柱状图表显示。“Northwind”数据库在随书光盘的本章源代码中提供，读者直接使用即可不需重新创建。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 8-9”。
- 02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入“视图编辑界面”，打开“设计

视图”，从工具箱中拖动 1 个 Chart 控件到视图中。

**03** 在 Chart 控件右上方有一个向右的黑色小三角，单击这个小按钮打开如图 8-37 所示的“Chart 控件任务”列表，展开“图表类型”下拉列表，从中选择“Column”柱状图。

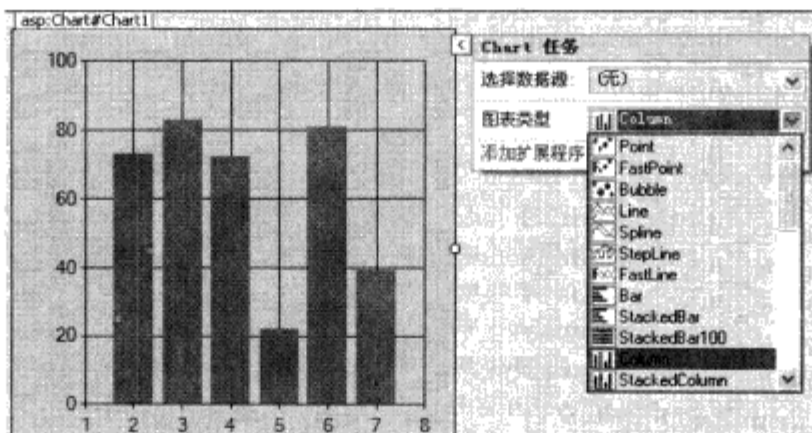


图 8-37 Chart 控件任务列表

**04** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```
1. protected void Page_Load(object sender, EventArgs {
2. SqlConnection cn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind;Integrated Security=True");
3. SqlCommand cmd = cn.CreateCommand();
4. cmd.CommandText = "SELECT CategoryName, COUNT(*) as ProductCount FROM Products p INNER JOIN
Categories c ON c.CategoryID = p.CategoryID GROUP BY CategoryName ORDER BY CategoryName";
5. cn.Open();
6. SqlDataReader dr = cmd.ExecuteReader();
7. Chart1.DataBindTable(dr, "CategoryName");
8. dr.Close();
9. cn.Close();
10. }
```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行创建一个 SqlConnection 对象 cn 并传递参数为连接字符串。第 3 行创建一个 SqlCommand 对象 cmd。第 4 行通过 cmd 对象的 CommandText 属性设置 Sql 查询命令，结合产品表和类别表查询出每种产品数量并排序。第 5 行通过 cn 对象的 open 方法打开数据库连接。第 6 行创建一个调用 cmd 对象的 ExecuteReader 方法创建一个数据读取器 SqlDataReader 对象 dr。第 7 行调用图表控件 Chart1 对象的 DataBindTable 方法自动创建数据列并绑定到数据表，同时指定填充 X 轴值为表中“CategoryName”字段的值。最后一行关闭数据库连接。

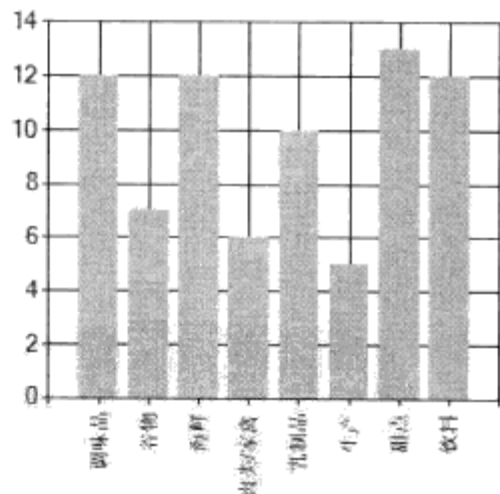


图 8-38 运行结果

**05** 按下“Ctrl+F5”，运行结果如图 8-38 所示，显示了各种产品数量的图表。



提示

在“Chart 控件任务”列表中，“图表类型”下拉列表内提供了三十多种不同类型的图表，如饼图、柱状图、曲线图、散点图、雷达图、面积图和股票图等。我们不需要编写代码就可以选择使用。

8.8 上机题

1. 利用第 5 章上机题 1 和 2 中创建的 School 数据库的 Students 数据表，读取表的数据并放在 DataSet 中，然后利用多值绑定方法把数据放在 GridView 中，程序运行结果如图 8-39 所示。

ID	StuName	Phone	Address	City
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海
5	赵芳	13658784596	海淀区	北京

图 8-39 运行结果

2. 使用 School 数据库中的 Students 数据表。通过 GridView 控件和数据源控件 SqlDataSource 实现 Students 数据表的列表显示，程序运行结果参考上图 8-39 所示。

3. 在上题的基础上，使用 GridView 控件配合 SqlDataSource 数据源控件实现对所选择的某条用户数据进行编辑操作，程序运行结果如图 8-40 所示。

ID	StuName	Phone	Address	City	
1	张琴	13256789023	朝阳门外大街	北京	编辑
2	邵飞	13698562314	海淀区清河镇	北京	编辑
3	王宁	15896325689	徐家汇美罗城	上海	编辑
4	黄韵琳	13965669856	浦东新区	上海	编辑
5	赵芳	13658784596	海淀区	北京	更新 取消

图 8-40 运行结果

4. 在上题的基础上，使用 GridView 控件配合 SqlDataSource 数据源控件实现对数据表进行分页和排序的操作，程序运行结果如图 8-41 所示。。

5.使用 GridView 控件的删除功能，配合 SqlDataSource 数据源控件实现对所选择的某条用户信息进行删除操作，程序运行结果如图 8-42 所示。

ID	StuName	Phone	Address	City
4	黄韵琳	13965669856	浦东新区	上海
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
12				

图 8-41 运行结果

ID	StuName	Phone	Address	City	
1	张琴	13256789023	朝阳门外大街	北京	删除
2	邵飞	13698562314	海淀区清河镇	北京	删除
3	王宁	15896325689	徐家汇美罗城	上海	删除
4	黄韵琳	13965669856	浦东新区	上海	删除
5	赵芳	13658784596	海淀区	北京	删除

图 8-42 运行结果

6. 使用 School 数据库中的 Students 数据表，通过 Repeater 控件的模板设计自定义的表格，以 SqlDataSource 控件为数据源显示数据表内容，程序运行结果如图 8-43 所示。

7. 通过 DataList 控件的模板，以 SqlDataSource 控件为数据源，实现在 School 数据库的 Students 数据表中当用户选择某条记录后，将展开显示该数据的详细记录，程序运行结果如图 8-44 所示。

编号	姓名	电话	地址	城市
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海
5	赵芳	13658784596	海淀区	北京

图 8-43 运行结果

选择 姓名 张琴 选择 姓名 邵飞

编号: 3  
姓名: 王宁  
电话: 15896325689  
地址: 徐家汇美罗城  
城市: 上海

选择 姓名 黄韵琳 选择 姓名 赵芳

图 8-44 运行结果

8. 使用 ListView 控件的模板，以 SqlDataSource 控件为数据源，实现显示 School 数据库中 Students 数据表的内容并可对数据进行插入操作，程序运行结果如图 8-45 所示。

	ID	StuName	Phone	Address	City
	1	张琴	13256789023	朝阳门外大街	北京
	2	邵飞	13698562314	海淀区清河镇	北京
	3	王宁	15896325689	徐家汇美罗城	上海
	4	黄韵琳	13965669856	浦东新区	上海
	5	赵芳	13658784596	海淀区	北京
插入	清除	6	李飞	13818765434	汉口区 武汉

图 8-45 运行结果

9. 使用 Chart 控件以 Column 类型的图表格式显示世界上获得围棋世界冠军最多的前 8 位选手的排名统计。本题相关数据库文件“go.mdf”在随书光盘内本章的源代码中提供，读者直接附加到 SQL Server 2005 使用即可。程序运行结果如图 8-46 所示。

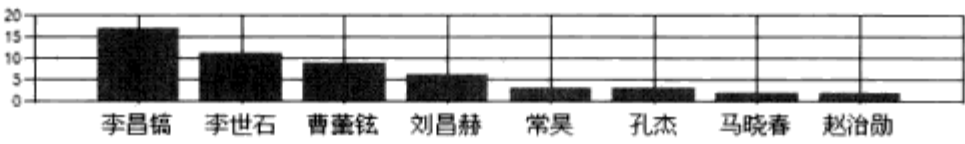


图 8-46 运行结果

# 第 9 章 文件操作

## 学习目标

在开发 Web 网站时，一般会把用户的数据存放到数据库中，但有的数据用数据库来保存不太方便，如图像、文档、文本等格式的数据和信息，此时可以采用文件形式来存储这些数据。尤其在 一些信息管理系统中，文档的处理和操作贯穿了整个系统的运行过程。所以本章就来介绍一下 .NET 4.0 框架中文件的操作以及文件操作在 ASP.NET 4.0 中的应用。

## 本章重点

- 掌握对目录的各种操作
- 掌握对文件的各种操作
- 使用 FileStream 类读写二进制文件
- 使用 StreamWriter 类和 StreamReader 类读写文本文件

### 9.1 获取驱动器信息

我们知道电脑中的文件都是存放在本地机器的各个驱动器中的，如果要找到文件的位置，最先要知道它处于哪个驱动器，然后才考虑文件的目录，因此，非常有必要获得驱动器的详细信息。在 ASP.NET 4.0 中，使用 DriveInfo 类就可以用来获得本地机器系统注册的驱动器详细信息，比如可以得到每个驱动器的名称、类型、大小和状态信息等等。

DriveInfo 类的主要属性和方法如表 9-1 所示。

表 9-1 DriveInfo 类的主要属性和方法

属性和方法	说明
TotalSize	获取驱动器上存储空间的总大小，以字节为单位，包含所有已分配的空间和空闲的空间
TotalFreeSpace	获取驱动器上的可用空闲空间总量
AvailableFreeSpace	指示驱动器上的可用空闲空间量，以字节为单位。其值可能小于 TotalFreeSpace，这是因为 ASP.NET 可用的驱动器配额可能有限制
DriveFormat	获取文件系统的名称，例如 NTFS 或 FAT32
DriveType	获取驱动器类型。表示驱动器为固态硬盘、网络驱动器、CD-ROM 还是可移动硬盘
IsReady	表示驱动器是否已准备好。返回一个 bool 类型的值，如果驱动器已准备好，则为 true；如果驱动器未准备好，则为 false

(续表)

属性和方法	说明
Name	获取驱动器的名称。比如“C:”或者“D:”
VolumeLabel	获取或设置驱动器的卷标。卷标的长度由操作系统确定。例如，NTFS 格式的驱动器上，允许卷标名最长达到 32 个字符。如果没有设置卷标，该属性值为“Null”
RootDirectory	获取驱动器的根目录
ToString	将驱动器名称作为字符串返回
GetDrives	该方法是一个静态方法，用于检索计算机上的所有逻辑驱动器的名称。该方法返回的是一个 DriveInfo 的对象集合

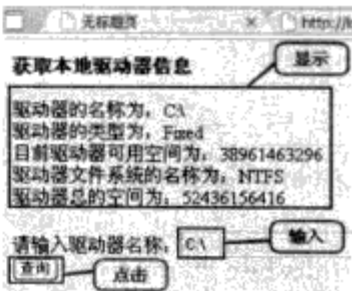
下面通过两个实例来学习如何使用 DriveInfo 类来获取本地驱动器的信息。

**【例 9-1】**这个例子实现查询本地磁盘驱动器的信息。当用户输入驱动器名称后，单击“查询”按钮，显示所查询驱动器的具体信息。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 9-1”。
- 02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动 1 个 TextBox 控件和 1 个 Button 控件。
- 03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```
1. protected void Button1_Click(object sender, EventArgs e){
2. string str = TextBox1.Text;
3. DriveInfo di = new DriveInfo(str);
4. Response.Write("<h4>获取本地驱动器信息</h4>");
5. Response.Write("驱动器的名称为: " + di.Name + "
");
6. Response.Write("驱动器的类型为: " + di.DriveType.ToString() + "
");
7. Response.Write("目前驱动器可用空间为: " + di.AvailableFreeSpace.ToString() + "
");
8. Response.Write("驱动器文件系统的名称为: " + di.DriveFormat + "
");
9. Response.Write("驱动器总的空间为: " + di.TotalSize.ToString() + "
");
10. }
```

代码说明：第 1 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 2 行获取用户输入的驱动器名称。第 3 行使用带一个参数的构造函数，实例化一个 DriveInfo 类对象 di，参数值为本地 D 盘的盘符。第 4 行显示标题文本。第 5 行使用 di 对象的 Name 属性显示驱动器的名称。第 6 行使用 di 对象的 DriveType 属性显示驱动器的类型。第 7 行使用 di 对象的 AvailableFreeSpace 属性显示驱动器的可用空间。第 8 行使用 di 的属性 DriveFormat 显示动器文件系统的名称。第 9 行使用 di 对象的 TotalSize 属性显示驱动器总的空间信息。



- 04** 按下“Ctrl+F5”，运行结果如图 9-1 所示。

图 9-1 运行结果



提示

在处理表示文件或目录路径的字符串中有许多“\”字符，所以加上@表示，这个字符串应逐个字符的解释，“\”不是解释为转义字符。如果没有@前缀，就要用“\\”来代替“\”，以避免把这个字符解释为转义字符。

**【例 9-2】**这个例子实现循环遍历本地机器中所有的驱动器，并显示可用空间的相关信息。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 9-2”。

**02** 用鼠标双击网站目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动一个 TreeView 控件。切换到“源视图”，在编辑区中的<form>和</form>标记间编写如下代码。

```
1. <h4>遍历所有本地驱动器</h4>
2. <asp:TreeView ID="TreeView1" runat="server"> </asp:TreeView>
```

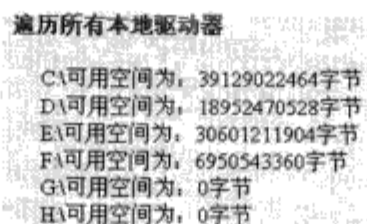
代码说明：第 1 行显示标题文本。第 2 行添加一个服务器树状控件 TreeView1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. foreach (DriveInfo di in DriveInfo.GetDrives()){
4. TreeNode node = new TreeNode();
5. node.Value = di.Name;
6. if (di.IsReady){
7. node.Text = di.Name + "可用空间为: " + di.AvailableFreeSpace + "" + "字节";
8. }
9. else {
10. node.Text = di.Name + "没有准备好! ";
11. }
12. this.TreeView1.Nodes.Add(node);
13. }
14. }
15. }
```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断当前加载的页面如果不是回传页面，则第 3 行使用 foreach 循环遍历本地所有的驱动器，其中使用 DriveInfo 类的 GetDrives 静态方法获得所有服务器的集合。第 4 行实例化一个 TreeNode 节点对象 node。通过 DriveInfo 类对象 di 的 Name 属性获得驱动器的名字并赋给节点的值。第 6 行判断如果 di 对象已经准备好，则第 7 行节点对象 node 的文本属性获得驱动器对象 di 的名称和目前可用的空间大小。否则第 10 行通过 node 对象的文本属性显示驱动器没有准备好的提示。第 12 行将每一个节点添加的到树状控件 TreeView1 的节点集合中。

**04** 按下“Ctrl+F5”，运行结果如图 9-2 所示。



```
遍历所有本地驱动器
C:\可用空间为: 39129022464字节
D:\可用空间为: 18952470528字节
E:\可用空间为: 30601211904字节
F:\可用空间为: 6950543360字节
G\可用空间为: 0字节
H\可用空间为: 0字节
```

图 9-2 运行结果

## 9.2 对目录的操作

获取目录信息也是常见的文件操作之一。在这个操作过程中，使用到两个重要的类

DirectoryDirectorySystem.IO.Directory 和 System.DirectoryInfo, 在使用前必须引用 System.IO 命名空间。下面对这两个类分别进行介绍。

### 9.2.1 Directory 类

Directory 类是一个静态类, 提供了许多的操作目录和子目录的静态方法, 可以用于对目录和子目录的创建、移动、复杂和删除等操作。由于这些方法都是静态方法, 因此可以在类上直接使用, 而不需要创建类的实例。Directory 类的主要静态方法有以下数种。

#### 1. CreateDirectory 方法

CreateDirectory 是创建目录的方法, 该方法的声明代码如下所示。

```
1. public static DirectoryInfo CreateDirectory(
2. String path
3.);
```

以上代码, 第 1 行定义一个 DirectoryInfo 类型的静态方法 CreateDirectory, 返回值是由 path 指定的 DirectoryInfo, 第 2 行中的字符串类型的对象参数 path 用于指定要创建的目录。

下面的示例代码在 D:\vs2010 文件夹下创建名为 Website 目录。

```
1. private void Create(){
2. Directory.CreateDirectory(@"D:\vs2010\Website");
3. }
```

以上代码, 第 1 行定义一个私有类型的方法 Create, 第 2 行调用 Directory 类的 CreateDirectory 静态方法在 D:\vs2010 文件夹下创建名为 Website 目录。

#### 2. Delete 方法

Delete 是删除目录的方法, 该方法的声明代码如下所示。

```
1. public static DirectoryInfo Delete(
2. String path,
3. bool recursive
4.);
```

以上代码, 第 1 行定义一个 DirectoryInfo 类型的静态方法 Delete。第 2 行中的字符串类型的对象参数 path 用于指定要删除的目录。第 3 行的第二个参数为 bool 类型, 可以指定是否删除非空目录, 如果该参数为 true, 将删除整个目录, 即使该目录下有文件或子目录; 如果该参数为 false, 则仅在目录为空时才可以删除。

下面的代码示例将 D:\vs2010 文件夹下创建名为 Website 目录删除。

```
1. private void DeleteDirectory(){
2. Directory.Delete(@"D:\vs2010\Website", true);
3. }
```

以上代码, 第 1 行定义一个私有类型的方法 DeleteDirectory, 第 2 行调用 Directory 类的静态方法 Delete 删除 D:\vs2010 文件夹下名为 Website 的目录。

### 3. Move 方法

Move 是移动目录的方法，该方法的声明代码如下所示。

```
1. public static void Move (
2. string sourceDirName,
3. string destDirName
4.);
```

以上代码，第 1 行定义一个静态方法 Move，第 2 行中的字符串类型的参数 sourceDirName 表示要移动的文件或目录的路径。第 3 行的第二个字符串类型的参数 destDirName 表示指向 sourceDirName 的新位置的路径。

下面的代码实现将目录 D:\vs2010 文件夹下名为 Website 目录移动到 C:\vs2010 文件夹下名为 Website 目录。

```
1. private void MoveDirectory(){
2. Directory.Move(@"D:\vs2010\Website", @"C:\vs2010\Website");
3. }
```

以上代码，第 1 行定义一个私有类型的方法 MoveDirectory。第 2 行调用 Directory 类的静态方法 Move 将目录 D:\vs2010 文件夹下名为 Website 目录移动到 C:\vs2010 文件夹下名为 Website 目录。

### 4. GetDirectories 方法

GetDirectories 是获取指定目录下所有子目录的方法，该方法的声明代码如下所示。

```
1. public static string[] GetDirectories (
2. string path
3.);
```

以上代码，第 1 行定义一个静态的 GetDirectories 方法，该方法返回一个字符串类型的数组，它包含 path 中子目录的名称。第 2 行的参数 path 为其返回子目录名称的数组的路径。

以下代码实现读取 D:\vs2010 文件夹下名为 Website 目录下的所有子目录，并将其保存到字符串数组中。

```
1. private void GetDirectory(){
2. string [] directorys;
3. directorys=Directory.GetDirectories(@"D:\vs2010\Website");
4. }
```

以上代码，第 1 行定义一个私有类型的方法 GetDirectory，第 2 行声明一个字符串数组 directorys，第 3 行使用 Directory 类的静态方法 GetDirectories 获得 D:\vs2010 文件夹下名为 Website 目录下的所有子目录。

### 5. GetFiles 方法

GetFiles 是获取指定目录下所有文件的方法，该方法的声明代码如下所示。

```
1. public static string[] GetFiles (
2. string path
3.);
```

以上代码，第 1 行定义一个静态的 `GetFiles` 方法，该方法返回一个字符串类型的数组，它包含 `path` 中子目录下所有文件的名称，文件名包含完整路径。第 2 行的参数 `path` 代表将从其开始检索文件的目录。

下面的代码实现了读取 `D:\vs2010` 文件夹下名为 `Website` 目录下的所有子目录，并将其保存到字符串数组中。

```
1. private void GetFile (){
2. string [] files;
3. files =Directory.GetFiles(@"D:\vs2010\Website");
4. }
```

以上代码，第 1 行定义一个私有类型的方法 `GetFile`，第 2 行声明一个字符串数组 `files`。第 3 行使用 `Directory` 类的静态方法 `GetFiles` 获得 `D:\vs2010` 文件夹下名为 `Website` 目录下的所有子目录。

## 6. Exist 方法

`Exist` 是判断指定目录是否存在的方法，该方法的声明代码如下所示。

```
1. public static bool Exists (
2. string path
3.);
```

以上代码，第 1 行定义一个静态的 `Exists` 方法，该方法返回一个布尔类型的值，如果目录存在返回值为 `true`，否则返回值为 `false`。第 2 行的参数 `path` 表示指定目录的路径。

下面的代码实现判断 `D:\vs2010` 文件夹下名为 `Website` 目录是否存在，如果存在则获取该目录下的子目录。

```
1. private void Handle(){
2. if(Directory.Exists(@"D:\vs2010\Website")){
3. string [] dis;
4. dis=GetDirectories ();
5. }
6. }
```

以上代码，第 1 行定义一个私有类型的方法 `Handle`。第 2 行调用 `Directory` 类的静态方法 `Exists` 判断如果 `D:\vs2010` 文件夹下名为 `Website` 目录存在，则第 3 行声明一个字符串数组 `dis`。第 4 行使用 `Directory` 类的静态方法 `GetDirectories` 获得 `D:\vs2010` 文件夹下名为 `Website` 目录下的所有子目录。

## 7. GetParent 方法

`GetParent` 是获取指定目录父目录的方法，该方法的声明代码如下所示。

```
1. public static DirectoryInfo GetParent (
2. string path
3.);
```

以上代码，第 1 行定义一个静态的方法 `GetParent`，返回值是由 `path` 指定的父目录。第 2 行中的字符串类型的对象参数 `path` 用于检索父目录的路径。

下面的代码实现返回 `D:\vs2010\Website` 目录的父目录 `D:\vs2010`。

```

1. private void GetLast(){
2. DirectoryInfo di;
3. Di=Directory. GetParent(@ "D:\vs2010\Website");
4. }

```

以上代码，第1行定义一个私有类型的方法 GetLast，第2行声明一个 DirectoryInfo 类型的对象 di 用于获取父目录的信息，第3行使用 Directory 类的静态方法 GetParent 获得 D:\vs2010 文件夹下名为 Website 目录的上一级父目录。



Directory 类的静态方法对所有方法都进行安全检查，如果打算多次重用某个对象，可以考虑改用 DirectoryInfo 的相应实例方法，这是因为其不总是需要安全检查。

提示

**【例 9-3】**这个例子将演示如何使用 Directory 类提供的静态方法创建、读取目录属性以及删除目录。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 9-3”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动 1 个 TextBox 控件、1 个 RadioButtonList 控件、1 个 Button 控件和 1 个 Label 控件。切换到“源视图”，在编辑区中的<form>和</form>标记间编写如下代码。

```

1. 请输入目录名
2. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
3. <asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True" RepeatDirection="Horizontal">
4. <asp:ListItem>删除目录</asp:ListItem>
5. <asp:ListItem Selected="True">创建目录</asp:ListItem>
6. </asp:RadioButtonList>
7. <asp:Button ID="Button1" runat="server" Text="确定" />

8. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>

```

代码说明：第2行添加一个服务器文本框控件 TextBox1。第3~6行添加了一个服务器单选按钮列表控件 RadioButtonList1 并设置自动回传服务器，控件布局方向为水平。第4第5行个添加了两个 RadioButtonList1 列表项，用于选择删除目录和创建目录操作的选择。第7行添加一个服务器按钮控件 Button1 并显示文本确定。第8行添加一个服务器标签控件 Label1 操作结果的文本显示。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```

1. protected void Button1_Click(object sender, EventArgs e){
2. try{
3. if (RadioButtonList1.SelectedValue == "创建目录"){
4. Directory.CreateDirectory(TextBox1.Text);
5. Label1.Text = "创建时间：" + Directory.GetCreationTime(TextBox1.Text) + "
" + "父目录：" +
Directory.GetParent(TextBox1.Text).FullName;
6. }
7. else{
8. if (Directory.Exists(TextBox1.Text)){
9. Directory.Delete(TextBox1.Text);
10. Label1.Text = "目录删除成功！";
11. }
12. Label1.Text = "此目录不存在!";
13. }

```

```

14. }
15. catch (Exception e1) {
16. Label1.Text = e1.Message;
17. }
18. }

```

代码说明：第 1 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 3 行判断如果创建目录的单选按钮被选中，则第 4 行调用 Directory 类的 CreateDirectory 方法创建文本框中输入的目录。第 5 行在标签控件上显示创建的时间和创建目录的父目录。第 7 行判断如果删除目录的单选按钮被选中，则第 8 行首先判断要删除的目录是否存在，如果存在则第 9 行调用 Directory 类的 Delete 方法删除该目录，第 10 行在标签控件上显示删除成功的文字。否则第 12 行在标签控件上显示目录不存在的提示。第 15 行如果操作过程中出现异常情况，第 16 行处理异常将异常对象 e1 的信息显示在标签控件上。

**04** 按下“Ctrl+F5”，运行结果如图 9-3 所示。

**05** 如果用户选择“删除目录”的单选按钮，运行结果如图 9-4 所示。

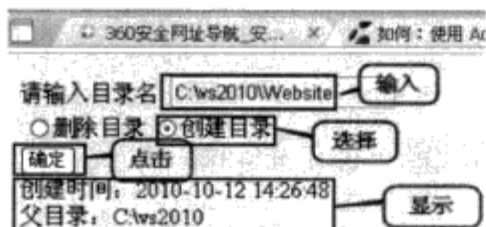


图 9-3 运行结果 1

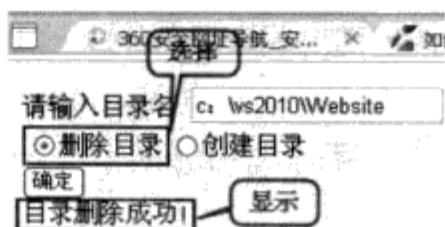


图 9-4 运行结果 2

## 9.2.2 DirectoryInfo 类

DirectoryInfo 类表示驱动器上的物理目录，DirectoryInfo 类和 Directory 类一样都包含了很多对目录进行操作的方法和属性，但是与 Directory 类不同的是，这些方法和属性不是静态的，需要实例化类的对象，将其和特定的目录联系起来。DirectoryInfo 类的构造函数声明如下：

```

1. public DirectoryInfo(
2. string path
3.)

```

以上代码，第 1 行定义了一个 DirectoryInfo 类的构造函数，它带有一个参数就是第 2 行的字符串对象 path，用来指定要在其中创建 DirectoryInfo 的路径。比如下面的代码创建一个与目录 D:\vs2010\Website 对应的 DirectoryInfo 实例对象。

```
DirectoryInfo di=new DirectoryInfo(@"D:\vs2010\Website");
```

### 1. DirectoryInfo 类的方法

DirectoryInfo 类的常用方法包括以下几种。

#### (1) Create 方法

Create 是创建目录的方法，该方法的声明代码如下所示。

```
public void Create ();
```

下面的代码演示创建一个名为 D:\vs2010\Website 的目录。

```
1. private void CreateDirectory(){
2. DirectoryInfo di=new DirectoryInfo(@"D:\vs2010\Website");
3. di.Create();
4. }
```

以上代码，第 1 行定义一个私有类型的方法 CreateDirectory。第 2 行创建一个与目录 D:\vs2010\Website 对应的 DirectoryInfo 实例对象 di。第 3 行调用 di 的 Create 方法创建该目录。

## (2) Delete 方法

Delete 是删除目录的方法，该方法的声明代码如下所示。

```
1. public void Delete (
2. bool recursive
3.);
```

以上代码，第 1 行定义一个方法 Delete。第 2 行中的参数 recursive 用来指定是否删除非空目录。如该参数为 true，将删除整个目录，即使该目录下有文件或子目录；如果参数为 false，则仅在目录为空时才可以删除，如果目录不为空则会引发异常。如果不指定 recursive，则默认为 false。

下面的代码示例将 D:\vs2010 文件夹下创建名为 Website 目录删除。

```
1. private void DeleteDirectory (){
2. DirectoryInfo di=new DirectoryInfo(@"D:\vs2010\Website");
3. Directory.Delete(true);
4. }
```

以上代码，第 1 行定义一个私有类型的方法 DeleteDirectory。第 2 行创建一个与目录 D:\vs2010\Website 对应的 DirectoryInfo 实例对象 di。第 3 行调用 di 的 Delete 方法删除该目录。

## (3) MoveTo 方法

MoveTo 是删除目录的方法，该方法的声明代码如下所示。

```
1. public void MoveTo (
2. string destDirName
3.);
```

以上代码，第 1 行定义一个方法 MoveTo。第 2 行中字符串类型的参数 destDirName，用来指定将要此目录移动到的目标位置的名称和路径。目标不能是另一个具有相同名称的目录。它可以是你将要此目录作为子目录添加到其中的一个现有目录。

下面的代码实现将目录 D:\vs2010 文件夹下名为 Website 目录移动到 C:\vs2010 文件夹下名为 Object 的目录。

```
1. private void MoveDirectory (){
2. DirectoryInfo di=new DirectoryInfo(@"D:\vs2010\Website");
3. di.MoveTo(@"C:\vs2010\Object");
4. }
```

以上代码，第 1 行定义一个私有类型的方法 MoveDirectory。第 2 行创建一个与目录 D:\vs2010\Website 对应的 DirectoryInfo 实例对象 di。第 3 行调用 di 的 MoveTo 方法移动到目标目

录 C:\vs2010\Object。



DirectoryInfo.MoveTo 方法不能将目录移动到一个已经存在的目录中，比如：如果试图将 D:\music 移动到 C:\music，而 C:\music 已经存在，则会引发异常。

提示

#### (4) CreateSubdirectory 方法

CreateSubdirectory 是在指定路径中创建一个或多个子目录的方法，该方法的声明代码如下所示。

```
1. public DirectoryInfo CreateSubdirectory (
2. string path
3.);
```

以上代码，第 1 行定义一个方法 CreateSubdirectory，返回值是在 path 中指定的最后一个目录。第 2 行中字符串类型的参数 path 用来指定子目录的路径，如果 path 所指定的子目录已经存在，则此时该方法不执行任何的操作。

下面的代码在 D:\vs2010 文件夹下创建名为 WebSite 的子目录。

```
1. private void CreateSubdirectory () {
2. DirectoryInfo di=new DirectoryInfo(@"D:\vs2010");
3. di.CreateSubdirectory ("WebSite");
4. }
```

以上代码，第 1 行定义一个私有类型的方法 CreateSubdirectory。第 2 行创建一个与目录 D:\vs2010 对应的 DirectoryInfo 实例对象 di。第 3 行调用 di 的 CreateSubdirectory 方法创建子目录 WebSite。

#### (5) GetFiles 方法

GetFiles 是返回当前目录的文件列表的方法，它有两个重载的方法，其声明代码分别如下所示。

```
1. public FileInfo[] GetFiles ();
2. public FileInfo[] GetFiles (
3. string searchPattern
4.);
```

以上代码，第 1 行定义的是不带参数的 GetFiles 方法，返回一个 FileInfo 类型的数组，其中包含了 DirectoryInfo 目录下所有的文件。第 2 行定义的是带一个参数的 GetFiles 方法，第 3 行字符串参数 searchPattern 用来指定搜索字符串，允许使用通配符。搜索出的目录文件列表以 FileInfo 类型的数组返回。

#### (6) GetDirectories 方法

GetDirectories 是返回当前目录子目录的方法，它也有两种重载的方法，其声明代码分别如下所示。

```
1. public DirectoryInfo[] GetDirectories ();
2. public DirectoryInfo[] GetDirectories (
3. string searchPattern
4.);
```

以上代码，第 1 行定义的是不带参数的 GetDirectories 方法，返回一个 FileInfo 类型的数组其中包含了 DirectoryInfo 目录下所有的子目录。第 2 行定义的是带一个参数的 GetDirectories 方法，第 3 行字符串参数 searchPattern 用来指定搜索字符串，允许使用通配符。搜索出的子目录以 FileInfo 类型的数组返回。

2. DircetoryInfo 类的属性

DirectoryInfo 类的主要属性有以下几种。

(1) Attributes 属性

Attributes 是获取和设置目录的属性，其使用 FileAttributes 枚举类型来获取和设置属性。FileAttributes 枚举提供文件和目录的属性，所包含的成员如表 9-2 所示。

表 9-2 FileAttributes 枚举成员

成员名称	说明
Archive	文件的存档状态。应用程序使用此属性为文件加上备份或移除标记
Compressed	文件已压缩
Device	保留供将来使用
Directory	文件为一个目录
Encrypted	该文件或目录是加密的。对于文件来说，表示文件中的所有数据都是加密的。对于目录来说，表示新创建的文件和目录在默认情况下是加密的
Hidden	文件是隐藏的，因此没有包括在普通的目录列表中
Normal	文件正常，没有设置其他的属性。此属性仅在单独使用时有效
NotContentIndexed	操作系统的内容索引服务不会创建此文件的索引
Offline	文件已脱机。文件数据不能立即提供使用
ReadOnly	文件为只读
ReparsePoint	文件包含一个重新分析点，它是一个与文件或目录关联的用户定义的数据块
SparseFile	文件为稀疏文件。稀疏文件一般是数据通常为零的大文件
System	文件为系统文件。文件是操作系统的一部分或由操作系统以独占方式使用
Temporary	文件是临时文件。文件系统试图将所有数据保留在内存中以便更快地访问，而不是将数据刷新回大容量存储器中。不再需要临时文件时，应用程序会立即将其删除

下面的代码设置 D:\vs2010\WebSite 目录为只读且隐藏。与文件属性相同，目录属性也是使用 FileAttributes 来进行设置的。

```
1. private void SetDirectory(){
2. DirectoryInfo di=new DirectoryInfo(@"D:\vs2010\WebSite");
3. Di. Attributes=FileAttributes.ReadOnly| FileAttributes.Hidden;
4. }
```

以上代码，第 1 行定义一个私有类型的方法 SetDirectory。第 2 行创建一个与目录 D:\vs2010\WebSite 对应的 DirectoryInfo 实例对象 di。第 3 行调用 di 的 Attributes 属性设置目录为只读并且隐藏。

## (2) CreationTime 属性

CreationTime 是获取目录创建时间的属性。下面的代码将返回目录 D:\vs2010\WebSite 的创建时间。

```
1. 0private void CreationTime (){
2. DirectoryInfo di=new DirectoryInfo(@"D:\vs2010\WebSite");
3. DateTime time=di.CreationTime;
4. }
```

以上代码，第 1 行定义一个私有类型的方法 CreationTime。第 2 行创建一个与目录 D:\vs2010\WebSite 对应的 DirectoryInfo 实例对象 di。第 3 行调用 di 的 CreationTime 属性获得创建目录的时间赋给 DateTime 类型的对象 time。

## (3) FullName 和 Name 属性

FullName 和 Name 都是获取目录名称的属性，Name 属性仅返回的是目录的名称，而 FullName 则可以返回目录的完整路径。下面的代码实现返回目录 C:\Program Files\360 的两种不同的名称。

```
1. DirectoryInfo dir = new DirectoryInfo(@"C:\Program Files\360");
2. string dirName = dir.Name;
3. string name = dir.FullName;
```

以上代码，第 1 行创建一个与目录 C:\Program Files\360 对应的 DirectoryInfo 实例对象 dir。第 3 行调用 dir 的 Name 属性获得创建目录的名称。第 4 行调用 dir 的 FullName 属性获得创建目录的完整物理路径。

## (4) Parent 属性

Parent 是获取指定子目录的父目录属性，如果目录不存在或者指定的目录是根目录的父目录（如“\”、“C”或“\\server\share”），则返回值为空引用“Null”。下面的代码实现返回目录 D:\vs2010\WebSite 的父目录 D:\vs2010。

```
1. DirectoryInfo di = new DirectoryInfo(@"D:\vs2010\WebSite");
2. DirectoryInfo pdir=di.Parent;
```

以上代码，第 1 行创建一个与目录 D:\vs2010\WebSite 对应的 DirectoryInfo 实例对象 di。第 2 行通过调用 di 对象的属性 Parent 获得父目录并赋给一个 DirectoryInfo 类型对象 pdir。

## (5) Root 属性

Root 是返回指定目录根目录的属性，该属性是一个只读的属性。下面的代码实现返回目录 D:\vs2010\WebSite 的根目录 D:\。

```
1. DirectoryInfo di = new DirectoryInfo(@"D:\vs2010\WebSite");
2. DirectoryInfo pdir=di.Root;
3. string str=pdir.FullName;
```

以上代码，第 1 行创建一个与目录 D:\vs2010\WebSite 对应的 DirectoryInfo 实例对象 di。第 2 行通过调用 di 对象的属性 Root 获得根目录并赋给 DirectoryInfo 类型的对象 pdir。第 3 行调用 pdir 的 FullName 属性获得根目录的完整路径并赋给一个字符串对象 str。



本章中所有类的成员，包括方法、属性和事件中涉及的路径名称最长不能超过 249 个字符。

**【例 9-4】** 本例实现当用户输入要检索的文件夹路径并单击“检索”按钮时，下面的列表中会显示出该文件夹中所有文件及目录。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 9-4”。

**02** 用鼠标双击网站的目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动 1 个 TextBox 控件、1 个 Button 控件 1 个 ListBox 和 1 个 Label 控件。切换到“源视图”，在编辑区中的<form>和</form>标记间编写如下代码。

```
1. <asp:Label ID="Label3" runat="server" Font-Size="9pt" Text="文件夹路径"></asp:Label>
2. <asp:TextBox ID="TextBox1" runat="server" Font-Size="9pt" Width="144px"></asp:TextBox>
3. <asp:Button ID="Button1" runat="server" Font-Size="9pt" OnClick="Button1_Click" Text="检索" Width="80px" />

4. <asp:Label ID="Label4" runat="server" Font-Size="9pt" Text="文件和目录"></asp:Label>
5. <asp:ListBox ID="ListBox1" runat="server" Height="176px" Width="150px"></asp:ListBox>
6.

7. <asp:Label ID="Label1" runat="server" ></asp:Label>
```

代码说明：第 1 行添加一个服务器标签控件 Label3 并设置显示的文字和大小。第 2 行添加一个服务器文本框控件并设置其宽度和文字的大小。第 3 行添加一个服务器按钮控件并设置其显示的文字和大小以及处理单击事件 Click。第 4 行添加一个服务器标签控件 Label4 并设置其显示的文字和大小。第 5 行添加一个服务器列表控件 ListBox1 并设置其高度和宽度。第 6 行添加一个服务器标签控件 Label1。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Button1_Click(object sender, EventArgs e){
2. if (Directory.Exists(TextBox1.Text) == false){
3. Label1.Text = "该文件不存在或路径错误! ";
4. return;
5. }
6. else{
7. DirectoryInfo di = new DirectoryInfo(TextBox1.Text);
8. FileSystemInfo[] dis = di.GetFileSystemInfos();
9. if (dis.Length < 1){
10. Label1.Text = "该文件夹是空文件夹! ";
11. }
12. else{
13. ListBox1.DataSource = dis;
14. ListBox1.DataBind();
15. Label1.Text = "检索成功，以上为该路径的文件和目录列表! ";
16. }
17. }
18. }
```

代码说明：第 1 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 2 行判断如果用户输入的目录不存在，则在第 3 行的 Label1 标签上显示“该文件夹是空文件夹！”的提示，否则第 7 行创建一个 DirectoryInfo 类型的实例 di 将用户输入的目录作为参数。第 8 行调用 di 的 GetFileSystemInfos 的方法获得指定目录中的所有文件和目录名，并赋给 FileSystemInfo 对象 dis

的数组。第 9 行判断 `dis` 中如果没有文件，则第 10 行在标签控件 `Label1` 上显示提示文件夹为空，否则第 13 行将 `FileSystemInfo` 数组对象 `dis` 作为列表控件的数据源。第 14 行使用列表控件的 `DataBind` 绑定数据的方法将数据显示出来。第 15 行在标签控件 `Label1` 上显示检索成功的提示。

**04** 按下“Ctrl+F5”，运行结果如图 9-5 所示。



图 9-5 运行结果

### 9.3 文件的操作

文件的操作有很多种，比如创建文件、复制文件、删除文件等，这些都是文件最基本的操作。在.NET 4.0 框架中可以使用 `System.IO` 命名空间中的 `File` 和 `FileInfo` 类。这两个类表示文件系统上的文件信息。

#### 9.3.1 File 类

`File` 类是一个静态的类，提供了许多用于处理文件的静态方法，如复制、移动、重命名、创建、打开及删除文件。也可以将 `File` 类用于获取和设置文件属性或有关文件创建、访问及写入操作的 `DateTime` 信息等。`File` 类的主要静态方法有以下数种：

##### 1. Open 方法

`Open` 是打开文件的方法，其声明的代码如下：

```
1. public static FileStream Open (
2. string path,
3. FileMode mode,
4. FileAccess access,
5.);
```

以上代码，第 1 行定义一个静态的方法 `Open`，它有一个返回值 `FileStream` 代表指定路径上的文件流。该方法有 3 个参数，第 1 行的 `path` 用来指定要打开文件的路径。第 2 行的 `Mode` 是一个 `FileMode` 枚举类型，用于指定在文件不存在时是否创建该文件，并确定是保留还是改写现有文件的内容。第 4 行的 `Access` 是一个 `FileAccess` 枚举类型，用于指定可以对文件执行的操作。下面表 9-3 和表 9-4 中分别列出了 `FileAccess` 和 `FileMode` 的成员。

表 9-3 FileAccess 的成员

成员名称	说明
Read	对文件的读访问。可从文件中读取数据。同 Write 组合即构成读写访问权
ReadWrite	对文件的读访问和写访问。可从文件读取数据和将数据写入文件
Write	文件的写访问。可将数据写入文件。同 Read 组合即构成读/写访问权

表 9-4 FileMode 的成员

成员名称	说明
Append	打开现有文件并查找到文件尾，或创建新文件。FileMode.Append 只能同 FileAccess.Write 一起使用。任何读尝试都将失败并引发 ArgumentException
Create	指定操作系统应创建新文件。如果文件已存在，它将被改写。这要求 FileIOPermissionAccess.Write。System.IO.FileMode.Create 等效于这样的请求：如果文件不存在，则使用 CreateNew；否则使用 Truncate
CreateNew	指定操作系统应创建新文件。此操作需要 FileIOPermissionAccess.Write。如果文件已存在，则将引发 IOException
Open	指定操作系统应打开现有文件。打开文件的能力取决于 FileAccess 指定的值。如果该文件不存在，则引发 System.IO.FileNotFoundException
OpenOrCreate	指定操作系统应打开文件（如果文件存在）；否则，应创建新文件。如果用 FileAccess.Read 打开文件，则需要 FileIOPermissionAccess.Read。如果文件访问为 FileAccess.Write 或 FileAccess.ReadWrite，则需要 FileIOPermissionAccess.Write。如果文件访问为 FileAccess.Append，则需要 FileIOPermissionAccess.Append
Truncate	指定操作系统应打开现有文件。文件一旦打开，就将被截断为零字节大小。此操作需要 FileIOPermissionAccess.Write。试图从使用 Truncate 打开的文件中进行读取将导致异常

下面的代码将打开存放在 C:\vs2010 目录下名为 New.txt 的文件。

```
1. private void OpenFile(){
2. File.Open(@"C:\vs2010\New.txt",FileMode.Append,FileAccess.Read);
3. }
```

以上代码，第 1 行定义一个私有类型的方法 OpenFile。第 2 行通 File 类的静态方法 Open 打开存放在 C:\vs2010 目录下名为 New.txt 的文件。

2. Create 方法

Create 是创建一个新文件的方法，它的声明代码如下所示。

```
1. public static FileStream Create (
2. string path
3.);
```

以上代码，第 1 行定义一个静态的方法 Create。第 2 行参数 path 用来指定要创建文件的路径和名称。如果 path 指定的文件不存在，则创建该文件；如果存在并且不是只读的，则将改写其内容。

下面的代码将实现如何在 C:\vs2010 目录下创建名为 New.txt 的文件。

```
1. private void CreateFile(){
2. FileStream fs=File.Create(@"C:\vs2010\New.txt");
3. fs.Colse;
4. }
```

以上代码，第 1 行定义一个私有类型的方法 CreateFile。第 2 行通过 File 类的静态方法 Create 在 C:\vs2010 目录下创建一个名为 New.txt 的文件，并赋给 FileStream 类型的对象 fs。第 3 行使用 fs 对象的 Colse 方法关闭所创建的文件。



由于 File.Create 方法默认向所有用户授予对新文件的完全读写访问权限，所以文件是用读写访问权限打开的，必须关闭后才能由其他应用程序打开。为此，需要使用 FileStream 类的 Close 方法将所创建的文件关闭。

### 3. Delete 方法

Delete 是删除指定目录文件的方法，该方法声明的代码如下：

```
1. public static void Delete (
2. string path
3.);
```

以上代码，第 1 行定义一个静态的方法 Delete。第 2 行参数 path 用来指定要创建文件的路径和名称。如果 path 指定的文件不存在，不会引发一个异常。

下面的代码将实现如何在 C:\vs2010 目录下删除名为 New.txt 的文件。

```
1. private void DeleteFile(){
2. File.Delete(@"C:\vs2010\New.txt");
3. }
```

以上代码，第 1 行定义一个私有类型的方法 DeleteFile。第 2 行通过 File 类的静态方法 Delete 删除在 C:\vs2010 目录下的名为 New.txt 的文件。

### 4. Copy 方法

Copy 是将现有文件复制到新文件的方法，该方法声明的代码如下：

```
1. public static void Copy (
2. string sourceFileName,
3. string destFileName,
4. bool overwrite
5.);
```

以上代码，第 1 行定义一个静态的方法 Copy。其中，第 2 行和第 3 行的参数 sourceFileName 和 destFileName 分别用来指定要复制的源文件和目标文件的名称。第 4 行的参数 overwrite 用来指定如果目标文件已经存在是否要覆盖它，是为 true，否为 false。

下面的代码演示将 C:\vs2010\New.txt 文件复制为 D:\vs2010\New.txt。

```
1. private void CopyFile(){
2. File.Copy(@"C:\vs2010\New.txt", @"D:\vs2010\New.txt",true);
3. }
```

以上代码，第 1 行定义一个私有类型的方法 CopyFile。第 2 行通过 File 类的静态方法 Copy 将 C:\vs2010\New.txt 文件复制为 D:\vs2010\New.txt。如果 D:\vs2010 目录中已经存在，将被复制的文件所覆盖。

### 5. Move 方法

Move 是将指定文件移动到新位置的方法，该方法声明的代码如下：

```
1. public static void Move (
2. string sourceFileName,
```

```
3. string destFileName
4.);
```

以上代码，第1行定义一个静态的方法 Move。第2行的参数 sourceFileName 用于指定要移动的文件名称。第3行的参数 destFileName 用于指定文件的新路径。如果源路径和目标路径相同，不会引发异常。

下面的代码实现将 D:\vs2010 下的 New.text 文件移动到 C 盘根目录下。

```
1. private void MoveFile(){
2. File.Move(@"D:\vs2010\New.text", @"C:\New.text");
3. }
```

以上代码，第1行定义一个私有类型的方法 MoveFile。第2行通过 File 类的静态方法 Move 将 D:\vs2010\New.text 文件移动到 C:\ 下。

## 6. SetAttributes 方法

SetAttributes 是设置指定路径上文件属性的方法，该方法声明的代码如下：

```
1. public static void SetAttributes (
2. string path,
3. FileAttributes fileAttributes
4.);
```

以上代码，第1行定义一个静态的方法 SetAttributes。第2行的参数 path 用来指定文件的路径。第3行的参数 fileAttributes 用于指定所需的 FileAttributes，比如 Hidden、ReadOnly、Normal 或 Archive。FileAttributes 的成员请参看表 9-1。

下面的代码实现设置文件 D:\vs2010\New.text 的属性为只读且隐藏。

```
1. private void SetFile(){
2. File.SetAttributes(@"D:\vs2010\New.text", FileAttributes.ReadOnly|FileAttributes.Hidden);
3. }
```

以上代码，第1行定义一个私有类型的方法 SetFile。第2行通过 File 类的静态方法 SetAttributes 设置 D:\vs2010\New.text 文件为只读并且隐藏。

## 7. Exists 方法

Exist 是判断指定的文件是否存在的方法，该方法声明的代码如下：

```
1. public static bool Exists (
2. string path
3.);
```

以上代码，第1行定义一个静态的 Exists 方法，该方法返回一个布尔类型的值，如果文件存在返回值为 true，否则返回值为 false。第2行的参数 path 指定要检查的文件。

下面的代码实现判断 D:\vs2010 文件夹下名为 Website 的文件是否存在，如果存在则复制文件然后将其删除。否则创建文件并打开。

```
1. private void Handle(){
2. if(File.Exists(@"D:\vs2010\New.text")){
3. CopyFile();
4. DeleteFile();
```

```

5. }
6. else{
7. Create.File();!
8. Open.File();
9. }
10. }

```

以上代码，第 1 行定义一个私有类型的方法 `Handle`。第 2 行通过 `File` 类的静态方法 `Exists` 判断 `D:\vs2010\New.text` 文件是否存在，如果存在，则第 3 行调用 `CopyFile` 复制该文件。第 4 行调用 `DeleteFile` 方法删除源文件。否则，第 7 行调用 `Create.File` 创建该文件。第 8 行调用 `Open.File` 方法打开新创建的文件。

### 9.3.2 FileInfo 类

`FileInfo` 类不是静态类，没有静态方法，仅可用于实例化的对象。`FileInfo` 对象表示磁盘或网络位置的物理文件。只要提供文件的路径就可以创建一个 `FileInfo` 对象，如以下代码。

```
FileInfo fi=new FileInfo(@"D:\vs2010\New.text");
```

`FileInfo` 类提供了许多类似于 `File` 类的方法，但是因为 `File` 类是静态类，需要一个字符串参数为每个方法调用指定文件的位置。下面的代码使用 `FileInfo` 类实现用来检查文件 `D:\vs2010\New.text` 是否存在，请大家区别它和 `File` 类的使用。

```

1. FileInfo fi=new FileInfo(@"D:\vs2010\New.text");
2. if(fi.Exists){
3. Response.Write("文件存在！");
4. }

```

以上代码，第 1 行实例化一个 `FileInfo` 类的对象 `fi`，并提供了文件路径。第 2 行调 `fi` 的方法 `Exists` 判断该文件是否存在。第 3 行在页面上显示提示文字。

`FileInfo` 类中常用方法和属性说明如下。

#### 1. Open 方法

`Open` 是打开文件方法，其声明的代码如下所示。

```

1. public FileStream Open (
2. FileMode mode,
3. FileAccess access
4.);

```

以上代码，第 1 行定义一个 `Open` 的方法，它有一个返回值 `FileStream` 代表指定路径上的文件流。该方法有 2 参数，第 2 行的 `Mode` 是一个 `FileMode` 枚举类型，用于指定在文件不存在时是否创建该文件，并确定是保留还是改写现有文件的内容。第 3 行的 `Access` 是一个 `FileAccess` 枚举类型，用于指定可以对文件执行的操作。可以看出，`FileInfo.Open` 方法只比 `File.Open` 方法少了一个参数 `path`，这是因为在实例化 `FileInfo` 时就已经给出了 `path`。

下面的代码使用 `FileInfo.Open` 方法打开存放在 `D:\vs2010` 目录下的 `New.text` 文件。

```

1. private void OpenFile(){
2. FileInfo fi=new FileInfo(@"D:\vs2010\New.text");

```

```
3. FileStream fs=fi.Open(FileMode.Append,FileAccess.Append)
4. }
```

以上代码：第 1 行定义一个私有类型的方法 `OpenFile`。第 2 行实例化一个 `FileInfo` 类的对象 `fi`，并提供了文件路径。第 3 行调用 `fi` 的 `Open` 方法打开文件并赋给一个 `FileStream` 类型的对象 `fs`。

2. `FileInfo` 类的其他方法

`FileInfo` 中的其它方法和 `Open` 的方法用法相似，表 9-5 中列出了它们的名称和用途。

表 9-5 `FileInfo` 类的其它方法

方法名称	说明
CopyTo	已重载。将现有文件复制到新文件
Create	创建文件
Delete	永久删除文件
MoveTo	将指定文件移到新位置，并提供指定新文件名的选项
Open	用各种读/写访问权限和共享特权打开文件

3. `FileInfo` 类的属性

另外，`FileInfo` 类也提供了与文件相关的属性，`FileInfo` 类的属性如表 9-6 所示，这些属性可以用来获取或更新文件的信息。

表 9-6 `FileInfo` 类的属性

属性	说明
Attributes	获取或设置当前 <code>FileSystemInfo</code> 的 <code>FileAttributes</code>
CreationTime	获取或设置当前 <code>FileSystemInfo</code> 对象的创建时间
Directory	获取父目录的实例
DirectoryName	获取表示目录的完整路径的字符串
Exists	获取指示文件是否存在的值
Extension	获取表示文件扩展名部分的字符串
FullName	获取文件的完整目录
IsReadOnly	获取或设置确定当前文件是否为只读的值
LastAccessTime	获取或设置上次访问当前文件或目录的时间
LastWriteTime	获取或设置上次写入当前文件或目录的时间
Length	获取当前文件的大小
Name	获取文件名



在使用 `FileInfo` 类和 `File` 类前，必须在 `Default.aspx.cs` 后台代码文件的命名空间区域引用 `using System.IO` 的命名空间。

【例 9-5】本例实现在应用程序的两个文件夹中进行创建文件、复制文件、移动文件和删除文件的操作。

- 01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 9-5”。
- 02** 在应用程序中创建两个文件夹, 一个命名为“Files”, 另一个命名为 File2。
- 03** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入的“视图编辑界面”, 从工具箱中拖动 3 个 Label 控件、1 个 TextBox 控件、1 个 DropDownList 控件、1 个 ListBox 控件和 4 个 Button 控件。
- 04** 用鼠标双击网站目录下的“Default.aspx.cs”文件, 编写代码如下:

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. List();
4. }
5. }
6. protected void Button1_Click(object sender, EventArgs e){
7. if (TextBox2.Text == ""){
8. Label2.Text = "请输入文件名! ";
9. }
10. else {
11. try{
12. string path = Server.MapPath("File") + "\\" + TextBox2.Text + DropDownList1.Text;
13. FileInfo fi = new FileInfo(path);
14. if (!fi.Exists) {
15. fi.Create();
16. Label2.Text = "创建成功! 文件名: " + TextBox2.Text + DropDownList1.Text;
17. List();
18. }
19. } catch (Exception error) {
20. Response.Write(error.ToString());
21. }
22. }
23. }
24. protected void Button2_Click(object sender, EventArgs e){
25. try{
26. string path = Server.MapPath("File/") + Session["txt"];
27. string path2 = Server.MapPath("File/") + "复制" + Session["txt"];
28. FileInfo fi = new FileInfo(path);
29. if (fi.Exists){
30. fi.CopyTo(path2);
31. }
32. Label2.Text = "复制" + Session["txt"] + "成功! " + "文件为:" + ("复制" + Session["txt"].ToString());
33. List();
34. } catch (Exception error){
35. Label2.Text = "复制文件出错, 该文件已被复制过! ";
36. }
37. }
38. protected void Button3_Click(object sender, EventArgs e){
39. if (Session["txt"] == null){
40. Label2.Text = "请选中文件后在执行删除操作! ";
41. }
42. FileInfo fi = new FileInfo(Server.MapPath("File/" + Session["txt"]));
43. if (fi.Exists){
44. fi.Delete();
45. Label2.Text = "删除" + Session["txt"] + "文件成功! ";
46. List();
47. Session.Clear();
48. }
49. }

```

```

50. protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e) {
51. Session["txt"] = ListBox1.SelectedValue.ToString();
52. }
53. public void List(){
54. DataTable dt = new DataTable();
55. dt.Columns.Add(new DataColumn("Name", typeof(string)));
56. string serverPath = Server.MapPath("File");
57. DirectoryInfo dir = new DirectoryInfo(serverPath);
58. foreach (FileInfo fileName in dir.GetFiles()){
59. DataRow dr = dt.NewRow();
60. dr[0] = fileName;
61. dt.Rows.Add(dr);
62. }
63. ListBox1.DataSource = dt;
64. ListBox1.DataTextField = "Name";
65. ListBox1.DataValueField = "Name";
66. ListBox1.SelectedIndex = 0;
67. ListBox1.DataBind();
68. }
69. protected void Button4_Click(object sender, EventArgs e){
70. string path = Server.MapPath("File/") + ListBox1.SelectedValue.ToString();
71. FileInfo fi = new FileInfo(path);
72. if (fi.Exists){
73. string path2 = Server.MapPath("File2/") + ListBox1.SelectedValue.ToString();
74. fi.MoveTo(path2); //将指定文件夹路径中的文件移动到另一个路径中的文件夹
75. List();
76. Label2.Text = "移动" + Session["txt"] + "文件成功! ";
77. }
78. }

```

代码说明：第1行定义处理页面 Page 加载事件 Load 的方法。第2行判断如果加载的页面不是回传页面，则第3行调用自定义的 List 方法绑定数据。第6行定义处理创建按钮 Button1 单击事件 Click 的方法。第7行判断如果用户没有输入文件名称，则在标签控件 Label2 上显示提示信息。否则，第12行获取用户输入的文件名和文件类型。第13行创建 FileInfo 对象 fi 并设置保存文件的路径。第14行判断文件是否存在。第15行调用 Create 方法创建此文件。第16行在标签控件 Label2 上显示创建成功的提示。第17行调用 List 方法绑定数据。

第24行定义处理复制按钮 Button2 单击事件 Click 的方法。第26行设置源文件的路径，第27行设置目标文件的路径。第28行创建 FileInfo 对象 fi 并将设置保存源文件的路径。第29行判断文件已经存在，第30行调用 Copy 方法复制文件。第32行在标签控件上显示复制成功的提示。第33行调用 List 方法绑定数据。

第38行定义处理删除按钮 Button3 单击事件 Click 的方法。第39行判断 Session 值如果空，则第40行在标签控件上提示用户选择要操作的文件。第42行创建 FileInfo 对象 fi 并将设置删除文件的路径。第43行判断文件已经存在，第44行调用 Delete 方法复制文件。第45行在标签控件上显示删除成功的提示。第46行调用 List 方法绑定数据。第47行将 Session 对象中的值清空。第50行定义处理列表控件 ListBox1 选中项更改事件 SelectedIndexChanged 的方法。第51行将选中项的值保存到 Session 对象中。

第53行自定义一个绑定数据的方法 List。第54行实例化一个 DataTable 对象 dt。第55行创建一个对象 DataColumn 并添加到 dt 的列集合中。第56行获取“File”目录的路径。第57行创建一个 DirectoryInfo 类的 dir。第58~62行使用 foreach 循环遍历“File”目录下所有的文件，将文件

名作为 DataTable 中每一列的值添加到 dt 对象行集合中。第 63 行将 dt 表对象作为列表控件 ListBox1 的数据源。第 64 和 65 行分别获得绑定字段的名称和值。第 66 行将列表控件选中项的值设置为 0。第 67 行调用 DataBind 方法绑定数据到列表控件 ListBox1。

第 69 行定义处理移动按钮 Button4 单击事件 Click 的方法。第 70 行获取要移动文件的路径。第 71 行创建 FileInfo 对象 fi 并将保存文件的路径。第 72 行判断文件已经存在。第 73 行获得要移动到的目标路径。第 74 行调用 fi 的 MoveTo 方法移动文件到目标路径。第 75 行调用 List 方法绑定数据。第 76 行在标签控件上显示移动成功的提示。

**05** 按下“Ctrl+F5”，运行结果如图 9-6 所示。



图 9-6 运行结果

## 9.4 读写文件

本节将介绍用于读写文件的类，这些类表示一个通用的概念——流。读写文件主要用到的流有 FileStream 类、StreamWriter 和 StreamReader 对象。

### 9.4.1 流

在 .NET 4.0 框架中进行所有的输入和输出工作都要用到流。流是一个用于传输数据的对象，数据的传输有两个方向，对应着两种类型的流。

- 输出流：用于将数据从程序传输到外部源，这里的外部源可以是物理磁盘文件，网络位置、打印机或另一个程序。
- 输入流：用于将数据从外部源传输到程序中，这里的外部源有键盘、磁盘文件等。

对应文件的读写，最常用的类有以下两种。

- FileStream（文件流）：主要用于二进制文件中读写二进制数据，也可以用于读写任何文件。
- StreamReader（流读取器）和 StreamWriter（流写入器）：专门用于读写文本文件。

### 9.4.2 FileStream 类

FileStream 类表示在磁盘上或网络路径上指定的文件流。这个类提供了在文件中读写二进制数据的方法。FileStream 类的构造函数如下。

```
1. public FileStream(
2. string path,
3. FileMode mode,
4. FileAccess access,
5.);
```

以上代码，第 1 行定义 FileStream 构造函数。第 2 行的参数 path 用来指定要访问的文件。第 3 行的参数 mode 是 FileMode 类型的一个枚举成员，用于指定打开文件的模式，关于 FileMode 的成员可以参考上文的表 9-4。第 4 行的参数 access 是 FileAccess 枚举的一个成员，用于指定访问文件的方式，关于 FileAccess 枚举的成员参见上文的表 9-3。如果在构造 FileStream 对象的时候没有指

定 `FileAccess` 参数，则默认为 `FileAccess.ReadWrite`（读写）。

使用完一个流后，应该使用 `FileStream.Close` 方法将其关闭。关闭流会释放与它相关的资源，允许其他的应用程序为同一个文件设置流。在打开和关闭流之间，可以读写其中的数据，`FileStream` 有许多的方法都可以进行文件的读写，下面介绍用得最多的三个方法。

### 1. Read 方法

`Read` 是从 `FileStream` 对象所指定的文件中读取数据的主要方法，该方法的声明代码如下。

```
1. public override int Read(
2. byte[] array,
3. int offset,
4. int count
5.);
```

以上代码，第 1 行定义一个重载的方法 `Read`，它有一个 `int` 类型的返回值表示读入缓冲区中的总字节数。如果当前的字节数没有所请求那么多，则总字节数可能小于所请求的字节数；如果已到达流的末尾，则为零。第 2 行的参数 `array` 是一个字节数组，此方法返回时包含指定的字节数组，数组中 `offset` 和 `(offset + count - 1)` 之间的值由从当前源中读取的字节替换。第 2 个参数 `offset` 表示 `array` 中的字节偏移量，从此处开始读取。第 3 行的参数 `count` 表示从文件中读取的字节数。

`FileStream` 对象只能处理二进制数据，这使得 `FileStream` 可以用于读写任何的文件，但是这使得 `FileStream` 不能直接读取字符串。然而，可以通过几种转换类把字节数组转换为字符串，或者将字符串转换成字节数组。`System.Text` 命名空间中的 `Decoder` 类，可以实现这种转换，比如以下的代码。

```
1. Decoder d=Encoding.UTF8.GetDecoder();
2. d.GetChars(byData,0,byData.Length,charData,0);
```

以上代码，第 1 行通过 `Encoding` 的 `UTF8.GetDecoder` 方法创建一个基于 `UTF8` 编码模式的 `Decoder` 对象。第 2 行调用 `GetChars` 方法将指定的字节数组转换为字符数组。

### 2. Write 方法

`Write` 是使用从缓冲区读取的数据将字节块写入流的方法。写入数据与读取数据非常类似，首先将要写入的内容存入一个字符数组中，然后利用 `System.Text.Encoder` 对象将其转换为一个字节数组，最后调用 `Write` 方法将字节数组写入文件中去。`Write` 方法的声明代码如下。

```
1. public override void Write(
2. byte[] array,
3. int offset,
4. int count
5.);
```

以上代码，第 1 行定义一个重载的方法 `Write`。它有三个参数，具体的含义和作用与 `Read` 方法相同，这里不再重复。

### 3. Seek 方法

`Seek` 是将该流的当前位置设置为给定值的方法。对文件进行读写操作的位置是由内部文件的指针决定的。在大多数的情况下，当打开文件时，就指向文件的开始位置，但是此指针是可以修改

的,这使得应用程序可以在文件的任何位置进行读写操作,随机访问文件或跳到文件的指定位置上。这样,当处理大型文件时会非常的省力。Seek 方法的声明代码如下。

```
1. public override long Seek(
2. long offset,
3. SeekOrigin origin
4.);
```

代码说明:第 1 行定义一个重载的方法 Seek,它有一个 long 类型的返回值表示流中的新位置。第 2 行中的参数 offset 用于规定文件指针以字节单位的移动距离;第 3 行中的参数 origin 是 SeekOrigin 枚举的一个成员,用于规定开始计算的起始位置。SeekOrigin 包含了 3 个值:Begin、Current 和 End。下面的代码将文件指针从文件的开始位置移动到文件的第 5 个字节。

```
1. FileStream fs=new FileStream(@"D:\vs2010\New.text");
2. fs.Seek(5, SeekOrigin. Begin);
```

以上代码,第 1 行创建一个文件流 FileStream 的对象 fs 并指定要访问的文件。第 2 行调用 fs 对象的 Seek 方法将文件指针从文件的开始位置移动到文件的第 5 个字节。

不仅如此,还可以指定负查找的位置,当 offset 参数为负时,表示向前移动。比如下面的代码实现将文件指针移动到倒数第 9 个字节。

```
1. FileStream fs=new FileStream(@"D:\vs2010\New.text");
2. fs.Seek(-9, SeekOrigin. End);
```

以上代码,第 1 行创建一个文件流 FileStream 的对象 fs 并指定要访问的文件。第 2 行调用 fs 对象的 Seek 方法将文件指针从文件的结束位置移动到文件的倒数第 9 个字节。



提示

File 类和 FileInfo 类也提供了创建流对象的方法:OpenWrite 和 OpenRead,前者创建一个只读流对象,后者创建一个只写流的对象。

**【例 9-6】**本例将实现从随机访问文件中读取数据,该文件是本程序中的 Default.aspx.cs 文件,从第 203 个字节处开始读取,并将读取的内容显示在页面中。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 9-6”。

**02** 用鼠标双击网站目录下的“Default.aspx.cs”文件,编写代码如下:

```
1. protected void Page_Load(object sender, EventArgs e){
2. byte[] by = new byte[1000];
3. char[] chby = new char[1000];
4. FileStream fs = null;
5. try{
6. fs = new FileStream(Server.MapPath(".") + "\\Default.aspx.cs", FileMode.Open);
7. fs.Seek(203, SeekOrigin.Begin);
8. fs.Read(by, 0,1000);
9. }
10. catch (IOException e1) {
11. Response.Write(e1.Message);
12. }
13. finally{
14. fs.Close();
```

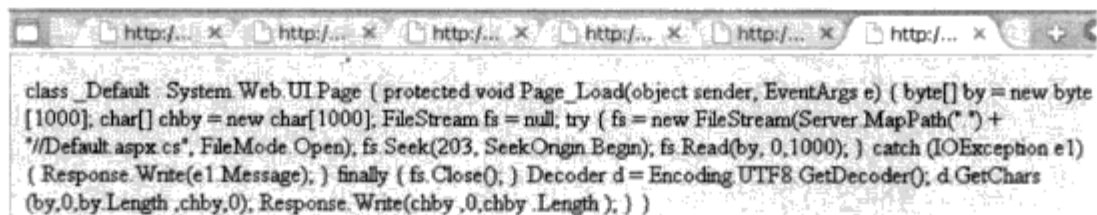
```

15. }
16. Decoder d = Encoding.UTF8.GetDecoder();
17. d.GetChars(by,0,by.Length, chby,0);
18. Response.Write(chby,0,chby.Length);
19. }

```

代码说明：第1行定义处理 Page 页面加载事件 Load 的方法。第2行定义一个字节数组 by 存放读取的文件内容。第3行定义一个字符数组 chby 存放每一个文件的字符。第4行实例化一个空的 FileStream 对象 fs。第5行给 fs 对象指定要读取的文件路径和文件模式。第6行调用 fs 的 Seek 方法将文件指针移动到文件的第203个字节。第7行调用 fs 的 Read 方法读取1000个字节到 by 数组中。第8行如果出现异常将异常信息输出到页面。第9行关闭 fs 对象。第10行通过 Encoding 的 UTF8.GetDecoder 方法创建一个基于 UTF8 编码模式的 Decoder 对象 d。第11行调用 GetChars 方法将指定的字节数组 by 转换为字符数组 chby。第12行将字符数组 chby 中的全部字符输出到页面上。

**03** 按下“Ctrl+F5”，运行结果如图9-7所示。本程序输出的就是自身的.cs文件，由于是从第203个字节处开始读取，所以是 class 中的“c”开始一直读到文件的结束。



```

class Default System.Web.UI.Page (protected void Page_Load(object sender, EventArgs e) { byte[] by = new byte
[1000]; char[] chby = new char[1000]; FileStream fs = null; try { fs = new FileStream(Server.MapPath("/") +
"/Default.aspx.cs", FileMode.Open); fs.Seek(203, SeekOrigin.Begin); fs.Read(by, 0,1000); } catch (IOException e1)
{ Response.Write(e1.Message); } finally { fs.Close(); } Decoder d = Encoding.UTF8.GetDecoder(); d.GetChars
(by,0,by.Length, chby,0); Response.Write(chby,0,chby.Length); })

```

图 9-7 运行结果

### 9.4.3 读写文本文件

除了读写字节的 FileStream 类，在 .NET 4.0 框架中还提供了 StreamWriter 类和 StreamReader 类专门用于处理文本文件。原因是操作二进制的文件比较繁琐，使得 FileStream 的使用相对困难，而使用 StreamWriter 和 StreamReader 对象却能够很方便地顺序访问整个文件。但是它们也有不足之处，那就是不能随意地改变文件指针的位置，无法实现对文件的随机访问。

#### 1. StreamWriter 类

StreamWriter 类允许将字符和字符串写入到文件中。有很多的方法可以用来创建 StreamWriter 对象，如果已经有了 FileStream 对象，则可以使用此对象来创建 StreamWriter 对象，代码如下所示。

```

1. FileStream fs=new FileStream(@"D:\vs2010\WebSite", FileMode.CreateNew);
2. StreamWriter sw=new StreamWriter(fs);

```

以上代码，第1行创建一个 FileStream 类的对象 fs 并指定访问文件的路径。第2行实例化一个 StreamWriter 对象 sw，参数是第1行创建的 fs 对象。

还有一种是直接从文件中创建 StreamWriter 对象的方法，代码如下所示。

```

StreamWriter sw=new StreamWriter(@"D:\vs2010\WebSite",true);

```

以上代码，创建 StreamWriter 对象 sw 使用的构造函数有两个参数，一个是文件名，一个是布尔值，这个布尔值规定了是添加到文件的末尾还是创建新文件：值为 false 时，如果文件存在，则

截取现有文件并打开该文件，否则创建一个新文件；值为 `true` 时，如果文件存在，则打开文件，保留原来的数据，否则创建一个新的文件。

与创建 `FileStream` 对象不同，创建 `StreamWriter` 对象不会提供一组类似的选项，除了使用布尔值时只是添加到文件的末尾或创建新文件之外，根本没有像 `FileStream` 类那样指定 `FileMode`、`FileAccess` 等属性的选项。如果需要使用这些高级参数，可以先在 `FileStream` 的构造函数中指定这些参数，然后利用 `FileStream` 对象来创建 `StreamWriter` 对象。

`StreamWriter` 对象提供了两个用于写入数据的方法——`Write` 和 `WriteLine`，这两个方法有许多的重载版本，可以完成高级的文件输出。`Write` 方法和 `WriteLine` 方法基本上相同，不同的是 `WriteLine` 方法在将传送给它的数据输出后，会再输入一个换行符。表 9-7 列出了 `WriteLine` 的部分重载版本。

表 9-7 WriteLine 方法的重载版本

方法	说明
<code>WriteLine()</code>	将行结束符写入文本流
<code>WriteLine(Boolean)</code>	将后跟行结束符的 <code>Boolean</code> 的文本表示形式写入文本流
<code>WriteLine(Char)</code>	将后跟行结束符的字符写入文本流
<code>WriteLine(Char[])</code>	将后跟行结束符的字符数组写入文本流
<code>WriteLine(Decimal)</code>	将后面带有行结束符的十进制值的文本表示形式写入文本流
<code>WriteLine(Double)</code>	将后跟行结束符的 8 字节浮点值的文本表示形式写入文本流
<code>WriteLine(Int32)</code>	将后跟行结束符的 4 字节有符号整数的文本表示形式写入文本流
<code>WriteLine(Int64)</code>	将后跟行结束符的 8 字节有符号整数的文本表示形式写入文本流
<code>WriteLine(Object)</code>	通过在对象上调用 <code>ToString</code> 将后跟行结束符的此对象的文本表示形式写入文本流
<code>WriteLine(String)</code>	将后跟行结束符的字符串写入文本流

2. StreamReader 类

`StreamReader` 类的工作方式与 `StreamWriter` 类似，但 `StreamReader` 是用于从文件或另一个流中读取数据的。`StreamReader` 对象的创建方式非常类似于 `StreamWriter` 对象，最常见的方式是使用 `StreamWriter` 对象，代码如下所示。

```
1. FileStream fs=new FileStream(@"D:\vs2010\WebSite", FileMode.CreatNew);
2. StreamReader sr=new StreamWriter(fs);
```

以上代码，第 1 行创建一个 `FileStream` 类的对象 `fs` 并指定访问文件的路径。第 2 行实例化一个 `StreamReader` 对象 `sr`，参数是第 1 行创建的 `fs` 对象。

同样，`StreamReader` 类也可以直接使用包含具体文件路径的字符串来创建对象，代码如下。

```
StreamReader sr=new StreamWriter(@"D:\vs2010\WebSite");
```

`StreamReader` 类中提供了常用的几个方法用于读取文件的数据，表 9-8 列出了常用的方法。

表 9-8 StreamReader 类的常用方法

方法	说明
Read()	读取输入流中的下一个字符或下一组字符
ReadLine()	从当前流中读取一行字符并将数据作为字符串返回
ReadToEnd()	从流的当前位置到末尾读取流

**【例 9-7】** 本例实现如何利用 FileInfo 和 StreamWriter 对象实现动态创建文件并输入文件内容的功能。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 9-7”。
- 02 在程序中创建一个文件夹，命名为“Files”。
- 03 用鼠标双击网站的目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动 1 个 Label 控件、2 个 TextBox 控件和 1 个 Button 控件。
- 04 用鼠标双击网站目录下的“Default.aspx.cs”文件，编写代码如下：

```
1. protected void Button1_Click(object sender, EventArgs e){
2. if (TextBox1.Text != "" && TextBox2.Text != ""){
3. string path = Server.MapPath("Files") + "\\" + TextBox1.Text;
4. FileInfo fi = new FileInfo(path);
5. if (!fi.Exists){
6. StreamWriter sw = fi.CreateText();
7. sw.WriteLine(TextBox2.Text);
8. sw.Flush();
9. sw.Close();
10. }
11. Label1.Text = "创建和写入文件成功！";
12. }
13. else{
14. Label1.Text = "请输入文件名和文件类型！";
15. }
16. }
```

代码说明：第 1 行定义按钮控件 Button1 单击事件 Click 的方法。第 2 行判断用户输入的文件名和输入的文件内容为空。第 3 行使用 Server 对象的 MapPath 方法获得创建文件的路径。第 4 行实例化一个 FileInfo 类的对象 fi 并指定文件路径。第 5 行判断如果要创建的文件不存在。就在第 6 行利用 fi 对象的 CreateText 方法实例化一个 StreamWriter 类的对象 sw。第 7 行调用 sw 的 WriteLine 方法将输入的文件内容写入文件。第 8 行调用 sw 的 flushf 方法清空缓冲流。第 9 行关闭当前 StreamWriter 对象。第 11 行在标签控件上显示提示操作成功的文字。第 13 行如果判断用户输入的文件名和输入的文件内容都为空，则在标签控件上显示提示信息。

- 05 按下“Ctrl+F5”，运行结果如图 9-8 所示。
- 06 此时在应用程序的“Files”文件夹中已经创建了“VS 2010”的文本文件，打开文件显示如图 9-9 所示的内容和用户输入完全一致。

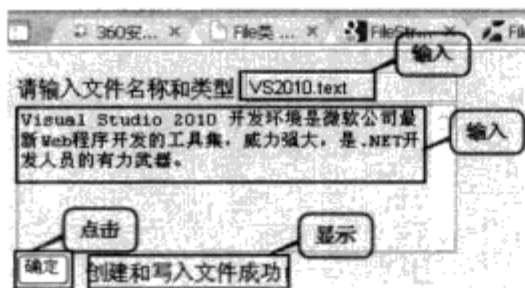


图 9-8 运行结果

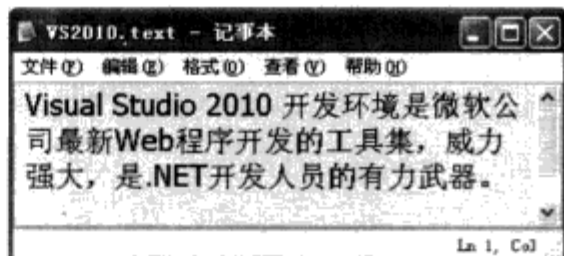


图 9-9 运行结果 2



Read 方法用来返回流中的下一个字符, 如果已经到达流的结尾, 则返回-1。而 RaedLine 方法到达文件的结尾时, 返回的是 null。

## 9.5 上机题

1. 本题实现查询显示本地机器 D 盘驱动器的具体的信息, 程序运行结果如图 9-10 所示。
2. 使用 DirectoryInfo 类遍历指定目录下的所有文件和目录, 程序运行结果如图 9-11 所示。
3. 本题要求动态修改服务器指定目录中的文件名称, 当用户选择要修改的文件后, 就可以在文本框中输入要更新的文件名称, 单击“确定”按钮执行修改文件的操作, 程序运行结果如图 9-12 所示。



图 9-10 运行结果



图 9-11 运行结果



图 9-12 运行结果

4. 实现动态修改文本文件内容的功能, 当用户选择了文件名称, 输入主要修改的文本内容后, 单击“确定更改”按钮, 程序会将更改的文本保存到选定的文件中, 程序运行结果如图 9-13 所示。
5. 本题要求实现在服务器指定文件夹中创建文件的功能, 当用户输入文件名称和内容后, 单击“确定”按钮, 程序会提示写文件成功, 运行结果如图 9-14 所示。
6. 本题要求实现在服务器指定文件夹中读取文件的功能, 当用户选择文件名称后, 单击“读文件”按钮, 程序会在将文件中的内容显示在 TextBox 中, 运行结果如图 9-15 所示。



图 9-13 运行结果



图 9-14 运行结果



图 9-15 运行结果

# 第 10 章 XML 数据操作

## 学习目标

XML 是当前 Web 程序开发的流行技术之一，它的主要作用是成为客户端和服务端之间数据交换的语言，同时还能够在服务器端以数据资源的形式存在。ASP.NET 4.0 框架提供了文档对象模型组件来方便地实现对 XML 数据的各种操作。由于 XML 在当前 Web 程序开发中占有很重要的地位，尤其在实现系统间的数据交换方面使用更为普遍，所以希望读者认真学习和掌握对 XML 格式的数据操作。

## 本章重点

- XML 的概念和语法
- 使用 XSL 转换 XML 文件
- 实现 XmlDocument 对象访问 XML 数据
- 掌握 XmlDataSource 控件的使用

## 10.1 XML 概述

XML 的英文全称是 eXtensible Markup Language，中文翻译为可扩展标记语言。它是网络应用开发的一项新技术。XML 同 HTML 一样，是一种标记语言，但是 XML 的数据描述的能力要比 HTML 强很多，XML 具有描述所有已知和未知数据的能力。XML 扩展性比较好，可以为新的数据类型制定新的数据描述规则，作为对标记集的扩展。

XML 出现以后就迅速走红，目前已经成为不同系统之间数据交换的基础。XML 的商用前景之所以非常广阔，也是因为它满足了当前商务数据交换的需求，XML 具有以下特点：

- XML 数据可以跨平台使用并可以被人阅读理解；
- XML 数据的内容和结构有明确的定义；
- XML 数据之间的关系得以强化；
- XML 数据的内容和数据的表现形式分离；
- XML 使用的结构是开放的、可扩展的。

因此，在利用 ASP.NET 开发的系统中，非常有必要利用 XML 这项 Web 程序开发的新技术。XML 可以作为数据资源的形式存在于服务器端，XML 还可以作为服务器端与客户端的数据交换语言。

而且，在 .NET 框架中，提供了一系列应用程序接口来实现 XML 数据的读写，比如使用 XmlDocument 类来实现 DOM 等，这些应用程序接口非常方便程序员用来操作 XML。

10.1.1 XML 的语法

XML 语言对格式有着严格的要求，主要包括格式良好和有效性两种要求。格式良好有利于 XML 文档被正确分析和处理，这一要求是相对于 HTML 语法的混乱而提出的，它大大提高 XML 的处理程序、处理 XML 数据的正确性和效率。XML 文档满足格式良好的要求后，会对文档进行有效性确认，有效性是通过对 DTD 或 Schema 的分析作出判断的。

一个 XML 文档有以下几个部分组成。

- (1) XML 的声明  
XML 声明具有如下形式：

```
<?xml version="1.0" encoding="GB2312"?>
```

XML 标准规定声明必须放在文档的第一行，声明其实也是处理指令的一种，一般都具有以上代码的形式。表 10-1 列举了声明的常用属性和其赋值。

表 10-1 XML 的声明的属性列表

属性	常用值	说明
Version	1.0	声明中必须包括此属性，而且必须放在第一位。它指定了文档所采用的 XML 版本号，现在 XML 的最新版本为 1.0
Encoding	GB2312 BIG5 UTF-8 UTF-16	文档使用的字符集为简体中文 文档使用的字符集为繁体中文 文档使用的字符集为压缩的 Unicode 编码 文档使用的字符集为 UCS 编码
Standalone	yes no	文档是独立文档，没有 DTD 文档与之配套 表示可能有 DTD 文档为本文档进行位置声明

- (2) 处理指令 PI  
处理指令 PI 为处理 XML 的应用程序提供信息。处理指令 PI 的格式为：

```
<? 处理指令名 处理指令信息?>
```

- (3) XML 元素  
元素是组成 XML 文档的核心，格式如下：

```
<标记>内容</标记>
```

XML 语法规则规定每个 XML 文档都要包括至少一个根元素，根标记必须是非空标记，包括整个文档的数据内容。数据内容则是位于标记之间的内容。

下面示例代码是一个标准的 XML 文档。

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <?xml-stylesheet type="text/xsl" href="style.xsl"?>
3. <DocumentElement>
4. <basic>
5. <ID>1</ID>
6. <NAME>三国演义</Name>
```

```
7. <AUTHOR>罗贯中</AUTHOR>
8. <PUBLISH>人民文学出版社</PUBLISH>
9. </basic>
10. <basic>
11. <ID>2</ID>
12. <NAME>红楼梦</Name>
13. <AUTHOR>曹雪芹</AUTHOR>
14. <PUBLISH>人民文学出版社</PUBLISH>
15. </basic>
16. <basic>
17. <ID>3</ID>
18. <NAME>西游记</Name>
19. <AUTHOR>吴承恩</AUTHOR>
20. <PUBLISH>人民文学出版社</PUBLISH>
21. </basic>
22. </DocumentElement>
```

代码说明：第 1 行为 XML 声明，表明该 XML 采用的版本是 1.0，字符编码为 utf-8。第 2 行为处理指令，表明该文档使用 xsl 进行转换，处理的文档是 style.xsl。第 3~22 行为 XML 元素，其中，第 4~9 行定义第 1 本图书的具体的信息，包括了图书的编号 ID、名称 Name、作者 AUTHOR 和出版社 PUBLISH 这样 4 个节点。第 10~15 行、第 16~21 行定义了其 2 本图书的信息。



Xml 和数据库在存储数据方面比较起来占用的空间小得多，而且简单易用，适用于中小型的项目的开发。

【例 10-1】本例实现在 Visual Studio 2010 开发环境中创建一个 XML 文件。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 10-1”
- 02 右键单击项目名称“例 10-1”，在弹出的快捷菜单选择“添加新项”命令。
- 03 在弹出如图 10-1 的“添加新项”对话框中选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“XML 文件”，然后在“名称”文本框输入该文件的名称“XMLFile.xml”，最后单击“添加”按钮。
- 04 在网站根目录下自动生成一个如图 10-2 所示的“XMLFile.xml”文件。

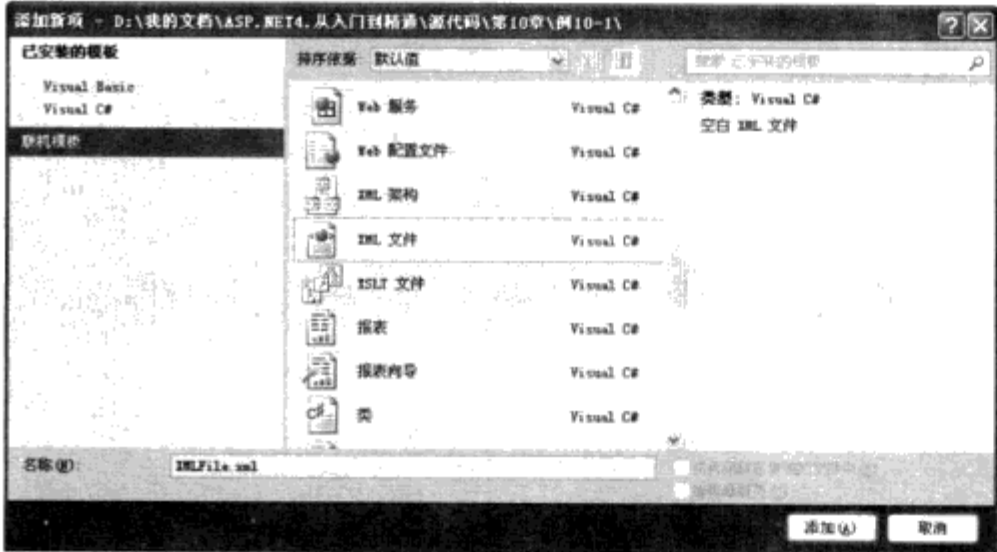


图 10-1 “添加新项”对话框

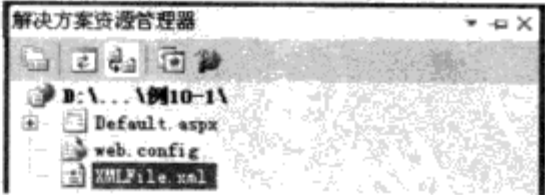


图 10-2 生成的 XML 文件

05 用鼠标双击“XMLFile.xml”文件，编写的代码如图 10-3 所示。

06 按下“Ctrl+F5”，运行结果如图 10-4 所示。

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<DocumentElement>
 <basic>
 <ID>1</ID>
 <NAME>三国演义</NAME>
 <AUTHOR>罗贯中</AUTHOR>
 <PUBLISH>人民文学出版社</PUBLISH>
 </basic>
 <basic>
 <ID>2</ID>
 <NAME>红楼梦</NAME>
 <AUTHOR>曹雪芹</AUTHOR>
 <PUBLISH>人民文学出版社</PUBLISH>
 </basic>
 <basic>
 <ID>3</ID>
 <NAME>西游记</NAME>
 <AUTHOR>吴承恩</AUTHOR>
 <PUBLISH>人民文学出版社</PUBLISH>
 </basic>
</DocumentElement>
```

图 10-3 XML 文件内容

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<DocumentElement>
 <basic>
 <ID>1</ID>
 <NAME>三国演义</NAME>
 <AUTHOR>罗贯中</AUTHOR>
 <PUBLISH>人民文学出版社</PUBLISH>
 </basic>
 <basic>
 <ID>2</ID>
 <NAME>红楼梦</NAME>
 <AUTHOR>曹雪芹</AUTHOR>
 <PUBLISH>人民文学出版社</PUBLISH>
 </basic>
 <basic>
 <ID>3</ID>
 <NAME>西游记</NAME>
 <AUTHOR>吴承恩</AUTHOR>
 <PUBLISH>人民文学出版社</PUBLISH>
 </basic>
</DocumentElement>
```

图 10-4 运行结果



XML 语法规则规定每个 XML 文档都要包括至少一个根元素。根标记必须是非空标记，包括整个文档的数据内容。数据内容则是位于标记之间的内容。

### 10.1.2 文档类型定义

文档类型定义 (Document Type Definition, DTD) 是一种规范，在 DTD 中可以向别人或 XML 的语法分析器解释 XML 文档标记集中每一个标记的含义。这就要求 DTD 必须包含所有将要使用的词汇列表，否则 XML 解析器无法根据 DTD 验证文档的有效性。

DTD 根据其出现的位置可以分为内部 DTD 和外部 DTD 两种。内部 DTD 是指 DTD 和相应的 XML 文档处在同一个文档中，外部 DTD 就是 DTD 与 XML 文档处在不同的文档之中。

下面示例代码是包含内部 DTD 的 XML 文档：

```
<?xml version="1.0" encoding="utf-8" ?>
2. <!DOCTYPE DocumentElement [
3. <!ELEMENT DocumentElement ANY>
4. <!ELEMENT basic (ID,NAME,AUTHOR,PUBLISH)>
5. <!ELEMENT ID (#PCDATA)>
6. <!ELEMENT NAME (#PCDATA)>
7. <!ELEMENT AUTHOR (#PCDATA)>
8. <!ELEMENT PUBLISH (#PCDATA)>
9.]>
10. <?xml-stylesheet type="text/xsl" href="style.xsl" ?>
11. <DocumentElement>
12. <basic>
13. <ID>1</ID>
14. <NAME>三国演义</NAME>
15. <AUTHOR>罗贯中</AUTHOR>
16. <PUBLISH>人民文学出版社</PUBLISH>
17. </basic>
18. <basic>
19. <ID>2</ID>
20. <NAME>红楼梦</NAME>
21. <AUTHOR>曹雪芹</AUTHOR>
```

```
22. <PUBLISH>人民文学出版社</PUBLISH>
23. </basic>
24. <basic>
25. <ID>3</ID>
26. <NAME> 西游记</NAME>
27. <AUTHOR>吴承恩</AUTHOR>
28. <PUBLISH>人民文学出版社</PUBLISH>
29. </basic>
30. </DocumentElement>
```

代码说明：第 2~9 行是该 XML 的文档类型定义，在这里定义了 XML 文档中包含的每一个标记元素，分别有 DocumentElement、basic、ID、NAME、AUTHOR 和 PUBLISH。第 10~30 行的 XML 文档的标记元素组成的内容，也就 XML 文档的具体内容。

从以上代码可以看出描述 DTD 文档也需要一套语法结构，关键字是组成语法结构的基础，表 10-2 列举了构建 DTD 时常用的关键字。

表 10-2 DTD 中常用的关键字

关键字	说明
ANY	数据既可是纯文本也可是子元素，多用来修饰根元素
ATTLIST	定义元素的属性
DOCTYPE	描述根元素
ELEMENT	描述所有子元素
EMPTY	空元素
SYSTEM	表示使用外部 DTD 文档
#FIXED	ATTLIST 定义的属性的值是固定
#IMPLIED	ATTLIST 定义的属性不是必须赋值的
#PCDATA	数据为纯文本
#REQUIRED	ATTLIST 定义的属性是必须赋值的
INCLUDE	表示包括的内容有效，类似与条件编译
IGNORE	与 INCLUDE 相应，表示包括的内容无效

此外 DTD 还提供了一些运算表达式来描述 XML 文档中的元素，常用的 DTD 运算表达式如表 10-3 所示，其中 A、B、C 代表 XML 文档中的元素。

表 10-3 DTD 中定义的表达式

表达式	说明
A+	元素 A 至少出现一次
A*	元素 A 可以出现很多次，也可以不出现
A?	元素 A 出现一次或不出现
(A B C)	元素 A，B，C 的间隔是空格，表示它们是无序排列
(A,B,C)	元素 A，B，C 的间隔是逗号，表示它们是有顺序排列
A B	元素 A，B 之间是逻辑或的关系

DTD 能够对 XML 文档结构进行描述，但 DTD 也有如下缺点。

- DTD 不支持数据类型，而在实际应用中往往会有多种复杂的数据类型，例如布尔型、时间等。
- DTD 的标记是固定的，用户不能扩充标记。
- DTD 使用不同于 XML 的独立的语法规则。



提示

目前出现了一种新的 XML 描述方法—Schema，该方法受到微软的推崇，并在.NET 框架中有所应用。Schema 的出现完善了 DTD 的不足，它本身就是一种 XML 的应用形式。Schema 对于文档的结构、数据的属性、类型的描述是全面的。此外 Schema 还是 DTD 的一种扩展和补充，有利于继承以前的数据。尽管 Schema 有如此多的好处，但它还并不是一种成熟的技术，目前还没有统一的国际标准。

10.1.3 XSL 语言

XSL 的英文是 eXtensible Stylesheet Language，翻译成中文就是可扩展样式语言。它是 W3C 制定的另一种表现 XML 文档的样式语言。XSL 是 XML 的应用，符合 XML 的语法规范，可以被 XML 的分析器处理。

XSL 是一种语言，通过对 XML 文档进行转换，然后将转换的结果表现出来。转换的过程是根据 XML 文档特性运行 XSLT（XSL Transformation）将 XML 文档转换成带信息的树型结果。然后按照 FO（Formatted Object）分析树，从而将 XML 文档表现出来。

XSL 转换 XML 文档分为两个步骤：建树和表现树。建树可以在服务器端执行，也可以在客户端执行。在服务器端执行时，把 XML 文档转换成 HTML 文档，然后发送到客户端。而在客户端执行建树的话，客户端必须支持 XML 和 XSL。

XSL 实际上就是通过模板将源文件文档按照模板的格式转换成结果文档的。模板定义了一系列的元素来描述源文档中的数据和属性等内容，在经过转换之后，建立树型结构。表 10-4 列举了 XSL 中常用的模板。

表 10-4 XSL 中常用的模板

关键字	说明
xsl:apply-import	调用导入的外部模板，可以应用为部分文档的模板
xsl:apply-templates	应用模板，通过“select”、“mode”两个属性确定要应用的模板
xsl:attribute	为元素输出定义属性节点
xsl:attribute-set	定义一组属性节点
xsl:call-template	调用由 call-template 指定的模板
xsl:choose	根据条件调用模板
xsl:comment	在输出加入注释
xsl:copy	复制当前节点到输出
xsl:element	在输出中创建新元素

(续表)

关键字	说明
xsl:for-each	循环调用模板匹配每个节点
xsl:if	模板在简单情况下的条件调用
xsl:message	发送文本信息给消息缓冲区或消息对话框
xsl:sort	排序节点
xsl:stylesheet	指定样式单
xsl:template	指定模板
xsl:value-of	为选定节点加入文本值

【例 10-2】本例实现自定义的 XSL 可扩展样式对指定的 XML 文档进行格式转换，转换为浏览器可读的信息。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 10-2”。

02 根据上例“例 10-1”的步骤创建好 test.xml 文件。

03 右键单击项目名称“例 10-2”，弹出的快捷菜单选择“添加新项”命令。

04 在弹出如图 10-5 的“添加新项”对话框中选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“样式表”，然后在“名称”文本框输入该文件的名称“trans.xml”，最后单击“添加”按钮。

05 在网站根目录自动生成一个如图 10-6 所示的“trans.xml”文件。

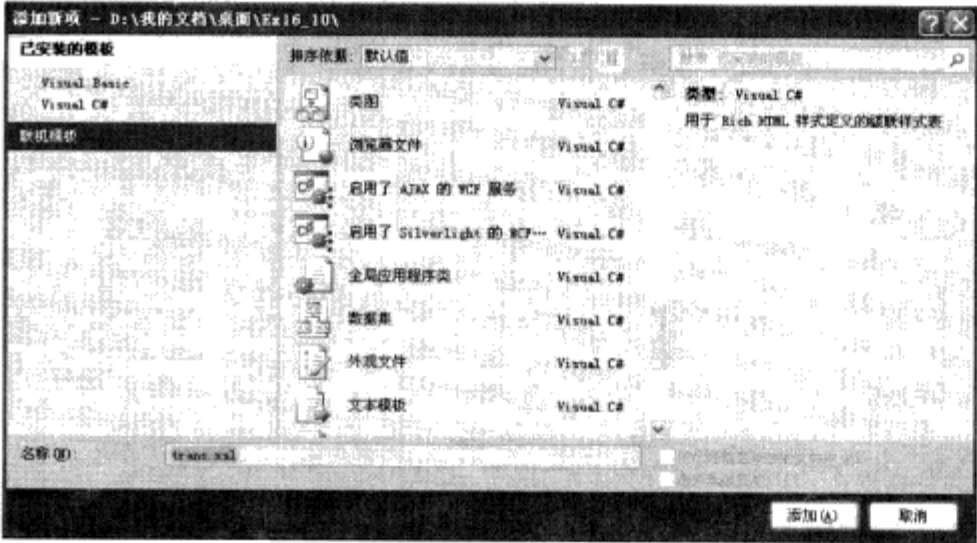


图 10-5 添加新项对话框



图 10-6 生成 xsl 文件

06 用鼠标双击“trans.xml”文件，编写如下代码：

```
1. <?xml version="1.0" encoding="gb2312"?>
2. <xsl:stylesheet version="1.0"
3. xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4. <xsl:template match="DocumentElement">
5. <html>
6. <body>
7. <table>
8. <tr>
9. <th>ID</th>
10. <th> NAME </th>
```

```

11. <th> AUTHOR </th>
12. <th> PUBLISH </th>
13. </tr>
14. <xsl:for-each select="basic">
15. <tr>
16. <td><xsl:value-of select="ID"/></td>
17. <td><xsl:value-of select="NAME"/></td>
18. <td><xsl:value-of select="AUTHOR"/></td>
19. <td><xsl:value-of select="PUBLISH"/></td>
20. </tr>
21. </xsl:for-each>
22. </table>
23. </body>
24. </html>
25. </xsl:template>
26. </xsl:stylesheet>

```

代码说明：第 1 行为 XML 声明，表明该 XML 采用的版本是 1.0，字符编码为 gb2312。第 2 行为 XSL 声明，表明该 XSL 采用的版本是 1.0。

第 4~25 行定义转换 XML 的模板（template），这里定义的模板是一个 HTML 标记，该 HTML 标记中包含一个 table 标记的定义，按照该模板，XSL 将把 XML 数据放到 HTML 标记中进行显示。其中，第 4 行绑定 XML 文档的根节点 DocumentElement；第 9~12 行定义表格的四个列标题；第 14~21 行循环 basic 节点，并读取其中的数据，根据表格的列标题，依次将数据值绑定到每个单元格中。

**07** 用鼠标双击网站的目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动 1 个 Xml 控件和 1 个 Button 控件。切换到“源视图”，在<form>和</form>标记之间编写代码如下。

```

1. XSL 转换 XML 文件

2. <asp:Xml ID="Xml1" runat="server"></asp:Xml>

3. <asp:Button ID="Button1" runat="server" Font-Size="9pt" OnClick="Button1_Click" Text="文件转换" />

```

代码说明：第 1 行定义显示的文本。第 2 行添加一个服务器 Xml 控件用于显示 Xml1 和 Xsl 文本的内容。第 3 行添加一个服务器按钮控件 Button1，并设置显示的文本和大小、单击事件 Click。

**08** 用鼠标双击网站目录下的“Default.aspx.cs”文件，在该文件中编写如下逻辑代码：

```

1. protected void Page_Load(object sender, EventArgs e){
2. XmlDocument doc = new XmlDocument();
3. doc.Load(Server.MapPath("test.xml"));
4. Xml1.Document = doc;
5. }
6. protected void Button1_Click(object sender, EventArgs e) {
7. XmlDocument xdoc = new XmlDocument();
8. xdoc.Load(Server.MapPath("test.xml"));
9. XslTransform xslt = new XslTransform();
10. xslt.Load(Server.MapPath("trans.xsl"));
11. Xml1.Document = xdoc;
12. Xml1.Transform = xslt;
13. }

```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行创建一个 XmlDocument 对象。第 3 行载入存储信息的 XML 文件。第 4 行设置在 Xml1 控件中显示的 Document。

第 6 行定义处理转换按钮 Button1 单击事件 Click 的方法。第 7 行创建一个 XmlDocument 对象。第 8 行载入存储信息的 XML 文件。第 9 行创建一个 XslTransform 对象。第 10 行导入 XSL 文件。第 11 行设置在 Xml1 控件中显示的 Document。第 12 行执行文档的转换操作。

**09** 按下“Ctrl+F5”，运行结果如图 10-7 所示。

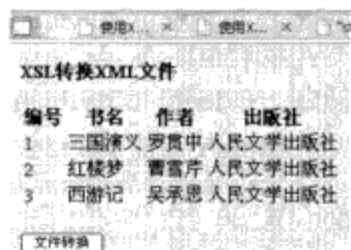


图 10-7 运行结果



**提示** XSLT 主要用来转换 XML 文档，在商业系统中它可以将 XML 文档转换成可以被各种系统或是应用程序解读的数据。这非常有利于各种商业系统之间的数据交换。

### 10.1.4 XPath

XPath 是 XSLT 的重要组成部分。XPath 的作用在于为 XML 文档的内容定位，并通过 XPath 来访问指定的 XML 元素。在利用 XSL 进行转换的过程中，匹配的概念非常重要。在模板声明语句 `xsl:template match = ""` 和模板应用语句 `xsl:apply-templates select = ""` 中，用引号括起来的部分必须能够精确地定位节点。具体的定位方法则在 XPath 中给出。

之所以要在 XSL 中引入 XPath 的概念，目的就是为了让在匹配 XML 文档结构树时能够准确地找到某一个节点元素。可以把 XPath 比作文件管理路径：通过文件管理路径，可以按照一定的规则查找到所需要的文件；同样，依据 XPath 所制定的规则，也可以很方便地找到 XML 结构文档树中的任何一个节点，显然这对 XSLT 来说是一个最基本的功能。

XPath 提供了一系列的节点匹配的方法：

- 路径匹配：路径匹配和文件路径的表示比较相似，通过一系列的符号来指定路径。
- 位置匹配：根据每个元素的子元素都是有序的原则来匹配。
- 亲属关系匹配：XML 是一个树型结构，因此在匹配时可以利用树型结构的“父子”关系。
- 条件匹配：利用一些函数的运算结果的布尔值来匹配符合条件的节点。



**提示** 以上简要概述了一下有关 XML 的知识，有关 XML 的主要知识包括以上这些内容，有兴趣的读者可以按照以上的知识框架来深入学习。

XPath 是专门用来在 XML 文档中查找信息的语言，相当于数据库系统中的 SQL 语句。目的是为了能够在 XML 结构树中准确地找到某一个节点元素。

## 10.2 访问和操作 XML

XML 数据的访问和操作是通过 DOM 来实现的，DOM (Document Object Model) 是一个程序接口，应用程序和脚本可以通过这个接口访问和修改 XML 文档数据。

### 10.2.1 创建 XML 文档

XML 语言仅仅是一种信息交换的载体，是一种信息交换的方法。而要使用 XML 文档则必须

通过使用一种称为接口的技术。正如使用 ODBC 接口访问数据库一样，DOM 接口应用程序使得对 XML 文档的访问变得简单。

DOM 是一个程序接口，应用程序和脚本可以通过这个接口访问和修改 XML 文档数据。DOM 接口定义了一系列对象来实现对 XML 文档数据的访问和修改。DOM 接口将 XML 文档转换为树型的文档结构，应用程序通过树型文档对 XML 文档进行层次化的访问，从而实现对 XML 文档的操作，比如访问树的节点、创建新节点等。

微软大力支持 XML 技术，在 .NET 框架中实现了对 DOM 规范的良好支持，并提供了一些扩展技术，使得程序员对 XML 文档的处理更加简便。而基于 .NET 框架的 ASP.NET，可以充分使用 .NET 类库来实现对 DOM 的支持。

.NET 类库中支持 DOM 的类主要存在于 System.Xml 和 System.Xml.XmlDocument 命名空间中。这些类分为两个层次：基础类和扩展类。基础类组包括了用来编写操纵 XML 文档的应用程序所需要的类，扩展类被定义用来简化程序员的开发工作的类。

在基础类中包含了三个类。

- XmlNode 类用来表示文档树中的单个节点，它描述了 XML 文档中各种具体节点类型的共性，它是一个抽象类，在扩展类层次中有它的具体实现。
- XmlNodeList 类用来表示一个节点的有序集合，它提供了对迭代操作和索引器的支持。
- XmlNamedNodeMap 类用来表示一个节点的集合，该集合中的元素可以使用节点名或索引来访问，支持了使用节点名称和迭代器来对属性集合的访问，并且包含了对命名空间的支持。

扩展类中主要包括了以下几个由 XmlNode 类派生出来的类，如表 10-5 所示。

表 10-5 扩展类中包含的主要的类

类	说明
XmlAttribute	表示一个属性。此属性的有效值和默认值在 DTD 或架构中进行定义
XmlAttributeCollection	表示属性集合，这些属性的有效值和默认值在 DTD 或架构中进行定义
XmlComment	表示 XML 文档中的注释内容
XmlDocument	表示 XML 文档
XmlDocumentType	表示 XML 文档的 DOCTYPE 声明节点
XmlElement	表示一个元素
XmlEntity	表示 XML 文档中一个解析过或为解析过的实体
XmlEntityReference	表示一个实体的引用
XmlLinkedNode	获取紧靠该节点（之前或之后）的节点
XmlReader	表示提供对 XML 数据进行快速、非缓存、只进访问的读取器
XmlText	表示元素或属性的文本内容
XmlTextReader	表示提供对 XML 数据进行快速、非缓存、只进访问的读取器
XmlTextWriter	表示提供快速、非缓存、只进方法的编写器，该方法生成包含 XML 数据（这些数据符合 W3C 可扩展标记语言 XML 1.0 和 XML 中命名空间的建议）的流或文件
XmlWriter	表示提供快速、非缓存、只进方法的编写器，该方法生成包含 XML 数据（这些数据符合 W3C 可扩展标记语言 XML 1.0 和 XML 中命名空间的建议）的流或文件

创建 XML 文档的方法有两种：

(1) 创建不带参数的 XmlDocument，代码如下。

```
XmlDocument doc = new XmlDocument();
```

以上代码，使用了不带参数的构造函数创建一个 XmlDocument 对象 doc。

(2) 创建带参数的 XmlDocument

创建一个 XmlDocument 并将 XmlNameTable 作为参数传递给它。XmlNameTable 类是原子化字符串对象的表。该表为 XML 分析器提供了一种高效的方法，即对 XML 文档中所有重复的元素和属性名使用相同的字符串对象。创建文档时，将自动创建 XmlNameTable，并在加载此文档时用属性和元素名加载 XmlNameTable。如果已经有一个包含名称表的文档，且这些名称在另一个文档中会很有用，则可使用将 XmlNameTable 作为参数的 Load 方法创建一个新文档。使用此方法创建文档后，该文档使用现有 XmlNameTable，后者包含所有已从其他文档加载到此文档中的属性和元素。它可用于有效地比较元素和属性名。以下示例是创建带参数的 XmlDocument 的代码：

```
System.Xml.XmlDocument doc = new XmlDocument(xmlNameTable);
```

以上代码，使用带参数的构造函数创建一个 XmlDocument 对象 doc，其中 System.Xml 是 XmlDocument 的命名空间。



在使用关于 XML 的类文件（如 XmlDocument）时，必须首先添加命名空间 using System.Xml。

### 10.2.2 XML 文档的保存

XML 创建完毕，必须使用 Save 方法保存 XML 文档。Save 方法有如下四个重载方法。

- Save(string filename): 将文档保存到文件 filename 的位置。
- Save(System.IO.Stream outputStream): 保存到流 outputStream 中，流的概念存在于文件操作中。
- Save(System.IO.TextWriter writer): 保存到 TextWriter 中，TextWriter 也是文件操作中的一个类。
- Save(XmlWriter w): 保存到 XmlWriter 中。

以下代码演示如可实现对 Xml 文档的保存。

```
1. XmlDocument doc = new XmlDocument();
2. doc.Save(Server.MapPath("XMLFile.xml"));
```

以上代码，第 1 行创建一个 XmlDocument 对象 doc。第 2 行调用 doc 对象的 Save 方法将文档数据保存到当前程序的 XMLFile.xml 文件中。

### 10.2.3 将 XML 读入文档

DOM 可以将不同格式的 XML 读入内存，这些格式可以是字符串、流、URL、文本读取器或

XmlReader 的派生类。

读取的 XML 数据的方法有两种：

(1) Load 方法，该方法加载指定的 XML 数据。总共包含四个重载函数：

- XmlDocument.Load(Stream): 从指定的流加载 XML 文档。
- XmlDocument.Load(String): 从指定的 URL 加载 XML 文档。
- XmlDocument.Load(Reader): 从指定的 TextReader 加载 XML 文档。
- XmlDocument.Load(XmlReader): 从指定的 XmlReader 加载 XML 文档。

(2) LoadXML 方法，该方法没有重载函数，仅仅是 XmlDocument.LoadXML(String)，从字符串中读取 XML。下面我们来看一个将 XML 读入文档的实例。

**【例 10-3】**使用 XmlDocument 对象加载 XML 字符串，然后将 XML 数据保存到一个 XML 的文件中。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 10-3”。
- 02** 在该网站中创建一个“XMLFile.xml”文件。
- 03** 用鼠标双击网站目录下的“Default.aspx.cs”文件，在该文件中编写如下逻辑代码：

```
1. protected void Page_Load(object sender, EventArgs e){
2. XmlDocument doc = new XmlDocument();
3. doc.LoadXml("<basic>" +
4. "<ID>1</ID>" +
5. "<NAME>三国演义</NAME>" +
6. "<AUTHOR>罗贯中</AUTHOR>" +
7. "<PUBLISH>人民文学出版社</PUBLISH>" +
8. "</basic>");
9. doc.Save(Server.MapPath("XMLFile.xml"));
10. }
```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行创建一个 XmlDocument 对象 doc。第 3~8 行利用 doc 对象的 LoadXML 方法从字符串中把 XML 数据加载到 doc 中，分别添加了 5 个标签和标签的内容。第 9 行利用 doc 的 Save 方法把 XML 数据保存到 XMLFile.xml 文件中。

**04** 右键单击“XMLFile.xml”文件，在弹出的菜单中选择“在浏览器中查看”命令。运行结果如图 10-8 所示。

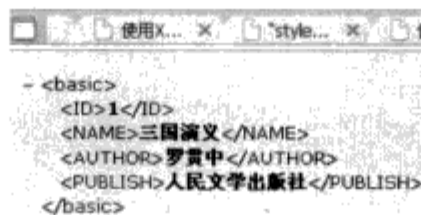


图 10-8 运行结果



提示

在进行 XML 文件的常用操作中，通常会使用递归的方法来循环遍历每一个子节点，这样比不使用递归的循环效率要高。

### 10.2.4 选择节点

ASP.NET 的 DOM 提供了基于 XPath 的导航方法,使用这些导航方法可以方便地查询 DOM 中的信息。DOM 提供了如下两种 XPath 导航方法。

- (1) SelectSingleNode 方法: 返回符合选择条件的第一个节点。
- (2) SelectNodes 方法: 返回包含匹配节点的 XmlNodeList。

**【例 10-4】**使用 SelectNodes 方法从 XML 文档中获取 basic 节点,并把获得每个节点的数据输出到页面上。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 10-4”。

**02** 在该网站中创建一个“XMLFile.xml”文件, 文件中的内容与“例 9-1”中“XMLFile.xml”文件相同。

**03** 用鼠标双击网站目录下的“Default.aspx.cs”文件, 在该文件中编写如下逻辑代码:

```

1. protected void Page_Load(object sender, EventArgs e){
2. XmlDocument doc = new XmlDocument();
3. doc.Load(Server.MapPath("XMLFile.xml"));
4. XmlNodeList nodeList;
5. XmlNode root = doc.DocumentElement;
6. nodeList = root.SelectNodes("//basic");
7. foreach (XmlNode xmlNode in nodeList) {
8. XmlNodeList list = xmlNode.ChildNodes;
9. foreach (XmlNode xmlNode1 in list) {
10. Response.Write(xmlNode1.InnerText);
11. Response.Write(" ");
12. }
13. Response.Write("
");
14. }
15. }
```

代码说明: 第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行创建 DOM 对象 doc。第 3 行把 XML 文档通过 Load 方法装入 doc。第 4 行定义节点列表 XmlNodeList 对象 nodeList。第 5 行通过 doc 对象的 DocumentElement 属性定义根节点对象 root。第 6 行查找 basic 节点列表。第 7 行使用 foreach 循环遍历访问节点列表。第 8 行使用 ChildNodes 属性获得 basic 节点下的所有子节点。第 9 行使用 foreach 循环遍历子节点的内容。第 10 行输出子节点包含的数据内容。

**04** 按下“Ctrl+F5”, 运行结果如图 10-9 所示。

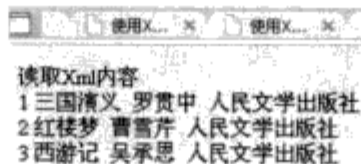


图 10-9 运行结果

### 10.2.5 新节点的创建

XmlDocument 具有用于所有节点类型的 Create 方法, 为该方法提供节点的名称、节点的内容

或节点的参数就可创建节点。表 10-6 列举了 XmlDocument 的常用的创建节点的方法。

表 10-6 XmlDocument 的常用的创建节点的方法

方法	说明
CreateAttribute	创建具有指定名称的 XmlAttribute
CreateCDataSection	创建包含指定数据的 XmlCDataSection
CreateComment	创建包含指定数据的 XmlComment
CreateDocumentType	创建新的 XmlDocumentType 对象
CreateElement	创建 XmlElement
CreateEntityReference	创建具有指定名称的 XmlEntityReference
CreateNode	创建 XmlNode
CreateTextNode	创建具有指定文本的 XmlText

在创建新节点后,就可以用方法来给新创建的节点添加信息,将其插入到 XML 结构树中。表 10-7 列出了这些常用的方法。

表 10-7 向 XML 结构树中插入节点的方法

方法	说明
InsertBefore	插入到引用节点之前
InsertAfter	插入到引用节点之后
AppendChild	将节点添加到给定节点的子节点列表的末尾
PrependChild	将节点添加到给定节点的子节点列表的开头
Append	将 XmlAttribute 节点追加到与元素关联的属性集合的末尾

**【例 10-5】**本例实现动态创建 XML 数据节点,用户在文本框中输入节点的信息,单击“创建”按钮,完成节点的创建。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 10-5”。

**02** 在该网站中创建一个“XMLFile.xml”文件,文件中的内容与“例 9-1”中“XMLFile.xml”文件相同。

**03** 用鼠标双击网站的目录下的“Default.aspx”文件,进入的“视图编辑界面”,从工具箱中拖动 1 个 Label 控件、1 个 Button 控件和 4 个 TextBox 控件。切换到“源视图”在<form>和</form>标记之间编写代码如下。

```

1. 请输入节点的内容

2. 编号: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. 书名: <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

4. 作者: <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

5. 出版社: <asp:TextBox ID="TextBox4" runat="server" Height="17px"
6. Width="132px"></asp:TextBox>

7. <asp:Button ID="Button1" runat="server" Text="创建" onclick="Button1_Click"/>
8. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>

```

代码说明：第 2~5 行分别添加了 4 个服务器文本框控件，用于用户输入编号、书名、作者和出版社的内容。第 7 行添加一个服务器按钮控件并设置显示的文本和单击按钮处理事件 Click。第 8 行添加一个服务器标签控件 Label1。

**04** 用鼠标双击网站目录下的“Default.aspx.cs”文件，在该文件中编写如下逻辑代码。

```
1. protected void Button1_Click(object sender, EventArgs e){
2. XmlDocument doc = new XmlDocument();
3. doc.Load(Server.MapPath("XMLFile.xml"));
4. string str = "<ID>" + TextBox1.Text + "</ID>";
5. string str1 = "<NAME>" + TextBox2.Text + "</NAME>";
6. string str2 = "<AUTHOR>" + TextBox3.Text + "</AUTHOR>";
7. string str3 = "<PUBLISH>" + TextBox4.Text + "</PUBLISH>";
8. string str4 = "<DocumentElement>" + "<basic>" + str + str1 + str2 + str3 + "</basic>" + "</DocumentElement>";
9. XmlDocument doc1 = new XmlDocument();
10. doc1.LoadXml(str4);
11. XmlNode node = doc.ImportNode(doc1.DocumentElement.LastChild, true);
12. doc.DocumentElement.AppendChild(node);
13. doc.Save(Server.MapPath("XMLFile.xml"));
14. Label1.Text = "创建新节点成功!";
15. }
```

代码说明：第 1 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 2 行定义 XmlDocument 对象 doc。第 3 行使用 doc 对象把 XMLFile.xml 文件加载到内存中。第 4~7 行设置文件的标记和从用户输入文本框的内容。第 8 行通过拼接字符串将添加 XML 文档内容赋给字符串对象 str4。第 9 行定义 XmlDocument 对象 doc1。第 10 行调用 doc1 的 LoadXml 方法将 str4 作为参数读入 XMLFile.xml 文件中。第 11 行调用 doc 对象的 ImportNode 方法将 doc1 对象的最后一根子节点导入 doc 对象加载的文档中，并赋给一个创建的节点对象 node。第 12 行调用 doc 对象根节点添加子节点的方法 AppendChild 添加 node 节点。第 13 行把修改后 doc 对象保存到“XMLFile.xml”。第 14 行在标签控件上显示创建成功的提示。

**05** 按下“Ctrl+F5”，运行结果如图 10-10 所示。用户输入新节点的内容，单击“创建”按钮，显示创建节点成功的提示。

**06** 右键单击“XMLFile.xml”文件，在弹出的菜单中选择“在浏览器中查看”命令，运行后结果如图 10-11 所示。



图 10-10 运行结果 1

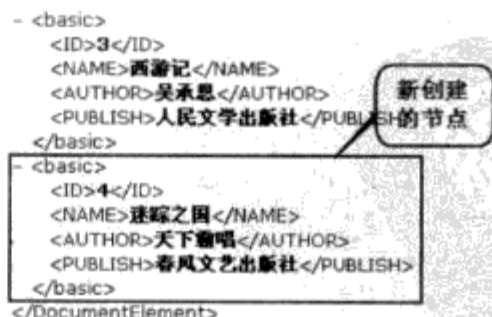


图 10-11 运行结果 2



提示

在操作插入一个新的节点时，必须从最底层的节点开始，一层一层进行添加（包括节点的文本和值）到上一个节点，直至添加到根节点为止。

## 10.2.6 XML 文档的修改

在.NET 4.0 框架下,使用 DOM 开发人员可以有多种方法来修改 XML 文档的节点、内容和值。常用的修改 XML 文档的方法如下。

- 使用 XmlNode.Value 方法更改节点值。
- 通过用新节点替换节点来修改全部节点集。这可使用 XmlNode.InnerXml 属性完成。
- 使用 XmlNode.ReplaceChild 方法用新节点替换现有节点。
- 使用 XmlCharacterData.AppendData 方法、XmlCharacterData.InsertData 方法或 XmlCharacterData.ReplaceData 方法将附加字符添加到从 XmlCharacter 类继承的节点。
- 对从 XmlCharacterData 继承的节点类型使用 DeleteData 方法移除某个范围的字符来修改内容。
- 使用 SetAttribute 方法更新属性值。如果不存在属性,SetAttribute 创建一个新属性;如果存在属性,则更新属性值。

**【例 10-6】**本例实现在应用程序中修改 XML 文件。用户在下拉列表中选择要修改的节点,然后在文本框中输入要更新的新节点名称,单击“修改”按钮,完成修改文件操作。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 10-6”。

**02** 在该网站中创建一个“XMLFile.xml”文件,文件中的内容与“例 9-1”中“XMLFile.xml”文件相同。

**03** 用鼠标双击网站的目录下的“Default.aspx”文件,进入的“视图编辑界面”,从工具箱中拖动 1 个 GridView 控件、1 个 DropDownList 控件、1 个 TextBox 控件和 1 个 Button 控件。切换到“源视图”在<form>和</form>标记之间编写代码如下:

```
1. <h3>修改 XML 文档</h3>
2. <asp:GridView ID="GridView1" runat="server"></asp:GridView>
3. 请选择节点<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"> </asp:DropDownList>

4. 新节点名<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
5. <asp:Button ID="Button1" runat="server" Text="修改" onclick="Button1_Click" />
```

代码说明:第 1 行显示标题文本。第 2 行添加一个服务器列表控件 GridView1 以显示 XML 文档的信息。第 3 行添加一个服务器下拉列表控件 DropDownList1,并设置自动回传服务器。第 4 行添加一个服务器文本框控件 TextBox1。第 5 行添加服务器按钮控件,并设置显示的文本和单击按钮处理事件 Click。

**04** 用鼠标双击网站目录下的“Default.aspx.cs”文件,在该文件中编写如下逻辑代码:

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. DataSet ds= new DataSet();
4. ds.ReadXml(Server.MapPath("XMLFile.xml"));
5. GridView1.DataSource = ds;
6. GridView1.DataBind();
7. DropDownList1.DataSource = ds;
8. DropDownList1.DataTextField = "NAME";
9. DropDownList1.DataBind();
10. }
```

```

11. }
12. protected void Button1_Click(object sender, EventArgs e){
13. XmlDocument doc = new XmlDocument();
14. doc.Load(Server.MapPath("XMLFile.xml"));
15. XmlNodeList xnl = doc.SelectSingleNode("DocumentElement").ChildNodes;
16. foreach (XmlNode xn in xnl){
17. XmlElement xe = (XmlElement)xn;
18. if (xe.Name == "basic"){
19. XmlNodeList xnlChild = xe.ChildNodes;
20. foreach (XmlNode xnChild in xnlChild){
21. XmlElement xeChild = (XmlElement)xnChild;
22. if (xeChild.Name == "NAME" && xeChild.InnerText == this.DropDownList1.SelectedValue.Trim()){
23. xeChild.InnerText = TextBox1.Text.Trim();
24. Response.Write("<script>alert('修改成功')</script>");
25. }
26. }
27. }
28. }
29. doc.Save(Server.MapPath("XMLFile.xml"));
30. Response.Write("<script>location='Default.aspx'</script>");
31. }

```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行判断如果当前加载的页面不是回传页面。第 3 行创建 DataSet 对象 ds。第 4 行调用 ds 对象 ReadXml 方法读取“XMLFile.xml”。第 5 行将 ds 作为列表控件 GridView1 的数据源。第 6 行绑定数据到 GridView1 控件。第 7 行将 ds 作为下拉列表控件 DropDownList1 的数据源。第 8 行绑定显示在 DropDownList1 的是“NAME 标记”中的内容。第 9 行绑定数据到 DropDownList1 控件。

第 12 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 13 行创建 XmlDocument 对象 doc。第 14 行加载“XMLFile.xml”文件。第 15 行获取 XML 文件中“DocumentElement”节点下所有子节点集合。第 16 行循环变量所有节点集合中的子节点。第 17 行获取每一个节点元素。第 18 行判断是否是节点是 basic。第 19 行获取 basic 节点下所有子节点的集合。第 20 行遍历所有子节点集合中的节点。第 21 行获取每一个节点元素。第 22 行判断如果节点元素的名称是“NAME”同时，节点元素的内容和下拉列表控件 DropDownList1 中用户选择的值相同，则第 23 行将用户输入文本框的新节点值赋给要修改的节点元素。第 24 行在页面显示修改成功的对话框。第 29 行把修改后 doc 对象保存到“XMLFile.xml”。第 30 行跳转页面到 Default.aspx。

**05** 按下“Ctrl+F5”，运行结果如图 10-12 所示。用户选择要修改的节点，输入新节点名称，单击“修改”按钮。

**06** 修改文档后的界面如图 10-13 所示。

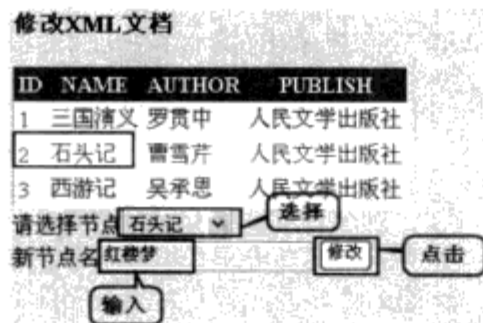


图 10-12 运行结果 1

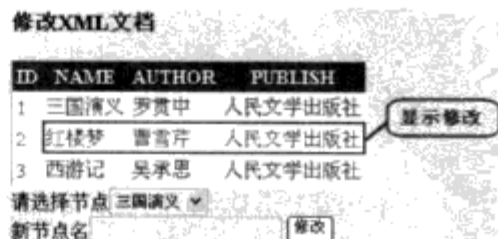


图 10-13 运行结果 2

## 10.2.7 XML 文档的删除

DOM 在内存中之后，可以删除 XML 中的节点，或删除特定节点类型中的内容和值。

### 1. 删除节点

若要从 DOM 中删除节点，可以使用 `RemoveChild` 方法移除特定节点。删除节点时，此方法移除属于所移除的节点的子树（即如果它不是叶节点）。

若要从 DOM 中移除多个节点，可以使用 `RemoveAll` 方法移除当前节点的所有子级和属性。

如果使用 `XmlNamedNodeMap`，则可以使用 `RemoveNamedItem` 方法移除节点。

### 2. 删除属性集中的属性

可以使用 `XmlAttributeCollection.Remove` 方法移除特定属性；也可以使用 `XmlAttributeCollection.RemoveAll` 方法移除集合中的所有属性，使元素不具有任何属性；或者使用 `XmlAttributeCollection.RemoveAt` 方法移除属性集中的属性（通过使用其索引号）。

### 3. 删除节点属性

使用 `XmlElement.RemoveAllAttributes` 移除属性集合；使用 `XmlElement.RemoveAttribute` 方法按名称移除集合中的单个属性；使用 `XmlElement.RemoveAttributeAt` 按索引号移除集合中的单个属性。

### 4. 删除节点内容

可以使用 `DeleteData` 方法移除字符，此方法从节点中移除某个范围的字符。如果要完全移除内容，则移除包含此内容的节点。如果要保留节点，但节点内容不正确，则修改内容。

**【例 10-7】** 本例实现在应用程序中删除 XML 文件的节点。用户在下拉列表中选择要删除的节点，单击“删除”按钮，完成删除操作。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 10-7”。

**02** 在该网站中创建一个“XMLFile.xml”文件，文件中的内容与“例 9-1”中“XMLFile.xml”文件相同。

**03** 用鼠标双击网站的目录下的“Default.aspx”文件，进入的“视图编辑界面”，从工具箱中拖动 1 个 GridView 控件、1 个 DropDownList 控件、和 1 个 Button 控件。

**04** 用鼠标双击网站目录下的“Default.aspx.cs”文件，在该文件中编写关键逻辑代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. XmlDocument doc = new XmlDocument();
3. doc.Load(Server.MapPath("XMLFile.xml"));
4. XmlNodeList node;
5. XmlElement root = doc.DocumentElement;
6. node = root.SelectNodes("descendant::basic[NAME='"+DropDownList1.Text.Trim()+"']");
7. foreach (XmlNode n in node){
8. root.RemoveChild(n);
9. }
10. Response.Write("<script>alert('删除节点成功!')</script>");
11. doc.Save(Server.MapPath("XMLFile.xml"));
12. }
```

代码说明:第 1 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 2 行创建 XmlDocument 对象 doc。第 3 行加载 XMLFile.xml 文件。第 4 行声明 XmlNodeList 对象 node。第 5 行定义调用 doc 对象的 DocumentElement 获得根节点对象 root。第 6 行通过根节点对象 root 的 SelectNodes 获得 basic 节点下用户选择的 NAME 子节点元素。第 7 行循环遍历用户选择的子节点元素下的所有子节点。第 8 行调用 root 对象的 RemoveChild 方法删除这些子节点内容。第 10 行显示删除成功对话框。第 11 行把商场数据后的 doc 对象保存到 XMLFile.xml 文件中。

**05** 按下“Ctrl+F5”，运行结果如图 10-14 所示。用户选择要删除的节点，单击“删除”按钮。

**06** 删除节点后的界面如图 10-15 所示。

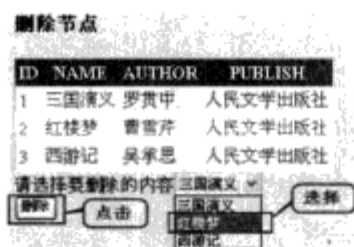


图 10-14 运行结果 1

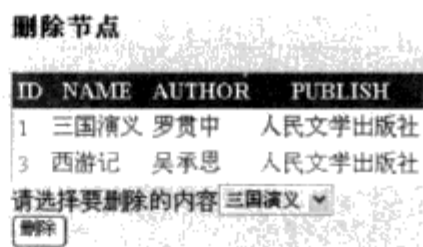


图 10-15 运行结果 2

### 10.3 XmlDataSource 控件

XmlDataSource 控件是 ASP.NET 4.0 中提供的专门用于访问 XML 文件的数据源控件。它特别适用于分层的 ASP.NET 服务器控件，如 TreeView 或 Menu 控件。它支持使用 XPath 表达式来实现筛选功能，并允许对数据应用 XSLT 转换，同时允许通过保存更改后的整个 XML 文档来更新数据。

XmlDataSource 控件与前面介绍的 SqlDataSource 工作原理一样，不过相比之下 XmlDataSource 控件还是有以下两点不同。

- XmlDataSource 控件从 XML 文件中获取信息，而不是从数据库中获取信息。
- XmlDataSource 控件返回的信息是分层次的，而且级别可以是无限级别的，而 SqlDataSource 返回的信息只能是一个表格形式的。

除了以上两点不同，XmlDataSource 控件同其他数据源控件的特性和用法一样。

可以利用 XmlDataSource 控件把 XML 数据绑定到 GridView 控件等表格控件中。要在像 GridView 控件这样的表格控件中显示 XML 数据，就必须使用 XPath 来指定要在 GridView 控件中显示的数据项。这是因为 XML 文件是有层次的，XmlDataSource 控件只是从 XML 文件中获得数据，并不能为 GridView 控件指明要显示的数据项，因此需要在 GridView 控件的列定义中利用 XPath 来指明要显示的数据项。

而利用 XmlDataSource 控件把 XML 数据绑定到 TreeView 控件等层次控件中就比较容易，由于 TreeView 控件是把数据分层次显示的，与 XML 描述的形式相似，因此这些控件显示 XML 数据就比较容易。

**【例 10-8】**本例演示如何利用 XmlDataSource 控件把 XML 文件中的数据显示在 GridView 控件中。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 10-8”。

**02** 在该网站中创建一个“XMLFile.xml”文件, 文件中的内容与“例 9-1”中“XMLFile.xml”文件相同。

**03** 用鼠标双击网站的目录下的“Default.aspx”文件, 进入的“视图编辑界面”, 从工具箱中拖动 1 个 GridView 控件、1 个 XmlDataSource 控件。

**04** 将鼠标移到 GridView1 控件上, 其上方会出现如图 10-16 所示一个向右的黑色小三角。单击它, 弹出“GridView 任务”列表。在“选择数据源”下拉列表中选中“XmlDataSource1”。

**05** 将鼠标移到 XmlDataSource1 控件上, 其上方会出现如图 10-17 所示一个向右的黑色小三角, 单击它, 弹出“XmlDataSource 任务”列表, 选择“配置数据源”选项。

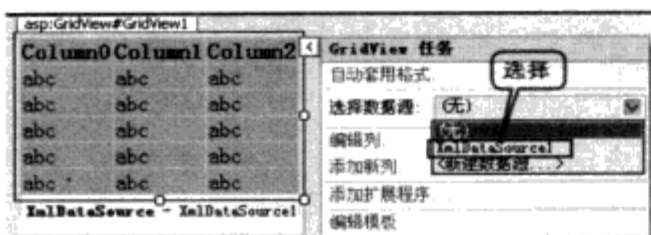


图 10-16 GridView 任务



图 10-17 XmlDataSource 任务

**06** 弹出如图 10-18 所示的“配置数据源”对话框, 单击数据文件文本框后的“浏览”按钮。



图 10-18 “配置数据源”对话框

**07** 弹出如图 10-19 所示的“选择 XML 文件”对话框中, 在项目文件夹列表中选择文件夹, 在文件夹内容列表中单击“XMLFile.xml”文件, 最后单击“确定”按钮。

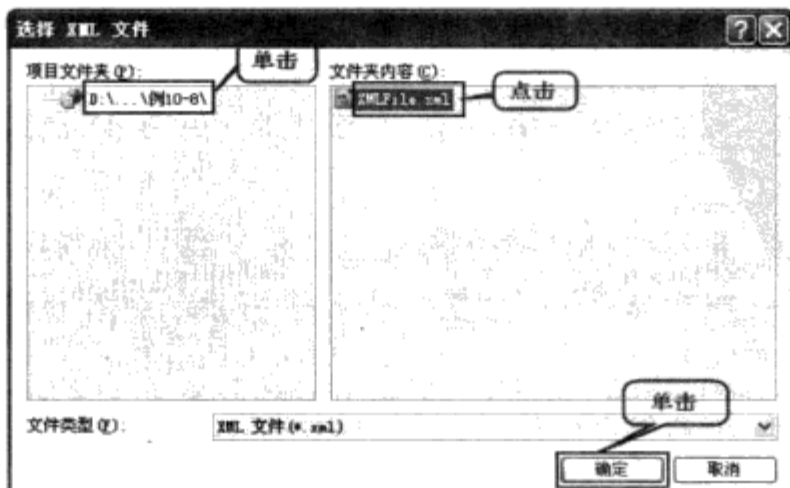


图 10-19 “选择 XML 文件”对话框

**08** 返回图 10-18 所示的窗体，单击“确定”按钮完成数据源的配置。

**09** 切换到“源视图”，在<form>和</form>标记之间编写如下代码。

```

1. <h3>XmlDataSources 数据绑定</h3>
2. <asp:GridView ID="GridView1" runat="server" DataSourceID="XmlDataSource1" AutoGenerateColumns="False">
3. <Columns>
4. <asp:TemplateField HeaderText="编号">
5. <ItemTemplate><%# XPath("ID") %></ItemTemplate>
6. </asp:TemplateField>
7. <asp:TemplateField HeaderText="书名">
8. <ItemTemplate><%# XPath("NAME") %></ItemTemplate>
9. </asp:TemplateField>
10. <asp:TemplateField HeaderText="作者">
11. <ItemTemplate><%# XPath("AUTHOR") %></ItemTemplate>
12. </asp:TemplateField>
13. <asp:TemplateField HeaderText="出版社">
14. <ItemTemplate><%# XPath("PUBLISH") %></ItemTemplate>
15. </asp:TemplateField>
16. </Columns>
17. </asp:GridView>

18. <asp:XmlDataSource ID="XmlDataSource1" runat="server" DataFile="~/XMLFile.xml"></asp:XmlDataSource>

```

代码说明：第 1 行显示标题文本。第 2 行定义服务器列表控件 GridView1，设置其数据源控件为 XmlDataSource1 和禁止自动加载列。第 3~16 行定义 GridView1 的列设置，这里利用模板列把要显示字段与 XML 文件中节点对应起来，而数据的获得则通过 XPath 来获得。其中，第 4~6 行设置模板定义列的标题“编号”，第 5 行设置 XPath 来指明要显示的数据项绑定。按照以上的方法依次定义其余的 3 个数据列，分别是书名、作者和出版社并分别设置 XPath 指定要在 GridView 控件中显示的数据项。第 17 行定义一个 XML 数据绑定控件 XmlDataSource1，通过属性 DataFile 指定要绑定数据文件是 XMLFile.xml。

**10** 按下“Ctrl+F5”，运行结果如图 10-20 所示。

XmlDataSources 数据绑定

编号	书名	作者	出版社
1	三国演义	罗贯中	人民文学出版社
2	红楼梦	曹雪芹	人民文学出版社
3	西游记	吴承恩	人民文学出版社

图 10-20 运行结果

## 10.4 上机题

1. 创建一个名为“Students”的 XML 文件，该文件包含数个学生的信息，每个学生将包含：编号、姓名、电话、地址和地区等信息，运行结果如图 10-21 所示。

2. 编写一个名为“Student”的 XSL 文件，利用该转换文件在应用程序中把上机题 1 创建的 Students.xml 文件按照图 10-22 所示的形式显示在页面上。

```
<DocumentElement>
- <Students>
 <ID>1</ID>
 <NAME>张琴</NAME>
 <Phone>13256789023</Phone>
 <Address>朝阳门外大街</Address>
 <City>北京</City>
</Students>
- <Students>
 <ID>2</ID>
 <NAME>邵飞</NAME>
 <Phone>13698562314</Phone>
 <Address>海淀区清河镇</Address>
 <City>北京</City>
</Students>
- <Students>
 <ID>3</ID>
 <NAME>王宁</NAME>
 <Phone>15896325689</Phone>
 <Address>徐家汇美罗城</Address>
 <City>上海</City>
</Students>
- <Students>
 <ID>4</ID>
 <NAME>黄韵琳</NAME>
 <Phone>13965669856</Phone>
 <Address>浦东新区</Address>
 <City>上海</City>
</Students>
</DocumentElement>
```

图 10-21 运行结果

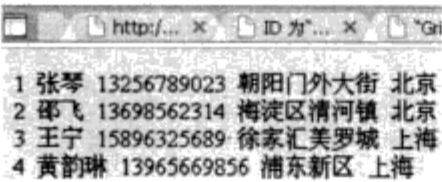


编号	姓名	电话	地址	地区
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海

图 10-22 运行结果

3. 使用 ASP.NET 4.0 提供的 Document 对象读取上机题 1 中所创建的 Students.xml 文件，并把其信息按照图 10-23 所示的格式显示出来。

4. 使用 Document 对象向上机题 1 中所创建的 Students.xml 文件中插入一条学生信息：编号：5，姓名：赵芳，电话：13658784596，地址：海淀区，地区：北京。运行结果如图 10-24 所示。



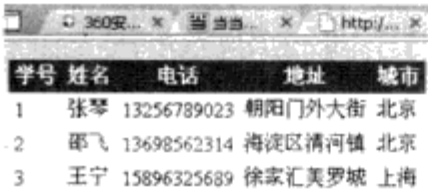
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海

图 10-23 运行结果

```
<Students>
 <ID>4</ID>
 <NAME>黄韵琳</NAME>
 <Phone>13965669856</Phone>
 <Address>浦东新区</Address>
 <City>上海</City>
</Students>
<Students>
 <ID>5</ID>
 <NAME>赵芳</NAME>
 <Phone>13658784596</Phone>
 <Address>海淀区</Address>
 <City>北京</City>
</Students>
</DocumentElement>
```

图 10-24 添加节点数据

- 5. 使用 Document 对象把上机题 4 中向 Students.xml 文件中插入的学生信息删除掉。
- 6. 利用 XmlDataSource 控件把 Students.xml 文件中的数据显示在 GridView 控件中，运行结果如图 10-25 所示。
- 7. 根据用户在文本框中输入的内容，从已经生成的 Students.xml 文件中检索符合条件的节点，程序运行结果如图 10-26 所示。



学号	姓名	电话	地址	城市
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海

图 10-25 运行结果



ID	NAME	Phone	Address	City
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13693562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海

请输入员工姓名:

图 10-26 运行结果

# 第 11 章 网站设计

## 学习目标

如何使网站页面的设计变得简单而高效一直都是开发人员需要考虑的重要课题，在 ASP 和 ASP.NET 早期版本的使用年代，二者很难得到兼顾，一直到出现了母版页机制和导航控件，这样的局面才得到了很大的改观，这二者更多的是作用于网站的外观而不是内部逻辑。母版页可以为应用程序中的页面创建统一的布局，实现了页面代码的重复使用。而导航控件使编程人员对站点导航的管理变得非常简单，几乎不用编写多少代码。如果读者能够熟练掌握这两个设计网站的技术，就会大大加快开发网站的速度。

## 本章重点

- 掌握母版页的创建和使用
- 熟练的进行站点地图的编写
- 使用 TreeView 控件绑定站点地图和 XML 文件数据
- 掌握 Menu 控件绑定数据的方法
- 利用 SiteMapPath 控件生成站点导航

## 11.1 母版页简介

母版页是 ASP.NET 提供的一种重用技术，使用母版页可以为应用程序中的页面创建一致的布局。单个母版页可以为应用程序中的所有页（或一组页）定义所需的外观和标准行为。然后可以创建包含要显示的内容的各个内容页。当用户请求内容页时，这些内容页与母版页合并以将母版页的布局与内容页的内容组合在一起输出。

### 11.1.1 母版页和内容页

母版页是具有扩展名.master 的 ASP.NET 文件，它具有可以包括静态文本、HTML 元素和服务器的预定义布局。母版页由特殊的 @Master 指令识别，该指令替换了用于普通.aspx 页的 @Page 指令。该指令的声明如下：

```
<%@ Master Language="C#" %>
```

@Master 指令中可以设置以下属性。

- CodeFile: 指定包含分部类的单独文件的名称，该分部类具有事件处理程序和特定于母版页的其他代码。

- **Debug**: 指示是否使用调试符号来编译母版页。如果要使用调试符号进行编译, 则为 `true`, 否则为 `false`。
- **Inherits**: 指定供页继承的代码隐藏类。它可以是从 `MasterPage` 类派生的任何类。
- **Language**: 指定在对页中所有内联呈现 (`<% %>` 和 `<%= %>`) 和代码声明块进行编译时使用的语言。值可以表示 .NET Framework 支持的任何语言, 包括 VB (Visual Basic)、C# 和 JScript。
- **SrC**: 指定在请求页时动态编译的代码隐藏类的源文件名称。可以选择将页的编程逻辑包含在代码隐藏类中或 .aspx 文件的代码声明块中。
- **MasterPageFile**: 指定用作某个母版页的 .master 文件, 定义嵌套母版页方案中的子母版页时, 在母版页中使用 `MasterPageFile` 属性。
- **AutoEventWireup**: 指是否可以使用语法 `Page` 且不使用任何显式挂钩或事件签名, 为特定的生命周期阶段定义简单的事件处理程序。如果启用了事件自动连接, 则为 `true`; 否则为 `false`。默认值为 `true`。
- **ClassName**: 指定自动从标记生成并在处理母版页时自动进行编译的类的类名, 此值可以是任何有效的类名, 并且可以包括命名空间。

例如, 下面的母版页指令包括一个隐藏代码文件的名称, 并将一个类名称分配给母版页。

```
<%@ Master Language="C#" CodeFile="MasterPage.master.cs" Inherits="MasterPage"%>
```

以上代码, 声明一个 `@Master` 指令, 设置程序语言为 C#, 设置 `CodeFile` 属性为隐藏代码文件的名称, 设置 `Inherits` 属性指定类名。

除了 `@Master` 指令外, 母版页还包含页的所有顶级 HTML 元素, 如 `html`、`head` 和 `form`。我们可以在母版页中使用任何 HTML 元素和 ASP.NET 元素。

除了在所有页上显示的静态文本和控件外, 母版页还包括一个或多个 `ContentPlaceHolder` 控件。`ContentPlaceHolder` 控件称为占位符控件, 这些占位符控件定义可替换内容出现的区域。

可替换内容是在内容页中定义的, 所谓内容页就是绑定到特定母版页的 ASP.NET 页 (.aspx 文件以及可选的代码隐藏文件), 通过创建各个内容页来定义母版页的占位符控件的内容, 从而实现页面的内容设计。

在内容页的 `@Page` 指令中通过使用 `MasterPageFile` 属性来指向要使用的母版页, 从而建立内容页和母版页的绑定。例如, 一个内容页可能包含 `@Page` 指令, 该指令将该内容页绑定到 `Master.master` 页, 代码如下。

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"%>
```

以上代码, 声明一个 `@Page` 指令, 设置程序语言为 C#, 设置母版页文件为 `MasterPage.master`, 设置启用事件自动连接。

在内容页的 `@Page` 指令中, 通过使用 `MasterPageFile` 属性来指向要使用的母版页, 从而建立内容页和母版页的绑定。例如, 一个内容页可能包含 `@Page` 指令, 该指令将该内容页绑定到 `Master.master` 页, 在内容页中, 通过添加 `Content` 控件并将这些控件映射到母版页上的 `ContentPlaceHolder` 控件来创建内容, 代码如下:

```

1. <%@Page Language="C#" MasterPageFile="~/Master.master" AutoEventWireup="true" %>
2. <asp:Content ID="Content1" ContentPlaceHolderID="Main" Runat="Server">
3. 主要内容
4. </asp:Content>

```

程序说明：第 1 行代码在 Page 指令中设置了属性 MasterPageFile 为 Master.master，表示该页面母版页为 Master.master，第 2 行声明 Content 控件并将这些控件映射到母版页上的 ContentPlaceHolder 控件。

创建 Content 控件后，就可以开始向这些控件添加文本和控件。在内容页中，Content 控件外的任何内容（除服务器代码的脚本外）都将导致错误。在 ASP.NET 页中所执行的所有任务都可以在内容页中执行。在母版页中创建为 ContentPlaceHolder 控件的区域在新的内容页显示为 Content 控件。

### 11.1.2 母版页的运行机制

母版页在运行时，按照下面的步骤进行处理。

- 用户通过键入内容页的 URL 来请求某页。
- 获取该页后，读取 @Page 指令。如果该指令引用一个母版页，则也读取该母版页。如果是第一次请求这两个页，则两个页都要进行编译。
- 包含更新内容的母版页合并到内容页的控件树中。
- 各个 Content 控件的内容合并到母版页中相应的 ContentPlaceHolder 控件中。
- 浏览器中呈现得到的合并页。

母版页提供了开发人员以传统的方式重复复制现有的代码、文本和控件元素，使用框架集，对通用元素使用包含文件和用户控件。母版页具有下面的优点：

- 使用母版页可以集中处理页的通用功能，以便可以只在一个位置上进行更新。
- 使用母版页可以方便地创建一组控件和代码，并将结果应用于一组页。例如，可以在母版页上使用控件来创建一个应用于所有页的菜单。
- 通过允许控制占位符控件的呈现方式，母版页使你可以在细节上控制最终页的布局。
- 母版页提供一个对象模型，使用该对象模型可以从各个内容页上自定义母版页。



提示

使用 Master 页面的一个优点是，在创建内容页面时，可以在 IDE 中看到模板，如果在处理页面时可以看到整个页面，就很容易开发出使用模板的内容页面，在处理内容页面时，所有的模板项都变灰且不能被编辑。

### 11.1.3 母版页的创建

母版页中包含的是页面公共部分，即网页模板。因此，在创建示例之前，必须判断哪些内容是页面公共部分，这就需要从分析页面结构开始。页面结构如图 11-1 所示。

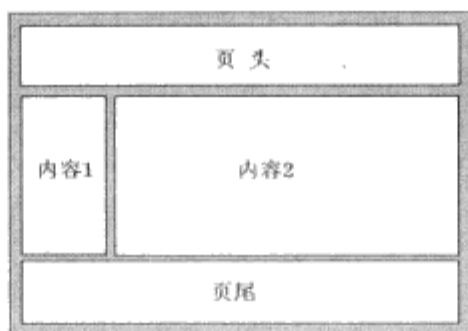


图 11-1 页面结构图

图 11-1 所示的页面由 4 个部分组成：页头、页尾、内容 1 和内容 2。其中页头和页尾是所在网站中页面的公共部分，网站中许多页面都包含相同的页头和页尾。内容 1 和内容 2 是页面的非公共部分，是页面所独有的。结合母版页和内容页的有关知识可知，如果使用母版页和内容页来创建页面，那么必须创建一个母版页 `MasterPage.master` 和一个内容页。其中母版页包含页头和页尾等内容，内容页中则包含内容 1 和内容 2。

**【例 11-1】** 本例演示如何实现创建一个母版页的过程。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 11-1”。
- 02** 在应用程序中创建一个 Images 的文件夹，其中包含页头背景图片文件“head.JPG”。
- 03** 用鼠标右键单击网站名，在图 11-2 所示的弹出菜单中选择“添加新项”命令。



图 11-2 添加新项

**04** 在弹出如图 11-3 的“添加新项”对话框中选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“母版页”，然后在“名称”文本框输入该文件的名称“`MasterPage.master`”，最后单击“添加”按钮。

**05** 此时在网站根目录下会创建一个如图 11-4 所示“`MasterPage.master`”文件和一个“`MasterPage.master.cs`”文件，前者是母版页页面设计文件，后者是代码编辑文件。

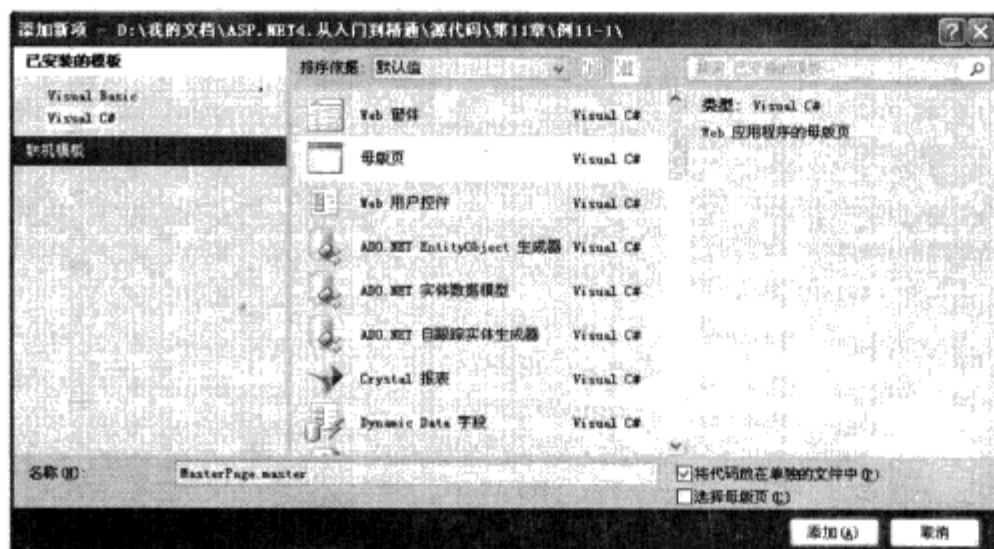


图 11-3 添加新项对话框

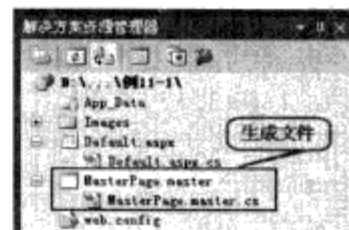


图 11-4 生成母版页文件

**06** 用鼠标双击打开“MasterPage.master”文件,编写代码如下。

```

1. <%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3. <html xmlns="http://www.w3.org/1999/xhtml">
4. <head runat="server">
5. <title></title>
6. </head>
7. <body leftmargin="0" topmargin="0" >
8. <form id="form1" runat="server">
9. <div align="center">
10. <table width="763" height="100%" border="0"
11. cellpadding="0" cellspacing="0" bgcolor="#FFFFFF">
12. <tr>
13. <td width="763" align="right"
14. valign="top" background="Images/head.jpg" class="style1">
15. </td>
16. </tr>
17. <tr>
18. <td width="763" valign="top">
19. <table width="100%" border="0"
20. cellspacing="0" cellpadding="0" style="height: 105px">
21. <tr>
22. <td width="244" valign="top">
23. <asp:ContentPlaceHolder
24. ID="ContentPlaceHolder1" runat="server">
25. <p style="height: 103px">
</p>
26. </asp:ContentPlaceHolder>
27. </td>
28. <td valign="top" align="left">
29. <asp:ContentPlaceHolder
30. ID="ContentPlaceHolder2" runat="server">
31. <p style="height: 103px">
</p>
32. </asp:ContentPlaceHolder>
33. </td>
34. </tr>
35. </table>
36. </td>
37. </tr>
38. <tr>
39. <td width="763" height="35" align="center" class="baseline">Copyright 2004-2010
40. </td>
41. </tr>
42. </table>
43. </div>
44. </form>
45. </body>
46. </html>

```

代码说明: 第 1 行声明一个@Master 指令。第 12 行通过一个 Table 元素构成整个页面结构。第 12~15 行定义表格的第 1 行第 1 列构成了页面中的页头,其中,第 13 行设置图片“Images/head.jpg”作为页头的背景图片。

第 17~37 行定义表格的第 2 行,这一行中嵌套了 1 个从第 19~35 行的 Table 元素,将这一行又分成了两个列。其中,第 21~27 行构成了第 1 个列,在第 23~26 行声明了控件 ContentPlaceHolder1,

用于在页面模板中为内容 1 占位。第 28~33 行构成了第 2 个列，在第 29~32 行声明了控件 ContentPlaceHolder2，用于在页面模板中为内容 2 占位。

第 38~41 行构成了页面中的页尾，显示一个网站的版本信息。

纵观整个代码，可以发现 MasterPage 页面与普通页面还是存在一定的差异，差异主要有：

- 第 1 行代码不同，母版页使用的是 Master，而普通.aspx 文件使用的是 Page。除此之外，二者在代码头方面是相同的。
- 是母版页中声明了控件 ContentPlaceHolder，而在普通.aspx 文件中是不允许使用该控件的。在 MasterPage.master 的源代码中，ContentPlaceHolder 控件本身并不包含具体内容设置，仅是一个控件声明。

**07** 切换到“源视图”，母版页的设计界面如图 11-5 所示。图中的两个矩形框表示 ContentPlaceHolder 控件。开发人员可以直接在矩形框中添加内容，所设置内容的代码将包含在 ContentPlaceHolder 控件声明代码中。



图 11-5 设计视图

使用 Visual Studio 2010 可以对母版页进行编辑，并且它完全支持“所见即所得”功能。无论是在代码模式下，还是设计模式下，使用 Visual Studio 2010 编辑母版页的方法与编辑普通.aspx 文件都是相同的。



提示

在应用程序中创建内容页面的 Page 指令时，使用 Title 属性可以指定页面的标题，这样就可以避免使用母版页中的标题了。

#### 11.1.4 内容页的创建

内容页用来定义母版页的占位符控件 ContentPlaceHolder 的内容，这些内容页为绑定到特定页母版页的 ASP.NET 页。通过包含指向要使用的母版页的 MasterPageFile 属性，在内容页的 @Page 指令中建立绑定。

内容页的创建有两种方法：第一种是在母版页中放入新建的内容页，第二种是在母版页中放入已经存在的内容页。

**【例 11-2】** 本例演示如何实现在母版页中放入新建的内容页面。

**01** 在“例 11-1”中，我们创建了一个母版页，本例在此基础上进行操作。用鼠标右键单击网站名称，在弹出快捷菜单中选择“添加新项”命令。

**02** 在弹出如图 11-6 所示的“添加新项”对话框中选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选“Web 窗体”，然后在“名称”文本框输入该文件的名称“Default.aspx”，

最重要的是选中“选择母版页”复选框，最后单击“添加”按钮。

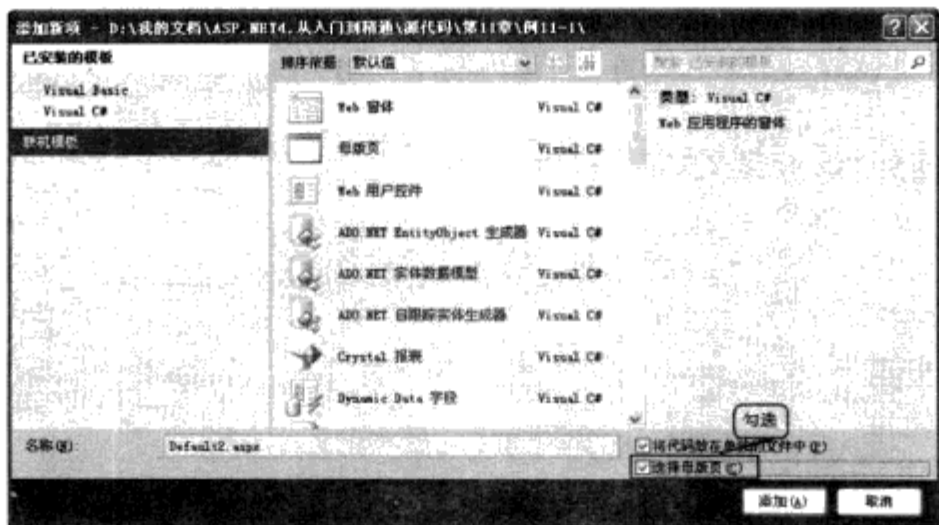


图 11-6 添加新项对话框

**03** 在弹出 11-7 所示的“选择母版页”对话框中，选择“文件夹内容”列表中“例 11-1”创建的母版页文件“MasterPage.master”，单击“确定”按钮，新创建的内容页就放入母版页中。

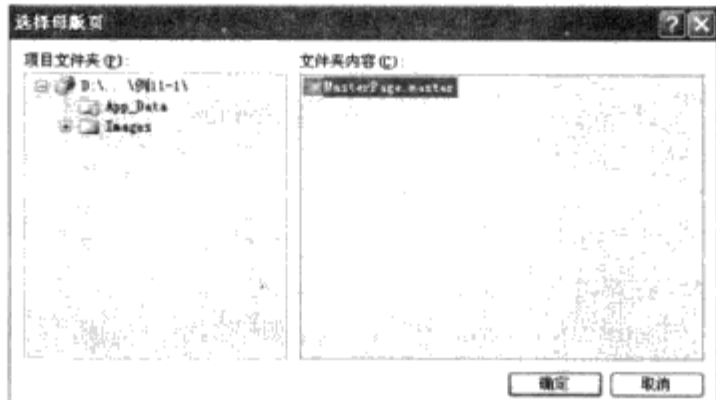


图 11-7 选择母版页对话框

**04** 用鼠标双击“Default.aspx”文件，编写代码如下。

```
1. <%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="Default2" %>
2. <asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
3. <asp:LinkButton ID="LinkButton1" runat="server">商品类别管理 </asp:LinkButton>

4. <asp:LinkButton ID="LinkButton2" runat="server">商品信息管理</asp:LinkButton>

5. <asp:LinkButton ID="LinkButton3" runat="server">商品订单管理</asp:LinkButton>

6. <asp:LinkButton ID="LinkButton4" runat="server">用户信息管理</asp:LinkButton>

7. <asp:LinkButton ID="LinkButton5" runat="server">个人资料管理</asp:LinkButton>

8. </asp:Content>
9. <asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2" Runat="Server">
10. <asp:Label ID="Label1" runat="server" Font-Size="XX-Large" style="color: #00345a; font-weight: 700;" Text="欢迎进入后台管理!"></asp:Label>
11. </asp:Content>
```

代码说明：第 1 行声明@Page 指令，其中 MasterPageFile 属性表示该页面是继承自母版页“MasterPage.master”。第 2 行通过添加 1 个 Content 控件 Content2，并将这些控件映射到母版页上的 ContentPlaceHolder1 控件来创建内容 1。在内容页面上，不比指定内容的位置，因为在母版页中定义了。所以只需要将适当的内容分在所提供的内容区域上。第 3~7 行就是添加要显示的内容，定义了 5 个服务器链接按钮控件，分别显示导航列表，即在母版页内容 1 中要显示的内容。第 9 行同样添加 1 个 Content

控件 Content3, 并将控件映射到母版页上的 ContentPlaceHolder2 控件来创建内容 2。第 10 行定义一个服务器标签控件来显示欢迎进入网站后台的信息, 即在母版页内容 2 中要显示的内容。在这个页面的代码中没有发现任何的 HTML 标记, 是因为它们都被包含在 Master.master 页面中了。

**05** 按下“Ctrl+F5”运行结果如图 11-8 所示。

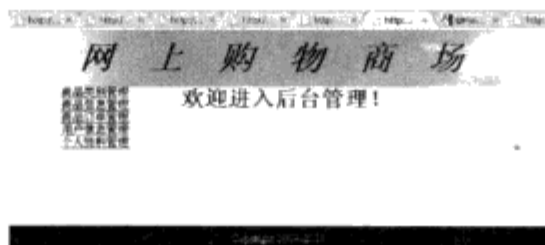


图 11-8 运行结果

另外一种方法, 在母版页放入已经存在的内容页, 通过手工加入或修改一些代码来使存在的网页嵌入到母版页中, 步骤如下。

**01** 进入已经存在的内容页的“源视图”, 在页面指示语句中增加与母版页相关联的属性, 代码如下。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
MasterPageFile="~/MasterPage.master"%>
```

代码说明: 实现的关键是 MasterPageFile 属性的设置, 它是与母版页相关联的属性, 只需将其值设置为相应的母版页文件所在路径即可。

**02** 删除原来代码中的 HTML 标记, 如<html>、<head>、<body>、<form>等, 因为母版页中已经存在相同的标记, 删除它们以避免重复。

**03** 增加<Content>标记, 并添加相应的属性, 也就是编写“例 11-2”中 Default.aspx 代码的第 2~11 行的内容。



内容页面继承某个母版页, 这一继承关系并不是一成不变的, 可以在内容页面的 Page\_PreInit 事件中变成动态指定一个母版页。

提示

## 11.2 网站导航

为了方便用户在网站中进行页面导航, 几乎每个网站, 都会使用页面导航控件。有了页面导航的功能, 用户可以很方便地在一个复杂的网站中进行页面之间的跳转。在以往的 Web 编程中, 要写一个好的页面导航功能, 并不是那么容易。ASP.NET 4.0 中提供了三种导航控件 TreeView 控件、Menu 控件和 SiteMapPath 控件, 同时提供了一个用于连接数据源的 SiteMapDataSource 控件, 利用三种导航控件与 SiteMapDataSource 控件相结合可以很轻松地实现页面导航功能。

### 11.2.1 网站地图

一个网站往往会包含很多页面, 而比较优秀的导航系统可以让用户很顺畅地在页面间进行穿

梭。显然，使用 ASP.NET 控件工具包可以实现几乎所有的导航系统，但真正实现起来还是需要完成很多麻烦的工作，然而 ASP.NET 所具有的一系列导航特性能够显著地简化这些工作。

ASP.NET 的导航是柔性的、可配置的，并且可插拔的，它主要包含三部分：

- 一种定义网站导航结构的方式，使用 XML 结构形式的网站地图文件来存储导航结构信息；
- 一种方便读取网站地图文件信息的方式，SiteMapDataSource 控件和 XmlSiteMapProvider 控件来实现这个功能；
- 一种把网站地图信息显示在用户浏览器上的方式，并且能够让用户方便地使用这个导航系统。可以使用绑定到 SiteMapDataSource 控件的导航控件来实现这个功能。

可以单独地扩展或自定义以上各个部分。例如，如果想要更改导航控件的外观，只需要把不同的控件绑定到 SiteMapDataSource 控件即可。如果想要从不同的类型或不同位置读取网站地图信息，只需要更改网站地图提供器即可。

ASP.NET 4.0 提供了名为 XmlSiteMapProvider 的网站地图提供器，使用 XmlSiteMapProvider 可以从 XML 文件中获取网站地图信息。如果要从其他位置或从一个自定义的格式获取网站地图信息，就需要创建定制的网站地图提供器，或者寻找一个第三方解决方案。

XmlSiteMapProvider 会从根目录中寻找名为 Web.sitemap 的文件来读取信息，它解析了 Web.sitemap 文件中的网站地图数据后创建一个网站地图对象，这个网站地图对象能够被 SiteMapDataSource 所使用，而 SiteMapDataSource 可以被放置在页面的导航控件所使用，最后由导航控件把网站的导航信息显示在页面上。



ASP.NET 的导航分层体系结构在底层的站点层次结构和 Web 站点上的控件之间制造了更为松散的耦合，提供了更大的灵活性，而且随着站点的不断发展，更容易实现体系结构和设计的驱动。

### 11.2.2 定义网站地图

网站地图的定义非常简单，可以直接使用文本编辑器进行编辑，还可以使用 Visual Studio 2010 创建，创建的站点地图文件可以自动生成组成网站地图的基本结构，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
 <siteMapNode url="" title="" description="">
 <siteMapNode url="" title="" description="" />
 <siteMapNode url="" title="" description="" />
 </siteMapNode>
</siteMap>
```

以上代码是自动生成的网站地图的基本结构信息组成代码，具体含义下面会介绍。

在添加了站点地图文件后，就可以按照自动生成的网站地图的基本结构添加适合本网站的数据信息。

下面介绍创建站点地图必须遵循的一些原则：

### (1) 网站地图以<siteMap>元素开始

每一个 Web.sitemap 文件都是以<siteMap>元素开始, 以与之相对的</siteMap>元素结束。其他信息则放在<siteMap>元素和</siteMap>元素之间, 代码如下:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
...
</siteMap>
```

以上代码中 xmlns 属性是必须的。如果使用文本编辑器编辑站点地图文件, 必须把上面代码中的 xmlns 属性值完全拷贝过去, 它告诉 ASP.NET, 这个 XML 文件使用了网站地图标准。

### (2) 每一页由<siteMapNode>元素来描述

每一个站点地图文件定义了一个网站的页面组织结构, 可以使用<siteMapNode>元素向这个组织结构插入一个页面, 这个页面将包含一些基本信息: 页面的名称(将显示在导航控件中)、页面的描述以及 URL(页面的链接地址)。代码如下:

```
<siteMapNode url="~/default.aspx" title="主页" description="网站的主页面" />
```

### (3) <siteMapNode>元素可以嵌套

一个<siteMapNode>元素表示一个页面, 通过嵌套<siteMapNode>元素可以形成树型结构的页面组织结构。代码如下:

```
<siteMapNode url="~/Default.aspx" title="主页" description="主页面">
 <siteMapNode url="~/WebForm1.aspx" title="页面 1" description="页面 1" />
 <siteMapNode url="~/WebForm2.aspx" title="页面 2" description="页面 2" />
</siteMapNode>
```

以上代码包含三个节点, 其中主页为顶层页面, 其他两个页面为下一级页面。

### (4) 每一个站点地图都是以单一的<siteMapNode>元素开始的

每一个站点地图都要包含一个根节点, 而所有其他的节点都包含在根节点中。

### (5) 不允许重复的 URL

在站点地图文件中, 可以没有 URL, 但不允许重复的 URL 出现, 这是因为 SiteMapProvider 以集合的形式来存储节点, 而每项是以 URL 为索引的。这样就会出现一个小问题, 如果想要在不同的层次引用相同的页面的话, 就不能实现了。但是只要稍微修改一下 URL 就照样可以使用站点地图文件来实现网站的导航, 代码如下:

```
<siteMapNode url="~/WebForm1.aspx?num=0" title="页面 1" description="页面 1" />
<siteMapNode url="~/WebForm1.aspx?num=1" title="页面 1" description="页面 1" />
```

以上代码是合法的, 虽然引用同样的页面, 但 URL 并不完全相同, 因此, 用户在这样的情况下就可以考虑适当修饰一下 URL, 即可突破不允许重复的 URL 的限制。

## 11.2.3 使用网站地图

当创建一个 Web.sitemap 文件后, 就可以在一个页面中使用它了。在一个页面上使用站点文件的步骤说明如下。

**01** 必须确定 Web.sitemap 文件列举的页面都已经存在于网站项目中，这些页面可以是空的，但必须存在，否则，在测试中就会出现问題。

**02** 在页面上添加一个 SiteMapDataSource 控件，可以从工具箱中把它拖拽到页面中，定义该控件的代码如下：

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

以上代码定义一个服务器网站地图控件 SiteMapDataSource1 并设置它的 ID 属性。在页面的“设计”视图中，SiteMapDataSource 控件呈现为一个灰色的方框，但它不会呈现在浏览器中。

**03** 添加一个绑定到 SiteMapDataSource 控件的导航控件。为了能够把导航控件与 SiteMapDataSource 控件联系起来，需要设置导航控件的属性 DataSourceID 为 SiteMapDataSource 控件的 ID，例如代码：

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1">
```

以上代码显示了一个 TreeView 控件的定义，其属性 DataSourceID 为 SiteMapDataSource1。

**【例 11-3】**在网站中创建一个商场积分后台管理导航的网站地图文件，并使用 TreeViwe 控件和 SiteMapDataSource 控件进行绑定显示。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 11-2”。

**02** 用鼠标右键单击网站名，在弹出的快捷菜单中选择“添加新项”命令。

**03** 在弹出如图 11-9 所示的“添加新项”对话框中选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“站点地图”，然后在“名称”文本框输入该文件的名称“Web.sitemap”，最后单击“添加”按钮。

**04** 此时在网站根目录下会创建一个如图 11-10 所示的“Web.sitemap”文件。

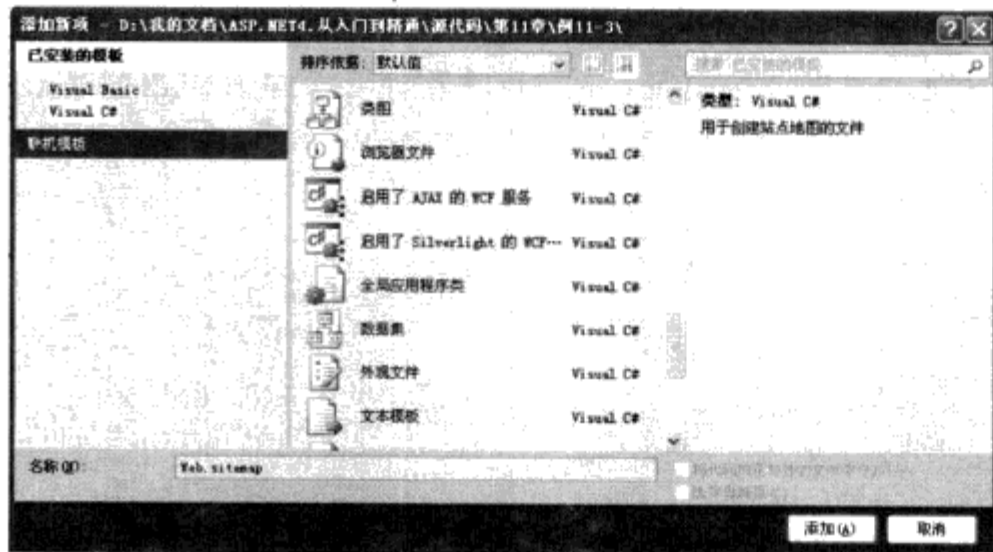


图 11-9 “添加新项”对话框

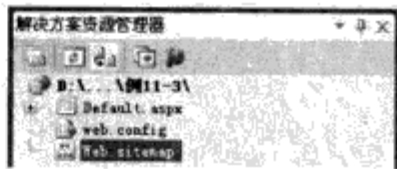


图 11-10 生成文件

**05** 用鼠标双击打开“Web.sitemap”文件，编写代码如下。

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
3. <siteMapNode url="Default.aspx" title="管理系统" description="">
4. <siteMapNode url="" title="商品管理" description="商品操作">
5. <siteMapNode url="MerchandiseSale.aspx" title="出售与退还" description="" />
```

```

6. <siteMapNode url="IntegralMerchandise.aspx" title="积分使用" description="" />
7. <siteMapNode url="Admin/IntegralUseRule.aspx" title="积分规则" description="" />
8. </siteMapNode>
9. <siteMapNode url="" title="卡类管理" description="会员卡操作">
10. <siteMapNode url="Admin/CardReset.aspx?PageView=AddCardType" title="添加卡类型" description="" />
11. <siteMapNode url="Admin/CardReset.aspx?PageView=UpdateCardType" title="卡类型修改" description="" />
12. <siteMapNode url="Admin/CardReset.aspx?PageView=UpdateRule" title="积分规则修改" description="" />
13. <siteMapNode url="Admin/CardReset.aspx?PageView=GetRule" title="积分规则获取" description="" />
14. </siteMapNode>
15. <siteMapNode url="" title="会员信息管理" description="会员详细信息">
16. <siteMapNode url="AddUserMember.aspx" title="会员信息添加" description="" />
17. <siteMapNode url="MemberInfoSelect.aspx" title="会员信息查询" description="" />
18. <siteMapNode url="MemberEdit.aspx" title="会员信息修改" description="" />
19. </siteMapNode>
20. <siteMapNode url="" title="积分管理" description="积分操作">
21. <siteMapNode url="IntegralInfo.aspx" title="积分查询" description="" />
22. <siteMapNode url="HistotySelect.aspx" title="积分历史记录" description="" />
23. <siteMapNode url="History.aspx" title="积分使用" description="" />
24. </siteMapNode>
25. </siteMapNode>
26. </siteMap>

```

代码说明：这段代码定义了一个具有三个层次的站点地图。第3行定义的是顶层的根节点“管理系统”。第2层是第4、9、15和20行定义“商品管理”、“卡类管理”、“会员信息管理”和“积分管理”的子节点。其中“商品管理”下又分别在第5~7行分别定义了三个低一级的子节点；“卡类管理”下在第10~13行分别定义了四个低一级的子节点；“会员信息管理”下在第16~18行分别定义了三个低一级的子节点；“积分管理”下在第21~23行分别定义了三个低一级的子节点。这些低一级的子节点构成了第三个层次。在每个层次中都有三个属性，URL 属性设置用于页面导航的地址，title 属性用于设置节点的名称，description 设置节点的说明文字。

**06** 用鼠标双击创建的网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动一个 TreeView 控件和 SiteMapDataSource 控件到“设计视图”，

**07** 将鼠标移到 TreeView 控件上，其上方会出现如图 11-11 所示一个向右的黑色小三角，单击它，弹出“TreeView 任务”列表。在“选择数据源”下拉列表中选中“SiteMapDataSource1”。

**08** 按下“Ctrl+F5”，运行结果如图 11-12 所示。

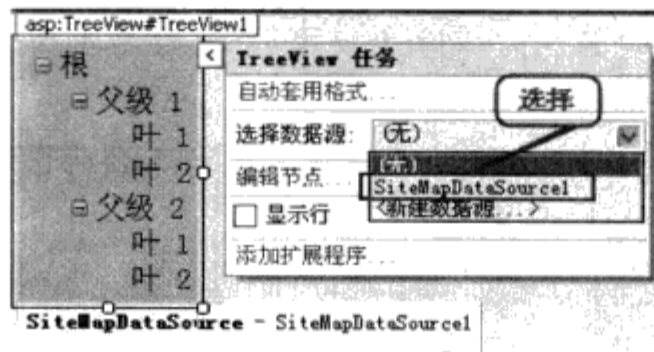


图 11-11 TreeView 任务列表

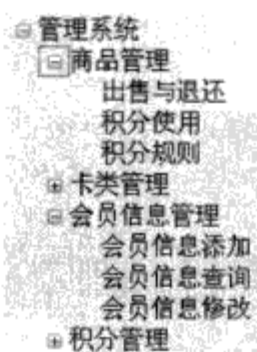


图 11-12 运行结果



提示

SiteMap 文件中的 siteMapNode 节点对应 SiteMapNode 类，该类包括几个用于描述网站中单个页（包括某一页）的属性，如 Url、Title 和 Description 属性，这些属性与 siteMapNode 节点的两个属性相对应。使用 SiteMapNode 类可以动态地对 SiteMap 文件进行编程。

## 11.3 导航控件

前面介绍了站点文件的相关知识，本节将介绍三种导航的控件 TreeView 控件、Menu 控件和 SiteMapPath 控件，这三种控件将以不同的形式在页面上展示网站的导航。

### 11.3.1 TreeView 控件

TreeView 控件以树型结构来对网站进行导航，它支持以下功能。

- 数据绑定，它允许控件的节点绑定到 XML、表格或关系数据。
- 站点导航，通过与 SiteMapDataSource 控件集成实现。
- 节点文本既可以显示为纯文本也可以显示为超链接。
- 借助编程方式访问 TreeView 对象模型以动态地创建树、填充节点、设置属性等。
- 客户端节点填充。
- 在每个节点旁显示复选框的功能。
- 通过主题、用户定义的图像和样式可实现自定义外观。

TreeView 控件由节点组成，树中的每一项都称为一个节点，它由一个 TreeNode 对象表示。节点有如下几种类型。

- 父节点，包含其他节点。
- 子节点，被其他节点包含。
- 叶节点，不包含子节点。
- 根节点，不被其他节点包含，同时是所有其他节点的上级节点。

一个节点可以同时为父节点和子节点，但不能同时为根节点、父节点和叶节点。而节点的类型决定着节点的可视化属性和行为属性。

TreeView 控件提供如表 11-1 所示的常用属性。

表 11-1 TreeView 控件的常用属性

属性	说明
CheckedNodes	获取 TreeNode 对象的集合，这些对象表示在 TreeView 控件中显示的选中了复选框的节点
ExpandDepth	获取或设置第一次显示 TreeView 控件时所展开的层次数
ImageSet	获取或设置用于 TreeView 控件的图像组
LevelStyles	获取 Style 对象的集合，这些对象表示树中各个级上的节点样式
MaxDataBindDepth	获取或设置要绑定到 TreeView 控件的最大树级数
Nodes	获取 TreeNode 对象的集合，它表示 TreeView 控件中根节点
ParentNodeStyle	获取对 TreeNodeStyle 对象的引用，该对象用于设置 TreeView 控件中父节点的外观
PathSeparator	获取或设置用于分隔由 ValuePath 属性指定的节点值的字符
RootNodeStyle	获取对 TreeNodeStyle 对象的引用，该对象用于设置 TreeView 控件中根节点的外观
SelectedNode	获取表示 TreeView 控件中选定节点的 TreeNode 对象

(续表)

属性	说明
SelectedNodeStyle	获取 TreeNodeStyle 对象，该对象控制 TreeView 控件中选定节点的外观
SeletedValue	获取选定节点的值
ShowCheckBoxes	获取或设置一个值，它指示哪些节点类型将在 TreeView 控件中显示复选框
ShowLines	获取或设置一个值，它指示是否显示连接子节点和父节点的线条
Target	获取或设置要在其中显示与节点相关联的网页内容的目标窗口或框架

【例 11-4】使用可视化编辑节点的方式创建一个旅游企业网站的 TreeView 控件导航。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 11-4”。
- 02 用鼠标双击创建的网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”拖动一个 TreeView 控件到“设计视图”。
- 03 将鼠标移到 TreeView 控件上，其上方会出现如图 11-13 所示一个向右的黑色小三角。单击它，弹出“TreeView 任务”列表。选择“编辑节点”命令。
- 04 在弹出图 11-14 的“TreeView 节点编辑器”中的“节点”列表中添加根节点，并在“属性”窗口中设置“Text”属性为“管理系统”。

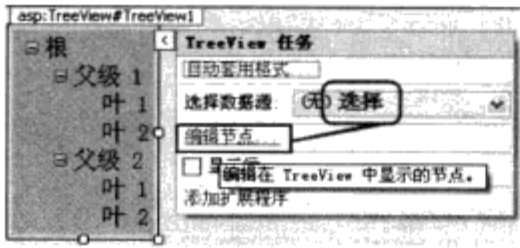


图 11-13 TreeView 任务列表

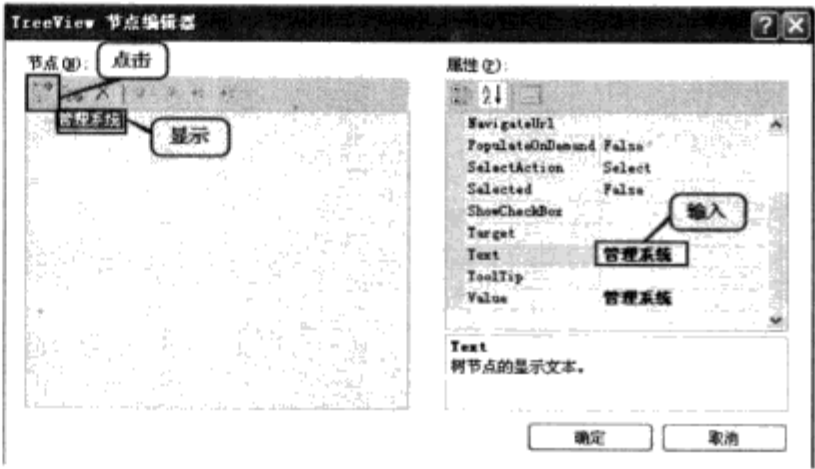


图 11-14 TreeView 节点编辑器

- 05 接着，在如图 11-15 所示“节点”列表中添加子节点，并在“属性”窗口中设置“Text”属性为“企业管理”。

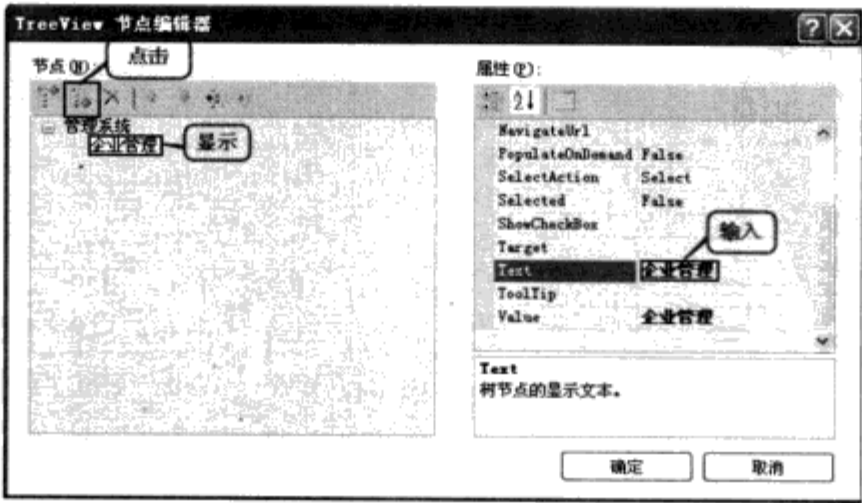


图 11-15 编辑子节点

**06** 继续在图 11-16 所示的“节点”列表中添加再次一级的节点，并在“属性”窗口中设置“Text”属性为“公司简介”。

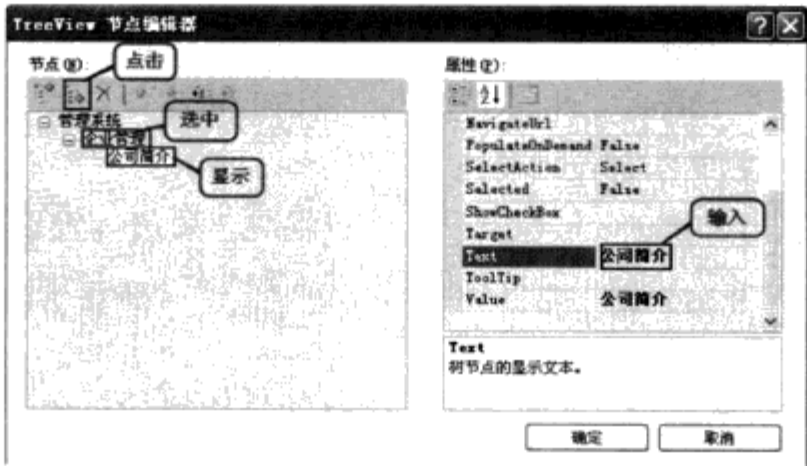


图 11-16 编辑底层节点

**07** 使用以上面添加节点的方法，完成所有 TreeView 控件节点的设置，单击“确定”按钮完成设置。在图 11-17 中单击 TreeView 控件右上角的箭头标记，弹出“TreeView”任务列表，选择其中的“自动套用格式”选项。

**08** 在弹出的“自动套用格式”对话框中选择“选择架构”列表里的“新闻”单击“确定”按钮。

**09** 按下“Ctrl+F5”，运行结果如图 11-18 所示。

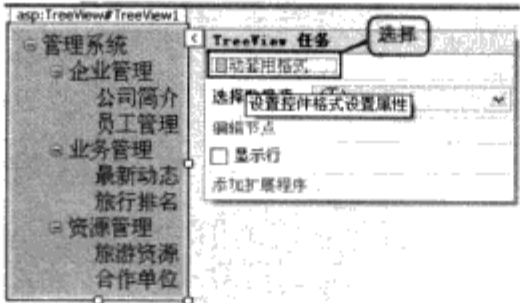


图 11-17 TreeView 任务



图 11-18 运行结果



提示

需要注意不要把 TreeView 控件的 ExpandAll 方法放在 Page\_Load 事件中，如果要这么做，应使用 TreeView 控件的 OnDataBound 属性，也还可以展开树中特定的节点，而不是展开整个列表。

### 11.3.2 Menu 控件

Menu 控件以菜单的结构形式来对网站进行导航，可以采用水平方向或竖直方向的形式导航，它支持以下功能。

- 通过与 SiteMapDataSource 控件集成，提供对站点导航的支持。
- 可以显示为可选择文本或超链接的节点文本。
- 通过编程访问 Menu 对象模型，使程序员可以动态地创建菜单，填充菜单项以及设置属性等。
- 能够采用水平方向或竖直方向的形式导航。

● 支持静态或动态的显示模式。

用户单击菜单项时，Menu 控件可以导航到所链接的网页或直接回发到服务器。如果设置了菜单项的 `NavigateUrl` 属性，则 Menu 控件导航到所链接的页；否则，该控件将页回发到服务器进行处理。默认情况下，链接页与 Menu 控件显示在同一窗口或框架中。若要在另一个窗口或框架中显示链接内容，请使用 Menu 控件的 `Target` 属性。

Menu 控件由菜单项（由 `MenuItem` 对象表示）树组成。顶级（级别 0）菜单项称为根菜单项，具有父菜单项的菜单项称为子菜单项，所有根菜单项都存储在 `Items` 集合中，子菜单项存储在父菜单项的 `ChildItems` 集合中。

每个菜单项都具有 `Text` 属性和 `Value` 属性。`Text` 属性的值显示在 Menu 控件中，而 `Value` 属性则用于存储菜单项的任何其他数据（如传递给与菜单项关联的回发事件的数据）。在单击时，菜单项可导航到 `NavigateUrl` 属性指示的另一个网页。

Menu 控件提供如表 11-2 所示的常用属性。

表 11-2 Menu 控件的常用属性

属性	说明
<code>DisappearAfter</code>	获取或设置鼠标指针不再置于菜单上后显示动态菜单的持续时间
<code>DynamicBottomSeparatorImageUrl</code>	获取或设置图像的 URL，该图像显示在各动态菜单项底部，将动态菜单项与其他菜单项隔开
<code>DynamicHoverStyle</code>	获取对 <code>Style</code> 对象的引用，使用该对象可以设置鼠标指针置于动态菜单项上时的菜单项外观
<code>DynamicItemFormatString</code>	获取或设置与所有动态显示的菜单项一起显示的附加文本
<code>DynamicItemTemplate</code>	获取或设置包含动态菜单自定义呈现内容的模板
<code>DynamicMenuItemStyle</code>	获取对 <code>MenuItemStyle</code> 对象的引用，使用该对象可以设置动态菜单中的菜单项的外观
<code>DynamicSelectedStyle</code>	获取对 <code>MenuItemStyle</code> 对象的引用，使用该对象可以设置用户所选动态菜单项的外观
<code>DynamicTopSeparatorImageUrl</code>	获取或设置图像的 URL，该图像显示在各动态菜单项顶部，将动态菜单项与其他菜单项隔开
<code>Items</code>	获取 <code>MenuItemCollection</code> 对象，该对象包含 Menu 控件中的所有菜单项
<code>LevelMenuItemStyles</code>	获取 <code>MenuItemStyleCollection</code> 对象，该对象包含的样式设置是根据菜单项在 Menu 控件中的级别应用于菜单项的
<code>LevelSelectedStyles</code>	获取 <code>MenuItemStyleCollection</code> 对象，该对象包含的样式设置是根据所选菜单项在 Menu 控件中的级别应用于该菜单项的
<code>MaximumDynamicDisplayLevels</code>	获取或设置动态菜单的菜单呈现级别数
<code>Orientation</code>	获取或设置 Menu 控件的呈现方向
<code>PathSeparator</code>	获取或设置用于分隔 Menu 控件的菜单项路径的字符
<code>SelectedItem</code>	获取选定的菜单项

(续表)

属性	说明
SelectedValue	获取选定菜单项的值
SkipLinkText	获取或设置屏幕读取器所读取的隐藏图像的替换文字，以提供跳过链接列表的功能
StaticBottomSeparatorImageUrl	获取或设置图像的 URL，该图像在各静态菜单项底部显示为分隔符
StaticDisplayLevels	获取或设置静态菜单的菜单显示级别数
StaticHoverStyle	获取对 Style 对象的引用，使用该对象可以设置鼠标指针置于静态菜单项上时的菜单项外观
StaticItemFormatString	获取或设置与所有静态显示的菜单项一起显示的附加文本
StaticItemTemplate	获取对 MenuItemStyle 对象的引用，使用该对象可以设置静态菜单中的菜单项的外观
StaticMenuStyle	获取对 MenuItemStyle 对象的引用，使用该对象可以设置静态菜单的外观
Target	获取或设置用来显示菜单项的关联网页内容的目标窗口或框架

**【例 11-5】** 将站点地图绑定到 Menu 控件，通过可视化的方式创建商场积分管理网站的菜单控件导航。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 11-5”。
- 02 在该网站中创建一个“Web.sitemap”文件。文件中的内容与“例 11-3”中的同名的文件相同。
- 03 用鼠标双击创建的网站根目录下的“Default.aspx”文件，进入到“视图设计器”。从“工具箱”拖动 1 个 Menu 控件到“设计视图”。
- 04 将鼠标移到 Menu 控件上，其上方会出现如图 11-19 所示一个向右的黑色小三角。单击它，弹出“Menu 任务”列表。展开“选择数据源”下拉列表，选择“编辑节点”命令。



图 11-19 Menu 任务列表

- 05 在弹出如图 11-20 所示的“数据源配置向导”对话框，在“应用程序从哪里获取数据”列表中选择“站点地图”，单击“确定”按钮。
- 06 在上面图 11-19 的“Menu 任务”列表中选择“自动套用格式”选项。在“自动套用格式”对话框中选择“选择架构”列表里的“彩色型”，然后单击“确定”按钮。

- 07
- 进入 Menu 控件“属性”窗口，设置 Orientation 属性为“Horizontal”表示菜单将以水平方向布局。
- 08
- 按下“Ctrl+F5”，运行结果如图 11-21 所示。

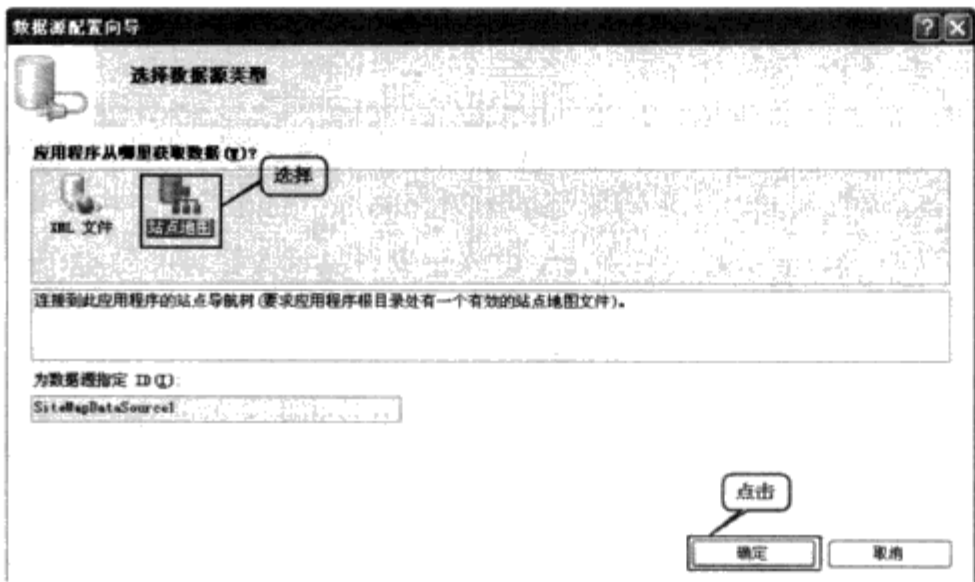


图 11-20 数据源配置向导对话框

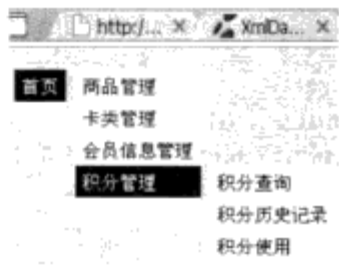


图 11-21 运行结果



在默认情况下，动态的菜单项从左到右显示。也就是说，菜单中的项在展开时，会以垂直方式显示，使用 Menu 控件的 Orientation 属性可以改变这种行为，把第一级菜单显示在第一个静态项的下面（水平）。

11.3.3 SiteMapPath 控件

SiteMapPath 控件显示一个导航路径，此路径为用户显示当前页的位置，并显示返回到主页的路径链接。SiteMapPath 控件包含来自站点地图的导航数据，此数据包括有关网站中页的信息，如 URL、标题、说明和导航层次结构中的位置。

SiteMapPath 控件使用起来非常简单，但却解决了很大的问题，在 ASP 和 ASP.NET 的早期版本中，当向网站添加一个页然后在网站内的其他各页中添加指向该新页的链接时，必须手动添加链接。现在只需要将导航数据存储在一个地方，通过修改该导航数据，就可以方便地在网站的导航栏目中添加和删除项。

SiteMapPath 由节点组成。路径中的每个元素均称为节点，用 SiteMapNodeItem 对象表示。锚定路径表示分层树的根节点称为根节点，表示当前显示页的节点称为当前节点，当前节点与根节点之间的任何其他节点都为父节点。SiteMapPath 包含如下几种节点类型。

- 根节点，锚定节点分层组的节点。
- 父节点，有一个或多个节点但不是当前节点的节点。
- 当前节点，表示当前显示页的节点。

SiteMapPath 控件提供如表 11-3 所示的常用属性。

表 11-3 SiteMapPath 控件的常用属性

属性	说明
CurrentNodeStyle	获取用于当前节点显示文本的样式
CurrentNodeTemplate	获取或设置一个控件模板，用于代表当前显示页的站点导航路径的节点
NodeStyle	获取用于站点导航路径中所有节点的显示文本的样式
NodeTemplate	获取或设置一个控件模板，用于站点导航路径的所有功能节点
ParentLevelsDisplayed	获取或设置控件显示的相对于当前显示节点的父节点级别数
PathDirection	获取或设置导航路径节点的呈现顺序
PathSeparator	获取或设置一个字符串，该字符串在呈现的导航路径中分隔 SiteMapPath 节点
PathSeparatorStyle	获取用于 PathSeparator 字符串的样式
PathSeparatorTemplate	获取或设置一个控件模板，用于站点导航路径的路径分隔符
Provider	获取或设置与 Web 服务器控件关联的 SiteMapProvider
RenderCurrentNodeAsLink	指示是否将表示当前显示页的站点导航节点呈现为超链接
RootNodeStyle	获取根节点显示文本的样式
RootNodeTemplate	获取或设置一个控件模板，用于站点导航路径的根节点
ShowToolTips	获取或设置一个值，该值指示 SiteMapPath 控件是否为超链接导航节点编写附加超链接属性。根据客户端支持，在将鼠标悬停在设置了附加属性的超链接上时，将显示相应的工具提示
SiteMapProvider	获取或设置用于呈现站点导航控件的 SiteMapProvider 的名称
SkipLinkText	获取或设置一个值，用于呈现替换文字，以让屏幕阅读器跳过控件内容

【例 11-6】使用站点地图 Web.sitemap 和 SiteMapPath 控件实现商场积分管理网站的网站导航功能。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 11-6”。
- 02 在该网站中创建一个“Web.sitemap”文件。文件中的内容与“例 11-3”中的同名的文件相同。
- 03 用鼠标双击创建的网站根目录下的“Default.aspx”文件，进入到“视图设计器”。从“工具箱”拖动 1 个 SiteMapPath 控件到“设计视图”。
- 04 将鼠标移到 SiteMapPath 控件上，其上方会出现如图 11-22 所示一个向右的黑色小三角，单击它，弹出“SiteMapPath 任务”列表。选择“自动套用格式”命令。
- 05 在弹出的“自动套用格式”对话框中，选择“选择架构”列表里的“彩色型”，单击“确定”按钮。
- 06 分别创建三个页面 History.aspx、HistotySelect.aspx 和 UseInfo.aspx 用于显示积分查询、积分历史记录和积分使用这三个页面。
- 07 在上面新创建的三个页面中各自添加一个 SiteMapPath 控件并设置相同的格式。
- 08 右键单击“History.aspx”文件，在弹出的菜单中选择“在浏览器中查看”命令，运行结果如图 11-23 所示。

09 右键单击“HistotySelect.aspx”文件，在弹出的菜单中选择“在浏览器中查看”命令，运行结果如图 11-24 所示。

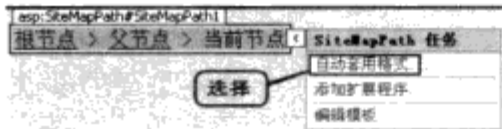


图 11-22 SiteMapPath 任务列表



图 11-23 运行结果 1



图 11-24 运行结果 2



提示

SiteMapPath 控件直接与站点文件建立连接，而 TreeView 控件和 Menu 控件则需要使用数据源控件 SiteMapDataSource 作为与站点文件连接的桥梁。这也是 SiteMapPath 控件与另外两个导航控件的区别，SiteMapPath 控件主要用来做导航的，而其他两个控件还可以有其他应用，比如，利用 TreeView 控件显示树型组织结构，利用 Menu 控件展示一些命令等。

## 11.4 上机题

1. 使用 MasterPage 母版页设计一个酒店管理系统的网站页面布局，运行结果如图 11-25 所示。



图 11-25 运行结果

2. 编写一个程序，要求使用 SiteMapPath 控件实现一个新闻网站的站点导航，运行结果如图 11-26 所示。
3. 使用 TreeView 控件实现企业门户网站后台导航功能，运行结果如图 11-27 所示。
4. 使用 TreeView 控件和 SiteMapDataSource 控件，以绑定站点地图的方法实现管理系统中的页面导航，运行结果如图 11-28 所示。



图 11-26 运行结果

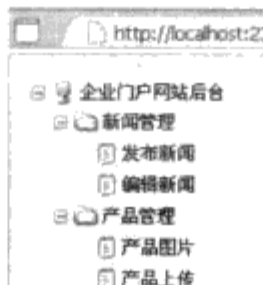


图 11-27 运行结果



图 11-28 运行结果

5. 使用 TreeView 控件和 SiteMapDataSource 控件，以绑定 XML 文件的方法实现管理系统中的页面导航，运行结果如图 11-29 所示。
6. 利用 Menu 控件设计一个新闻网站的页面菜单导航，运行结果如图 11-30 所示。
7. 创建一个 XML 文件，将其中的数据绑定到 Menu 控件，实现新闻网站页面的菜单层次导航，运行结果如图 11-31 所示。

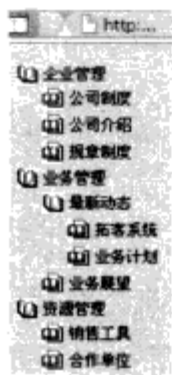


图 11-29 运行结果



图 11-30 运行结果

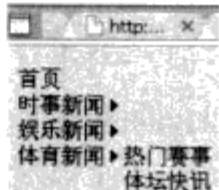


图 11-31 运行结果

# 第 12 章 主题和样式

## 学习目标

样式是对网站页面中 HTML 元素的外观进行设置和控制的主要手段，它可以让网页的样式独立出来，使批量处理样式更新的问题变得简单和方便。同时，通过统一的样式定义能够使网站的显示格式标准化。但是由于样式是基于 HTML 标准的，因此无法与 ASP.NET 元素很好地相结合。而 ASP.NET 4.0 中的主题机制不仅能完成样式所具有的功能，并且可以很好地与 ASP.NET 中的标准控件相结合。这样既能够节省网站的开发和维护费用，并且可以让网站页面变得更加专业。所以读者通过本章的学习，掌握这两种控制页面外观的方法是非常必要的。

## 本章重点

- 主题的创建和应用
- 掌握样式的基本语法
- 在页面中引用外部样式表
- 在 VS 2010 中使用样式创建器

## 12.1 主题

在 ASP.NET 4.0 中内置了主题机制，使得开发人员可以很轻松地对页面的设置实现更多选择。它在处理主题的设置时提供了清晰的目录结构，资源文件的层级关系非常清晰。在易于查找和管理的同时，提供了良好的扩展性。因此使用主题可以提高设计和维护网站的效率。

### 12.1.1 主题简述

ASP.NET 4.0 提供了一个新的功能——主题，用来集中制作每个页面、每个服务器控件或对象的外观样式，所以在 ASP.NET 4.0 中能很容易实现个性化皮肤的定制。主题以文本的方式集中设置样式，有效解决了应用程序界面风格统一和多样化的矛盾。

主题是有关页面和控件的外观属性设置的集合，由一组元素组成，包括外观文件、级联样式表（CSS）、图像和其他资源。

主题至少包含一个外观文件（.skin 文件），它是在网站或 Web 服务器上特殊目录中定义的，一般把这个特殊目录称为专用目录，这个专用目录的名字为“App\_Themes”。App\_Themes 目录下可以包含多个主题目录，主题目录的命名由程序员自己决定。而外观文件等资源则是放在主题目录下。主题的目录结构如图 12-1 所示，专用的目录 App\_themes 下包含三个主题目录，每个主题目录下包含了一个外观文件。

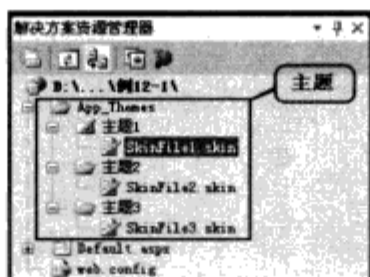


图 12-1 主题目录结构

### 1. 组成主题的元素

主题由外观、级联样式表（CSS）、图像和其他资源组成，它是在网站或 Web 服务器上的特殊目录中定义的。

#### （1）外观文件

外观文件又称皮肤文件，是具有文件扩展名 .skin 的文件，在皮肤文件里，可以定义控件的外观属性。皮肤文件形式一般具有下面代码的形式：

```
<asp:LinkButton runat="server" BackColor="Red"></asp: LinkButton >
```

上面的代码与定义一个 LinkButton 控件，除了不包含 ID、Text 等属性外，和通常定义 LinkButton 控件的代码几乎一样。但就是这样一行简单代码就定义了 LinkButton 控件的一个皮肤，可以在网页引用该皮肤去设置 LinkButton 控件的外观。

#### （2）级联样式表

级联样式表（Cascading Style Sheet）简称样式表。就是常用的 CSS 文件，是具有文件扩展名.css 的文件，也是用来存放定义控件外观属性的代码的文件。在页面开发中，采用级联样式表，可以有效地对页面的布局、字体、颜色、背景和其它效果实现更加精确的控制，而且只要对相应的代码做一些简单的修改，就可以改变同一页面的不同部分外观属性，或者页数不同的网页的外观和格式。正是级联样式表具有这样的特性，所以在主题技术中综合了级联样式表的技术。级联样式表一般具有下面代码的形式：

```
1. TextBox{
2. border-right: darkgray 1px ridge;
3. border-top: darkgray 1px ridge;
4. border-left: darkgray 1px ridge;
5. border-bottom: darkgray 1px ridge;
6. }
7. .Button{
8. font-size: x-small;
9. color: navy;
10. }
```

代码说明：第 1 行定义 TextBox 控件的样式。第 3~6 行分别设置控件边上下左右边框的颜色和粗细。第 8 行定义 Button 控件的样式。第 10 行设置控件上显示文字的尺寸。第 11 行设置控件的颜色。

#### （3）图像和其他资源

图像就是图形文件，其他资源可能是声音文件、脚本文件等。有时候为了控件美观，只是靠

颜色、大小和轮廓来定义并不能满足要求,这时候就会考虑把一些图片、声音等加到控件外观属性定义中去。例如可以在为 LinkButton 控件的单击加上特殊的音效,为 TreeView 控件的展开和收起按钮定义不同的图片。

## 2. 主题应用范围

主题的应用范围可以分为以下两种:

(1) 页面主题应用于单个 Web 应用程序,它是一个主题文件夹,其中包含控件外观、样式表、图形文件和其他资源,该文件夹是作为网站中的\App\_Themes 文件夹的子文件夹创建的。每个主题都是\App\_Themes 文件夹的一个不同的子文件夹。

(2) 全局主题可应用于服务器上的所有网站,全局主题与页面主题类似,因为它们都包括属性设置、样式表设置和图形。但是,全局主题存储在对 Web 服务器具有全局性质的名为 Themes 的文件夹中。服务器上的任何网站以及任何网站中的任何页面都可以引用全局主题。

## 3. 使用主题的注意点

- 主题只在每个页面、每个服务器控件或对象中才有效。
- 母版页 (Master Page) 上不能设置主题,但是主题可以在内容页面上设置。
- 主题上设置的服务器控件的样式会覆盖页面上设置的样式。
- 如果在页面上设置 “EnableTheming” 属性为 false,则主题设置无效。
- 要在页面中动态设置主题,必须在页面生命周期 Page\_Preinit 事件之前。



外观文件中对控件的定义也包含 runat="server" 属性,但是外观文件中控件定义不包含 ID 属性,外观文件中不能设置该控件不支持的外观属性。

### 12.1.2 主题的创建

【例 12-1】本例将学习如何在 Visual Studio 2010 的网站中创建一个主题和外观文件。

**01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 12-1”。

**02** 用鼠标右键单击网站名,在弹出的图 12-2 的快捷菜单中选择“添加 ASP.NET 文件夹”| “主题”命令。

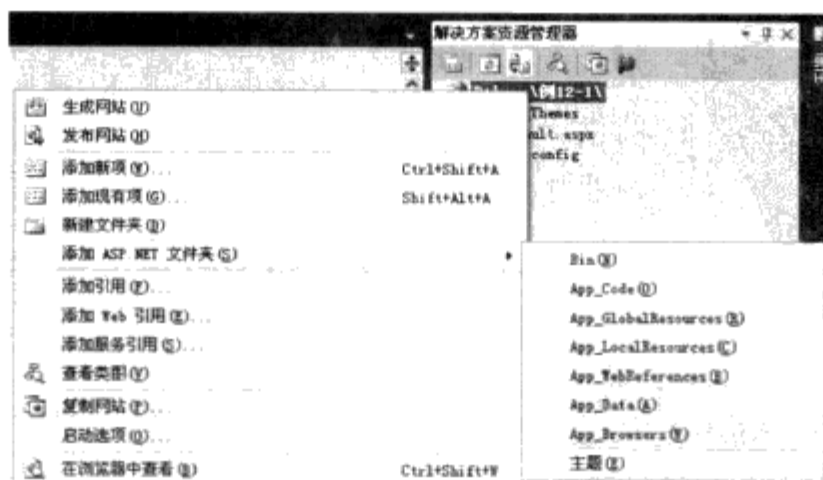


图 12-2 添加 ASP.NET 文件夹

**03** 此时就会在该网站项目下添加一个名为“App\_Themes”文件夹，并在该文件夹中自动添加一个默认名为“主题1”主题文件夹。

**04** 右键单击“主题1”的文件夹，在弹出如图 12-3 的快捷菜单里选择“添加新项”命令。

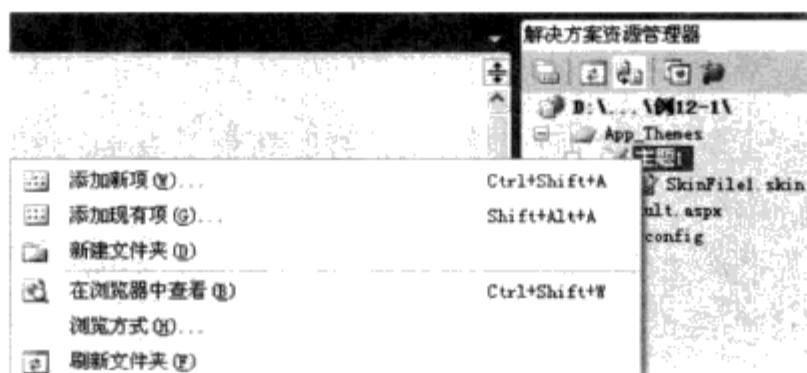


图 12-3 添加新项

**05** 弹出如图 12-4 所示的“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“外观文件”，然后在“名称”文本框输入该文件的名称“SkinFile.skin”，最后单击“添加”按钮。

**06** 此时在“主题1”的文件夹会生成一个如图 12-5 所示的“SkinFile.skin”皮肤文件。

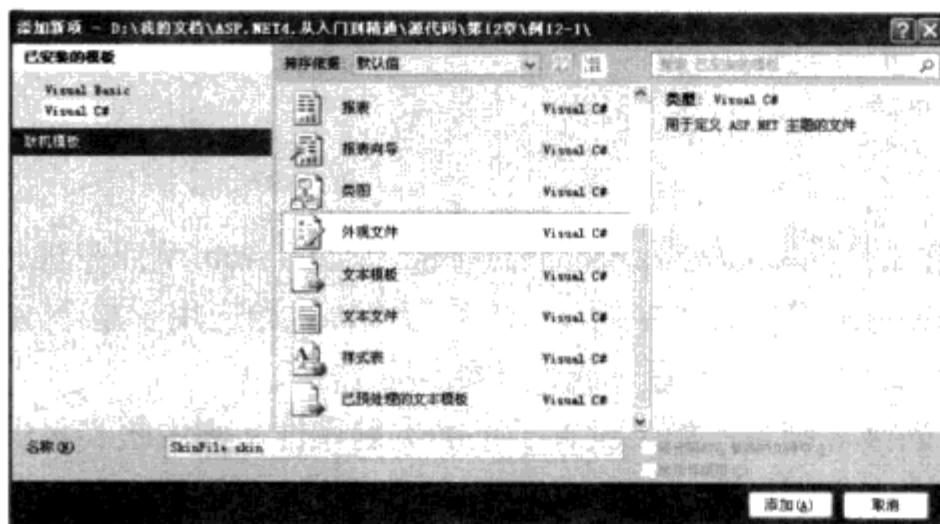


图 12-4 添加新项对话框

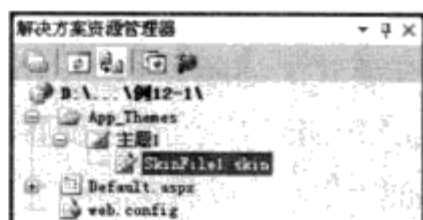


图 12-5 生成皮肤文件

**07** 用鼠标双击生成的“SkinFile.skin”文件，打开该文件，在里面可以看到的代码如下：

```
1. <%--
2. 默认的外观模板。以下外观仅作为示例提供。
3. 1. 命名的控件外观。SkinId 的定义应唯一，因为在同一主题中不允许一个控件类型有重复的 SkinId。
4. <asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >
5. <AlternatingRowStyle BackColor="Blue" />
6. </asp:GridView>
7. 2. 默认外观。未定义 SkinId。在同一主题中每个控件类型只允许有一个默认的外观。
8. <asp:Image runat="server" ImageUrl="~/images/image1.jpg" />
9. --%>
```

上面代码是一段对外观文件编写的说明性文字，告诉开发人员以何种格式来编写控件的外观属性定义。其中，第 4 行和第 8 行提供了两个外观定义的示例，一个是服务器列表控件，另一个是服务器图像控件 Image。

**08** 按照以上的说明格式，编写一个 Button 控件的外观属性定义，代码如下：

```
<asp:Button runat="server" SkinID="Steady" BackColor="Black" ForeColor="White" ></asp:Button >
```

代码说明：定义了一个服务器按钮控件 Button 标签控件的外观，设置其 SkinID 属性的名称以及背景和文字的显示颜色。

通过以上步骤，一个可以应用于整个网站项目的主题就建立完成了。



提示

在建立主题的时候需要注意以下几项：① 主题目录放在专用目录 App\_Themes 的下面；② 专用目录下可以放多个主题目录；③ 皮肤文件放在“主题目录”下；④ 每个主题目录下面可以放多个皮肤文件，但系统会把多个皮肤文件合并在一起，把这些文件视为一个文件；⑤ 对控件显示属性的定义放在以“.skin”为后缀的皮肤文件中。

### 12.1.3 主题的应用

在网页中使用某个主题都会在网页定义中加上“Theme=[主题目录]”的属性，代码如下：

```
<%@ Page Theme="主题 1" ... %>
```

为了将主题应用于整个项目，可以在项目的根目录下的 Web.config 文件里进行配置，示代码如下：

```
1. <configuration>
2. <system.web>
3. <Pages Themes="主题 1"></Pages>
4. </system.web>
5. </configuration>
```

代码说明：主题的配置是在配置文件的 configuration 节点下的 system.web 节点中进行。第 3 行代码通过使用 Pages 节点把属性 Themes 设置为“主题 1”，从而将该主题应用于整个项目。

只有遵守上述配置规则，在皮肤文件中定义的显示属性才能够起作用。

在 ASP.NET 中，属性设置的优先级规则是：如果设置了页的主题属性，则主题和页中的控件设置将进行合并，以构成控件的最终属性设置。如果同时在控件和主题中定义了同样的属性，则主题中的控件属性设置将重写控件上的任何页设置。使用这种属性规则的明显好处是：通过主题可以为页面上的控件定义统一的外观，同时如果修改了主题的定义，页面上的控件的属性也会跟着发生统一的变化。

**【例 12-2】**本例实现动态加载页面主题的功能，通过下拉列表中选择的两个选项，使日历控件显示不同的主题。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 12-2”。

**02** 在网站中创建两个主题目录，并在主题目录下各创建一个皮肤文件，如图 12-6 所示。

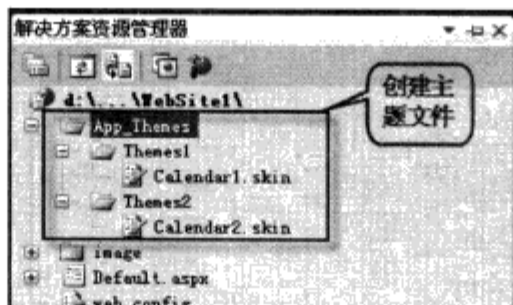


图 12-6 创建主题

**03** 用鼠标双击创建网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 DropDownList 控件、2 个 TextBox 控件和 1 个 Button 控件到“设计视图”中。切换到“源视图”，添加如下代码：

```

1. <table style="width: 225px;background-image: url(image/主题背景图片.gif)" border="1">
2. <tr>
3. <td>欢迎光临本网</td>
4. </tr>
5. <tr>
6. <td style="height: 2px"><asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="true"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged" Width="105px">
7. <asp:ListItem Value="Themes1">黄色主题日历</asp:ListItem>
8. <asp:ListItem Value="Themes2">蓝色主题日历</asp:ListItem>
9. </asp:DropDownList>博客日历
10. <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
11. </td>
12. </tr>
13. <tr><td class="style4">最新博客列表</td></tr>
14. <tr><td class="style1">
15. <asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="http://www.google.com" style="font-size: small">个人
日记</asp:HyperLink></td>
16. </tr>
17. <tr><td class="style1"><asp:HyperLink ID="HyperLink3" runat="server" NavigateUrl="http://www.google.com"
style="font-size: small">天下杂侃</asp:HyperLink></td>
18. </tr>
19. <tr><td class="style1"><asp:HyperLink ID="HyperLink2" runat="server" NavigateUrl="http://www.google.com"
style="font-size: small">情感天地</asp:HyperLink></td>
20. </tr>
21. </table>

```

代码说明：第 1 行定义一个 7 行 1 列的表格控件并设置其宽度、背景图片和边框。第 6 行定义一个下拉列表控件并设置其自动回传服务器属性和定义选中项改变事件的方法。第 7 和第 8 行给 DropDownList1 控件添加两个列表项。第 10 行定义 1 个服务器日历控件 Calendar1。第 15~19 行各自定义一个服务器超链接控件 HyperLink，并设置其文字的大小、链接的 Url 地址和显示的文本。

**04** 用鼠标双击新建的“Calendar1.skin”文件，在其中添加如下代码：

```

1. <asp:Calendar runat="server" BackColor="#FFFFCC" BorderColor="#FFCC66" BorderWidth="1px" DayNameFormat="Shortest"
2. FontNames="Verdana" Font-Size="8pt" ForeColor="#663399" Height="200px" ShowGridLines="True" Width="220px">
3. <SelectedDayStyle BackColor="#CCCCFF" Font-Bold="True" />
4. <TodayDayStyle BackColor="#FFCC66" ForeColor="White" />
5. <SelectorStyle BackColor="#FFCC66" />
6. <OtherMonthDayStyle ForeColor="#CC9966" />
7. <NextPrevStyle Font-Size="9pt" ForeColor="#FFFFCC" />
8. <DayHeaderStyle BackColor="#FFCC66" Font-Bold="True" Height="1px" />
9. <TitleStyle BackColor="#990000" Font-Bold="True" Font-Size="9pt" ForeColor="#FFFFCC" />
10. </asp:Calendar>

```

代码说明：第 1~9 行定义了一个蓝色皮肤的 Calendar 控件。其中，第 1 行设置控件的背景色、边框颜色、边框的宽度和各天名称的格式。第 2 行设置控件的文字格式和大小、前景色和大小尺寸。第 3 行设置用户选中日期的背景色、字体和前景色。第 4 行设置当前日期的背景色、前景色。第 5 行设置位于左侧列的背景色和前景色。第 6 行设置上个月和下个月中天的颜色。第 7 行设置标题栏左端和右端放置月份的颜色和文字大小。第 8 行设置日历上方显示日名称的背景色、前景色和高度。

第9行设置控件顶端标题栏的背景色、前景色、边框颜色和宽度、字体和高度。

**05** 用鼠标双击新建的“Calendar2.skin”文件，在其中添加如下代码：

```
1. <asp:Calendar runat="server" BackColor="White" BorderColor="#3366CC" BorderWidth="1px" CellPadding="1"
DayNameFormat="Shortest" Font-Names="Verdana" Font-Size="8pt" ForeColor="#003399" Height="200px" Width="220px">
2. <SelectedDayStyle BackColor="#009999" Font-Bold="True" ForeColor="#CCFF99" />
3. <TodayDayStyle BackColor="#99CCCC" ForeColor="White" />
4. <SelectorStyle BackColor="#99CCCC" ForeColor="#336666" />
5. <WeekendDayStyle BackColor="#CCCCFF" />
6. <OtherMonthDayStyle ForeColor="#999999" />
7. <NextPrevStyle Font-Size="8pt" ForeColor="#CCCCFF" />
8. <DayHeaderStyle BackColor="#99CCCC" ForeColor="#336666" Height="1px" />
9. <TitleStyle BackColor="#003399" BorderColor="#3366CC" BorderWidth="1px" Font-Bold="True" Font-Size="10pt"
ForeColor="#CCCCFF" Height="25px" />
10. </asp:Calendar>
```

代码说明：这里的代码和“Calendar1.skin”文件中的代码含义大都一致，只是属性设置的值不同，就不再重复说明了。

**06** 用鼠标双击“Default.aspx.cd”文件，在其中添加如下代码：

```
1. <asp:Calendar runat="server" BackColor="White" BorderColor="#3366CC" BorderWidth="1px" CellPadding="1"
DayNameFormat="Shortest" Font-Names="Verdana" Font-Size="8pt" ForeColor="#003399" Height="200px" Width="220px">
2. <SelectedDayStyle BackColor="#009999" Font-Bold="True" ForeColor="#CCFF99" />
3. <TodayDayStyle BackColor="#99CCCC" ForeColor="White" />
4. <SelectorStyle BackColor="#99CCCC" ForeColor="#336666" />
5. <WeekendDayStyle BackColor="#CCCCFF" />
6. <OtherMonthDayStyle ForeColor="#999999" />
7. <NextPrevStyle Font-Size="8pt" ForeColor="#CCCCFF" />
8. <DayHeaderStyle BackColor="#99CCCC" ForeColor="#336666" Height="1px" />
9. <TitleStyle BackColor="#003399" BorderColor="#3366CC" BorderWidth="1px" Font-Bold="True" Font-Size="10pt"
ForeColor="#CCCCFF" Height="25px" />
10. </asp:Calendar>
```

代码说明：第1行定义处理下拉列表控件DropDownList1选中项更改事件SelectedIndexChanged的方法。第2行设置访问页面的路径，其中将DropDownList1中选择项的值作为theme的参数值传递到页面。第3行跳转到访问页面。第5行定义页面对象Page的PreInit事件的方法，使用编程方式动态指定主题必须在这一事件中进行。第7行判断如果传递到页面中的theme参数中的值不为空，就设置字符串对象的theme的值为“Themes1”。否则，第11行将theme参数中的值赋给theme。第12行通过页面对象Page的Theme属性调用theme表示的主题。第14行获得当前文本框中选中项的ListItem对象。

第15行判断如果该项不为空，将项在DropDownList1中作为选择项。

**07** 按下“Ctrl+F5”，运行结果如图12-7所示，显示默认的主题日历。

**08** 用户如果选择下列列表中的“蓝色主题日历”选项，结果如图12-8所示，显示不同的控件外观。



图 12-7 运行结果 1



图 12-8 运行结果 2

### 12.1.4 SkinID 的应用

SkinID 是 ASP.NET 为 Web 控件提供的一个联系到皮肤的属性,用来标识控件使用哪种皮肤。有时需要同时为一种控件定义不同的显示风格,这时可以在皮肤文件中定义 SkinID 属性来区别不同的显示风格。以下代码中对 Label 控件定义了三种不同的皮肤:

```
1. <asp:Textbox runat="server" CssClass="commonText"></asp: Textbox >
2. <asp: Textbox runat="server" CssClass="MsgText" SkinID="MsgText"></asp: Textbox >
3. <asp: Textbox runat="server" CssClass="PromptText" SkinID="PromptText"></asp: Textbox >
```

代码说明:第1行代码是默认定义,不包含 SkinID 属性,该定义作用于所有不声明 SkinID 属性的 Label 控件。第2和第3行代码声明了 SkinID 属性,当使用其中一种样式定义时,就需要在相应的 Label 控件里声明相应的 SkinID 属性。

**【例 12-3】**使用 SkinID 来选择不同的外观定义,从而实现 2 个 TextBox 控件和 2 个 Button 控件的不同显示。

- 01** 启动 Visual Studio 2010,创建一个 ASP.NET Web 应用程序,命名为“例 12-3”。
- 02** 在网站中创建 1 个主题目录“主题 1”,并在该目录下创建一个皮肤文件“SkinFile.skin”。
- 03** 用鼠标双击创建网站的根目录下“Default.aspx”文件,进入到“视图设计器”。从“工具箱”分别拖动 2 个 TextBox 控件和 2 个 Button 控件到“设计视图”中。切换到“源视图”,添加如下代码:

```
1. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" Theme="主题 1" %>
2. <h1> 会员登录返回首页>></h1>
3. <table width="300" border="1" cellspacing="0" cellpadding="0">
4. <tr>
5. <td style="width:100px;">会员名</td>
6. <td><asp:TextBox ID="TextBox1" SkinID="Text1" runat="server"></asp:TextBox></td>
7. </tr>
8. <tr>
9. <td style="width:100px;">密码</td>
10. <td><asp:TextBox ID="TextBox2" SkinID="Text2" TextMode="Password" Text="123456"
runat="server"></asp:TextBox></td>
11. </tr>
12. </table>
13. <p>
```

```
14. <asp:Button ID="Button1" SkinID ="Button1" runat="server" Text="登录" /> &~
15. <asp:Button ID="Button2" SkinID ="Button2" runat="server" Text="退出" />
```

代码说明：第 1 行定义@Page 指令，设置关键属性 Theme 的值为主题目录的名称“主题 1”，表明把“主题 1”应用于该页面。第 2 行定义一个标题文本。第 3~12 行定义了一个 2 行 2 列的表格。其中第 6 行定义一个服务器文本框控件 TextBox1 并设置 SkinID 的属性。第 10 行定义一个服务器文本框控件 TextBox2 并设置 SkinID 的属性和文本模式已经显示的文本。第 14 和 15 行各自定义一个服务器按钮控件并设置 SkinID 属性和显示的文本。

**04** 用鼠标双击新建的“SkinFile1.skin”文件，在其中添加如下代码。

1. `<asp:TextBox runat="server" SkinID="Text1" Text="huiyuan001" style="width:180px; background-color:#6699CC; height:18px;color:#FFFFFF;"></asp:TextBox>`
2. `<asp:TextBox runat="server" SkinID="Text2" style="width:180px; background-color:#6699CC;height:18px;color:#FFFFFF;"></asp:TextBox>`
3. `<asp:Button runat="server" SkinID="Button1" style="width:50px;height:20px;background-color:#CCFF00;border:#0000CC 2px groove;cursor:pointer;"></asp:Button>`
4. `<asp:Button runat="server" SkinID="Button2" style="width:50px;height:20px;background-color:#6666CC;border:#0000CC 2px groove;cursor:pointer;color:#FFFFFF;"></asp:Button>`

代码说明：第 1 行是定义文本框控件的皮肤外观，设置 SkinID 属性和页面控件 TextBox1 相关联，当使用其中一种样式定义时就需要在相应的控件里声明相应的 SkinID 属性。设置该控件显示的文字、大小、字体颜色和背景颜色。第 2 行同样是定义 1 个文本框控件的皮肤外观，设置 SkinID 属性和页面控件 TextBox2 相关联。设置该控件的大小、字体颜色和背景颜色。第 3 行定义按钮控件的外观皮肤，设置 SkinID 属性和页面控件 Button1 关联。设置该控件的大小、背景样式、边框的颜色、粗细、样式以及鼠标经过时显示的样式。定义按钮控件的外观皮肤，设置 SkinID 属性和页面控件 Button2 关联。设置该控件的大小、背景样式、边框的颜色、粗细、样式、鼠标经过时显示的样式和控件的颜色。

**05** 按下“Ctrl+F5”，运行结果如图 12-9 所示。可以看到 4 个控件显示的颜色或外观各不相同。



图 12-9 运行结果

### 12.1.5 禁用主题

主题将重写页和控件外观的本地设置，而当控件或页已经有预定义的外观，且又不希望主题重写它时，就可以利用禁用方法来忽略主题的作用。

禁用页的主题通过设置@Page 指令的 EnableTheming 属性为 false 来实现，例如：

<%@ Page EnableTheming="false" %>

以上代码使用 `EnableTheming` 属性设置主题的禁用。

禁用控件的主题通过将控件的 `EnableTheming` 属性设置为 `false` 来实现，例如：

```
<asp:Calendar id="Calendar1" runat="server" EnableTheming="false" />
```

以上代码实现在控件中使用 `EnableTheming` 属性设置主题的禁用。



提示

必须在页面上静态控件的 `Page_PreInit` 事件触发之前设置 `Page` 属性的主题。如果使用动态控件，就应该在把控件添加到 `Controls` 集合之前，设置 `Theme` 属性。

## 12.2 样式

CSS 是 “Cascading Style Sheet” 的简称，中文的意思就是 “层叠样式表”，简称 “样式表”。它是一种用户增强控制页面样式，并允许将样式信息与页面内容分离的标记性语言。它可以很容易地控制页面中的 HTML 元素的背景与颜色、元素框的样式、定位、文字字体等属性。

CSS 的作用可以概括为以下几点：

- 内容与表现分离。
- 表现的统一，可以使网页的表现非常统一，并且容易修改。
- 减少重复代码的编写。
- 增加网页的浏览速度。
- 减少硬盘容量。

CSS 按其在 HTML 文档中的位置可以分为三种：

### （1）内嵌样式表

内嵌样式表的 CSS 是写在 HTML 标签里面的，只对当前的标签起作用。

### （2）内部样式表

内部样式表是写在 HTML 的 `<head>` 和 `</head>` 里面，由 `<style>` 和 `</style>` 标记，内部样式表只对所在的网页有效。

### （3）外部样式表

为了能够让很多页面共享同样的样式表，可以把样式表的定义写在一个扩展名为 `.css` 的文件里，然后在每个需要使用该样式表的页面添加对该样式表文件的引用。

### 12.2.1 样式的语法

CSS 也是一种语言，也有着自己语法结构，下面就简要介绍一下 CSS 的语法。

最基本的 CSS 是由三个部分构成：选择符 (selector)、属性 (properties) 和属性的取值 (value)，格式如下：

选择符{属性: 值}，即: `selector{property: value}`

一般情况下，选择符是要为之定义样式的 HTML 标记，例如 BODY、P、TABLE 等等，可以通过此方法定义相应标记的属性和值，属性和值之间用冒号隔开，例如：

```
body {color: black}
```

以上代码中，选择符 body 是指页面主体部分，color 是控制文字颜色的属性，black 是颜色的值，该代码的效果是使页面中的文字为黑色。

如果属性的值是多个单词组成，必须在值上加引号，例如字体的名称经常是几个单词的组合，示例代码如下：

```
p {font-family: "sans serif"}
```

以上代码定义段落的字体为 sans serif。

如果需要对一个选择符指定多个属性时，则要使用分号将所有的属性和值分开，例如：

```
p {text-align: center; color: red}
```

以上代码表示设置了两个段落 p 的属性，居中排列和段落中的文字为红色，两个属性间用了分号隔开。

当然为了使定义的样式表方便阅读，也可以采用分行的书写格式，例如以下代码：

```
1. p
2. {
3. text-align: center;
4. color: black;
5. font-family: arial
6. }
```

代码说明：第 1 行定义选择符段落标记 p，第 3 行代码表示段落排列居中，第 4 行表示段落中文字为黑色，第 5 行表示字体是 arial。

还可以把具有相同属性和值的选择符组合起来书写，用逗号将选择符分开，这样可以减少样式重复定义，例如：

```
h1, h2, h3, h4, h5, h6 { color: green }
```

以上代码的含义是定义 6 种标题元素的文字都为绿色，它和分开定义每个标题元素的字体为绿色的效果一样，显然这样写节省代码。

其实选择符并不一定是 HTML 标记，选择符的种类有其它几种，现说明如下：

#### (1) 类选择符

用类选择符能够把相同的元素分类定义不同的样式，定义类选择符时，在类的名称前面加一个点号，点号前加上相应的 HTML 标记，而 HTML 标记可以省略。例如以下代码：

```
1. p.right {text-align: right}
2. center {text-align: center}
```

代码说明：第 1 行定义 HTML 标记的类选择符，这种情况将只定义标记 p 的样式，而第 2 行代码则具有通用性，可用于任何标记的样式定义。

类的名称可以是任意英文单词或以英文开头与数字的组合，但类的命名最好能够说明类功能。

## (2) ID 选择符

在 HTML 文档中, ID 参数指定了某个单一元素, ID 选择符是用来对这个单一元素定义单独的样式。

定义 ID 选择符要在 ID 名称前加上一个“#”号。和类选择符相同, 定义 ID 选择符的属性也有两种方法: 第 1 种是“# + ID”, 第 2 种是“标记 + # + ID”, 第 1 种方法具有通用性, 可用于所有 ID, 第 2 种只用于指定的标记。例如以下代码:

```
1. #intro
2. {
3. font-size:80%;
4. }
5. p#headLine
6. {
7. font-size:100%;
8. }
```

代码说明: 第 1~4 行采用第 1 种方法定义 ID 选择符号, 将可用于所有字体大小的定义, 第 5~8 行采用第 2 种方法定义 ID 选择符号, 将只用于匹配 id="intro" 的段落元素。

ID 选择符局限性很大, 只能单独定义某个元素的样式, 一般只在特殊情况下使用。

## (3) 包含选择符

包含选择符是可以单独对某种元素包含关系 (元素 1 里包含元素 2) 定义的样式表, 这种方式只对元素 1 里的元素 2 定义, 对单独的元素 1 或元素 2 无定义, 例如代码:

```
1. table a
2. {
3. font-size: 12px
4. }
```

代码说明: 以上代码的含义是为在表格内的链接定义了样式, 文字大小为 12 像素, 而表格外链接的文字则不接受该样式的定义。

在样式的语法中, 还有一个概念比较重要就是伪类, 很多特效可以利用伪类来实现。伪类可以看做是一种特殊的类选择符, 是能被支持 CSS 的浏览器自动识别的特殊选择符。它的最大用处就是, 可以针对链接在不同状态下定义不同的样式效果。

伪类的定义格式如下:

选择符:伪类 {属性: 值}, 即: selector:pseudo-class {property: value}

伪类和类不同, 是 CSS 已经定义好的, 不能类选择符一样随意用别的名字, 根据上面的语法可以解释为对象 (选择符) 在某个特殊状态下 (伪类) 的样式。

最常用的是 4 种 a (链接) 元素的伪类, 它表示动态链接在 4 种不同的状态: link (未访问的链接)、visited (已访问的链接)、active (激活链接)、hover (鼠标停留在链接上)。可以把它们分别定义不同的效果, 以区别以上四种状态, 例如以下代码:

```
1. a:link {color: #FF0000; text-decoration: none}
2. a:visited {color: #00FF00; text-decoration: none}
3. a:hover {color: #FF00FF; text-decoration: underline}
4. a:active {color: #0000FF; text-decoration: underline}
```

代码说明：第 1 行定义未访问的链接，第 2 行定义已访问的链接，第 3 行定义鼠标在链接上，第 4 行定义激活链接。



提示

有时链接访问前鼠标指向链接时有效果，而链接访问后鼠标再次指向链接时无效果了。这是因为把 `a:hover` 放在了 `a:visited` 的前面，由于后面的优先级高，当访问链接后就忽略了 `a:hover` 的效果。所以根据叠层顺序，在定义这些链接样式时，一定要按照 `a:link`、`a:visited`、`a:hover`、`a:active` 的顺序书写。

## 12.2.2 使用样式

有四种常用的在页面中插入样式表的方法：链入外部样式表、内部样式表、导入外表样式表和内嵌样式。

### (1) 链入外部样式表：

链入外部样式表是把样式表保存为一个样式表文件，然后在页面中 `<link>` 标记链接到这个样式表文件，这个 `<link>` 标记必须放到页面的 `<head>` 区内，例如以下代码：

```
1. <head>
2. ...
3. <link rel="stylesheet" type="text/css" href="mystyle.css">
4. ...
5. </head>
```

代码说明：第 3 行代码表示浏览器从 `mystyle.css` 文件中以文档格式读出定义的样式表，`rel="stylesheet"` 是指在页面中使用这个外部的样式表，`type="text/css"` 是指文件的类型是样式表文本，`href="mystyle.css"` 是文件所在的位置。



提示

一个外部样式表文件可以应用于多个页面。当改变这个样式表文件时，所有页面的样式都随之而改变。在制作大量相同样式页面的网站时，非常有用，不仅减少了重复的工作量，而且有利于以后的修改、编辑，浏览时也减少了重复下载代码。

### (2) 内部样式表

内部样式表是把样式表放到页面 `<head>` 区里，这些定义的样式就应用到页面中了，样式表是用 `<style>` 标记插入的，例如代码：

```
1. <head>
2. ...
3. <style type="text/css">
4. hr {color: sienna}
5. p {margin-left: 20px}
6. body {background-image: url("images/back40.gif")}
7. </style>
8. ...
9. </head>
```

代码说明：第 3~7 行代码定义了几个标记的样式，其中，定义了分割线 `hr` 的颜色，段落的左

边距和页面的背景图片。这些样式会自动应用到在<body>和</body>之间定义的标记上。

### (3) 导入外部样式表

导入外部样式表是指在内部样式表的<style>里导入一个外部样式表，导入时用@import，例如以下代码：

```
1. <head>
2. ...
3. <style type="text/css">
4. <!--
5. @import "mystyle.css"
6. /*其他样式表的声明*/
7. -->
8. </style>
9. ...
10. </head>
```

代码说明：第5行代码中@import "mystyle.css" 表示导入 mystyle.css 样式表。

### (4) 内嵌样式

内嵌样式是混合在 HTML 标记里使用的，用这种方法，可以很简单地对某个元素单独定义样式。内嵌样式的使用是直接将在 HTML 标记里加入 style 参数。而 style 参数的内容就是 CSS 的属性和值，例如代码：

```
<p style="color: sienna; margin-left: 20px">
 这是一个段落
</p>
```

代码说明：第1行代码中利用 style 参数来定义段落显示的样式。第2行是显示的文字。

以上几节概要介绍了 CSS 的基本语法和用法，关于 CSS 可以定义的详细属性以及相应值，读者可以去查阅资料以获取比较详尽的属性列表，这里就不再一一列举了。

通过样式的语法介绍可以发现，虽然样式的语法不是很复杂，然而由于没有特殊的编辑器，想要创建正确的样式还是一件比较麻烦的事情，首先必须了解语法，此外还需要记住很多属性的含义，而 ASP.NET 4.0 让创建样式变得轻松起来。

**【例 12-4】**使用 CSS 外部样式表定义一个网站注册页面的布局设计和外观的显示，最后在程序中导入外部样式表。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 12-4”。

**02** 用鼠标双击网站的根目录下“Default.aspx”文件，进入到“视图设计器”，切换到“源视图”，在<form>和</form>标记之间编写如下代码。

```
1. <fieldset>
2. <legend>注册信息</legend>
3. <p>
4. <label for="name">用户名:</label>

5. <input type="text" id="name" name="name" tabindex="1" class="txt"/>
6. </p>
7. <p>
8. <label for="email">登录邮箱:</label>

9. <input type="text" id="email" name="email" tabindex="2" class="txt"/>
```

```

10. </p>
11. <p>
12. <label for="email">登录密码:</label>

13. <input type="password" id="email" name="email" tabindex="2" class="txt"/>
14. </p>
15. <p>
16. <input type="checkbox" id="remeber" name="remeber" tabindex="3" style="background-color:#FF0000;"/>
17. <label for="remeber"> 接受用户协议 </label>
18. </p>
19. <p>
20. <input type="submit" value="提交注册" tabindex="4" class="but"/><input name="" type="reset" value="全部重置"
class="but" />
21. </p>
22. </fieldset>

```

代码说明：第 1 行使用 `fieldset` 标签定义一组表单元素的范围。第 2 行定义一个名为“注册信息”的表单标题。第 4 行定义一个 HTML 的文本标签显示用户名的字样。第 5 行定义一个 HTML 的文本框用于用户名的输入并设置样式的类名。第 8 行定义一个 HTML 的文本标签显示登录邮箱的字样。第 9 行定义一个 HTML 的文本框用于邮箱的输入并设置样式的类名。第 12 行定义一个 HTML 的文本标签显示登录密码的字样。第 13 行定义一个 HTML 的文本框用于登录密码的输入并设置样式的类名。第 16 行定义一个 HTML 多选框用于是否选择接受用户协议。第 17 行定义一个 HTML 的文本标签显示接受用户协议的字样。第 20 行定义了一个 HTML 的提交按钮和一个 HTML 的重载按钮。

**03** 右键单击项目名称“例 12-4”，在弹出的快捷菜单选择“添加新项”命令。

**04** 在弹出如图 12-10 的“添加新项”对话框中选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“样式表”，然后在“名称”文本框输入该文件的名称“StyleSheet.css”，最后单击“添加”按钮。



图 12-10 “添加新项”对话框

**05** 此时在网站根目录自动生成一个如图 12-11 所示的“StyleSheet.css”文件。

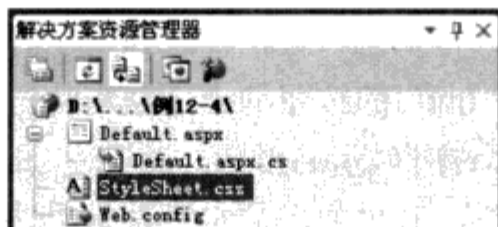


图 12-11 生成样式表

**06** 用鼠标双击“StyleSheet.css”文件，编写的代码如下。

```

1. *{ padding:0px;
2. margin:0px;
3. }
4. body{ font-family:"宋体";
5. font-size:12px;
6. }
7. .big { width:300px;
8. margin:0 auto 0 auto;
9. }
10. .big p { margin:5px 0 0 5px;
11. }
12. #thisform { font-size:12px;
13. color:#999;
14. }
15. #thisform label { font-weight:bold;
16. color:#660000;
17. }
18. #thisform fieldset { border:1px solid #cc;
19. padding:0 20px;
20. }
21. #thisform legend { font-weight:bold;
22. font-size:90%;
23. color:#666;
24. background:#eee;
25. border:1px solid #ccc;
26. border-bottom-color:#999;
27. border-right-color:#999;
28. padding:4px 8px;
29. }
30. .txt{ width:200px;
31. background-color:#CCCCFF;
32. border:#6666FF 1px solid;
33. color:#0066FF;
34. }
35. .but{ border:0px#93bee2solid;
36. border-bottom:#93bee21pxsolid;
37. border-left:#93bee21pxsolid;
38. border-right:#93bee21pxsolid;
39. border-top:#93bee21pxsolid;*/
40. background-color:#3399CC;
41. cursor:hand;
42. font-style:normal;
43. color:#FFFFFF;
44. }

```

代码说明：第 1~3 行定义整个表单的内外边距。第 4~6 行定义 body 标记之间元素的字体和大小。第 7~9 行定义 form 标记间元素的宽度和外边距。第 10 行定义 form 标记中段落间的外边距。第 12~14 行定义 form 标记间元素边框样式和内边距。第 15~17 行定义 form 标记中文本标签的字重和颜色。第 18~20 行定义 form 标记中 fieldset 元素的边框样式以及内边距。第 21~29 行定义 form 标记中标题元素的字重、字体大小、颜色、背景色、边框样式以及内边距。第 30~34 行定义选择

符类名是“txt”的文本标签的宽度、背景色、边框样式和文字颜色。第 35~44 行定义选择符类名是“but”的按钮的上下左右边框的样式、背景色、文字字体和鼠标经过时显示的形状。第 12、15、18、21 行的样式使用的是 ID 选择符，其余使用的定义都是类选择符。

07 将解决方案资源管理器中的“StyleSheet.css”文件，拖动到“源视图”中的<head>和</head>标记内。

08 按下“Ctrl+F5”，运行结果如图 12-12 所示。



图 12-12 运行结果



导入外部样式表的路径的方法，与链入样式表的方法很相似，但导入外部样式表输入方式更有优势。实质上它相当于存在内部样式表中的。导入外部样式表必须在样式表的开始部分，在其他内部样式表的前面。

### 12.2.3 样式创建器

使用 Visual Studio 2010 提供的“样式创建器”能够很方便地创建样式。只需要根据它提供的“新建样式”对话框进行一些选择和设置就可生成满足需要的样式。选择菜单栏中的“格式”|“新建样式”命令，打开如图 12-13 所示的“新建样式”对话框，在这个对话框里只需要进行一些选择和设置就可以创建各种 CSS 样式。



图 12-13 “新建样式”对话框

在上图的“新建样式”对话框里，“选择器”下拉列表框中提供了 100 多种选择器类型供用户使用。“定义位置”下拉列表指定当前样式是用于当前网页，还是新建的样式表，或者使用现有的样式表。类别列表中提供了可以定义样式的内容，当选择某项样式类别，在对话框的中间就会出现该类别可以设置的各种属性。表 12-1 中列出了可定义样式的说明。

表 12-1 “新建样式”对话框中可以定义的样式

样式分类	说明
字体	设置字体类型、字体大小和文本颜色，还可以设置诸如加黑等字体特性
块	对文本进行设置，可以设置排列方式、字间距、在第一行的缩进量等
背景	设置背景色或背景图片
边框	设置元素的边框，可以设置边框的样式、厚度和颜色等
方框	设置元素的边界和容器之间的区域和元素的边界和里面内容之间的区域的样式
定位	为元素设置一个固定的宽度和高度，还可以为元素设置元素在页面上位置
布局	控制各种复杂的布局，可以指定某个元素是显示还是隐藏，可以设置某个元素在页面的边界时是否浮动，还可以设置鼠标在滑过某些内容时显示的样式，等等
列表	设置列表样式
表格	设置表格样式

为了能够把将要创建的样式应用于选中的内容，还必须选中“将新样式应用于文档选择内容”复选框。当选择一些样式后，在“预览”文本框中可以看到选择的样式的预览效果。

【例 12-5】通过本例讲解如何使用样式创建器实现一个组成搜索界面各种元素的样式和外观。

- 01
- 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 12-5”。
- 02
- 用鼠标双击创建网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 HTML 的文本框、1 个 HTML 的下拉列表框和 2 个 HTML 的按钮到“设计视图”中。
- 03
- 用鼠标单击 HTML 的文本框。选择菜单栏中的“格式”|“新建样式”命令，打开如图 12-14 所示的“新建样式”对话框。在“选择器”下拉列表中选择“.text”，在“定义位置”下拉列表中选择“当前网页”，选中“将新样式应用于文档选择内容”复选框；在“类别”列表中选择“字体”和“背景”，分别设置相应的属性，最后单击“确定”按钮。

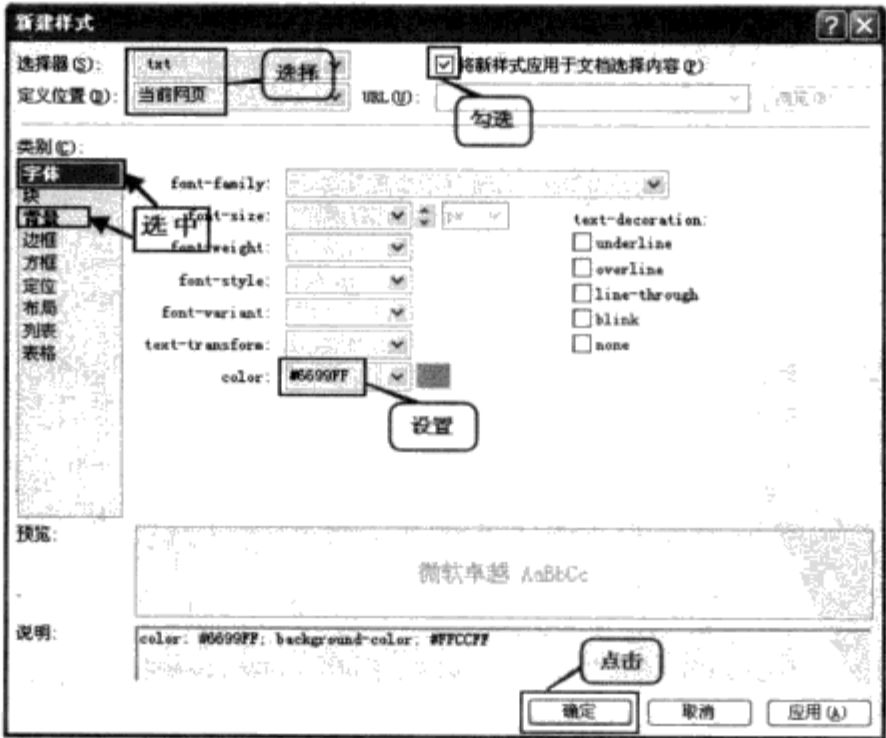


图 12-14 “新建样式”对话框

- 04
- 此时，在“源视图”的<head>下的<style>标记中自动生成的样式代码如下。

```

1. .txt{
2. background-color:#FFCCFF;
3. color:#6699FF;
4. }

```

代码说明：第 1 行为标签控件创建一个内部的样式表，使用类选择符，类名为“txt”。第 2 行设置文本框的背景颜色。第 3 行设置文字的颜色。

**05** 分别用使用以上的步骤，对所有的 HTML 标记和表单元素在“新建样式”对话框中进行所见即所得的可视化的设置。完成设置后，自动生成的样式代码如下所示。

```

1. * { padding:0px;
2. margin:0px;
3. }
4. body { font-family:"宋体";
5. font-size:12px;
6. }
7. .big { width:300px;
8. margin:0 auto 0 auto;
9. border:#999999 1px solid;
10. }
11. h1 { display:block;
12. height:25px;
13. line-height:25px;
14. font-size:14px;
15. background-color:#9999CC;
16. }
17. .big p { margin:5px 0 0 5px;}
18. .sle{ background-color:#00CCCC;
19. font-style:oblique;
20. }
21. .one{ background-color:#99FF99;
22. }
23. .but{ color:#FFF;
24. background-color:#FF3366;
25. border:1px#93bee2solid;
26. border-bottom:#93bee21pxsolid;
27. border-left:#93bee21pxsolid;
28. border-right:#93bee21pxsolid;
29. border-top:#93bee21pxsolid;
30. cursor:hand;
31. font-style:normal;
32. }

```

代码说明：第 1~3 行用定义整个表单的内外边距。第 4~6 行定义 body 标记之间元素的字体和大小。第 7~10 行定义选择符类名是“big”的 form 标记间元素的宽度、外边距和边框样式。第 11 行到 16 行定义标题标记的高度、线高、字体大小、背景色和显示的方式。第 17 行定义 form 标记中段落的外边距。第 18~20 行定义选择符类名是“sel”的下拉列表框的字体和背景色。第 21 行定义选择符类名是“one”的下拉列表框中列表项的背景色。第 23~32 行定义选择符类名是“but”的按钮上下左右边框的样式、背景色、文字字体和鼠标经过时显示的形状。

**06** 切换到“源视图”，可以看到在定义 HTML 标记元素的代码中自动生成了关联样式表的属性。

**07** 按下“Ctrl+F5”，运行结果如图 12-15 所示。



图 12-15 运行结果

12.2.4 CSS 属性窗口

在 Visual Studio 2010 中提供了 CSS 属性窗口，可以通过该窗口方便地查看或修改任何页面中的样式。只要选择菜单栏中的“视图”|“CSS 属性”命令，就能可以打开如图 12-16 所示的 CSS 属性窗口。

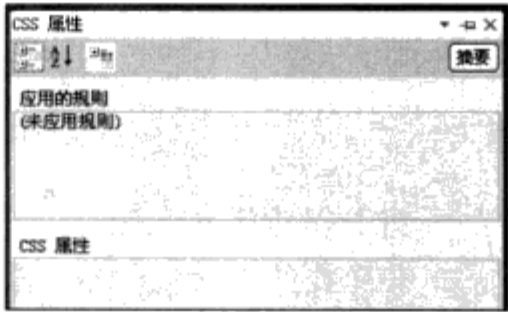


图 12-16 CSS 属性窗口

当打开 CSS 属性窗口后，就可以使用它查看已有的样式。在页面中选择应用样式的标记或控件，就可以在 CSS 属性窗口看到该样式的详细内容。比如查看“例 12-5”创建的样式，可以在“设计视图”中选择相关控件，这样在如图 12-17 所示的 CSS 属性窗口中，可以看到应用该标记样式的详细内容。CSS 属性窗口的上半部分显示了应用到当前选择的标记或控件的样式列表，而下半部分详细地显示了被选择的样式的详细内容，而且在属性窗口中，可以随时更改样式的详细定义。

在 CSS 属性窗口中，右键单击样式列表或者详细属性，则会弹出如图 12-18 所示的一个菜单。

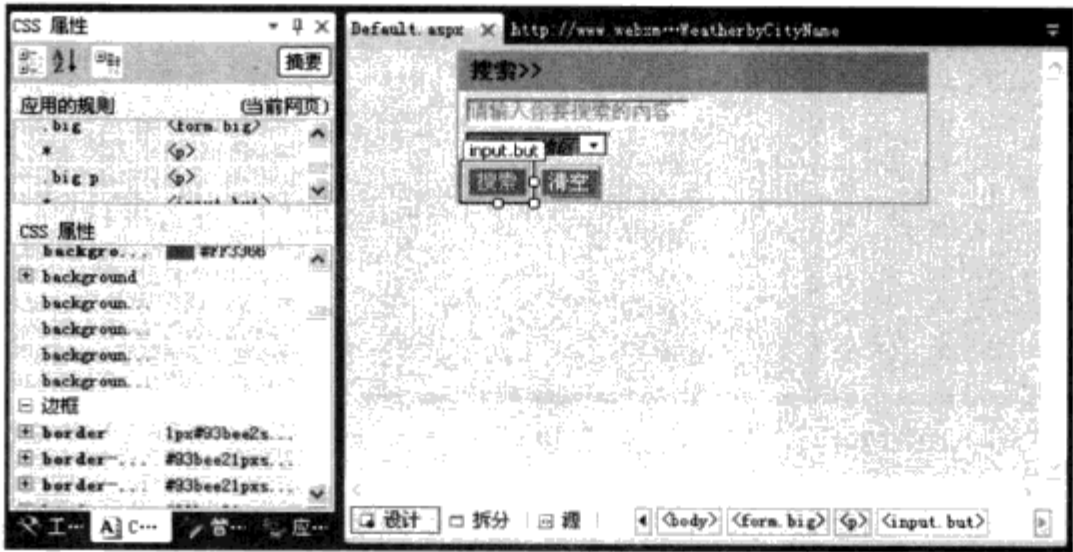


图 12-17 查看存在的样式



图 12-18 CSS 属性右键菜单

在上图 12-17 中可以看到，该菜单包含了以下的一系列命令，我们可以根据实际需要选择相应的菜单项进行操作。菜单中的命令选项的含义说明如下。

- 转到代码，用来显示样式定义的代码，选择该命令时，就会在右边的页面窗口中显示出定义的样式代码。
- 新建样式，用来新建创建一个样式，选择该命令时，则会弹出前面介绍的样式创建对话框。
- 新建样式副本，用来创建当前样式的一个副本，选择该命令时，同样会弹出样式创建对话框，在该对话框里显示当前样式的详细信息，而样式名则会在原来的名字后面加上 Copy

表示新创建的样式是当前样式的副本。

- 修改样式，用来修改当前样式，选择该命令时，则弹出修改样式对话框，这个对话框和新建样式对话框一样，只不过这里将显示当前样式的详细的信息。
- 新建级联样式，用来创建级联样式，选择该命令时，则会在属性窗口添加一个新的样式，选中该样式就可以在属性窗口中对该样式进行详细定义。
- 删除类，用来删除当前的样式。

### 12.2.5 创建和应用样式文件

在很多情况下，通常并不直接在页面中创建样式，而是创建一个或几个存储通用样式的文件，然而在页面引用这些文件定义的样式。这样有助于样式的统一管理。具体操作步骤如下：

**01** 创建样式文件包括两种情况：第一种是创建一个新样式文件，把创建的样式放到该文件中；第二种是向已经存在的样式文件中添加新的样式。这两种创建的方式都可以在“新建样式”对话框中进行。把对话框中的“定义位置”下拉列表选择为“新建样式表”或“现有样式表”即可。然后进行样式的定义，完成后单击“确定”按钮，即可创建一个新的样式文件或添加样式到已有的样式文件中。

**02** 打开新创建的样式文件，可以看到新创建的样式被存储在该文件中。按“Ctrl+Alt+T”快捷键，会出现如图 12-19 所示“CSS 大纲”窗口，左侧的窗口就是“CSS 大纲”窗口，该窗口显示了当前样式文件中包含的元素、类、元素 ID 以及@Blocks 等的概要内容。

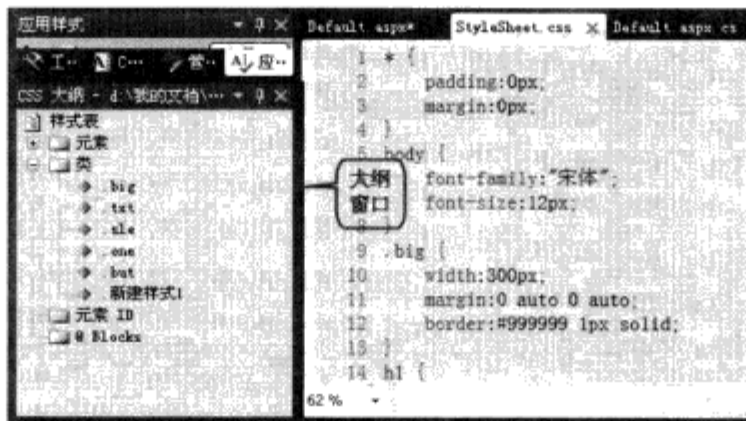


图 12-19 “CSS 大纲”窗口

**03** 在前文中已经讲过可以利用<link>标记将样式文件(.css 文件)引用的到页面文件(.aspx)。而在 Visual Studio 2010 中，可以直接通过把样式文件拖动到要应用该样式的页面，这样，<link>语句就会自动出现在该页面文件中。

**04** 当把样式文件的引用添加到页面上后，就能够把样式应用到页面上的控件和标记上：可直接利用代码添加样式的引用，也可以通过选择菜单栏上的“视图”|“应用样式”命令，打开如图 12-20 所示的“应用样式”窗口，可以看到外部样式表中的各种样式已经根据类选择器的名称和 HTML 的标记关联了起来并显示出了外观效果。

**05** 运行程序后搜索页面组成元素的外观与图 12-15 相比发生了明显变化，如图 12-21 所示。



图 12-20 “应用样式”窗口

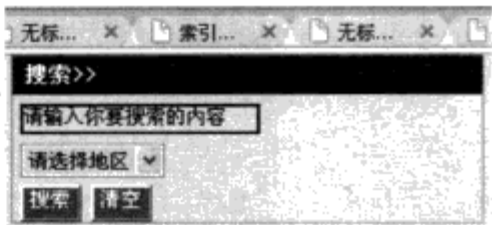


图 12-21 应用样式

由此可见，在 Visual Studio 2010 中将“样式创建器”、“CSS 属性”窗口和“应用样式”窗口这三者配合使用，可以非常方便地进行 Web 页面的 CSS 样式的设计。

### 12.3 上机题

1. 设计网页时，为了使网站风格统一和整体美观，通常会对页面中的每一个元素应用 CSS 样式。本题要求使用 CSS 样式对窗体表单的元素进行样式的设计，运行结果如图 12-22 所示。
2. 使用 CSS，将导航栏设置成有动态效果，这种实现方式在很多网站中是非常多见的，本题要求使用两张不同的背景图片，在鼠标停留时，背景图片进行切换，同时文字的颜色和样式发生变化，运行结果如图 12-23 所示。

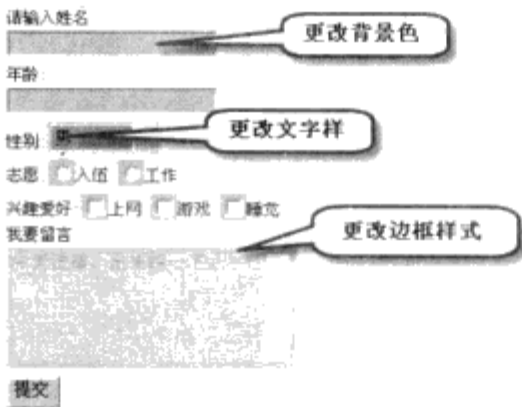


图 12-22 运行结果



图 12-23 运行结果

3. 为了使网站中的超级链接更加吸引人的眼球，可以制作动态的超级链接。当用户鼠标的指针经过超级链接按钮时，会出现“字体变蓝色且文字带下划线”的效果，运行结果如图 12-24 所示。

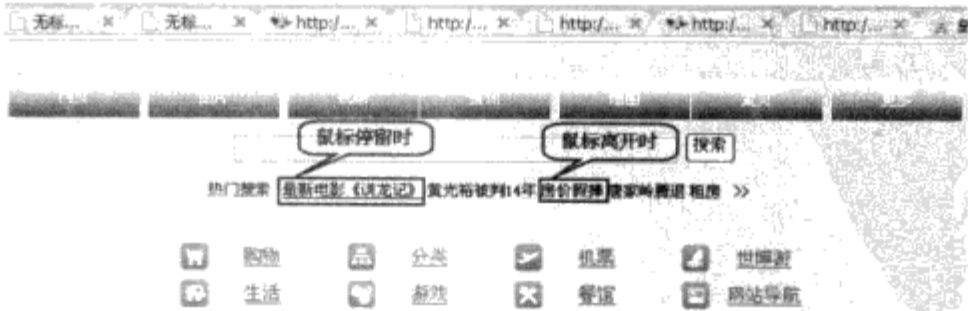


图 12-24 运行结果

4. 使用内部样式表为浏览器窗口的滚动条添加如图 12-25 所示的颜色和格式。

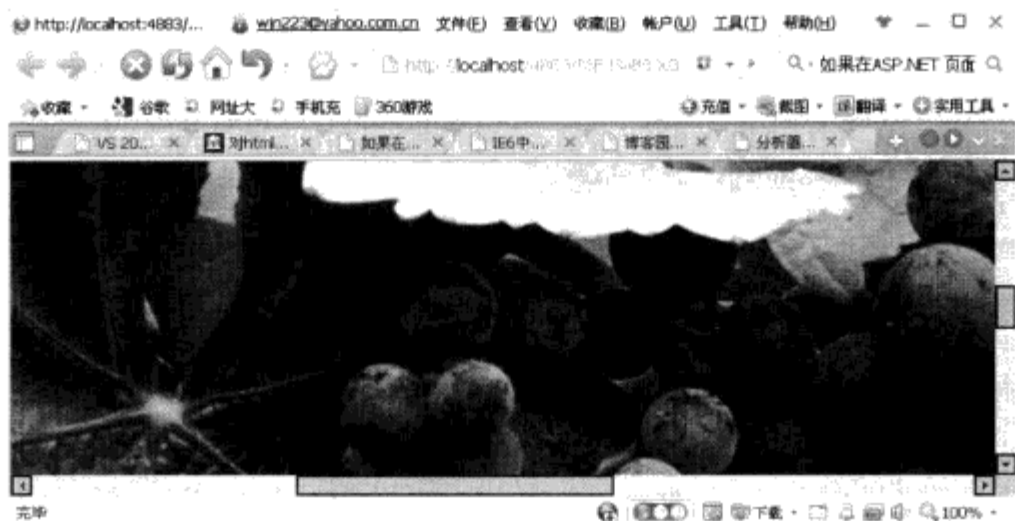


图 12-25 运行结果

- 通过主题外观文件创建控件的主题。要求为 3 个 TextBox 控件设置不同的外观，其中一个不设置外观保持默认的外观，其余两个设置命名外观，运行结果如图 12-26 所示。
- 在上题的基础上，再创建一个控制 3 个 TextBox 控件外观的主题，要求以编程的方式通过页面的 Page\_ProInit 事件实现根据用户的选择对主题进行动态的加载，运行结果如图 12-27 所示。



图 12-26 运行结果



图 12-27 运行结果



图 12-28 运行结果

- 使用主题和主题中的皮肤文件定义页面上的菜单控件 Menu 的外观样式，运行程序后，菜单显示结果如图 12-28 所示。

# 第 13 章 LINQ 查询

## 学习目标

LINQ 是微软公司提供的一种统一数据查询模式，并与 .NET 开发语言进行了高度的集成，很大程度上简化了数据查询的编码和调试工作，提高了数据处理的性能。借助于 LINQ 中的丰富组件配合专用于 LINQ 查询的数据源控件 LinqDataSoure 以及查询扩展控件 QueryExtender，编程人员可以在代码编写量很少的情况下，方便地实现对数据库的各类查询操作。希望读者以本章为基础能够慢慢地进入 LINQ 技术的大门。

## 本章重点

- LINQ 查询的基本语法
- 使用 LINQ to SQL 访问数据库
- LinqDataSoure 与数据绑定控件的配合使用
- QueryExtender 控件的筛选表达式

## 13.1 LINQ 简述

LINQ 是 Language Integrated Query 的缩写，中文名字是“语言集成查询”，最初在 Visual Studio 2008 中发布。LINQ 引入了标准的、易于学习的查询模式和更新模式，可以对其进行扩展以便支持几乎任何类型的数据存储。它提供给编程人员一个统一的编程概念和语法，开发人员不需要关心将要访问的是关系数据库还是 XML 数据，或是远程对象，它都采用同样的访问方式。Visual Studio 2010 包含 LINQ 提供程序的程序集，这些程序集支持 LINQ 与 .NET Framework、SQL Server 数据库、ADO.NET 数据集以及 XML 文档一起使用。

LINQ 包含一系列的查询技术。其中 LINQ 到对象是对内存进行操作，LINQ 到 SQL 是对数据库的操作，LINQ 到 XML 是对 XML 数据进行操作，LINQ 到实体是对实体对象模型数据进行操作。由于篇幅有限，本章主要介绍的是 LINQ 到 SQL 的查询操作技术。

图 13-1 描述了 LINQ 技术的体系结构。

由于 LINQ 的出现，开发人员可以使用关键字和运算符实现针对强类型化对象集的查询操作。在编写查询过程时，可以获得编译时的语法检查、元数据、智能感知和静态类型等强类型语言所带来的优势。并且它还可以方便地查询内存中的信息而不仅仅只是外部数据。

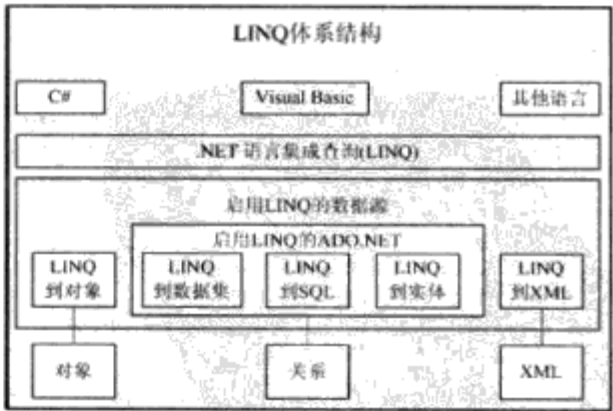


图 13-1 LINQ 体系结构

在 Visual Studio 2010 中, 可以使用 C# 语言为各种数据源编写 LINQ 查询: 包括 SQL Server 数据库、XML 文档、ADO.NET 数据集以及支持 IEnumerable 接口 (包括泛型) 的任意对象集合。除了这几种常见的数据源之外, .NET 4.0 还为用户扩展 LINQ 提供支持, 用户可以根据需要实现第三方的 LINQ 支持程序, 然后通过 LINQ 获取自定义的数据源。

LINQ 查询既可在新项目中使用, 也可在现有项目中与非 LINQ 查询一起使用。唯一的要求是项目必须与 .NET Framework 4.0 版本相兼容。



LINQ 是在 .NET 3.5 版之后新增的技术, 所以在 .NET 2.0 和早期的版本程序中是不能直接使用 LINQ 查询的, 要在 .NET 2.0 和早期的版本程序中使用 LINQ, 首先需要通过 Visual Studio 2010 将程序自动转化为 .NET 4.0 版本。

## 13.2 LINQ 基础知识

LINQ 作为一种数据查询编码方式, 本身并不是独立的开发语言, 也不能进行应用程序的开发。但是在 ASP.NET 4.0 中, 通过 C# 语言集成 LINQ 查询代码, 可以在任何源代码文件中使用。但要注意的是, 使用 LINQ 查询功能必须引用 System.Linq 命名空间。

### 13.2.1 LINQ 查询步骤

查询是一种从数据源检索数据的表达式, 通常使用专门的查询语言来表示。随着编程技术的不断发展, 人们已经为各种数据源开发了不同的语言, 编程人员不得不对每种数据源或数据格式进行有针对性的学习。而 LINQ 的出现则改变了这种情况, 它可以使用通用的基本编码模式来查询和转换不同的数据源, 如 XML 文档、SQL 数据库、ADO.NET 数据集和 .NET 集合中的数据等。这样就让程序员从不断的查询语言学习中摆脱了出来, 可以专注于业务功能的开发。

使用 LINQ 的查询通常由以下三个不同的操作步骤组成:

- 获得数据源。
- 创建查询。
- 执行查询。

**【例 13-1】** 通过使用标准的 LINQ 查询语句获得字符串数组中每个元素的大小写格式来演示查询操作的三个步骤。

- 01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 13-1”。
- 02** 用鼠标双击网站根目录下的“Default.aspx.cs”文件, 编写代码如下。

```
1. protected void Page_Load(object sender, EventArgs e){
2. string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry", "HAryY", "IOvE"};
3. int i = 0;
4. var upperLowerWords =
5. from w in words
6. select new { Upper = w.ToUpper(), Lower = w.ToLower() };
7. foreach (var ul in upperLowerWords){
```

```

8. Response.Write(words[i]+ "单词的大写为:" + ul.Upper + "
");
9. Response.Write(words[i]+ "单词的小写为:" + ul.Lower + "
");
10. i++;
11. }
12. }

```

代码说明：第 1 行处理页面 Page 加载的事件 Load。第 2 行定义了 string 类型的数组 words。这个 words 就是 LINQ 的查询操作中的第一步获得数据源。

第 4、5 行定义隐藏变量 upperLowerWords 获得通过关键字 from 和 select 创建 LINQ 查询语句查询 words 数组中的每个 string 元素。第 6 行执行二次查询，获得 words 数值中元素的大写格式和小写格式。以上是 LINQ 查询操作中的第二步创建查询。

第 7~11 行通过 foreach 循环语句将查询的结果输出到页面显示，这是 LINQ 查询操作中的第三步执行查询。

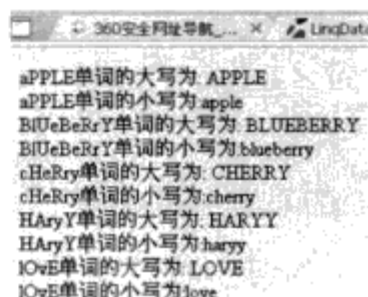


图 13-2 运行结果

**03** 按下“Ctrl+F5”，程序运行结果如图 13-2 所示。



LINQ 最具突破性的优势在于将文本和对象操作完美集成，它让查询数据和操作对象一样安全和轻松。查询（Query）是 LINQ 的核心概念之一。

提示

### 13.2.2 LINQ 和泛型

LINQ 查询是建立在泛型这种数据类型的基础上，所以从 .NET Framework 2.0 开始就引入了泛型数据。虽然编程人员无须深入了解泛型技术就可以开始编写 LINQ 查询，但以下两个泛型的基本概念还是有必要了解的。

(1) 当创建泛型集合类（如 List<Of<(T)>>）的实例时，将“T”替换为集合中指定的对象类型。例如，字符串集合表示为 List<string>，Student 对象集合表示为 List<Student>。因为泛型集合是强类型的，所以比将元素存储为 Object 类型的集合要强大的多。如果尝试将 Student 添加到 List<string>，则会在编译时出现一条错误。泛型集合易于使用的原因是不必执行运行时进行强制类型转换。

(2) IEnumerable<Of<(T)>> 表示是一个接口，通过该接口，可以使用 foreach 语句来遍历泛型集合类。LINQ 查询变量可以类型化为 IEnumerable<Of<(T)>> 或者它的派生类型如 IQueryable<Of<(T)>>。当看到类型化为 IEnumerable<Student> 的查询变量时，这意味着在执行该查询时，该查询将生成包含零个或多个 Student 对象的集合。例如代码：

```

1. IEnumerable<Student> Student Query =
2. from s in Students
3. where s.Name == "张琴"
4. select s;
5. foreach (Student s in Student Query){
6. Console.WriteLine(s.Name + ", " + s.age);
7. }

```

代码说明：第 1 行定义一个查询变量 Student Query，该变量的类型为 IEnumerable< Student >，

第 2~4 行定义了具体的 LINQ 查询指令，从学生中查询叫张琴的学生对象。第 5~8 行通过 foreach 循环遍历显示查询该对象的姓名和年龄信息。

为了避免使用泛型语法，我们可以使用匿名类型来声明查询，即使用 var 关键字来声明查询。var 关键字指示编译器通过查看在 from 子句中指定的数据来推断查询变量的类型，例如：

```
1. var Student Query =
2. from s in Students
3. where s.Name == "张琴"
4. select s;
5. foreach (var s in Student Query){
6. Console.WriteLine(s.Name + ", " + s.age);
7. }
```

代码说明：除第 1 行外，这段代码和前面的代码相同。这里查询变量的类型 var 和 Student 相同，而 Student 的类型是 IEnumerable<Student>。因此，这段代码和前面的代码具有相同的效果。



提示

如果没有特别的需要，建议使用不指定数据类型的本地变量（包括泛型集合），让编译器自动根据数据源判断具体的元素类型。

### 13.2.3 基本的查询操作

对于编写查询的开发人员来说，LINQ 最明显的“语言集成”部分是查询表达式。查询表达式使用 C# 3.0 中引入的声明性查询语法编写。通过使用查询语法，开发人员可以使用最少的代码对数据源执行复杂的筛选、排序和分组操作。也可以查询和转换 SQL 数据库、ADO.NET 数据集、XML 文档、流以及 .NET 集合中的数据。

查询表达式是由查询关键字和对应的操作数组成的表达式整体，其中，查询关键字是常用的查询运算符。C# 为这些运算符提供对应的关键字，从而能更好地与 LINQ 集成。

查询表达式必须以 from 为关键字的子句开头，并且必须以 select 或 group 关键字的子句结尾。在第一个 from 子句和最后一个 select 或 group 子句之间，查询表达式可以包含一个或多个由下列关键字组成的可选子句：where、orderby、join、select 等关键字。同时还可以使用 into 关键字让 join 或 group 子句的结果能够作为同一查询表达式中附加查询子句的数据源。

#### 1. from 子句

查询表达式必须以 from 子句开头。它同时指定了数据源和范围变量。在对数据源进行遍历的过程中，范围变量表示数据源中的每个元素，并根据数据源中元素类型对范围变量进行强类型化。

#### 2. select 子句

使用 select 子句可以查询所有类型的数据源。简单的 select 子句只能查询与数据源中所包含的元素具有相同类型的对象。例如以下代码：

```
1. IEnumerable< Student> StudentQuery =
2. from age in Students
3. orderby Student.age
4. select age;
```

代码说明：第 1 行定义查询变量 StudentQuery，第 2 行定义数据源，该查询的数据源包含 age 对象。第 3 行的 orderby 子句将根据 age 的大小重新排序，第 4 行的 select 子句查询出已经重新排序后的集合。

### 3. group 子句

使用 group 子句可获得按照指定的键进行分组的元素，键可以采用任何数据类型。例如根据 age 属性进行分组查询的代码如下。

```
1. var StudentQueryByName =
2. from s in students
3. group s by s.age;
4. foreach (var studentGroup in StudentQueryByName){
5. Console.WriteLine(studentGroup.Key);
6. foreach (student s in studentGroup){
7. Console.WriteLine(" {0}", s.age);
8. }
9. }
```

代码说明：第 3 行根据 age 对查询结果进行分组。第 4~9 行通过 foreach 循环遍历查询结果。在使用 group 子句结束查询时，结果保存在嵌套的集合中，即集合中的每个元素又是另一集合，该子集合中包含根据 Key 键划分的每个分组对象。在循环访问生成分组的对象时，必须使用嵌套的 foreach 循环。外部循环用于循环访问每个分组对象，内部循环用于循环访问每个组的成员。

如果必须引用分组操作的结果，可以使用 into 关键字来创建进一步的查询。下面的查询只返回那些包含两个以上的客户的分组。

```
1. var studentQuery =
2. from s in students
3. group s by s.age into studentGroup
4. where student.Count() > 1
5. orderby studentGroup.Key
6. select studentGroup;
```

代码说明：第 3 行使用 into 关键字表示把 group 分组的结果保存在 studentGroup 中，第 4 行设置查询的条件为返回学生数量大于 1 个人的分组。

### 4. where 子句

where 子句是通过条件的设定对查询的结果进行过滤，从数据源中排除指定的元素。在下面的示例中，只返回姓名是“张琴”的学生：

```
var queryStudent =
from s in Students
where s.Name == "张琴"
select s;
```

代码说明：第 3 行设置查询的条件是姓名叫“张琴”，通过 where 子句，排除姓名不叫张琴的学生。

如果要使用多个过滤条件的话，需要使用逻辑运算符，如&&、||等。例如，下面的代码只返回年龄是 20 岁且姓名为“张琴”的学生：

```
where s.Name=="张琴" && s.age == "20"
```

### 5. orderby 子句

使用 orderby 子句可以很方便地对返回的数据进行排序。OrderBy 子句对查询返回的元素根据指定的排序类型进行排序。例如根据 age 属性对查询返回的结果进行排序：

```
1. var queryStudent =
2. from s in Students
3. where s.age == 20
4. orderby s.age ascending
5. select s;
```

代码说明：第 4 行使用 orderby 关键字进行排序。Student 类型的 age 属性是整型值，执行学生的年龄从大到小排序。Ascending 关键字表示以默认方式按递增的顺序进行排列。Descending 关键字则表示把查询出的数据进行逆序排列。

### 6. 联接和投影

在 LINQ 中，join 子句可以将来自不同数据源中没有直接关系的元素进行关联，但是要求两个不同数据源中必须有一个相等元素的值。例如下面对两个数据集 arry1 和 arry2 进行连接查询。

```
1. int[] arry1={7,17,27,33,35,51}
2. int[] arry2={13,23,33,53,63,73,83}
3. var query=from val1 in arry1
4. join val2 in arry2 on val1%6 equals val2%16
5. select new {VAL1= val1,VAL2= val2};
```

代码说明：第 1 行创建整型数组 arry1 作为数据源。第 2 行创建整型数组 arry2 作为数据源。第 3 行表示联接的第一个集合为 arry1。第 4 行表示联接的第二个集合为 arry2，第 5 行表示当 val1%6 和 val2%16 有相同的值时，select 子句将 val1 和 val2 选择为查询结果。

投影操作和 SQL 查询语句中的 SELECT 基本类似，投影操作能够指定数据源并选择相应的数据源，能够将集合中的元素投影到新的集合中去，并能够指定元素的类型和表现形式。示例代码如下。

```
1. int[] array={5,6,7,8,9,10,11,12,13,14}
2. var lint=array.Select(i=>i);
3. foreach(var a in lint){
4. Console.WriteLine(a.ToString())
5. }
```

代码说明：第 1 行创建整型数组 array 作为数据源，第 2 行使用 Select 进行同行投影操作将符合条件的元素投影到新的集合 lint 中去。第 3~5 行循环遍历集合并输出对象。



提示

其实，创建对象模型，就是基于关系数据库来创建这些 LINQ 到 SQL 对象模型中最基本的元素并进行一一的对应映射。

**【例 13-2】**使用 LINQ 基本的查询操作，从学生的集合中查找地址在北京的学生并将查询结果显示在页面上。

**02** 右键单击“网站项目名称”，在弹出的快捷菜单中选择“添加新项”命令。

**03** 打开如图 13-3 所示“添加新项”对话框，选择“已安装模板”下的“Visual C#”模板，模板文件列表中选中“类”，然后在“名称”文本框输入该文件的名称“Student.cs”，最后“添加”按钮。

**04** 此时在网站目录的“App.Code 文件夹”下添加了一个如图 13-4 所示的“Student.cs”文件。

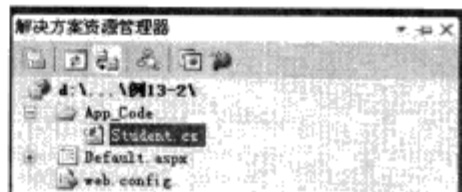


图 13-4 生成类文件

**05** 用鼠标双击“Student.cs”文件，编写代码如下。

```

1. public class Student {
2. public String name{ get; set; }
3. public int age{ get; set; }
4. public String address{ get; set; }
5. }

```

代码说明：第 1 行定义了一个 Student 的学生类。第 2~4 行定义了三个私有的属性代表学生的姓名、年龄和住址。

**06** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. List<Student> stud = new List<Student>{
3. new Student { name ="Mary",age =21,address ="北京"},
4. new Student { name ="John",age =37,address ="北京"},
5. new Student { name ="Make",age =40,address ="北京"},
6. new Student { name ="Rose",age =50,address ="北京"},
7. new Student { name ="Peter",age =38,address ="上海"},
8. new Student { name ="Hanson",age =30,address ="上海"}
9. };
10. IEnumerable<string> studentQuery =
11. from s in stud
12. where s.address == "北京"
13. orderby s.age descending
14. select s.name ;
15. Response.Write("地址在北京的学生有: " + "
");
16. foreach (var a in studentQuery){
17. Response.Write(a + "
");

```

```

18. }
19. }

```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2~9 行初始化一个 List 类型的学生类集合，其中定义了六个学生对象。第 10~14 行定义了一个查询表达式，其中，第 10 行定义了一个查询变量 studentQuery，第 11 行定义了查询表达式的 from 子句，stud 是由 Student 组成的集合，这里被指定为查询的数据源，s 是范围变量，因为 stud 是 Student 对象组成的，所以范围变量 s 也被类型化为 Student，这样就可以在第 12 行使用点运算符来访问该类型的 address 成员。第 13 行设置查询的排序方式是按照年龄大小降序排列。第 14 行查询返回的结果是符合地址在北京的所有学生的姓名。第 16~18 行通过 foreach 循环遍历符合要求的变量，并把符合要求的学生的名字显示到网页上。

**07** 按下“Ctrl+F5”，程序运行结果如图 13-5 所示。

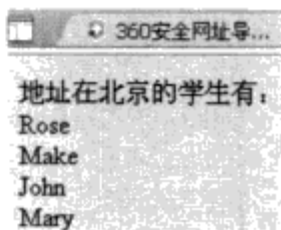


图 13-5 运行结果

## 13.3 LINQ 和数据库操作

LINQ 查询技术主要是用于操作关系型数据库的，其中 LINQ to SQL 是 LINQ 操作数据库中最重要技术。它提供运行时的基础结构，将关系数据库作为对象进行管理。本节将着重介绍有关 LINQ to SQL 的使用方法。

### 13.3.1 LINQ to SQL

LINQ to SQL 是 ADO.NET 和 LINQ 结合的产物，它将关系数据库模型映射到编程语言所表示的对象模型。开发人员通过使用对象模型来实现对数据库数据的操作。在操作过程中，LINQ to SQL 会将对象模型中的语言集成查询转换为 SQL，然后将它们发送到数据库进行执行。当数据库返回结果时，LINQ to SQL 会将它们转换成相应的编程语言处理对象。

使用 LINQ to SQL 可以完成的常用数据库操作包括：

- 选择
- 插入
- 更新
- 删除

以上四大操作包含了数据库应用程序的所有功能，LINQ to SQL 全部能够实现。因此，在掌握了 LINQ 技术后，读者就不需要再针对特殊的数据库学习特别的 SQL 语法了。

LINQ to SQL 的使用主要可以分为两大步骤：

#### 1. 创建对象模型

要实现 LINQ to SQL，首先必须根据现有关系数据库的元数据创建对象模型。对象模型就是按照开发人员所用的编程语言来表示的数据库。有了这个表示数据库的对象模型，才能创建查询语句操作数据库。

## 2. 使用对象模型

在创建了对象模型后，就可以在该模型中请求和操作数据了。使用对象模型的基本步骤说明如下：

- (1) 创建查询以便从数据库中检索信息。
- (2) 重写 Insert、Update 和 Delete 的默认方法。
- (3) 设置适当的选项以便检测和报告可能产生的并发冲突。
- (4) 建立继承层次结构。
- (5) 提供合适的用户界面。
- (6) 调试并测试应用程序。

以上只是使用对象模型的基本步骤，其中很多步骤都是可选的，在实际应用中，有些步骤可能并不会每次都需要使用到。



where 子句中的条件尽量简短易懂，并且还可以通过函数等方式来提供判断条件。当出现多个逻辑并（&&）运算的条件时，可以考虑使用多个并列的 where 子句代替。

### 13.3.2 创建对象模型

对象模型是关系数据库在编程语言中表示的数据模型，对对象模型的操作就是对关系数据库的操作。表 13-1 列举了 LINQ 到 SQL 对象模型中的元素与关系数据库中元素的对应关系。

表 13-1 LINQ to SQL 对象模型中的基本元素

LINQ to SQL 对象模型	关系数据模型
实体类	表
类成员	列
关联	外键关系
方法	存储过程或函数

创建对象模型的方法有三种，说明如下。

- ① 使用对象关系设计器，对象关系设计器提供了从现有数据库创建对象模型的可视化操作，它被集成在 Visual Studio 2010 中，比较适用于小型或中型的数据库。
- ② 使用 SQLMetal 代码生成工具，这个工具适合大型数据库的开发，因此对于普通读者来说，这种方法并不常用了。
- ③ 直接编写创建对象的代码。这种方法在有对象关系设计器的情况下不建议使用。

在实际开发过程中，几乎都使用第 1 种方法来创建对象模型，所以这里只介绍如何使用最常用的对象关系设计器创建对象模型。

对象关系设计器（O/R 设计器）提供了一个可视化设计界面，用于在应用程序中创建映射到数

数据库中的对象模型。同时，它还生成一个强类型 DataContext，用于在实体类与数据库之间发送和接收数据。强类型 DataContext 对应于 DataContext 类，它表示 LINQ to SQL 框架的主入口点，充当 SQL Server 数据库与映射到数据库的 LINQ to SQL 实体类之间管道。

DataContext 类包含用于连接数据库以及操作数据库数据的连接字符串信息和方法，也可以将新方法添加到 DataContext 类。DataContext 类提供了如表 13-2 和表 13-3 所示的属性和方法。

表 13-2 DataContext 类的属性

属性	说明
CommandTimeout	增大查询的超时期限，如果不增大则会在默认超时期限间出现超时
Connection	返回由框架使用的连接
LoadOptions	获取或设置与此 DataContext 关联的 DataLoadOptions
Log	指定要写入 SQL 查询或命令的目标
Mapping	返回映射所基于的 MetaModel
ObjectTrackingEabled	指示框架跟踪此 DataContext 的原始值和对象标识
Transaction	为.NET 框架设置要用于访问数据库的本地事务

表 13-3 DataContext 类的方法

方法	说明
CreateDatabase	在服务器上创建数据库
DatabaseExists	确定是否可以打开关联数据库
DeleteDataBase	删除关联数据库
ExecuteCommand	直接对数据库执行 SQL 命令
ExecuteDynamicDelete	在删除重写方法中调用，以向 LINQ 到 SQL 重新委托生成和执行删除操作的动态 SQL 的任务
ExecuteDynamicInsert	在插入重写方法中调用，以向 LINQ 到 SQL 重新委托生成和执行插入操作的动态 SQL 的任务
ExecuteDynamicUpdate	在更新重写方法中调用，以向 LINQ 到 SQL 重新委托生成和执行更新操作的动态 SQL 的任务
ExecuteQuery	已重载，直接对数据库执行 SQL 查询
GetChangeSet	提供对由 DataContext 跟踪的已修改对象的访问
GetCommand	提供有关由 LINQ 到 SQL 生成的 SQL 命令的信息
GetTable	已重载，返回表对象的集合
Refresh	已重载，使用数据库中数据刷新对象状态
SubmitChanges	已重载，计算要插入、更新或删除的已修改对象的集合，并执行相应命令以实现数据库的更改
Translate	已重载，将现有 IDataReader 转换为对象

**【例 13-3】** 本例演示如何使用对象关系设计器来创建 LINQ to SQL 实体类，并创建了一个 Students 对象。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 13-3”。

**02** 执行菜单栏上的“视图” | “服务器资源管理器”命令，弹出如图 13-6 所示的“服务器资源管理器窗口”。右键单击“数据连接”，在菜单中选择“添加连接”的命令。

**03** 在弹出的“添加连接”对话框中单击“浏览”，选择第 6 章中“上机题 1”创建好的“Shool.mdf”的数据库文件。单击“测试连接”按钮，如果连接成功，会弹出“连接成功的对话框”，然后单击该对话框中的“确定”按钮。最后回到“添加连接”对话框中单击“确定”按钮，如图 13-7 所示。



图 13-6 服务器资源管理器



图 13-7 “添加连接”对话框

**04** 这时在“服务器资源管理器窗口”中的“数据连接”节点下会出现刚才添加好的“Shool.mdf”数据库。展开“Shool.mdf”节点 | “表”节点，可以看到如图 13-8 所示“Students”数据表。

**05** 右键单击网站名称。选择“添加新项”菜单选项，弹出如图 13-9 所示的“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“LINQ to SQL”，然后在“名称”文本框输入该文件的名称“DataClasses.dbml”，最后单击“添加”按钮。



图 13-8 生成连接

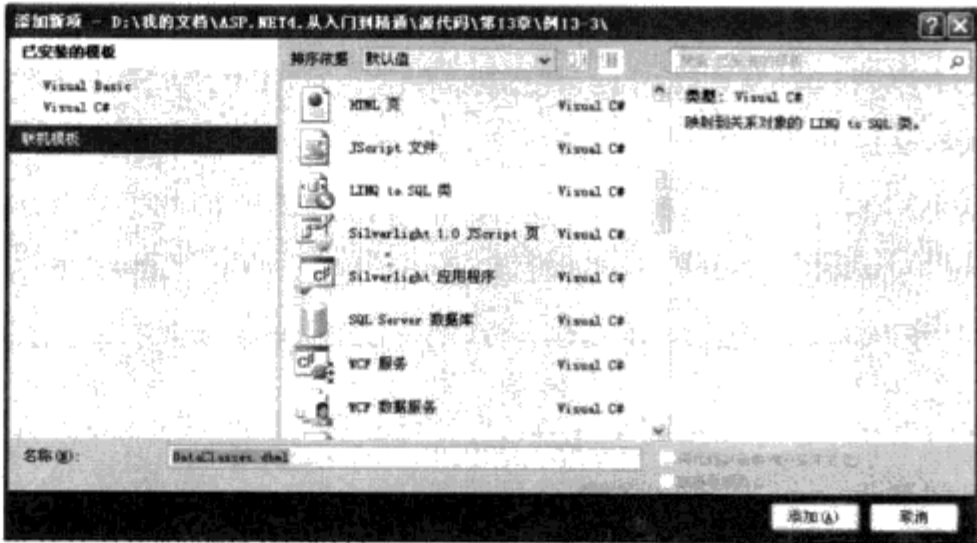


图 13-9 添加新项对话框

**06** 此时在网站根目录下会生成如图 13-10 所示的“App\_Code 文件夹”，在该文件夹中会自动生成一个“DataClasses.dbml”的文件，该文件又包含了一个“DataClasses.dbml.layout”文件和一个“DataClasses.designer.cs”文件。

**07** 双击“DataClasses.dbml”文件，出现 LINQ to SQL 类的“对象关系设计器”界面。在此界面中，可以通过拖拽方式来定义与数据库相对应的实体和关系。将“服务器资源管理器窗口”中“Hotel.mdf”节点 | “表”节点下的“Students”表拖拽到“对象关系设计器”的界面上，这时就

会生成一个如图 13-11 所示的实体类，该类包含了与表“Students”的字段对应的属性。

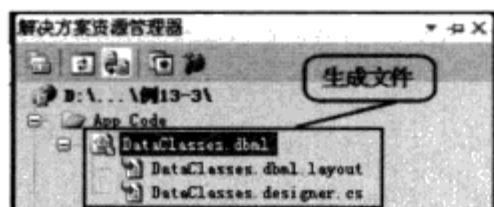


图 13-10 生成 DataClasses.dbml 文件



图 13-11 对象关系设计器

**08** 打开文件 DataClasses.designer.cs，可以看到该文件自动生成了包含 LINQ 到 SQL 实体类以及强类型 DataClassesDataContext 的定义。至此，实体类 Students 就创建完毕了，在页面代码中就可以像使用其他类型的类一样使用。



提示

在进行数据库映射前，必须把数据库中各个表中的关键字和各个表的主键都设计好，这样声明的映射类自动包含表的各种关联属性。如果没有选中表的主键字段，则不允许支持增加、删除、修改的操作。

### 13.3.3 LINQ 查询数据库

创建了对象模型后，就可以查询数据库了。LINQ to SQL 会将编写的查询转换成等效的 SQL 语句，然后把它们发送到服务器进行处理。具体来说，应用程序将使用 LINQ to SQL API 来请求查询执行，LINQ to SQL 提供程序随后会将查询转换成 SQL 文本，并委托 ADO 提供程序执行。ADO 提供程序将查询结果作为 DataReader 返回，而 LINQ to SQL 提供程序将 ADO 结果转换成用户对象的 IQueryable 集合。

LINQ to SQL 中的查询与 LINQ 中的查询使用相同的语法，只不过它们操作的对象有所差异，LINQ to SQL 查询中引用的对象是映射到数据库中的元素，示例代码如下。

```
1. DataClassesDataContext data = new DataClassesDataContext ();
2. var studentQuery = from s in data.Students
3. select s;
```

代码说明：第 1 行定义声明强类型 DataClassesDataContext 的对象 data。第 2、3 行定义隐藏变量 studentQuery，通过 LINQ 查询从实体类 Students 中获取查询到的数据。

**【例 13-4】**本例演示利用上例“例 13-3”中创建的实体类 Students 从数据表 Students 中获取数据并显示在 GridView 控件。

**01** 继续上面“例 13-3”的开发，用鼠标双击网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 GridView 控件、1 个 TextBox 控件和 1 个 Button 控件到“设计视图”中。切换到“源视图”，在<form>和</form>标记间添加如下代码。

```
1. <asp:GridView ID="GridView1" runat="server"></asp:GridView>
2.

```

```

3. 请输入姓名<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
4. <asp:Button ID="Button1" runat="server" Text="Button" style="height: 21px" />

```

代码说明：第 1 行定义一个服务器列表控件 GridView1。第 3 行定义一个服务器文本框控件 TextBox1。第 4 行定义一个服务器按钮控件 Button1。

**02** 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮，打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“穆哈咖啡”，最后，单击“确定”按钮。

**03** 打开 Button 控件的“属性”窗口，设置 Text 属性为“查询”，设置 onclick 属性为“Button1\_Click”。

**04** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. DataClassesDataContext da = new DataClassesDataContext();
4. var user = from s in da.Students
5. select s;
6. GridView1.DataSource = user;
7. GridView1.DataBind();
8. }
9. }
10. protected void Button1_Click(object sender, EventArgs e){
11. DataClassesDataContext da = new DataClassesDataContext();
12. var user=from s in da.Students
13. where s.StuName ==TextBox1.Text
14. select s;
15. GridView1.DataSource = user;
16. GridView1.DataBind();
17. }

```

代码说明：第 1 行处理 Default.aspx 页面 Page 的加载事件 Load。第 2 行判断当前加载的页面如果不是回传页面，则第 3 行定义声明强类型 DataClassesDataContext 的对象 da。第 4、5 行创建 LINQ 查询语句查询 Students 表中所有的实体对象。第 6 行将查询的结果作为列表控件 GridView1 的数据源。第 7 行将数据绑定到 GridView1 中显示。

第 10 行处理定义按钮控件 Button1 单击事件 Click 的方法。第 11 行定义声明强类型 DataClassesDataContext 的对象 da。第 12~14 行定义隐藏变量 user 通过 LINQ 查询从 Students 表中获得用户输入名字对象的信息数据。第 15 行将查询到的数据作为列表控件 GridView1 的数据源。第 16 行将数据绑定到 GridView1 中显示。

**05** 按下“Ctrl+F5”，运行结果如图 13-12 所示。

**06** 用户输入查询的名字，单击“查询”按钮，显示出如图 13-13 所示的查询结果。

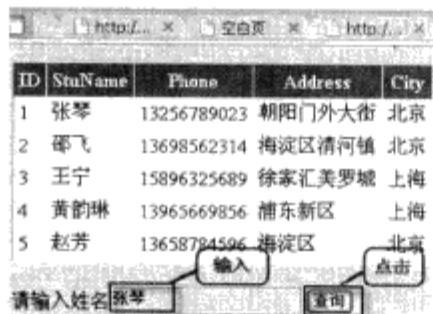


图 13-12 运行结果 1



图 13-13 运行结果 2

### 13.3.4 LINQ 更改数据库

开发人员可以使用 LINQ to SQL 对数据库进行插入、更新和删除操作。在 LINQ to SQL 中执行插入、更新和删除操作的方法是：向对象模型中添加对象、更改和移除对象模型中的对象，然后 LINQ to SQL 会把所做的操作转化成 SQL，最后把这些 SQL 提交到数据库执行。在默认情况下，LINQ to SQL 就会自动生成动态 SQL 来实现插入、读取、更新操作。我们也可以自定义 SQL 来实现一些特殊的功能。

#### 1. LINQ 插入数据库

使用 LINQ 向数据库插入行的操作步骤说明如下。

- ① 创建一个要提交到数据库的新对象。
- ② 将这个新对象添加到与数据库中目标数据表关联的 LINQ to SQL Table 集合。
- ③ 将更改提交到数据库。

**【例 13-5】**本例演示利用“例 13-1”中创建的实体类 Students，接受用户输入的数据并插入到数据库中。

**01** 继续上面“例 13-4”的开发，在程序中创建一个新的页面 Insert.aspx。

**02** 用鼠标双击“Insert.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 GridView 控件、5 个 TextBox 控件和 1 个 Button 控件到“设计视图”中。切换到“源视图”，在 <form>和</form>标记间添加如下代码。

```
1. <asp:GridView ID="GridView1" runat="server" ></asp:GridView>
2. 学号<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

3. 姓名<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

4. 电话<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

5. 地址<asp:TextBox ID="TextBox4" runat="server"></asp:TextBox>

6. 城市<asp:TextBox ID="TextBox5" runat="server"></asp:TextBox>

7. <asp:Button ID="Button1" runat="server" Text="Button" />
```

代码说明：第 1 行定义一个服务器列表控件 GridView1。第 2~6 行定义了 5 个服务器文本框控件 TextBox。第 7 行定义一个服务器按钮控件 Button1。

**03** 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“穆哈咖啡”，最后单击“确定”按钮。

**04** 打开 Button 控件的“属性”窗口，设置 Text 属性为“添加”，设置 onclick 属性为“Button1\_Click”。

**05** 用鼠标双击网站根目录下的“Insert.aspx.cs”文件，编写实现插入数据库的关键代码如下。

```
1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. Binder();
4. }
5. }
6. protected void Button1_Click(object sender, EventArgs e){
```

```

7. DataClassesDataContext da = new DataClassesDataContext();
8. Students s = new Students();
9. s.ID = Convert.ToInt32(TextBox1.Text);
10. s.StuName = TextBox2.Text;
11. s.Phone = TextBox3.Text;
12. s.Address = TextBox4.Text;
13. s.City = TextBox5.Text;
14. da.Students.InsertOnSubmit(s);
15. da.SubmitChanges();
16. Binder()
17. }
18. private void Binder(){
19. DataClassesDataContext da = new DataClassesDataContext();
20. var user = from s in da.Students
21. select s;
22. GridView1.DataSource = user;
23. GridView1.DataBind();
24. }

```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行判断如果当前加载的页面不是回传页面则第 3 行调用 Binder 方法显示数据。

第 6 行处理定义按钮控件 Button1 单击事件 Click 的方法。第 7 行定义声明强类型 DataClassesDataContext 的对象 da。第 8 行声明了一个 Students 实体类的对象 s，这是第一步。第 9~13 行给 s 对象的 5 个属性赋值，这些值是用户输入在文本框中的内容。第 14 行调用 InsertOnSubmit 方法向 LINQ to SQL Table (TEntity) 集合中插入该条数据，这是第二步。第 15 行调用方法 SubmitChanges 提交更改，这是第三步。第 16 行调用 Binder 方法显示数据。

第 18 行定义绑定列表控件 GridView1 控件的方法 Binder。第 19 行定义声明强类型 DataClasses1DataContext 的对象 da。第 20、21 行创建 LINQ 查询语句查询 Students 表中所有的实体对象。第 22 行将查询的结果作为列表控件 GridView1 的数据源。第 23 行调用 GridView1 控件的 DataBind 方法绑定数据。

**06** 右键单击网站根目录下的“Insert.aspx”文件，在弹出的菜单中选择“在浏览器中查看”命令，运行结果如图 13-14 所示。



图 13-14 运行结果

## 2. LINQ 修改数据库

使用 LINQ 修改数据库数据的操作步骤如下：

- 01 查询数据库中要更新的数据行。
- 02 对得到的 LINQ to SQL 对象中成员值进行更改。
- 03 将更改提交到数据库。

【例 13-6】本例演示利用上面的“例 13-1”中创建的实体类 Students，接受用户输入的新数据，并修改数据库中已有的信息。

- 01 继续上面“例 13-5”的开发，在程序中创建一个新的页面 Update.aspx。
- 02 用鼠标双击“Update.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 GridView 控件、3 个 TextBox 控件、1 个 Button 控件和 1 个 DropDownList 到“设计视图”中。切换到“源视图”，在<form>和</form>标记间添加如下代码。

```

1. <asp:GridView ID="GridView1" runat="server"></asp:GridView>

2. 姓名<asp:DropDownList ID="DropDownList1"
3. runat="server" Height="16px" Width="150px">
4. </asp:DropDownList>

5. 电话<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

6. 地址<asp:TextBox ID="TextBox4" runat="server"></asp:TextBox>

7. 城市<asp:TextBox ID="TextBox5" runat="server"></asp:TextBox>

8. <asp:Button ID="Button1" runat="server" Text="Button"
9. style="height: 21px" />


```

代码说明：第 1 行定义一个服务器列表控件 GridView1。第 2 行定义一个服务器下拉列表控件 DropDownList1 并设置其长度和高度。第 5~7 行定义了 3 个服务器文本框控件 TextBox。第 8 行定义一个服务器按钮控件 Button1 并设置其高度。

03 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮，打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“穆哈咖啡”，最后单击“确定”按钮

04 打开 Button 控件的“属性”窗口，设置 Text 属性为“更新”，设置 onclick 属性为“Button1\_Click”。

05 打开 DropDownList 控件的“属性”窗口，设置 AutoPostBack 属性为 True。

06 用鼠标双击网站根目录下的“Update.aspx.cs”文件，编写实现修改数据库的关键代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. Bind();
4. Binder();
5. }
6. }
7. protected void Button1_Click(object sender, EventArgs e) {
8. DataClassesDataContext da = new DataClassesDataContext();
9. var rusult = from s in da.Students
10. where s.StuName == DropDownList1.Text
11. select s;
12. foreach (Students s in rusult){
13. s.StuName = DropDownList1.Text;
14. s.Phone = TextBox3.Text;
15. s.Address = TextBox4.Text;
16. s.City = TextBox5.Text;

```

```
17. }
18. da.SubmitChanges();
19. Binder();
20. TextBox3.Text = "";
21. TextBox4.Text = "";
22. TextBox5.Text = "";
23. }
24. private void Bind(){
25. DataClassesDataContext da = new DataClassesDataContext();
26. var user = from s in da.Students
27. select s.StuName ;
28. DropDownList1.DataSource = user;
29. DropDownList1.DataBind();
30.
31. }
```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行判断如果当前加载的页面不是回传页面，则第 3 行调用 Binder 方法显示 GridView1 控件的数据。第 4 行调用 Bind 方法显示 DropDownList1 的控件数据的方法。

第 7 行处理定义按钮控件 Button1 单击事件 Click 的方法。第 8 行定义声明强类型 DataClassesDataContext 的对象 da。第 9~11 行利用 LINQ to SQL 从数据库查询到用户选择查询者的对象信息，这是第一步。然后在第 12~16 行通过 foreach 循环更新查询对象的属性值，这些值是用用户输入在文本框中的内容，这是第二步。第 18 行把更新提交到数据库以对数据库进行更新，这是第三步。第 19 行调用 Binder 方法显示数据。第 20~22 行将用户输入在文本框中的数据清空。

第 24 行定义绑定下拉列表控件数据的方法 Bind。第 25 行定义声明强类型 DataClassesDataContext 的对象 da。第 26~27 行利用 LINQ to SQL 从数据库查询出 Students 表中所有对象的姓名。第 28 行将查询的结果作为列表控件 DropDownList1 的数据源。第 29 行调用 DropDownList1 控件的 DataBind 方法绑定数据。

**07** 右键单击网站根目录下的“Update.aspx.cs”文件，在弹出的菜单中选择“在浏览器中查看”命令。运行结果如图 13-15 所示。

**08** 用户选择查询者，然后再输入修改的内容，最后单击“更新”按钮。修改后的页面如图 13-16 所示。



图 13-15 运行结果 1

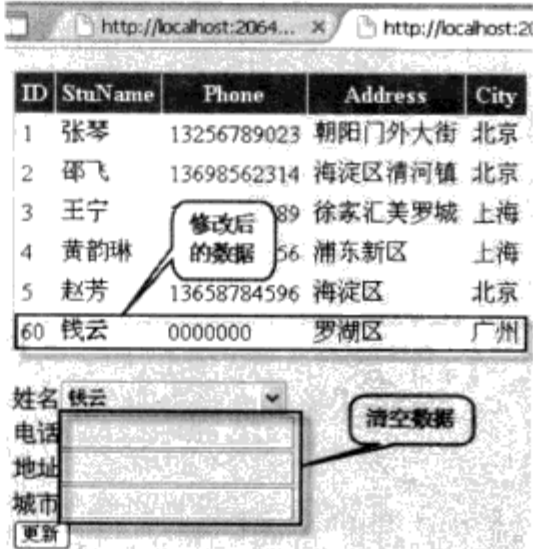


图 13-16 运行结果 2

### 3. LINQ 删除数据库

可以通过将对应的 LINQ to SQL 对象从相关的集合中去除来实现删除数据库中的行。不过，LINQ to SQL 不支持且无法识别级联删除操作。如果要在对行有约束的表中删除数据，则必须符合下面的条件之一：

- 在数据库的外键约束中设置 ON DELETE CASCADE 规则。
- 编写代码先删除约束表的级联关系。

删除数据库中数据行的操作步骤说明如下。

- 01 查询数据库中要删除的行。
- 02 调用 DeleteOnSubmit 方法。
- 03 将更改提交到数据库。

**【例 13-7】**本例演示利用上面的“例 13-1”中创建的实体类 Students，接受用户输入的姓名，删除数据库中该对象的数据信息。

- 01 继续上面“例 13-6”的开发，在程序中创建一个新的页面 Delete.aspx。

02 用鼠标双击“Delete.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 GridView 控件、1 个 Button 控件和 1 个 DropDownList 到“设计视图”中。切换到“源视图”，在 <form>和</form>标记间添加如下代码。

```
1. <asp:GridView ID="GridView1" runat="server"></asp:GridView>
2.
 请选择姓名 <asp:DropDownList ID="DropDownList1" runat="server" Height="16px" Width="73px">
</asp:DropDownList>
3. <asp:Button ID="Button1" runat="server" Text="Button" />
```

代码说明：第 1 行定义一个服务器列表控件 GridView1。第 2 行定义一个服务器下拉列表控件 DropDownList1 并设置其长度和高度。第 3 行定义一个服务器按钮控件 Button1。

03 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“穆哈咖啡”，最后单击“确定”按钮。

04 打开 Button 控件的“属性”窗口，设置 Text 属性为“删除”，设置 onclick 属性为“Button1\_Click”。

05 打开 DropDownList 控件的“属性”窗口，设置 AutoPostBack 属性为 True。

06 用鼠标双击网站根目录下的“Delete.aspx.cs”文件，编写实现删除数据库数据的关键代码如下。

```
1. protected void Button1_Click(object sender, EventArgs e){
2. DataClassesDataContext da = new DataClassesDataContext();
3. var result = from s in da.Students
4. where s.StuName == DropDownList1.Text
5. select s;
6. da.Students.DeleteAllOnSubmit(result);
7. da.SubmitChanges();
8. Binder();
9. Bind();
10. }
```

代码说明：第 1 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 2 行定义声明强类型 DataClasses1DataContext 的对象 data。第 3~5 行利用 LINQ to SQL 从数据库查询到用户选择的对象。第 6 行调用方法 DeleteOnSubmit 删除获得对象。第 7 行把更改提交到数据库对数据进行删除。第 8 行调用 Binder 方法显示 GridView1 控件的数据。第 9 行调用 Bind 方法显示 DropDownList1 的控件数据的方法。

**07** 右键单击网站根目录下的“Delete.aspx.cs”文件，在弹出的菜单中选择“在浏览器中查看”命令。运行结果如图 13-17 所示。

**08** 用户选择要删除的对象，然后单击“删除”按钮，删除对象后显示的页面如图 13-18 所示。



图 13-17 运行结果 1



图 13-18 运行结果 2



LINQ to SQL 类包含了一个代码文件\*.cs。开发人员可以在该文件中对每个类添加自己需要的实现，然后在 LINQ 查询中使用，从而大大增强了查询的功能。

### 13.4 LinqDataSource 控件

LinqDataSource 控件为用户提供了一种将数据控件连接到多种数据源的方法，其中包括数据库数据、数据源类和内存中的集合。通过使用 LinqDataSource 控件，用户可以针对所有这些类型的数据源指定类似于数据库检索的任务（选择、筛选、分组和排序）。可以指定针对数据库表的修改任务（更新、删除和插入）。

用户可以使用 LinqDataSource 控件连接存储在公共字段或属性中的任何类型的数据集合。对于所有数据源来说，用于执行数据操作的声明性标记和代码都是相同的。用户可以使用相同的语法，与数据库表中的数据或数据集合（与数组类似）中的数据进行交互。

如果要显示 LinqDataSource 控件中的数据，可将数据绑定控件绑定到 LinqDataSource 控件。例如，将 DetailsView 控件、GridView 控件或 ListView 控件绑定到 LinqDataSource 控件。为此，必须将数据绑定控件的 DataSourceID 属性设置为 LinqDataSource 控件的 ID。

数据绑定控件将自动创建用户界面以显示 LinqDataSource 控件中的数据，它还提供用于对数据进行排序和分页的界面。在启用数据修改后，数据绑定控件会提供用于更新、插入和删除记录的界面。

通过将数据绑定控件配置为不自动生成数据控件字段，可以限制显示的数据（属性）。然后可以在数据绑定控件中显式定义这些字段。虽然 LinqDataSource 控件会检索所有属性，但数据绑

定控件仅显示指定的属性。

LinqDataSource 控件的常用属性如表 13-4 所示。

表 13-4 LinqDataSource 控件的常用属性

属性	说明
AutoPage	获取或设置一个值，该值指示 LinqDataSource 控件是否支持在运行时对数据的各部分进行导航
AutoSort	获取或设置一个值，该值指示 LinqDataSource 控件是否支持在运行时对数据进行排序
ClientID	获取由 ASP.NET 生成的服务器控件标识符
Context	为当前 Web 请求获取与服务器控件关联的 HttpContext 对象
Controls	获取 ControlCollection 对象，该对象表示 UI 层次结构中指定服务器控件的子控件
DeleteParameters	获取在删除操作过程中使用的参数的集合
EnableDelete	获取或设置一个值，该值指示是否可以通过 LinqDataSource 控件删除数据记录
EnableInsert	获取或设置一个值，该值指示是否可以通过 LinqDataSource 控件插入数据记录
EnableTheming	获取一个值，该值指示此控件是否支持主题
EnableUpdate	获取或设置一个值，该值指示是否可以通过 LinqDataSource 控件更新数据记录
GroupBy	获取或设置一个值，该值指定用于对检索到的数据进行分组的属性
GroupByParameters	获取用于创建 Group By 子句的参数的集合
ID	获取或设置分配给服务器控件的编程标识符
InsertParameters	获取在插入操作过程中使用的参数的集合
OrderBy	获取或设置一个值，该值指定用于对检索到的数据进行排序的字段
OrderByParameters	获取用于创建 Order By 子句的参数的集合
OrderGroupsBy	获取或设置用于对分组数据进行排序的字段
OrderGroupsByParameters	获取用于创建 Order Groups By 子句的参数集合
Select	获取或设置属性和计算值，它们包含在检索到的数据中
SelectParameters	获取在数据检索操作过程中使用的参数的集合
TemplateControl	获取或设置对包含该控件的模板的引用
UpdateParameters	获取在更新操作过程中使用的参数的集合
Visible	获取或设置一个值，该值指示是否以可视化方式显示控件
Where	获取或设置一个值，该值指定要将记录包含在检索到的数据中必须为真的条件
WhereParameters	获取用于创建 Where 子句的参数集合

【例 13-8】本例演示如何利用 LinqDataSource 控件来通过“例 13-3”中创建的对象模型 DataClassesDataContext 把从实体类 Students 获得的数据绑定到 GridView 控件中并实现编辑、更新和删除的功能。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 13-8”。
- 02 按照前面“例 13-3”的步骤，创建数据库实体映射类 Students。
- 03 用鼠标双击网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 GridView 控件到“设计视图”中。
- 04 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮，打开“GridView 任务”列表，在如图 13-19 所示的“选择数据源”下拉列表中选择“新建数据源”。

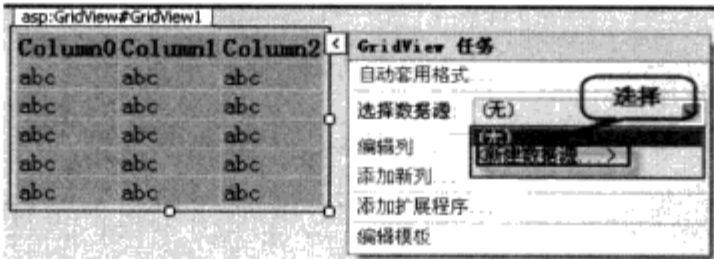


图 13-19 “GridView 任务”列表

05 弹出如图 13-20 所示“数据源配置向导”对话框，在“应用程序从哪里获取数据？”列表中选择“LINQ”数据源，将生成的 LinqDataSource 控件的 ID 属性命名为“LinqDataSource1”，单击“确定”按钮。

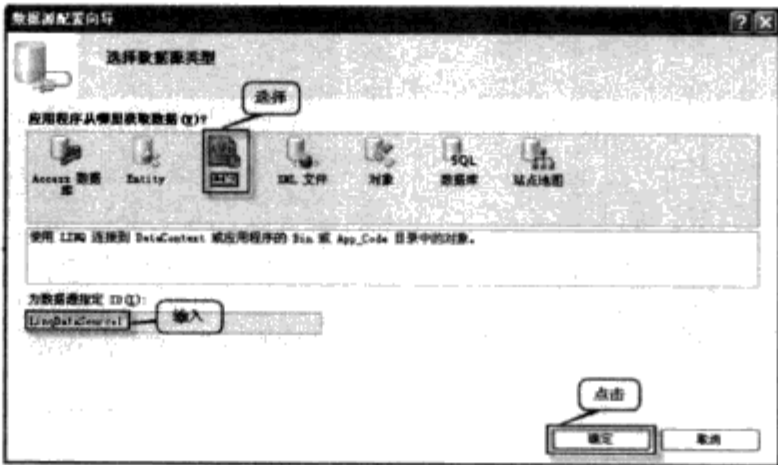


图 13-20 “数据源配置向导”对话框

06 弹出如图 13-21 所示的“选择上下文对象”对话框。由于创建了数据库实体映射类 Students，会自动生成一个上下文对象 DataBaseDataContext，通常在一个项目中只有一个数据库上下文对象，选择这一对象，单击“下一步”按钮。

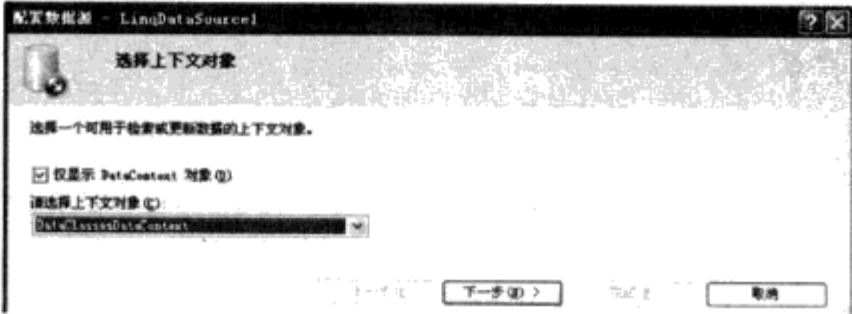


图 13-21 “选择上下文对象”对话框

07 弹出如图 13-22 所示“配置数据选择”对话框，单击“高级”按钮。

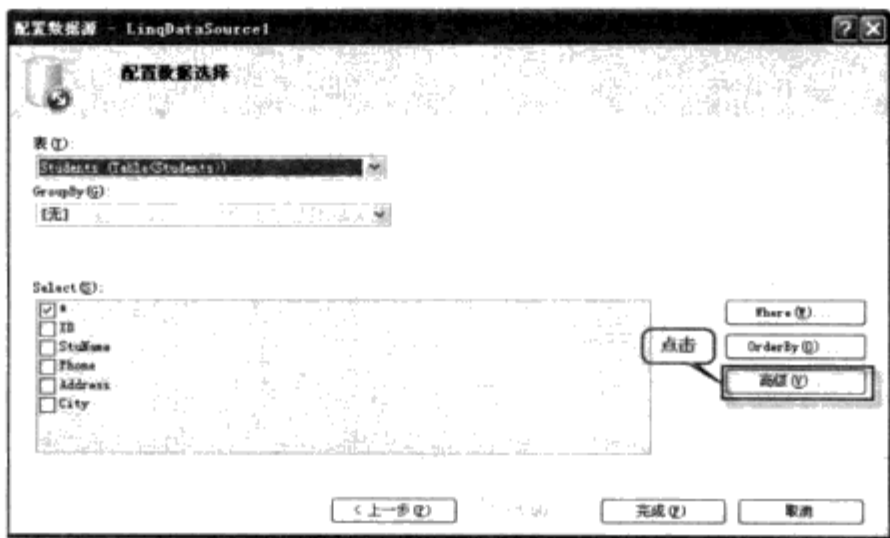


图 13-22 “配置数据选择”对话框

08 弹出如图 13-23 所示的“高级选项”对话框。选中所有的复选按钮，启用 LinqDataSource 控件的自动删除、插入和更新的功能，然后，单击“确定”按钮，回到上图 13-22 “配置数据选择”对话框界面，单击“完成”按钮。结束数据源的配置。完成配置后，自动生成一个名为 LinqDataSource1 的数据源配置控件，它支持添加、删除、修改操作。

09 打开如图 13-24 所示的“GridView 任务”列表，选中“启动分页”、“启动排序”、“启动编辑”和“启动删除”四个复选按钮。

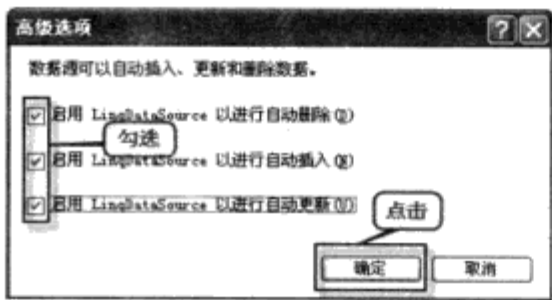


图 13-23 “高级选项”对话框



图 13-24 “GridView 任务”列表

10 切换到“源视图”，在<form>和</form>标记间自动生成如下代码。

```
1. <asp:GridView ID="GridView1" runat="server" AllowPaging="True" AllowSorting="True" AutoGenerateColumns="False"
DataKeyNames="ID" DataSourceID="LinqDataSource1">
2. <Columns>
3. <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
4. <asp:BoundField DataField="ID" HeaderText="ID" ReadOnly="True" SortExpression="ID" />
5. <asp:BoundField DataField="StuName" HeaderText="StuName" SortExpression="StuName" />
6. <asp:BoundField DataField="Phone" HeaderText="Phone" SortExpression="Phone" />
7. <asp:BoundField DataField="Address" HeaderText="Address" SortExpression="Address" />
8. <asp:BoundField DataField="City" HeaderText="City" SortExpression="City" />
9. </Columns>
10. </asp:GridView>
11. <asp:LinqDataSource ID="LinqDataSource1" runat="server" ContextTypeName="DataClassesDataContext"
EnableDelete="True" EnableInsert="True" EnableUpdate="True">
```

代码说明：第 1 行定义一个服务器列表控件 GridView1，设置其允许分页、允许排序、禁止自动生成列、数据源为 LinqDataSource1、数据主键为 ID 字段。第 2~9 行定义 GridView1 控件的列。

其中第 3 行设置显示操作删除和编辑的命令按钮。第 4~8 行分别定义列表控件的四个列字段，它们关联到数据表 `Students` 中的四个字段并设置显示的列标题和绑定的字段值以及排序表达式的字段。第 11 行定义一个服务器 `LinqDataSource1` 控件并设置能够进行删除、插入、更新操作以及包含表属性的上下文类型。

**11** 打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“简明型”，单击“确定”按钮。

**12** 按快捷键“Ctrl+F5”，运行结果如图 13-25 所示。



ID	StuName	Phone	Address	City
1	张琴	13256789023	朝阳门外大街	北京
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海
5	赵芳	13658784596	海淀区	北京

图 13-25 运行结果 1

**13** 单击表中第一行数据中的“编辑”按钮。进入如图 13-26 所示的编辑操作。每一列可编辑的数据都以文本框的形式出现，这样用户就可以修改其中的数据。输入新的姓名“钱云”、新的电话“13918756643”和新的地址“汉阳区汉正街”和新的城市“武汉”。如果想取消更新操作，可以单击“取消”按钮，回到图 13-25 所示的界面。如果确认要进行更新操作，就单击“更新”按钮。



ID	StuName	Phone	Address	City
1	钱云	13918756643	汉阳区汉正街	武汉
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海
5	赵芳	13658784596	海淀区	北京

图 13-26 运行结果 2

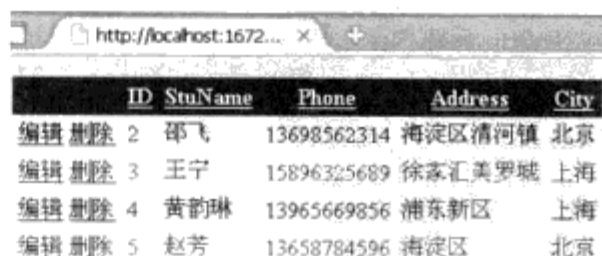
**14** GridView 控件显示如图 13-27 所示的更新数据后的界面。第一行中新的数据显示在列表中，如果要进行删除此条新的数据，单击“删除”按钮即可。

**15** 删除第一条数据后的 GridView 控件如图 13-28 所示。



ID	StuName	Phone	Address	City
1	钱云	13918756643	汉阳区汉正街	武汉
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海
5	赵芳	13658784596	海淀区	北京

图 13-27 运行结果 3



ID	StuName	Phone	Address	City
2	邵飞	13698562314	海淀区清河镇	北京
3	王宁	15896325689	徐家汇美罗城	上海
4	黄韵琳	13965669856	浦东新区	上海
5	赵芳	13658784596	海淀区	北京

图 13-28 运行结果 4



在进行数据库映射前，必须把数据库各个表中的关键字和各个表的主键都设计好，这样声明的映射类自动包含表的各种关联属性。如果没有选中表的主键字段，则不允许支持增加、删除、修改的操作。

### 13.5 新增的查询扩展控件——QueryExtender

任何以数据驱动的 Web 网站，创建搜索页面都是一项常见而重复的工作。通常情况下，开发人员需要创建一个带 where 条件的 select 查询，由页面的输入控件提供查询参数。从 .Net 2.0 框架开始，在 DataSource 的数据访问控件集的帮助下，数据访问变得相对的容易。但是，已有的数据源控件对于创建复杂过滤条件的查询页面仍然无法轻易地完成。因此，微软公司在 ASP.NET 4.0 中引入了一个扩展查询的控件——QueryExtender。

QueryExtender 控件是为了简化 LinqDataSource 控件或 EntityDataSource 控件返回的数据过滤而设计的，它主要是将过滤数据的逻辑从数据控件中分离出来。QueryExtender 控件的使用非常简单，只需要往页面上增加一个 QueryExtender 控件，指定其数据源是哪个控件并设置过滤条件就可以了。比如，当在页面中显示产品的信息时，可以使用该控件去显示那些在某个价格范围的产品，也可以搜索用户指定名称的产品。

当然，不使用 QueryExtender 控件的话，LinqDataSource 和 EntityDataSource 控件也都是可以过滤数据的。因为这两个控件都有一个 where 的属性，能指定过滤数据的条件。但是 QueryExtender 控件提供的是一种更为简单的方式去过滤数据。

QueryExtender 控件使用筛选器从数据源中检索数据，并且在数据源中不使用显式的 Where 子句。利用该控件，能够通过声明性语法从数据源中筛选出数据。使用 QueryExtender 控件有以下优点。

- 与编写 Where 子句相比，可以提供功能更丰富的筛选表达式。
- 提供一种 LinqDataSource 和 EntityDataSource 控件均可使用的查询语言。例如，如果将 QueryExtender 与这些数据源控件配合使用，则可以在网页中提供搜索功能，而不必编写特定于模型的 Where 子句或 SQL 语句。
- 能够与 LinqDataSource 或 EntityDataSource 控件配合使用或与第三方数据源配合使用。
- 支持多种可单独和共同使用的筛选选项。

QueryExtender 控件支持多种可用于筛选数据的选项。该控件支持搜索字符串、搜索指定范围内的值、将表中的属性值与指定的值进行比较、排序和自定义查询。在 QueryExtender 控件中以 LINQ 表达式的形式提供这些选项。QueryExtender 控件还支持 ASP.NET 动态数据专用的表达式。表 13-5 列出了 QueryExtender 控件的筛选选项。

表 13-5 QueryExtender 控件的筛选选项。

表达式	说明
QueryExtender	表示控件的主类
CustomExpression	为数据源指定用户定义的表达式。自定义表达式可以位于函数中，并且可以从页面标记中调用
OrderByExpression	将排序表达式应用于 IQueryable 数据源对象
PropertyExpression	根据 WhereParameters 集合中的指定参数创建 Where 子句
RangeExpression	确定值大于还是小于指定的值，或者值是否在两个指定的值之间

(续表)

表达式	说明
SearchExpression	搜索一个或多个字段中的字符串值，并将这些值与指定的字符串值进行比较
ThenByExpressions	应用 OrderByExpression 表达式后将排序表达式应用于 IQueryable 数据源对象
DynamicFilterExpression	使用指定的筛选器控件生成数据库查询
ControlFilterExpression	使用在源数据绑定控件中选择的数据键生成数据库查询

**【例 13-9】**本例演示如何利用 LinqDataSource 控件和 QueryExtender 控件实现在页面对指定的字符串和指定范围值的筛选查询。使用的数据库“st\_FlowerPrearrange.mdf”和数据表“Flower”在本例的源代码中提供给读者，不需重新创建直接附加到数据库使用即可。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 13-9”。
- 02** 按照“例 13-3”的步骤，创建数据库实体映射类 Flower。
- 03** 用鼠标双击网站的根目录下“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 3 个 Textbox 控件、1 个 Button 控件、1 个 GridView 控件和 1 个 QueryExtender 控件到“设计视图”中。
- 04** 打开 GridView 控件的“GridView 任务”列表，在“选择数据源”下拉列表中选择“新建数据源”。
- 05** 在弹出“数据源配置向导”对话框中选择“LINQ”数据源，单击“确定”按钮。
- 06** 在弹出的“选择上下文对象”对话框中单击“下一步”按钮。
- 07** 在弹出“配置数据选择”对话框中单击“完成”按钮。结束 LinqDataSource 控件的数据源配置。
- 08** 打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“雪松”，单击“确定”按钮。
- 09** 切换到“源视图”，在<form>和</form>标记间编写 QueryExtender 控件的定义代码。

```
1. <asp:QueryExtender ID="QueryExtender1" runat="server" TargetControlID="LinqDataSource1">
2. <asp:SearchExpression DataFields="NAME" SearchType="StartsWith">
3. <asp:ControlParameter ControlID="TextBox1" />
4. </asp:SearchExpression>
5. <asp:RangeExpression DataField="Price" MaxType="Inclusive" MinType="Inclusive">
6. <asp:ControlParameter ControlID="TextBox2" />
7. <asp:ControlParameter ControlID="TextBox3" />
8. </asp:RangeExpression>
9. </asp:QueryExtender>
```

代码说明：第 1 行定义一个服务器查询扩展控件 QueryExtender1 并设置其获取数据的关联控件为 LinqDataSource1。第 2~4 行定义该控件搜索字符串筛选表达式 SearchExpression。其中，第 2 行设置绑定搜索字段为 Flower 表中的 Name 字段，设置搜索类型为从字段的任意位置开始搜索。第 3 行设置从文本框控件 TextBox1 获得查询的控件参数。第 5~8 行定义 QueryExtender1 控件搜索值范围的筛选表达式 RangeExpression。其中，第 5 行设置控件的绑定搜索字段为 Flower 表中的 Price 字段以及设置搜索范围包括最大值和最小值。第 6~7 行设置从文本框控件 TextBox2 和 TextBox3 分别获得查询的控件参数。

10 按下“Ctrl+F5”，运行结果如图 13-29 所示。



图 13-29 运行结果 1

11 用户输入查询字符“爱”，价格范围从 100 元到 200 元，然后单击“搜索”按钮。结果如图 13-30 所示，它显示所有符合查询条件的结果。



图 13-30 运行结果 2



QueryExtender 控件只能配合 LinqDataSource 和 EntityDataSource 控件使用，它不能去过滤 SqlDataSource 控件的数据集。这也就意味着，使用该控件的话，必须使用 LINQ 或者 ADO.NET Entity 框架。

### 13.6 上机题

- 1. 在 SQL Server 2005 中创建数据库“Manager”和员工信息表“Employee”，表结构如图 13-31 所示。要求创建一个对象模型 DataClassesDataContext，把员工信息表映射到内存中。
- 2. 利用上机题 1 创建的对象模型 DataClassesDataContext，查询出员工信息表“Employee”中包含的所有数据，并显示在 GridView 控件中，运行结果如图 13-32 所示。

列名	数据类型	允许空
ID	nvarchar(50)	<input type="checkbox"/>
Name	nvarchar(50)	<input type="checkbox"/>
Age	int	<input type="checkbox"/>
Sex	nvarchar(50)	<input type="checkbox"/>
Salary	int	<input type="checkbox"/>

图 13-31 数据表结构

ID	Name	Age	Sex	Salary
1	李明	38	男	3000
2	林梅	35	女	2800
3	吴平	48	男	3300
4	刘海	40	男	3100
5	王芬	40	女	3000
6	程峰	39	男	2900
7	潘超	38	男	3200

图 13-32 运行结果

- 3. 使用 LINQ to SQL 技术，通过 LinqDataSource 控件和 DetailView 控件实现添加员工信息表

“Employee” 中数据的功能并显示在 Gridview 控件上，运行结果如图 13-33 所示。

ID	Name	Age	Sex	Salary
1	李明	38	男	3000
2	林梅	35	女	2800
3	吴平	48	男	3300
4	刘海	40	男	3100
5	王芬	40	女	3000
6	程峰	39	男	2900
7	潘超	38	男	3200

ID

8

Name

胡虎

Age

37

Sex

男

Salary

2700

插入

取消

图 13-33 运行结果

4. 使用 LINQ to SQL 技术，通过 LinqDataSource 控件和 ListView 控件实现修改员工信息表 “Employee” 中数据的功能，将编号为 “1” 的员工姓名修改为 “李飞”、年龄修改为 “48”、性别修改为 “女”、工资修改为 “3200”，运行结果如图 13-34 所示。

	ID	Name	Age	Sex	Salary
更新	1	李飞	48	女	3200
编辑	2	林梅	35	女	2800
编辑	3	吴平	48	男	3300
编辑	4	刘海	40	男	3100
编辑	5	王芬	40	女	3000

第一页12最后一页

图 13-34 运行结果

5. 使用 LINQ to SQL 技术，通过 LinqDataSource 控件和 ListView 控件实现删除员工信息表 “Employee” 中数据的功能，运行结果如图 13-35 所示。

6. 使用 LINQ to SQL 技术，通过 LinqDataSource 控件、QueryExtender 控件和 ListView 控件实现查询员工信息表 “Employee” 中工资在 3000~4000 元的所有员工信息，运行结果如图 13-36 所示。

7. 通过 LINQ 查询表达式从数字集合 { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 } 中，获取小于 5 的数组元素，并显示在网页上，运行结果如图 13-37 所示。

	ID	Name	Age	Sex	Salary
删除	1	李飞	48	女	3200
删除	2	林梅	35	女	2800
删除	3	吴平	48	男	3300
删除	4	刘海	40	男	3100
删除	5	王芬	40	女	3000
删除	6	程峰	39	男	2900

第一页12最后一页

图 13-35 运行结果

从3000元到4000元查询

ID	Name	Age	Sex	Salary
1	李飞	48	女	3200
3	吴平	48	男	3300
4	刘海	40	男	3100
5	王芬	40	女	3000
7	潘超	38	男	3200

第一页1最后一页

图 13-36 运行结果

小于5的数字有:

4

1

3

2

0

图 13-37 运行结果

# 第 14 章 ASP.NET AJAX 技术

## 学习目标

由 ASP.NET 2.0 扩展而来的 ASP.NET AJAX 能够快速创建包含具有响应能力和丰富用户界面的网页。它包括了一个客户端脚本库，这些脚本将跨浏览器和动态技术结合在一起，并与基于 ASP.NET 服务器的开发平台集成。通过使用 ASP.NET AJAX 功能，可以很大程度上改进用户的页面体验，并提高 Web 应用程序的开发效率。同时 AJAX Control Toolkit 工具集的出现，使 ASP.NET AJAX 得到了更为广泛的应用。读者通过本章内容的学习，就基本能够掌握好 ASP.NET AJAX 技术的使用。

## 本章重点

- 理解 ASP.NET AJAX 的概念和原理
- 掌握如何使用 ScriptManager 控件调用 Web 服务和 JS 文件
- 熟练的运用 UpdatePanel 控件实现局部刷新功能
- 掌握 UpdateProgress 和 Timer 控件的使用

## 14.1 ASP.NET AJAX 技术概述

ASP.NET AJAX 是微软公司专门为 ASP.NET 应用程序提供 AJAX 技术支持的开发框架，通过它原有的 ASP.NET 应用程序可以很轻松地使用 ASP.NET AJAX 所提供的基础架构，开发出具有 AJAX 能力的 Web 应用程序。

ASP.NET AJAX 能够快速创建具有丰富用户体验的页面，而且这些页面由安全的用户接口元素组成。ASP.NET AJAX 提供了一个客户端脚本（client-script）库，包含跨浏览器的 ECMAScript（如 JavaScript）和动态 HTML（DHTML）技术，而且 ASP.NET AJAX 把这些技术同 ASP.NET 开发平台结合起来。使用 ASP.NET AJAX，可以很大程度上的改善 Web 程序的用户体验和提高应用程序执行效率。

与那些完全基于服务器端的 Web 应用程序相比，ASP.NET AJAX 能够创建丰富的 Web 应用程序，它提供了以下优势。

- 提高浏览器中 Web 页面的执行效率。
- 包含了开发人员熟悉的 UI 元素，比如进程指标控件、Tooltips 控件和弹出式的窗口。
- 实现了页面的局部刷新，只刷新已被更新的页面。
- 实现客户端与 ASP.NET 应用服务的集成以进行表单认证和用户配置。

- 通过调用 Web 服务整合不同的数据源数据。
- 简化了服务器控件的定制来实现客户端功能。
- 支持最流行的和通用的浏览器，包括 IE，Firefox 和 Safari。
- 具有可视化的开发界面，使用 Visual Studio 2010 可以轻松自如地开发 AJAX 程序。

### 14.1.1 体系结构

ASP.NET AJAX 由客户端脚本库和服务端组件组成，它们互相配合提供了一个健壮的开发框架。除了 ASP.NET AJAX，还可使用 ASP.NET AJAX Control Toolkit 控件工具集。

#### 1. 客户端特征

ASP.NET AJAX 客户端脚本库是纯面向对象的 JavaScript 客户端脚本框架并且是可扩展的，允许开发人员很容易地构建拥有丰富 UI 功能和连接 Web Service 的 AJAX 网页应用程序。使用 ASP.NET AJAX，开发人员能够使用 DHTML、JavaScript 和 XMLHttpRequest 来编写 Web 应用程序，而无需掌握这些技术的细节。

ASP.NET AJAX 客户端脚本框架可以在所有常用浏览器上运行，而不需要 Web 服务器。它不需要安装，只要在页面中引用正确的脚本文件即可。

ASP.NET AJAX 客户端脚本框架包括以下各层内容。

- 浏览器兼容层。这个层为 ASP.NET AJAX 脚本提供了各种常用浏览器的兼容性，这些浏览器包括微软的 IE，Mozilla 的 Firefox，苹果的 Safari 等。
- ASP.NET AJAX 核心服务。这个核心服务扩展了 JavaScript，例如把类、命名空间、事件句柄、继承、数据类型、对象序列化扩展到 JavaScript 中。
- ASP.NET AJAX 的基础类库。这个类库包括组件，例如字符串创建器和扩展错误处理。
- 网络层。该层用来处理基于 Web 服务和应用程序的通信，以及管理异步远程方法的调用。

#### 2. 服务器端特征

微软公司专门为 ASP.NET 应用程序设计了一组 AJAX 风格的服务器控件，并且加强了现有的 ASP.NET 页面框架和控件，以便支持 ASP.NET AJAX 客户端脚本框架。

##### (1) 脚本支持，包括对“异步客户端回调”的支持

“异步客户端回调”的特性，使得构建没有中断的页面变得很容易。“异步客户端回调”包装了 XMLHttpRequest，能够在很多浏览器上工作。ASP.NET 本身包括了很多使用回调的控件，包括具有客户端分页和排序功能的 GridView 和 DataView 控件，以及 TreeView 控件的虚拟列表支持。ASP.NET AJAX 客户端脚本框架将完全支持 ASP.NET 的回调，但微软希望进一步增强浏览器和服务端之间的集成性。例如，可以将 ASP.NET AJAX 客户端控件的数据绑定为服务器上的 ASP.NET 数据源控件，并且可以从客户端异步地控制 Web 页面的显示。

##### (2) Web Service 集成

服务端框架使用了一套扩展的机制使程序中的 Web Service 可以被客户端 JavaScript 直接访问。只要在 Web Service 上标记 [ScriptService] 的属性，就可以简单地使该 Web Service 能够被客

户端的 JavaScript 直接访问。

(3) 应用程序服务

服务端框架提供了一些内置的应用服务，如授权服务 Authentication 和个性化支持服务 Prifile。

(4) 服务器端控件

ASP.NET AJAX 服务器控件包括服务器和代码，以实现类似于 AJAX 的行为。表 14-1 列出了最常用的 ASP.NET AJAX 服务器控件。

表 14-1 最常用的 ASP.NET AJAX 服务器控件

控件	描述
ScriptManager	管理客户端组件的脚本资源，局部页面的绘制，本地化和全局文件，并且可以定制用户脚本。为了使用 UpdatePanel，Updateprogress 和 Timer 控件，ScriptManager 控件是必须的
UpdatePanel	通过异步调用来刷新部分页面而不是刷新整个页面
Updateprogress	提供 UpdatePanel 控件中部分页面更新的状态信息
Timer	定义执行回调的时间区间。可以使用 Timer 控件来发送整个页面，也可以把它和 UpdatePane 控件一起使用在一个时间区间以执行局部页面刷新

上面描述的 ASP.NET AJAX 体系结构可以用图 14-1 来表示。



图 14-1 ASP.NET AJAX 体系结构

14.1.2 创建 ASP.NET AJAX 程序

在.NET 框架 4.0 中，ASP.NET AJAX 框架技术已经完全集成进来，所以，在使用 Visual Studio 2010 开发 ASP.NET AJAX 程序时候，就不需要再单独安装 ASP.NET AJAX 框架，而是可以直接创建 ASP.NET AJAX 程序。

下面通过一个例子来演示如何在 Visual Studio 2010 中创建 ASP.NET AJAX 程序。

- 01
- 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“AjaxWeb”。
- 02
- 用鼠标双击网站根目录下的“Default.aspx”文件，切换到“设计”视图。打开“工具箱”窗口，可以看到 ASP.NET AJAX 服务器控件，它们在如图 14-2 所示的 AJAX Extentions 选项卡中。

**03** 可以像拖拽其它控件一样，把 ASP.NET AJAX 服务器控件拖动到页面内，现在向页面拖放一个 ScriptManager 控件和 UpdatePanel 控件，如图 14-3 所示。

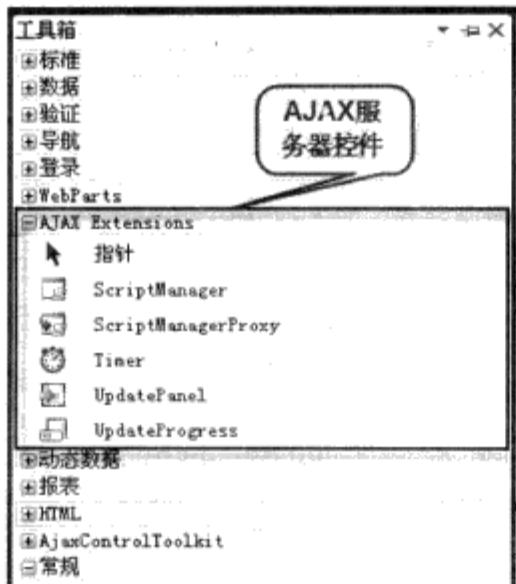


图 14-2 ASP.NET AJAX 服务器控件

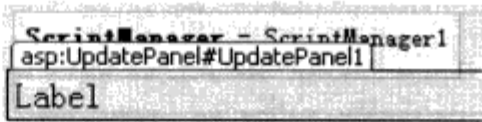


图 14-3 设计视图

**04** 切换到“源视图”，在<form></form>标记之间生成代码如下所示。

```
1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. </asp:ScriptManager>
3. <asp:UpdatePanel ID="UpdatePanel1" runat="server">
4. <ContentTemplate>
5. <asp:Label runat="server" Text="Label"></asp:Label>
6. </ContentTemplate>
7. </asp:UpdatePanel>
```

代码说明：第 1、2 行定义一个服务器脚本管理控件 ScriptManager1。第 3~7 行定义一个服务器更新面板控件 UpdatePanel1，其中在第 5 行又定义了一个服务器标签控件 Label1。

**05** 然后，在 Default.aspx.cs 文件中编辑后台的逻辑代码实现功能即可。



使用 Visual Studio 2010 可以轻松创建 AJAX 程序，只需要把相关控件拖拽到页面即可，而其余的工作就和创建普通的 ASP.NET 应用程序完全一样。

## 14.2 ASP.NET AJAX 核心控件

ASP.NET AJAX 提供了实现 AJAX 功能的服务器端控件，通过这些控件即使不懂任何客户端 AJAX Library 脚本的编程人员，也能够 在 ASP.NET 应用程序中创建简单的 AJAX 应用。这些核心的服务器控件有 ScriptManager、UpdatePanel、UpdateProgress 和 Timer 控件。

### 14.2.1 ScriptManager 控件

脚本管理控件 ScriptManager 是 AJAX 程序运行的基础。ScriptManager 控件包括在 ASP.NET 4.0 AJAX Extension 中，它用来处理页面上所有组件以及页面局部更新，生成相关客户端代理脚本以便能够在 JavaScript 中访问 Web Service 等。

1. ScriptManager 的结构

在支持 ASP.NET AJAX 的 ASP.NET 页面中，有且只能有一个 ScriptManager 控件来管理 ASP.NET AJAX 相关的控件和脚本。可以在 ScriptManager 控件中指定需要的脚本库，也可以通过注册 JavaScript 脚本来调用 Web 服务等。

一个 ScriptManager 的典型定义如下：

```
1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. <Scripts/>
3. <ProfileService />
4. <AuthenticationService />
5. </asp:ScriptManager>
```

以上代码中 Scripts、Services、ProfileService、AuthenticationService 等子标签都是可选的，这些子标签的意义如表 14-2 所示。

表 14-2 ScriptManager 子标签的含义

标签	描述
Scripts	对脚本的调用，其中可以嵌套多个 ScriptReference 模板以实现对多个脚本文件的调用
Services	对 Web 服务的调用，可以嵌套多个多个 ScriptReference 模板以实现对多个脚本文件的调用
ProfileService	表示提供个性化服务的路径，Profile 是在 .NET 3.5 新增的个性设置
AuthenticationService	用来表示提供验证服务的路径

上表中使用最多的是 Scripts、Services 两个标签，Scripts 标签引用自定义的 Javascript 的语法为：

```
1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. <Scripts >
3. <asp:ScriptReference Path ="Javascript 文件的路径" />
4.
5. </Scripts>
6. </asp:ScriptManager>
```

以上代码，第 1 行定义服务器脚本管理控件 ScriptManager1。第 2~4 行定义 Scripts 标签。第 3 行定义 ScriptReference 标签来指定引用的 Javascript 脚本文件并设置属性 Path 获得脚本文件的路径。

Scripts 标签引用的 Javascript 脚本文件可以超过一个，只要逐一应用<asp:ScriptReference>标签列出即可。如果引用的不是独立的 Javascript 文件，而是 Javascript 函数库中的某一个 Javascript 程序，则要使用<asp:ScriptReference>标签的另外两个属性 Assembly 和 Name。示例代码如下。

```
1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. <Scripts >
3. <asp:ScriptReference Assembly="Javascript 文件的路径" Name="Javascript 文件"/>
4.
5. </Scripts>
6. </asp:ScriptManager>
```

代码说明：第 1 行定义服务器脚本管理控件 ScriptManager1。第 2~4 行定义 Scripts 标签。第 3

行定义 ScriptReference 标签来指定引用的 Javascript 脚本文件并设置属性 Assembly 获得脚本函数库的名称、设置属性 Name 获得 Javascript 脚本文件。

Services 标签引用 Web Service 程序文件 (\*.asmx) 的语法为：

```
1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. <Services >
3. <asp:ScriptReference Path ="Web Service 程序的路径" />
4.
5. </Services>
6. </asp:ScriptManager>
```

以上代码，第 1 行定义服务器脚本管理控件 ScriptManager1。第 2~4 行定义 Services 标签。第 3 行定义 ScriptReference 标签来指定引用的 Web Service 程序并设置属性 Path 获得 Web Service 程序的路径。

另外，ScriptManager 还具有如表 14-3 所示的主要属性成员。

表 14-3 ScriptManager 的主要成员属性

属性	描述
AllowCustomError	和 Web.config 中的自定义错误配置区<customError>相联系，是否使用自定义的错误处理，默认值为 true
AsyncPostBackErrorMessage	获取或设置错误信息。当在一个异步回送过程中出现未处理的服务器异常时这个错误信息会被发送到客户端
AsyncPostBackTimeout	异步回送超时限制，默认值为 90，单位是秒
EnablePartialRendering	布尔值，可读写，当值为 True 时表示可使用 UpdatePanel 控件进行部分页面刷新，当值为 False 时表示不可以
ScriptMode	指定 ScriptManager 发送到客户端的脚本的模式，有四种模式：Auto、Inheit、Debug 和 Release,默认值为 Auto
ScriptPath	设置所有的脚本块的根目录，做为全局属性，包括自定义脚本块或者引用第三方的脚本块
OnAsyncPostBackError	异步回传发生异常时的事件，用于指定一个服务端的处理函数，在这里可以捕获异常信息并做相应处理
OnResolveScriptReference	指定 ResolveScriptReference 事件的服务器端处理函数，在该函数中可以修改某一条脚本的相关信息如路径、版本等

ScriptManager 控件可以管理为执行部分页面的控件创建的资源，这些资源包括脚本、样式、隐藏区域和数组。ScriptManager 控件包括脚本集合，在这个集合中包含了用于浏览器的脚本引用对象 ScriptReference，可以以声明或者编程的方式添加这些脚本，然后通过脚本引用来访问这些脚本。

ScriptManager 控件还包括了注册方法，使用这些方法可以以编程的方式来注册脚本和隐藏区域。

ScriptManager 控件的服务集合包括了 ServerReference 对象，ServerReference 对象绑定到每个注册到 ScriptManager 控件里 Web 服务。ASP.NET AJAX 框架为每个服务集合的 ServerReference 对象生成了一个代理对象，这些代理对象和它们提供的方法可以使客户端脚本调用 Web 服务变得简单。

也可以以编程方式把 `ServerReference` 对象注册到服务集合中，从而把 Web 服务注册到 `ScriptManager` 控件中，这样客户端就可以调用注册的 Web 服务。

另外需要关注一下 `ScriptMode`，指定 `ScriptManager` 发送到客户端的脚本的模式，有四种模式：`Auto`、`Inherit`、`Debug` 和 `Release`，默认值为 `Auto`。

- **Auto 模式：**会根据 Web 站点的 `Web.config` 配置文件来决定使用哪种模式，如果配置文件中的 `retail` 属性设置为 `true`，则把 `Release` 模式的脚本发送到客户端，反之，则发送 `Debug` 脚本。
- **Debug 模式：**若 `retail` 配置值不为 `true`，则发送 `Debug` 模式的客户端脚本。
- **Release 模式：**若 `retail` 配置值不为 `false`，则发送 `Release` 模式的客户端脚本。
- **Inherit 模式：**意义同默认值 `Auto` 的用法。

`ScriptManager` 控件处理程序发生异常时可以单独使用，其他情况下需要与别的 ASP.NET AJAX 服务器控件配合才能达到效果。



所有需要支持 ASP.NET AJAX 的页面上有且只能有一个 `ScriptManager` 控件，它必须放在其他元素的前面，最好接着 `form` 元素放置。

## 2. 调用 Web 服务

`ScriptManager` 的一个主要作用是在客户端注册一些服务器端的代码，最常用的就是将 Web Service 注册在客户端，这样就可以在 JavaScript 脚本中实现对 Web 服务的调用。要在 JavaScript 中调用 Web 服务需要经过 3 个步骤：

- 01 创建 Web 服务。
- 02 在客户端注册 Web 服务。
- 03 在 JavaScript 中引用服务的方法。

**【例 14-1】**本例演示使用 `ScriptManager` 控件在客户端调用 Web 服务，此 Web 服务实现一个四则运算的 calculators 的运算功能。由于是在客户端调用所以不能使用服务器控件而是使用 HTML 的标记控件。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 14-1”。

02 右键单击网站名称。选择“添加新项”菜单选项，弹出的“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表选中“Web 服务”，然后在“名称”文本框输入该文件的名称“`WebService.asmx`”，最后单击“添加”按钮。

03 双击网站根目录下的文件夹“`App_Code`”中的 `WebService.cs` 文件，在其中添加关键代码如下。

```
1. [System.Web.Script.Services.ScriptService]
2. [WebMethod]
3. public int GetTotal(string s,int x, int y){
4. if(s==""){
5. return x + y;
```

```

6. }
7. if(s == "-"){
8. return x - y;
9. }
10. if(s == "*"){
11. return x * y;
12. }
13. if(s == "/"){
14. return x / y;
15. }
16. else{
17. return 0;
18. }
19. }

```

代码说明：第 1 行如果要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务必须添加这一属性。第 2 行一个名为 WebMethod 的属性，该属性用来标志方法可以被远程的客户端访问。第 3 行定义一个 GetTotal 的，返回一个整型的值，它有三个参数，第一个参数是传递的运算符，第二个是第一个文本框中的数字，第三个是第二个文本框中的数字。第 4 行判断如果运算符是加号则第 5 行计算二数的和并返回该值。第 7 行判断如果运算符是减号则第 8 行计算二数的差并返回该值。第 10 行判断如果运算符是乘号则第 11 行计算二数的积并返回该值。第 13 行判断如果运算符是除号则第 14 行计算二数的商并返回该值。否则第 17 行返回 0 值。

**04** 用鼠标双击“Default.aspx”文件，进入到“视图设计器”，切换到“源视图”，在<form>和</form>标记间添加如下代码。

```

1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. <Services>
3. <asp:ServiceReference Path="~/WebService.asmx" />
4. </Services>
5. </asp:ScriptManager>
6. 请输入计算的两个数字

7. <input id="Text1" type="text" />
8. <select id="Select1" name="D1">
9. <option value="+" selected="selected">+</option>
10. <option value="-">-</option>
11. <option value="*">*</option>
12. <option value="/">/</option>
13. </select>
14. <input id="Text2" type="text" />
15. <input id="Button1" type="button" value="=" onclick="RefService()" style="height: 21px; width: 31px;" /><input
id="Text3" type="text" />
16.


```

代码说明：第 1 行定义一个服务器脚本管理控件 ScriptManager1。第 2~4 行定义 Services 标签。其中，第 3 行定义 ScriptReference 标签来指定引用的 Web Service 程序并设置属性 Path 获得 WebService.asmx 文件的路径。第 7 行定义了一个 HTML 文本框控件 Text1 获取用户输入的数字。第 8 行定义了一个 HTML 下列列表控件 select 用于显示运算符。第 9~12 行添加加减乘除 4 个列表选项。第 14 行定义一个 HTML 文本框控件 Text2 获取用户输入的第二个数字。第 15 行定义一个 HTML 按钮控件 Button1 设置显示的文本和设置该控件的单击事件为 RefService()。由于是 HTML 控件所以这个事件的方法不是在 aspx.cs 文件中进行处理。

**05** 在“源视图”的<head>和</head>标记之间编写 Javascript 脚本代码。

```

1. <script language="javascript" type="text/javascript">
2. function RefService() {
3. var num1 = document.getElementById("Text1").value;
4. var num2 = document.getElementById("Text2").value;
5. var num3 = document.getElementById("Select1").value;
6. WebService.GetTotal(num3,num1, num2, GetResult);
7. }
8. function GetResult(result) {
9. document.getElementById("Text3").value = result;
10. }
11. </script>

```

代码说明：第 1~10 行使用标记<script></script>定义标记之间的代码是 Javascript 脚本代码。第 2 行定义处理按钮单击事件的方法 RefService。第 3 和 4 行通过 document.getElementById 方法获取页面文本框用户输入的值。第 5 行获取页面下列列表框中用户选择的选项。第 6 行调用 WebService 服务中的定义的 GetTotal 方法，此方法比原来定义的多了一个参数，多的这个参数是下面 1 行我们自己定义的一个获得结果方法。第 8 行自定义一个获得结果的方法，它的参数 result 是 WebService 服务中 GetTotal 方法的一个 int 类型返回值也就是计算的值。第 8 行将值赋给文本框控件 Text3 的值 value 显示。

**06** 按快捷键“Ctrl+F5”，运行结果如图 14-4 所示。可以看到获得计算结果页面并没有进行刷新，这都要归功于 ScriptManager 控件的作用。

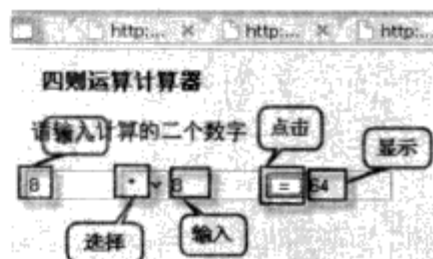


图 14-4 运行结果

### 3. ScriptManager 的异常处理

ScriptManager 控件可以让用户在浏览器端直接获得服务器产生的错误，并加以处理。当一般网页发生错误时，会在网页中显示服务器返回的错误信息，网页的浏览就中断了！最糟糕的是，有时错误信息中会附带重要的服务器信息，比如数据库在服务器中的路径。这样就等于无意中将一些重要的服务器信息告知大众，为网络黑客提供了绝佳的机会。所以在网站的开发中，对这个问题要加以重视。而 ScriptManager 控件可以很好地解决这个问题。下面通过一个实例来说明。

**【例 14-2】**本例演示使用 ScriptManager 控件处理自定义的异常，要实现的基本功能和上面“例 14-1”相同，不同的是做了异常处理。但请注意实现的过程与上例完全不同。因为不是通过客户端调用的方法来实现的，所以使用的都是服务器端的控件。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 14-2”。

**02** 用鼠标双击“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 3 个 TextBox 控件、1 个 Button 控件、1 个 ScriptManager 控件和 1 个 UpdatePanel 控件到“设计视图”中。切换到“源视图”，在<form>和</form>标记间添加如下代码。

```

1. <asp:ScriptManager ID="ScriptManager1" runat="server"
2. onasyncpostbackerror="ScriptManager1_ScriptManager1">
3. </asp:ScriptManager>
4. 请输入计算的数字

5. <asp:UpdatePanel ID="UpdatePanel1" runat="server">

```

```

6. <ContentTemplate >
7. <asp:TextBox ID="TextBox1" runat="server" Width="64px"></asp:TextBox>
8. <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True" Height="16px" Width="39px">
9. <asp:ListItem>+</asp:ListItem>
10. <asp:ListItem>-</asp:ListItem>
11. <asp:ListItem>*</asp:ListItem>
12. <asp:ListItem>/</asp:ListItem>
13. <asp:ListItem></asp:ListItem>
14. </asp:DropDownList>
15. <asp:TextBox ID="TextBox2" runat="server" Width="61px"></asp:TextBox>
16. <asp:Button ID="Button2" runat="server" Text="=" onclick="Button1_Click" style="height: 21px" Width="27px" />
17. <asp:TextBox ID="TextBox3" runat="server" Width="59px"></asp:TextBox>
18.

19. </ContentTemplate>
20. </asp:UpdatePanel>

```

代码说明：第 1 行定义服务器脚本管理控件 ScriptManager1。第 2 行设置该控件的异步回传错误处理事件为 ScriptManager1，这一部分最为重要。第 5~12 行定义一个服务器更新面板控件 UpdatePanel 用于实现局部刷新。但必须将控件放在第 6~11 行的<ContentTemplate >标记和</ContentTemplate >标记之间。第 7 行定义一个服务器文本框控件 TextBox1 接受用户输入的第一个数字。第 8 行定义一个服务器下拉列表控件 DropDownList1 并设置控件的大小和自动回传数据。第 9 行到 13 行添加加减乘除四个运算符作为列表选项。第 15 行定义一个服务器按钮控件 Button1 并设置单击事件为 Button1\_Click。第 17 行再定义一个服务器文本框控件接受用户的的输入的第二个数字。

**03** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. int a = Int32.Parse(TextBox1.Text);
3. int b = Int32.Parse(TextBox2.Text);
4. try{
5. if(DropDownList1.SelectedValue == "+")
6. TextBox3.Text = (a + b).ToString();
7. if (DropDownList1.SelectedValue == "-")
8. TextBox3.Text = (a - b).ToString();
9. if (DropDownList1.SelectedValue == "*")
10. TextBox3.Text = (a * b).ToString();
11. else if (DropDownList1.SelectedValue == "/")
12. TextBox3.Text = (a / b).ToString();
13. }
14. catch (Exception ex){
15. if (TextBox1.Text.Length > 0 && TextBox2.Text.Length > 0) {
16. ex.Data["ExtraInfo"] = " 不能将 " +
17. TextBox1.Text + "除以" + TextBox2.Text + ".";
18. }
19. throw ex;
20. }
21. }
22. protected void ScriptManager1_AsyncPostBackError(object sender, AsyncPostBackErrorMessageEventArgs e){
23. if (e.Exception.Data["ExtraInfo"] != null){
24. ScriptManager1.AsyncPostBackErrorMessage =
25. e.Exception.Message +
26. e.Exception.Data["ExtraInfo"].ToString();
27. }
28. else{
29. ScriptManager1.AsyncPostBackErrorMessage =

```

```

30. "未指定的错误发生。";
31. }
32. }

```

代码说明：第 1 行定义处理服务器按钮控件 Button1 单击事件 Click 的方法。第 2、3 行获取用户输入的两个文本框内的值。第 5 行判断如果下拉列表用户选择的是加法则第 6 行将二数计算的和显示在文本框中。第 7 行判断如果下拉列表用户选择的是减法则第 8 行将二数计算的差显示在文本框中。第 9 行判断如果下拉列表用户选择的是乘法，则第 10 行将二数计算的积显示在文本框中，否则第 12 行将二数计算的商显示在文本框中。第 14 行抛出异常。如果第 18 行用户输入的值都小于 0，则第 16、17 行自定义异常处理的信息的具体内容，不能将二数进行相除。

第 22 行定义处理服务器脚本管理控件 ScriptManager1 异步回传错误处理事件的 AsyncPostBackError 方法。第 23 行如果判断异常数据不为空，第 24 行将显示异步回传错误信息的具体内容，否则第 29 行将异步回传错误信息的内容显示为“未指定的错误发生。”。

**04** 按下“Ctrl+F5”，运行结果如图 14-5 所示。向每个文本框中添加一个大于零的数，然后单击“计算”按钮实现计算结果并实现局部刷新。

**05** 将第二个文本框的值输入为 0，然后单击“计算”按钮。浏览器显示一个如图 14-6 所示的消息框，该消息框包含在服务器代码中设置的消息。

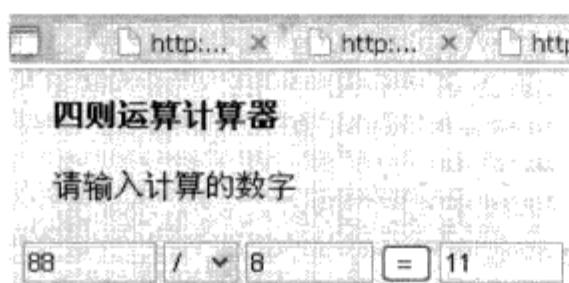


图 14-5 运行结果 1



图 14-6 运行结果 2

## 14.2.2 UpdatePanel 控件

局部更新是 ASP.NET AJAX 中最基本、最重要的技术。UpdatePanel 可以用来创建丰富的局部更新 Web 应用程序，其强大之处在于不用编写任何客户端脚本就可以自动实现局部更新。

### 1. UpdatePanel 的结构

UpdatePanel 控件是一个服务器控件，它能够帮助我们开发具有复杂客户端行为的 Web 页面，它能够使页面对终端用户更具有吸引力。协调服务器和客户端以更新一个页面的指定部位，通常需要具有很深的 ECMAScript (JavaScript) 知识。然而，使用 UpdatePanel 控件，可以让页面实现局部更新，而且不需要编写任何客户端脚本。此外，如果有必要的话，可以添加定制的客户端脚本以提高客户端的用户体验。当使用 UpdatePanel 控件时，页面上的行为具有浏览器独立性，并且能够潜在地减少客户端和服务器之间数据量的传输。

UpdatePanel 控件能够刷新指定的页面区域，而不是刷新整个页面。整个过程是由服务器控件 ScriptManager 和客户端类 PageRequestManager 来进行协调的。当部分页面更新被激活时，控件能够被异步地传递到服务器端。异步的传递行为就像通常的页面传递行为一样，所产生的服务器页面执行和控制页面生命周期。然而，随着一个异步的页面传递，页面更新局限于被 UpdatePanel 控件

包含和被标识为要更新的页面区域。服务器只为那些受到影响的浏览器元素返回 HTML 标记。在浏览器中，客户端类 PageRequestManager 执行文档对象模型（DOM）的操纵，以使用更新的标记来替换当前存在的 HTML 片段。

UpdatePanel 控件用来控制页面的局部更新，这些更新依赖于 ScriptManager 的 EnablePartialRendering 属性，如果此属性设置为 false，则局部更新将失去作用。一个 UpdatePanel 的定义可以包括如下部分。

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" ChildrenAsTriggers="true" UpdateMode="always"
RenderMode="Inline" >
 <ContentTemplate>
 <ContentTemplate>
 <Triggers>
 <asp:AsyncPostBackTrigger/>
 <asp:PostBackTrigger/>
 </Triggers>
</asp:UpdatePanel>
```

以上代码中 UpdatePanel 控件的各属性含义如表 14-4 所示。

表 14-4 UpdatePanel 主要属性

属性	描述
ChildrenAsTriggers	当属性 UpdateMode 为 Condition 时，UpdatePanel 中的子控件的异步传送是否引发 UpdatePanel 控件的更新
RenderMode	表示 UpdatePanel 控件最终呈现的 HTML 元素。其中值 Block 表示<div>，Inline 表示<span>
UpdateMode	表示 UpdatePanel 控件的更新模式。其中值 Always 是不管有没有 Trigger，其它控件都将更新更新该 UpdatePanel 控件，Conditional 表示只有当前 UpdatePanel 控件的 Trigger，或 ChildrenTriggers 属性为 true 时当前 UpdatePanel 控件中的控件引发的异步回送或整页回送，或是服务器端调用 Update()方法才回引发更新该 UpdatePanel 控件

对于 UpdatePanel 控件而言，有两个重要的子标签：

- （1）ContentTemplate 子标签：在 UpdatePanel 控件的 ContentTemplate 标签中，开发人员能够放置任何 ASP.NET 控件，这些控件在 ContentTemplate 标签中，就能够实现页面无刷新的更新操作。可以毫不夸张的说这个标签是 UpdatePanel 控件最重要的组成部分。
- （2）Triggers 子标签：表示局部更新的触发器，包括两种触发器：
- （3）AsyncPostBackTrigge 异步回传触发器：来指定某个控件的某个事件引发异步回传（asynchronous postback），即部分更新。属性有 ControlID 和 EventName。分别用来指定控件 ID 和控件事件，若没有明确指定 EventName 的值，则自动采用控件的默认值，比如 Button 按钮就是 Click 单击事件。把 ContorlID 设为 UpdatePanel 外部控件的 ID，可以使外部控件控制 UpdatePanel 的更新。
- （4）PostBackTrigge 不使用异步回传触发器：用来指定在 UpdatePanel 中的某个服务端控件，

它所引发的回送不使用异步回送，而仍然是传统的整页回送。



## 提示

默认情况下，任何在 UpdatePanel 控件中的回送控件都会引起异步回送和刷新面板的内容。当然，也可以配置页面上的其它控件以刷新一个 UpdatePanel 控件，可以通过为该控件定义一个触发器实现配置。触发器用来绑定哪个回送控件和事件引发面板的更新。当被指定的触发器控件的事件被触发（例如，Button 控件的 Click 事件）时，要更新的面板被刷新。

## 2. 局部更新示例

了解了 UpdatePanel 的结构，现在来演示一下 UpdatePanel 控件在网页的 AJAX 应用上带来的便利。

**【例 14-3】**本例要完成的功能是添加图书的信息~6 章“例 6-1”中创建的 BookStor 数据库的“BookInfo 表”中，同时更新 GridView 控件中的显示。添加完成后清空用户输入的数据。在页面上实现两个局部的更新：一个是更新 GridView 控件显示刚添加的数据。第二个是更新用户的输入控件 TextBox 清空页面的数据。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 14-3”。

**02** 执行菜单栏上的“视图”|“服务器资源管理器”命令，弹出“服务器资源管理器窗口”。右键单击“数据连接”，在菜单中选择“添加连接”的命令。

**03** 在弹出的“添加连接”对话框中单击“浏览”，选择第 6 章中上机题 1 创建好的“Shool.mdf”的数据库文件。然后单击“确定”按钮。

**04** 右键单击网站名称。选择“添加新项”菜单选项，弹出“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选“LINQ to SQL”，然后在“名称”文本框输入该文件的名称“DataClasses.dbml”，最后单击“添加”按钮。

**05** 双击生成的“DataClasses.dbml”文件，出现 LINQ to SQL 类的“对象关系设计器”界面。将服务器资源管理器中的“BookInfo 表”拖动到服务器资源管理器。

**06** 用鼠标双击网站目录中的“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 ScriptManager 控件、2 个 UpdatePanel 控件、1 个 GridView 控件、4 个 TextBox 控件和 1 个 Button 控件到“设计视图”中。切换到“源视图”，在<form>和</form>标记之间编辑关键代码如下。

```

1. <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
2. <asp:UpdatePanel ID="UpdatePanel1" runat="server">
3. <ContentTemplate>
4. <asp:GridView ID="GridView1" runat="server"></asp:GridView>
5. </ContentTemplate>
6. <Triggers>
7. <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
8. </Triggers>
9. </asp:UpdatePanel>

10. 请输入添加图书的信息
11. <asp:UpdatePanel ID="UpdatePanel2" runat="server">
12. <ContentTemplate>

```

```

13. 编号: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

14. 书名: <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

15. 作者: <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

16. 出版社: <asp:TextBox ID="TextBox4" runat="server" Height="19px" Width="128px"></asp:TextBox>

17. </ContentTemplate>
18. </asp:UpdatePanel>

19. <asp:Button ID="Button1" runat="server" Text="添加" onclick="Button1_Click" />

```

代码说明: 第 1 行添加一个脚本管理控件 ScriptManager1 对页面中的 AJAX 控件进行管理。第 2 行添加一个服务器更新面板控件 UpdatePanel1 用于页面的局部更新。第 4 行在 UpdatePanel1 的子标签 ContentTemplate 内添加一个服务器列表控件 GridView1。第 7 行在 UpdatePanel1 的子标签 Triggers 内通过 AsyncPostBackTrigger 属性绑定添加按钮和关联事件的名称,为按钮的单击事件实现异步更新。

第 11 行添加一个 UpdatePanel2 控件用于页面的第二处局部更新。第 13~16 行在 UpdatePanel2 的子标签 ContentTemplate 内定义了 4 个服务器文本框控件 TextBox 用于输入编号、书名、作者和出版社的数据信息。第 19 行添加一个服务器按钮控件 Button1 并设置文本和单击的事件。

**07** 在 GridView1 控件右上方有一个向右的黑色小三角,单击这个小按钮打开“GridView 任务”列表,选择“自动套用格式”。弹出“自动套用格式对话框”,在左边的选择架构列表中选中“苜蓿地”,最后单击“确定”按钮。

**08** 用鼠标双击网站根目录下的“Default.aspx.cs”文件,编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack{
3. Binder();
4. }
5. }
6. protected void Button1_Click(object sender, EventArgs e) {
7. DataClassesDataContext da = new DataClassesDataContext();
8. BookInfo book = new BookInfo();
9. book.ID = TextBox1.Text;
10. book.Name = TextBox2.Text;
11. book.Author = TextBox3.Text;
12. book.Press = TextBox4.Text;
13. da.BookInfo.InsertOnSubmit(book);
14. da.SubmitChanges();
15. Binder();
16. TextBox1.Text = "";
17. TextBox2.Text = "";
18. TextBox3.Text = "";
19. TextBox4.Text = "";
20. }
21. protected void Binder(){
22. DataClassesDataContext da = new DataClassesDataContext();
23. GridView1.DataSource = da.BookInfo.ToList();
24. GridView1.DataBind();
25. }

```

代码说明: 第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行判断如果当前加载的页面不是回传页面,则第 3 行调用 Binder 方法显示数据。

第 6 行处理定义按钮控件 Button1 单击事件 Click 的方法。第 7 行定义声明强类型

DataClasses1DataContext 的对象 da。第 8 行声明了一个 BookInfo 实体类的对象 book。第 9~12 行给 book 对象的 4 个属性赋值, 这些值是用户输入在文本框中的内容。第 13 行调用 InsertOnSubmit 方法向 LINQ to SQL Table (TEntity) 集合中插入该条数据。第 14 行调用方法 SubmitChanges 提交更改数据库数据。第 15 行调用 Binder 方法显示数据。第 16~19 行将 4 个文本框中的数据清空。

第 21 行定义绑定列表控件 GridView1 控件的方法 Binder。第 22 行定义声明强类型 DataClasses1DataContext 的对象 da。第 23 行通过 ToList 方法获取 BookInfo 表中所有的实体对象作为列表控件 GridView1 的数据源。第 24 行调用 GridView1 控件的 DataBind 方法绑定数据。

**09** 按下“Ctrl+F5”, 运行结果如图 14-7 所示。在各文本框中输入内容, 单击“添加”按钮。

**10** 在列表控件中显示了如图 14-8 所示的添加信息, 并且可以观察到 GridView 控件显示数据和文本框清空内容时, 都没有刷新页面。



图 14-7 运行结果 1

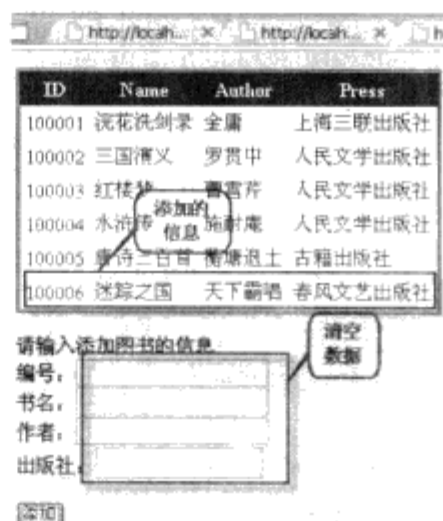


图 14-8 运行结果 2

### 14.2.3 UpdateProgress 控件

UpdateProgress 控件帮助开发人员设计一个直观的用户界面, 这个用户界面用来显示一个页面中的一个或多个 UpdatePanel 控件实现部分页面刷新的过程信息。如果一个部分页面刷新过程是缓慢的, 就可以利用 UpdateProgress 控件提供更新过程的可视化的状态信息。此外在一个页面可以使用多个 UpdateProgress 控件, 每个与不同的 UpdatePanel 控件相配合。此外, 可以使用一个 UpdateProgress 控件与页面上的所有 UpdatePanel 控件相配合。

UpdateProgress 的结构非常的简单, 下面是一个使用的例子。

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" AssociatedUpdatePanelID="UpdatePanel1">
 <ProgressTemplate>
 正在更新数据, 请等待。...
 </ProgressTemplate>
</asp:UpdateProgress>
```

以上代码中, UpdateProgress 控件的常用属性如表 14-5 所示。

表 14-5 UpdateProgress 控件的常用属性

属性	说明
AssociatedUpdatePanelID	获取或设置 UpdateProgress 控件显示其状态的 UpdatePanel 控件的 ID
DisplayAfter	获取或设置显示 UpdateProgress 控件之前所经过的时间值（以毫秒为单位）
DynamicLayout	获取或设置一个值，该值可确定是否动态呈现进度模板
ProgressTemplate	获取或设置定义 UpdateProgress 控件内容的模板
Visible	获取或设置一个值，该值指示服务器控件是否作为 UI 呈现在页上

属性 AssociatedUpdatePanelID 默认值为空字符串，也就是说 UpdateProgress 控件不与特定的 UpdatePanel 控件关联，因此，对于源于任何 UpdatePanel 控件的异步回送，或来自充当面板触发器的控件的回送，都会导致 UpdateProgress 控件显示其 ProgressTemplate 内容。此外，可以将 AssociatedUpdatePanelID 属性设置为同一命名容器、父命名容器或页中的控件。

属性 DynamicLayout 为布尔值，如果动态呈现进度模板，则为 true，否则为 false，默认值为 true。如果 DynamicLayout 属性为 true，则在首次呈现页时，不会为进度模板内容分配空间。但在显示内容时，就可以根据需要进行动态更改，包含进度模板的 div 元素的 style 属性将设置为 none。如果 DynamicLayout 属性为 false，则会在首次呈现页时，为进度模板内容分配空间，并且 UpdateProgress 控件是页面布局的物理组成部分，包含进度模板的 div 元素的 style 属性将设置为 block，并且其可视性最初会设置为 hidden。

属性 ProgressTemplate 默认值为 null。必须为 UpdateProgress 控件定义模板，否则，在 UpdateProgress 控件的 Init 事件发生期间会引发异常。可通过将标记添加到 ProgressTemplate 元素，以声明方式指定 ProgressTemplate 属性。如果 ProgressTemplate 元素中没有标记，则不会为 UpdateProgress 控件显示任何内容。如果要动态创建 UpdateProgress 控件，则可以创建一个从 ITemplate 控件继承的自定义模板。在 InstantiateIn 方法中指定标记，然后将动态创建的 UpdateProgress 控件的 ProgressTemplate 属性设置为自定义模板的新实例。如果要动态创建 UpdateProgress 控件，则应在页的 PreRender 事件发生期间或发生之前进行创建。如果在页生命周期晚期创建 UpdateProgress 控件，则不显示进度。

UpdateProgress 实际上是一个 div，通过代码控制 div 的显示或隐藏来实现更新提示。在 B/S 应用程序中，如果需要大量的数据交换，则必须使用 UpdateProgress，同时设计良好的等待界面，这样才能保证与用户的交互。



UpdateProgress 实际上是一个 div，通过代码控制 div 的显示或隐藏来实现更新提示。在 B/S 应用程序中，如果需要大量的数据交换，则必须使用 UpdateProgress，同时设计良好的等待界面，这样才能保证与用户的交互。

【例 14-4】使用 GridView 控件显示例 14-3 中“BookInfo 表”的数据，并允许分页显示。当用户翻页时实行局部刷新。此外在翻页时通过 UpdateProgress 控件给用户一个等待的提示图片。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 14-4”。

**02** 用鼠标双击网站目录中的“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 ScriptManager 控件、1 个 UpdatePanel 控件、1 个 GridView 控件、1 个 SqlDataSource 控件和一个过 UpdateProgress 控件到“设计视图”中。切换到“源视图”，在<form>和</form>标记之间编辑关键代码如下。

```

1. <asp:ScriptManager ID="ScriptManager1" runat="server">
2. </asp:ScriptManager>
3. <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
4. <asp:UpdatePanel ID="UpdatePanel1" runat="server">
5. <ContentTemplate>
6. <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
7. AutoGenerateColumns="False" DataKeyNames="ID" DataSourceID="SqlDataSource1" PageSize="5"
onpageindexchanged="GridView1_PageIndexChanged">
8. <Columns>
9. <asp:BoundField DataField="ID" HeaderText="ID" ReadOnly="True" SortExpression="ID" />
10. <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
11. <asp:BoundField DataField="Author" HeaderText="Author" SortExpression="Author" />
12. <asp:BoundField DataField="Press" HeaderText="Press" SortExpression="Press" />
13. </Columns>
14. </asp:GridView>
15. <asp:SqlDataSource ID="SqlDataSource1" runat="server"
16. ConnectionString="<%= $ConnectionStrings.BookStorConnectionString %>" SelectCommand="SELECT * FROM
[BookInfo]"></asp:SqlDataSource>
17. </ContentTemplate>
18. </asp:UpdatePanel>
19. <asp:UpdateProgress ID="UpdateProgress1" runat="server">
20. <ProgressTemplate>
21. 正在翻页，请等待.....
22. </ProgressTemplate>
23. </asp:UpdateProgress>

```

代码说明：第 1 行添加一个脚本管理控件 ScriptManager1 对页面中的 AJAX 控件进行管理。第 2 行添加一个服务器标签控件 Label1 用于显示当前系统的时间。第 4 行定义一个服务器更新面板控件 UpdatePanel。第 6 行在 UpdatePanel1 的子标签 ContentTemplate 内添加一个列表控件 GridView1 并设置其数据源为 SqlDataSource1、允许分页每页显示 5 条数据、禁止自动列和设置分页索引更改的事件。第 9 行定义绑定数据字段 ID 的图书编号数据列。第 10 行定义绑定数据字段 Name 的书名数据列。第 11 行定义绑定数据字段 Author 的作者数据列。第 12 行定义绑定数据字段 Press 的出版社数据列。第 15 行添加一个服务器数据源控件 SqlDataSource1。第 16 行设置该控件的数据库连接字符串和 sql 查询语句。第 19 行添加一个服务器更新进程控件 UpdateProgress1。第 21 行在 UpdateProgress1 的子标签 ProgressTemplate 中添加一个图片标记 img，并设置图片的路径和等待时提示的文字。

**03** 在 GridView1 控件右上方有一个向右的黑色小三角，单击这个小按钮，打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“苜蓿地”，最后单击“确定”按钮。

**04** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

protected void Page_Load(object sender, EventArgs e){
 Label1.Text = DateTime.Now.ToString();
}
protected void GridView1_PageIndexChanged(object sender, EventArgs {

```

```
System.Threading.Thread.Sleep(8000);
}
```

代码说明：第 1 行定义处理页面 Page 对象加载事件 Load 方法。第 2 行将当前系统是时间显示在标签控件 Label1 上。第 4 行定义处理列表控件 GridView1 页索引更改后事件 PageIndexChanged 的方法。第 5 行设置延时 8 秒钟，以观看 UpdateProgres1 的效果。

**05** 按下“Ctrl+F5”，运行结果如图 14-9 所示。单击列表下面的页码按钮，出现等待的图片和提示文字。

**06** 当提示翻页的信息消失，列表中显示如图 14-10 所示用户选择页码的 5 条图书信息。这里要注意的是两个运行结果的显示当前系统时间是相同的，这就说明了列表控件翻页并没有刷新页面。

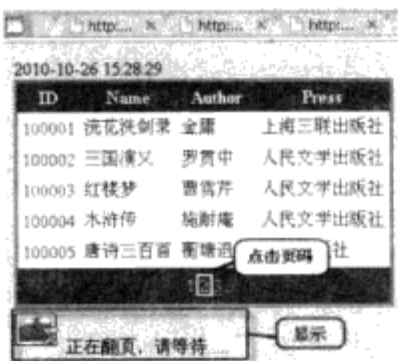


图 14-9 运行结果 1

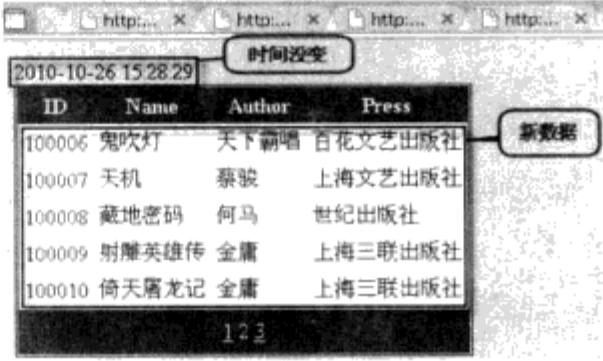


图 14-10 运行结果 2



UpdateProgress 的主要功能是当局部更新的内容比较多，时间上产生了延迟，为了让用户等待的过程中不至于太枯燥，通常使用呈现一些等待的 UI 或进度条。

14.2.4 Timer 控件

定时器控件 Timer 属于无人管理自动完成任务的一种特殊控件。Timer 控件的功能与大多数编程工具中提供的 Timer 一样都是按照特定的时间间隔执行指定的代码，ASP.NET AJAX 中的 Timer 也是如此。

Timer 的结构比较的简单，下面是一个使用的例子。

```
<asp:Timer ID="Timer1" runat="server" Interval="3000" Enabled="true" ontick="Timer1_Tick" Visible="true">
</asp:Timer>
```

Timer 控件的常用属性如表 14-6 所示。

表 14-6 Timer 控件的常用属性

属性	说明
Enabled	获取或设置一个值来指明 Timer 控件是否定时引发一个回送到服务器上，包含两个值：true 表示定时引发一个回送，false 则表示不引发回送
Interval	获取或设置定时引发一个回送的时间间隔，单位是毫秒。注意时间间隔要大于异步回送所消耗的时间，否则就会取消前一次异步刷新
Visible	获取或设置一个值，该值指示服务器控件是否作为 UI 呈现在页上

Timer 控件能够定时引发整个页面回送, 当它与 UpdatePanel 控件搭配使用时, 就可以定时引发异步回送, 并局部刷新 UpdatePanel 控件的内容。

Timer 控件可以用在下列场合。

- 定期更新一个或多个 UpdatePanel 控件的内容, 而且不需要刷新整个页面。
- 每当 Timer 控件引发回送时就运行服务器的代码。
- 定时同步地把整个页面发送到服务器。

Timer 控件是一个将 Javascript 组件绑定在 Web 页面中的服务器控件, 而这些 Javascript 组件在经过在 Interval 属性中定义的间隔后启动来自浏览器的回送。程序员可以在服务器上运行的代码中设置 Timer 控件的属性, 这些属性都会被传送给 Javascript 组件。

在使用 Timer 控件时, 页面中必须包含一个 ScriptManager 控件, 这是 ASP.NET AJAX 控件的基本要求。

当 Timer 控件启动一个回送时, Timer 控件在服务器端触发 Tick 事件, 可以为 Tick 事件创建一个处理程序来执行页面发送回服务器的请求。

设置 Interval 属性以指定回送发生的频率, 设置 Enabled 属性以开启或关闭 Timer。

如果不同的 UpdatePanel 必须以不同的时间间隔更新, 那么就可以在同一页面中包含多个 Timer 控件。另一种选择是, 单个 Timer 控件实例可以是同一页面中多个 UpdatePanel 控件的触发器。

此外, Timer 控件可以放在 UpdatePanel 控件内部, 也可以放在 UpdatePanel 控件外部。当 Timer 控件位于 UpdatePanel 控件内部时, 则 JavaScript 计时器组件只有在每一次回送完成时才会重新建立, 也就是说, 直到页面回送之前, 定时器间隔时间不会从头计算, 例如, 若 Timer 控件的 Interval 属性设置为 10 秒, 但是回送过程本身却花了 2 秒才完成, 这样下一次的回送将发生在前一次回送被引发之后的 12 秒。当 Timer 控件位于 UpdatePanel 控件之外时, 当回送正在处理时, JavaScript 计时器组件仍然会持续计时, 例如, 若 Timer 控件的 Interval 属性设置为 10 秒, 而回送过程本身花了 2 秒完成, 但下一次的回送仍将发生在前一次回送被引发之后的 10 秒, 也就是说用户在看到 UpdatePanel 控件的内容被更新 8 秒后, 又会看到 UpdatePanel 控件再度被刷新。



提示

Timer 控件只有一个独立的事件 Tick, 其他都从模板继承而来。当用其作为局部更新触发器时所应用的 EventName 属性多是 Tick。一个页面中只能有一个 Timer 控件的实例。

**【例 14-5】**在开发聊天室时, 如果不实现无刷新功能的话, 用户会感到界面一直在闪动。所以本例通过 Timer 控件实现无刷新模式的聊天室中信息的发送功能。

**01** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 14-5”。

**02** 用鼠标双击网站目录中的“Default.aspx”文件, 进入到“视图设计器”。从“工具箱”分别拖动 1 个 ScriptManager 控件、1 个 UpdatePanel 控件、1 个 Timer 控件、1 个 Label 控件、1 个 DropDownList 控件、1 个 TextBox 控件和 1 个 Button 控件到“设计视图”中。切换到“源视图”, 在<form>和</form>标记之间编辑关键代码如下。

```
1. <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
```

```

2. <asp:UpdatePanel ID="UpdatePanel1" runat="server">
3. <ContentTemplate>
4. <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
5. <asp:Timer ID="Timer1" runat="server" Interval="1000" ontick="Timer1_Tick">
6. </asp:Timer>
7. </ContentTemplate>
8. <Triggers>
9. <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click"></asp:AsyncPostBackTrigger>
10. </Triggers>
11. </asp:UpdatePanel>
12. 输入聊天信息: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox> 选择字体颜色:
13. <asp:DropDownList ID="DropDownList1" runat="server">
14. <asp:ListItem>Red</asp:ListItem>
15. <asp:ListItem>Green</asp:ListItem>
16. <asp:ListItem>Yellow</asp:ListItem>
17. </asp:DropDownList>
18. <asp:Button ID="Button1" runat="server" Text="发送" onclick="Button1_Click" />

```

代码说明：第 1 行定义一个服务器脚本管理控件 ScriptManager1。第 2 行定义一个服务器更新面板控件 UpdatePanel1。第 4 行和第 5 行在 UpdatePanel1 的子标签 ContentTemplate 中定义一个服务器标签控件 Label1 和一个服务器时间控件 Timer1。同时设置 Timer1 触发事件已经时间间隔为 1 秒。第 9 行在 UpdatePanel1 的子标签 Triggers 内通过 AsyncPostBackTrigger 属性绑定按钮 Button1 关联事件的名称，为按钮的单击事件实现异步更新。第 12 行定义一个服务器下拉列表控件 DropDownList1。第 14~16 行设置列表项为红绿蓝三种颜色。第 18 行定义一个服务器按钮控件 Button1 并设置显示的文本和触发的事件。

### 03 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. Application.Set("Msg", "" + Application["Msg"] + "
" + Request.UserHostName + "进入聊天室[" + DateTime.Now.ToString() + "]");
4. }
5. }
6. protected void Timer1_Tick(object sender, EventArgs e){
7. try{
8. Label1.Text = Application["Msg"].ToString();
9. }
10. catch (Exception ex){
11. Response.Write(ex.Message);
12. }
13. }
14. protected void Button1_Click(object sender, EventArgs e){
15. Application.Set("Msg", Application["Msg"] + "
" + Request.UserHostName + "说: " + TextBox1.Text + " [" + DateTime.Now.ToString() + "]");
16. }
17. }

```

代码说明：第 1 行定义处理页面 Page 对象加载事件 Load 方法。第 2 行判断如果当前加载的页面不是回传页面，则在第 3 行使用 Application 对象的 Set 方法将用户进入聊天室的本地 IP 地址和事件保存到 Application 集合中。第 6 行定义处理 Timer1 控件 Tick 事件的方法。第 8 行将 Application 集合中的数据在标签控件 Label1 上显示如果出现异常情况，则在第 11 行输出异常信息到页面。第 14 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 15 行使用 Application 对象

的 Set 方法将用户输入文本框的文字和选择文字的颜色以及本地 IP 地址的内容保存到 Application 集合中。

**04** 按下“Ctrl+F5”，运行结果如图 14-11 所示。在文本框中输入聊天信息，在下拉列表中选择文字显示的颜色，单击“发送”按钮。信息会添加到页面，同时实现页面无刷新。



图 14-11 运行结果

## 14.3 AJAX Control Toolkit

说到 ASP.NET AJAX 技术，就不能不提到 AJAX Control Toolkit 控件包。它是 CodePlex 开源社区与微软之间的一个联合项目，是建立在 ASP.NET AJAX 扩展之上，目前 AJAX Control Toolkit 已经成为 ASP.NET AJAX 所有可用的 Web 客户端组件中最大、最好的一个工具集。

### 14.3.1 AJAX Control Toolkit 简介

AJAX Control Toolkit 是由 CodePlex 开源社区和 Microsoft 共同开发的一个 ASP.NET AJAX 扩展控件包，其中包含了 30 多种基于 ASP.NET AJAX 的、提供某一专一功能的服务端控件。它可以在不重新载入整个页面的情况下实现最终更新页面或只刷新 Web 页中被更新的部分，并且它是一个免费的资源，任何程序开发人员都可以使用该资源。

AJAX Control Toolkit 构建在 ASP.NET 2.0 AJAX Extensions 之上，满足了三个需要。首先，它提供了一个组件集，使网站开发者可以直接使用，从而快速完成 Web 应用程序的开发而不用编写过多的代码；其次，它给那些希望写客户端代码的人提供了很好的范例；第三，它还能使最好的脚本开发者工作脱颖而出。总而言之，AJAX Control Toolkit 是一组功能强大的 Web 客户端工具集，能大大提高 Web 应用程序的开发效率及其质量。

Visual Studio 2010 本身并没有自带 AjaxControlToolkit 控件，必须下载安装后才能使用。下载地址是 <http://asp.net/ajax/downloads/default.aspx>，选择 AjaxControlToolkit-Framework3[1].5-NoSource.zip 文件下载。下载后解压缩文件，生成 AjaxControlExtender 文件夹和 SampleWebSite 文件夹。AjaxControlExtender 文件夹内是安装程序模板，安装程序文件名为 AjaxControlExtender.vsi。SampleWebSite 文件夹内是一个网站示例，包括所有的控件，你可以在 Visual Studio 2010 中打开该网站了解这些控件的功能和使用方法。

单击 AjaxControlExtender.vsi 文件，出现如图 14-12 所示的“安装程序”对话框。

选择所需的模板，单击“下一步”执行安装。安装完毕后，在如图 14-13 所示的 Visual Studio 2010 Web 项目的工具箱中，单击右键，选择“添加选项卡”。

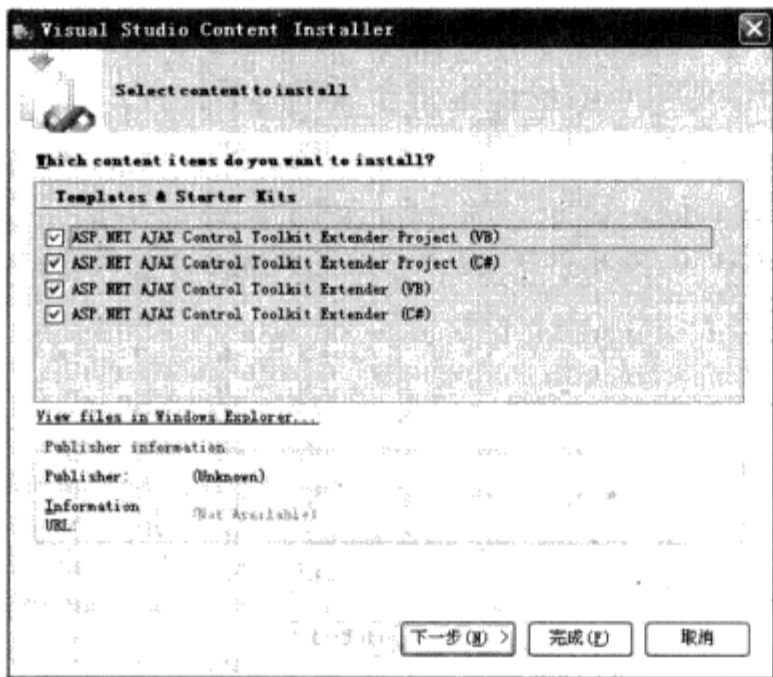


图 14-12 “安装程序”对话框

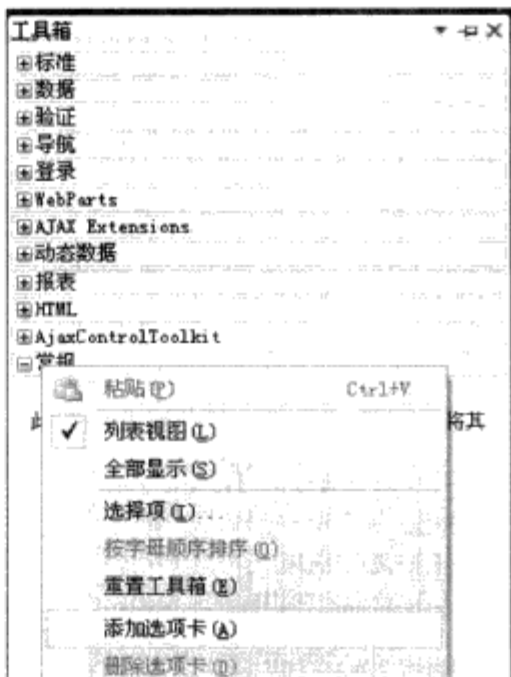


图 14-13 工具箱

在出现选项卡文本框中填写“AjaxControlToolkit”，按回车键。然后在创建好的 AjaxControlToolkit 的选项卡上单击右键，选择“选择项”，弹出如图 14-14 所示“选择工具箱项”对话框。在对话框中选中 Accordion 和 AccordionPane 选项，单击“确定”（如果在对话框中没有 Accordion 和 AccordionPane 这两个选项，则可以单击“浏览”按钮，在弹出的“打开”对话框中，选择 SampleWebSite 文件夹下 bin 目录中的 AjaxControlToolkit.dll 文件，单击“打开”按钮，就引入了 Accordion 和 AccordionPane 这两个选项）。

这样就将 SampleWebSite 文件夹下 bin 目录中的 AjaxControlToolkit.dll 的组件引入了工具箱。在如图 14-15 所示的 AjaxControlToolkit 选项卡下会出现 30 多个 AJAX 控件。

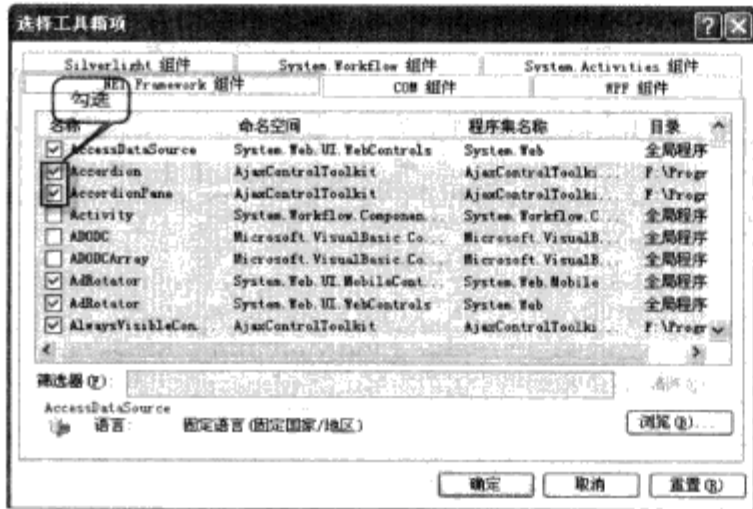


图 14-14 选择工具箱项对话框

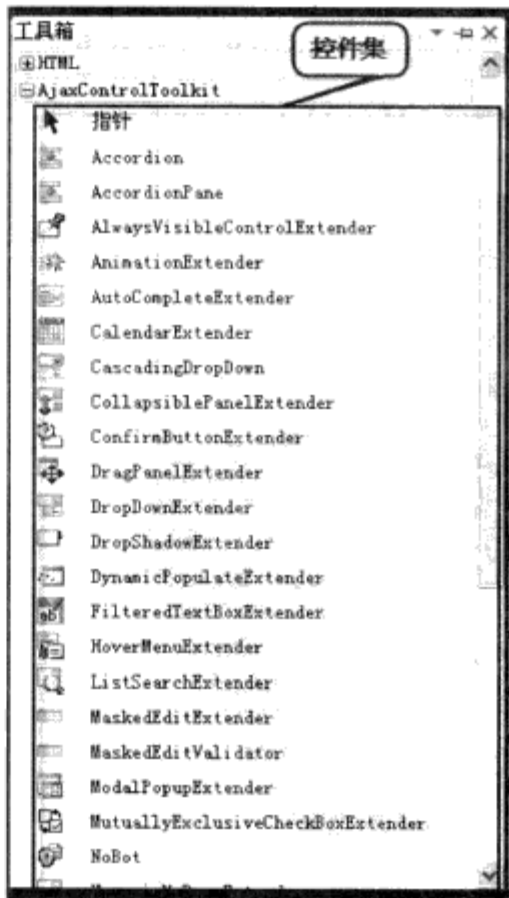


图 14-15 展开的 AjaxControlToolkit 选项卡

然后，我们就可以像使用工具箱中其他控件一样地使用这些 AjaxControlToolkit 控件。由于该控件集中控件较多，这里无法为大家一一介绍，所以下面仅介绍两个最常用的控件。



提示

在使用 AJAX Control Toolkit 的页面中，除了可以使用脚本控制器 ScriptManager 来管理 AJAX Control Toolkit 控件，也可以使用 ToolkitScriptManager 控件来进行管理。

### 14.3.2 CalendaeExtend 控件

在 Visual Studio 2010 的工具箱中，有一个常用的 Calendae 日历控件，但这个控件一直让用户感觉不好，因为在选择日期时会刷新页面。现在 AjaxControlToolkit 中针对了 TextBox 控件设计了一个 Calendae 的扩展控件 CalendaeExtend，使日历与 TextBox 控件完全结合，在 CalendaeExtend 中选择日期，会直接反应在 TextBox 控件上，不需要写任何的程序代码。CalendaeExtend 控件改进了 ASP.NET 日历控件的多项缺陷，比如原来日历在切换时，只能用月份转换，如果要输入距离现在很久的日期时，切换日历需要花费极多的时间，而在 CalendaeExtend 中可使用年份和月份切换，且采取了列表的方式显示，保证用户可以在任何日期中转换，同时能够实现选择日期时页面的无刷新，功能很强大。

CalendaeExtend 控件有三个常用而重要的属性。

- TargetControlID 属性用于设置关联的文本框控件编号，当用户单击关联的文本框时，日历会自动弹出，当选择好日期后，日历会自动的消失，所选的日期会显示在文本框中。
- Format 属性用于设置显示在关联文本框 TextBox 中日期的格式。
- CssClass 属性用于设置此日历控件的 Css 外观格式。

**【例 14-6】**使用 CalendarExtend 日历扩展控件配合 TextBox 控件实现酒店预订时无刷新的选择日期并显示选择结果。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 14-6”。

**02** 用鼠标双击网站目录中的“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 ScriptManager 控件、2 个 CalendarExtender 控件、2 个 TextBox 控件、1 个 Label 控件和 1 个 Button 控件到“设计视图”中。切换到“源视图”，在<form>和</form>标记之间编辑关键代码如下。

```

1. <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
2. <h4>酒店预订</h4>
3. <cc1:CalendarExtender ID="CalendarExtender1" runat="server" TargetControlID="TextBox1" Format="yyyy-MM-dd"
 CssClass="MyCalendar">
4. </cc1:CalendarExtender>
5. <cc1:CalendarExtender ID="CalendarExtender2" runat="server" TargetControlID="TextBox2"
 Format="yyyy-MM-dd">
6. </cc1:CalendarExtender>
7. 入住日期

8. <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
9.

10. 离店日期

11. <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

```

```

12. <asp:Button ID="Button1" runat="server" Text="提交"
13. onclick="Button1_Click" />

14. <asp:Label ID="Label1" runat="server" Text=""></asp:Label>

```

代码说明：第 1 行定义一个服务器脚本管理控件 ScriptManager1 管理页面中的 AJAX 控件。第 2 行定义显示标题文本。第 3 行定义一个服务器日历扩展控件 CalendarExtender1，设置其与文本框控件 TextBox1 关联、日期的显示格式为 yyyy-MM-dd 以及使用 MyCalendarCss 格式作为控件的样式。第 5 行定义一个服务器日历扩展控件 CalendarExtender2，设置其与文本框控件 TextBox2 关联、日期的显示格式为 yyyy-MM-dd。第 8 和 11 行各自定义一个服务器文本框控件 TextBox，分别用于用户选择的入住时间和离店时间的显示。第 12 行定义一个服务器按钮控件 Button1，并设置其显示的文字和触发的事件。第 14 行定义一个服务器标签控件用于显示用户选择日期的结果。

**03** 在“源视图”的<head>和</head>标记之间编辑 CSS 脚本文件，用于对 CalendarExtender1 控件的外观样式进行设置。

```

1. <style type="text/css">
2. .MyCalendar.ajax__calendar_container{
3. border: 1px solid #646464;
4. background-color: #faac38;
5. }
6. .MyCalendar.ajax__calendar_other.ajax__calendar_day, .MyCalendar.ajax__calendar_other.ajax__calendar_year{
7. color: #ffffff;
8. }
9. .MyCalendar.ajax__calendar_hover.ajax__calendar_day{
10. color: red;
11. background-color: #e8e8e8;
12. }
13. .MyCalendar.ajax__calendar_active.ajax__calendar_day{
14. color: blue;
15. font-weight: bolder;
16. background-color: #e8e8e8;
17. }
18. </style>

```

代码说明：第 1 行指明 style 标记内文本的类型是 CSS 样式表。第 2~5 行设置控件的边框和背景色。第 6~8 行设置控件上年份显示的颜色。第 9~12 行设置控件上日期的颜色和背景。第 13~17 行设置控件上日期被单击后的颜色，字重和背景。

**04** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Button1_Click(object sender, EventArgs e){
2. Label1.Text = "你选择的入住日期是：" + TextBox1.Text + "，离店日期是：" + TextBox2.Text + "。";
3. }

```

代码说明：第 1 行定义处理服务器按钮控件 Button1 单击事件 Click 的方法。第 2 行获取两个文本框中用户输入的入住日期和离店日期并显示在标签控件 Label1 上。

**05** 按下“Ctrl+F5”，运行结果如图 14-16 所示。通过两个日历扩展控件分别选择入住和离店的日期后，单击“提交”按钮。

**06** 最后页面显示如图 14-17 所示的提示信息。



图 14-16 运行结果 1

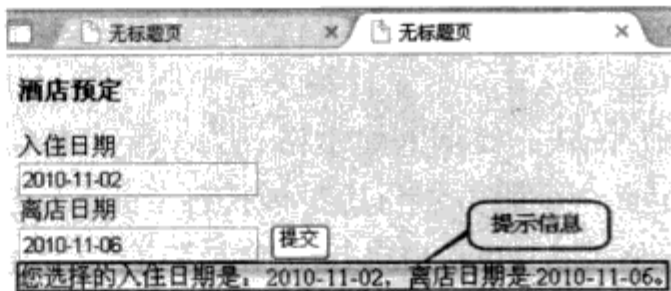


图 14-17 运行结果 2

14.3.3 SlideShowExtender 控件

在网站的首页中，通常会进行图片的滚动式的播放来美化网页的效果，以达到吸引用户对本网站进行关注的作用。以往要实现这样的图片播放系统，必须非相当大的力气才能做到，但现在借助于 AJAX CONTROL TOOLKIT 工具集中的 SlideShowExtender 控件，只要设置一些相关属性，就能在网页中拥有功能强大的图片播放系统。

SlideShowExtender 控件具备图片播放的常用功能：是否自动播放、图片显示的间隔时间、文字或图片形式的按钮、图片的文字说明等都可以由开发人员设置。

SlideShowExtender 控件的的常用属性如表 14-7 所示。

表 14-7 SlideShowExtender 控件的的常用属性

属性	描述
TargetControllID	设置显示图片的 Image 控件的名称
SlideShowServicePath	提供图片的 Web 服务的位置路径和文件名，如果使用的是页面方法，则不需要
SlideShowServiceMethod	设置用来调用 Web 服务的方法
AutoPlay	是否在页面加载完毕之后，自动播放图片
ImageDescriptionLabelID	用来显示目前所显示的图片说明的 Label 控件
NextButtonID	用来播放下一张图片的按钮控件 ID
StopButtonText	停止播放按钮上显示的文字
PlayButtonText	播放按钮上显示的文字
PreviousButtonID	用来播放上一张图片的按钮的控件 ID
Loop	设置是否允许循环播放
PlayInterval	自动播放时的播放间隔时间，单位是毫秒，默认值为 3000。也就是每隔 3 秒播放一张图片
PlayButtonID	开始和停止播放图片按钮控件的 ID

【例 14-7】利用 SlideShowExtender 控件和 Image 控件实现一个具备导航功能和自动播放图片功能的页面相册。

01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 14-7”。

**02** 右键单击网站名称。选择“添加新项”菜单选项，弹出的“添加新项对话框”，选择“已模板”下的“Visual C#”模板，并在模板文件列表中选中“Web 服务”，然后在“名称”文输入该文件的名称“PhotoService.asmx”，最后单击“添加”按钮。

**03** 双击网站根目录下的文件夹“App\_Code”中的 PhotoService.cs 文件，在其中添加关键代码如下。

```
1. using AjaxControlToolkit;
2. [System.Web.Script.Services.ScriptService()]
3. [WebMethod]
4. public Slide[] GetPhoto() {
5. return new AjaxControlToolkit.Slide[] {
6. new AjaxControlToolkit.Slide("Photo/1024f0304_10.jpg","雪山",string.Empty),
7. new AjaxControlToolkit.Slide("Photo/1024f0304_11.jpg","麦田",string.Empty),
8. new AjaxControlToolkit.Slide("Photo/1024f0304_23.jpg","色彩",string.Empty),
9. new AjaxControlToolkit.Slide("Photo/1024f0304_5.jpg","向阳",string.Empty),
10. new AjaxControlToolkit.Slide("Photo/1024f0304_8.jpg","高山深湖",string.Empty),
11. new AjaxControlToolkit.Slide("Photo/800f0304_47.jpg","风景",string.Empty)
12. };
13. }
```

代码说明：第 1 行引入 AjaxControlToolkit 命名空间。第 2 行添加服务器脚本。第 3 行添加方法属性。第 4 行定义一个获取图片集合的方法 GetPhoto。第 5 行返回一个 Slide 类型的数组。第 6~11 行分别输入化 6 张图片作为 Slide 类型的数组集合的元素内容。

**04** 在网站根目录下创建一个“Photo”文件夹，在其中添加6张风景图片。

**05** 用鼠标双击网站目录中的“Default.aspx”文件，进入到“视图设计器”。从“工具箱”拖动 1 个 ScriptManager 控件、1 个 UpdatePanel 控件、1 个 Image 控件、1 个 SlideShowExtender、2 个 Label 控件和 3 个 ImageButton 控件到“设计视图”中。切换到“源视图”，在<form>form>标记之间编辑关键代码如下。

[illegible]

```

24. </cc1:SlideShowExtender>
25. </ContentTemplate>
26. </asp:UpdatePanel>

```

代码说明：第 1 行添加一个脚本管理控件 `ScriptManager1` 对页面中的 AJAX 控件进行管理。第 2 行添加一个服务器更新面板控件 `UpdatePanel1` 用于页面的局部更新。第 4 行在 `UpdatePanel1` 的子标签 `ContentTemplate` 内定义一个 3 行 1 列的表格。第 6 行添加一个服务器图片控件 `Image1` 并设置显示图片的尺寸。第 11 和 12 行各自添加了一个服务器标签控件 `Label` 并设置其颜色或显示的文本。第 15~17 行分别添加一个服务器图片按钮控件 `ImageButton` 并设置显示图片的路径。第 20 行添加一个服务器 `SlideShowExtender` 并设置其关联的图片控件 `Image1`。第 21 行设置该控件自动播放、图片描述标签控件为“`lblShow`”、下一张按钮控件为“`btnNext`”并允许自动循环播放。第 22 行继续设置该控件上一张按钮控件为“`btnPrev`”、播放按钮控件为“`btnPlay`”、播放间隔时间为默认的 3 秒、播放按钮上的文字是“自动播放”。第 23 行设置 `SlideShowExtender` 控件 Web 服务的位置路径、调用 Web 服务中方法的名称。



图 14-18 运行结果

**06** 按下“`Ctrl+F5`”，运行结果如图 14-18 所示。单击相册下面的三种不同的按钮，可以显示不同的图片。

## 14.4 上机题

1. 使用 `ScriptManager` 控件在客户端调用 Web 服务。当用户在文本框中输入姓名后单击“获取 Web 服务”按钮，弹出欢迎用户的信息提示框，运行结果如图 14-19 所示。

2. 使用 ASP.NET AJAX 技术改进 Menu 导航控件每次选择后都要重新加载页面的缺陷。提示借助于 `UpdatePanel` 控件实现，运行结果如图 14-20 所示。



图 14-19 运行结果



图 14-20 运行结果

3. 通过 `Timer` 控件和 `UpdatePanel` 控件实现局部更新和定时获取 2011 年春节倒计时的时间程序，运行结果如图 14-21 所示。

4. 设计一个程序，当用户选择下列列表中某个学生的名字，在还没有从数据库中获得该学生的信息显示在列表控件 `GridView` 前，出现等待提示的图片和文字。获取数据完毕后，等待提示消失，数据显示在列表中。本题的数据库文件是第 6 章上机题 1 中创建的“`School.mdf`”，运行结果如图 14-22 和 14-23 所示。

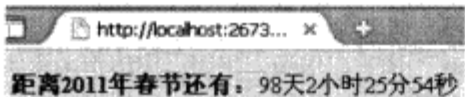


图 14-21 运行结果



图 14-22 运行结果 1



图 14-23 运行结果 2

5. 在本章中借助了 AjaxControlToolkit 控件集中的 CalendaeExtend 控件实现了日历选择日期的无刷新页面效果, 本题要求仅使用最普通的日历控件 Calendae 来实现同样的无刷新页面的效果, 运行程序后的界面如图 14-24 所示。

6. 使用 GridView 控件和 DetailView 控件实现数据库数据主细表在实际编程过程中是经常要用到的, 在没有学 AJAX 技术之前虽然能用但每次选择都会刷新页面, 给用户的体验不是很好, 所以本题要求使用本章学到的 ASP.NET.AJAX 技术实现不刷新页面的主细表, 数据库使用第 6 章中创建的 “BookStor.mdf” 文件, 运行结果如图 14-25 所示。



图 14-24 运行结果

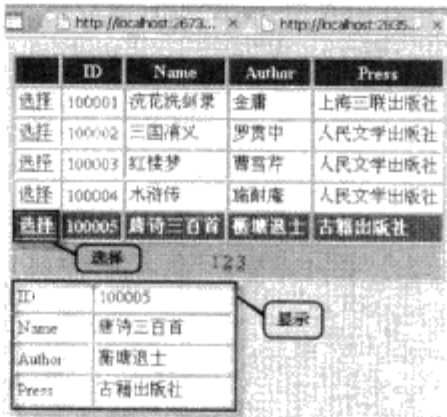


图 14-25 运行结果

# 第 15 章 Web 服务

## 学习目标

Web 服务即 Web Service,它是系统提供的一组接口,通过接口可以使用系统的功能。同在 Windows 系统中应用程序通过 API 接口函数使用系统提供的服务一样,在 Web 站点之间,如果想要使用其他站点的资源,就需要其他站点提供服务,这个服务就是 Web 服务。本章将从 Web Service 的基本概念、协议和应用这三个方面介绍目前这一流行技术以及如何基于 .NET 4.0 框架来实现 Web Service。

## 本章重点

- 理解 Web 服务的基本构成
- 掌握创建 Web 服务的方法
- 熟练调用网上的 Web 服务的资源
- 掌握对自定义 Web 服务的应用

## 15.1 Web Service 概述

Web 服务是从英文 Web Services 直接翻译过来的。很多编程人员初次接触 Web 服务,会认为这是一个新的系统架构和新的编程环境。其实,虽然 Web 服务是一个新的概念,但它的系统架构和实现技术却是完完全全继承已有技术的,绝对不会使现有的应用推倒重来,而是现有技术在 Internet 网络应用上的一个延伸和扩充。

### 15.1.1 Web Service 的概念

Web 服务其实就是一种无需购买并部署的组件,这种组件是被一次部署到 Internet 中,然后到处可以使用的一种新型组件,只要能够接入 Internet 网络,就可以使用和集成 Web 服务。Web 服务基于一套描述软件通信语法和语义的核心标准。XML 提供表示数据的通用语法,简单对象访问协议(SOAP)提供数据交换的语义,Web 服务描述语言(WSDL)提供描述 Web 服务功能的机制,其他规范统称为 WS-\*体系结构,用于定义 Web 服务发现、事件、附件、安全性、可靠的消息传送、事务和管理方面的功能。

再简单一点的说,Web 服务就是一种远程访问的标准。它的优点首先是跨平台,HTTP 和 SOAP 等已经是互联网上通用的协议;其次是可以解决防火墙的问题,如果使用 DCOM 或 CORBA 来访问 Web 组件,将会被挡在防火墙外面,而使用 SOAP 则不会有防火墙的问题。要发展 Web 服务需要更多的软件厂商来参与开发,让基于 Web Service 的软件服务数量和规模逐渐多起来。

虽然远程访问数据和应用程序逻辑不是一个新概念,但是以松耦合的方式执行这种操作却是一个全新的概念。以前的操作(例如 DCOM、IIOP 和 Java/RMI)要求在客户端和服务端之间进行紧密

集成，并要求使用特定的平台和二进制数据格式。虽然这些协议要求特定组件技术或对象调用约定，但现在的 Web 服务却不需要这样做。它在客户端和服务端之间所做的唯一假设就是接收方可以理解收到的消息。换句话说，客户端和服务端同意一个协定（在此所述的情况下，使用 WSDL 和 XSD），然后通过指定的传输协议（例如 HTTP）之上生成遵守该协定的消息来进行通信。因此，用任何语言编写、使用任何组件模型并在任何操作系统上运行的程序，都可以访问 Web 服务。此外，使用文本格式（如 XML）的灵活性使消息交换随时间的推移以一种松耦合的方式进行成为了可能。

我们来看一个实际应用的例子，大家都知道目前许多的网站，特别是门户或者是网址导航的网站都有提供各个城市的天气预报功能，如图 15-1 所示，可以通过定制省份和相应的城市来获取该城市的天气预报信息。事实上，这种天气预报并非该网站本身实现的功能，只是使用了互联网上其它提供天气预报网站的 Web 服务而已

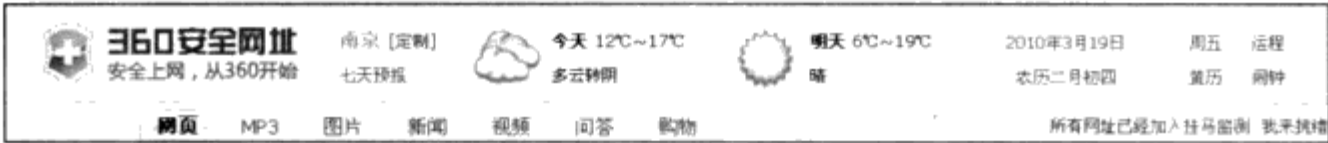


图 15-1 提供天气预报的网站

我们打开如图 15-2 所示这样一张页面，该页面的功能实现这样一种功能，在文本框中输入一个城市名称，然后单击“调用”按钮。

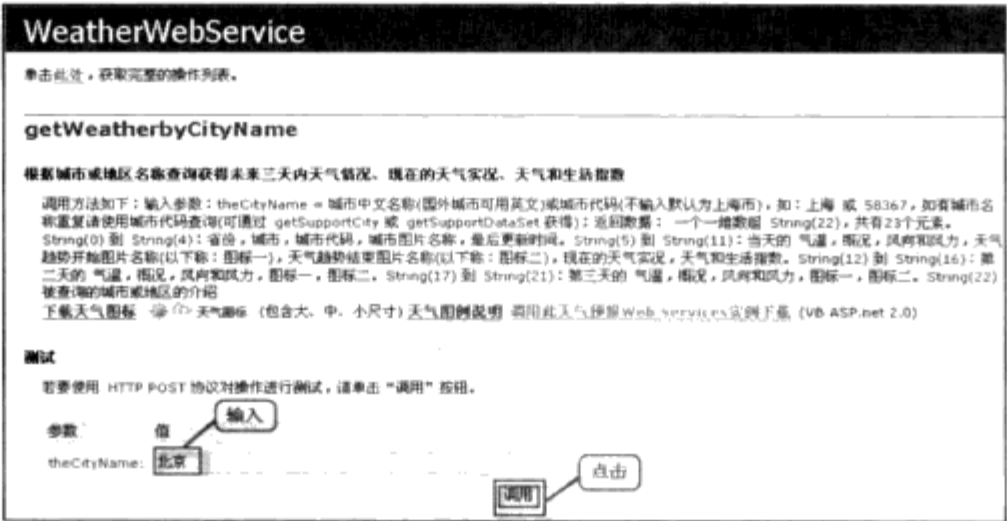


图 15-2 天气查询 Web 服务

浏览器的网页中显示如图 15-3 所示的一个 XML 文件，内容是用户所选城市的天气预报详细情况。

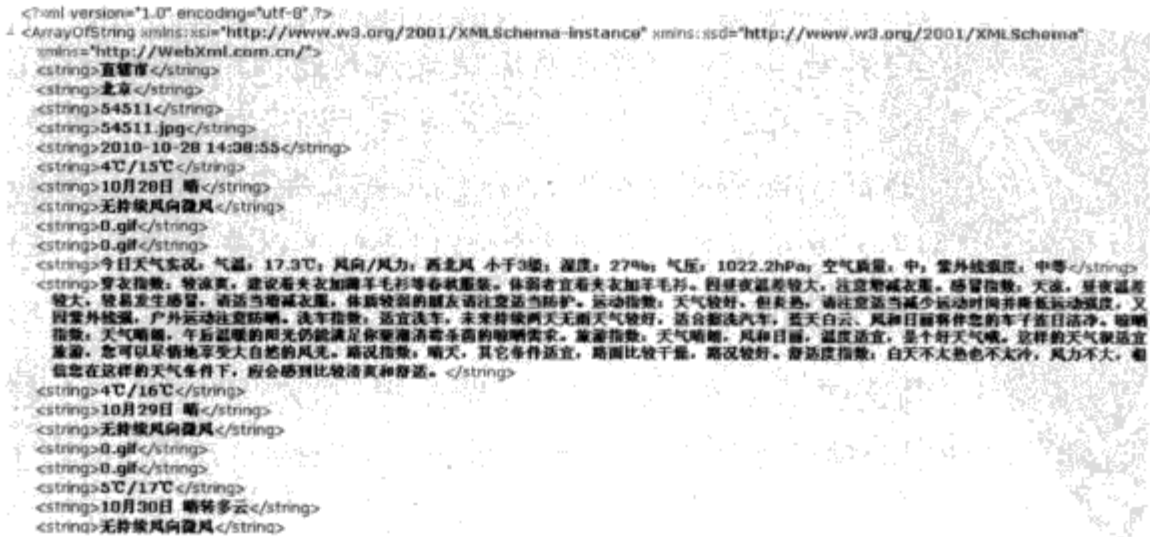


图 15-3 返回天气预报

现在大家都知道了，原来在互联网上还存在着这样一种提供信息的途径，接下来就是如何利用这些信息，以我们需要的形式运用于我们自己的程序中。用这种方式提供的信息不但可以应用于 Web 应用程序，还可以用于 Windows 应用程序。返回的信息采用了 XML 的格式，这样做的好处在于可以在不同的系统之间传递数据。

Web 服务就像组件一样，类似于一个封装了一定功能的黑匣子，用户可以重复用它而不用关心它是如何实现的。Web 服务提供了定义良好的接口，这些接口描述了它所提供的服务，用户可以通过这些接口来调用 Web 服务提供的功能。开发人员可以通过把远程服务、本地服务和用户代码结合在一起来创建应用程序。

Web 服务既可以在内部由单个应用程序使用，也可通过 Internet 公开供任意数量的应用程序使用。由于可以通过标准接口访问，因此 Web 服务使异构系统能够作为一个计算网络协同运行。

Web 服务并不追求一般的代码可移植性功能，而是为实现数据和系统的互操作性提供了一种可行的解决方案。Web 服务使用基于 XML 的消息处理作为基本的数据通信方式，以帮助消除使用不同组件模型、操作系统和编程语言的系统之间存在的差异。开发人员过去在创建分布式应用程序时通常使用组件，现在可以使用与此大致相同的方式来创建将各种 Web 服务组合在一起的应用程序。

Web 服务正在创造一个分布式应用程序开发的新时代。在使用专用基础结构将系统紧密耦合在一起时，是以牺牲应用程序互操作性为代价来实现的。Web 服务否定这种得不偿失的方式，并在全新的级别上提供一种互操作性。作为 Internet 的下一个革命性的进步，Web 服务将成为把所有计算设备连接到一起的基本结构。

### 15.1.2 Web Service 的基本构成

Web 服务在涉及到操作系统、对象模型和编程语言的选择时，不能带有任何的倾向性。要做到这一点必须使 Web 服务能够像其他基于 Web 的技术一样被广泛采用，所以它要求符合下列的前提条件。

- 松耦合：如果对两个系统的唯一要求是能彼此理解自我描述的文本消息，那么这两个系统就可以被认为是松耦合的。而紧耦合系统要求用大量自定义系统开销来进行通信以实现系统之间有更多的了解。
- 常见的通信：当今的计算机操作系统都是能够连接到 Internet 的，因此，需要提供常见的网络通信信道，并尽可能地具有能够将所有系统或设备连接到 Internet 的能力。
- 通用的数据格式：通过用现有的开放式标准而不是专用的通信方法，使任何支持同样开放式标准的系统都能够理解 Web 服务。同时，Web 服务在利用 XML 获得自我描述的文本消息时，它和客户端都不需要知道每个基础系统的构成就可以共享消息，这使得不同的系统之间的通信成为了一种可能。

Web 服务采用的基本结构提供了下列内容：定位 Web 服务的发现机制、定义如何使用这些服务的服务描述以及通信时使用的标准连网形式。Web 服务基本结构中的组件如表 15-1 所示。

表 15-1 Web 服务基本结构组件表

组件	角色
Web 服务目录	Web 服务目录（如 UDDI 注册表）用于定位其他组织提供的 Web 服务
Web 服务发现	Web 服务发现是定位（或发现）使用 Web 服务描述语言（WSDL）描述特定 Web 服务的一个或多个相关文档的过程。DISCO 规范定义定位服务描述的算法。如果 Web 服务客户端知道服务描述的位置，则可以跳过发现过程
Web 服务描述	要了解如何与特定的 Web 服务进行交互，需要提供定义该 Web 服务支持的交互功能的服务描述。Web 服务客户端必须知道如何与 Web 服务进行交互才可以使用该服务
Web 服务连网形式	为实现通用的通信，Web 服务使用开放式联网形式进行通信，这些格式是任何能够支持最常见的 Web 标准的系统都可以理解的协议。SOAP 是 Web 服务通信的主要协议

Web 服务的设计是基于兼容性很强的开放式标准。为了确保最大限度的兼容性和可扩展性，Web 服务体系被建设得尽可能通用，这意味着需要对用于向 Web 服务发送和获取信息的格式和编码进行一些假设。而所有这些细节都是以一个灵活的方式来界定，使用诸如 SOAP 和 WSDL 标准来定义。为了使客户端能够连接上 Web 服务，在后台有很多烦琐工作需要进行以便能够执行和解释 SOAP 和 WSDL 信息。这些烦琐工作会占用一些性能上的开销，但它不会影响一个设计良好的 Web 服务。表 15-2 列举了 Web 服务的标准。

表 15-2 Web 服务的标准

标准	说明
WSDL	告诉客户端一个 Web 服务里都提供了什么方法，这些方法包含什么参数以及将要返回什么值以及如何与这些方法进行交互
SOAP	在信息发送到一个 Web 服务之前，提供对信息进行编码的标准
HTTP	所有的 Web 服务交互发生时所遵循的协议，比如，SOAP 信息通过 HTTP 通道被发送
DISCO	该标准提供包含对 Web 服务的链接或以一种特殊的途径来提供 Web 服务的列表
UDDI	这个标准提供创建业务的信息，比如公司信息、提供的 Web 服务和用于 DISCO 或 WSDL 的相应的标准

Web 服务体系结构有三种角色：服务提供者（商）、服务注册和服务需求者，这三者之间的交互包括发布、查找和绑定等操作，其工作原理如图 15-4 所示。

服务提供者是服务的拥有者，它为用户提供服务功能。服务提供者首先要向服务注册中心注册自己的服务描述和访问接口（发布操作）。服务注册中心可以把服务提供者和服务请求者绑定在一起，提供服务发布和查询功能。服务请求者是 Web 服务功能的使用者，它首先向注册中心查找所需要的服务，注册服务中心根据服务请求者的请求把相关的 Web 服务和服务请求者进行绑定，这样服务请求者就可以从服务器提供者获得需要的服务。

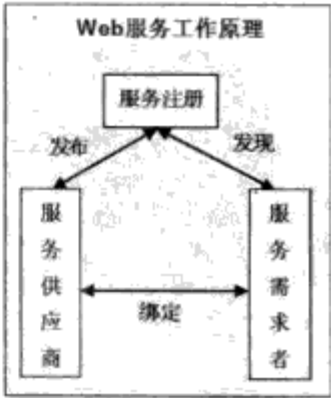


图 15-4 Web 服务工作原理



每个 Web 服务都需要一个唯一的命名空间。以便客户端应用程序可以将它与 Web 上的其他服务区分开。开发阶段用一个默认的命名空间是：`http://tempuri.org/`，在正式发布以前应该修改成可由 Web 服务提供者可以控制的命名空间。

### 15.1.3 实现一个基本的 Web 服务

上面介绍了 Web 服务的基本概念，本小节通过实现一个最简单的 Web 服务来加深对理论的理解。

#### 1. 创建 Web 服务

**【例 15-1】** 创建一个简单的 Web Service，在应用程序中通过调用创建的 Web 服务方法获得 Web 服务中具体内容并显示在网页中。

- 01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 15-1”。
- 02** 右键单击网站名称。在弹出如图 15-5 所示的快捷菜单中选择“添加新项”命令。

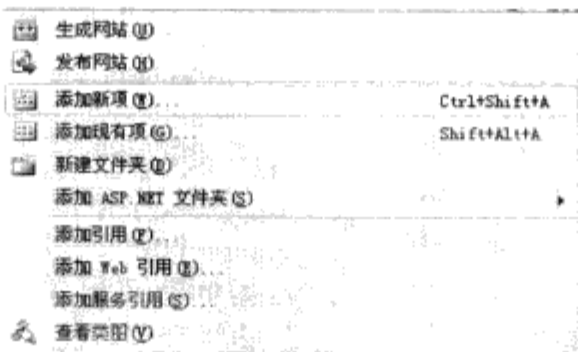


图 15-5 “添加新项”命令

**03** 弹出如图 15-6 所示的“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选“Web 服务”，然后在“名称”文本框输入该文件的名称“WebService.asmx”，最后单击“添加”按钮。

**04** 在解决方案资源管理器中出现如图 15-7 所示 Web 服务的文件。我们会发现多了两个文件：一个是 App\_Code 文件夹下的 Service.cs 的文件，另一个是 Service.asmx 文件。Service.asmx 就是刚才创建的 Web 服务文件，而 Service.cs 文件是该 Web 服务的后台代码文件，并且这个文件自动被放在了 App\_Code 文件夹中。

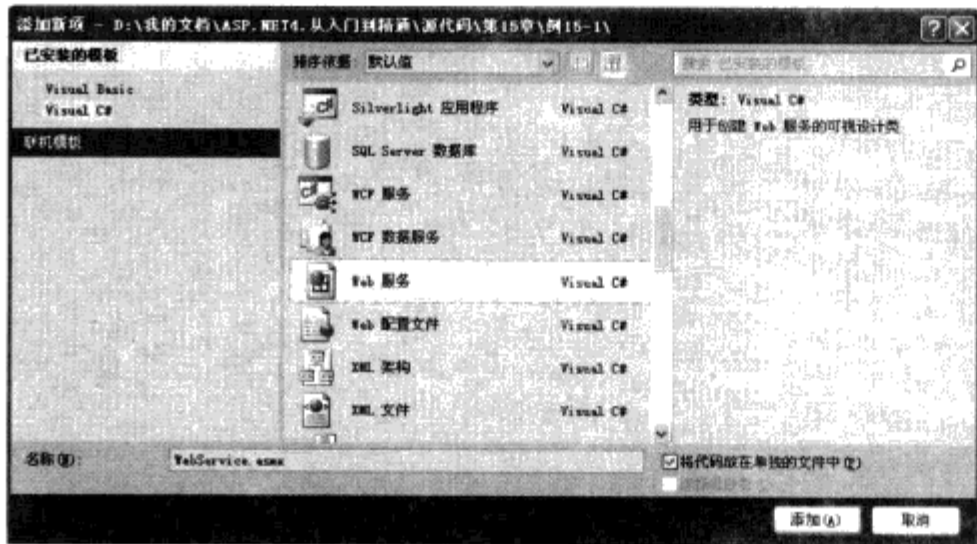


图 15-6 添加新项对话框



图 15-7 解决方案资源管理器

05 用鼠标双击进入“Service.cs”文件，文件中生成的代码如下所示。

```
1. [WebService(Namespace = "http://tempuri.org/")]
2. [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
3. //若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
4. // [System.Web.Script.Services.ScriptService]
5. public class Service : System.Web.Services.WebService {
6. public Service () {
7. //如果使用设计的组件，请取消注释以下行
8. //InitializeComponent();
9. }
10. [WebMethod]
11. public string HelloWorld() {
12. return "Hello World";
13. }
14. }
```

代码说明：第 1 行[WebService (Namespace = "http://tempuri.org/")]指出这个类是一个 Web 服务，并使用 Namespace 指出服务的唯一标示符即命名空间。第 2 行的[WebServiceBinding (ConformsTo = WsiProfiles.BasicProfile1\_1)]中 ConformsTo 属性指出了这个 Web 服务遵循的标准。第 5 行定义了一个名为 Service1 的类，该类继承于 System.Web.WebService，在 ASP.NET 中，所有的 Web 服务类都会继承 System.Web.WebService 类。该类包含一个构造函数，一般情况下可以不需要该构造函数。第 11 和 13 行还包含一个服务方法 HelloWorld，这其实是一段示例代码，告诉开发人员如何编写 Web 服务的方法。方法 HelloWorld 很简单，和一般类的方法没有什么区别。删除第 12 行代码，编写新的代码如下。

```
return "我的第一个 Web 服务程序！"
```

以上代码，返回一个字符串文本对象。

这里要注意的是 HelloWorld 方法上面第 10 行添加了一个名为 WebMethod 的属性，该属性用来标志方法可以被远程的客户端访问。WebMethod 包含 6 个属性用来提供描述它所标识的方法的接口，WebMethod 属性如表 15-3 所示。

表 15-3 WebMethod 的属性

属性	说明
Description	Web 服务的方法描述的信息。对 Web 服务的方法的功能注释，可以让调用者看见的注释
EnableSession	指示 Web 服务否启动 Session 标志，主要通过 Cookie 完成的，默认 false
MessageName	主要实现方法重载后的重命名
TransactionOption	指示 XML Web Services 方法的事务支持
CacheDuration	指定缓存时间的属性
BufferResponse	配置 Web 服务的方法是否等到响应被完全缓冲完，才发送信息给请求端

06 用鼠标双击打开“Service.asmx”文件，生成的代码如下所示。

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

在文件中只有一句代码。其中，“@WebService ”指令说明这是一个 Web 服务，Language

属性设置后台代码采用 C# 来编写,“CodeBehind”属性设置后台代码在程序中的目录地址。“Class”属性设置 Web 服务类的名字。

## 2. 测试 Web 中的操作

通过上面一节我们创建好了一个 Web 服务,接着我们将测试一下这个 Web 服务是否可用。

**01** 按下“Ctrl+F5”,运行结果如图 15-所示。该图和前文的图 15-8 很相似,页面都显示了服务的名称和所有的操作列表即服务的目录,单击图中仅有一个名为“Hello World”的操作。

**02** 弹出如图 15-9 所示的测试“Hello World”的操作的页面,单击“调用”按钮。

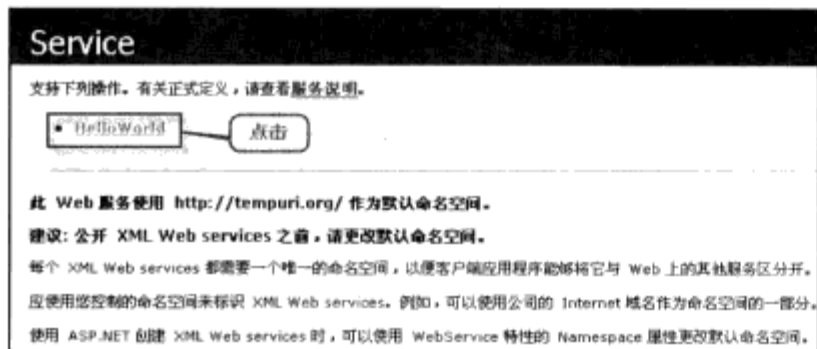


图 15-8 显示服务页面



图 15-9 Web 服务测试页面

**03** 显示该操作的结果,呈现一个包含如图 15-10 所示的 XML 文档信息的页面。返回了字符串“我的第一个 Web 服务程序! ”。这是通过 Service.cs 文件中定义的方法 HelloWorld 实现的。



图 15-10 获得操作结果

## 3. 引用和调用 Web 服务

至此一个完整的 Web 服务已经创建成功。接着需要把该服务添加到我们应用程序中。

**01** 右键单击网站根目录,在如图 15-11 所示的快捷菜单中选择“添加 Web 引用”菜单命令。

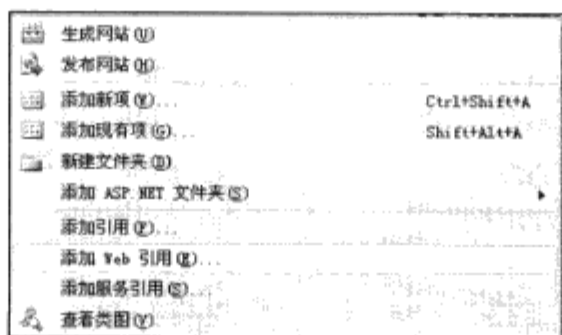


图 15-11 添加 Web 引用

**02** 打开如图 15-12 所示的“添加 Web 引用”对话框。其中,有三个选项链接:“此解决方案中的 Web 服务”选项用于添加创建在应用程序中的 Web 服务,“本地计算机上的 Web 服务”用于添加在本地机器中存在的 Web 服务,“浏览本地网络上的 UDDI 服务”用于添加在互联网中存在的 Web 服务。由于前面创建的“Service.asmx”是保存在我们的应用程序中,所以此处选择单击“此解决方案中的 Web 服务”。

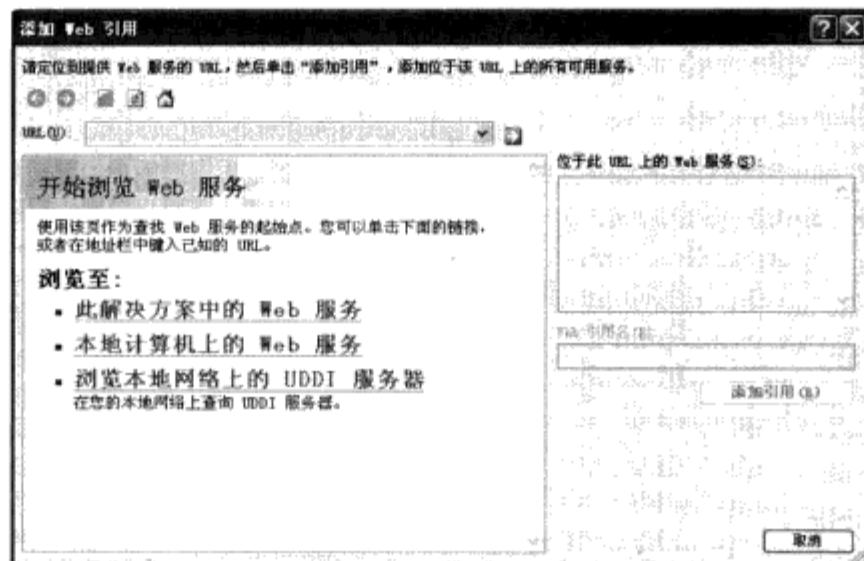


图 15-12 添加 Web 引用对话框

**03** 在“显示 Web 服务”的对话框中可以看到如图 15-13 所示的本解决方案中所有存在的 Web 服务，单击“WebService”服务名称。

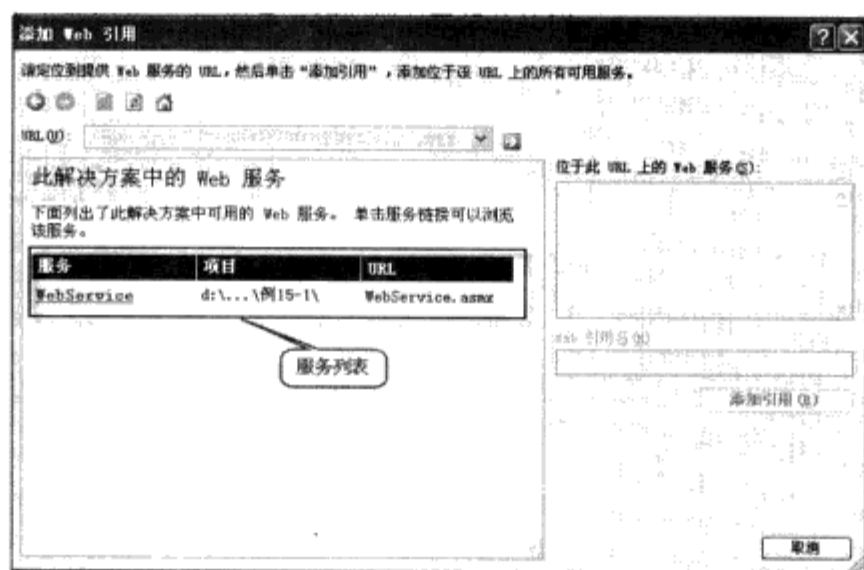


图 15-13 显示所有服务

**04** 打开如图 15-14 所示的显示操作目录的页面，可以看到 Web 服务所在的 Url 路径，Visual Studio 2010 能够根据这个路径找到这个 Web 服务。我们可以修改 Web 引用名为“localhost”，最后单击“添加引用”按钮。

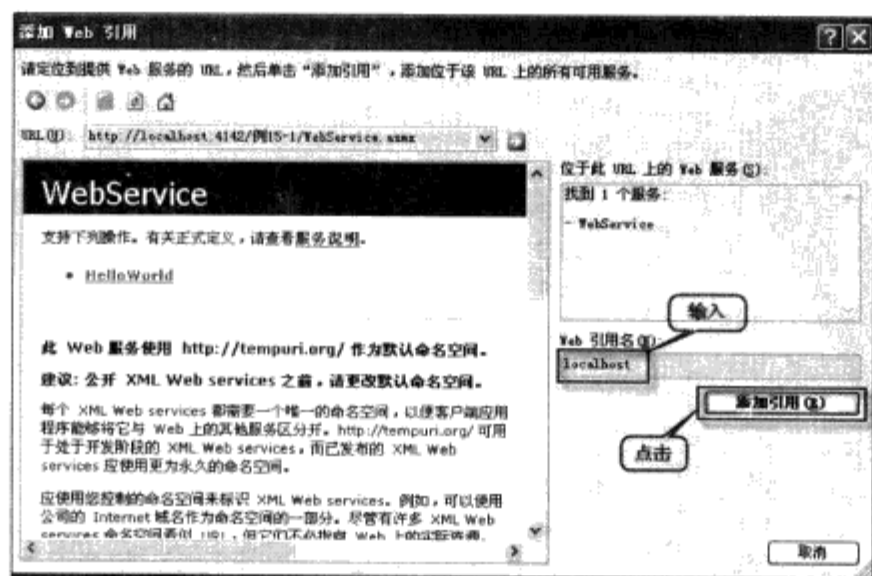


图 15-14 显示操作目录

**05** 这时网站文件目录结构发生了如图 15-15 所示的变化。在“解决方案资源管理器”中多了一个文件夹“App\_WebReferences”，其中还包含了一个子文件夹“MyWebService”，里面有三个文件，他们都以服务的名字为文件名，分别以“.disco”、“discomap”、“wsdl”为扩展名。这三个文件和前面我们介绍的 Web 服务标准相对应，在下面的章节会进行详细说明。



图 15-15 解决方案资源管理器

**06** 用鼠标双击“Default.aspx”文件，进入到“视图设计器”从“工具箱”中拖动 1 个 Label 控件到“设计视图”中，切换到“源视图”，编写代码如下。

```
<asp:Label ID="Label1" runat="server" Text=""></asp:Label>
```

以上代码定义了一个服务器标签控件 Label1。

**07** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，在窗体加载事件 Page\_Load 事件中添加如下代码。

```
localhost.WebService lw = new localhost.WebService();
Label1.Text = lw.HelloWorld();
```

代码说明：第 1 行先实例化 Web 服务 WebService 对象 lw，然后在第 2 行通过调用服务中的 HelloWorld 方法就能实现使用 Web 服务的功能。

**08** 按下“Ctrl+F5”，运行结果如图 15-16 所示。

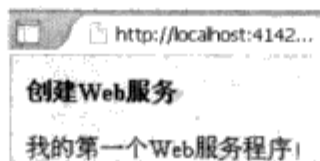


图 15-16 程序运行结果



提示

Web 服务中的命名空间和程序代码中的命名空间不是一回事，千万不可混为一谈。

## 15.2 Web 服务协议

通过上面的一个例子，我们初步了解了 Web 服务的作用和使用方法，之所以能够在应用程序中实现 Web 服务是因为 Web 服务协议的存在。在 Web 服务体系结构中主要包括以下三个核心服务，分别表示了三种 Web 服务协议：

- SOAP (简单对象访问协议): 用于数据传输。
- WSDL (Web 服务描述语言): 用于描述服务。
- UDDI (统一描述、发现和集成协议): 用于获取可用的服务。

下面分别这三个核心服务进行介绍。

### 15.2.1 WSDL (Web 服务描述语言)

还记得在上一节图 15-15 中的三个文件, 他们都以服务的名字为文件名, 分别以 “.disco”、“discomap”、“wsdl” 为扩展名。其中的 disco 文件能够发现每个 Web 服务的功能 (通过文档), 以及如何与它们进行交互 (通过 WSDL)。该文件是 Visual Studio 2010 在 “添加 Web 引用” 时自动生成的。让我们来看一下这个文档的内容, 它是一个 XML 文档, 只包含了该 Web 服务链接到其他资源的地址, 代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/disco/">
 <contractRef ref="http://localhost:1856/Sample/Service.asmx?wsdl" docRef="http://localhost:1856/Sample/Service.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
 <soap address="http://localhost:1856/Sample/Service.asmx" xmlns:q1="http://tempuri.org/" binding="q1:ServiceSoap"
xmlns="http://schemas.xmlsoap.org/disco/soap/" />
 <soap address="http://localhost:1856/Sample/Service.asmx" xmlns:q2="http://tempuri.org/" binding="q2:ServiceSoap12"
xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

以上代码中, <discovery> 元素指出了它对其他资源的应用。<contractRef> 元素的 ref 属性指向了 Web 服务的 WSDL 文档, 是用来描述这个服务的。根据 disco 文件我们获得了 WSDL 文档。

WSDL 是一个基于 XML 的标准, 它指定客户端如何与 Web 服务进行交互, 包括诸如一条信息中的参数和返回值如何被编码, 以及在互联网上传输时应该使用何种协议等等。目前, 有三种标准支持实际的 Web 服务信息的传送: HTTP GET、HTTP POST 和 SOAP。

读者可以在 <http://www.w3.org/TR/wsdl> 看到完整的 WSDL 标准。这个标准相当复杂, 但是这个标准背后的逻辑, 对于进行 ASP.NET 开发的编程人员来说是隐藏的, 这就像 ASP.NET 的 Web 控件抽象行为被封装一样。开发人员不需要知道这个标准具体的逻辑关系, 只需要知道如何使用这个标准即可, 把那些复杂逻辑行为留给系统和框架来解释执行。ASP.NET 可以创建一个基于 WSDL 文档的代理类。这个代理类允许客户端调用 Web 服务, 而不用担心网络或格式的问题。很多非 .NET 平台提供了相似的工具来完成同样的事务, 例如 VB 6.0 和 C++ 程序员也可以使用 SOAP 工具包。

WSDL 是一种规范, 它定义了如何用共同的 XML 语法描述 Web 服务。WSDL 描述了四种关键的数据:

- 描述所有公用函数的接口信息;
- 所有消息请求和消息响应的数据类型信息;
- 所使用的传输协议的绑定信息;
- 用来定位指定服务的地址信息。

总之, WSDL 在服务请求者和服务提供者之间提供一个协议。WSDL 独立于平台和语言, 主

要用于描述 SOAP 服务。客户端可以用 WSDL 找到 Web 服务，并调用其任何公用函数。还能够使用可识别 WSDL 的工具自动完成这个过程，使应用程序只需少量甚至不需手工编码就可以容易地连接新服务。WSDL 为描述服务提供了一种共同的语言，并为自动连接服务提供了一个平台，因此，它是 Web 服务结构中的基石。

WSDL 是描述 Web 服务的 XML 语法，这个规范本身分为六个主要的元素：

#### (1) definitions

definitions 元素必须是所有 WSDL 文档的根元素。它定义 Web 服务的名称，声明文档其他部分使用的多个名称空间，并包含这里描述的所有服务元素。

#### (2) types

types 元素描述在客户端和服务端之间使用的所有数据类型。虽然 WSDL 没有专门被绑定到某个特定的类型系统上，但它以 XML Schema 规范作为其默认的选择。如果服务只用到诸如字符串型或整型等 XML Schema 内置的简单类型，它就不需要 types 元素。

#### (3) message

message 元素描述一个单向消息，无论是单一的消息请求还是单一的消息响应，它都描述进行。message 元素定义消息名称，它可以包含零个或更多的引用消息参数或消息返回值的消息 part 元素。

#### (4) portType

portType 元素结合多个 message 元素，形成一个完整的单向或往返操作。一个 portType 可以定义多个操作。

#### (5) binding

binding 元素描述了在 Internet 上实现服务的具体细节。WSDL 包含定义 SOAP 服务的内置扩展，因此，SOAP 特有的信息会转到这里。

#### (6) service

service 元素定义调用指定服务的地址。一般包含调用 SOAP 服务的 URL。

#### (7) documentation

documentation 元素用于提供一个可阅读的文档，可以将它包含在任何其他 WSDL 元素中。

除了上述主要的元素，WSDL 规范还定义了其它实用元素，但是没有以上这些元素用的多，这里就省略介绍了。

WSDL 文件中最重要的一部分也就是对类型的定义。这一部分使用 XML 模式去描述数据交换的格式，数据交换的格式要通过使用 XML 元素和元素之间的关系来定义。这些主要元素的关系如图 15-17 所示。

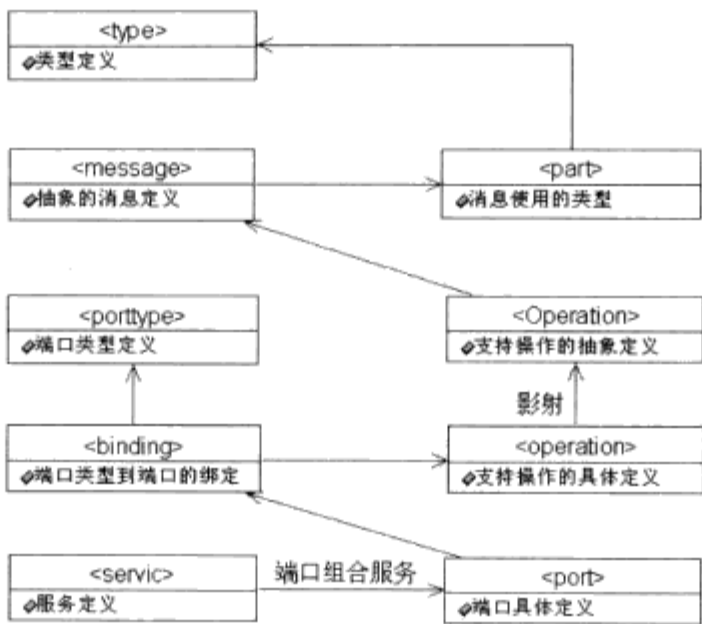


图 15-17 WSDL 元素关系

要查看 WSDL 文档的内容只需在如图 15-14 所示的页面中单击“服务说明”的链接，就能进入文档页面如图 12-18 所示。

```
<?xml version="1.0" encoding="utf-8" ?>
<wsc:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://tempuri.org/"
 xmlns:wsc="http://schemas.xmlsoap.org/wsdl/">
 <wsc:types>
 <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
 <s:element name="HelloWorld">
 <s:complexType/>
 </s:element>
 <s:element name="HelloWorldResponse">
 <s:complexType>
 <s:sequence>
 <s:element minOccurs="0" maxOccurs="1" name="HelloWorldResult"
 type="s:string"/>
 </s:sequence>
 </s:complexType>
 </s:element>
 </s:schema>
 </wsc:types>
 <wsc:message name="HelloWorldSoapIn">
 <wsc:part name="parameters" element="tns:HelloWorld"/>
 </wsc:message>
```

图 15-18 WSDL 文档部分内容



Web 服务最强大的特性是使用 XML 支持的跨平台的兼容性。只要能连接到 Internet，访问给定的 Web 服务都一样的容易。

15.2.2 SOAP（简单对象访问协议）

在 SOAP 出现之前，在 .NET 框架中，客户端与 Web 服务交互时有两种协议可以使用。

- HTTP GET: 使用该协议与 Web 服务交互时，会把客户端发送的信息编码，然后放在查询字符串里，而客户端获取的 Web 服务的信息则是以一个基本的 XML 文档的形式存在。
- HTTP POST: 使用该协议与 Web 服务交互时，会把参数放在请求体里面，而获取的信息则是以一个基本的 XML 文档的形式存在。

但是,随着信息的丰富化,我们需要传输的数据往往是结构化的,这样就出现了简单对象访问协议 SOAP (Simple Object Access Protocol),这是一种轻量的、简单的、基于 XML 的协议,它被设计成在 Web 上交换结构化的和固化的信息。SOAP 可以和现存的许多因特网协议和格式结合使用。包括超文本传输协议 (HTTP)、简单邮件传输协议 (SMTP) 和多用途网际邮件扩充协议 (MIME)。HTTP 是 SOAP 消息反复发送的结果,它好比一个邮递员拿着 SOAP 信封去目的地一样。SOAP 消息基本上是从发送端到接收端的单向传输,但它们常常结合起来执行类似于请求/应答的模式。SOAP 使用基于 XML 的数据结构和超文本传输协议 (HTTP) 的组合,定义了一个标准的方法来使用 Internet 上各种不同操作环境中的分布式对象。

SOAP 在 Web 服务的技术层次中起到的作用是:作为对应用共享的消息进行包装的标准协议。SOAP 规范定义了简单的基于 XML 包装传递信息和将与平台相关的应用数据类型转化成 XML 表示的一些规则。SOAP 的设计非常适合处理多种应用消息传递和集成模式。这一点是 SOAP 使用非常普遍的主要原因。

SOAP 规范主要定义了三个部分:

#### (1) SOAP 信封规范

SOAP XML 信封 (SOAP XML Envelope) 对在计算机间传递的数据如何封装定义了具体的规则。这包括应用特定的数据,如要调用的方法名、方法参数或返回值,还包括谁将处理封装内容,失败时如何编码错误消息等信息。

#### (2) 数据编码规则

为了交换数据,计算机必须在编码特定数据类型的规则上达成一致。SOAP 必须有一套自己的编码数据类型的约定。大部分约定都基于 W3C XML Schema 规范。

#### (3) RPC 协定

SOAP 能用于单向和双向等各种消息收发系统。SOAP 为双向消息收发定义了一个简单的协定来进行远程过程调用和响应,这使得客户端应用可以指定远程方法名,获取任意多个参数并接收来自服务器的响应。

关于 SOAP 标准的更多、更详细的信息,读者可以到 <http://www.w3.org/TR/SOAP> 阅读全部的规范。



提示

应用程序不能直接处理 SOAP 信息。相反,在应用程序使用数据之前,.NET 会把 SOAP 信息转换成相应的 .NET 数据类型。也就是说允许应用程序使用与其他对象一样的交互方式来同 Web 服务交互。

### 15.2.3 UDDI (统一描述、发现和集成协议)

UDDI (Universal Description Discovery and Integration) 是 Web 服务家族中最新和发展最快的标准之一。它最初被设计出来的目的是能够让开发人员非常容易地定位到任何服务器上的 Web 服务。

要定位 Web 服务,客户端必须知道特定的 URL 位置。通过发现文件把不同的 Web 服务放到一个文件中可以让这一过程变得相对的容易一些。但是它并没有提供任何明显的方法来检测一个公

司提供的 Web 服务。UDDI 的目的是：提供一个库，在这个库中商业公司可以为他们所拥有的 Web 服务做广告。比如，一个公司可能列出所有用于业务文件交换的服务，这些业务文件交换服务具有提交购买定单和跟踪获取的信息等功能。但为了能让客户端获取这些 Web 服务，这些 Web 服务必须注册在 UDDI 库中。

对于 Web 服务，UDDI 就相当于一个搜索引擎，就像互联网上的 Google。但 UDDI 却也有很大的不同，大部分搜索引擎试图搜索整个互联网，而为所有的 Web 服务建立一个 UDDI 注册却不需要达到那样的程度，因为不同的工业有着不同的需要，并且一个非组织的搜集并不能让所有人满意的。相反，它更像是公司的组织和联盟把他们这个领域的 UDDI 的注册绑定在一起。

有趣的是，UDDI 注册定义了一个完全编程接口，这个接口说明了 SOAP 信息能够被用来获取一个商务信息或为一个商务注册 Web 服务。换句话说，UDDI 注册本身就是一个 Web 服务！这个标准还没有被推广使用，但是读者在 <http://uddi.microsoft.com> 可以找到详细的说明。

## 15.3 Web 服务的应用

Web 服务目前使用得非常广泛，我们可以使用网络上已经存在的 Web 服务，也可以从数据库服务器中提取数据供 Web 服务使用者使用。更可以应用于企业内部的局域网，内部员工可以任意调用公司提供的服务资源，简化自己的工作。本节通过几个代表性的实例介绍 Web 服务的实际应用。

### 15.3.1 使用存在的 Web 服务

使用存在的 Web 服务是指使用在互联网上由服务供应者提供的各种功能性的 Web 服务，比如查询某地的天气预报、查询火车时刻表等等。演示一个例子来调用这样的两个 Web 服务。

**【例 15-2】**借助于提供火车运行时刻表的 Web 服务，通过选择起始站和终点站，查询相关的火车运行信息并显示在列表中。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 15-2”。

**02** 右键单击该网站名称，在弹出的快捷菜单中选择“添加 Web 引用”选项，打开如图 15-19 的“添加 Web 引用对话框”，在 URL 地址栏中输入提供 Web 服务的地址：  
<http://webservice.webxml.com.cn/WebServices/TrainTimeWebService.asmx>，单击“前往”按钮。

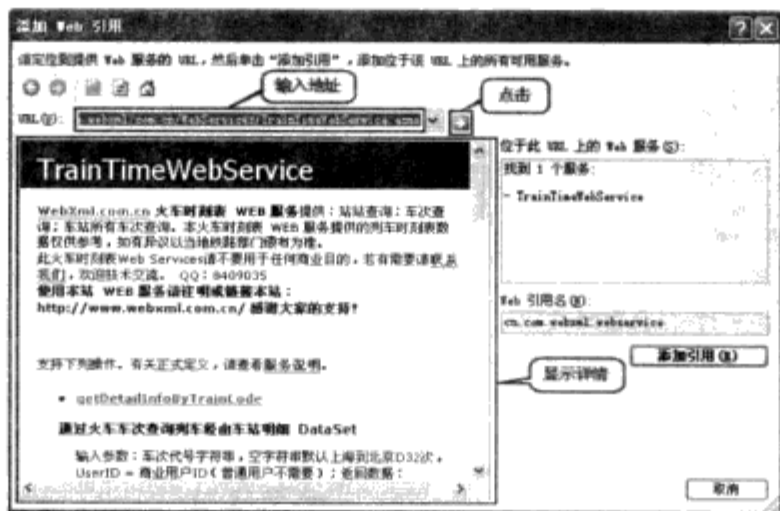


图 15-19 服务操作列表窗口

**03** 进入服务操作列表窗口。下面的服务详情窗口中共列出了八个获取火车运行时刻的操作，这里需要用到的是其中的两个：

- 通过发车站和到达站查询火车时刻表的操作 `getStationAndTimeByStationName`。输入参数：`StartStation` = 发车站，`ArriveStation` = 到达站（支持第一个字匹配模糊查询），空字符串默认发车站上海和到达站北京，`UserID` = 商业用户 ID（普通用户不需要）；返回数据：`DataSet`，`Item.(TrainCode)`=车次、`Item.(FirstStation)`=始发站、`Item.(LastStation)`=终点站、`Item.(StartStation)`=发车站、`Item.(StartTime)`=发车时间、`Item.(ArriveStation)`=到达站、`Item.(ArriveTime)`=到达时间、`Item.(KM)`=里程(KM)、`Item.(UseDate)`=历时。
- 获得本火车时刻表 Web Services 的全部始发站名称的操作 `getStationName`。输入参数：无，输出参数 `String`。

我们的应用程序根据这两个操作的要求对应编写调用的方法即可。可以测试一下该服务是否能正常使用，单击“`getStationAndTimeByStationName`”进入如图 15-20 所示测试窗口。根据说明在参数 `StartStation` 起始站后的文本框中输入“上海”，在参数 `ArriveStation` 终点站后的文本框中输入“北京”，单击“调用”按钮。

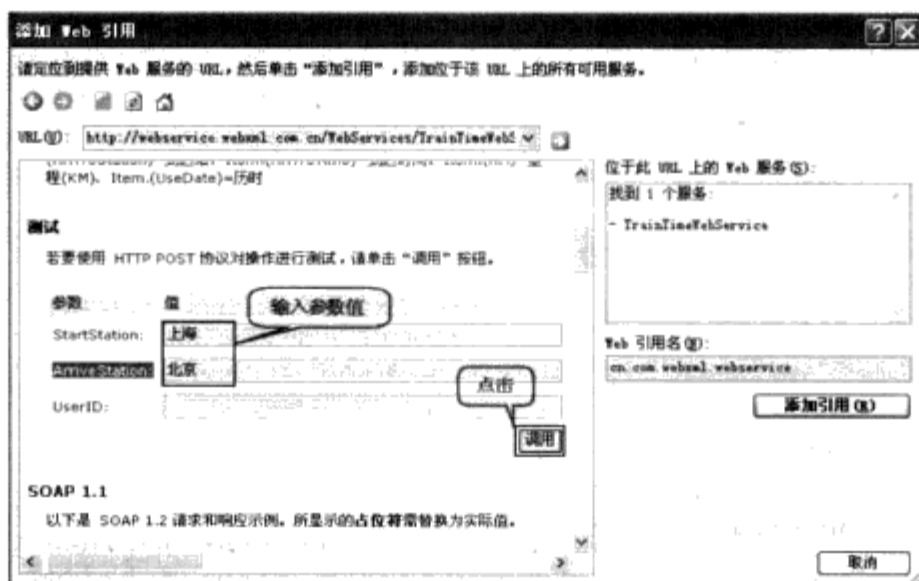


图 15-20 测试窗口

**04** 在浏览器中出现如图 15-21 的 XML 格式测试结果页面，显示了上海到北京的火车时刻表信息。

```

<?xml version="1.0" encoding="utf-8" ?>
<diffgr xmlns="urn:schemas-microsoft-com:xml-diffgram-v1">
 <getStationAndTime xmlns="">
 <TimeTable diffgr:id="TimeTable1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
 <TrainCode>1462</TrainCode>
 <FirstStation>上海</FirstStation>
 <LastStation>北京</LastStation>
 <StartStation>上海</StartStation>
 <StartTime>12:10:00</StartTime>
 <ArriveStation>北京</ArriveStation>
 <ArriveTime>10:00:00</ArriveTime>
 <KM>1463</KM>
 <UseDate>21:50</UseDate>
 </TimeTable>
 <TimeTable diffgr:id="TimeTable2" msdata:rowOrder="1" diffgr:hasChanges="inserted">
 <TrainCode>D30</TrainCode>
 </TimeTable>
 </getStationAndTime>
</diffgr>

```

图 15-21 测试结果页面

**05** 通过测试说明此 Web 服务能够正常使用，最后在图 15-20 所示的“Web 引用名”文本框中输入“localhost”，单击“添加引用”按钮，完成 Web 服务的创建。

**06** 用鼠标双击“Default.aspx”文件，进入到“视图设计器”。从“工具箱”分别拖动 1 个 GridView 控件、2 个 DropDownList 控件和 1 个 Button 控件“设计视图”中。切换到“源视图”，在<form>和</form>标记间添加如下代码。

```
1. <h4>火车时刻表查询</h4>
2. 起始站: <asp:DropDownList ID="DropDownList1" runat="server"> </asp:DropDownList>
3. 终点站<asp:DropDownList ID="DropDownList2" runat="server"> </asp:DropDownList>
4. <asp:Button ID="Button1" runat="server" Text="查询" onclick="Button1_Click" />

5. <asp:GridView ID="GridView1" runat="server" ></asp:GridView>
```

代码说明：第 1 行定义标题文本。第 2 行定义了一个服务器下拉列表控件 DropDownList1 放置所有起始站的列表项。第 3 行定义了一个服务器下拉列表控件 DropDownList2 放置所有的终点站列表项。第 4 行定义一个服务器按钮控件 Button1 用于执行查询操作并设置其触发的单击事件 Click。第 5 行定义一个服务器列表控件 GridView1。

**07** 打开“GridView 任务”列表，选择“自动套用格式”，弹出“自动套用格式对话框”，在左边的选择架构列表中选中“雨天”，单击“确定”按钮。

**08** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```
1. protected void Page_Load(object sender, EventArgs e){
2. localhost.TrainTimeWebService lt = new localhost.TrainTimeWebService();
3. string[] str=lt.getStationName();
4. for (int i = 0; i < str.Length; i++){
5. DropDownList1.Items.Add(str[i]);
6. DropDownList2.Items.Add(str[i]);
7. }
8. }
9. protected void Button1_Click(object sender, EventArgs e){
10. localhost.TrainTimeWebService lt = new localhost.TrainTimeWebService();
11. GridView1.DataSource = lt.getStationAndTimeByStationName(DropDownList1.Text, DropDownList2.Text, "");
12. GridView1.DataBind();
13. }
```

代码说明：第 1 行定义处理页面 Page 对象加载事件 Load 的方法。第 2 行实例化 Web 服务 TrainTimeWebService 的对象 lt。第 3 行定义一个字符串数组对象 str 通过调用 Web 服务中的 getStationName 方法获得火车时刻表中全部始发站名称。第 4~7 行使用 for 循环将获得的始发站和终点站依次绑定到下拉列表控件 DropDownList1 和 DropDownList2 的列表项中。第 9 行定义处理按钮控件 Button 单击事件 Click 的方法。第 10 行实例化 Web 服务 TrainTimeWebService 的对象 lt。第 11 行通过调用 Web 服务中的 getStationAndTimeByStationName 方法根据起始站和终点站获得查询出火车时刻表并作为列表控件 GridView1 的数据源。第 12 行调用 GridView1 的 DataBind 方法将数据绑定到控件显示。

**09** 按下“Ctrl+F5”，运行结果如图 15-22 所示。选择好起始站和终点站，单击“查询”按钮，显示火车班次的详情。

火车时刻表查询

选择 选择 点击

起始站: 上海 终点站: 北京 查询

TrainCode	FirstStation	LastStation	StartStation	StartTime	ArriveStation	ArriveTime	KM	UseDate
1462	上海	北京	上海	12:10:00	北京	10:00:00	1463	21:50
D30	上海	北京南	上海	07:19:00	北京南	18:18:00	1463	10:59
D302	上海虹桥	北京南	上海虹桥	21:35:00	北京南	07:29:00	1454	09:54
D306	上海虹桥	北京南	上海虹桥	21:30:00	北京南	07:24:00	1454	09:54
D308	上海虹桥	北京南	上海虹桥	21:25:00	北京南	07:19:00	1454	09:54
D314	上海虹桥	北京南	上海虹桥	21:40:00	北京南	07:34:00	1454	09:54
D32	上海	北京南	上海	10:31:00	北京南	20:48:00	1463	10:17
D322	上海虹桥	北京南	上海虹桥	21:45:00	北京南	07:39:00	1454	09:54
K314	上海	北京	上海	21:36:00	北京	17:55:00	1463	20:19
T104	上海	北京	上海	21:58:00	北京	11:17:00	1463	13:19
T110	上海	北京	上海	22:04:00	北京	11:23:00	1463	13:19
T282T283	杭州	包头	上海南	19:15:00	北京西	11:24:00	1467	16:09
T36	上海	北京	上海	14:04:00	北京	05:13:00	1463	15:09

图 15-22 运行结果



通过网络获得的 Web Service 返回的信息采用的是 XML，这样的好处在于可以在不同的系统之间传递数据。

【例 15-3】借助于提供天气预报的 Web Service，通过选择省份和相应的城市来获取该城市的天气预报信息。

- 01 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 15-3”。
- 02 右键单击该网站名称，在弹出的快捷菜单中选择“添加 Web 引用”选项，打开如图 15-23 的“添加 Web 引用对话框”，在 URL 地址栏中输入提供 Web 服务的地址：<http://www.webxml.com.cn/WebServices/WeatherWebService.asmx>，单击“前往”按钮。

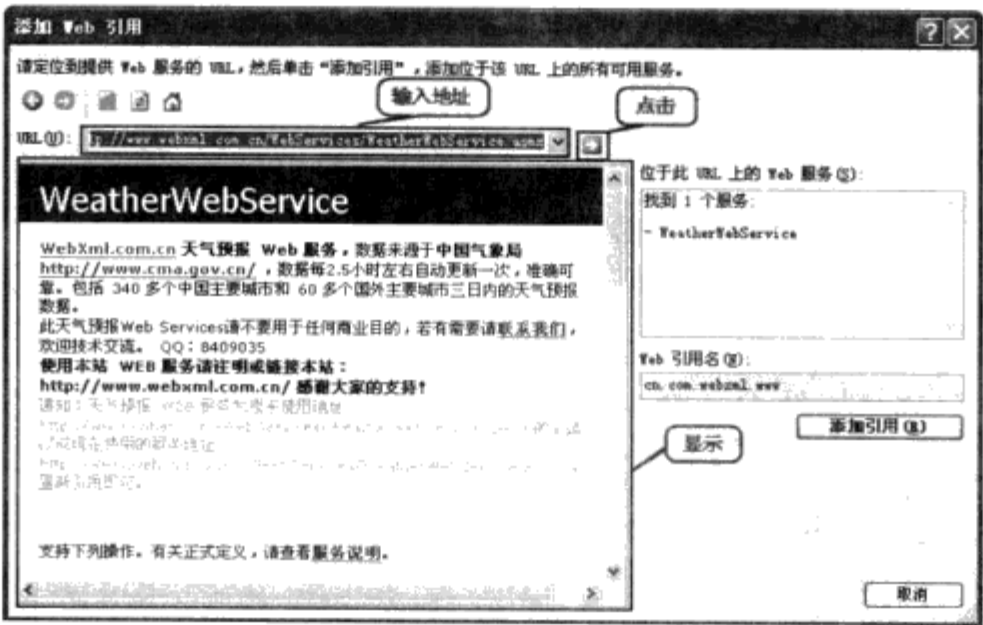


图 15-23 服务操作列表窗口

03 进入服务操作列表窗口。在服务详情中共列出了五个获取天气预报的相关操作，我们这里需要用到的是其中的三个：

- 获得本天气预报 Web Services 支持的国内省份信息的操作 `getSupportProvince`。输入参数：无；返回数据：一个一维字符串数组 `String`，内容为国内省份的名称。
- 获得本天气预报 Web Services 支持的国内城市信息的操作 `getSupportCity`。输入参数：

byProvinceName = 指定的国内省份, 若为 ALL 或空则表示返回全部城市; 返回数据: 一个一维字符串数组 String, 结构为: 城市名称 (城市代码)。

- 根据城市或地区名称查询获得未来三天内天气情况、现在的天气实况、天气和生活指数的操作: getWeatherbyCityName。调用方法如下: 输入参数: theCityName = 城市名称或城市代码 (不输入默认为上海市), 如: 上海 或 58367。返回数据: 一个一维数组 String, 共有 23 个元素。

我们的应用程序中根据这三个操作的要求对应编写调用的方法即可, 可以测试一下该服务是否能正常使用, 单击“getWeatherbyCityName”进入如图 15-24 所示测试窗口。根据说明在参数 theCityName 城市名称后的文本框中输入“北京”, 单击“调用”按钮。

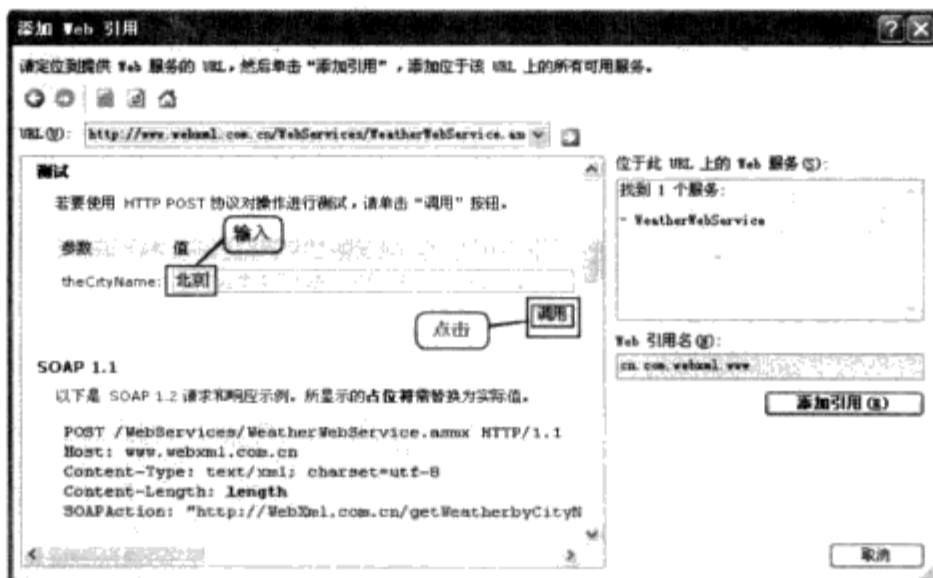


图 15-24 测试窗口

- 04 在浏览器中出现如图 15-25 的 XML 格式测试结果页面, 显示了北京的天气预报信息。

```

<?xml version='1.0' encoding='utf-8'>
<string>直辖市</string>
<string>北京</string>
<string>54511</string>
<string>54511.jpg</string>
<string>2010-10-29 16:01:45</string>
<string>5℃/18℃</string>
<string>10月29日 晴</string>
<string>无持续风向微风</string>
<string>0.gif</string>
<string>0.gif</string>
<string>今日天气实况: 气温: 19.5℃; 风向/风力: 南风 小于3级; 湿度: 20%; 气压: 1021.3hPa; 空气质量: 中; 紫外线强度: 中等</string>
<string>穿衣指数: 较凉爽, 建议着大衣加羊毛衫等春秋服装。体弱者宜着大衣加羊毛衫。因昼夜温差较大, 注意增减衣服。感冒指数: 昼夜温差较大, 较易发生感冒, 请适当增减衣服。体质较弱的朋友请注意防护。运动指

```

图 15-25 测试结果页面

- 05 通过测试说明此 Web 服务能够正常使用, 最后在上面图 15-24 的“Web 引用名”文本框中输入“localhost”, 单击“添加引用”按钮, 完成 Web 服务的创建。

- 06 用鼠标双击“Default.aspx”文件, 进入到“视图设计器”。从“工具箱”分别拖动 2 个 DropDownList 控件、1 个 Button 控件和 1 个 Label 控件到“设计视图”中。切换到“源视图”, 在<form>和</form>标记间添加如下代码。

1. <h4>获取天气预报</h4>
2. 请输入: 省份<asp:DropDownList ID="DropDownList1" runat="server" onselectedindexchanged="DropDownList1\_SelectedIndexChanged" AutoPostBack="True"></asp:DropDownList>
3. &nbsp;  城市<asp:DropDownList ID="DropDownList2" runat="server"></asp:DropDownList>

```

4. <asp:Button ID="Button1" runat="server" Text="查询" onclick="Button1_Click" style="height: 21px" />
5.

6. <asp:Label ID="Label1" runat="server" style="font-weight: 700; font-size: small" Text="" Width="550"></asp:Label>

```

代码说明：第 1 行定义标题文本。第 2 行定义了一个服务器下拉列表控件 DropDownList1 放置所有的省份选项，并设置启用自动回传功能和触发选中项改变事件。第 3 行定义了一个服务器下拉列表控件 DropDownList2 放置所有的城市选项。第 4 行定义了一个服务器按钮控件 Button1 用于执行查询操作并设置单击触发的事件。第 6 行定义一个服务器标签控件 Label1 用来显示城市名称、天气情况、温度等信息。

**07** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. BindofPrivice();
4. BindofCity();
5. }
6. }
7. protected void Button1_Click(object sender, EventArgs e){
8. Label1.Text = "";
9. localhost.WeatherWebService lw = new localhost.WeatherWebService();
10. string[] str = lw.getWeatherbyCityName(DropDownList2.Text);
11. for (int i = 0; i < str.Length; i++){
12. if (str[i].Contains("gif")){
13. i = i + 2;
14. }
15. Label1.Text += str[i] + "
";
16. i++;
17. }
18. }
19. protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e){
20. DropDownList2.Items.Clear();
21. BindofCity()
22. }
23. protected void BindofCity(){
24. localhost.WeatherWebService lw = new localhost.WeatherWebService();
25. string[] str = lw.getSupportCity(DropDownList1.Text);
26. for (int i = 0; i < str.Length; i++) {
27. DropDownList2.Items.Add(str[i].subString(0,3));
28. }
29. }
30. protected void BindofPrivice(){
31. localhost.WeatherWebService lw = new localhost.WeatherWebService();
32. string[] str = lw.getSupportProvince();
33. for (int i = 0; i < str.Length; i++){
34. DropDownList1.Items.Add(str[i]);
35. }
36. }

```

代码说明：第 1 行定义处理页面 Page 加载事件 Load 的方法。第 2 行判断当前加载的页面如果不是回传页面，则第 3 行调用 BindofPrivice 方法绑定省份信息。第 4 行调用 BindofCity 方法绑定相应城市信息。

第 7 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 9 行实例化一个 WeatherWebService 服务对象 lw。第 10 行定义字符串数组 str，通过 lw 对象的 getWeatherbyCityName 方法获得用户在

下拉列表 DropDownList2 中所选城市的天气情况。第 11~17 行使用 for 循环将天气预报的详细信息显示在 Label1 控件上。

第 19 行定义处理下拉列表控件 DropDownList1 选择索引更改事件 SelectedIndexChanged 的方法。第 20 行清空下拉列表控件 DropDownList1 的列表项内容。第 21 行调用 BindofCity 方法绑定相应的城市。

第 23 行定义一个 BindofCity 方法绑定可以查询的所有城市。第 24 行实例化一个 WeatherWebService 服务对象 lw。第 25 行定义字符串数组 str 通过 lw 对象的 getSupportCity 方法获得用户在下拉列表 DropDownList1 中所选省份的全部城市集合。第 26~28 行使用 for 循环将获得的的城市（仅显示文字部分，忽略数字部分）依次绑定到列表控件 DropDownList2 的列表项中。

第 30 行定义一个 BindofProvice 方法绑定可以查询的省份。第 31 行实例化一个 WeatherWebService 服务对象 lw。第 32 行定义字符串数组 str 通过 lw 对象的 getSupportProvice 方法获得所有省份的集合。第 33~35 行使用 for 循环将获得的所有省份依次绑定到列表控件 DropDownList1 的列表项中。

**08** 按下“Ctrl+F5”，程序运行结果如图 15-26 所示。用户先选择省份下拉列表中的省份名称，然后在从城市下拉列表中选择要查询的城市名称，最后单击“查询”按钮。下面就显示所选城市的天气预报详情。

**09** 用户可以选择不同的省份和城市，获得如图 15-27 所示的相应城市的天气预报信息。

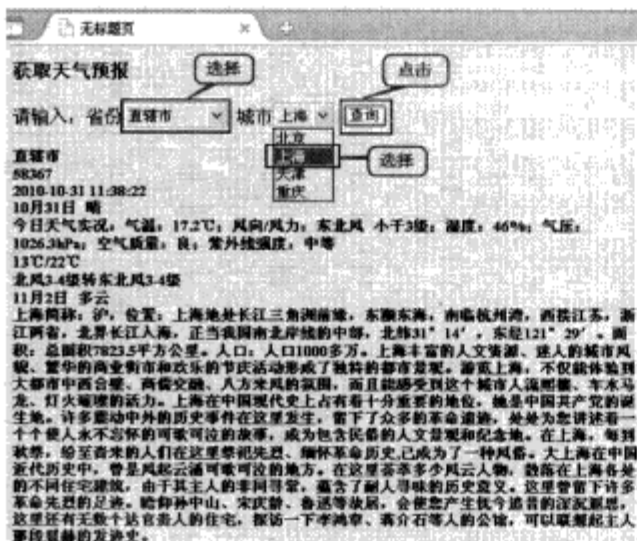


图 15-26 运行结果 1

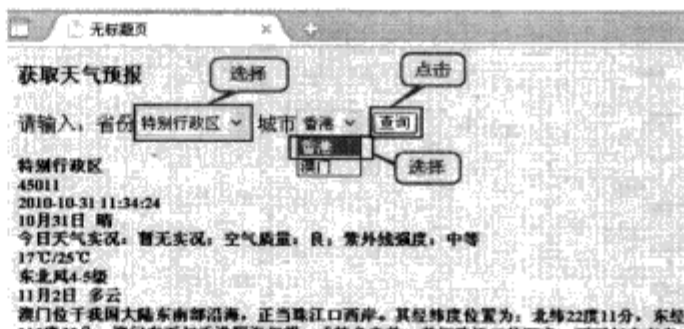


图 15-27 运行结果 2



通过互联网获得提供的 Web Service 不但可以应用于 Web 应用程序，还可以用于 Windows 桌面应用程序。

提示

### 15.3.2 调用自定义的 Web 服务

除了使用已经存在的 Web 服务以外，大多数的时候需要创建特定的 Web 服务在特定的网络应用程序运用，比如对网页中数据库的操作等。所以，下面介绍的两个实例将通过调用自己定义的 Web 服务完成对应用程序的开发。

**【例 15-4】**本例实现在网站的页面获得用户的输入，然后通过调用创建的 Web 服务中的方法

完成对数据库数据表中的数据进行修改的操作。

**01** 在 SQL Server 2005 中创建数据库 db\_16 和数据表 Users。数据表的数据结构如图 15-28 所示。

	列名	数据类型	允许空
PK	id	int	<input type="checkbox"/>
	用户名	nvarchar(50)	<input checked="" type="checkbox"/>
	密码	nvarchar(50)	<input checked="" type="checkbox"/>
	Email	nvarchar(50)	<input checked="" type="checkbox"/>
	家庭电话	nvarchar(50)	<input checked="" type="checkbox"/>
	手机号码	nvarchar(50)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

图 15-28 数据表结构

**02** 启动 Visual Studio 2010, 创建一个 ASP.NET Web 应用程序, 命名为“例 15-4”。

**03** 右键单击网站名称，在弹出的快捷菜单中选择“添加新项”命令。弹出“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“Web 服务”，然后在“名称”文本框输入该文件的名称“SqlService.asmx”，最后单击“添加”按钮。

**04** 双击网站根目录下 App\_Code 文件夹下 “SqlService.cs” 文件，添加以下代码。

```

1. [WebMethod]
2. public bool CommandSql(string SqlConnectionString, string Cmdtxt){
3. SqlConnection Con = new SqlConnection(SqlConnectionString);
4. try {
5. Con.Open();
6. SqlCommand Com = new SqlCommand(Cmdtxt, Con);
7. Com.ExecuteNonQuery();
8. return true;
9. }
10. catch (Exception ms){
11. System.Web.UI.Page tt = new System.Web.UI.Page();
12. tt.Response.Write(ms.Message);
13. return false;
14. }
15. finally{
16. Con.Close();
17. }
18. return true;
19.
20. }

```

代码说明：第 1 行添加了一个名为 WebMethod 的属性，该属性用来标志方法可以被远程的客户端访问。第 2 行定义了添加用户信息到数据库的方法 CommandSql，参数是一个数据库连接字符串和 SQL 添加语句。第 3 行创建数据库连接对象 Con。第 5 行打开数据库连接。第 6 行创建数据库命令对象 Con。第 7 行执行 sql 语句。第 10~14 行定义如果操作时发生异常情况，第 11 行实例化一个页面 Page 对象 tt。第 12 行将异常信息输出到页面显示。第 16 行关闭数据库连接。

**05** 用鼠标双击网站根目录下“Default.aspx”文件，进入“视图设计器”。从“工具箱”分别拖动 4 个 TextBox 控件，1 个 DropDownList，2 个 Button 控件和一个 GridView 控件到“设计视图”中，切换到“源视图”，在<form>和</form>标记间添加如下代码。

1. `<span style="font-size: 9pt">用户</span>`
2. `<asp:DropDownList ID="DropDownList1" runat="server" Height="18px" Width="101px"></asp:DropDownList><br />`
3. `<span style="font-size: 9pt">密  码</span>`

```

4. <asp:TextBox ID="txtPwd" runat="server" Font-Size="9pt" TextMode="Password" Width="100px"></asp:TextBox>

5. Email
6. <asp:TextBox ID="txtEmail" runat="server" Font-Size="9pt" Width="100px" style="margin-left: 0px"></asp:TextBox>

7. 家庭电话
8. <asp:TextBox ID="txtPhoneJ" runat="server" Font-Size="9pt" Width="100px"></asp:TextBox>

9. 手机号码
10. <asp:TextBox ID="txtPhoneM" runat="server" Font-Size="9pt" Width="100px"></asp:TextBox>

11. <asp:Button ID="btnOK" runat="server" Font-Size="9pt" OnClick="Button1_Click" Text="修改" />
12. <input id="Reset1" style="font-size: 9pt" type="reset" value="重置" />
13. <asp:GridView ID="GridView1" runat="server"></asp:GridView>

```

代码说明：第2行定义一个服务器下拉列表控件 DropDownList1 并设置其大小。第4行定义一个服务器文本框控件 txtPwd 用于用户输入修改的密码值，同时设置控件的宽度、文本模式和大小。第6行定义一个服务器文本框控件 txtEmail 用于用户输入修改的电子邮件值并设置控件的宽度和文本大小。第8行定义一个服务器文本框控件 txtPhoneJ 用于用户输入修改家庭电话的值并设置控件的宽度和文本大小。第10行定义一个服务器文本框控件 txtPhoneM 用于用户修改手机号码的值并设置控件的宽度和文本大小。第11行定义一个服务器按钮控件 btnOK 并设置控件显示的文字和大小同时设置触发的单击事件。第12行定义一个 HTML 的重载按钮并设置其显示的文字和大小。第13行定义一个服务器列表控件 GridView1。

**06** 在 GridView 控件右上方有一个向右的黑色小三角，单击这个小按钮打开“GridView 任务”列表，选择“自动套用格式”。弹出“自动套用格式对话框”，在左边的选择架构列表中选中“薄雾中的紫丁香”，最后单击“确定”按钮。

**07** 右键单击网站根目录，在快捷菜单中选择“添加 Web 引用”菜单选项，打开的“添加 Web 引用对话框”，选择“此解决方案中的 Web 服务”，单击“SqlService”，修改 Web 引用名为“Service”，单击“添加引用”按钮。

**08** 执行菜单栏上的“视图”|“服务器资源管理器”命令，弹出“服务器资源管理器窗口”。右键单击“数据连接”，在菜单中选择“添加连接”的命令。

**09** 在弹出的“添加连接”对话框中单击“浏览”，选择在 SQL Server 2005 创建的“db\_16.mdf”的数据库文件，单击“确定”按钮。

**10** 用鼠标双击网站根目录下的“Default.aspx.cs”文件，编写代码如下。

```

1. string ConnectionString = "Data Source=.\SQLEXPRESS;AttachDbFilename=D:\\我的文档\\ASP.NET4.从入门到精通
\\源代码\\第15章\\例15-5\\App_Data\\db_16_Data.MDF;Integrated Security=True;Connect Timeout=30;User Instance=True";
2. protected void Page_Load(object sender, EventArgs e){
3. if (!IsPostBack){
4. string cmdtxt = "SELECT * FROM Users";
5. SqlConnection Con = new SqlConnection(ConnectionString);
6. Con.Open();
7. SqlDataAdapter da = new SqlDataAdapter(cmdtxt, Con);
8. DataSet ds = new DataSet();
9. da.Fill(ds);
10. this.GridView1.DataSource = ds;
11. this.GridView1.DataBind();
12. DropDownList1.DataSource = ds;
13. DropDownList1.DataTextField = "用户名";
14. DropDownList1.DataValueField = "ID";
15. DropDownList1.DataBind();
16. }

```

```

17. }
18. protected void Button1_Click(object sender, EventArgs e){
19. string cmdtxt = "Update Users Set 密码='"+txtPwd.Text+"',Email='"+txtEmail.Text+"', 家庭电话
 ='"+txtPhoneJ.Text+"',手机号码='"+txtPhoneM.Text+"' where ID='"+DropDownList1.Selected.Value+"'";
20. SqlService service= new SqlService();
21. bool i = service.CommandSql(ConnectionStrings, cmdtxt);
22. if (i == true) {
23. Response.Write("<script>alert('添加成功!');location='Default.aspx'</script>");
24. }
25. else{
26. Response.Write("<script>alert('添加失败!');location='Default.aspx'</script>");
27. }
28. }

```

代码说明：第 1 行定义数据库连接字符串。第 2 行定义处理 Default.aspx 页面 Page 加载事件 Load 的方法。第 3 行判断当前加载的页面，如果不是回传页面，则第 4 行创建查询语句。第 5 行创建数据库连接对象。第 6 行打开数据库连接。第 7 行创建数据适配器对象 da。第 8 行创建数据集对象 ds。第 9 行将查询的结果填充到数据集对象中。第 10 行将数据集对象 ds 作为列表控件 GridView1 的数据源。第 11 行绑定数据到 GridView1 控件。第 12 行将数据集对象 ds 作为 DropDownList1 的数据源。第 13 行将数据表中的“用户名”字段作为下拉列表 DropDownList1 的显示文本。第 14 行将数据表中的“ID”字段作为下拉列表 DropDownList1 列表值。第 15 行使用 DataBind 方法绑定数据。

第 18 行定义处理按钮控件 Button1 单击事件 Click 的方法。第 19 行创建更新数据库数据的 SQL 语句，将用户输入的文本框的各种值作为参数。第 20 行创建 Web 服务类对象 service。第 21 行调用 service 对象的 CommandSql 方法判断 SQL 语句执行是否成功。第 22 行判断如果执行成功，第 23 行在页面显示删除数据成功的对话框。否则，第 26 行在页面显示删除数据失败的对话框。

**11** 按下“Ctrl+F5”，运行程序的结果如图 15-29 所示。

**12** 用户输入修改的内容，单击“修改”按钮。修改后的页面如图 15-30 所示。



图 15-29 运行结果 1



图 15-30 运行结果 2

**【例 15-5】**本例通过在客户端生成字母和数字混合验证字符串，然后调用 Web 服务中的方法，将字符串绘制成图片在传送到客户端的登录界面，并根据用户的输入正确与否给出相应的提示。

**01** 启动 Visual Studio 2010，创建一个 ASP.NET Web 应用程序，命名为“例 15-5”。

**02** 右键单击网站名称，在弹出的快捷菜单中选择“添加新项”命令。弹出“添加新项对话

框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“Web 服务”，然后在“名称”文本框输入该文件的名称“WebService.asmx”，最后单击“添加”按钮。

**03** 用鼠标双击网站根目录下“Default.aspx.”文件，进入“视图设计器”。从“工具箱”分别拖动 3 个 TextBox 控件，2 个 Button 控件和 2 个 Image 控件到“设计视图”中。

**04** 用鼠标双击“WebService.cs”文件，编写代码如下。

```

1. [WebMethod]
2. public byte[] CheckCodeService(int nLen, ref string strKey){
3. int nBmpWidth = 13 * nLen + 5;
4. int nBmpHeight = 25;
5. System.Drawing.Bitmap bmp = new System.Drawing.Bitmap(nBmpWidth, nBmpHeight);
6. int nRed, nGreen, nBlue;
7. System.Random rd = new Random((int)System.DateTime.Now.Ticks);
8. nRed = rd.Next(255) % 128 + 128;
9. nGreen = rd.Next(255) % 128 + 128;
10. nBlue = rd.Next(255) % 128 + 128;
11. System.Drawing.Graphics graph = System.Drawing.Graphics.FromImage(bmp);
12. graph.FillRectangle(new System.Drawing.SolidBrush(System.Drawing.Color.AliceBlue), 0, 0, nBmpWidth,
nBmpHeight);
13. int nLines = 3;
14. System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Color.FromArgb(nRed - 17, nGreen - 17,
nBlue - 17), 2);
15. for (int a = 0; a < nLines; a++){
16. int x1 = rd.Next(nBmpWidth);
17. int y1 = rd.Next(nBmpHeight);
18. int x2 = rd.Next(nBmpWidth);
19. int y2 = rd.Next(nBmpHeight);
20. graph.DrawLine(pen, x1, y1, x2, y2);
21. }
22. for (int i = 0; i < 100; i++){
23. int x = rd.Next(bmp.Width);
24. int y = rd.Next(bmp.Height);
25. bmp.SetPixel(x, y, System.Drawing.Color.FromArgb(rd.Next()));
26. }
27. System.Drawing.Font font = new System.Drawing.Font("Courier New", 14 + rd.Next() % 4,
System.Drawing.FontStyle.Bold);
28. System.Drawing.Drawing2D.LinearGradientBrush brush = new System.Drawing.Drawing2D.LinearGradientBrush(
29. new System.Drawing.Rectangle(0, 0, bmp.Width, bmp.Height), System.Drawing.Color.Blue,
System.Drawing.Color.DarkRed, 1.2f, true);
30. graph.DrawString(strKey, font, brush, 2, 2);
31. System.IO.MemoryStream stream = new System.IO.MemoryStream();
32. bmp.Save(stream, System.Drawing.Imaging.ImageFormat.Jpeg);
33. bmp.Dispose();
34. graph.Dispose();
35. byte[] byteReturn = stream.ToArray();
36. stream.Close();
37. return byteReturn;
38. }
```

代码说明：第 1 行添加了一个名为 WebMethod 的属性，该属性用来标志方法可以被远程的客户端访问。第 2 行定义绘制验证码的方法 CheckCodeService。第 3、4 行输出图片的长度和宽度。第 5 行创建图像。第 6~10 行生成随机三元背景色。第 11 行从图像获取一个绘图面。第 12 行填充背景。第 15~21 行绘制干扰线条 3 条，采用比背景略深一些的颜色。第 22~26 行画图片的前景噪音点 100 个。第 27~30 行定义绘制文字的字体。第 31~36 行输出字节流。第 37 行返回绘制的验证码字节数据。

**05** 右键单击网站根目录，在快捷菜单中选择“添加 Web 引用”菜单选项，打开“添加 Web 引用对话框”，选择“此解决方案中的 Web 服务”，单击“WebService”，修改 Web 引用名为“localhost”，单击“添加引用”按钮。

**06** 右键单击“网站项目名称”，在弹出的菜单中选择“添加新项”命令，打开“添加新项对话框”，选择“Web 窗体”模板，修改文件名为“CheckCode.aspx”，最后单击“添加”按钮。在网站根目录中会生成“CheckCode.aspx”文件和“CheckCode.aspx.cs”文件。

**07** 用鼠标双击“CheckCode.aspx.cs”文件文件，在该文件中编写如下逻辑代码。

```

1. protected void Page_Load(object sender, EventArgs e){
2. localhost.WebService image = new localhost.WebService();
3. int length = 4;
4. string strKey = CheckCode(length);
5. byte[] data = image.CheckCodeService(length, ref strKey);
6. Response.OutputStream.Write(data, 0, data.Length);
7. }
8. public string CheckCode(int length){
9. string strResult = "";
10. string strCode = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
11. Random rd = new Random();
12. for (int i = 0; i < length; i++){
13. char c = strCode[rd.Next(strCode.Length)];
14. strResult += c.ToString();
15. }
16. Session["CheckCode"] = strResult;
17. return strResult;
18. }

```

代码说明：第 1 行处理页面 Page 的加载事件 Load。第 2 行实例化 Web 服务的对象 image。第 4 行调用 CheckCode 方法获得数字和字母生成的随机字符串。第 5 行调用 image 对象的 CheckCodeService 方法绘制验证码。第 6 行以二进制的形式写入输出流。第 8 行定义了字母和数字混合随机生成的字符串方法 CheckCode。第 9~15 行生成 4 位随机获取的字符。第 16 行将生成的字符保存到 Session 对象中。第 17 行返回生成的字符串

**08** 用鼠标双击网站目录下的“Default.aspx.cs”文件，在该文件中编写如下逻辑代码。

```

protected void Button1_Click(object sender, EventArgs e){
 if (Session["CheckCode"].ToString() == this.txtCode.Text.Trim()){
 Response.Write("<script>alert('验证码正确!')</script>");
 }
 else{
 Response.Write("<script>alert('验证码错误!')</script>");
 }
}

```

代码说明：第 1 行处理 Button1 按钮的单击事件 Click。第 2 行判断用户输入的验证码和 Session 中保存的验证如果是一致的。第 3 行给出验证码正确的提示框。否则，第 6 行给出验证码错误的提示框。

**09** 按下“Ctrl+F5”，程序运行结果如图 15-31 所示。验证码显示在了登录页面中，用户输入用户名、密码和验证码，单击“确定”按钮。

**10** 页面弹出如图 15-32 所示验证码正确的信息提示框。

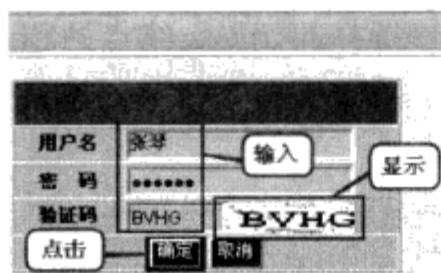


图 15-31 运行结果 1



图 15-32 运行结果 2



提示

要想在网络上引用 Web Service, 必须为该项目配置虚拟目录, 并通过在 URL 中输入 Web 服务的网络路径将服务添加到项目中。

## 15.4 上机题

1. 利用 Web Service 获取手机号码所在地的信息。该 Web Service 的地址为 <http://webservice.webxml.com.cn/WebServices/MobileCodeWS.aspx>, 运行结果如图 15-33 所示。

2. 创建一个 ASP.NET Web 服务, 进行两个整数求积运算, 并返回结果。然后在网页中调用该服务实现两个数相乘的功能, 运行结果如 15-34 所示。

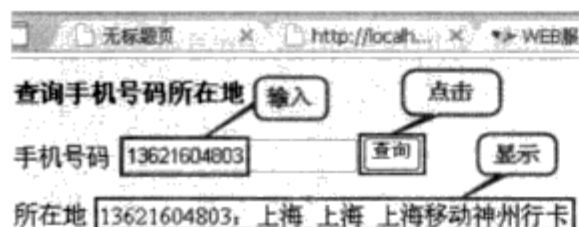


图 15-33 运行结果

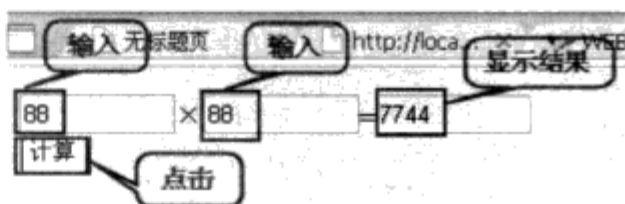


图 15-34 运行结果

3. 在调用 Web 服务时, 可以发送和接收一些数据, 这些数据包括简单的数据 (例如 int, string 等简单数据类型) 和结构体、数组、类的对象等, 还可以传送 DataSet 数据集, 本题要求通过 Web 服务使用 DataSet 对象获取本章“例 15-4”中创建的数据库表的内容, 运行结果如图 15-35 所示。

4. 利用一个现有的可获得 QQ 是否在线的 Web 服务, 实现查询获得结果。这个 Web 服务是 <http://webservice.webxml.com.cn/webservices/qqOnlineWebService.aspx>。在这个服务中提供了一个方法 qqCheckOnline, 运行结果如图 15-36 所示。



图 15-35 运行结果

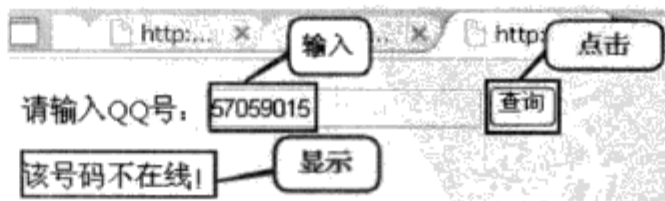


图 15-36 运行结果

5. 设计一个程序, 在程序中创建一个 Web 服务, 其中有一个验证登录名和密码是否正确的方法, 然后在页面中调用该 Web 服务完成登录验证的功能, 运行结果如图 15-37 所示。

6. 借助于提供飞机航班时刻表的 Web 服务，通过选择起始站、终点站和航班日期，查询相关的飞机航班信息并显示在列表中。该 Web 服务的地址：<http://webservice.webxml.com.cn/webservices/DomesticAirline.aspx>，运行结果如图 15-38 所示。



图 15-37 运行结果



图 15-38 运行结果

# 第 16 章 ASP.NET MVC 框架

## 学习目标

MVC 模式是一种较为广泛应用的结构设计模式,广泛应用于企业级 Web 应用的开发中。MVC 设计模式将一般的应用程序根据功能的不同,划分为 3 个主要部分,它们分别是模型、视图及控制器。微软公司新推出的基于 MVC 设计模式的 ASP.NET MVC 2 框架提供了集成于 Visual Studio 的模板,开发人员利用这个模板可以方便地构建扩展性高、容易测试的 ASP.NET MVC Web 应用程序。

## 本章重点

- 理解 ASP.NET MVC 框架的基本概念
- 在 ASP.NET MVC 中使用路由
- 能够在 ASP.NET MVC 2 框架中创建实体数据模型
- 掌握 ASP.NET MVC 2 模板的应用
- 掌握视图和控制器之间数据传递的方法

## 16.1 ASP.NET MVC 简介

在 ASP.NET 的前几个版本中,很多开发人员也都试图利用 MVC 设计模式来构建 Web 系统,但由于 ASP.NET 并没有提供像支持 Java 的 Struts 一样的 MVC 框架,应用程序员要想自己设计 MVC 结构就显得比较复杂,于是 .NET 编程人员就迫切要求微软推出支持 ASP.NET 的 MVC 框架。终于,现在微软将 ASP.NET MVC 2.0 集成到了 Visual Studio 2010 开发环境中了。

### 16.1.1 何谓 MVC

MVC 英文是 Model-View-Controller,即把一个 Web 应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离,这样,一个应用被分成三层——模型层、视图层、控制层。

视图 (View) 代表用户交互界面,对于 Web 应用来说,可以概括为 HTML 界面,但有可能为 XHTML、XML 和 Applet。随着应用的复杂性和规模性,界面的处理也变得具有挑战性。一个应用可能有很多不同的视图,MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理,以及用户的请求,而不包括在视图上的业务流程的处理。业务流程的处理交予模型 (Model) 处理。比如一个查询的视图只接受来自模型的数据并显示给用户,以及将用户界面的输入数据和请求传递给控制和模型。

模型 (Model): 就是业务流程/状态的处理以及业务规则的制定。业务流程的处理过程对其它层来说是黑箱操作,模型接受视图请求的数据,并返回最终的处理结果。业务模型的设计可以说是



## 16.1.2 ASP.NET MVC

在 MVC 设计引进到 ASP.NET Web 应用开发中之前,编程人员都在采用 Web 表单的方式来开发应用程序。Web 表单的指导思想是把 Windows 桌面应用中的表单模型引入到 Web 应用程序的开发中。这种模型很快就吸引了大批的传统 Windows 桌面应用开发人员,特别是以前的 VB 6.0 开发人员。今天,许多 VB 6.0 开发者已经转到了 ASP.NET Web 开发领域,但是他们并没有基本的 HTTP 与 Web 基本知识。为了模拟传统型 Windows 桌面应用程序中的表单开发体验,Web 表单引入了事件驱动的方法,而且还引入了 Viewstate 和 Postback 等相关概念。最终,Web 表单技术彻底地攻克了 Web 中无状态特征这个难关。

但正是基于这种指导思想,Web 表单技术存在以下无法避免的缺点。

- Viewstate 和 Postback 提高了 Web 应用程序开发的复杂性。例如,即使一些非常简单的 Web 页面也有可能产生大于 100KB 的 Viewstate,这当然会在某些情况下严重影响系统的性能。
- 开发人员无法控制 Web 表单生成的 HTML。
- ASP.NET 服务器控件生成的 HTML,既混杂有内联方式也包含不符合标准的过时标签。
- 与 JavaScript 框架的集成比较困难,这主要是因为生成的 HTML 的命名惯例所造成的。
- Web 表单相应的页面生命周期太复杂了,在整个 ASP.NET 框架中,所有内容都是紧耦合型的,并且仅使用一个类来负责显示输出和处理用户输入。因而,单元测试几乎是一项不可能完成的任务。
- 随着 Web 应用越来越复杂化,不容易测试也越来越成为实际应用开发中的一个棘手的问题。

微软公司的 ASP.NET 在最初开始的几个版中并没有提供支持 MVC 设计模式的框架,但是为了满足市场的需要和广大 ASP.NET 开发人员的要求,终于在 2008 年 3 月,微软发布了 ASP.NET MVC 预览版 2,在这个预览版中,提供了 MVC routing,并对测试功能进行了改进。另外,它还提供了 Visual Studio 2008 开发环境中第一个支持 MVC 的模板,而且对动态数据进行了改进。这个版本才是真正意义上的 ASP.NET MVC 框架。此后,这个框架经过不断更新,目前集成在 Visual Studio 2010 开发环境中的是 ASP.NET MVC 2 正式版。

ASP.NET MVC 框架为创建基于 MVC 设计模式的 Web 应用程序提供了设计框架和技术基础。它是一个轻量级的、高度可测试的演示框架,并且它结合了现有的 ASP.NET 特性(如母版页等)。MVC 框架被定义在 System.Web.Mvc 命名空间,并且是被 System.Web 命名空间所支持的。

ASP.NET MVC 框架具有如下一些特性。

- ASP.NET MVC 框架深度整合许多用户熟悉的平台特性,如运行时、身份验证、安全性、缓存和配置特性等。
- 整个架构是基于标准组件的,所以开发人员可以根据自己的需要分解或替换每个组件。
- ASP.NET MVC 框架使用用户熟悉的 ASPX 和 ASCX 文件进行开发,然后在运行时生成 HTML 的方式,并且在 Visual Studio 2010 中支持母版嵌套的功能。
- 在这个框架中,URL 将不再映射到 ASPX 文件,而是映射到一些控制类(controller classes)。所谓控制类,是一些不包含 UI 组件的标准类。
- ASP.NET MVC 框架实现了 System.Web.IHttpRequest 和 IHttpResponse 接口,这使得单元测

试能力得到了增强。

- 在进行测试时，不必再通过 Web 请求，单元测试可以撇开控制器而直接进行。
- 可以在没有 ASP.NET 运行环境的机器上进行单元测试。

ASP.NET MVC 架构能够简化 ASP.NET Web 表单方案编程中存在的复杂部分，但是在威力与灵活性方面将一点也不会逊色于后者。ASP.NET MVC 架构要实现的是在 Web 应用程序开发中引入模型-视图-控制器（即“Model-View-Controller”）UI 模式，此模式将有助于开发人员最大限度地以松耦合方式开发自己的程序。MVC 模式把应用程序分成三个部分——模型部分，视图部分以及控制器部分。

### 1. 视图部分

负责生成应用程序的用户接口，也就是说，它仅仅是填充有自控制器部分传递而来的应用程序数据的 HTML 模板。

### 2. 模型部分

负责实现应用程序的数据逻辑，它所描述的是应用程序（它使用视图部分来生成相应的用户接口部分）的业务对象。

### 3. 控制器部分

对应一组处理函数，由控制器来响应用户的输入与交互情况。也就是说，Web 请求都将由控制器来处理，控制器会决定使用哪些模型以及生成哪些视图。

MVC 模型将使用其特定的控制器动作（Action）来代替 Web 表单事件。因此，使用 MVC 模型的主要优点在于，它能够更清晰地分离关注点，更方便进行单元测试，从而能够更好地控制 URL 和 HTML 内容。值得注意的是，MVC 模型不使用 Viewstate、Postback、服务器控件以及基于服务器技术的表单，因而能够使开发人员全面地控制视图部分所生成的 HTML 内容。MVC 模型使用了基于 REST（Representational state transfer）的 URL 来取代 Web 表单模型中所使用的文件名扩展方法，从而可以使我们构造出更符合搜索引擎优化（SEO）标准的 URL。



虽然 MVC 设计模式存在种种优势，但并不代表着 MVC 设计模式能够完全取代 ASP.NET Web 表单方案。因此在实际的项目开发中，读者要根据自己的需要来选择相应的解决方案。

## 16.2 ASP.NET MVC 应用程序

在一个 ASP.NET Web 网站中，URLS 映射到存储在硬盘上的文件通常是.aspx 文件。这些.aspx 文件包括标记和被执行的代码，以对请求做出反应。与传统的 ASP.NET Web 网站不同是 ASP.NET MVC 框架应用程序则把 URLS 映射到服务器代码，它不是把 URLS 映射到 ASP.NET 页面或处理器，而是把 URLS 映射到控制器类。控制器类处理传入的诸如用户输入和交互请求，并执行相应的应用程序和数据逻辑，最后控制器类通常调用视图组件来生成 HTML 输出。

## 16.2.1 MVC 应用程序的创建

ASP.NET MVC 框架包含一个 Visual Studio 项目模板，这个模板可以为创建基于 MVC 设计模式的 Web 应用程序提供帮助。它创建一个新的 MVC Web 应用程序，并且提供了需要的文件夹、项模板和配置文件入口。

**【例 16-1】**在 Visual Studio 2010 开发环境中，创建一个基于 ASP.NET MVC 2 框架的 Web 应用程序并运行。

**01** 打开 Visual Studio 2010，选择如图 16-2 所示的菜单栏上的“文件”|“新建项目”命令。

**02** 打开如图 16-3 所示的“新建项目”对话框。选择“已安装模板”下的“Visual C#”模板中的“Web”，并在模板文件列表中选中“ASP.NET MVC 2 Web 应用程序”，然后在“名称”文本框输入“MvcApplication”，最后单击“确定”按钮。



图 16-2 新建项目

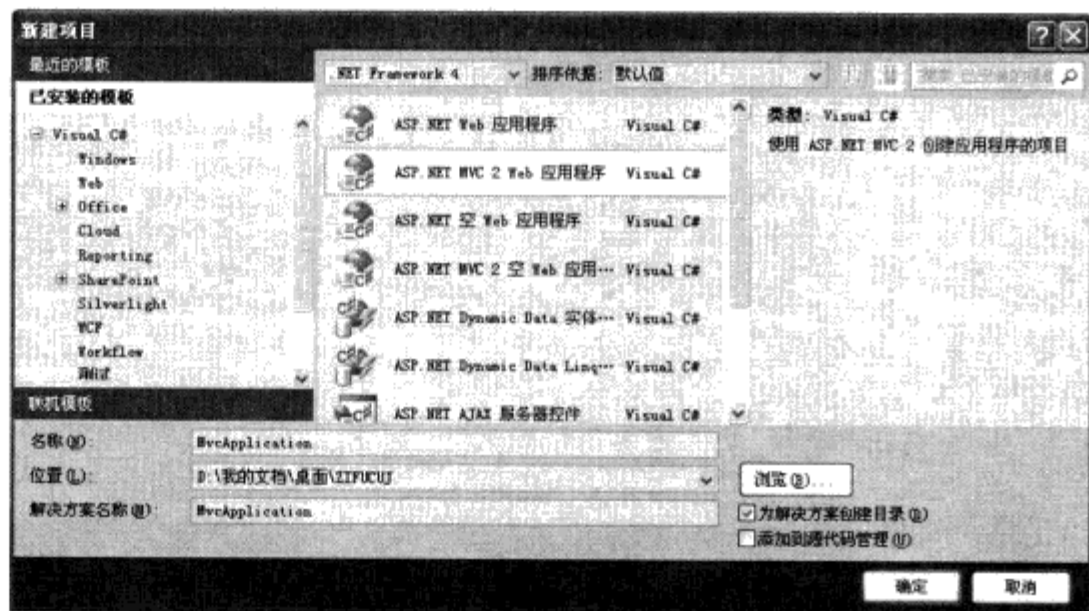


图 16-3 新建项目对话框

**03** 弹出如图 16-4 所示的“创建单元测试项目”对话框，选择“否，不创建单元测试”单选按钮，表示不创建带有单元测试项目的 MVC 网页应用，然后单击“确定”按钮。

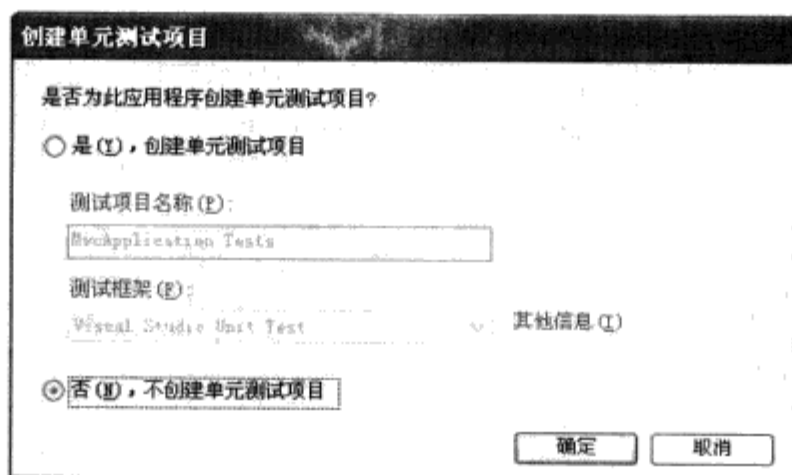


图 16-4 创建单元测试项目对话框

**04** 此时，在“解决方案资源管理器”中会自动生成一个如图 16-5 所示的基本的 ASP.NET MVC Web 应用项目“MvcApplication”。

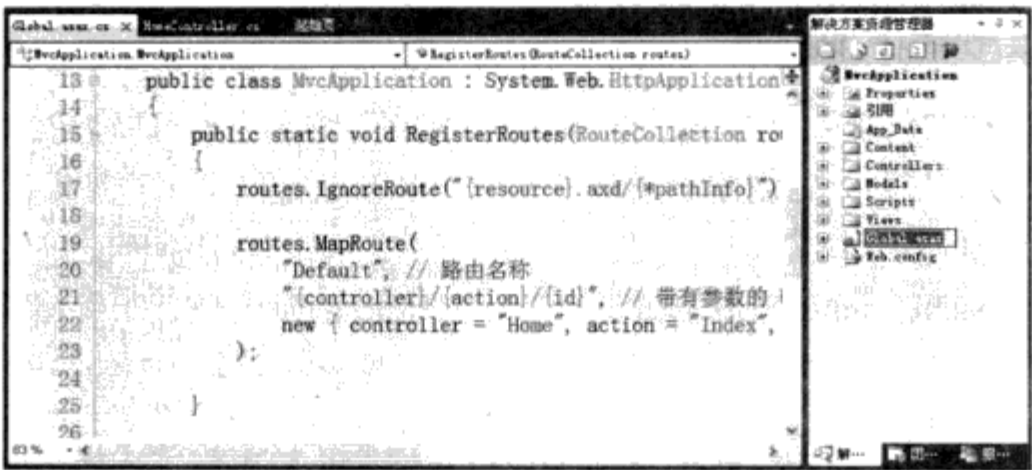


图 16-5 ASP.NET MVC 应用程序

**05** 按下“Ctrl+F5”，运行结果如图 16-6 所示，界面程序显示了“MvcApplication”项目首页。细心的读者会问还没有编写任何的代码，怎么就会出现设计好的页面呢。其实在 Visual Studio2010 开发环境中创建的每一个 ASP.NET MVC 2.0 项目都是一个模板也就是一个示例，可以为开发基于 MVC 设计模式的 Web 应用程序提供帮助。



图 16-6 程序首页

16.2.2 MVC 应用程序的结构

通过 ASP.NET MVC 项目模板创建 ASP.NET MVC 网站时，根据 ASP.NET MVC 框架的规定，ASP.NET MVC 应用程序将模型、视图和控制器组件及其他内容分别放在不同的项目目录中，以便开发者维护和管理，ASP.NET MVC 网站的目录结构如图 16-7 所示。

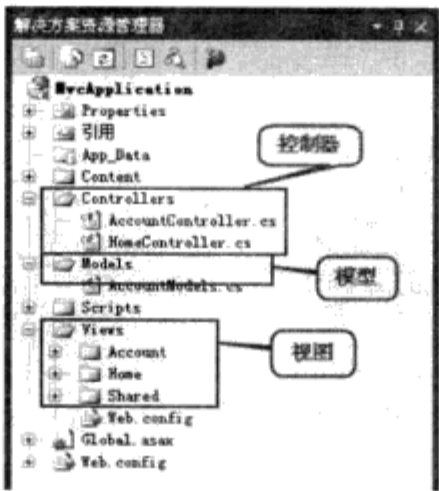


图 16-7 网站目录结构

从网站目录图中可以看到，在新创建的网站项目中包含了很多自动生成的文件夹和文件，有

些在创建其它类型的网站项目中已经介绍过，有些则是第一次看到。为了能够方便代码的管理，利用 ASP.NET MVC 框架创建出的网站项目会自动生成这些文件夹和文件：

#### (1) App\_Data 文件夹

用来存储数据，与基于 Web 表单的 ASP.NET Web 应用程序中的 App\_Data 文件夹具有相同的功能。

#### (2) Content 文件夹

存放应用程序所需要的一些静态资源文件，例如图片、CSS 样式文件等。

#### (3) Scripts 文件夹

存放 JavaScript 等脚本文件。

#### (4) Models 文件夹

模型组件一般存放在 Models 文件夹中，例如 LINQ to SQL 类或者 ADO.NET Entity Data Model 就可以存放在该目录中。该目录还可以存放有关数据访问操作的一些类和对象的定义等。

#### (5) Views 文件夹

视图组件一般存放在 Views 文件夹中，可以存放的类型包括：.aspx 页面、.ascx 控件及 .master 母版页等。这里需要说明的是对每一个控制器，在 View 文件夹中都有一个与控制器对应的目录。例如，存在一个控制器 HomeController，那么在 Views 文件夹中，就必须创建一个 Home（控制器 HomeController 名称的前面部分）的目录，这样当 ASP.NET MVC 框架通过控制器 HomeController 加载相关的视图时，就会自动寻找 Views/Home 目录下的相关 .aspx 页面。

#### (6) Shared 文件夹

对于视图组件中的公用部分，可以创建一个名为“Shared”的文件夹，该目录不属于单个的控制器，而是属于所有的控制器，在 Shared 中可以存放母版页、CSS 样式表等文件。

#### (7) Controllers 文件夹

控制器组件一般存放在 Controllers 文件夹中，控制器的命名约定采用 XXXController 的方式。

#### (8) Global.asax 文件

这一文件非常的重要，因为在 ASP.NET MVC 中，使用了 Global.asax 文件中的后置代码文件 Global.asax.cs，在它里面默认生成了 URL 寻址代码以及自定义的相关的路由逻辑。打开该文件可以看到如下的代码：

```

1 public class MvcApplication : System.Web.HttpApplication{
2 public static void RegisterRoutes(RouteCollection routes){
3 routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
4 routes.MapRoute(
5 "Default", //路由名称
6 "{controller}/{action}/{id}", //带参数的 URL
7 new { controller = "Home", action = "Index", id = "" } // 默认的参数值
8);
9 }
10 protected void Application_Start(){ //在程序启动时执行这个事件

```

```
11 RegisterRoutes(RouteTable.Routes);
12 }
13 }
```

以上代码，第 1 行定义了一个 `MvcApplication` 类继承 `System.Web.HttpApplication`。第 2 行定义了静态方法 `RegisterRoutes`，它用来实现 MVC 应用程序的 URL 寻址功能。第 3 行定义了可以忽略的路由配置，也就是说，不需要路由处理程序去处理这些路由。第 4~9 行定义了一个默认的路由。第 10 行在 `Application` 的开始事件 `Start` 中调用上面的 `RegisterRoutes` 方法。当程序运行后，程序就会按照方法 `RegisterRoute` 定义的寻址功能来实现应用程序的寻址。



在 ASP.NET MVC 框架中，有时还需要通过配置 `Web.config` 注册专门的 HTTP 模块，在 `httpModules` 节点中，注册 `UrlRoutingModule` 类，用于解析 URL 的路由，这是使用 ASP.NET MVC 框架和传统的 ASP.NET 程序的根本区别。

16.2.3 路由

对初次接触 ASP.NET MVC 的读者来说，路由是学习 ASP.NET MVC 框架最重要和最难以理解的一个概念。所谓 URL 路由（URL Routing），指的是在基于 ASP.NET MVC 的网站中，URL 不再是文件目录中的一个文件，而是一个说明有关 URL 路由的字符串，开发人员可以自行定义该字符串的格式，方便使用者理解相关的页面功能。

1. 定义路由

定义 URL 路由，就是设置 URL 模式。在 URL 路由中，通过大括号“{}”定义占位符，这些占位符就是 URL 路由的参数，而字符中的“/”、“.”等符号则被作为分隔符被 URL 路由解析这些离散的数据，对于不在小括号或者方括号中的信息则被视为一个常量。表 16-1 说明了如何定义 URL 路由。

表 16-1 定义 URL 路由

有效的 URL 路由定义	匹配的 URL 例子
{controller}/{action}/{id}	/Products/show/beverages
{table}/Details.aspx	/Products/Details.aspx
blog/{action}/{entry}	/blog/show/123
{reporttype}/{year}/{month}/{day}	/sales/2008/1/5

在表 16-1 中可以看到，第一行定义了含有 3 个 URL 路由参数的 URL 路由，此时例子中的 `Products` 就是控制器的名称，`show` 就是该控制器中所定义的一个方法，而 `beverages` 则是一个 `id` 变量。对于第 2 行所定义的 URL 路由来说，例中的 `Products` 是一个数据表名称，而 `Details.aspx` 则是一个常量。第 3 行定义了一个含有 2 个 URL 路由参数的 URL 路由，此时例子中的 `blog` 是一个常量，`show` 是相关控制器中所定义的一个方法，而 `123` 则是一个 `enty` 变量。第 4 行定义了含有 4 个 URL 路由参数的 URL 路由，此时 `sales` 是一个 `repotrtype` 变量，`2008` 是一个 `year` 变量，`1` 是一个 `month` 的变量，`5` 是 `day` 的变量。因此，通过定义 URL 路由，非常有利于对相关页面功能的理解。

通常情况下，路由的添加是在文件 Global.asax 的 Application\_Start 事件处理函数中进行的，这样可以确保当应用程序启动时路由是可用的，并且在对应用程序进行单元测试时还支持直接调用该方法。如果想在单元测试应用程序时直接调用它，那么，必须把注册路由的方法设置为静态的，并且为其提供一个参数 RouteCollection。

我们一般是通过把各个路由添加到 RouteTable 类的静态 Routes 属性中实现最终添加路由的。其中，属性 Routes 是一个 RouteCollection 对象集合，它存储了 ASP.NET 应用程序所有的路由。

下面的代码展示了来自于文件 Global.asax 中的代码片断，在代码中添加了一个 Route 对象，此对象中定义了两个名字分别为 action 和 categoryName 的 URL 参数。

```

1. protected void Application_Start(object sender, EventArgs e){
2. RegisterRoutes(RouteTable.Routes);
3. }
4. public static void RegisterRoutes(RouteCollection routes){
5. routes.Add(new Route
6. (
7. "Category/{action}/{categoryName}"//定义路由方式
8. , new CategoryRouteHandler()//默认路由
9.));
10. }

```

在上述代码中，第 1 行定义处理 Application 对象开始事件 Start 的方法，调用下面定义的 RegisterRoutes 方法。第 2 行调用第 4 行所定义的 RegisterRoutes 方法，在该方法中定义了一个 URL 路由，其中定义了两个 URL 路由参数，它们分别是 Action 变量和 categoryName。当 URL 路由处理 URL 请求时，首先去寻找匹配的 URL 路由，例如被请求的 URL 为 http://server/application/Category/Show/Tools 时，根据代码中所定义的 URL 路由，被请求的 URL 是匹配该 URL 路由的，因此 URL 路由参数中的 Action 变量为对应控制器中的 Show 方法，categoryName 变量则为 Tools。如果被请求的 URL 为 http://server/application/Category/Add，此时该 URL 与代码中所定义的 URL 路由不匹配，并且也没有定义默认的 URL 路由，那么此时的 URL 路由将不处理该请求，这个 URL 请求被当做普通的页面由传统的 ASP.NET 应用程序处理。

## 2. 设定 URL 路由参数的默认值

当定义好一个路由后，可以把一个默认的值赋给一个参数。如果 URL 中没有提供此参数值，那么将使用此默认值。为一个路由设置默认值，可以通过把一个字典赋值给 Route 类的 Defaults 属性来实现。

下面代码演示了如何为 URL 路由的参数设置默认值。

```

1. void Application_Start(object sender, EventArgs e){
2. RegisterRoutes(RouteTable.Routes);
3. }
4. public static void RegisterRoutes(RouteCollection routes){
5. routes.Add(new Route
6. (
7. //定义新的路由，寻址模式 Category/{action}/{categoryName}
8. "Category/{action}/{categoryName}",
9. new CategoryRouteHandler()
10.)
11. {
12. Defaults = new RouteValueDictionary //默认的地址

```

```
13. {"categoryName", "food"}, {"action", "show"}}
14. }
15.);
```

以上代码中，第 5~10 行创建了带两个参数“action”和“categoryName”的 URL 路由。第 12~14 行为创建好的 URL 路由中的参数设置默认值，即 categoryName 变量的默认值是 food，而 Action 方法则是对应控制器中的 show 方法。表 16-2 说明如何使用 URL 路由参数的默认值。

表 16-2 URL 路由参数的默认值

被请求的 URL	参数值
/Category	action = "show"    categoryName = "food"
/Category/add	action = "add"    categoryName = "food"
/Category/add/beverages	action = "add"    categoryName= "beverages"

从表 16-2 中可以看到，第一行被请求的 URL 中没有包括任何的 URL 路由参数，因此 URL 路由将使用设定的默认值，此时 categoryName 变量的默认值是 food，而 Action 方法则是对应控制器中的 show 方法。第 2 行被请求的 URL 中包括一个 URL 路由参数，因此 URL 路由解析该 URL 后，此时 categoryName 变量的默认值是 food。而 Action 方法则是对应控制器中的 add 方法。第 3 行被请求的 URL 中包括完整的 URL 路由参数，因此 URL 路由解析该 URL 后，此时 categoryName 变量的默认值是 beverages，而 Action 方法则是对应控制器中的 add 方法。

3. 使用 URL 路由

在 ASP.NET MVC 框架中，使用 URL 路由将请求的 URL 参数映射到控制器中的相关方法。下面说明在 ASP.NET MVC 中如何使用 URL 路由。

(1) 设定默认的 URL 路由

在通过 ASP.NET MVC 项目模板所建立的一个基本 MVC 网站中，在 Global.asax 文件中就已经设定了默认的 URL 路由，以便我们即刻运行所建立的 MVC 网站。表 16-3 说明了所设定的默认 URL 路由。

表 16-3 ASP.NET MVC 支持的默认的 URL 模式

默认的 URL 模式	匹配 URL 的例子
{controller}/{action}/{id}	http://server/application/Products/show/beverages
Default.aspx	http://server/application/Default.aspx

从表 16-3 中可以看到，第一行设定了 URL 路由应当包括 3 个部分的路径参数，对于右边匹配的 URL 来说，对应的控制器名称为 ProductsController，控制器 ProductsController 中的执行方法为 show，而 id 变量则为 beverages。第 2 行设定了默认的 URL 路由为 Default.aspx 页面，该页面所对应的 URL 为 http://server/application/Default.aspx。

在设定了默认路径之后，如果被请求的 URL 没有包括相关的 URL 路由参数，那么 ASP.NET MVC 框架会设置默认的 URL 路由参数，表 16-4 说明了如何使用默认的 URL 路由参数。

表 16-4 对应两种模式相应的默认的 URL 路由参数

默认的 URL 模式	默认值
{controller}/{action}/{id}	action="Index" id=null
Default.aspx	controller="Home" action="Index" id=null

从表 14-4 中可以看到，如果第一行所定义的 URL 路由，那么此时默认的执行方法为 Index，而 id 变量则为 null。如果被请求的页面为 Default.aspx，那么 ASP.NET MVC 框架使用默认的控制  
器为 HomeController，默认的执行方法为 Index，而 id 变量则为 null。

(2) 设置路由的类

在 ASP.NET 4.0 中，包括了一个新的命名空间 System.Web.Routing，该程序集下的各个类主要实现路由的定义、解析、匹配等功能。也就是说，路由并不是专门服务于 ASP.NET MVC 框架，它同样可以运用在 WebForm 程序中。

① Route 类

Route 类是抽象类 RouteBase 的子类，在 Route 类中，设置了路由中的 5 个基本属性，它们分别是路由的约束 Constraints、路由的命名空间 Constraints、路由参数的默认值 Defaults、路由处理程序 RouteHandler 和路由 Url。Route 类还定义了 4 个重载的构造函数如表 16-5 所示。

表 16-5 Route 类的构造函数列表

Route 类的构造函数	说明
public Route(string url, IRouteHandler routeHandler) public Route(string url, RouteValueDictionary defaults, IRouteHandler routeHandler)	使用指定的 URL 模式和处理程序类初始化 System.Web.Routing.Route 类的新实例 使用指定的 URL 模式、默认参数值和处理程序类初始化 System.Web.Routing.Route 类的新实例
public Route(string url, RouteValueDictionary defaults, RouteValueDictionary constraints, IRouteHandler routeHandler)	使用指定的 URL 模式、默认参数值、约束和处理程序类初始化 System.Web.Routing.Route 类的新实例
public Route(string url, RouteValueDictionary defaults, RouteValueDictionary constraints, RouteValueDictionary dataTokens, IRouteHandler routeHandler)	使用指定的 URL 模式、默认参数值、约束、自定义值和处理程序类初始化 System.Web.Routing.Route 类的新实例

从表中可以看出，在最简单的构造函数中，需要输入 URL 路由和路由处理程序两个参数；在最复杂的构造函数中，则需要输入 Route 类中的 5 个基本属性。

以下说明如何使用包括 Route 类中 5 个基本属性的构造函数，代码如下：

```
1. Route route=new Route("Archive/{entryDate}");
2. new RouteValueDictionary{{"controller","Blog"}, {"action","Archive"}},
3. new RouteValueDictionary{{"entryDate",@"\d{2}-d{2}-d{4}"}};
```

```

4. new RouteValueDictionary {{"namespace","Spencer.Route"}},
5. new MvcRouteHandler()
6.);

```

以上代码定义了一个路由“Archive/{entryDate}”、一个 URL 路由参数的默认值 `new RouteValueDictionary{{"controller", "Blog"}, {"action", "Archive"}}`、一个利用正则表达式定义输入参数 `entryDate` 为指定日期格式的约束 `new RouteValueDictionary{{"entryDate", @"\d{2}-\d{2}-\d{4}"}}`、一个定义命名空间的 `new RouteValueDictionary {{"namespace", "Spencer.Route"}}` 以及路由处理程序 `new MvcRouteHandler()`。



提示

这里需要说明的是，在定义路由相关参数的约束时，除了使用正则表达式设置字符串的模式之外，还可以使用 `HttpMethodConstraint` 和自定义路由约束。

## ② RouteCollection 类

在实际的路由运用中，需要创建多个路由，而 `RouteCollection` 类就是用来管理这些路由集合的。通过 `RouteTable` 类的静态属性 `Routes` 可以获得 `RouteCollection` 类的实例化对象。利用这一特性，可以在 `Global.asax` 文件中设置多个路由，设置的代码如下。

```

1. protected void Application_Start() {
2. RegisterRoutes(RouteTable.Routes);
3. }
4. public static void RegisterRoutes(RouteCollection routes) {
5. routes.Add(new Route("{controller}/{active}/{id}",
6. new RouteValueDictionary {{"controller","Home"},
7. {"action","Index"}, {"id"," "},
8. new MvcRouteHandler())
9.);
10. routes.Add(new Route("Category/{Active}/{categoryName}",
11. new RouteValueDictionary {{"categoryName", "food"},
12. {"Action","show"}},
13. new MvcRouteHandler())
14.);
15. }

```

以上代码中，第 2 行的方法参数使用了 `RouteTable.Routes` 属性，以便获得 `RouteCollection` 类的实例化对象，然后通过第 5 行、第 10 行 `RouteCollection` 类的 `Add` 方法在集合类中添加新的路由。

## ③ MapRoute 扩展方法

添加路由最简单的方法是使用位于命名空间 `System.Web.Mvc` 中的 `RouteCollectionExtension` 静态类，在这个类中针对路由集合类 `RouteCollection` 扩展了两个方法，他们分别是 `IgnoreRoute` 方法和 `MapRoute` 方法。`IgnoreRoute()` 方法主要用于设置不需要使用路由解析的 URL 地址，有两个重载的方法。`MapRoute()` 方法则用于设置各种的路径，一共有 6 个重载的方法。`RouteCollectionExtension` 类的扩展方法如表 16-6 所示。

表 16-6 RouteCollectionExtension 类的扩展方法

扩展方法名称	说明
IgnoreRoute(this RouteCollection routes, string url)	忽略给定可用路由列表和约束列表的指定 URL 路由, 需要输入 URL 路由参数
IgnoreRoute(this RouteCollection routes, string url, object constraints)	映射指定的 URL 路由, 需要输入 URL 路由参数和约束路由
MapRoute(this RouteCollection routes, string name, string url)	映射指定的 URL 路由并设置默认路由值, 需要输入 URL 路由名称和路由参数
MapRoute(this RouteCollection routes, string name, string url, object defaults)	映射指定的 URL 路由并设置命名空间, 需要输入 URL 路由名称、路由和路由默认参数
MapRoute(this RouteCollection routes, string name, string url, string[] namespaces)	映射指定的 URL 路由并设置默认路由值和约束, 需要输入 URL 路由名称、路由和命名空间参数
MapRoute(this RouteCollection routes, string name, string url, object defaults, object constraints)	映射指定的 URL 路由并设置默认的路由值和命名空间, 需要输入 URL 路由名称、路由、路由默认值和约束参数
MapRoute(this RouteCollection routes, string name, string url, object defaults, string[] namespaces)	映射指定的 URL 路由并设置默认的路由值、约束和命名空间, 需要输入 URL 路由名称、路由、路由默认值和命名空间参数
MapRoute(this RouteCollection routes, string name, string url, object defaults, object constraints, string[] namespaces)	需要输入 URL 路由名称、路由、路由默认值、约束和命名空间参数

从上表中可以看出, 各种扩展方法中的输入参数仍然是 Route 类中的 5 个基本属性, 不过通过定义新的扩展方法, 这些基本属性的变量类型有所改变, 以便开发者更加方便地设置这些属性。如路由的默认值由原来的 RouteValueDictionary 类型, 改变为现在的 object 类型; 命名空间也由原来的 RouteValueDictionary 类型, 改变为现有的 string[] 类型。

**【例 16-2】** 在 ASP.NET MVC 2 项目中添加 Default.aspx 页面的路由, 并实现路由解析该页面。

- 01 打开 Visual Studio 2010, 选择菜单栏上的“文件”|“新建项目”命令。
- 02 打开 “新建项目”对话框。选择“已安装模板”下的“Visual C#”模板中的“Web”, 并在模板文件列表中选中“ASP.NET MVC 2 Web 应用程序”, 然后在“名称”文本框输入“MvcApplication2”, 最后单击“确定”按钮。
- 03 弹出 “创建单元测试项目”对话框, 选择“否, 不创建单元测试”单选按钮, 表示不创

建带有单元测试项目的 MVC 网页应用，然后单击“确定”按钮

**04** 此时，在“解决方案资源管理器”中会自动生成一个基本的 ASP.NET MVC 的 Web 应用项目“MvcApplication2”。

**05** 用鼠标双击网站根目录下的“Global.asax”文件夹中的“Global.asax.cs”文件，在文件中编写代码如下。

```

1. public static void RegisterRoutes(RouteCollection routes){
2. routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
3. routes.MapRoute(
4. "Start",
5. "Default.aspx",
6. new { controller = "Home", action = "Index", id = "" }
7.);
8. routes.MapRoute(
9. "Default",
10. "{controller}/{action}/{id}",
11. new { controller="Home",action="Index",id="" }
12.);
13. }
14. protected void Application_Start() {
15. RegisterRoutes(RouteTable.Routes);
16. RouteTable.Routes.RouteExistingFiles = true;
17. }

```

代码说明：第 1 行定义了静态方法 RegisterRoutes 添加路由，参数是 RouteCollection 类的对象 routes。第 2 行使用 RouteCollection 类的对象 routes 的 IgnoreRoute 方法设置不需要使用路由解析的 URL 地址。第 3~7 行使用 routes 对象的 MapRoute 方法添加了 Default.aspx 页面的路由。其中第 4 行的“State”是 URL 路由的名称。第 5 行设置具体的路由。第 6 行设置路由的参数。第 8~12 行使用 routes 对象的 MapRoute 方法添加网站默认的路由，同样使用了三个参数的方法。第 14 行定义处理 Application 对象开始事件 Start 的方法。第 15 行调用上面第 1 行定义的 RegisterRoutes 的注册路由的方法 RegisterRoutes。第 18 行设置了 RouteExistingFiles 的属性为 true，表示 ASP.NET MVC 框架中的路由将要解析被请求的 Default.aspx 页面，如果设置了 RouteExistingFiles 的属性为 false（RouteExistingFiles 属性的默认值）。表示 ASP.NET MVC 框架中的路由并不处理 MVC 网站中现有的 Web 文件页面，也就是说，在默认情况下，路由没有解析 Default.aspx 页面，把该页面当做普通 Web Form 页面来执行。

**06** 用鼠标双击网站根目录下的“Delete.aspx.cs”文件，将 Page\_Load 事件中的代码注释或删除掉。

**07** 按下“Ctrl+F5”，运行结果如图 16-8 所示。



图 16-8 运行结果



提示 尽量把最常用的路由存放在路由表的最前面。这样不仅可以提高生成 URL 的效率，而且还能提高路由解析的效率。因为路由是按照路由的设置次序解析的，一旦发现有匹配的路由，就不会再进行后面的路由检索，直接结束路由的解析过程。

### 16.2.4 MVC 应用程序的执行过程

当请求一个 ASP.NET MVC 应用程序时，请求首先要传递到 `UrlRoutingModule` 对象，这个对象是一个 HTTP 模块，它解析请求并执行路由选择。`UrlRoutingModule` 对象选择第一个与当前请求匹配的路由对象，路由对象通常是 `Route` 类的实例，并实现 `RouteBase` 基类。如果没有匹配的路由，`UrlRoutingModule` 对象将不会做任何事情，并让请求返回到常规的 ASP.NET 或 IIS 请求处理程序中。

`UrlRoutingModule` 对象从被选择 `Route` 对象中获取 `IRouteHandler` 对象。通常，在一个 MVC 应用程序中，`IRouteHandler` 对象将会是 `MvcRouteHandler` 类的一个实例。`IRouteHandler` 实例创建一个 `IHttpHandler` 对象并把它传递给 `IHttpContext` 对象。默认情况下，`IHttpHandler` 实例是一个 `MvcHandler` 对象。接着，`MvcHandler` 对象选择能够处理请求的控制器。

以上模块和处理器是进入 ASP.NET MVC 框架的入口，进入 MVC 框架后将执行如下的行为。

- 选择适当的控制器。
- 获得指定的控制器实例。
- 调用控制器的可执行方法。

总之，在一个 MVC Web 项目执行过程中，将经历如下几个阶段：

- ① 获取第一个请求。在 `Global.asax` 文件中，`Route` 对象被添加到 `RouteTable` 对象中。
- ② 执行路由。`UrlRoutingModule` 对象使用 `RouteTable` 集合中第一个匹配的 `Route` 对象以创建 `RouteData` 对象，利用这个对象以生成 `RequestContext` 对象（`IHttpContext` 对象）。
- ③ 创建 MVC 请求处理。`MvcRouteHandler` 对象创建一个 `MvcHandler` 类的实例，并把它传递到 `RequestContext` 实例。
- ④ 创建控制器。`MvcHandler` 对象使用 `RequestContext` 实例去确认 `IControllerFactory` 对象以创建控制器实例。
- ⑤ 执行控制器。`MvcHandler` 实例调用控制器的可执行方法。
- ⑥ 触发行为。很多控制器都继承自 `Controller` 基础类，而同控制器结合在一起的 `ControllerActionInvoker` 对象来决定控制器类调用哪个方法并调用这个方法。
- ⑦ 执行结果。一个典型的行为方法可能接收用户输入，准备适当响应数据，并通过返回一个结果类型来执行结果。可被执行的内置的结果类型包括：`ViewResult`（用来渲染视图，并且是最常用的结果类型）、`RedirectToRouteResult`、`RedirectResult`、`ContentResult`、`JsonResult` 和 `EmptyResult`。



提示

URL 路由类库位于命名空间 `System.Web.Routing` 之中，与 ASP.NET MVC 框架的命名空间 `System.Web.Mvc` 是独立的，因此可以在非 ASP.NET MVC 框架的网站中使用 URL 路由功能。

### 16.2.5 构建模型

在 ASP.NET MVC 框架中，模型主要实现应用程序中数据访问和业务逻辑，按照规定，这些模型类均存放在 `Models` 文件夹中。我们可以使用各种各样不同的技术来实现数据访问和业务逻辑。

比如 Microsoft Entity Framework、NHibernate、Subsonic 或者 ADO.NET 类来构建数据访问类。当然，最为常用的是 LINQ to SQL 类和 ADO.NET Entity Data Model。下面我们以 ADO.NET 实体数据模型为例来实现 ASP.NET MVC 程序中模型的创建。

**【例 16-3】**在 ASP.NET MVC 2 Web 应用程序的 Models 文件夹中构建在第 6 章中创建的“BookStor”数据库和“bookInfo”数据表的实体数据模型。

**01** 打开 Visual Studio 2010，选择菜单栏上的“文件”|“新建项目”命令。

**02** 打开“新建项目”对话框。选择“已安装模板”下的“Visual C#”模板中的“Web”，并在模板文件列表中选中“ASP.NET MVC 2 Web 应用程序”，然后在“名称”文本框输入“MvcApplication3”，最后单击“确定”按钮。

**03** 弹出“创建单元测试项目”对话框，选中“否，不创建单元测试”单选按钮，表示不创建带有单元测试项目的 MVC 网页应用，然后单击“确定”按钮。

**04** 此时，在“解决方案资源管理器”中会自动生成一个基本的 ASP.NET MVC 的 Web 应用项目“MvcApplication3”。

**05** 在“解决方案资源管理”窗口中的“MvcApplication3”项目内的“Models”文件夹上单击鼠标右键，在弹出如图 16-9 所示的快捷菜单中选择“添加”|“新建项”命令。

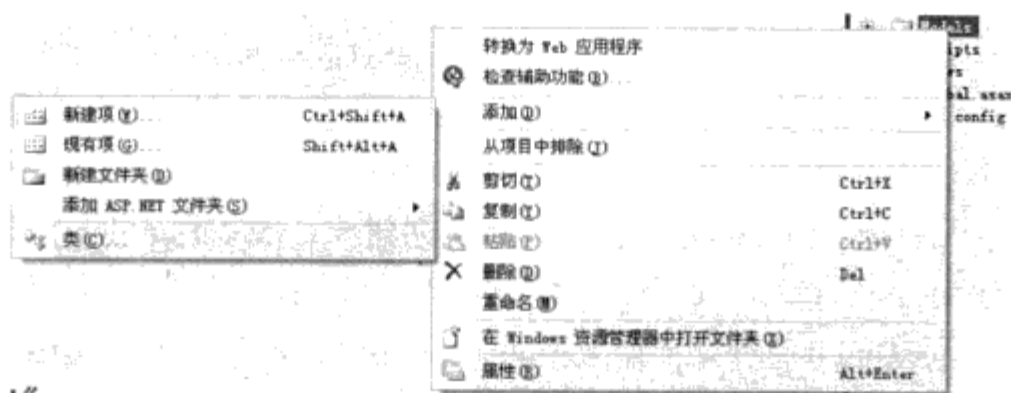


图 16-9 新建项

**06** 打开如图 16-10 所示的“添加新项”对话框。选择“已安装模板”下的“Visual C#”，并在模板文件列表中选中“ADO.NET 实体数据模型”，然后在“名称”文本框输入“bookInfo.edmx”，最后单击“添加”按钮。



图 16-10 “添加新项”对话框

07 打开如图 16-11 所示的实体数据模型向导——“选择模型内容”对话框。选择“从数据库生成”，表明 ADO.NET 实体框架从数据库直接生成实体数据模型，然后单击“下一步”按钮。

08 打开如图 16-12 所示的“选择你的数据连接”对话框，选择“BookInfo.mdf”数据库，选中“将 Web.Config 中的实体连接设置另存为”复选框，单击“下一步”按钮。



图 16-11 选择模型内容对话框

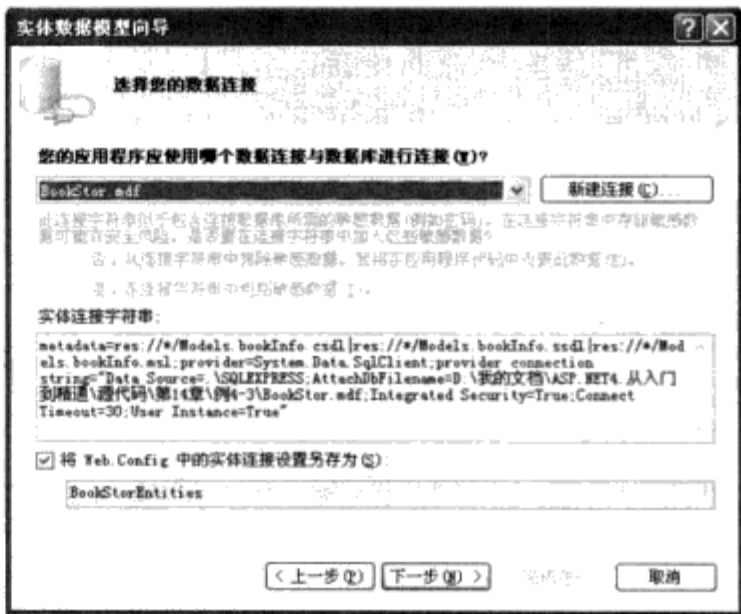


图 16-12 选择数据连接对话框

09 弹出如图 16-13 所示的“选择数据库对象”对话框。根据需要可以选择数据表、视图或储存过程。这里选择“表”和“bookInfo”，然后单击“完成”按钮。

10 ADO.NET 实体框架即可自动生成如图 16-14 所示的“bookInfo”数据表所对应的实体数据模型。在 Models 文件夹中生成了“bookInfo.edmx”文件，这个文件自动生成了 bookInfo 实体的属性和各种操作的方法。我们可以在 Controller 控制器中直接调用这些属性和方法进行对数据库进行访问并处理数据。

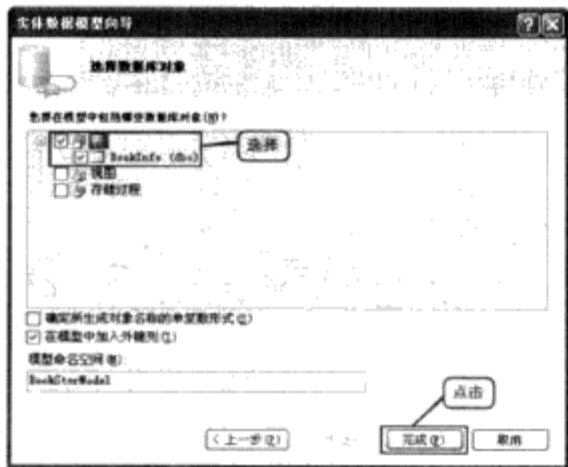


图 16-13 选择数据库对象对话框

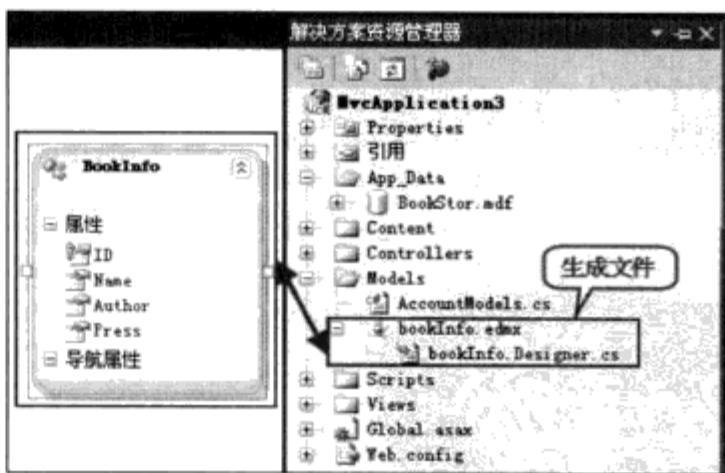


图 16-14 实体数据模型

16.2.6 控制器

在 ASP.NET MVC 中，控制器有着非常重要的作用，控制器处理用户的请求，将用户请求的 URL 路由，分发到控制器中的相关动作方法并执行适当的业务逻辑。控制器类通常调用一个单独的视图组件来生成 HTML 标记以对请求做出反应。

1. 控制器类

所有控制器的基类都是 Controller 类，这个类提供通用的 ASP.NET MVC 处理功能。Controller 类实现了 IController、IActionFilter 和 IDisposable 接口。

Controller 基类负责以下事务。

- 定位适当的行为方法。
- 获取行为方法参数的值。
- 处理在执行行为方法过程中可能出现的所有错误。
- 提供默认的 WebFormViewFactory 类以用来渲染 ASP.NET 页面类型（视图）。

ASP.NET MVC 2 框架默认在项目的“Controller”文件夹下创建“HomeController.cs”类文件来实现“Home”视图的控制器，下面代码展示了 HomeController 控制器类的定义。

```
1. public class HomeController : Controller {
2. public ActionResult Index() {
3. ViewData["Message"] = "欢迎学习 ASP.NET MVC!";
4. return View();
5. }
6. public ActionResult About() {
7. return View();
8. }
9. }
```

在上面的代码中，第 1 行定义了控制器 HomeController。所有控制器的名称必须命名为形如“XXXController”的格式，并且必须实现接口 Icontroller，或者继承抽象类 Controller 类。第 3 行定义了一个动作方法 Index，该方法的返回类型是 ActionResult。第 5 行设置了 ViewData 的字典数据，以便将控制器中指定的数据传递到视图。第 6 行调用 Controller 类中的 View 方法，返回的是一个 ViewResult 的实例化对象，将指定的内容输出到浏览器中。第 8 行定义了另一个行为方法 About，该方法的返回类型也是 ActionResult。

ActionResult 是一个抽象类，因此实际返回的类型是该抽象类的子类。ActionResult 的子类列表如表 16-7 所示。

表 16-7 ActionResult 的子类列表

ActionResult 的子类列表	说明
ViewResult	表示 HTML 的页面内容
EmptyResult	表示空白的页面内容
RedirectResult	表示定位到另一个 URL
JsonResult	表示可以运用到 AJAX 程序中 Json 结果
JavaScriptResult	表示一个 JavaScript 对象
ContentResult	表示一个文本内容
FileContentResult	表示一个可以下载的、二进制内容的文件
FilePathResult	表示一个可以下载的、指定路径的文件
FileStreamResult	表示一个可以下载的、流式的文件

2. 行为方法

在不使用 MVC 框架的 ASP.NET 应用程序中，用户交互都是围绕着页面以及引发和处理这些页面的事件进行组织的。相比之下，使用 ASP.NET MVC 应用程序的用户交互则围绕控制器及其中的行为方法进行组织。

行为方法是在控制器中定义的。通常，行为中针对每一个用户的交互都创建一个一一对应的映射，用户的交互包括在浏览器中输入一个 URL，单击一个链接，以及提交一个表单，等等。每一个这类用户交互都会导致把一个请求发送到服务器。而请求 URL 中都会包括相应的信息以便 MVC 框架调用一个相应的行为方法。

例如，当用户在浏览器输入一个 URL 时，MVC 应用程序使用定义于 Global.asax 文件中的路由规则来分析该 URL 并决定指向控制器的路径。然后，该控制器定位适当的行为方法来处理这一请求。根据具体需要，控制器中可以定义尽可能多的行为方法。

默认情况下，一个请求 URL 被当作一个子路径被解析，其中包括控制器名，后面跟着行为名。例如，如果一个用户输入 URL 为 `http://contoso.com/MyWebSite/Products/Categories`，则子路径为 `“/Products/Categories”`。默认的路由规则总是把 `“Products”` 作为控制器名，而把 `“Categories”` 作为行为名。于是，该路由规则将调用 `Products` 控制器的 `Categories` 方法来处理该请求。如果 URL 以 `“/Products/Detail/5”` 结尾，则默认的路由规则把 `“Detail”` 作为行为名，并且调用 `Products` 控制器的 `Detail` 方法来处理请求。默认情况下，URL 中的 `“5”` 将被传递为 `Detail` 方法的一个参数。

下面这段代码定义了一个控制器，并在该控制器中定义了一个行为方法：

```
1. public class MyController : Controller{
2. public ActionResult Hello(){ //控制器方法
3. return View("HelloWorld");
4. }
5. }
```

以上代码中第 2 行定义了一个名为 `“Hello”` 的行为方法，它的返回类型是 `ActionResult` 抽象类的一个子类。

在 `Controller` 类中的相关方法和返回对象的列表如表 16-8 所示。

表 16-8 控制器中的方法与返回对象列表

控制器中的方法	返回对象
View	ViewResult
Redirect	RedirectResult
RedirectToAction	RedirectToRouteResult
RedirectToRoute	RedirectToRouteResult
Json	JsonResult
JavaScriptResult	JavaScriptResult
Content	ContentResult
File	FileContentResult、FilePathResult 和 FileStreamResult

在定义控制器中的行为方法时要留意，ASP.NET MVC 框架认为所有的 `public` 方法都是行为方

法, 因此, 如果控制器类包含一个不是 public 的行为方法, 那么必须使用 NonActionAttribute 属性标记它。

**【例 16-4】**继续“例 16-3”的开发, 为创建好的“bookInfo”数据表的实体数据模建立控制器实现对“bookInfo”数据表的显示、编辑、添加、删除和查询详情的业务逻辑, 在控制器中创建相关的动作方法 Index、Edit、Create、Delete 和 Detail。

**01** 用鼠标双击项目根目录下“Controllers”文件夹中的“HomeController.cs 文件”, 在文件中编写如下代码。

```

1. [HandleError]
2. public class HomeController : Controller{
3. BookStorEntities bs = new BookStorEntities();
4. public ActionResult Index({
5. var model = bs.BookInfo.ToList();
6. return View(model);
7. }
8. [AcceptVerbs(HttpVerbs.Get)]
9. public ActionResult Edit(int id){
10. string num = id.ToString();
11. var model = bs.BookInfo.First(c => c.ID == num);
12. return View(model);
13. }
14. [AcceptVerbs(HttpVerbs.Post)]
15. public ActionResult Edit(int id, FormCollection form){
16. string num = id.ToString();
17. var model = bs.BookInfo.First(c => c.ID == num);
18. UpdateModel(model, new[] { "Name", "Author", "Press" });
19. bs.SaveChanges();
20. return RedirectToAction("Index");
21. }
22. [AcceptVerbs(HttpVerbs.Get)]
23. public ActionResult Detail(int id){
24. string num = id.ToString();
25. var model = bs.BookInfo.First(c => c.ID == num);
26. return View(model);
27. }
28. [AcceptVerbs(HttpVerbs.Get)]
29. public ActionResult Create(){
30. BookInfo bi = new BookInfo();
31. return View(bi);
32. }
33. [AcceptVerbs(HttpVerbs.Post)]
34. public ActionResult Create(int id, FormCollection form){
35. string num = id.ToString();
36. var model = bs.BookInfo.FirstOrDefault(c => c.ID == num);
37. if (model == null){
38. BookInfo bi = new BookInfo();
39. UpdateModel(bi, new[] { "ID", "Name", "Author", "Press" });
40. bs.AddToBookInfo(bi);
41. bs.SaveChanges();
42. return RedirectToAction("Index");
43. }
44. else
45. return RedirectToAction("Create");
46. }
47. [AcceptVerbs(HttpVerbs.Get)]

```

```

48. public ActionResult Delete(int id){
49. string num = id.ToString();
50. var model = bs.BookInfo.First(c => c.ID == num);
51. return View(model);
52. }
53. [AcceptVerbs(HttpVerbs.Post)]
54. public ActionResult Delete(int id, FormCollection form){
55. string num = id.ToString();
56. var model = bs.BookInfo.First(c => c.ID == num);
57. bs.DeleteObject(model);
58. bs.SaveChanges();
59. return RedirectToAction("Index");
60. }
61. }

```

代码说明：第 1 行设置了 `HandleError` 属性，当下面的控制器如果出现任何异常，就会打开个性化的异常处理页面 `Error.aspx`。第 2 行定义“Home”视图控制器类 `HomeController` 并继承 `Controller` 抽象类。第 3 行实例化实体模型类 `BookStorEntities` 的 `bs` 对象。第 4 行定义动作方法 `Index`，对应的视图是 `Index.aspx`。第 5 行调用 `bs` 的 `BookInfo` 对象的 `ToList` 方法，获得全部种类的图书信息并赋给隐藏变量 `model`。第 6 行调用控制器的 `View` 方法将 `model` 对象传递到对应的 `Index` 视图显示。

第 8 行设置 `[AcceptVerbs (HttpVerbs.Get)]` 属性，表示下面的方法只接受用户通过 `Get` 方法发送的表单数据。第 9 行定义动作方法 `Edit`，对应的视图是 `Edit.aspx`，方法的参数是图书编号 `id`。第 10 行获取方法的参数图书编号。第 11 行调用 `BookStorEntities` 的 `BookInfo` 对象 `First` 方法获得“bookinfo”数据表中指定图书的编号（ID）的数据。第 12 行将图书编号的数据传递到对应的 `Edit` 视图中。第 14 行设置 `[AcceptVerbs (HttpVerbs.Post)]` 属性，表示下面的方法只接受用户通过 `Post` 方法发送表单数据。第 15 行定义了一个重载的 `Edit` 动作方法，对应的视图是 `Edit.aspx`，方法的参数是图书编号 `id` 和 `FormCollection` 窗体传递值的集合对象。第 16 行获取方法的参数图书编号。第 17 行获得数据表“bookinfo”中指定图书编号（ID）的数据。第 18 行通过 `UpdateModel` 方法，获得 `Edit.aspx` 编辑页面中用户所修改的图书名称 `Name`、作者 `Author` 和出版社 `Press` 的值，并更新查询到的 `model` 数据。第 19 行调用 `SaveChanges` 方法将修改的数据保存到数据库。最后通过第 20 行将数据传递到对应的 `Index` 视图页面中显示。

第 22 行的属性表示下面的方法只接受用户通过 `Get` 方法发送表单数据。第 23 行定义动作方法 `Details`，对于的视图是 `Details.aspx`，方法的参数是图书编号 `id`。第 24 行获取方法的参数图书编号。第 25 行获得数据表“bookinfo”中指定图书编号 ID 的数据，并通过 26 行传递给相应的 `Details` 视图显示。

第 28 行表示下面的方法只接受用户通过 `Get` 方法发送表单数据。第 29 行定义动作方法 `Create`，对应的视图 `Create.aspx`。第 30 行创建一个新数据库表 `BookInfo` 的实例化对象 `bi`，并通过第 31 行传递到相应的 `Create` 视图显示。第 33 行表示下面的方法只接受用户通过 `Post` 方法发送表单数据。第 34 行定义一个重载的 `Create` 动作方法，对应的视图 `Create.aspx`，方法的参数是图书编号 `id` 和 `FormCollection` 窗体传递值的集合对象。第 35 行获取方法的参数图书编号。第 36 行获得数据表“bookinfo”中指定的图书编号 ID 数据。第 37 行判断如果获得了数据则第 38 行创建一个新数据库表 `BookInfo` 的实例化对象 `bi`。第 39 行通过 `UpdateModel` 方法，获得 `Create.aspx` 编辑页面中用户所修改的图书编号 ID、图书名称 `Name`、作者 `Author` 和出版社 `Press` 的值，并更新数据。第 40

行调用 `AddToBookInfo` 方法将新的对象添加到数据实体。第 41 行调用 `SaveChanges` 方法将修改的数据保存到数据库。最后通过第 42 行将数据传递到对应的 `Index` 视图页面中显示。如果第 44 行判断没有获得数据，则第 45 行重新返回到 `Create` 视图页面，等待用户继续输入添加的内容。

第 47 行属性表示下面的方法只接受用户通过 `Get` 方法发送表单数据。第 48 行定义动作方法 `Delete`，对应的视图 `Delete.aspx`，方法的参数是图书编号 `id`。第 49 行获取方法的参数图书编号。第 50 行调用 `BookStorEntities` 的 `BookInfo` 对象 `First` 方法获得“bookinfo”数据表中指定图书的编号（ID）的数据。第 51 行将图书编号的数据传递到对应的 `Delete` 视图中。第 53 行属性表示下面的方法只接受用户通过 `Post` 方法发送表单数据。第 54 行定义了一个重载的 `Delete` 动作方法，对应的视图是 `Delete.aspx`，方法的参数是图书编号 `id` 和 `FormCollection` 窗体传递值的集合对象。第 55 行获取方法的参数图书编号。第 56 行获得数据表“bookinfo”中指定图书编号（ID）的数据。第 57 行调用 `DeleteObject` 方法删除获得的数据。第 58 行调用 `SaveChanges` 方法将修改的数据保存到数据库。通过第 59 行将数据传递到对应的 `Index` 视图页面中显示。



提示

ASP.NET MVC 框架可以自动地把 URL 参数值映射为行为方法的参数值。默认情况下，如果一个行为方法中包含一个参数，那么 MVC 框架将分析到来的请求数据并决定该请求是否包含一个含有相同名字的 HTTP 请求值。如果包含的话，则该请求值会被自动地传递给该行为方法。

### 16.2.7 视图

ASP.NET MVC 框架提供一个视图引擎以生成视图。默认情况下，MVC 框架使用定制的类型来生成视图，而这个类型继承自己已经存在的 ASP.NET 页面（.aspx）、母版页（.master）和用户控件（.ascx）。在一个 ASP.NET MVC Web 应用程序的工作流程中，控制器行为方法处理收到的 Web 请求。这些行为方法使用收到的参数值来执行程序代码，并从数据库的模型中获取或更新数据。最后，他们选择一个视图来渲染用户界面。

一般情况下，在基于 MVC 的 Web 工程构架中，推荐把视图全部放到 `Views` 文件夹的下面。根据 MVC 框架要求，视图中不应该包含任何应用程序逻辑或数据库检索代码。所有的应用程序逻辑都应该由控制器来负责处理。

借助于控制器行为方法提供的与 MVC 视图相关的数据对象，由视图负责渲染相应的用户接口界面。

视图页面是一个 `ViewPage` 类，这个类继承自 `Page` 类，并实现 `IViewDataContainer` 接口。`IViewDataContainer` 接口提供了一个 `ViewData` 属性，这个属性返回一个 `ViewDataDictionary` 对象，这个对象包含视图要显示的数据。

可以使用 ASP.NET MVC Web 应用程序项目中提供的模板来创建视图。默认情况下，视图是一个由 MVC 框架渲染的 ASP.NET Web 页面。MVC 框架利用 URL 路由来决定调用哪个控制器行为，而控制器行为决定要渲染哪个视图。

下面的代码演示了对 `Index.aspx` 视图页面的定义。

```
1. <%@ Page Language="C#" MasterPageFile="~/Views/Masters/Site.Master" AutoEventWireup="true"2
2. CodeBehind="Index.aspx.cs" Inherits="MvcApplication.Views.Home.Index" %>
3. <asp:Content ID="Content2"
```

```
4. ContentPlaceHolderID="BodyContentPlaceHolder" runat="server">
5. 欢迎!
6. </asp:Content>
```

以上代码是普通的母版页和内容页的定义代码。其中，第 1 和第 2 行@Page 指令包含了 Inherits 属性，它定义了应用程序与视图之间的关系。第 3~6 行在内容页面中添加欢迎的文字。

【例 16-5】继续“例 16-3”的开发，创建了 HomeController 控制器的各种行为方法后，就要对应于这些行为方法来创建相关的视图页面。

01 双击打开 HomeController 控制器，选中 Index()方法，然后单击鼠标右键，在弹出如图 16-15 所示的快捷菜单中选择“添加视图”命令。

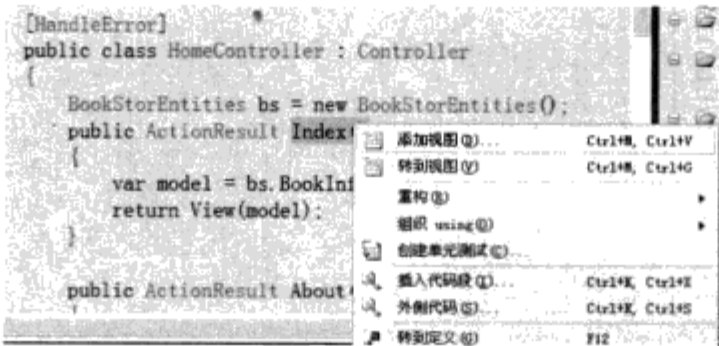


图 16-15 添加视图

02 打开如图 16-16 所示的“添加视图”对话框。选中“创建强类型视图”复选框，之所以要选择这个是因为我们是通过实体数据模型来创建数据访问的。在“视图数据类”下拉列表中找到实体数据类“MvcApplication.Modes.BookInfo”，在“视图内容”下拉列表中选择“List”列表。在这个下拉列表中有六个选项，每一个选项就是一个视图的模板：“Create”选项用于创建新建数据的视图、“Delete”选项用于创建删除数据的视图、“Details”选项用于创建单条数据详情的视图、“Edit”选项用于创建编辑数据的视图，“Empty”选项用于创建一个空的模板视图、“List”选项用于创建显示数据的视图。然后选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的 ID。如果不使用母版页这些都可以不选，最后单击“添加”按钮。ASP.NET MVC 2 就会自动创建一个 Index.aspx 页面。

03 ASP.NET MVC 2 框架就会自动在如图 16-17 所示的网站根目录“Views”文件夹下的“Home”子文件夹创建一个 Index.aspx 页面。

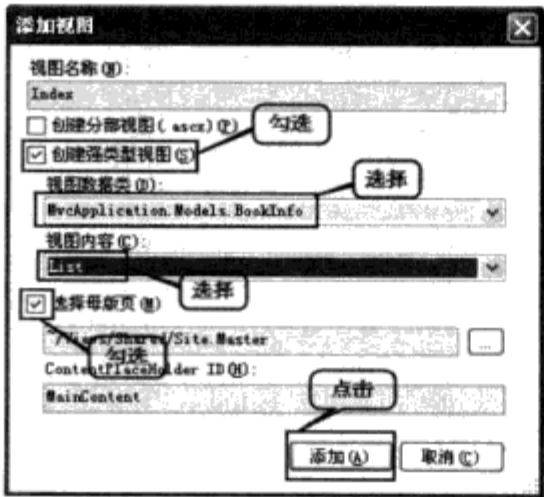


图 16-16 “添加视图”对话框

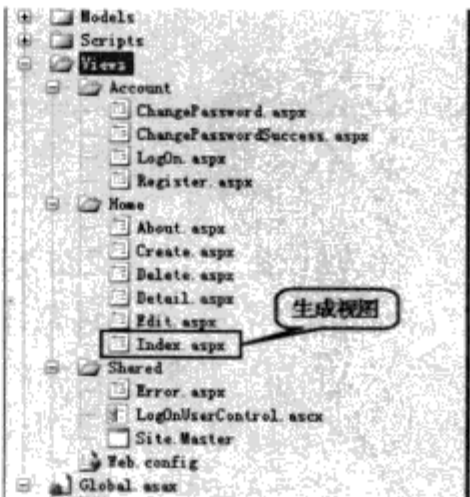


图 16-17 生成视图文件

**04** 双击 Index.aspx 文件，在文件中生成的实现代码如下。

```

1. <asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
2. Index
3. </asp:Content>
4. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
5. <h2>Index</h2>
6. <table>
7. <tr><th></th>
8. <th>
9. 编号
10. </th>
11. <th>
12. 书名
13. </th>
14. <th>
15. 作者
16. </th>
17. <th>
18. 出版社
19. </th>
20. </tr>
21. <% foreach (var item in Model) { %>
22. <tr>
23. <td>
24. <%: Html.ActionLink("编辑", "Edit", new { id=item.ID }) %> |
25. <%: Html.ActionLink("详情", "Detail", new { id=item.ID })%> |
26. <%: Html.ActionLink("删除", "Delete", new { id=item.ID })%>
27. </td>
28. <td>
29. <%: item.ID %>
30. </td>
31. <td>
32. <%: item.Name %>
33. </td>
34. <td>
35. <%: item.Author %>
36. </td>
37. <td>
38. <%: item.Press %>
39. </td>
40. </tr>
41. <% } %>
42. </table>
43. <p>
44. <%: Html.ActionLink("新建", "Create") %>
45. </p>
46. </asp:Content>

```

代码说明：上面代码设置了一个表格来显示数据表“bookInfo”的数据。第 7~20 行设置了表格的第一行，分为 5 列。第 21~41 行通过常用的 foreach 循环语句，在从控制器传递到视图的 Model 数据中，分别读取“bookInfo”的数据表中的 ID 字段（第 29 行）、Name 字段（第 32 行）、Author 字段（第 35 行）和 Press 字段（第 38 行）。其中第 24~26 行使用了 HTML 中的扩展方法 ActionLink，设置“编辑”、“详情”和“删除”链接，用于编辑指定记录或查看该记录的详细信息。第 44 行设置“新建”链接，用于添加新的图书数据。

**05** 视图 Index.aspx 的运行结果如图 16-18 所示。用户单击“编辑”按钮，就会进入图书编辑的页面；如果单击“详情”按钮，可以进入图书详情页面；如果单击“删除”按钮，将进入删除图书的页面；单击“表格”下方的“新建”链接，将进入添加图书的页面。

**06** 选择控制器 HomeController 中的 Edit()方法，然后单击鼠标右键，在弹出的快捷菜单中选择“添加视图”命令。

**07** 打开如图 16-19 所示的“添加视图”对话框。选中“创建强类型视图”复选框，在“视图数据类”下拉列表中找到实体数据类“MvcApplication.Models.BookInfo”。在“视图内容”下拉列表中选择“Edit”。然后选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动在网站根目录“Views”文件夹下的“Home”子文件夹创建一个 Edit.aspx 页面。



图 16-18 Index.aspx 运行结果



图 16-19 “添加视图”对话框

**08** 双击 Index.aspx 文件，在文件中生成的实现代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <h2>Edit</h2>
3. <% using (Html.BeginForm()) {%>
4. <fieldset>
5. <legend>Fields</legend>
6. <div class="display-label">编号</div>
7. <div class="editor-field">
8. <%= Html.TextBoxFor(model => model.ID)%>
9. </div>
10. <div class="display-label">书名</div>
11. <div class="editor-field">
12. <%= Html.TextBoxFor(model => model.Name) %>
13. </div>
14. <div class="display-label">作者</div>
15. <div class="editor-field">
16. <%= Html.TextBoxFor(model => model.Author) %>
17. </div>
18. <div class="display-label">出版社</div>
19. <div class="editor-field">
20. <%= Html.TextBoxFor(model => model.Press) %>
21. </div>
```

```

22. <p>
23. <input type="submit" value="保存" />
24. </p>
25. </fieldset>
26. <% } %>
27. <div>
28. <%: Html.ActionLink("回到列表", "Index") %>
29. </div>
30. </asp:Content>
31.

```

代码说明：第3行引入 Html 的 BeginForm 方法定义表单的开始部分。第4行设置一个验证摘要信息控件 ValidationSummary，当表单的文本框中含有空白输入时，该控件就会在表单页面中输入指定的错误信息。第5~30行设置了一个边框，边框内以8行的方式分别显示了数据表“bookInfo”中的ID标签和字段、Name标签和字段、Author标签和字段以及Press标签和字段。在边框下面，第28行还设置了一个“回到列表”的链接，单击它会返回到 Index.aspx 视图页面。

**09** 视图 Edit.aspx 页面的运行结果如图 16-20 所示。用户可以选择修改书名、作者或出版社的内容，然后单击“保存”按钮，保存修改的内容并打开图 16-18 所示的 Index.aspx 页面。

**10** 选择控制器 HomeController 中的 Create()方法，然后单击鼠标右键，在弹出的快捷菜单中选择“添加视图”命令。

**11** 打开“添加视图”对话框。选中“创建强类型视图”复选框；在“视图数据类”下拉列表中找到实体数据类“MvcApplication.Modes.BookInfo”。在“视图内容”下拉列表中选择“Create”，然后选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的“Home”子文件夹中创建一个 Create.aspx 页面。

**12** Create.aspx 文件中生成的实现代码与 Edit.aspx 极其类似，这里就不再重复介绍。

**13** 视图 Create.aspx 页面的运行结果如图 16-21 所示。用户输入编号、书名、作者和出版社的内容，然后单击“新建”按钮，就会添加一条新的图书数据并打开图 16-18 所示的 Index.aspx 页面。

图 16-20 Edit.aspx 运行结果

图 16-21 Create.aspx 运行结果

**14** 选择控制器 HomeController 中的 Detail()方法，然后单击鼠标右键，在弹出的快捷菜单中

选择“添加视图”命令。

**15** 打开“添加视图”对话框。选中“创建强类型视图”复选框；在“视图数据类”下拉列表中找到我们的实体数据类“MvcApplication.Modes.BookInfo”。在“视图内容”下拉列表中选择“Details”，然后选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的“Home”子文件夹中创建一个 Detail.aspx 页面。

**16** 双击 Detail.aspx 文件，在文件中生成的实现代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <h2>Detail</h2>
3. <fieldset>
4. <legend>Fields</legend>
5. <div class="display-label">编号</div>
6. <div class="display-field"><%= Model.ID %></div>
7. <div class="display-label">书名</div>
8. <div class="display-field"><%= Model.Name %></div>
9. <div class="display-label">作者</div>
10. <div class="display-field"><%= Model.Author %></div>
11. <div class="display-label">出版社</div>
12. <div class="display-field"><%= Model.Press %></div>
13. </fieldset>
14. <p>
15. <%= Html.ActionLink("编辑", "Edit", new { id=Model.ID }) %> |
16. <%= Html.ActionLink("回到列表", "Index") %>
17. </p>
18. </asp:Content>
```

以上代码中，第 3~13 行设置了一个边框，边框内以 3 行的方法分别显示了数据表“bookInfo”中的 ID、Name、Author 和 Press 字段的内容；第 15 行和 16 行设置了一个“编辑”链接和“回到列表”链接。

**17** 视图 Create.aspx 页面的运行结果如图 16-22 所示。用户单击“编辑”按钮可以回到图 16-20 所示的 Edit.aspx 页面。单击“回到列表”按钮，可以回到前面图 16-18 所示的 Index.aspx 页面。

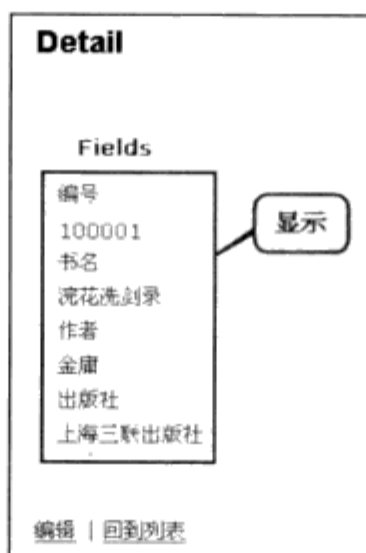


图 16-22 Detail.aspx 运行结果

**18** 选择控制器 HomeController 中的 Delete()方法，然后单击鼠标右键，在弹出的快捷菜单中选择“添加视图”命令。

**19** 打开“添加视图”对话框。选中“创建强类型视图”复选框；在“视图数据类”下拉列表中找到实体数据类“MvcApplication.Modes.BookInfo”。在“视图内容”下拉列表中选择“Delete”，然后选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的“Home”子文件夹中创建一个 Delete.aspx 页面。

**20** Delete.aspx 文件中生成的实现代码与 Detail.aspx 极其类似，这里就不再重复介绍。

**21** 视图 Delete.aspx 文件运行的结果如图 16-23 所示。用户单击“删除”按钮，将显示的该条数据从数据库表中删除，并回到图 16-18 所示的 Index.aspx 页面。

The screenshot shows a web page titled "Delete". Below the title is a confirmation message: "您确定要删除吗?". Underneath is a section labeled "Fields" containing the following information:

编号	100001
书名	浣花洗剑录
作者	金庸
出版社	上海三联出版社

At the bottom of the form, there are two buttons: "删除" (Delete) and "回到列表" (Return to List).

图 16-23 Delete.aspx 运行结果

## 16.2.8 行为过滤器

在 ASP.NET MVC 应用程序中，控制器定义的行为方法都和可能的用户交互（单击一个链接或提交一个表单等）具有一对一的关系。例如，当用户单击一个链接，一个请求被路由到指定的控制器，相应的行为方法就会被调用。

然而，有时可能需要在调用行为方法之前或之后执行逻辑操作。为了支持这样的操作，ASP.NET MVC 提供了行为过滤器，行为过滤器提供了一种向控制器行为方法中添加前行为（Pre-action）和后行为（Post-action）的方法。

行为过滤器是 ASP.NET MVC 中一个非常有用的扩展，它最初是在 Preview 2 版本中出现的，允许在对 MVC 控制器的请求中注入拦截代码，这些代码在 Controller 和它的 Action 方法执行的前后执行，这样就可以以一种非常干净、声明的方式轻松地封装和重用功能。行为过滤器继承自 `ActionFilterAttribute` 类。可以通过行为过滤器特性来标记任何行为方法或控制器以表明行为过滤器应用于该方法或该控制器内的所有方法。

ASP.NET MVC 2 提供了四种类型的行为过滤器。

- 授权（Authorize）。该过滤器用来限制进入控制器或控制器行为。

- 处理错误 (HandleError)。该过滤器用来指定一个行为, 这个行为用来处理某个行为方法中抛出的异常。
- 缓存输出 (OutputCache)。该过滤器用来为行为方法提供输出缓存。
- 自定义过滤器。自定义过滤器允许开发人员自己创建行为过滤器以执行所需要的功能。比如自定义的过滤器包括日志、权限、本地化以及认证功能。

### 1. Authorize 过滤器

很多 Web 应用程序要求用户在使用限制的内容之前必须先进行登录。为了限制进入某个 ASP.NET MVC 视图, 可以限制对渲染该视图的行为方法的进入。ASP.NET MVC 提供的 Authorize 过滤器特性可以实现这个功能。

当使用 Authorize 特性标记一个行为方法时候, 该方法的进入就被限制为被认证和被授权的用户。如果一个控制器被 Authorize 过滤器特性所标记, 则该控制器内所有行为方法的访问都具有限制性。

Authorize 特性让开发人员指明行为方法被限制于那些预定义的角色或用户, 这样程序员就具有了允许用户查看网站内页面的最大的控制权限。

如果一个未被授权的用户试图进入被 Authorize 特性标记的行为方法, MVC 框架就会抛出 401 HTTP 状态码。如果网站被配置为使用 ASP.NET 表单认证, 401 状态码将会把用户导航到登录页面。

Authorize 过滤器的实现依赖于 AuthorizeAttribute 类。该类设置了两个属性, 分别是 Users 和 Roles, 分别表示成员的用户名和角色。通过对这两个属性的控制完成成员和角色的验证功能。

**【例 16-7】**在“例 6-6”的基础上完成指定页面的成员管理功能, 不是指定的用户无法登录访问该页面。

**01** 双击打开项目中“Controller”文件夹下的“HomeController”控制器文件, 并添加以下代码。

```
1. [Authorize]
2. public ActionResult Authorize(){
3. ViewData["Message"] = "该页面只有注册用户才能访问";
4. return View();
5. }
```

代码说明: 第 1 行使用 [Authorize] 属性实现对访问视图的用户验证。第 2 行添加了一个动作方法 Authorize 对应于视图页面 Authorize.aspx。如果运行该页面, 由于 [Authorize] 属性设置在 Authorize 的方法上, 所以 Authorize.aspx 只有注册用户登录后才可以访问, 因此网站将转移到登录页面。

**02** 选择 Authorize 方法, 然后单击鼠标右键, 在弹出的快捷菜单中选择“添加视图”命令。

**03** 打开“添加视图”对话框。选中“选择母版页”复选框, 在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的“Home”子文件夹中创建一个 Authorize.aspx 页面。

**04** 双击打开“Authorize.aspx”, 在文件中编写如下代码。

```
1. <%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits="System.Web.Mvc.ViewPage" %>
2. <asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
```

```

3. 验证
4. </asp:Content>
5. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
6. <h2><%: ViewData["Message"] %></h2>
7. </asp:Content>

```

代码说明：第 1 行设置 @ Page 指令，要注意 Inherits 继承属性必须设置为“System.Web.Mvc.ViewPage”。第 6 行通过 ViewData 数据字典在视图中接受控制器中传递的值。

**05** 运行 Authorize.aspx 页面，我们会发现无法进入该页面，而是转到了如图 16-24 所示的 LogOn.aspx 的登录页面。因为我们还不是注册用户，所以要单击“注册”链接注册一下。

**06** 进入如图 16-25 所示的“Register.aspx”注册页面，输入注册信息后，单击“注册”按钮。

图 16-24 登录页面

图 16-25 注册页面

图 16-26 验证页面

**07** 当我们完成注册后，会进入了项目的首页 Index.aspx。此时再运行 Authorize.aspx 页面，我们就可以进入如图 16-27 所示的该页面。

上面需要验证的 Authorize.aspx 页面，任何注册用户均可以登录后访问，如果需要只有指定的用户名才能访问，可以设置如下的 Authorize 属性。

```
[Authorize (Users=“用户名 1, 用户名 2, 用户名 3.....”)]
```

上面代码中 User 参数可以设置一个或多个已注册的用户名，只有这些用户才能登录后访问需要验证的页面。

如果需要指定的用户名太多，可以将这些用户设置为一类角色，通过如下的角色参数来设置。

```
[Authorize (Roles=“角色名 1, 角色名 2, 角色名 3”)]
```

上面代码中，Roles 参数可以设置一个或多个已注册的角色名，只有属于这些角色的注册用户才能登录访问需要验证的页面。

2. OutputCache 过滤器

OutputCache 过滤器实现的主要功能是借用了 ASP.NET 2.0 中的页面缓存机制,实现 ASP.NET MVC 2 网站中指定页面的缓存,从而提高网站的性能。

一般来说,在控制器或者控制器内的行为方法上,设置[OutputCache]属性,便在该属性中指定相关的缓存参数;还可以在视图页面中,直接设置缓存参数;或者在 Web.config 配置文件中设置相关的缓存参数。

OutputCache 过滤器的实现是通过 OutputCachAttribute 类,该类包括多个属性,这些属性的使用说明如表 16-9 所示。

表 16-9 OutputCachAttribute 类的属性

属性名称	说明
Duration	输出缓存的时间,单位是秒
Location	指定输出缓存的位置。其属性是 OutputCacheLoaction 的枚举值,它们分别 Any、Client、Downstream、None、Server 和 ServerAnd Client。是默认是 Any
Shared	它是一个布尔值,用来决定输出是否页面进行共享。默认是 false
VaryByCustom	设置自定义输出缓存请求的任意文本
VaryByHeader	设置用户已改变缓存输出的所有以逗号分开的 HTTP 标头的列表
VaryByParam	设置影响缓存的参数列表,这些参数由 HTTP GET 或 HTTP POST 接收。以逗号分开的字符串对应于 GET 方法的查询字符串值或 POST 方法的参数值
VaryByContentEncoding	设置用于改变输出缓存的 ContentEncoding 标头列表。以逗号分开的字符串被用于不同的输出缓存
CacheProfile	定义该缓存的名称。一般在配置文件 Web.config 中设置该属性,就可以在页面,或者控制器中引用该名称来设置缓存
CacheSettings	获取缓存参数 OutputCacheParameters 类的实例化对象
NoStore	一个布尔值,设置是否阻止缓存信息的二级缓存
SqlDependency	设置输出缓存的相关数据库和数据表

【例 16-8】在“例 6-6”的基础通过使用 OutputCache 过滤器实现视图页面的缓存功能。

01 双击打开项目中“Controller”文件夹下的“HomeController”控制器文件并添加以下代码。

```
1. public ActionResult OutputCache(){
2. ViewData["Message"] = "当前时间是: " + DateTime.Now.ToString();
3. return View();
4. }
```

代码说明:第 1 行添加了一个 OutputCache 动作方法。第 2 行通过 ViewData 字典属性储存了系统当前时间。

02 选择 OutputCache 方法,然后单击鼠标右键,在弹出的快捷菜单中选择“添加视图”命令。

03 打开“添加视图”对话框。选中“选择母版页”复选框,在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的

“Home”子文件夹中创建一个 OutputCache.aspx 页面。

**04** 双击打开“OutputCache.aspx”文件，在代码中编写关键代码如下。

```
1. <asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
2. 验证
3. </asp:Content>
4. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
5. <h2><%= ViewData["Message"] %></h2>
6. </asp:Content>
```

代码说明，第 5 行通过 ViewData 数据字典在视图中接受控制器中传递的值。

**05** 运行 OutputCache.aspx 页面显示如图 16-27 的效果。如果用户多次单击浏览器的“刷新”按钮，则系统的当前时间将会不断地更新。



图 16-27 运行结果

**06** 在“HomeController”控制器中的 OutputCache 动作方法上添加如下一行代码。

```
[OutputCache(Duration = 10, VaryByParam = "None")]
```

以上代码在动作方法 OutputCache 上设置[OutputCache]属性，使得 OutputCache.aspx 页面具有缓存功能，指定属性 Duration 的值为 10 秒，属性 VaryByParam 的值为 none。

**07** 再次运行 OutputCache.aspx 页面，此时如果单击浏览器中的“刷新”按钮，系统的当前时间将不会马上更新。因为 OutputCache.aspx 页面被设置为缓存 10 秒，只有 10 秒后，如果再单击浏览器中的“刷新”按钮，系统时间才会被更新。

**08** 还可以直接在 OutputCache.aspx 页面设置缓存，双击打开“OutputCache.aspx”文件，在 @Page 指令下面添加一行代码如下。

```
<% @OutputCache Duration = "10" VaryByParam = "None" %>
```

以上代码通过 @ OutputCache 指令同样设置该页面 OutputCache 属性，使得该页面也具有了缓存的功能。

为了方便开发人员修改缓存特性中的相关属性参数，可以在配置文件 Web.config 中的 <outputCacheProfiles></outputCacheProfiles> 节，设置如下代码。

```
1. <caching>
2. <outputCacheSetting>
3. <outputCacheProfiles>
4. <add name="OutputCache" duration="10" varyByParam="none" />
5. </outputCacheProfiles>
6. </outputCacheProfiles>
7. </outputCacheSetting>
8. </caching>
```

代码说明：第 1 行和第 5 行的< caching>和</ caching>节点必须位于 Web.config 文件的< system.web>和</ system.web>节点之中。第 3~5 行设置缓存。其中第 4 行设置缓存的名称 OutputCache、输出缓存的时间 10 和没有缓存参数。这样在控制器或者相关的缓存页面中，就可以通过 outputCacheProfile 属性直接引用了。

在“OutputCache.aspx”视图页面@Page 指令下面添加一行代码如下。

```
<% @OutputCache CacheProfile="OutputCache" %>
```

以上代码仍然通过 OutputCache 属性直接在页面中设置缓存，但是通过设置 CacheProfile 属性，引用了配置文件中名称为“OutputCache”的缓存，便于开发人员在配置文件中修改缓存参数，不需要重新编译页面或控制器，可以提高网站的性能。

### 3. HandleError 过滤器

通过 ASP.NET MVC 2 应用程序中的 HandleError 过滤器可以指定如何处理行为方法中抛出的异常。默认情况下，如果一个被标记了 HandleError 属性的行为方法抛出任何异常，MVC 框架都会显示应用程序目录下“View”文件夹中“Shared”文件夹内的“Error.aspx”视图。

HandleError 过滤器的实现是通过 HandleErrorAttribute 类，该类提供了如下几个属性：

- ExceptionType: 指定过滤器要处理的异常类型。如果该属性未被指定，则过滤器处理所有的异常。
- View: 指定要显示的视图的名称。
- Master: 指定要使用的母版视图的名称。
- Order: 指定过滤器被应用的顺序。

其中，Order 属性用来决定哪个 HandleError 过滤器来处理一个异常。可以把 Order 属性设置为整数值以指定优先级，整数值的范围是从-1 开始到任何正整数，整数值越高优先级越低，属性 Order 遵循如下规则。

- 应用于控制器的过滤器自动应用于控制器的每个方法。
- 当应用于控制器的过滤器与应用于行为方法的过滤器具有相同的 Order 值时，应用于控制器的过滤器优先级高。
- 如果没有指定 Order 值，Order 值是-1，这就意味着该过滤器比其他 Order 值不是-1 的过滤器具有较高的优先级。
- 错误处理停止后，第一个 HandleError 过滤器就会被调用。

**【例 16-8】**在“例 6-6”的基础上通过使用 HandleError 过滤器实现处理控制器动作方法的异常情况。

**01** 双击打开项目中“Controller”文件夹下的“HomeController”控制器文件，并添加以下代码。

```
1. public ActionResult HandleError(){
2. throw new Exception("运行出现异常!");
```

```
3. }
```

代码说明：第 1 行添加了一个 `HandleError` 的动作方法。第 2 行人为抛出一个运行出现的异常。

**02** 选择 `HandleError` 方法，然后单击鼠标右键，在弹出的快捷菜单中选择“添加视图”命令。

**03** 打开“添加视图”对话框。选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的“Home”子文件夹中创建一个 `HandleError.aspx` 页面。

**04** 运行 `HandleError.aspx` 页面，由于在执行动作方法 `HandleError` 中抛出了异常，将会显示如图 16-28 所示的界面。

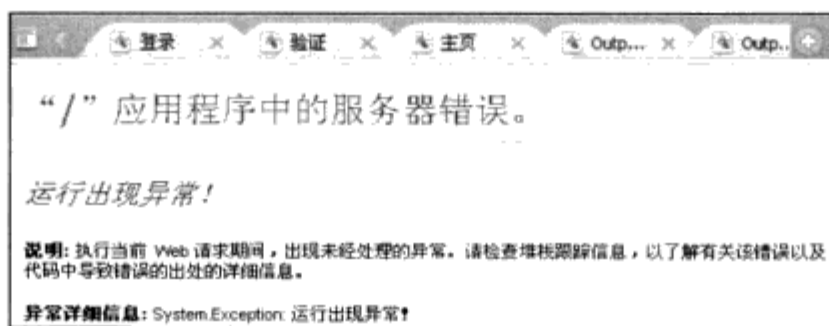


图 16-28 `HandleError.aspx` 页面运行结果

**05** 以上的运行结果对用户来说感觉会很不爽，所以有必要做一些处理。在“HomeController”控制器中的 `HandleError` 动作方法上添加如下一行代码。

```
[HandleError]
```

这行代码使用 `[HandleError]` 属性设置了动作方法 `HandleError` 的异常处理页面为“Share”目录下的个性化异常处理页面 `Error.aspx`。

**06** 在配置文件 `Web.config` 中的 `<system.web>` 和 `</system.web>` 节点中，添加代码如下。

```
<customErrors mode="On"></customErrors>
```

以上代码将异常处理的模式设置为自定义。

**07** 再次运行 `HandleError.aspx` 页面，此时就会打开如图 16-29 所示的人性化异常处理页面 `Error.aspx`。

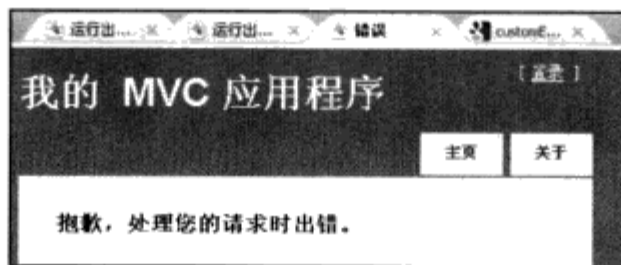


图 16-29 出现异常的运行结果



提示

将 `[HandleError]` 属性设置在控制器上方，如果网站在执行控制器中出现任何的异常，同样会打开个性化的异常处理页面 `Error.aspx`。

### 16.2.9 表单数据绑定

在 ASP.NET MVC 2 的框架中, 将视图中的数据传递到控制器中, 主要通过发送表单的方式来实现。

#### 1. Request.Form 读取表单数据

Request.Form 用于获取 ASP.NET 中窗体变量集合。它的基本语法如下:

```
变量名=Request.Form("element")
Name= Request.Form("name");
```

代码中参数“element”指定集合要检索的表格元素的名称。Form 集合按请求正文中参数的名称来索引。Request.Form(element)的值是请求正文中所有 element 值的数组, 通过调用 Request.Form(element).Count 来确定参数中值的个数。如果参数未关联多个值, 则计数为 1。如果找不到参数, 计数为 0。

要引用有多个值的表格元素中的单个值, 必须指定 index 值。index 参数可以是 1 到 Request.Form(element).Count 中的任意数字。如果引用多个表格参数中的一个, 而未指定 index 值, 返回的数据将是以逗号分隔的字符串。

在使用 Request.Form 参数时, Web 服务器将分析 HTTP 请求正文并返回指定的数据。如果应用程序需要未分析的表格数据, 可以通过调用不带参数的 Request.Form 访问该数据。

下面的代码演示如何使用 Request.Form 读取表单数据。首先定义一个 Studnet 学生类。

```
1. public class Student{
2. public string Name{get;set}
3. public int Age{get;set}
4. }
```

上面的代码中, 定义了 Student 类的两个属性, 分别是 Name 和 Age, 并设置了它们的自动实现属性。

然后在视图页面中, 设置了如下的表单, 代码如下。

```
1. <% using (Html.BeginForm("RequestForm", "Home")){ %>
2. Name: <% =Html.TextBox("Name")%>

3. Age: <% =Html.TextBox("Age")%>

4. <input type="submit" name="submit" value="RequestForm" />

5. <%} %>
```

在上面的表单中, 处理表单数据的动作方法为控制器中 RequestForm 方法, 发送的表单数据是两个文本框“Name”和“Age”中的数据。

最后在控制器的 RequestForm 方法中实现获取表单数据的代码如下。

```
1. [AcceptVerbs(HttpVerbs.Post)]
2. public ActionResult RequestForm() {
3. Student stu = new Student ();
4. stu.Name =Request.Form["Name"];
5. stu.Age = Request.Form["Age "];
6. return View(stu);
7. }
```

代码说明：第 1 行设置[AcceptVerbs (HttpVerbs.Post)]属性，表示下面的方法只接受用户通过 Post 方法发送表单数据。第 4 和第 5 行通过 Request.Form 来分别读取两个文本框“Name”和“Age”中的数据，然后得到 Student 类的实例化对象 stu。



提示

Request.Form 是接收 post 方法的对象。所以在客户端的表单发送中，一定要注明 post 方法，如<form method=post>。

## 2. FormCollection 读取表单数据

FormCollection 是用于获取 Form 表单中元素的集合。以上面创建的 Student 类为例，以 FormCollection 获取表单数据在视图页面中的表单代码如下。

```
1. <% using (Html.BeginForm("FormCollection", "Home")){ %>
2. Name: <% =Html.TextBox("Name")%>

3. Age: <% =Html.TextBox("Age")%>

4. <input type="submit" name="submit" value=" FormCollection" />

5. <%} %>
```

以上代码中，处理表单数据动作方法为控制器中的 FormCollection 方法，发送的表单数据仍然是两个文本框“Name”和“Age”中的数据。代码中的粗体字显示与 RequestForm 表单的区别。

然后在控制器的 FormCollection 方法中实现获取表单中的数据，代码如下。

```
1. [AcceptVerbs(HttpVerbs.Post)]
2. public ActionResult FormCollection (FormCollection formCollection){
3. Student stu = new Student ();
4. stu.Name = formCollection["Name"];
5. stu.Age = formCollection ["Age "];
6. return View(stu);
7. }
```

代码说明：第 1 行设置[AcceptVerbs (HttpVerbs.Post)]属性，表示下面的方法只接受用户通过 Post 方法发送表单数据。第 2 行 FormCollection 方法中传入了 FormCollection 方法类型的参数 formCollection，该参数会自动绑定表单中所有的数据。第 4 和第 5 行通过 formCollection 参数可以分别读取两个文本框“Name”和“Age”中的数据，然后就可以得到 Student 类的实例化对象 stu。

在 ASP.NET MVC 2 项目中，通过 FormCollection 可以读取表单中指定的数据，借助控制器中的 UpdateModel 方法或 TryUpdateModel 方法，可以非常方便的将数据对象中相关属性的数据进行更新。比如下面的示例代码。

```
1. [AcceptVerbs(HttpVerbs.Post)]
2. public ActionResult FormCollection (FormCollection formCollection){
3. Student stu = new Student ();
4. stu.Name = "NewName";
5. stu.Age = "NewAge";
6. UpdateModel(stu,new []{"Name"});
7. return View(stu);
8. }
```

以上代码中，第 4、5 行给 stu 两个属性赋值。第 6 行使用 UpdateModel 方法，第 1 个参数指定需

要更新数据的对象为 stu，第 2 个参数指定需要被更新的对象中的属性名称，即只更新 stu 对象中的“Name”属性，因此上面代码的执行结果是，只读取表单发送过来的 Name 文本框中的数据。

### 3. 直接读取表单数据对象

还是以上面创建的 Student 对象为例，直接读取表单数据对象时在视图页面中的表单代码如下。

```
1. <% using (Html.BeginForm("Student ", "Home")){ %>
2. Name: <%=Html.TextBox("Name")%>

3. Age: <%=Html.TextBox("Age")%>

4. <input type="submit" name="submit" value=" Student" />

5. <%} %>
```

以上代码中，处理表单数据动作方法为控制器中的 Student 方法，发送的表单数据仍然是两个文本框“Name”和“Age”中的数据。代码中的粗体字显示与前两种读取方式的区别。

然后在控制器的 Student 方法中实现读取取表单中的数据，代码如下。

```
1. [AcceptVerbs(HttpVerbs.Post)]
2. public ActionResult Student (Student stu){
3. return View(stu);
4. }
```

上面代码中，Student 方法中传入了 Student 类型的参数，其内部读取了两个文本框“Name”和“Age”中的数据，直接得到 Student 类的实例化对象 stu。这里要注意的是直接读取表单对象时，发送表单的文本框名称必须与数据对象的熟悉名称相一致。



提示

在直接读取表单数据对象后，调用 UpdateModel 方法来更新数据时，如果开发人员不更新全部的属性数据，则入侵者可以发送包括所有属性的表单数据，给网站带来安全隐患。

#### 16.2.10 ASP.NET MVC 中的数据传递

在 ASP.NET MVC 中，数据传递指控制器和视图之间的数据交互，包括两个方向上数据的交互，一个是将控制器中设置的数据传递到视图中，在视图中如何显示这些数据，另一个是将视图中的数据传递到控制器中，如何在控制器中读取、处理这些数据。

##### 1. 使用 ViewData 传递数据

在 ASP.NET MVC 2 框架中，所有的控制器必须继承 Controller 类，而 Controller 类又是 ControllerBase 的子类。根据 ControllerBase 类中的 ViewData 属性，可以在控制器中的相关动作方法中设置该视图数据字典 (ViewDataDictionary) 的值，在默认新建“MvcApplication”项目的“Home”控制器 Index 动作方法中，对 ViewData 视图数据字典设置如下代码。

```
ViewData["Message"]="欢迎学习 ASP.NET MVC! ";
```

而要在视图页面 Index.aspx 中，读取上述控制器中被设置的 ViewData 数据，也就是说实现数据从控制器到视图的传递，只需要设置如下代码。

```
<%=Html.Encode(ViewData["Message"])%>
```

从上述代码中可以看出，在视图页面 Index.aspx 中只需要通过读取该页面 ViewPage 类中的 ViewData 属性，即可获得控制器中所设置的 ViewData 属性值。而 ViewData["Message"] 实际上是 this.ViewData["Message"] 的简化形式。

## 2. 使用 TempData 传递数据

根据 ControllerBase 类中的 TempData 属性，同样可以在控制器中的相关动作方法中设置该 TempData 属性的值。在默认新建“MvcApplication”项目的“Home”控制器的 Index 动作方法中，可以对 TempData 属性设置如下代码。

```
TempData["Message"] = "欢迎学习 ASP.NET MVC!";
```

而要在视图页面 Index.aspx 中，读取上述控制器中被设置的 TempData 数据，也就是说实现数据从控制器到视图的传递，只需要设置如下代码。

```
<%=Html.Encode(TempData["Message"])%>
```

从上述代码中可以看出，在视图页面 Index.aspx 中只需要通过读取该页面 ViewPage 类中的 TempData 属性，即可获得控制器中所设置的 TempData 属性值。

需要说明的是，ViewData 和 TempData 是两个完全不同的数据类型，ViewData 的数据类型是 ViewDataDictionary 类的实例化对象，而 TempData 的数据类型则是 TempDataDictionary 类的实例化对象；ViewData 只能在一个动作方法中或者多个页面中设置和读取，只对当前的视图页面有效，而 TempData 则可以在多个动作方法中或者多个页面中设置和读取。

## 3. 使用 Model 传递数据

通过设置、读取上面介绍的 ViewData 和 TempData 属性，可以将数据从控制器中传递到视图中，但需要说明的是，在视图中读取这些数据字典中的数据时，由于不能使用强类型数据，所以不利于代码的书写和查错。

通过在控制器的 View 方法中，传递实例化的对象，可以将该对象传递到视图中。当在视图中读取该对象的某些属性时，由于是强类型的，所以书写代码时具有代码智能感知功能，有利于代码书写与查错。

当在控制器 View 方法中传递实例化对象时，控制器就会将 ViewDataDictionary 类的实例化对象的 Model 属性设置成为需要被传递的对象。在视图中，只需要读取 ViewPage 类中的 Model 属性，就可以获得控制器中所设置的实例化对象。

例如在新建项目“MvcApplication”的“HomeController”控制器中，设置如下代码：

```
1. public ActionResult bookInfo(){
2. bookInfoDataContext bdc=new bookInfoDataContext();
3. var model=dbc.bookInfos;
4. return View(model);
5. }
```

以上代码，将 dbc.bookInfos 对象设置为需要被传递的对象，传入 View 方法中，完成了在控制器端设置需要被传递对象的工作。

然后为 bookInfo 方法使用“添加视图”对话框完成“bookInfo”视图的创建。在视图中通过循

遍历 model 对象就能得到其中的数据。这里需要说明的是，在视图页面中设置了 ViewPage 的类型为 `IEnumerable<实体数据模型命名空间.bookInfo>`，正是由于设置了该代码，才使得 ViewPage 的 Model 属性为可遍历的 bookInfo 实例化对象。



提示

TempData 实际上保存在 Session 中，控制器每次执行请求时都会从 Session 中获取 TempData 数据并删除该 Session。需要注意的是，TempData 数据只能在控制器中传递一次，其中的每个元素也只能被访问一次，访问之后会被自动删除。

### 16.2.11 传递多个数据对象

在用 Model 传递数据的时候，如果需要同时传递多个对象，则需要新建一个专门的类，来封装这些需要被传递的多个对象。

**【例 16-6】** 创建一个 ASP.NET MVC 项目，使用第 8 章“例 8-9”中用到的“Northwind”数据库的“Products”和“Categories”数据表。在项目的“Home”文件夹中创建一个视图“ProductCategories.aspx”显示 NorthWind 数据库中产品目录和对应该目录下产品名称的信息。

**01** 打开 Visual Studio 2010，创建一个 ASP.NET MVC 2 的项目“例 16-6”。

**02** 在“解决方案资源管理”窗口中的“例 16-6”项目的“Models”文件夹上单击鼠标右键，在弹出快捷菜单中选择“添加”|“新建项”命令。

**03** 打开“添加新项”对话框。选择“已安装模板”下的“Visual C#”，并在模板文件列表中选“ADO.NET 实体数据模型”，然后在“名称”文本框输入“Model.edmx”，最后单击“添加”按钮。

**04** 打开实体数据模型向导——“选择模型内容”对话框，选择“从数据库生成”，表明 ADO.NET 实体框架从数据库直接生成实体数据模型，然后单击“下一步”按钮。

**05** 打开“选择你的数据连接”对话框。选择“northwnd.mdf”数据库，选中“将 Web.Config 中的实体连接设置另存为”复选框，单击“下一步”按钮。

**06** 弹出“选择数据库对象”对话框，选择“表”、“Products”和“Categories”，然后，单击“完成”按钮。

**07** 右键单击“解决方案资源管理器”中“Models”文件夹，在弹出的快捷菜单中选择“添加”|“类”命令。

**08** 打开“新建项目”对话框，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选“类”，然后在“名称”文本框输入“ProductModel.cs”，最后单击“确定”按钮。

**09** 双击打开“ProductModel.cs”文件，在文件中编辑代码如下。

```
1. public class ProductModel{
2. public IEnumerable<Categories> categories;
3. public IEnumerable<Products> products;
4. }
```

代码说明：第 1 行专门创建了一个类 ProductModel 来封装 Products 和 Categories。代码很简单，其中设置了两个属性 Products 和 Categories，以便在视图中遍历读取 Products 和 Categories 的相关属性。

**10** 双击打开项目中“Controller”文件夹下的“HomeController”控制器文件并添加以下代码。

```
1. public ActionResult ProductCategories(){
2. northwndEntities ne = new northwndEntities();
3. var model = new ProductModel();
4. model.categories = ne.Categories;
5. model.products = ne.Products;
6. return View(model);
7. }
```

代码说明：第 1 行定义一个名为 ProductCategories 的动作方法。第 2 行实例化 NorthWind 数据库实体数据模型对象 ne。第 3 行创建需要被传递的对象 Model。第 4、5 行设置 ProductModel 类的两个属性给 model。第 6 行将实例化对象 model 传入 View 方法中，这样就完成了在控制器中设置需要被传递多个实例化对象的工作。

**11** 选择 ProductCategorie 方法，然后单击鼠标右键，在弹出的快捷菜单中选择“添加视图”命令。

**12** 打开“添加视图”对话框，选中“创建强类型视图”复选框，在“视图数据类”下拉列表中找到我们的实体数据类“ProductModel”。在“视图内容”下拉列表中选择“List”，然后选中“选择母版页”复选框，在下面的文本框中输入母版页的路径以及内容占位符的 ID。ASP.NET MVC 2 就会自动的在网站根目录“Views”文件夹下的“Home”子文件夹中创建一个 ProductCategories.aspx 页面。

**13** 双击 ProductCategories.aspx 文件，编辑关键代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <h2>产品目录</h2>
3. <table>
4. <tr><th></th></tr>
5. <% foreach (var item in Model.categories) { %>
6. <tr>
7. <td>
8. <%=item.CategoryName%>

9. <% foreach (var item1 in Model.products){ %>
10. <%if (item.CategoryID == item1.CategoryID){ %>
11. <%=item1.ProductName%> &~
12. <% } } %>
13. </td>
14. </tr>
15. <% } %>
16. </table>
17. </asp:Content>
```

上面代码中,通过一个表格来显示产品目录和该目录下的产品名称。其中第5~13行设置 `foreach` 循环语句遍历数据 `Model.Categories`。第8行获取产品目录的目录名称;第9~12行嵌套设置 `foreach` 循环语句遍历数据 `Model.products`。第10行和第11行获取指定产品目录下的产品名称。这里需要说明的是,由于 `Model` 属性被设置为 `ProductModel` 类的实例化对象,所以在视图 `ProductCategories.aspx` 页面中, `ViewPage` 的类型必须从 `IEnumerable<ProductModel>` 修改为 `ProductModel`。

14 运行 ProductCategories.aspx 视图页面，结果如图 16-30 所示。

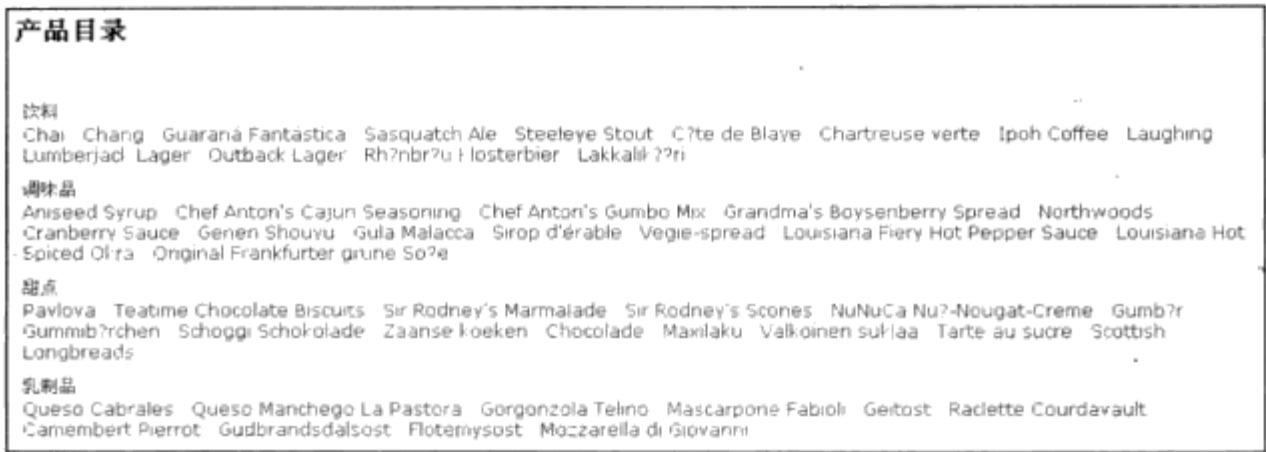


图 16-30 运行结果

### 16.3 上机题

1. 在 Visual Studio 2010 中创建一个名为“StudentMvc”的 ASP.NET MVC 2 的项目，在生成的目录 Models 文件夹中创建数据实体模型“student.edmx”，数据库表使用第 6 章上机题 1 创建的数据库“School”中的“Students”数据表。创建完成后，Models 文件夹中的内容如图 16-31 所示。
2. 上题创建的“StudentMvc”项目中，在“Controllers”文件夹下的“HomeController.cs”控制器文件中，定义一个“Index”动作方法，实现获取“Students”数据表的所有学生的信息传递到“Views”文件夹下“Home”子文件夹下的“Index.aspx”视图页面列表中显示，并在列表中添加“编辑”、“详情”、“删除”和“新建”4 个链接，运行结果如图 16-32 所示。

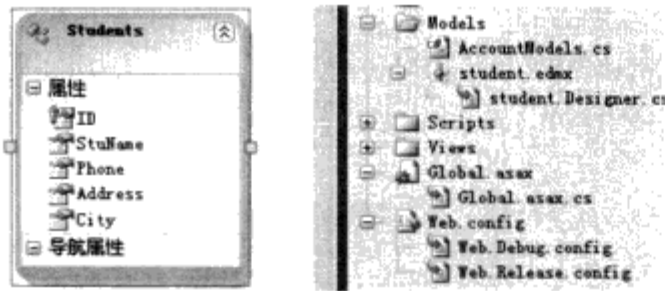


图 16-31 Models 文件夹

学生列表					
学号	姓名	电话	地址	城市	
1	张琴	13256789023	朝阳区外大街	北京	<a href="#">编辑</a>   <a href="#">详情</a>   <a href="#">删除</a>
2	邵飞	13698562314	海淀区清河镇	北京	<a href="#">编辑</a>   <a href="#">详情</a>   <a href="#">删除</a>
3	王宇	15896325689	浦东新区美罗城	上海	<a href="#">编辑</a>   <a href="#">详情</a>   <a href="#">删除</a>
4	董韵琳	13965669856	浦东新区	上海	<a href="#">编辑</a>   <a href="#">详情</a>   <a href="#">删除</a>
5	赵芳	13658784596	海淀区	北京	<a href="#">编辑</a>   <a href="#">详情</a>   <a href="#">删除</a>
<a href="#">新建</a>					

图 16-32 Index 视图显示页面

3. 在上题的“HomeController.cs”控制器文件中，定义两个“Edit”动作方法。一个是获取指定学号学生的信息并进入要求创建的 Edit.aspx 视图页面；另一个是修改指定学生的信息并返回信息 Index.aspx 视图页面。在 Edit.aspx 视图中能显示指定学生的信息并添加“修改”的按钮。运行程序后，在 Index.aspx 视图页面单击“编辑”按钮，进入如图 16-33 所示的 Edit.aspx 视图进行修改操作，最后回到 Index.aspx 视图可看到修改的结果。
4. 在上题的“HomeController.cs”控制器文件中，定义一个“Details”动作方法获取指定学号学生的信息并显示在要求创建的 Details.aspx 视图页面。运行程序后，在 Index.aspx 视图页面单击“详情”按钮，进入如图 16-34 所示的 Details.aspx 视图的界面。
5. 在上题的“HomeController.cs”控制器文件中，定义两个“Create”动作方法，一个是创建一个 Students 对象并传递到要求创建的 Create.aspx 视图，另一个是新建一个学生的信息并返回信

息 Index.aspx 视图页面。在 Create.aspx 视图添加“新建”的按钮，运行程序后，在 Index.aspx 视图页面单击“新建”按钮，进入如图 16-35 所示的 Create.aspx 视图，输入新学生的信息，单击“新建”按钮后，回到 Index.aspx 视图可看到添加的学生信息。

编辑

ID

5

StuName

赵芳芳

Phone

13818765467

Address

浦东新区

City

上海

修改

学生详情

学号

5

姓名

赵芳

电话

13918765489

地址

浦东新区

城市

上海

编辑

回到列表

新建

ID

6

StuName

刘平

Phone

13917657432

Address

汉口区

City

武汉

新建

图 16-33 Edit 视图显示页面    图 16-34 Details 视图显示页面    图 16-35 Create 视图显示页面

6. 在上题的“HomeController.cs”控制器文件中，定义两个“Delete”动作方法，一个是获取指定学号学生的信息，并进入要求创建的 Delete.aspx 视图页面；另一个是删除指定学生的信息并返回信息 Index.aspx 视图页面。在 Delete.aspx 视图中，能显示要删除学生的信息并添加“删除”的按钮。运行程序后，在 Index.aspx 视图页面单击“删除”按钮，进入如图 16-36 所示的 Delete.aspx 视图进行删除操作，最后回到 Index.aspx 视图可以看到删除后的结果。

7. 新建一个 MvcModelBindingDemo 的 MVC 网站。在 View 文件夹下创建 4 个视图文件：FormCollection.aspx、Person.aspx、RequestForm.aspx 显示获取的表单数据，Index.aspx 实现用户输入，如图 16-37 所示。在 Controller 文件夹下创建 HomeController.cs 控制器类文件，在文件中新建三个动作方法，其中 RequestForm 方法实现通过使用 Request.Form 读取表单数据，FormCollection 方法实现通过使用 FormCollection 读取表单数据，Person 方法实现直接读取表单数据对象。

您确认要删除该数据吗?

删除

学号

6

姓名

刘平

电话

13917657432

地址

汉口区

城市

武汉

删除

回到列表

图 16-36 Delete 视图显示页面

用户: 张琴

密码: 585858

RequestForm提交数据

用户: 王宁

密码: 123456

FormCollection提交数据

用户: 高梅

密码: 888888

直接读取对象

图 16-37 用户输入界面

# 第 17 章 网上个人博客

Blog 是 Weblog 的简称，而 Weblog 则是由 Web 和 Log 两个英文单词组合而成。Weblog 就是在网络上发布和阅读的流水记录，通常称为网络日志，简称为网志。它是继 Email、BBS、IM 之后出现的第四种全新的网络交流方式。它绝不仅仅是一种单向的发布系统，而是以网络作为载体，迅速、便捷地发布自己的心得，及时有效地与他人进行交流，再集丰富多彩的个性化展示于一体的综合性平台。

## 17.1 系统分析与设计

本系统是一个使用 LINQ to SQL、母版页、ASP.NET AJAX 和 Web service 等流行技术开发的网上个人博客空间。

### 17.1.1 系统需求分析

本系统的用户包括：普通游客、博客主人。

- ① 普通游客进入网站后可以浏览博客文章和博客相册，并在浏览博客后进行评论。
- ② 博客主人必须在登录页面输入用户名、密码，通过身份验证后，才可以进入博客管理界面。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ③ 如果博主用户无法成功登录必须先进行注册，输入个人信息，然后重新登录。
- ④ 在博客管理界面，博主可以对自己的博客类型进行管理。包括：添加、编辑和删除博客类型。
- ⑤ 博主可以对博客文章进行管理，添加新的博客、编辑现有的博客、将博客放入回收站。
- ⑥ 博主可以暂时不发布博客，而是将文章先保存到草稿箱等以后要发布时，再从草稿箱中提出。还能够将回收站中的博客还原或者将博客从回收站中彻底删除。
- ⑦ 博主还可以上传图片文件，同时可以管理这些上传的图片文件，包括浏览和删除的操作。
- ⑧ 博主能够对游客的评论进行管理，包括查询和删除的操作。
- ⑨ 博主可以浏览自己的服务器的配置情况以便更好地管理网站。

### 17.1.2 系统模块设计

根据上面的系统需求分析，我们将本系统分为如下五个功能模块。

- ① 实体类模块：对应数据库中的六张表，通过 LINQ to SQL 类的形式实现数据库实体的完全映射。
- ② 用户登录模块：实现用户注册后经过登录验证进入后台管理的功能。
- ③ 浏览博客模块：实现对网站前台的博客内容进行浏览的功能。

- ④ 管理博客模块：实现用户对网站博客进行后台管理的操作。
- ⑤ 系统首页：实现系统的首页。

各模块所包含的文件及其功能如表 17-1 所示。

表 17-1 网上博客模块一览表

模块名	文件名	功能描述
实体类模块	BlogDataClasses.dbml/ BlogDataClasses.dbml.layout	LINQ to SQL 类生成 O/R 设计器布局文件
	BlogDataClasses.dbml/ BlogDataClasses.designer.cs	LINQ to SQL 类生成的实体类映射文件
用户登录模块	Register.aspx	设计用户注册页面代码文件
	Register.aspx.cs	用户注册页面后台代码文件
	Manage/ZhuXiao.aspx	设计注销页面代码文件
	Manage/ZhuXiao.aspx.cs	注销页面后台代码文件
	Manage/logout.aspx	设计退出页面代码文件
	Manage/logout.aspx.cs	退出页面后台代码文件
	yanzheng.aspx	设计生成验证码文件
	yanzheng.aspx.cs	生成验证码文件后台代码文件
	Manage/Login.aspx	设计用户登录页面代码文件
	Manage/Login.aspx.cs	用户登录页面后台代码文件
博客浏览模块	Next.aspx	设计浏览博客详情页面代码文件
	Next.aspx.cs	浏览博客详情页面后台代码文件
	Photo.aspx	设计显示博客相册页面代码文件
	Photo.aspx.cs	显示博客相册页面后台代码文件
	PhotoInfo.aspx	设计浏览相册详情页面代码文件
	PhotoInfo.aspx.cs	浏览相册详情页面后台代码文件
	Blog.aspx	设计显示博客页面代码文件
	Blog.aspx.cs	显示博客页面后台代码文件
	Controls/huifu.ascx	设计回复用户控件代码文件
	Controls/huifu.ascx.cs	回复用户控件后台代码文件
	Controls/liuyan.ascx	评论用户控件后台代码文件
	Controls/liuyan.ascx	评论用户控件代码文件
	WebService.asmx.cs	WebService 代码文件
	Manage/Message.aspx	设计访客评论页面代码文件
	Manage/Message.aspx.cs	访客评论页面后台代码文件
博客管理模块	Manage/AddNews.aspx	设计添加博客页面代码文件
	Manage/AddNews.aspx.cs	添加博客页面后台代码文件

(续表)

模块名	文件名	功能描述
博客管理模块	Manage/ AddPhoto.aspx	设计添加相片页面代码文件
	Manage/ AddPhoto.aspx.cs	添加相片页面后台代码文件
	Manage/ CallBack.aspx	设计草稿箱页面代码文件
	Manage/ CallBack.aspx.cs	草稿箱页面后台代码文件
	Manage/EditClass.aspx	设计编辑博客类别代码文件
	Manage/EditClass.aspx.cs	编辑博客类别后台代码文件
	Manage/ManageClass.aspx	设计管理博客类型页面代码文件
	Manage/ ManageClass.aspx.cs	管理博客类型后台代码文件
	Manage/ ManagePhoto.aspx	设计管理相册页面代码文件
	Manage/ ManagePhoto.aspx.cs	管理相册页面后台代码文件
	Manage/ManangeNews.aspx	设计编辑博客页面代码文件
	Manage/ManangeNews.aspx.cs	编辑博客页面后台代码文件
	Manage/PhotoEdit.aspx	设计编辑相片页面代码文件
	Manage/ PhotoEdit.aspx.cs	编辑相片页面后台代码文件
	Manage/ Xman.aspx	设计用户配置页面代码文件
	Manage/ Xman.aspx.cs	用户配置页面后台代码文件
	CuteSoft_Client/NewEdit.aspx	设计添加博客页面代码文件
	CuteSoft_Client/NewEdit.aspx.cs	添加博客页面后台代码文件
	CuteSoft_Client/ NewSave2.aspx	设计草稿箱页面代码文件
	CuteSoft_Client/ NewSave2.aspx.cs	草稿箱页面后台代码文件
	CuteSoft_Client/CuteEditor	CuteEditor 文字编辑器组件文件夹
	Manage/ManageMessage.aspx	设计管理评论页面代码文件
	Manage/ ManageMessage.aspx.cs	管理评论页面后台代码文件
首页显示模块	Site2.Master/	设计母版页代码文件
	Site2.Master/Site2.Master.cs	母版页后台代码文件
	First Page.aspx	设计首页的代码文件
	First Page.aspx.cs	首页后台代码文件
	Manage/ Hou.aspx	设计博客管理页面代码文件
	Manage/ Hou.aspx.cs	博客管理页面后台代码文件
	Html/Footer.htm	后台首页页脚设计代码文件
	Html/ Main.htm	后台首页内容页面设计代码文件
	Html/ top.htm	后台首页页首设计代码文件
	Html/ outlookleft.html	后台首页导航菜单设计代码文件
	Controls/ LeftList.ascx	设计首页左侧栏用户控件代码文件

(续表)

模块名	文件名	功能描述
首页显示模块	Controls/ LeftList.ascx.cs	首页左侧栏用户控件后台代码文件
	Controls/ User.ascx	设计首页链接用户控件代码文件
	Controls/ User.ascx.cs	首页链接用户控件后台代码文件
其他辅助功能	css	放置 CSS 样式文件的文件夹
	face	放置小图标的文件夹
	icon	放置用户头像的文件
	Web.config	网站配置文件
	Global.asax	网站全局配置文件

17.1.3 系统运行演示

运行本系统，首先出现的是如图 17-1 所示的博客网站的首页。



图 17-1 网站首页

在首页中，用户可以选择想要浏览的博客文章，单击博客文章的标题，进入如图 17-2 所示的浏览指定博客的页面。

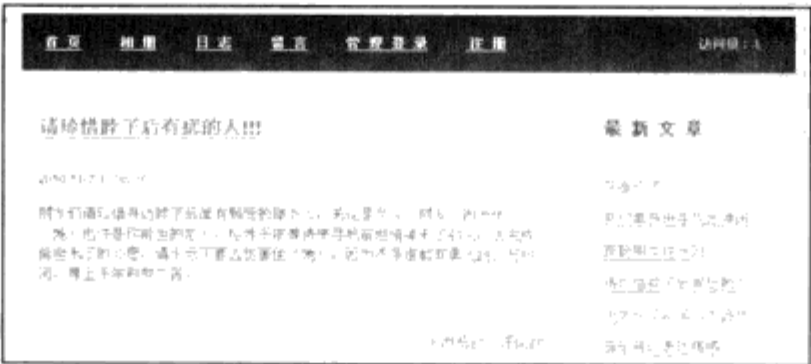


图 17-2 浏览指定博客

用户可以在首页中单击菜单栏上的“相册”按钮，进入如图 17-3 所示的博客相册界面。用户可以在首页中单击菜单栏上的“评论”按钮，进入如图 17-4 所示的评论界面。



图 17-3 博客相册界面

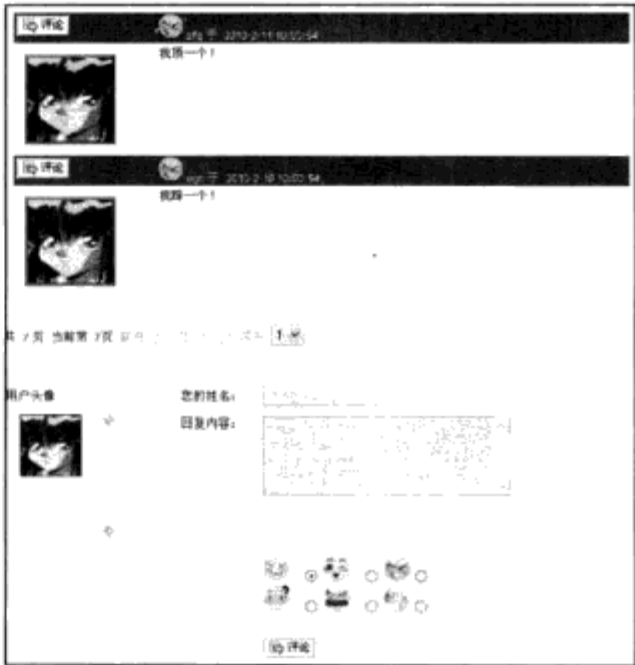


图 17-4 游客评论页面

用户可以在首页中单击菜单栏上的“管理”按钮，进入如图 17-5 所示的后台管理登录界面，输入用户名和密码，单击“登录”按钮。

通过身份验证后，进入如图 17-6 所示的后台管理界面，在该页面中可以对博客进行管理。

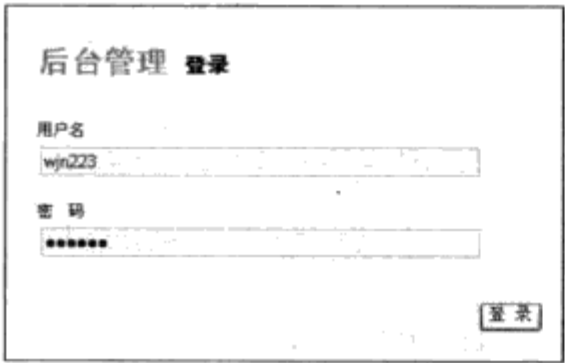


图 17-5 后台管理登录界面



图 17-6 博客管理界面

在上图界面中，单击“添加文章”链接，可以进入添加博客文章的界面，如图 17-7 所示。用户可以选择博客文章的类型、填写文章的标题和内容，最后单击“添加文章”按钮完成博客文章的发布。或者暂时不发布，而是通过单击“添加到草稿箱”按钮将文章保存起来，方便以后再发布。

在博客管理页面中，选择某一博客文章后面的“删除”按钮，可将该博客放入“回收站”。而单击“编辑”链接，进入如图 17-8 所示的博客页面，对所选的博客文章进行编辑，单击“保存修改”进行保存，单击“发布文章”按钮对博客进行发布。



图 17-7 添加博客界面

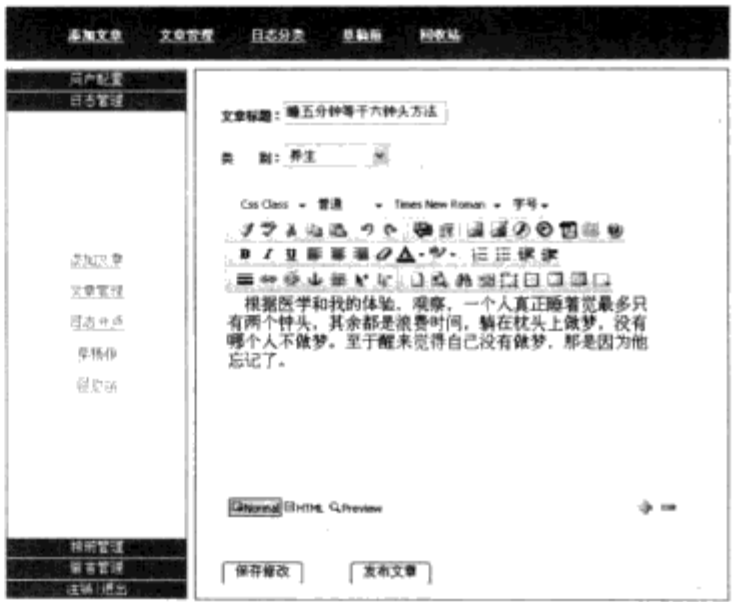


图 17-8 博客编辑界面

在博客管理页面中,选择菜单栏中的“草稿箱”链接,可进入图 17-9 所示的草稿箱页面,在该页面中可以对草稿箱中的博客文章进行发布或者删除到回收站的操作。

在博客管理页面中,选择菜单栏中的“回收站”链接,可进入图 17-10 所示的回收站页面,在该页面中可以对回收站中的博客文章进行还原或者从系统中彻底删除文章的操作。

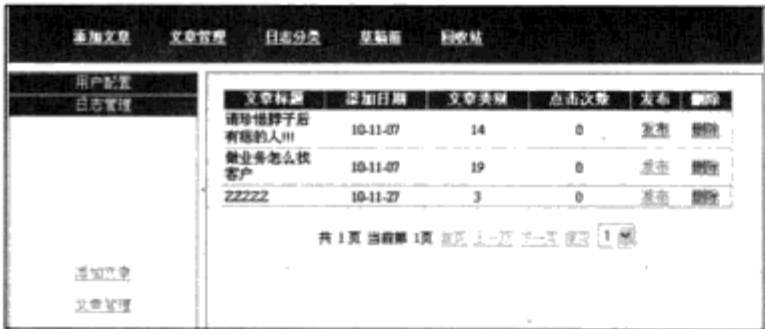


图 17-9 草稿箱界面



图 17-10 回收站界面

在博客管理页面中,选择菜单栏中的“日志分类”链接,可进入图 17-11 所示的博客类别管理的页面,在该页面中可以对博客文章的类别进行添加、编辑和删除的操作。

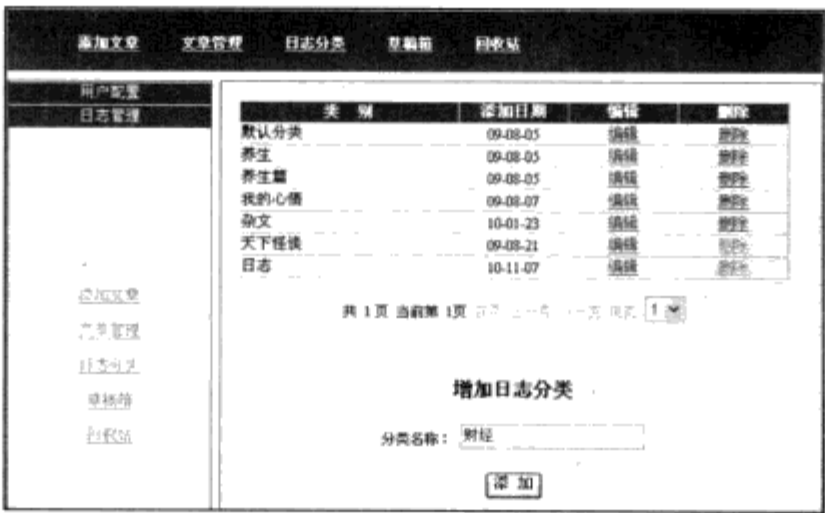


图 17-11 博客类别管理界面

该系统中其他功能界面与上述相似,此处不再赘述,请读者自行运行光盘中的程序代码进行查看。

## 17.2 系统数据库设计

根据系统需求分析和保证数据统一、完整和高效的原则，需要对数据库进行合理的设计。首先在 Sql Server 2005 中建立一个名为“Blog”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [Blog] //创建数据库
USE [Blog] //使用数据库
```

### 17.2.1 数据库表设计

为满足本系统功能的需要，设计数据库表说明如下。

(1) 用户注册表（Register），用来保存系统中注册用户的信息。该表的字段结构如表 17-2 所示。

表 17-2 authors 表结构

字段	中文描述	数据类型	是否为空	备注
user_id	用户编号	int	否	主键
user_name	用户名	varchar(15)	否	
password	用户密码	varchar(15)	否	
sendpassword	确认用户密码	varchar(15)	否	
question	安全提示问题	varchar(30)	否	
answer	安全答案	varchar(30)	否	
emile	电子邮件	varchar(30)	是	
relname	真实姓名	varchar(15)	是	
address	居住地址	varchar(200)	是	
age	年龄	int	否	
sex	性别	char(2)	是	
phone	联系电话	varchar(20)	否	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Register](
 [user_id] [int] IDENTITY(1,1) NOT NULL, //设置主键
 [user_name] [varchar](15) COLLATE Chinese_PRC_CI_AS NOT NULL,
 [password] [varchar](15) COLLATE Chinese_PRC_CI_AS NOT NULL,
 [sendpassword] [varchar](15) COLLATE Chinese_PRC_CI_AS NOT NULL,
 [question] [varchar](30) COLLATE Chinese_PRC_CI_AS NOT NULL,
 [answer] [varchar](30) COLLATE Chinese_PRC_CI_AS NOT NULL,
 [emile] [varchar](30) COLLATE Chinese_PRC_CI_AS NULL,
 [relname] [varchar](15) COLLATE Chinese_PRC_CI_AS NULL,
 [address] [varchar](200) COLLATE Chinese_PRC_CI_AS NULL,
 [age] [int] NOT NULL,
 [sex] [char](2) COLLATE Chinese_PRC_CI_AS NULL,
 [phone] [varchar](20) COLLATE Chinese_PRC_CI_AS NULL
)
```

(2) 博客类别表 (Class) 用来保存网站中博客文章的分类目录信息, 该表的字段结构如表 17-3 所示。

表 17-3 Class 表结构

字段	中文描述	数据类型	是否为空	备注
Class_id	类别编号	int	否	主键
Class_name	类别名称	nvarchar(30)	是	
AddDate	创建时间	datetime	是	

创建上表的语句如下:

```
CREATE TABLE [dbo].[Class](
[Class_id] [int] IDENTITY(1,1) NOT NULL, //设置主键
[Class_name] [nvarchar](30) COLLATE Chinese_PRC_CI_AS NULL CONSTRAINT
[DF_Class_Class_name] DEFAULT (N'默认分类'),
[AddDate] [datetime] NULL
)
```

(3) 博客文章表 (News): 用来保存网站中博客文章的详细信息, 该表的字段结构如表 17-4 所示。

表 17-4 News 表结构

字段	中文描述	数据类型	是否为空	备注
News_id	博客文章编号	int	否	主键
Title	文章标题	nvarchar(30)	是	
Body	文章内容	nvarchar(3000)	是	
AddDate	创建日期	datetime	是	
Click	浏览次数	int	是	
Re	回复次数	int	是	
Class	文章类别	nvarchar(15)	是	
IsSave	是否保存到草稿箱	bit	是	
IsDel	是否保存到回收站	bit	是	

创建上表的语句如下:

```
CREATE TABLE [dbo].[News](
[News_id] [int] IDENTITY(1,1) NOT NULL, //设置主键
[Title] [nvarchar](30) COLLATE Chinese_PRC_CI_AS NULL,
[Body] [nvarchar](3000) COLLATE Chinese_PRC_CI_AS NULL,
[AddDate] [datetime] NULL,
[Click] [int] NULL,
[Re] [int] NULL,
[Class] [nvarchar](15) COLLATE Chinese_PRC_CI_AS NULL,
[IsSave] [bit] NULL,
[IsDel] [bit] NULL
)
```

(4) 博客评论表 (Message): 用来保存网站中对博客文章评论的信息, 该表的字段结构如表 17-5 所示。

表 17-5 Message 表结构

字段	中文描述	数据类型	是否为空	备注
id	博客评论编号	int	否	主键
UserName	评论人名称	varchar(20)	是	
date	评论时间	datetime	是	外键
icon	评论人头像	varchar(50)	是	
body	评论内容	varchar(100)	是	
face	表情	varchar(50)	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Message](
 [id] [int] IDENTITY(1,1) NOT NULL, //设置主键
 [UserName] [varchar](20) COLLATE Chinese_PRC_CI_AS NULL,
 [date] [datetime] NULL,
 [icon] [varchar](50) COLLATE Chinese_PRC_CI_AS NULL,
 [body] [varchar](100) COLLATE Chinese_PRC_CI_AS NULL,
 [face] [varchar](50) COLLATE Chinese_PRC_CI_AS NULL
)
```

(5) 图片文件表（Photo）：用来保存网站中的图片文件信息，该表的字段结构如表 17-6 所示。

表 17-6 Photo 表结构

字段	中文描述	数据类型	是否为空	备注
photo_id	图片编号	int	否	主键
Title	图片标题	varchar(50)	是	
Info	图片描述	varchar(50)	是	
AddDate	创建时间	datetime	是	
Url	图片保存路径	varchar(50)	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Photo](
 [photo_id] [int] IDENTITY(1,1) NOT NULL, //设置主键
 [Title] [varchar](50) COLLATE Chinese_PRC_CI_AS NULL,
 [Info] [varchar](50) COLLATE Chinese_PRC_CI_AS NULL,
 [AddDate] [datetime] NULL,
 [Url] [varchar](50) COLLATE Chinese_PRC_CI_AS NULL,
)
```

17.2.2 设计系统存储过程

构建了数据库表结构后，接下来创建数据表中信息选择、添加、更新和删除操作的相关存储过程。由于使用存储过程会给系统带来优化和提高效率，系统使用了 23 个存储过程，这些存储过程说明如下。

#### (1) GetNumber 存储过程

该存储过程实现从博客文章表 (News) 查询当前存在博客文章的数量并返回该数量。

#### (2) pro\_AddClass 存储过程

该存储过程实现添加一个新的博客文章类别到博客类别表 (Class) 中, 需要使用两个参数, 一个是类别的名称, 一个是创建的时间。

#### (3) pro\_AddMessage 存储过程

该存储过程实现添加一条新的评论到博客评论表 (Message) 中, 需要使用评论人姓名、创建时间、评论人头像保存路径、评论内容和评论人表情保存路径共 4 个参数。

#### (4) AddNew 存储过程

该存储过程实现添加一篇新的博客文章到博客文章表 (News) 中, 需要使用文章标题、文章内容、添加时间、浏览次数、评论次数、文章类别、是否保存到草稿箱和是否保存到回收站共 8 个参数。

#### (5) pro\_AddPhoto 存储过程

该存储过程实现将一张新的图像添加到图片文件表 (Photo) 中, 需要使用图像标题、图像描述、图像创建日期和图像保存路径共 4 个参数。

#### (6) pro\_AddUser 存储过程

该存储过程实现向用户注册表 (Register) 表中添加一个用户, 需要使用用户名称、用户密码、确认密码、安全问题、安全回答、电子邮件、真实姓名、地址、年龄、性别和电话共 11 个参数。

#### (7) pro\_DDNews 存储过程

该存储过程实现从博客文章表 (News) 中删除指定编号的博客文章, 需要的参数是博客文章的编号。

#### (8) pro\_DelClass 存储过程

该存储过程实现从博客类别表 (Class) 中删除指定编号的博客类别, 需要的参数是博客类别的编号。

#### (9) pro\_DelPhoto 存储过程

该存储过程实现从图片文件表 (Photo) 中删除指定编号的图片, 需要的参数是图片的编号。

#### (10) pro\_GePhotoByID 存储过程

该存储过程实现根据指定的图片编号查询该图片的信息, 需要的参数是图片的编号。

#### (11) pro\_GetAllNews 存储过程

该存储过程实现查询在博客文章表 (News) 中前 50 篇已经发布的博客文章信息 (即在回收站和保存箱中不存在的博客文章)。

(12) pro\_GetAllPhoto 存储过程

该存储过程实现查询图片文件表 (Photo) 中所有的图片。

(13) pro\_GetCallBack 存储过程

该存储过程实现查询回收站中所用博客文章信息, 即在博客文章表 (News) 中 IsDel 字段值为 1 的所有博客文章。

(14) pro\_GetClassByID 存储过程

该存储过程实现根据指定的博客类别编号查询该博客类别的信息, 需要的参数是博客类别的编号。

(15) pro\_GetNewByID 存储过程

该存储过程实现根据指定的博客文章编号查询该博客文章的信息, 需要的参数是博客文章的编号。

(16) pro\_GetNewsSave 存储过程

该存储过程实现查询出保存箱中所有的博客文章信息即在博客文章表 (News) 中 IsDel 字段值为 0 而 IsSav 字段值为 1 的所有博客文章。需要两个参数代表 IsDel 和 IsSav 的值。

(17) pro\_Revert 存储过程

该存储过程实现从回收站中将指定的博客文章还原即将博客文章表 (News) 中 IsDel 字段值从 1 更新为 0, 需要博客文章编号作为参数。

(18) pro\_Select 存储过程

该存储过程实现根据用户名和密码对登录用户进行身份验证, 需要用户名和密码两个参数。

(19) pro\_UpdateClass 存储过程

该存储过程实现对指定博客类型进行更新, 需要类型编号、类型名称和修改日期共三个参数。

(20) pro\_UpdateNew 存储过程

该存储过程实现对指定博客文章进行更新, 需要博客文章编号、文章标题、文章类别、文章内容和是否放入保存箱共五个参数。

(21) pro\_UpdateNewClass 存储过程

该存储过程实现对指定博客文章进行类别更新, 需要博客文章编号作为参数。

(22) pro\_UpdatePhoto 存储过程

该存储过程实现对指定图片进行更新。需要图片编号、图片描述、图片标题共三个参数。

(23) pro\_UpdateSX 存储过程

该存储过程实现将指定博客文章放入回收站即将博客文章表 (News) 中 IsDel 字段值设置为 1, 需要博客文章编号作为参数。

### 17.3 实体类模块

本系统的实体类模块由 LINQ to SQL 类自动生成的数据库实体映射文件 BlogDataClasses.dbml，创建数据库实体映射文件的具体步骤如下。

**01** 选择菜单栏上的“视图” | “服务器资源管理器”命令，弹出如图 17-12 所示的“服务器资源管理器窗口”，右键单击“数据连接”，在菜单中选择“添加连接”的命令。

**02** 在弹出的“添加连接”对话框中的数据源文本框中输入“Microsoft SQL Server (SqlClient)”，服务器名文本框输入本地计算机的名称。在选择数据库下拉列表中选择“Blog”数据库。单击“测试连接”按钮，如果连接成功，会弹出“连接成功的对话框”，然后单击该对话框中的“确定”按钮，最后回到“添加连接”对话框中单击“确定”按钮，如图 17-13 所示。



图 17-12 服务器资源管理器



图 17-13 “添加连接”对话框

**03** 这时，在“服务器资源管理器窗口”中的“数据连接”节点下，会出现刚才添加好的如图 17-14 所示的“Blog.dbo”数据库。

**04** 右键单击网站名称，选择“添加新项”菜单选项，弹出如图 17-15 所示的“添加新项对话框”，选择“已安装模板”下的“Visual C#”模板，并在模板文件列表中选中“LINQ to SQL”，然后在“名称”文本框输入该文件的名称“BlogDataClasses.dbml”，最后单击“添加”按钮。

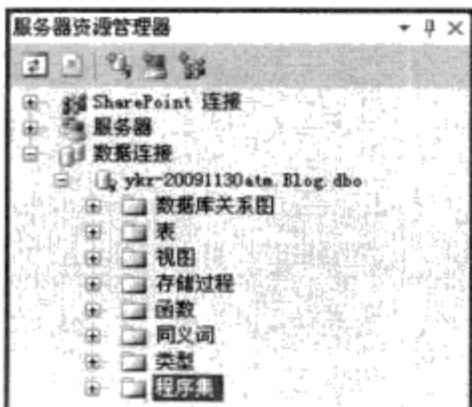


图 17-14 生成连接



图 17-15 添加新项对话框

**05** 此时在网站根目录下会自动生成如图 17-16 所示的“BlogDataClasses.dbml”的文件，该文件又包含了一个“BlogDataClasses.dbml.layout”文件和一个“BlogDataClasses.designer.cs”文件。

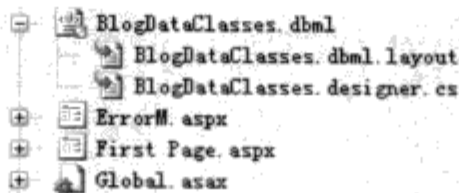


图 17-16 生成 BlogDataClasses.dbml 文件

**06** 双击“BlogDataClasses.dbml”文件，出现 LINQ to SQL 类的“对象关系设计器”界面，在此界面中，可以通过拖拽方式来定义与数据库相对应的实体和关系。将“服务器资源管理器窗口”中“Blog.dbo”下“表”文件夹中所有的数据表和“存储过程”文件中所有的存储过程依次拖曳到“对象关系设计器”界面中，这时就会生成一个如图 17-17 所示的数据库实体映射实类。

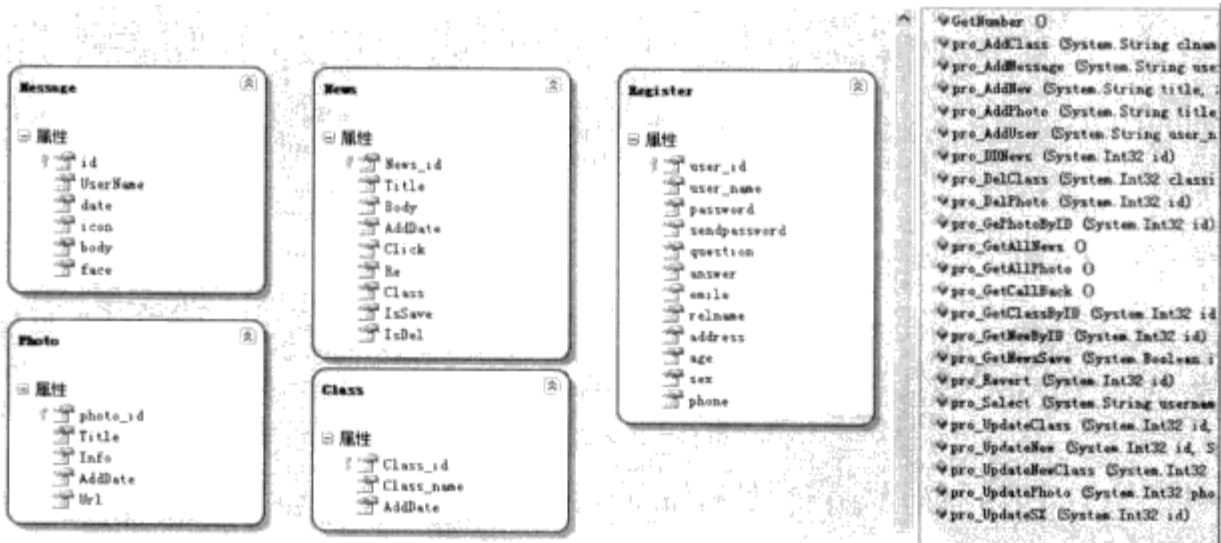


图 17-17 对象关系设计器

**07** 打开文件 BlogDataClasses1.designer.cs，可以看到该文件自动生成了包含 LINQ 到 SQL 实体类以及强类型 BlogDataClasses1DataContext 的定义。至此，数据库实体映射文件类创建完毕，在页面代码中就可以像使用其他类型的类一样使用它。

## 17.4 首页显示模块

在本系统中的首页显示模块有前台首页和后台首页构成。在前台所有的页面中使用了母版页机制，方便页面的统一管理。

### 17.4.1 设计母版页

Site2.Master 母版页设计了整个网站的结构布局，它由上中下三个部分组成。上部是页面的头部，包含了一个网站 LOGO 的 Banner 条和菜单栏。下部是页脚，包含网站的版权信息。中部是分为左右两个部分，左边显示的具体的博客内容，占了大部分的位置，右边是侧边栏，显示一些导航链接。设计母版页的代码位于 Site2.Master 文件中，关键的 HTML 代码如下：

```

1. <div id="outer">
2. <div id="inner">
3. <div id="header"><h1>我的博客</h1></div>
4. <div id="splash"></div>
5. <div id="menu">
6.
7. <li class="first"><ucl:User ID="User1" runat="server" />
8. 相册
9. 日志
10. 留言
11. 管理登录 注册
12.
13. <div id="date">访问量: <%= Application["num"] %></div>
14. </div>
15. <div id="primarycontent">
16. <asp:ContentPlaceHolder ID="Right" runat="server">
17. </asp:ContentPlaceHolder>
18. </div>
19. <div id="secondarycontent">
20. <asp:ContentPlaceHolder ID="Left" runat="server">
21. </asp:ContentPlaceHolder>
22. </div>
23. <div id="footer">© My Blog .
24. </div>
25. </div>
26. </div>

```

代码说明：该母版页使用了 div 进行布局。第 1 行定义 div 的容器。第 3~14 行定义页面的头部，其中，第 3 行定义 LOGO 文字，第 4 行定义 LOGO 图片。第 5~13 行定义了菜单栏。第 15~22 行定义页面的中部，其中第 15~18 行显示中部左侧栏定义占位符控件 Right。第 19~22 行是页面中部的右边部分，定义占位符控件 Left。第 23 行定义页面的页脚部分。

用户控件 User 在 Controls 文件夹下的 User.ascx 文件中定义，关键代码如下：

```

1. <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="User.ascx.cs" Inherits="Web.Controls.User" %>
2. 首页

```

代码说明：第 1 行使用 @Control 指令设置用户控件的属性。第 2 行定义一个首页的超链接。母版页 Site2.Master 最后设计完毕的效果如图 17-18 所示。

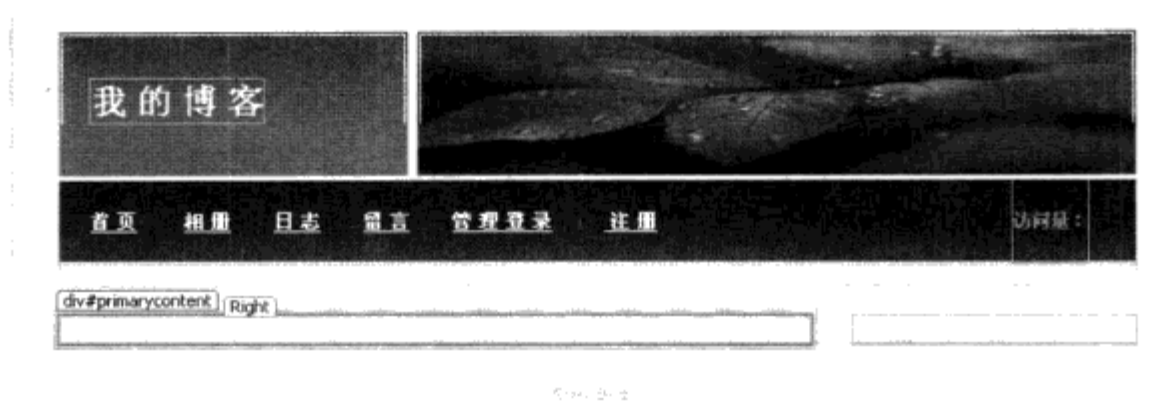


图 17-18 设计后的母版页

## 17.4.2 首页

首页作为内容页面被包含在母版页 Site2.master 的占位符控件显示最新文章、友情链接和博客文章。

(1) 设计首页的代码位于 First Page.aspx 文件中，关键的 HTML 代码如下。

```

1. <asp:Content ID="Content2" ContentPlaceHolderID="Left" runat="server">
2. <uc1:LeftList ID="LeftList1" runat="server" />
3. </asp:Content>
4. <asp:Content ID="Content3" ContentPlaceHolderID="Right" runat="server">
5. <asp:Repeater ID="Repeater2" runat="server">
6. <ItemTemplate>
7. <div id="c-right">
8. <div class="Page">
9. <h3>
10. <a href='<%=# Eval("news_id","Next.aspx?id={0}") %>' target="_blank"><%=# Eval("Title") %>
11. </h3>
12. <div class="content">
13. <p class="botime"><%=# Eval("AddDate") %></p>
14. <p class="bod"><%=# BindBody(Eval("Body").ToString()) %></p>
15. </div>
16. <div class="footer">
17. <li class="readmore"> 浏览[<%=# Eval("Click") %>] | 评论[<%=# Eval("Re") %>]
18. </div>
19. </div>
20. </div>
21. </ItemTemplate>
22. </asp:Repeater>
23. </asp:Content>

```

代码说明：第 1~3 行在母版页的占位符控件 Content2 中添加一个用户控件 LeftList。第 4~23 行在母版页的占位符控件 Content3 定义一个服务器 Repeater 控件 Repeater2，其中，第 6~21 行在 ItemTemplate 模板中定义博客显示的内容：博客的标题、博客的创建时间、博客的内容和博客的浏览和评论的数量。

(2) 首页后台代码位于 First Page.aspx.cs 文件中，关键代码如下。

```

1. protected void Page_Load(object sender, EventArgs e) {
2. if (!IsPostBack){
3. Bind();
4. }

```

```

5. }
6. protected void Bind(){
7. var result = (from n in bdc.News
8. select n).Take(3);
9. Repeater2.DataSource = result;
10. Repeater2.DataBind();
11. }

```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断如果不是回传页面第 4 行调用 Bind 方法绑定数据。第 6 行定义 Bind 方法，第 7、8 行通过 LINQ 语句查询获得数据库博客文章表 News 中前 3 篇博客文章的对象列表 result。第 12 行将该列表作为 Repeater2 控件的数据源。第 13 行调用 Repeater2 控件的 DataBind 方法将数据绑定显示。

(3) 首页中的用户控件 LeftList 在 Controls 文件夹下的 LeftList.ascx 文件中定义，关键代码如下。

```

1. <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="LeftList.ascx.cs" Inherits="Web.LeftList" %>
2. <h3>最新文章</h3>
3. <div class="content">
4. <asp:Repeater ID="Repeater1" runat="server">
5. <HeaderTemplate><ul class="linklist"></HeaderTemplate>
6. <ItemTemplate>
7. <a href='<%# Eval("news_id", "Next.aspx?id={0}") %>'><%# Eval("Title") %>
8. </ItemTemplate>
9. <FooterTemplate></FooterTemplate>
10. </asp:Repeater>
11. </div>
12. <h3>友情链接</h3>
13. <div class="content">
14. <ul class="linklist">
15. 雅虎
16. 腾讯
17. 百度
18. 新浪
19.
20. </div>

```

代码说明：第 1 行使用 @Control 指令设置用户控件的属性。第 3~11 行在 div 中定义一个服务器 Repeater 控件 Repeater1。第 5 行通过 HeaderTemplate 模板定义控件头部的显示。第 6~8 行通过 ItemTemplate 模板定义显示博客文章的标题。第 9 行使用 FooterTemplate 模板定义控件脚部的显示。第 14~19 行在 div 中显示 4 个链接的无序列表。

(4) 用户控件 LeftList 的后台代码位于 LeftList.ascx.cs 文件中，关键代码如下。

```

1. BlogDataClassesDataContext bdc = new BlogDataClassesDataContext();
2. protected void Page_Load(object sender, EventArgs e) {
3. if (!IsPostBack){
4. Bind();
5. }
6. }
7. protected void Bind() {
8. var result = (from n in bdc.News
9. where n.IsDel == false && n.IsSave == false
10. orderby n.News_id descending
11. select n).Take(10);
12. Repeater1.DataSource = result;

```

```

13. Repeater1.DataBind();
14. }

```

代码说明：第 1 行实例化数据实体上下文对象 bdc。第 2 行定义处理页面 Page 加载事件的方法 Load。第 3 行判断如果不是回传页面第 4 行调用 Bind 方法绑定数据。第 7 行定义 Bind 方法，第 8~11 行通过 LINQ 语句查询获得最新发布的前十位博客文章的对象列表 result。第 12 行将该列表作为 Repeater1 控件的数据源。第 13 行调用 Repeater1 控件的 DataBind 方法将数据绑定显示。设计后首页的效果如图 17-19 所示。

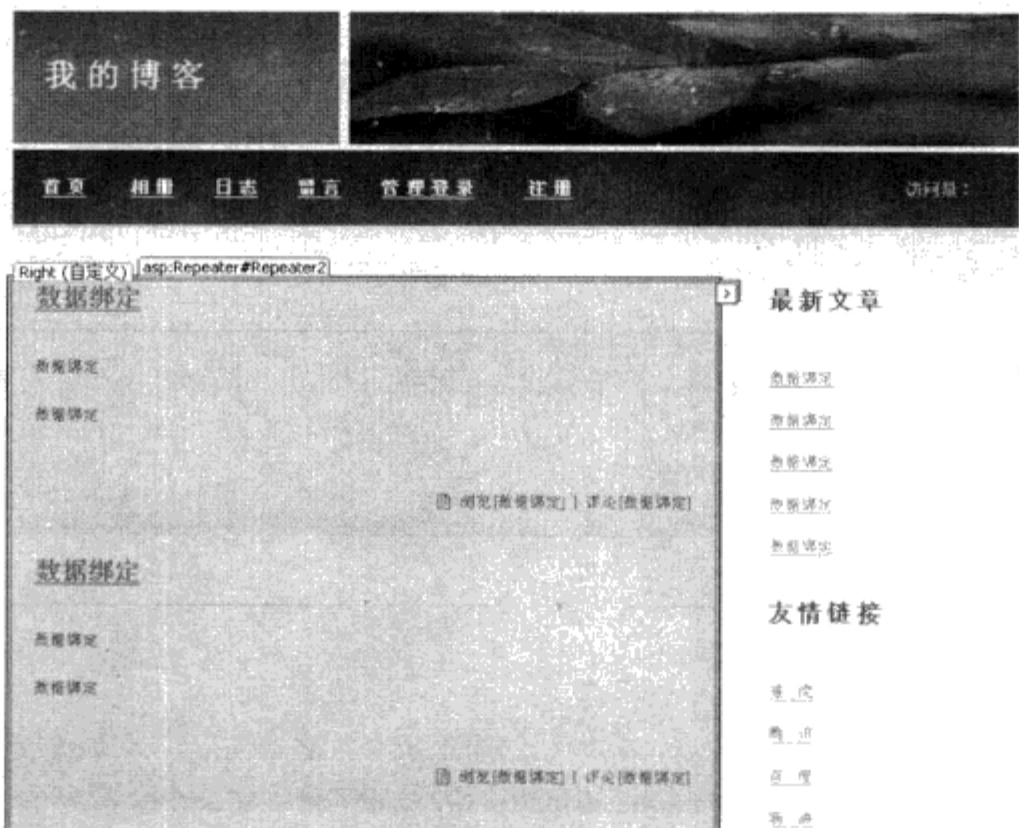


图 17-19 设计后的首页界面

## 17.5 用户登录模块

本节实现用户注册后经过登录验证进入后台管理的功能。登录页面设计代码位于 Manage 文件夹下的 Login.aspx 文件中，关键 HTML 代码如下。

```

1. <form id="form1" runat="server">
2. <div id="login">
3. <table width="100%" style="height: 222px">
4. <tr><td height="80px" class="style2">后台管理 登录</td></tr>
5. <tr><td>用户名</td></tr>
6. <tr><td class="style1">
7. <asp:TextBox ID="txtuid" runat="server" CssClass="text" ></asp:TextBox>
8. <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" Font-Size="Small"
9. ControlToValidate="txtuid" Display="Dynamic" ErrorMessage="**必填"></asp:RequiredFieldValidator>
10. </td></tr>
11. <tr><td height="13px"></td></tr>
12. <tr><td>密 码</td></tr>
13. <tr><td><asp:TextBox ID="txtpwd" runat="server" TextMode="Password" CssClass="text" style="vertical-align: top"
14. ></asp:TextBox>
15. <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ControlToValidate="txtpwd"
16. Display="Dynamic" ErrorMessage="**必填"></asp:RequiredFieldValidator>

```

```

14. </td></tr>
15. </tr></td>
16. <p style="text-align:right; margin-top:30px; height: 40px;">
17. <asp:Button ID="btnlogin" runat="server" Text="登 录"
18. onclick="btnlogin_Click" /></p>
19. </td></tr>
20. </table>
21. </div>
22. </form>

```

代码说明：第 3~20 行设计了一个 7 行 1 列表格来显示登录界面。其中，第 4 行是表格的第一行显示登录的标题。第 5 行是表格的第二行显示用户名。第 6~9 行是表格的第三行，添加了文本框控件 txtuid 和验证控件 RequiredFieldValidator1。第 11 行是表格的第五行显示密码。第 12~14 行是表格的第六行添加了文本框控件 txtpwd 和验证控件 RequiredFieldValidator2。第 15~19 行是表格的第七行添加一个段落并设置登录的按钮控件 btnlogin。

登录页面的后台代码位于 Manage 文件夹下的 Login.aspx.cs 文件中，关键代码如下。

```

1. protected void btnlogin_Click(object sender, EventArgs e){
2. BlogDataClassesDataContext bdc = new BlogDataClassesDataContext();
3. string uid = txtuid.Text.Trim().ToString();
4. string pwd = txtpwd.Text.Trim().ToString();
5. var result = from n in bdc.Register
6. where n.user_name == uid && n.password == pwd
7. select n;
8. foreach(var n in result){
9. if(n.user_name!=null){
10. Session["username"] = uid;
11. Response.Redirect("../Manage/Hou.aspx",true);
12. }
13. }
14. Response.Write("<script>alert('用户名或密码错误，请重新输入！');history.back()</script>");
15. }

```

代码说明：第 1 行定义处理登录按钮 btnlogin 单击事件 Click 的方法。第 2 行实例化实体数据上下文对象 bdc。第 3、4 行获取用户输入的用户名和密码。第 5 行使用 LINQ 查询获得用户注册表中该用户的用户名。第 8~12 行使用循环变量查询结果 result 中如果用户名不为空，将该用户名保存到 Session 对象中并重新定向页面到后台管理界面。第 14 行如果查询结果中没有数据，给出错误提示。

设计后登录页面如图 17-20 所示。

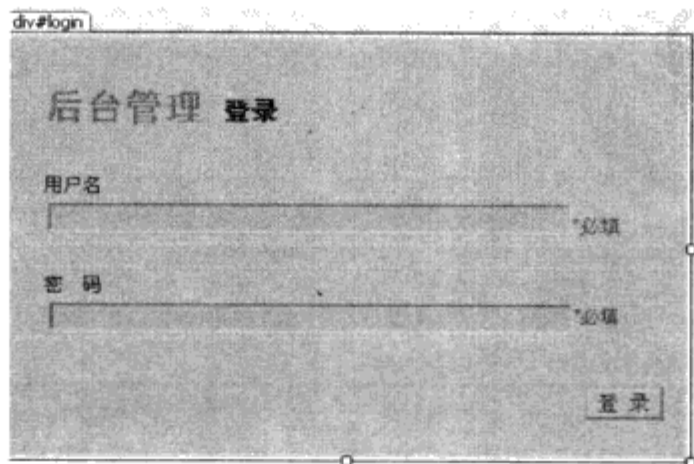


图 17-20 设计后的登录页面

## 17.6 浏览博客模块

该模块实现对网站前台的博客内容进行浏览的功能，包括浏览博客、浏览相册和访客评论。

### 17.6.1 浏览博客

在该功能中创建 Web Service 定义获得博客文字的方法，然后在页面中调用该方法在页面进行和数据控件进行绑定显示。

(1) 创建 Web Service 的代码定义在 WebService.asmx 下的 WebService.asmx.cs 文件中，关键代码如下：

```

1. public class WebService : System.Web.Services.WebService{
2. [WebMethod]
3. public DataSet GetNews(){
4. string conn = ConfigurationManager.ConnectionStrings["Conneciton"].ConnectionString;
5. SqlDataAdapter da = new SqlDataAdapter("select top 50 * from News where issave=0 and isdel=0", conn);
6. DataSet ds = new DataSet();
7. da.Fill(ds, "news");
8. return ds;
9. }
10. }

```

代码说明：第 1 行定义 WebService 类继承与 System.Web.Services.WebService。第 2 行使用 [WebMethod] 属性指定下面的方法是 WebService 方法。第 3 行定义返回 DataSet 对象的方法 GetNews。第 4 行定义数据库连接字符串。第 5 行定义 SqlDataAdapter 对象 da 获得已发表的前 50 位的博客文章信息。第 6 行定义 DataSet 对象 ds。第 7 行调用 Fill 方法填充数据集。第 8 行方法数据集对象 ds。

(2) 设计浏览博客页面的代码位于 Blog.aspx 文件中，关键的 HTML 代码如下。

```

1. <asp:Content ID="Content3" ContentPlaceHolderID="Right" runat="server">
2. <asp:Repeater ID="Repeater1" runat="server">
3. <ItemTemplate>
4. <div id="c-right">
5. <div class="Page">
6. <h3><a href='<# Eval("news_id","Next.aspx?id={0}") %>' target="_blank"><# Eval("Title") %></h3>
7. <p class="botime"><# Eval("AddDate") %></p>
8. <p class="bod"><# BindBody(Eval("Body").ToString()) %></p>
9. <p class="show"> 浏览[<# Eval("Click") %>] | 评论[<# Eval("Re") %>]</p>

10. </div>
11. </div>
12. </ItemTemplate>
13. </asp:Repeater>

14. 共<asp:Label ID="lb_page" runat="server" Text="Label"></asp:Label>页
15. 当前第 <asp:Label ID="lb_currentpage" runat="server" Text="1"></asp:Label>页
16. <asp:LinkButton ID="lbtn_frist" runat="server" OnClick="lbtn_frist_Click">首页</asp:LinkButton>
17. <asp:LinkButton ID="lbtn_up" runat="server" OnClick="lbtn_up_Click">上一页</asp:LinkButton>
18. <asp:LinkButton ID="lbtn_down" runat="server" OnClick="lbtn_down_Click">下一页</asp:LinkButton>
19. <asp:LinkButton ID="lbtn_last" runat="server" OnClick="lbtn_last_Click">尾页</asp:LinkButton>
20. <asp:DropDownList
21. ID="DropDownList1"
22. runat="server"
23. AutoPostBack="True"
24. OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged"></asp:DropDownList>
25. </asp:Content>

```

代码说明：第 2~13 行定义一个服务器 Repeater 控件 Repeater1 显示博客文章的数据，其中，第 6~9 行在 ItemTemplate 模板中定义博客显示的内容：博客的标题、博客的创建时间、博客的内容和博客的浏览和评论的数量。第 14~19 行定义分页导航，由 2 个 Label 控件和 4 个链接按钮控件组成分别表示总页数、当前页、首页、上一页、下一页和尾页。第 20 行添加一个下拉列表控件显示页数列表，通过选择具体页数可以显示相应页的信息。

(3) 浏览博客页面的后台代码位于 Blog.aspx.cs 文件中，关键的代码如下。

```

1. protected void Page_Load(object sender, EventArgs e) {
2. if (!IsPostBack){
3. dlbind();
4. }
5. }
6. public void dlbind(){
7. WebService ws = new WebService();
8. int curpage = Convert.ToInt32(lb_currentpage.Text);
9. PagedDataSource ps = new PagedDataSource();
10. ps.DataSource = ws.GetNews().Tables["news"].DefaultView;
11. ps.AllowPaging = true;
12. ps.PageSize = 3;
13. ps.CurrentPageIndex = curpage - 1;
14. lb_page.Text = Convert.ToString(ps.PageCount);
15. if (!IsPostBack){
16. for (int i = 1; i <= ps.PageCount; i++){
17. DropDownList1.Items.Add(i.ToString());
18. }
19. DropDownList1.SelectedItem.Text = curpage.ToString();
20. }
21. lbtn_frist.Enabled = true;
22. lbtn_up.Enabled = true;
23. lbtn_down.Enabled = true;
24. lbtn_last.Enabled = true;
25. if (curpage == 1){
26. lbtn_frist.Enabled = false;
27. lbtn_up.Enabled = false;
28. }
29. if (curpage == ps.PageCount){
30. lbtn_down.Enabled = false;
31. lbtn_last.Enabled = false;
32. }
33. Repeater1.DataSource = ps;
34. Repeater1.DataBind();
35. }
36. protected void lbtn_frist_Click(object sender, EventArgs e){
37. lb_currentpage.Text = "1";
38. dlbind();
39. }
40. protected void lbtn_up_Click(object sender, EventArgs e){
41. lb_currentpage.Text = Convert.ToString(Convert.ToInt32(lb_currentpage.Text) - 1);
42. dlbind();
43. }
44. protected void lbtn_down_Click(object sender, EventArgs e) {
45. lb_currentpage.Text = Convert.ToString(Convert.ToInt32(lb_currentpage.Text) + 1);
46. dlbind();
47. }
48. protected void lbtn_last_Click(object sender, EventArgs e){

```

```

49. lb_currentpage.Text = lb_page.Text;
50. dlbind();
51. }
52. protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e){
53. int page = Convert.ToInt32(DropDownList1.SelectedItem.Value);
54. lb_currentpage.Text = page.ToString();
55. dlbind();
56. }

```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断如果不是回传页面第 3 行调用 dlbind 方法绑定数据。第 6 行定义 dlbind 方法实现绑定数据和分页。第 7 行实例化 WebService 对象 ws。第 8 行定义并获取当前页数。第 9 行实例化分页数据源对象 ps。第 10 行调用 WebService 中的 GetNews 方法，获取数据集中数据表 news 作为 ps 对象的数据源。第 11 行启动分页功能。第 12 行指每页显示的数据数量。第 13 行取得当前页的页码。第 14 行获得并显示总页数。第 15~20 行生成下拉列表内的列表项内容并显示当前页数。第 21~24 行设置 4 个链接按钮可用状态。第 25~32 行当前页选择开启或禁用 4 个链接按钮。第 33 行将分页数据源对象 ps 作为 Repeater1 的数据源。第 34 行调用 Bind 方法绑定数据到 Repeater1。

第 36~51 行定义处理四个链接按钮单击事件的方法，根据当前页执行上一页、下一页、首页和尾页的操作并再次调用 dlbind 方法绑定数据。

第 52 行定义处理下拉列表控件 DropDownList1 选择项被更新事件的方法 SelectedIndexChanged。第 53 行获取用户选中的页数。第 54 行将该页数作为当前页。第 55 并再次调用 dabind 方法绑定数据。

设计后的浏览博客页面如图 17-21 所示。

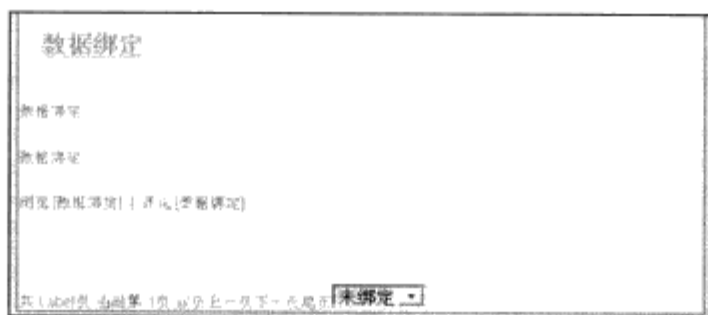


图 17-21 设计后的浏览博客界面

## 17.6.2 访客评论

在访客评论的功能中设计了评论和回复两个用户控件，然后在页面中注册这两个控件来实现页面的显示。

(1) 设计评论用户控件的代码定义在 Controls 文件夹下的 liuyan.ascx 文件中，关键的 HTML 代码如下。

```

1. <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="liuyan.ascx.cs" Inherits="Web.Controls.liuyan" %>
2. <asp:Repeater ID="Repeater1" runat="server">
3. <HeaderTemplate>
4. <table style="width:640px; height:100%; margin:10px;" cellpadding="0" cellspacing="0">
5. </HeaderTemplate>
6. <ItemTemplate>
7. <tr style="background-color:#575733">
8. <td style="width:150px; height:31px">
9. </td>

```

```

10. <td style="text-align:left; color:White"><img src='<%# Eval("face") %>' width="24px" height="24px"
style="padding:2px"/> <%# Eval("UserName")%> 于 <%# Eval("date") %></td>
11. </tr>
12. <tr style="height:90px">
13. <td style="text-align:left; vertical-align:top;">
14. <table>
15. <tr>
16. <td style="padding:10px"><div style="padding:1px; border:1px solid #000000"><img
style="width:90px;height:90px" src='<%# Eval("icon") %>' /></div></td>
17. </tr>
18. </table>
19. </td>
20. <td style="text-align:left; vertical-align:top">
21. <table style="width:100%">
22. <tr>
23. <td><%# Eval("body") %></td>
24. </tr>
25. </table>
26. </td>
27. </tr>
28. </ItemTemplate>
29. <FooterTemplate>
30. <table>
31. <FooterTemplate>
32. </asp:Repeater>

```

代码说明：第 1 行使用 @Control 指令定义该用户控件的属性。第 2~23 行定义一个 Repeater 服务器控件 Repeater1，其中，第 3~5 行在控件中定义另一个表格。第 6~28 行在 ItemTemplate 模板中显示评论图标、评论人头像、评论人姓名、评论时间、评论内容。

评论用户控件的后台代码定义在 Controls 文件夹下的 liuyan.ascx.cs 文件中，关键的代码如下。

```

1. protected void Page_Load(object sender, EventArgs e){
2. if (!IsPostBack){
3. dlbind();
4. }
5. }

```

代码说明：第 1 行定义处理页面 Page 加载事件的方法 Load。第 2 行判断如果不是回传页面第 3 行调用 dlbind 方法绑定数据。dlbind 方法是实现页面绑定和分页功能的方法具体代码和浏览博客页面的后台代码 Blog.aspx.cs 文件中的 dlbind 方法类似，这里不再重复。

设计后的评论用户控件如图 17-22 所示。

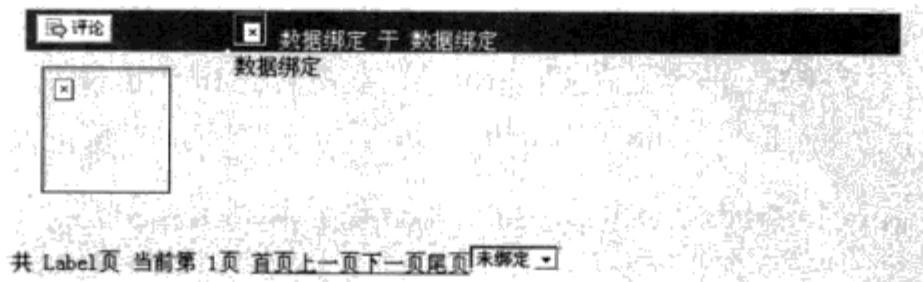


图 17-22 设计后评论用户控件

(2) 设计回复用户控件的代码定义在 Controls 文件夹下的 huifu.ascx 文件中，关键的 HTML 代码如下。

```

1. <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="huifu.ascx.cs" Inherits="Web.Controls.huifu" %>
2. <table style="width:650px; height:300px">
3. <tr>
4. <td style="width:100px">用户头像</td>
5. <td class="style1"></td>
6. <td class="style2">您的姓名: </td>
7. <td style="text-align:left"><input id="txtname" type="text" runat="server" /></td>
8. <td><asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
9. ControlToValidate="txtname" Display="Dynamic" ErrorMessage="* 不 能 为 空 "
Font-Size=Small></asp:RequiredFieldValidator></td>
10. </tr>
11. <tr>
12. <td style="vertical-align:top" rowspan="2"><asp:Image style="padding:0 0 0 15px;" ID="imgicon"
ImageUrl="~/icon/4.gif" runat="server"></asp:Image></td>
13. <td style="vertical-align:top" class="style1"><asp:LinkButton ID="btnup" ToolTip="上一个头像" runat="server"
CssClass="btn1" CausesValidation="False" onclick="btnup_Click"></asp:LinkButton></td>
14. <td style="vertical-align:top" rowspan="2">回复内容: </td>
15. <td style="text-align:left; vertical-align:top" rowspan="2"><textarea id="TextArea1" style="width: 255px; height: 80px"
runat="server"></textarea></td>
16. <td rowspan="2">
17. <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ControlToValidate="TextArea1"
Display="Dynamic" ErrorMessage="* 不能为空" Font-Size=Small></asp:RequiredFieldValidator>
18. </td>
19. </tr>
20. <tr><td style="vertical-align:bottom" class="style3"><asp:LinkButton ID="btndown" ToolTip="下一个头像"
runat="server" OnClick="btndown_Click" CausesValidation="False" ></asp:LinkButton></td>
21. </tr>
22. <tr>
23. <td style="width:100px"> </td>
24. <td class="style1"> </td>
25. <td class="style2"> </td>
26. <td style="text-align:left">
27.
28. <asp:RadioButton ID="rdo1" runat="server" Checked="True" GroupName="aaa" />
29.
30. <asp:RadioButton ID="rdo2" runat="server" GroupName="aaa" />
31.
32. <asp:RadioButton ID="rdo3" runat="server" GroupName="aaa" />

33.
34. <asp:RadioButton ID="rdo4" runat="server" GroupName="aaa" />
35. <asp:RadioButton
ID="rdo5" runat="server" GroupName="aaa" />
36. <asp:RadioButton ID="rdo6" runat="server"
GroupName="aaa" />
37. </td>
38. <td></td>
39. </tr>
40. <tr>
41. <td> </td>
42. <td class="style1"> </td>
43. </td> <td class="style2"></td>
44. <td style="text-align:left"><asp:ImageButton ID="btnSubmit" runat="server" ImageUrl="~/face/pinglun.gif"
onclick="btnSubmit_Click" /></td>
45. <td><asp:Label ID="txtinfo" runat="server" Text="Label" Visible="False"></asp:Label></td>
46. </tr>
47. </table>

```

代码说明: 第 1 行使用@Control 指令定义该用户控件的属性。第 2~47 行定义一个 5 行 5 列表

的表格。第 3~10 行定义了表格的第 1 行 5 个列分别显示文本、文本框控件 txtname 和验证服务器控件 RequiredFieldValidator1。第 11~19 行定义表格的第 3、4 行的 5 个列，分别显示 1 个图像控件 imgicon、链接按钮控件 btnup、显示的文本、多行文本框 TextArea1 和验证控件 RequiredFieldValidator2。第 20~21 行定义第 3 行第 2 列显示显示 1 个图像标记和链接按钮控件 btndown。

第 22~39 行定义表格第 4 行的 5 个列的内容，在第 4 列中显示 6 个图标图片和 6 个服务器单选按钮 RadioButton。第 40~46 行定义表格最后一行的 5 个列的内容，在第 4 列中显示一个服务器图像按钮控件 btnSubmit。在第 5 列中显示一个标签控件 txtinfo。

评论用户控件的后台代码定义在 Controls 文件夹下的 huifu.ascx.ascx.cs 文件中，关键的代码如下。

```

1. protected void btnSubmit_Click(object sender, ImageClickEventArgs e){
2. BlogDataClassesDataContext bdc = new BlogDataClassesDataContext();
3. string name = txtname.Value.Trim();
4. string body = TextArea1.Value.Trim();
5. string face = "";
6. if (rdo1.Checked) {
7. face = "../face/hi.gif";
8. }
9. else if (rdo2.Checked){
10. face = "../face/bang.gif";
11. }
12. else if (rdo3.Checked) {
13. face = "../face/ku.gif";
14. }
15. else if (rdo4.Checked){
16. face = "../face/nu.gif";
17. }
18. else if (rdo5.Checked){
19. face = "../face/why.gif";
20. }
21. else if (rdo6.Checked){
22. face = "../face/yan.gif";
23. }
24. if (bdc.pro_AddMessage (name,DateTime.Now,imgicon.ImageUrl,body,face)<=0){
25. Response.Redirect("Message.aspx");
26. }
27. else{
28. Response.Write("<script>alert('评论失败!')</script>");
29. }
30. txtname.Value = "";
31. TextArea1.Value = "";
32. }

```

代码说明：第 1 行定义处理按钮 btnSubmit 单击事件 Click 的方法，第 2 行实例化实体数据上下文对象 bdc。第 3、4 行获取访客的名称和评论内容。第 6~23 行判断访客所选择的图标。第 24 行调用 bdc 对象的 AddMessage 方法判断如果添加评论成功，第 25 行重新定向到访客评论页面。否则第 28 行给出评论失败的提示框。第 30、31 行将两个文本框清空。

设计后的回复用户控件如图 17-23 所示。

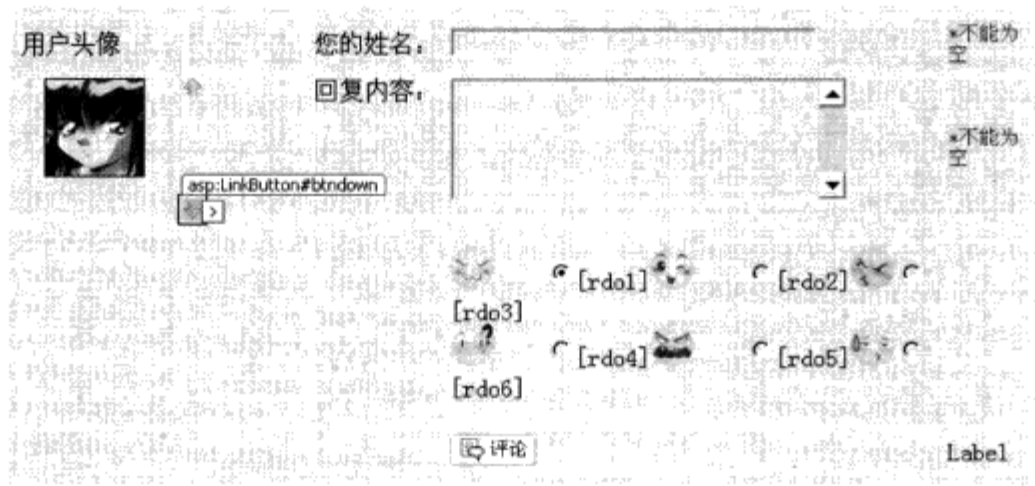


图 17-23 设计后的回复用户控件

(3) 设计访客评论页面的代码定义在 Manage 文件夹下的 Message.aspx 文件中，其关键的 HTML 代码如下。

```
1. <div id="inner">
2. <div>
3. <uc1:liuyan ID="liuyan1" runat="server" />
4. </div>
5.

6. <uc2:huifu ID="huifu1" runat="server" />
7. </div>
```

代码说明：第 1~7 行使用 div 进行布局。其中，第 3 行和第 6 行分别使用评论用户控件和回复用户控件。

本模块中的浏览相册与浏览博客的实现基本相同，仅查询的数据库表不同，所以不再重复介绍。

## 17.7 管理博客模块

该模块也是本系统的后台模块，博客用户可以在后台进行对博客的管理。包括：包括文章的管理、博客中图片的管理、对访客的评论进行管理等。由于这些管理中的许多功能都有类似的情况，所以只对比较有代表性的内容进行介绍。

### 17.7.1 添加博客

设计添加博客页面的代码定义在 Manage 文件夹下的 AddNews.aspx 文件中，其关键的 HTML 代码如下。

```
1. <div>
2. 类别： <asp:DropDownList ID="Ttype" runat="server" Width="100px" AutoPostBack="True" Height="16px" />
3. 创建新类别

4. 文章标题： <asp:TextBox ID="txtTitle" runat="server" Width="157px" />
5. <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="txtTitle" ErrorMessage="*文章标题不能为空" />
6.

7. <CE:Editor ID="Editor1" runat="server" EditorWysiwygModeCss="..css/example.css" AutoConfigure="Simple" Width="430" />
8. <FrameStyle BackColor="White" BorderColor="#DDDDDD" BorderWidth="1px" BorderStyle="Solid" Height="100%" Width="100%" />
9. </div>
```

```

10. <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ControlToValidate="Editor1" ErrorMessage="文
章内容不能位空"></asp:RequiredFieldValidator>
11.

12. <asp:Button ID="btnAdd" runat="server" Text="添加文章" onclick="btnAdd_Click" />
13. <asp:Button ID="btndraft" runat="server" Text="添加到草稿箱" onclick="btndraft_Click" />
14. </div>

```

代码说明：第 2 行定义显示类别的服务器下拉列表控件 Ttype。第 3 行定义一个创建新类别的超链接。第 4 行定义输入文章标题的服务器文本框控件 txtTitle。第 5 行定义验证控件 RequiredFieldValidator1 验证文本框不能为空。第 7~9 行定义一个自定义文本编辑器控件 Editor1 并设置其外观样式属性。第 10 行定义验证控件 RequiredFieldValidator2 验证编辑器中内容不得为空。第 12 和 13 行定义两个按钮控件用于添加文章和添加文章到草稿箱的操作。

添加博客页面的后台代码定义在 Manage 文件夹下的 AddNews.aspx.cs 文件中，其主要的代码如下。

```

1. BlogDataClassesDataContext bdc = new BlogDataClassesDataContext();
2. protected void Page_Load(object sender, EventArgs e) {
3. if (!IsPostBack) {
4. BindDrop();
5. }
6. }
7. protected void BindDrop(){
8. TType.DataSource = bdc.Class.ToList();
9. TType.DataTextField = "class_name";
10. TType.DataValueField = "class_id";
11. TType.DataBind();
12. }
13. protected void btnAdd_Click(object sender, EventArgs e){
14. if (bdc.pro_AddNew(txtTitle.Text.Trim(),text,DateTime.Now,0,0,TType.SelectedValue,false,false)>0){
15. Response.Write("<script>alert('添加文章失败！');location.href('ManageNews.aspx');</script>");
16. }
17. else {
18. ClientScriptManager script = this.ClientScript;
19. script.RegisterStartupScript(this.GetType(), "123", "<script>alert('添加文章成功');location.href('ManageNews.aspx');</script>");
20. }
21. }
22. protected void btndraft_Click(object sender, EventArgs e){
23. if ((bdc.pro_AddNew(txtTitle.Text.Trim(), Editor1.Text.Trim(), DateTime.Now, 0, 0, TType.SelectedValue, true, false) > 0) {
24. Response.Write("<script>alert('未能添加到草稿箱！')</script>");
25. }
26. else{
27. Response.Redirect("~/CuteSoft_Client/NewSave2.aspx");
28. }
29. }

```

代码说明：第 1 行实例化实体数据上下文对象 bdc。第 2 行定义处理页面 Page 加载事件的方法 Load。第 3 行判断当前加载的页面不是回传页面，第 4 行调用定义的 BindDrop 方法。第 7 行定义 BindDrop 绑定方法。第 8 行调用 bdc 中 Class 对象的 ToList 方法获得所有博客类别的列表对象，作为下拉列表控件 Ttype 的数据源。第 9 行设置 Ttype 显示的文本为博客类型名称字段的值。第 10 行设置 Ttype 列表项中的值为博客类型编号字段的值。第 11 行调用 DataBind 方法绑定数据到 Ttype。

第 13 行定义处理添加文章按钮 btnAdd 单击事件 Click 的方法。第 14 行通过调用 bdc 对象的添加文章方法 AddNew 判断是否添加失败，如果添加失败，则第 15 行给出提示信息，否则第 19 行给出添加成功的提示，并重定向页面到管理博客的页面。

第 22 行处理添加到草稿箱按钮 btnAdd 单击事件 Click 的方法。第 23 行调用 bdc 对象的 AddNew 方法判断如果没能将文章添加到草稿箱，则在第 24 行给出添加失败的提示信息，否则重定向页面到草稿箱页面。

设计后添加博客文章的页面如图 17-24 所示。



图 17-24 设计后添加博客界面

### 17.7.2 管理博客

设计管理博客页面的代码定义在 Manage 文件夹下的 ManangeNews.aspx 文件中，其关键的 HTML 代码如下。

```

1. <div>
2. <asp:Repeater ID="Repeater1" runat="server">
3. <HeaderTemplate>
4. <table style="width:100%" class="Header" border="1" >
5. <tr>
6. <th style="width:35%; height:23px" >文章标题</th>
7. <th style="width:15%">添加日期</th>
8. <th style="width:15%">文章类别</th>
9. <th style="width:15%">单击次数</th>
10. <th style="width:10%">编辑</th>
11. <th style="width:10%">删除</th>
12. </tr>
13. </table>
14. </HeaderTemplate>
15. <ItemTemplate>
16. <table style="width:100%; color:Black; border-collapse :collapse " border="1px">
17. <tr>
18. <td style="width:35%; text-align:left"><%# Eval("Title") %></td>
19. <td style="width:15%"><%# Convert.ToDateTime(Eval("AddDate")).ToString("yy-MM-dd") %></td>
20. <td style="width:15%"><%# Eval("Class") %></td>
21. <td style="width:15%"><%# Eval("Click") %></td>
22. <td style="width:10%"><a href='<%# Eval("news_id","../CuteSoft_Client/NewEdit.aspx?id={0}") %>'>编辑
</td>
23. <td style="width:10%"><asp:LinkButton ID="LinkButton1" runat="server" CommandName='<%#
Eval("news_id") %>' OnCommand="LinkButton_Command">删除</asp:LinkButton> </td>
24. </tr>
25. </table>
26. </ItemTemplate>
27. </asp:Repeater>

```

代码说明：第 2~27 行定义一个 Repeater 服务器控件 Repeater1，其中，第 3~14 行在 HeaderTemplate 模板中定义一个表格用于显示列的 6 个标题。分别是文章标题、添加日期、文章类

别、单击次数、编辑和删除。第 14~26 行在 temTemplate 模板中定义了另一个表格用于显示具体的数据值，其中第 18~21 行使用绑定表达式将数据表 News 的字段值 Title、AddDate、Class 和 Click 绑定显示。第 22 行使用绑定表单式将字段 news\_id 的值作为参数传递到编辑博客的页面。第 23 行在列中定义一个链接按钮服务器控件用于删除操作。

管理博客页面的后台代码定义在 Manage 文件夹下的 ManangeNews.aspx.cs 文件中，其主要的代码如下。

```
1. BlogDataClassesDataContext bdc = new BlogDataClassesDataContext();
2. protected void Page_Load(object sender, EventArgs e){
3. if (!IsPostBack){
4. dlbind();
5. }
6. }
7. protected void LinkButton_Command(object sender, CommandEventArgs e){
8. int id = int.Parse(e.CommandName);
9. if (bdc.pro_UpdateSX(id)>0) {
10. Response.Write("<script>alert('删除未成功！')</script>");
11. }
12. else{
13. ClientScriptManager cli = this.ClientScript;
14. cli.RegisterStartupScript(this.GetType(), "45", "<script>alert('已放入回收站！');location.href ('CallBack.aspx')</script>");
15. }
16. }
```

代码说明：第 1 行实例化实体数据上下文对象 bdc。第 2 行定义处理页面 Page 加载事件的方法 Load。第 3 行判断如果当前加载的不是回传页面，则第 4 行调用 dlbind 方法实现数据绑定和分页功能，该方法与浏览博客中的同名方法相同。第 7 行定义处理删除链接按钮命令事件 Command 的方法。第 8 行获得用户选择的博客文章编号。第 9 行调用 bdc 对象的 pro\_UpdateSX 方法判断如果删除文章到回收站失败，第 10 行给出提示信息框，否则第 13、14 行先给出操作成功的提示框，再重定向到回收站页面。

设计后的管理博客页面如图 17-25 所示。

文章标题	添加日期	文章类别	点击次数	编辑	删除
数据绑定	数据绑定	数据绑定	数据绑定	<a href="#">编辑</a>	<a href="#">删除</a>
数据绑定	数据绑定	数据绑定	数据绑定	<a href="#">编辑</a>	<a href="#">删除</a>
数据绑定	数据绑定	数据绑定	数据绑定	<a href="#">编辑</a>	<a href="#">删除</a>
数据绑定	数据绑定	数据绑定	数据绑定	<a href="#">编辑</a>	<a href="#">删除</a>
数据绑定	数据绑定	数据绑定	数据绑定	<a href="#">编辑</a>	<a href="#">删除</a>

共 Label 页 当前第 1 页 [首页](#) [上一页](#) [下一页](#) [尾页](#) 未绑定

图 17-25 设计后博客管理页面

17.7.3 编辑博客图片

设计编辑相册页面的代码定义在 Manage 文件夹下的 PhotoEdit.aspx 文件中，其关键的 HTML 代码如下。

```
1. <div>
2. <asp:DataList ID="dlphotoinfo" runat="server" Width="100%">
3. <ItemTemplate>
4. <table border="1" cellpadding="0" cellspacing="0" style="width: 200px; border-collapse: collapse" align="center">
5. <tr><td>
6. <a href='<#Eval("url")%>' target="_blank"> <img alt='单击放大' src='<#Eval("url")%>' style="width: 390px;
```

```

height: 225px; border: 1px solid #fff" /></td>
7. </tr>
8. <tr><td>
9. <p style="text-align:left; margin:0; padding:0;">照片名称: <asp:TextBox ID="txttitle" runat="server"
Text='<%=#Eval("title")%>' Width="107px"></asp:TextBox>
10. 上传日期: <asp:Label ID="date" runat="server" Text='<%=#Eval("AddDate")%>'></asp:Label></p></td>
11. </tr>
12. <tr><td><p style="text-align:left;">照片简介: </p></td></tr>
13. <tr><td height="90px">
14. <asp:TextBox ID="txtinfo" runat="server" Text='<%=#Eval("info")%>' TextMode="MultiLine" Height="80px"
Width="300px"></asp:TextBox></td>
15. </tr>
16. </table>
17. </ItemTemplate>
18. </asp:DataList>
19. <p style="text-align:center"><asp:Button ID="butsubmit" runat="server" Text="修改" ToolTip="提交修改信息"
OnClick="butsubmit_Click"></asp:Button>
20. <asp:Button ID="btnreturn" runat="server" ToolTip="返回相册列表" Text="返回" OnClick=
"btnreturn_Click1"></asp:Button></p>
21. </div>

```

代码说明: 第 2~18 行定义一个 DataList 控件显示博客图片的信息。第 3~17 行在 ItemTemplate 模板中定义一个 4 行 1 列的表格。其中, 第 5~7 行在表格第 1 行中显示博客图片。第 8~11 行在表格的第 2 行显示照片名称和上传日期。第 12~15 行在表格的第 3 行显示照片的简介。第 19~20 行定义 2 个服务器按钮控件用于修改图片名和返回相册列表。

编辑相册页面的后台代码定义在 Manage 文件夹下的 PhotoEdit.aspx.cs 文件中, 其主要的代码如下。

```

1. BlogDataClassesDataContext bdc = new BlogDataClassesDataContext();
2. protected void Page_Load(object sender, EventArgs e) {
3. if(!IsPostBack){
4. Bind();
5. }
6. }
7. protected void Bind(){
8. int id = Convert.ToInt32(Request.QueryString["id"]);
9. var result = from p in bdc.Photo
10. where p.photo_id == id
11. select p;
12. dlphotoinfo.DataSource = result;
13. dlphotoinfo.DataBind();
14. }
15. protected void butsubmit_Click(object sender, EventArgs e){
16. string title = "";
17. string info = "";
18. int id = Convert.ToInt32(Request.QueryString["id"]);
19. for (int i = 0; i < dlphotoinfo.Controls.Count; i++){
20. title = ((TextBox)dlphotoinfo.Controls[i].FindControl("txttitle")).Text;
21. info = ((TextBox)dlphotoinfo.Controls[i].FindControl("txtinfo")).Text;
22. }
23. if (bdc.pro_UpdatePhoto(id,title,info)<=0){
24. ClientScriptManager script = this.ClientScript;
25. script.RegisterStartupScript(this.GetType(), "123", "<script>alert('修改图片成功! ');location.href
('ManagePhoto.aspx');</script>");
26. }
27. else{
28. Response.Write("<script>alert('修改图片失败! ');</script>");
29. }
30. }
31. protected void btnreturn_Click1(object sender, EventArgs e){
32. Response.Redirect("ManagePhoto.aspx", true);
33. }

```

代码说明：第 1 行实例化实体数据上下文对象 bdc。第 2 行定义处理页面 Page 加载事件的方法 Load。第 3 行判断如果当前加载的不是回传页面，则第 4 行调用 Bind 方法绑定数据。第 7 行定义绑定数据的方法 Bind。第 8 行通过 Request 对象 QueryString 方法获得传递到页面的图片编号。第 9~11 行通过 LINQ 查询获得指定图片编号的博客图片对象 result。第 10 行将该对象作为 DataList 控件 dlphotoinfo 的数据源。第 11 行调用 DataBind 方法绑定数据显示。

第 15 行定义处理 butsubmit 按钮单击事件 Click 的方法。第 19~22 行循环遍历 DataList 控件中所有的内部控件，获得图片标题和图片简介的内容。第 23 行调用 bdc 对象的 pro\_UpdatePhoto 方法判断如果修改图片信息成功，则在第 25 行先显示修改成功的提示框，然后重定向到管理图片页面。否则，第 28 行显示修改图片失败的提示框。

第 31 行定义处理 btnreturn 按钮单击事件 Click 的方法。第 32 行重新定向到管理图片的页面。运行后编辑相册页面如图 17-26 所示。



图 17-26 运行后编辑相册页面

#### 17.7.4 管理评论

在管理评论功能中使用了 AJAX Control Toolkit 工具集中的 CalendarExtender 控件实现选择查询日期时页面无刷新的效果。设计管理评论页面的代码定义在 Manage 文件夹下的 ManageMessage.aspx 文件中，其关键的 HTML 代码如下。

```

1. <cc1:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server"> </cc1:ToolkitScriptManager>
2. <div style="width:100%; text-align:left">
3. 关键字: <asp:TextBox ID="TextBox1" runat="server" Height="13px" Width="116px"></asp:TextBox>

4. 开始日期: <asp:TextBox ID="TextBox2" runat="server" Height="13px" Width="116px"></asp:TextBox>
5. <cc1:CalendarExtender ID="CalendarExtender1" runat="server" TargetControlID="TextBox2" Format="yyyy-MM-dd"
 CssClass="MyCalendar"></cc1:CalendarExtender>
6. 结束日期: <asp:TextBox ID="TextBox3" runat="server" Height="13px" Width="116px"></asp:TextBox>
7. <cc1:CalendarExtender ID="CalendarExtender2" runat="server" TargetControlID="TextBox3" Format="yyyy-MM-dd"
 CssClass="MyCalendar"></cc1:CalendarExtender>
8. <asp:Button ID="Button1" runat="server" Text="查询" onclick="Button1_Click" />
9. <asp:ListView ID="ListView1" runat="server" DataKeyNames="id" DataSourceID="LinqDataSource1" style="width:100%; text-align:left">
</asp:ListView>
10. <asp:LinqDataSource ID="LinqDataSource1" runat="server"
 ContextTypeName="Web.BlogDataClassesDataContext" EnableDelete="True"
 EntityTypeName="" TableName="Message"></asp:LinqDataSource>
11. </div>
12. <asp:QueryExtender ID="QueryExtender1" runat="server" TargetControlID="LinqDataSource1">
13. <asp:SearchExpression DataFields="body" SearchType="StartsWith">

```

```
14. <asp:ControlParameter ControlID="TextBox1" />
15. </asp:SearchExpression>
16. <asp:RangeExpression DataField="date" MaxType="Inclusive" MinType="Inclusive" >
17. <asp:ControlParameter ControlID="TextBox2" DbType="DateTime" />
18. <asp:ControlParameter ControlID="TextBox3" DbType="DateTime" />
19. </asp:RangeExpression>
20. </asp:QueryExtender>
```

代码说明：第 1 行添加一个 AJAX Control ToolKit 工具集中 ToolkitScriptManager 用于页面中使用的 AJAX Control ToolKit 工具集中控件的管理。第 3 行添加一个文本框控件 TextBox1 给用户输入查询关键字。第 4 和第 6 行各添加一个文本框控件用于获得查询的开始日期和结束日期。

第 5 和第 7 行各自添加一个 AJAX 日历扩展控件 CalendarExtender，并设置关联文本控件的 ID、日历文本显示格式和 CSS 类选择的类名。第 8 行添加一个服务器按钮控件 Button 用于查询的操作。第 9 行添加一个服务器控件 ListView 显示博客评论的信息。第 10 行添加一个 LinqDataSource 控件获取数据库 Message 博客评论表的数据。

第 12 行添加一个服务器 QueryExtender 查询筛选控件实现页面查询功能，并设置其获取数据的关联控件为 LinqDataSource1。第 13~15 行定义该控件搜索字符串筛选表达式 SearchExpression，其中，第 13 行设置绑定搜索字段为 Message 表中的 body 字段，设置搜索类型为从字段的任意位置开始搜索。第 14 行设置从文本框控件 TextBox1 获得查询的控件参数。第 16~19 行定义 QueryExtender1 控件搜索值范围的筛选表达式 RangeExpression。其中，第 16 行设置控件的绑定搜索字段为 Message 表中的 date 字段以及设置搜索范围包括最大值和最小值。第 17 行和第 18 行设置从文本框控件 TextBox2 和 TextBox3 分别获得查询的控件参数。

运行后的管理评论的页面如图 17-27 所示。



图 17-27 运行后管理评论页面

本模块中其他功能的实现与上面介绍的基本相同，故不再重复介绍，读者可以浏览光盘中的源代码进行学习。

# 第 18 章 网上音乐商店

随着网络的不断发展,网络购物已经日渐成为年轻消费者的一种生活习惯,人们已经开始认同这种在网上消费的方式。各种商家竞相在网络上建立网上商店。本章介绍的网上音乐商店就是这一背景下的产物。它结合了网络技术和销售唱片的实体商店优点,减少了流通环节,降低了交易成本,打破时空和地域的限制。使用户可以通过网络在音乐商店中挑选和购买自己心仪的唱片,同时,可以体会到网络给我们带来的购物乐趣。

## 18.1 系统分析与设计

网上音乐商店是一个基于 Internet 的轻量级的在线音乐商店,使用 ASP.NET 4.0 中的 ASP.NET MVC 2 进行开发。系统运用了完整的 ASP.NET MVC 2 功能和数据实体框架来访问数据库的技术,同时使用 ASP.NET AJAX 的辅助方法实现了部分页面的局部刷新功能。

### 18.1.1 系统需求分析

根据网上音乐商店的日常经营和管理。本系统的用户主要有两类:一类是网站的用户,也可以说是网上音乐商店的顾客;另一类是网站的管理员。二者的身份不同,权限不同,所以,具体的功能需求也不同。

对于商店的顾客来说,实现的具体功能说明如下。

- ① 顾客进入音乐商店的首页。在首页中可以浏览商店中唱片专辑的分类和当前最为热门的唱片专辑。
- ② 顾客选择不同的专辑分类可以进入相应类型的页面浏览该类下的唱片专辑。
- ③ 顾客通过单击唱片的图片或名称可以查看该唱片专辑的详细信息。如果想购买,可以将选择的专辑添加到购物车中。
- ④ 在购物车中顾客可以进行清除购买的专辑和继续添加唱片的操作。选择完毕可以进行结账。
- ⑤ 结账前,顾客必须进行注册成为音乐商店的用户才可以进行结账操作。
- ⑥ 顾客注册成为用户后,在首页中输入注册的用户账号和密码,通过身份验证才能进入结账页面。
- ⑦ 在结账页面输入送货的配送信息,提交订单并获得订单号码。

对于网站管理员而言,主要对网站的后台进行日常的管理,实现的具体功能说明如下。

- ① 管理员必须在管理页面进行登录,输入自己的账户名和密码。只有通过身份验证才能进入唱片专辑管理的页面。

② 管理员能够对网站的唱片专辑进行管理, 包括添加新的唱片专辑、编辑现有的唱片和删除唱片的操作。

③ 管理员可以通过网站配置工具对网站的用户进行管理, 包括角色和用户信息的管理。

### 18.1.2 系统模块设计

根据上述的系统需求分析和功能, 我们把本系统分成数据访问、实体类、用户登录、购物车和后台管理五个主要的模块。其中, 数据访问模块使用 ASP.NET MVC 中的 Controller (控制器) 来实现, 实体类模块主要使用 ASP.NET MVC 中的 Model (模型) 来实现。而页面的显示则使用 ASP.NET MVC 中的 View (视图) 来实现。

各模块所包含的文件及其功能如表 18-1 所示。

表 18-1 网上音乐商店各模块一览表

模块名	文件名	功能描述
数据访问模块	Controllers/AccountController.cs	用户账户管理控制器文件
	Controllers/CheckoutController.cs	用户结账控制器文件
	Controllers/HomeController.cs	首页控件器文件
	Controllers/ShoppingCartController.cs	购物车控制器文件
	Controllers/StoreController.cs	商店唱片浏览控制器文件
	Controllers/StoreManagerController.cs	后台管理控制器文件
实体类模块	Models/AccountModels.cs	用户账户模型文件
	Models/ Order.cs	订单详情实体类文件
	Models/ShoppingCart.cs	购物车实体类文件
	Models/StoreDB.edmx/StoreDB.Designer.cs	ADO.NET 实体数据模型自动生成的数据库实体映射文件
	ViewModels/ShoppingCartRemoveView Model.cs	删除购物车视图模型
	ViewModels/ShoppingCartViewModel.cs	购物车视图模型
	ViewModels/StoreBrowseViewModel.cs	浏览唱片视图模型
	ViewModels/StoreIndexViewModel.cs	唱片索引视图模型
	ViewModels/StoreManagerViewModel.cs	唱片管理视图模型
用户登录模块	Views/Account/ LogOn.aspx	用户登录视图页面
	Views/Account/Register.aspx	用户注册视图页面
	Views/Home/ Index.aspx	网站首页视图页面
购物车模块	Views/Store/ Browse.aspx	唱片专辑浏览视图页面
	Views/Store/ Details.aspx	唱片详情浏览视图页面
	Views/Store/ Index.aspx	根据类型浏览唱片视图页面

(续表)

模块名	文件名	功能描述
购物车模块	Views/ShoppingCart/Index.aspx	购物车视图页面
	Views/Checkout/Complete.aspx	完成结账视图页面
	Views/Checkout/AddressAndPayment.aspx	填写订单视图页面
后台管理模块	Views/StoreManager/Create.aspx	新建唱片视图页面
	Views/StoreManager/Delete.aspx	删除唱片视图页面
	Views/StoreManager/Deleted.aspx	删除唱片成功视图页面
	Views/StoreManager/Edit.aspx	编辑唱片视图页面
	Views/StoreManager/Index.aspx	管理产品视图页面
其他辅助功能	Helpers/ HtmlHelpers.cs	HTML 帮助扩展类文件
	Scripts	JQuery 和 ASP.NET AJAX 脚本库
	Shared/EditorTemplates/Album.ascx	编辑唱片用户控件
	Shared/Error.aspx	显示错误的视图页面
	Shared/Site.Master	系统母版页文件
	Views/Store/GenreMenu.ascx	专辑分类菜单用户控件
	Views/ShoppingCart/CartSummary.ascx	进入购物车用户控件
	App_Data/MvcMusicStore.mdf	网站数据库文件（SQL Server 2005）
	Content/ Images	网站图片文件夹
	Content/ Site.css	网站样式表文件
	Web.config	网站配置文件
	Global.asax/ Global.asax.cs	应用程序文件

18.1.3 系统运行演示

运行本系统，出现的首先是如图 18-1 所示网站首页。



图 18-1 网站首页

在首页中显示了菜单、唱片类型和最新热门唱片。顾客可以选择“摇滚”类型的链接，进入如图 18-2 所示的摇滚唱片专辑的浏览页面。



图 18-2 专辑浏览页面

在唱片专辑的浏览页面，顾客可以单击所选唱片的图片或文字链接进入如图 18-3 所示的唱片详情浏览页面。



图 18-3 唱片详情页面

在唱片详情页面，单击“添加到购物车”按钮，可进入如图 18-4 所示购物车的界面。



图 18-4 购物车页面

在购物车页面，顾客可以浏览自己购物的内容、进行移除商品或继续购买商品的操作。购物完毕后，顾客可以单击“结账”按钮，进入如图 18-5 所示的用户登录的页面。



图 18-5 登录页面

由于没有在网站注册过用户身份的顾客无法登录结账，顾客必须在登录界面中单击“注册”链接进入如图 18-6 所示注册页面注册成为网站的用户。

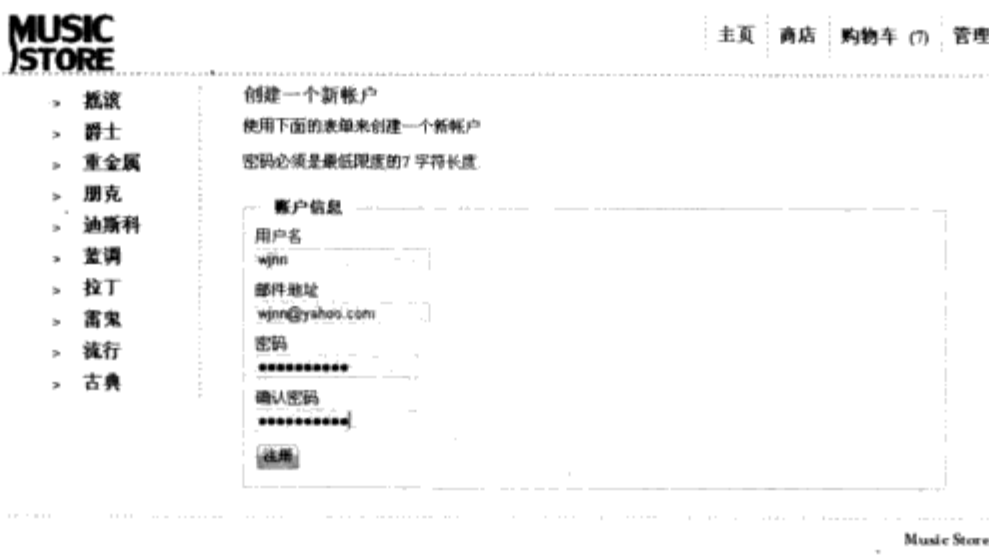


图 18-6 注册页面

在注册页面，输入用户名、密码、确认密码和电子邮件地址，单击“注册”按钮，完成注册操作进入如图 18-7 所示的结账页面。

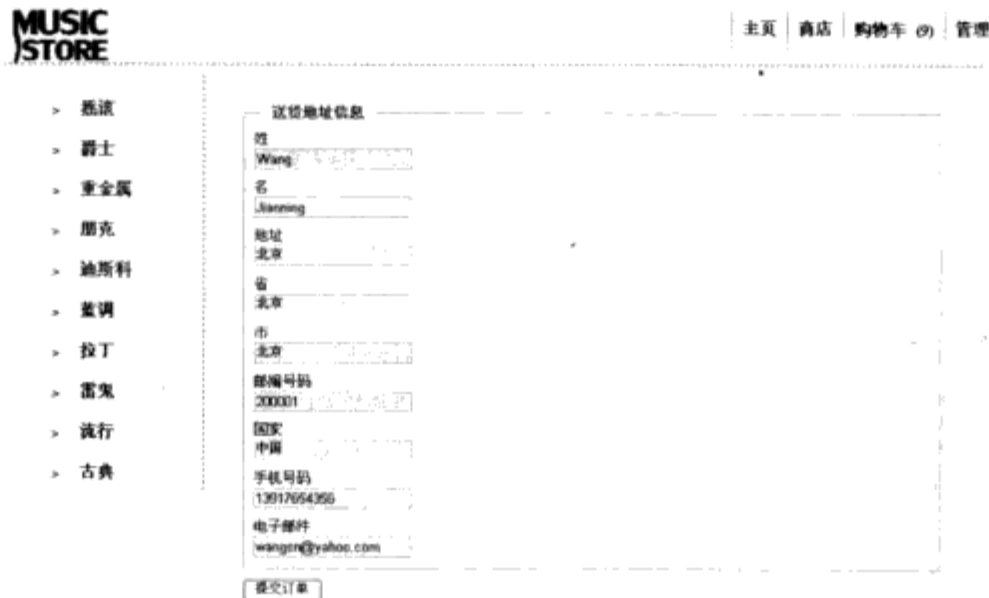


图 18-7 结账页面

在结账页面，填写订单，输入姓名、地址、邮编、省市、国家、手机、电子邮件和优惠码等信息，最后单击“提交订单”按钮，进入如图 18-8 所示的完成购物界面，在该页面中用户可以

得到订单的编号和选择是否购物。



图 18-8 完成结账页面

对本系统的另一类用户——管理员来说，同样必须要在上面的登录页面中输入用户名和密码，通过身份验证后才能进入如图 18-9 所示的后台管理页面。

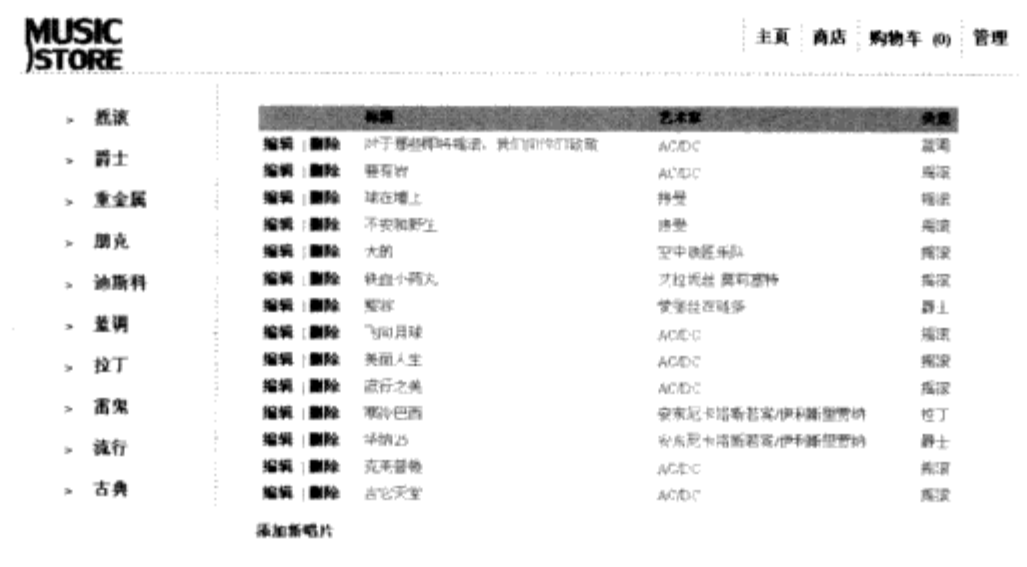


图 18-9 管理唱片页面

在后台管理页面单击“添加新产品”链接，管理员可进入如图 18-10 所示的创建唱片专辑的页面。在该页面中选择唱片的类型、艺术家，输入唱片标题、价格和唱片图片路径，最后单击“保存”按钮完成创建操作并返回到后台管理页面。

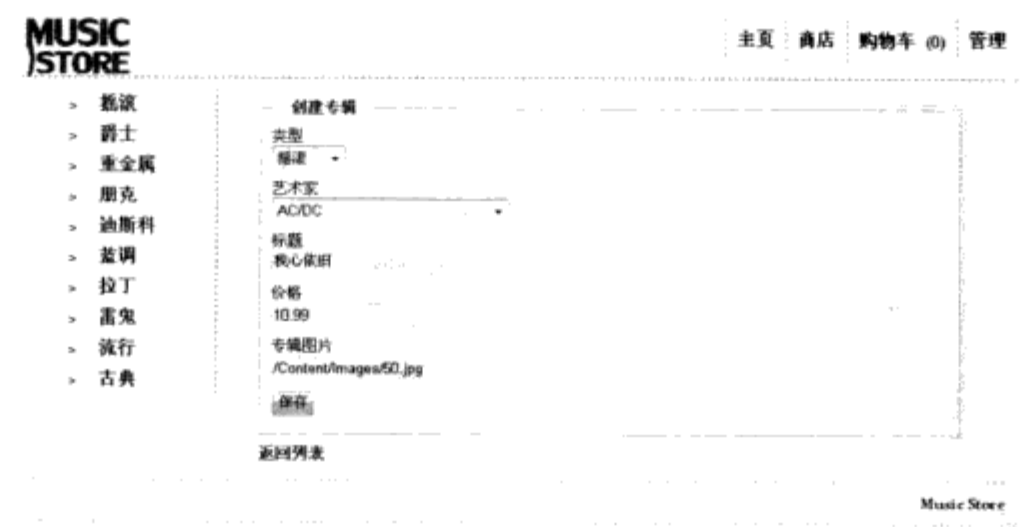


图 18-10 新建唱片页面

在后台管理页面单击“编辑”链接，可进入如图 18-11 所示的编辑唱片页面。管理员可修改唱

片专辑的类型、艺术家、唱片的标题、价格和唱片图片的路径，最后单击“保存”按钮，完成编辑唱片的操作并返回到后台管理页面。

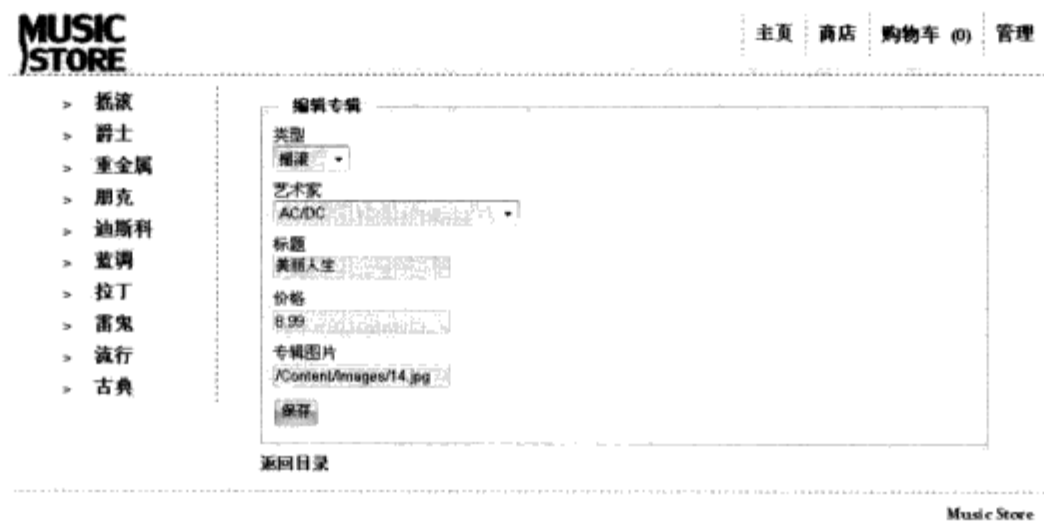


图 18-11 编辑唱片页面

在后台管理页面单击“删除”链接，管理员可进入如图 18-12 所示的删除专辑页面。

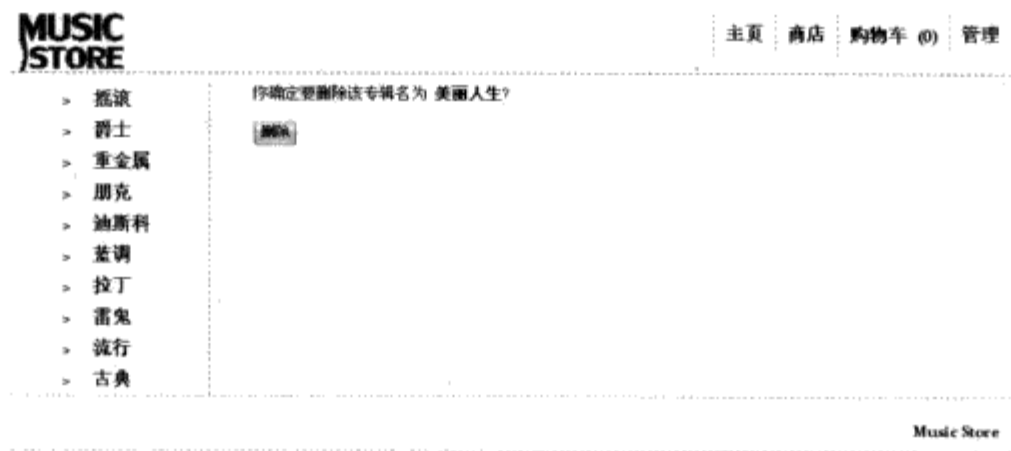


图 18-12 删除唱片页面

在删除专辑页面中确认删除后，单击“删除”按钮后，进入如图 18-13 完成删除页面。页面中显示删除唱片专辑成功的提示，单击“返回专辑列表”连接可回到后台管理页面。



图 18-13 完成删除页面

本系统中其他页面与上述演示的页面相似，所以，不再一一演示，读者可以运行随书光盘的源代码进行学习。

## 18.2 系统数据库设计

根据系统的需求分析，我们要对数据库进行合理的设计。首先，在 Sql Server 2005 中建立一个名为“MvcMusicStore”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [MvcMusicStore] //创建数据库
USE [MvcMusicStore] //使用数据库
```

### 18.2.1 数据库表设计

为满足本系统的功能需求，设计数据库表如下。

(1) 唱片专辑表（Album），用来存放网站中所有的唱片专辑的信息。该表的字段结构如表 18-2 所示。

表 18-2 Album 表结构

字段	中文描述	数据类型	是否为空	备注
AlbumId	唱片编号	int	否	主键
GenreId	类型编号	int	否	外键
ArtistId	歌手编号	int	否	外键
Title	唱片标题	nvarchar(160)	否	
Price	唱片价格	numeric(10,2)	否	
AlbumArtUrl	唱片图片路径	nvarchar(1024)	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Album](
[AlbumId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY, //设置主键,
[GenreId] [int] NOT NULL,
[ArtistId] [int] NOT NULL,
[Title] [nvarchar](160) COLLATE Chinese_PRC_CI_AS NOT NULL,
[Price] [numeric](10, 2) NOT NULL,
[AlbumArtUrl] [nvarchar](1024) COLLATE Chinese_PRC_CI_AS NULL CONSTRAINT [DF_Album_AlbumArtUrl] DEFAULT
(N'/Content/Images/placeholder.gif'),
CONSTRAINT [FK_Album_ArtistId_276EDEB3] FOREIGN KEY([ArtistId])
REFERENCES [dbo].[Artist] ([ArtistId]), //设置外键
CONSTRAINT [FK_Album_Genre] FOREIGN KEY([GenreId])
REFERENCES [dbo].[Genre] ([GenreId]) //设置外键
)
```

(2) 演唱者信息表（Artist），用来保存网站中所有唱片演唱者的详细信息，该表的字段结构如表 18-3 所示。

表 18-3 Artist 表结构

字段	中文描述	数据类型	是否为空	备注
ArtistId	演唱者编号	int	否	主键
Name	演唱者姓名	nvarchar(120)	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Artist](
 [ArtistId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY, //设置主键,
 [Name] [nvarchar](120) COLLATE Chinese_PRC_CI_AS NULL
)
```

（3）唱片类型表（Genre），用来记录网站中所有唱片类型的详细信息，该表的字段结构如表 18-4 所示。

表 18-4 Genre 表结构

字段	中文描述	数据类型	是否为空	备注
GenreId	类型编号	int	否	主键
Name	类型名称	nvarchar(120)	是	
Description	类型描述	nvarchar(4000)	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Genre](
 [GenreId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY, //设置主键,
 [Name] [nvarchar](120) COLLATE Chinese_PRC_CI_AS NULL,
 [Description] [nvarchar](4000) COLLATE Chinese_PRC_CI_AS NULL
)
```

（4）订单详情表（OrderDetails），用于保存所有用户购买订单的详细信息，该表的字段结构如表 18-5 所示。

表 18-5 OrderDetails 表结构

字段	中文描述	数据类型	是否为空	备注
OrderDetailId	订单详情编号	int	否	主键
OrderId	订单编号	int	否	外键
AlbumId	唱片编号	int	否	外键
UnitPrice	唱片单价	numeric(10,2)	否	
Quantity	订单数量	int	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[OrderDetail](
 [OrderDetailId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY, //设置主键,
 [OrderId] [int] NOT NULL,
 [AlbumId] [int] NOT NULL,
 [Quantity] [int] NOT NULL,
 [UnitPrice] [numeric](10, 2) NOT NULL,
 CONSTRAINT [FK_InvoiceLi_Invoi_2F10007B] FOREIGN KEY([OrderId])
REFERENCES [dbo].[Order] ([OrderId]), //设置外键
CONSTRAINT [FK_InvoiceLine_Album] FOREIGN KEY([AlbumId])
REFERENCES [dbo].[Album] ([AlbumId]) //设置外键
)
```

(5) 订单表 (Order)，用于保存用户购买唱片订单的信息，该表的字段结构如表 18-6 所示。

表 18-6 Order 表结构

字段	中文描述	数据类型	是否为空	备注
OrderId	订单编号	int	否	主键
Username	用户姓名	nvarchar(256)	是	
OrderDate	订单生成日期	datetime	否	
FirstName	用户的姓	nvarchar(160)	是	
LastName	用户的名	nvarchar(160)	是	
Address	用户地址	nvarchar(70)	是	
City	用户所在城市	nvarchar(40)	是	
State	用户所在的州	nvarchar(40)	是	
PostalCode	用户邮政编码	nvarchar(10)	是	
Country	用户所在国家	nvarchar(40)	是	
Phone	用户联系电话	nvarchar(24)	是	
Email	用户电子邮件	nvarchar(160)	是	
Total	订单总价	numeric(10.2)	否	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Order](
 [OrderId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY, //设置主键,
 [OrderDate] [datetime] NOT NULL,
 [Username] [nvarchar](256) COLLATE Chinese_PRC_CI_AS NULL,
 [FirstName] [nvarchar](160) COLLATE Chinese_PRC_CI_AS NULL,
 [LastName] [nvarchar](160) COLLATE Chinese_PRC_CI_AS NULL,
 [Address] [nvarchar](70) COLLATE Chinese_PRC_CI_AS NULL,
 [City] [nvarchar](40) COLLATE Chinese_PRC_CI_AS NULL,
 [State] [nvarchar](40) COLLATE Chinese_PRC_CI_AS NULL,
 [PostalCode] [nvarchar](10) COLLATE Chinese_PRC_CI_AS NULL,
 [Country] [nvarchar](40) COLLATE Chinese_PRC_CI_AS NULL,
 [Phone] [nvarchar](24) COLLATE Chinese_PRC_CI_AS NULL,
 [Email] [nvarchar](160) COLLATE Chinese_PRC_CI_AS NULL,
 [Total] [numeric](10, 2) NOT NULL
)
```

(6) 购物车信息表 (Cart)，用于保存用户购物车中所选择的唱片信息。该表的字段结构如表 18-7 所示。

表 18-7 Cart 表结构

字段	中文描述	数据类型	是否为空	备注
RecordId	记录编号	int	否	主键
CartId	购物车编号	varchar(50)	否	
AlbumId	唱片编号	int	否	外键
Count	数量	int	否	
DateCreated	创建日期	datetime	否	

创建上表的语句如下：

```
CREATE TABLE [dbo].[Cart](
 [RecordId] [int] IDENTITY(1,1) NOT NULL //设置主键,
 [CartId] [varchar](50) COLLATE Chinese_PRC_CI_AS NOT NULL,
 [AlbumId] [int] NOT NULL,
 [Count] [int] NOT NULL,
 [DateCreated] [datetime] NOT NULL,
 CONSTRAINT [FK_Cart_Album] FOREIGN KEY([AlbumId])
 REFERENCES [dbo].[Album] ([AlbumId]) //设置外键
)
```

18.2.2 Visual Studio 2010 自动生成的数据库

本系统使用了 ASP.NET 网站配置工具来实现网站用户身份的验证和角色管理。所以，在 Visual Studio 2010 中会自动生成一个系统自带数据库“ASPNETDB”保存所需要的数据，所有的用户信息、个性化信息和基本配置等都保存在该数据库中。

在该数据库中共有 11 张数据表，表的名称都以“aspnet\_”开头。本系统中主要使用到了其中的 4 张表，说明如下。

(1) aspnet\_Users 表，用于快速提取记用户的信息，该表的字段结构如表 18-8 所示。

表 18-8 aspnet\_Users 表结构

字段	中文描述	数据类型	是否为空	备注
ApplicationId	应用程序编号	uniqueidentifier	否	外键
UserId	用户编号	uniqueidentifier	否	主键
UserName	用户名称	nvarchar(256)	否	
LoweredUserName	小写用户名	nvarchar(256)	否	
MobileAlias	移动用户别名	nvarchar(16)	是	
IsAnonymous	是否是匿名用户	bit	否	
LastActivityDate	最后访问日期	datetime	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[aspnet_Users](
 [ApplicationId] [uniqueidentifier] NOT NULL,
 [UserId] [uniqueidentifier] NOT NULL DEFAULT (newid())PRIMARY KEY, //设置主键
 [UserName] [nvarchar](256) NOT NULL,
 [LoweredUserName] [nvarchar](256) NOT NULL,
 [MobileAlias] [nvarchar](16) NULL DEFAULT (NULL),
 [IsAnonymous] [bit] NOT NULL DEFAULT ((0)),
 [LastActivityDate] [datetime] NOT NULL,
 FOREIGN KEY([ApplicationId]) REFERENCES [dbo].[aspnet_Applications] ([ApplicationId]), //设置外键
)
```

(2) aspnet\_Roles 表，用于保存系统设置的角色信息，该表的字段结构如表 18-9 所示。

表 18-9 aspnet\_Roles 表结构

字段	中文描述	数据类型	是否为空	备注
ApplicationId	应用程序编号	uniqueidentifier	否	外键
RoleId	角色编号	uniqueidentifier	否	主键
RoleName	角色名称	nvarchar(256)	否	
LoweredRoleName	小写的角色名称	nvarchar(256)	否	
Description	角色描述	int	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[aspnet_Roles](
 [ApplicationId] [uniqueidentifier] NOT NULL,
 [RoleId] [uniqueidentifier] NOT NULL DEFAULT (newid()) PRIMARY KEY, //设置主键
 [RoleName] [nvarchar](256) NOT NULL,
 [LoweredRoleName] [nvarchar](256) NOT NULL,
 [Description] [nvarchar](256) NULL,
 FOREIGN KEY([ApplicationId]) REFERENCES [dbo].[aspnet_Applications] ([ApplicationId]) //设置外键
)
```

(3) aspnet\_Membership 表，用于记录用户的详细信息，该表的字段结构如表 18-10 所示。

表 18-10 aspnet\_Membership 表结构

字段	中文描述	数据类型	是否为空	备注
ApplicationId	应用程序编号	uniqueidentifier	否	外键
UserId	用户编号	uniqueidentifier	否	主键
Password	用户密码	nvarchar(128)	否	
PasswordFormat	密码格式	int	否	
PasswordSalt	密码加密格式	nvarchar(128)	否	
MobilePIN	移动用户 PIN 码	nvarchar(16)	是	
Email	邮件	nvarchar(256)	是	
LoweredEmail	小写的邮件	nvarchar(256)	是	
PasswordQuestion	找回密码问题	nvarchar(256)	是	
PasswordAnswer	找回密码答案	nvarchar(128)	是	
IsApproved	是否可进行验证	bit	否	
IsLockedOut	是否因锁定不验证	bit	否	
CreateDate	创建时间	datetime	否	
LastLoginDate	最后登录时间	datetime	否	
LastPasswordChangedDate	最后更改密码时间	datetime	否	
LastLockoutDate	最后锁定时间	datetime	否	
FailedPasswordAttemptCount	失败尝试次数	int	否	
FailedPasswordAttemptWindowStart	密码失败尝试窗口打开时间	datetime	否	

(续表)

字段	中文描述	数据类型	是否为空	备注
FailedPasswordAnswerAttemptCount	找回密码时尝试次数	int	否	
FailedPasswordAnswerAttemptWindowStart	找回密码时尝试窗口打开时间	datetime	否	
Comment	最后修改时间	ntext	是	

创建上表的语句如下：

```
CREATE TABLE [dbo].[aspnet_Membership](
 [ApplicationId] [uniqueidentifier] NOT NULL,
 [UserId] [uniqueidentifier] NOT NULL PRIMARY KEY, //设置主键
 [Password] [nvarchar](128) NOT NULL,
 [PasswordFormat] [int] NOT NULL DEFAULT ((0)),
 [PasswordSalt] [nvarchar](128) NOT NULL,
 [MobilePIN] [nvarchar](16) NULL,
 [Email] [nvarchar](256) NULL,
 [LoweredEmail] [nvarchar](256) NULL,
 [PasswordQuestion] [nvarchar](256) NULL,
 [PasswordAnswer] [nvarchar](128) NULL,
 [IsApproved] [bit] NOT NULL,
 [IsLockedOut] [bit] NOT NULL,
 [CreateDate] [datetime] NOT NULL,
 [LastLoginDate] [datetime] NOT NULL,
 [LastPasswordChangedDate] [datetime] NOT NULL,
 [LastLockoutDate] [datetime] NOT NULL,
 [FailedPasswordAttemptCount] [int] NOT NULL,
 [FailedPasswordAttemptWindowStart] [datetime] NOT NULL,
 [FailedPasswordAnswerAttemptCount] [int] NOT NULL,
 [FailedPasswordAnswerAttemptWindowStart] [datetime] NOT NULL,
 [Comment] [ntext] NULL,
 FOREIGN KEY([ApplicationId]) REFERENCES [dbo].[aspnet_Applications] ([ApplicationId]), //设置外键
)
```

(4) aspnet\_UsersInRoles 表，是用户和角色的关联表，该表的字段结构如表 18-11 所示。

表 18-11 aspnet\_UsersInRoles 表结构

字段	中文描述	数据类型	是否为空	备注
UserId	用户编号	uniqueidentifier	否	主键
RoleId	角色编号	uniqueidentifier	否	主键

创建上表的语句如下：

```
CREATE TABLE [dbo].[aspnet_UsersInRoles](
 [UserId] [uniqueidentifier] NOT NULL PRIMARY KEY, //设置主键
 [RoleId] [uniqueidentifier] NOT NULL PRIMARY KEY, //设置主键
)
```

18.2.3 数据库表关系

在本系统创建的“MvcMusicStore”数据库中，各表间存在则一些引用和关联关系，说明如下。

① 唱片专辑表（Album）通过使用外键 GenreId 与唱片类型表（Genre）形成关联。

- ② 唱片专辑表 (Album) 通过使用外键 ArtistId 与演唱者信息表 (Artist) 形成关联。
- ③ 购物车信息表 (Cart) 通过使用外键 AlbumId 与唱片专辑表 (Album) 形成关联。
- ④ 订单详情表 (OrderDetails) 通过使用外键 OrderId 与订单表 (Order) 形成关联。
- ⑤ 订单详情表 (OrderDetails) 通过使用外键 AlbumId 与唱片专辑表 (Album) 形成关联。

各个表之间的关系如图 18-14 所示。

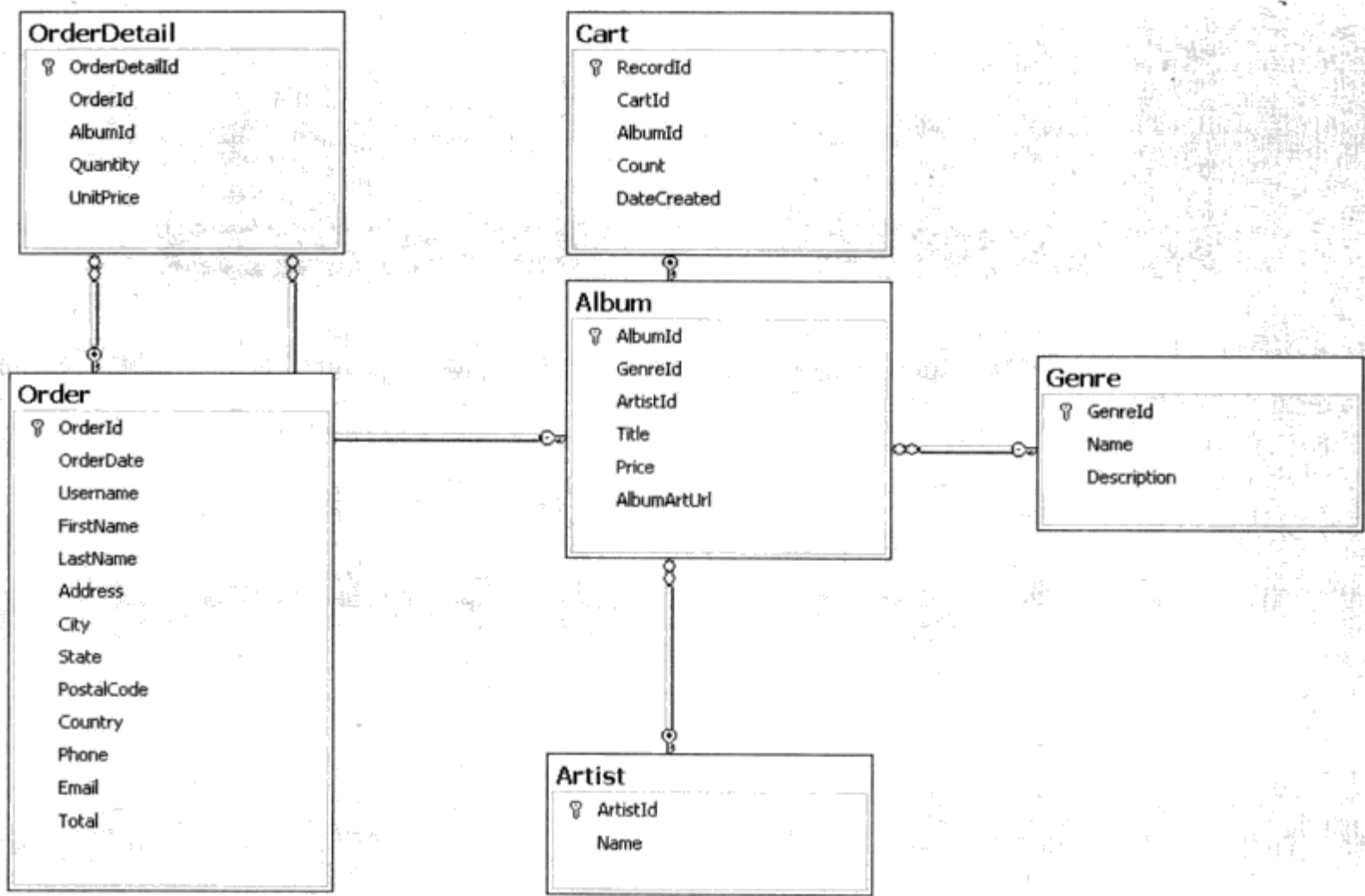


图 18-14 数据库表关系

### 18.3 实体类模块

本系统的实体类模块由 ADO.NET 实体数据模型自动生成的数据库实体映射文件 StoreDB.edmx、自定义的实体类和视图模型组成。本节分别说明这三类实体类模块的创建过程。

#### 18.3.1 创建数据库实体类映射

创建数据库实体映射类的具体步骤说明如下。

**01** 在 Visual Studio 2010 的“解决方案资源管理器”中，用鼠标右键单击“Models”文件夹，在弹出的快捷菜单中单击“添加”|“新建项”命令，在打开的如图 18-15 所示的“添加新项”对话框中先选择已安装模板列表中的“Visual C#”，再单击项目列表中“ADO.NET 实体数据模型”设置名称为“StoreDB.edmx”，然后单击“添加”按钮。



图 18-15 “添加新项”对话框

**02** 打开如图 18-16 所示的实体数据模型向导——“选择模型内容”对话框，选择“从数据库生成”，表明 ADO.NET 实体框架从数据库直接生成实体数据模型，然后单击“下一步”按钮。

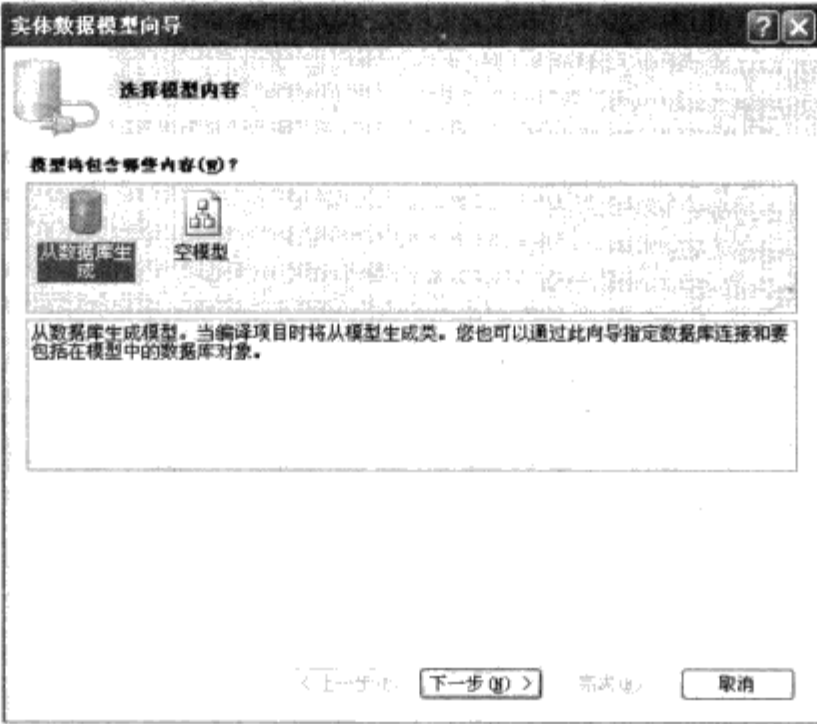


图 18-16 选择模型内容对话框

**03** 打开如图 18-17 所示的“选择你的数据连接”对话框，选择“y kz-20091130atm.MvcMusicStore.mdf”数据库（.mdf 文件前是服务器的名称，根据具体服务器名称会有所变化），选中“将 Web.Config 中的实体连接设置另存为”复选框，在文本框中输入“MvcStoreEntitise”，单击“下一步”按钮。

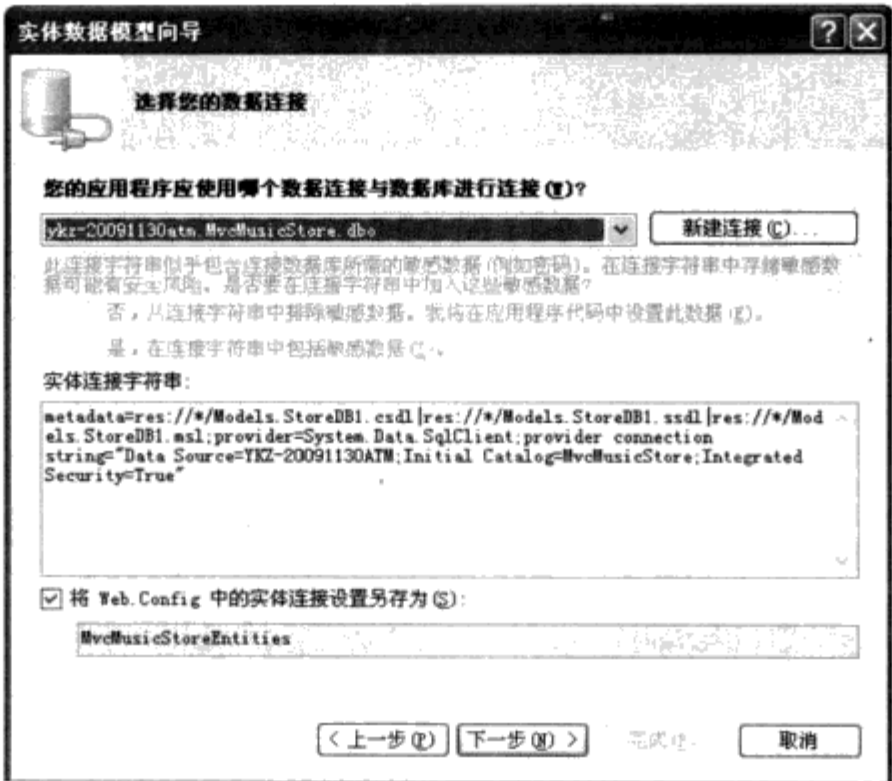


图 18-17 选择数据连接对话框

04 弹出如图 18-18 所示的“选择数据库对象”对话框，对应于本系统创建的数据库（不包括 Visual Studio 2010 自己创建的数库 ASPNETDB），选择“表”、“Album”、“Artist”、“Genre”、“OrderDetails”、“Order”和“Cart”，然后单击“完成”按钮。



图 18-18 选择数据库对象对话框

05 ADO.NET 实体框架即可在 Models 文件夹中自动生成如图 18-19 所示的“StoreDB.edmx”实体数据模型和 StoreDB.Designer.cs 的文件。在 StoreDB.Designer.cs 的文件自动成“MvcMusicStore”数据库中 5 张数据表的属性和各种操作的方法。我们可以在 Controller 控制器中直接调用这些属性和方法对数据库进行访问并处理数据。



图 18-19 实体数据模型

### 18.3.2 创建自定义的实体类

由于使用 ADO.NET 实体数据模型自动生成的 StoreDB.Designer.cs 文件的功能还不能全部满足本系统的实际需要，所以我们在 Models 文件夹中自定义了两个实体类：ShoppingCart 购物车操作类和 Order 订单类。

#### 1. 购物车操作类

购物车操作类 ShoppingCart 定义了购物车操作类的属性和各种操作方法，代码保存在 Models 文件夹下的 ShoppingCart.cs 文件中。

(1) 定义该类的属性和获得购物车编号的方法，关键代码如下。

```

1. MusicStoreEntities storeDB = new MusicStoreEntities();
2. string shoppingCartId { get; set; }
3. public const string CartSessionKey = "CartId";
4. public static ShoppingCart GetCart(HttpContextBase context){
5. var cart = new ShoppingCart();
6. cart.shoppingCartId = cart.GetCartId(context);
7. return cart;
8. }
9. public String GetCartId(HttpContextBase context){
10. if (context.Session[CartSessionKey] == null){
11. if (!string.IsNullOrEmpty(context.User.Identity.Name)){
12. context.Session[CartSessionKey] = context.User.Identity.Name;
13. }
14. else {
15. Guid tempCartId = Guid.NewGuid();
16. context.Session[CartSessionKey] = tempCartId.ToString();
17. }
18. }
19. return context.Session[CartSessionKey].ToString();
20. }
```

代码说明：第 1 行实例化实体数据模型 MusicStoreEntities 的上下文对象 storeDB。第 2 行定义

购物车操作类的编号属性。第3行定义一个字符串类型的常量 `CartSessionKey` 赋值为“`CartId`”，表示购物车 Session 的键名称。第4行定义 `GetCart` 方法，参数是一个包含 HTTP 请求信息的对象，返回值是购物车操作类对象。第5行创建一个购物车操作类的对象 `cart`。第6行通过调用 `GetCartId` 方法获得购物车操作类对象 `cart` 的编号。第7行返回该购物车操作类对象。

第9行定义获得购物车编号的方法 `GetCartId`，参数是一个包含 HTTP 请求信息的对象。第10行判断传递的 HTTP 请求信息中 Session 集合中 `CartId` 键的值如果为空，第11行再判断如果 HTTP 请求信息中的用户名不为空，则第12行将用户姓名保存到 Session 集合中作为 `CartId` 键的值。第14行判断如果 HTTP 请求信息中的用户名为空，则第15行通过 `Guid` 类的 `NewGuid` 方法实例化一个 `Guid` 类对象 `tempCartId` 获得一个唯一的临时标识。第16行将该临时标识保存到 Session 集合中作为 `CartId` 键的值。第19行返回 Session 集合中作为 `CartId` 键的值。

(2) 定义将唱片添加到购物车和从购物车中移除唱片的方法，关键代码如下。

```

1. public void AddToCart(Album album){
2. var cartItem = storeDB.Carts.SingleOrDefault(
3. c => c.CartId == shoppingCartId && c.AlbumId == album.AlbumId);
4. if (cartItem == null){
5. cartItem = new Cart{
6. AlbumId = album.AlbumId, CartId = shoppingCartId,
7. Count = 1, DateCreated = DateTime.Now
8. };
9. storeDB.AddToCarts(cartItem);
10. }
11. else{
12. cartItem.Count++;
13. }
14. storeDB.SaveChanges();
15. }
16. public void RemoveFromCart(int id){
17. var cartItem = storeDB.Carts.Single(
18. cart => cart.CartId == shoppingCartId && cart.RecordId == id);
19. if (cartItem != null){
20. if (cartItem.Count > 1){
21. cartItem.Count--;
22. }
23. else{
24. storeDB.Carts.DeleteObject(cartItem);
25. }
26. storeDB.SaveChanges();
27. }
28. }
```

代码说明，第1行定义将唱片添加到购物车 `AddToCart` 方法，参数是唱片类 `Album` 的对象。第2行使用实体数据上下文对象 `storeDB.Carts` 的方法 `SingleOrDefault` 通过唱片对象的购物车编号和唱片编号获得该购物车对象。第4行判断如果该对象不存在，则在第5~8行创建一个新的购物车对象，并为该对象的属性赋值。第9行调用实体数据上下文对象 `storeDB` 的 `AddToCarts` 方法将新创建的购物车类添加到 `Cart` 数据表中。如果第11行判断前面第2行查询的唱片对象已经存在，则第12行将该对象的购物数量加1。第14行调用实体数据上下文对象 `storeDB` 的 `SaveChanges` 方法保存数据库的修改。

第 16 行定义从购物车中移除唱片对象的方法 `RemoveFromCart`，参数是购物车记录编号。第 17 行使用实体数据上下文对象 `storeDB.Carts` 的方法 `Single` 通过唱片对象的购物车编号和购物车记录编号获得该购物车对象。第 19 行判断该对象如果存在，则在第 20 行继续判断，如果该对象的购物数量大于 1，在第 21 行将该对象的购物数量减 1。如果第 23 行判断该购物车对象不存在，第 24 行调用实体数据上下文对象 `storeDB.Carts` 的 `DeleteObject` 方法，将该购物车对象从购物车表中删除。第 26 行调用实体数据上下文对象 `storeDB` 的 `SaveChanges` 方法保存数据库的修改。

(3) 定义清空购物车和获得购物车内唱片信息的方法，关键代码如下。

```

1. public void EmptyCart(){
2. var cartItems = storeDB.Carts.Where(cart => cart.CartId == shoppingCartId);
3. foreach (var cartItem in cartItems){
4. storeDB.DeleteObject(cartItem);
5. }
6. storeDB.SaveChanges();
7. }
8. public List<Cart> GetCartItems() {
9. var cartItems = (from cart in storeDB.Carts
10. where cart.CartId == shoppingCartId
11. select cart).ToList();
12. return cartItems;
13. }
14.

```

代码说明：第 1 行定义了清空购物车的方法 `EmptyCart`。第 2 行使用实体数据上下文对象 `storeDB.Carts` 方法的 `Where` 子句通过购物车操作类编号获得指定购物车对象中唱片的集合。第 3~5 行使用 `foreach` 循环遍历删除该购物车对象集合中所有的对象。第 6 行数据库的更改。第 8 行定义获得购物车内唱片信息的方法。第 9 行通过购物车操作类编号获得指定购物车对象中唱片的集合列表。第 12 行返回该列表。

(4) 定义获得购物车中唱片购买数量和总价的方法，关键代码如下。

```

1. public int GetCount(){
2. int? count = (from cartItems in storeDB.Carts
3. where cartItems.CartId == shoppingCartId
4. select (int?)cartItems.Count).Sum();
5. return count ?? 0;
6. }
7. public decimal GetTotal(){
8. decimal? total =(from cartItems in storeDB.Carts
9. where cartItems.CartId == shoppingCartId
10. select (int?)cartItems.Count * cartItems.Album.Price)
11. .Sum();
12. return total ?? decimal.Zero;
13. }

```

代码说明：第 1 行定义获得购物车中唱片购买数量的方法 `GetCount`。第 2~4 行使用实体数据上下文对象 `storeDB.Carts` 方法的 `Where` 子句，通过购物车操作类编号获得指定购物车对象中唱片购买数量的总数。第 5 行返回该总数。第 7 行定义获得购物车中唱片总价的方法 `GetTotal`。第 8~11 行使用实体数据上下文对象 `storeDB.Carts` 方法的 `Where` 子句，通过购物车操作类编号获得指定购

购物车对象中唱片购买的总价。第 12 行返回该总价。

(5) 定义创建订单和迁移购物车的方法, 关键代码如下。

```

1. public int CreateOrder(Order order){
2. decimal orderTotal = 0;
3. var cartItems = GetCartItems();
4. foreach (var cartItem in cartItems){
5. var orderDetails = new OrderDetail{
6. AlbumId = cartItem.AlbumId,
7. OrderId = order.OrderId,
8. UnitPrice = cartItem.Album.Price
9. };
10. storeDB.OrderDetails.AddObject(orderDetails);
11. orderTotal += (cartItem.Count * cartItem.Album.Price);
12. }
13. storeDB.SaveChanges();
14. EmptyCart();
15. return order.OrderId;
16. }
17. public void MigrateCart(string userName){
18. var shoppingCart = storeDB.Carts
19. .Where(c => c.CartId == shoppingCartId);
20. foreach (Cart item in shoppingCart){
21. item.CartId = userName;
22. }
23. storeDB.SaveChanges();
24. }

```

代码说明: 第 1 行创建订单的方法 `CreateOrder`, 参数是一个订单类 `Order` 对象。第 2 行初始化订单总价变量 `orderTotal` 为 0。第 3 行调用 `GetCartItems` 方法获得购物车内唱片信息并创建一个列表集合对象 `cartItems`。第 4 行使用循环遍历 `cartItems` 对象。第 5 行创建一个订单详情类的对象 `orderDetails`。第 6~8 行给 `orderDetails` 对象的唱片编号、订单编号和唱片单价三个属性赋值。第 10 通过实体数据上下文对象 `storeDB.OrderDetails` 的方法 `AddObject` 将 `orderDetails` 对象添加到 `orderDetail` 数据表中。第 11 行计算获得订单总价。第 13 行保存数据库的修改。第 11 行调用 `EmptyCart` 方法清空当前购物车中的对象。第 15 行返回订单的编号。

第 17 行定义迁移购物车的方法 `MigrateCart`, 参数是用户名。第 18~19 行使用实体数据上下文对象 `storeDB.Carts` 方法的 `Where` 子句通过购物车操作类编号获得指定购物车对象中唱片的集合。第 20~22 行使用 `foreach` 循环遍历将用户名作为购物车对象的购物车编号属性的值。第 23 行保存购物车的修改。

## 2. Order 订单类

在该类中定义了说明订单元数据类型的类 `OrderMetadata`, 并创建了订单的各种属性。`Order` 订单类定义在 `Models` 文件夹下的 `Order.cs` 文件中, 关键代码如下。

```

1. public class OrderMetadata{
2. [Required(ErrorMessage = "姓确认")]
3. [DisplayName("姓")]
4. [StringLength(160)]
5. public object FirstName { get; set; }
6. [Required(ErrorMessage = "名确认")]
7. [DisplayName("名")]

```

```

8. [StringLength(160)]
9. public object LastName { get; set; }
10. [Required(ErrorMessage = "地址必须填写")]
11. [DisplayName("地址")]
12. [StringLength(70)]
13. public object Address { get; set; }
14. [Required(ErrorMessage = "城市必须填写")]
15. [DisplayName("省")]
16. [StringLength(40)]
17. public object City { get; set; }
18. [Required(ErrorMessage = "状态必须填写")]
19. [DisplayName("市")]
20. [StringLength(40)]
21. public object State { get; set; }
22. [Required(ErrorMessage = "邮政编码必须填写")]
23. [DisplayName("邮政编码")]
24. [StringLength(10)]
25. public object PostalCode { get; set; }
26. [Required(ErrorMessage = "国家必须填写")]
27. [DisplayName("国家")]
28. [StringLength(40)]
29. public object Country { get; set; }
30. [Required(ErrorMessage = "手机号码必须填写")]
31. [DisplayName("手机号码")]
32. [StringLength(24)]
33. public object Phone { get; set; }
34. [Required(ErrorMessage = "电子邮件地址是必需的")]
35. [DisplayName("电子邮件")]
36. [RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}", ErrorMessage = "Email is is not valid.")]
37. [DataType(DataType.EmailAddress)]
38. public object Email { get; set; }
39. [ScaffoldColumn(false)]
40. public object OrderId { get; set; }
41. [ScaffoldColumn(false)]
42. public object OrderDate { get; set; }
43. [ScaffoldColumn(false)]
44. public object Username { get; set; }
45. [ScaffoldColumn(false)]
46. public object Total { get; set; }
47. }

```

代码说明：第 1 行定义订单元数据类型类 OrderMetadata。第 2 行属性 Required 指定数据字段 ErrorMessage 的显示文本，用于 OrderMetadata 类的属性 FirstName 验证失败时使用。第 3 行属性 DisplayName 指定 OrderMetadata 类的属性 FirstName 的显示文本。第 4 行 StringLength 属性指定 OrderMetadata 类的属性 FirstName 输入值的字符最大长度。第 5 行定义 OrderMetadata 类的属性 FirstName。第 6~38 行使用相同的方法分别定义了 OrderMetadata 类的属性：LastName、Address、City、State、PostalCode、Country、Phone 和 Email。

第 41 行 ScaffoldColumn 属性指定 OrderMetadata 类的属性 OrderId 不启用基架的值。第 42 行定义 OrderMetadata 类的属性 OrderId。第 43~46 行使用相同的方法分别定义 OrderMetadata 类的属性：OrderDate、Username 和 Total。

### 18.3.3 创建自定义的视图模型

为了在视图页面更方便地获取到控制器传回的数据，我们自定义了 5 个视图模型，也就是自

定义了 5 种数据类型用于页面数据的接收，它们都保存在 ViewModels 文件夹下。

(1) 定义 ShoppingCartRemoveViewModel 移除购物车模型视图的代码。

```
1. public class ShoppingCartRemoveViewModel {
2. public string Message { get; set; }
3. public decimal CartTotal { get; set; }
4. public int CartCount { get; set; }
5. public int DeleteId { get; set; }
6. }
```

代码说明：第 1 行定义移除购物车模型视图类 ShoppingCartRemoveViewModel。第 2~5 行分别定义该类的 4 个属性：Message 显示的文本、CartTotal 购物车总价、CartCount 购物车数量和 DeleteId 移除的编号。

(2) 定义 ShoppingCartViewModel 购物车模型视图的代码。

```
1. public class ShoppingCartViewModel {
2. public List<Cart> CartItems { get; set; }
3. public decimal CartTotal { get; set; }
4. }
```

代码说明：第 1 行定义购物车模型视图类 ShoppingCartViewModel。第 2 行定义该类的购物车 Cart 的 List 泛型集合类属性 CartItems。第 3 行定义该类的购物车总价属性 CartTotal。

(3) 定义 StoreBrowseViewModel 浏览唱片模型视图的代码。

```
1. public class StoreBrowseViewModel {
2. public Genre Genre { get; set; }
3. public List<Album> Albums { get; set; }
4. }
```

代码说明：第 1 行定义浏览唱片模型视图类 StoreBrowseViewModel。第 2 行定义该类的唱片类型属性 Genre。第 3 行定义唱片 Album 的 List 泛型集合类属性 Albums。

(4) 定义 StoreIndexViewModel 唱片索引模型视图的代码。

```
1. public class StoreIndexViewModel {
2. public int NumberOfGenres { get; set; }
3. public List<string> Genres { get; set; }
4. }
```

代码说明：第 1 行定义唱片索引模型视图类 StoreIndexViewModel。第 2 行定义该类的唱片类型数量属性 NumberOfGenres。第 3 行定义泛型集合类列表属性 Genres。

(5) 定义 StoreManagerViewModel 唱片管理模型视图的代码。

```
1. public class StoreManagerViewModel {
2. public Album Album { get; set; }
3. public List<Artist> Artists { get; set; }
4. public List<Genre> Genres { get; set; }
5. }
```

代码说明：第 1 行定义唱片管理模型视图类 StoreManagerViewModel。第 2 行定义该类的唱片

类属性 Album。第 3 行定义演唱者 Artist 的 List 泛型集合类属性 Artists。第 4 行定义唱片类型 Genre 的 List 泛型集合类属性 Genres。

## 18.4 用户登录模块

用户登录模块包括系统的首页、用户登录和和用户注册这三个功能组成，每个功能由控制器和视图共同来实现。

### 18.4.1 使用母版页

为了满足系统页面设计的需要，我们使用了母版页机制，在 Shared 文件夹下的 Site.Master 母版页文件中设计了整个网站的结构布局。

(1) 母版页整个界面由三个 div 组成，关键的 HTML 代码如下。

```

1. <body>
2. <div id="container">
3. <div id="header">
4. <h1></h1>
5. <ul id="navlist">
6. <li class="first">主页
7. 商店
8. <% Html.RenderAction("CartSummary", "ShoppingCart"); %>
9.
10. 管理
11.
12. </div>
13. <% Html.RenderAction("GenreMenu", "Store"); %>
14. <div id="main">
15. <asp:ContentPlaceHolder ID="MainContent" runat="server" />
16. </div>
17. <div id="footer">
18. Music Store
19. </div>
20. </div>
21. </body>

```

代码说明：第 3~12 行是第 1 个 div。其中第 4 行使用 HTML 的 h1 标题标记。第 5~11 行使用 HTML 的无序列表。第 6 行使用“li”标记显示列表的内容为主页的链接。第 7 行使用“li”标记显示列表的内容为商店链接。第 8 行使用“li”标记显示列表的内容为一个用户控件 CartSummary.ascx 通过 Html.RenderAction 方法呈现用户控件内容。第 9 行使用“li”标记显示列表的内容为管理的链接。

第 13 行使用一个用户控件 GenreMenu.ascx 通过 Html.RenderAction 方法呈现用户控件内容。第 14~16 行是第 2 个 div。第 15 行在该 div 中定义了一个占位符控件 MainContent 用于内容页面的填充。

第 17~19 行是第 3 个 div。第 18 行使用“b”标记粗体显示文本。

(2) 用户控件 CartSummary 在 ShoppingCart 文件夹下的 CartSummary.ascx 文件中定义，代码如下。

1. `<%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl<dynamic>" %>`
2. `<%: Html.ActionLink("购物车 (" + ViewData["CartCount"] + ") ", "Index", "ShoppingCart", new { id = "cart-status" })%>`

代码说明：第 1 行使用 `@Control` 指令设置用户控件继承于 `System.Web.Mvc.ViewUserControl` 类，数据类型为 `dynamic`。第 2 行使用 `Html.ActionLink` 方法定义页面的超链接，参数是显示的文本、动作方法、控制器名称和传递的参数。

(3) 用户控件 GenreMenu 在 Store 文件夹下的 GenreMenu.ascx 文件中定义, 代码如下:

```

1. <%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl<IEnumerable<MvcMusicStore.Models.Genre>>" %>
2. <ul id="categories">
3. <% foreach (var genre in Model) { %>
4. <li style="padding-left :47px">
5. &~
null)%>
6.
7. <% } %>
8.

```

代码说明：第 1 行使用 `@Control` 指令设置用户控件继承于 `System.Web.Mvc.ViewUserControl` 类，数据类型为 `IEnumerable<MvcMusicStore.Models.Genre>`。第 4~10 行使用一个无序列表。第 5 行通过循环遍历从控制器传递的 `Model` 对象的每个元素。第 6~8 行在列表内容标签中显示产品类型的超链接以及显示的文本、动作方法 `Browse`、控制器 `Store` 和传递的参数。

(4) 母版页的控制器动作方法 `CartSummary` 定义在 `Controllers` 文件夹下的 `ShoppingCartController.cs` 文件夹中, 关键代码如下。

```
public ActionResult CartSummary() {
 var cart = ShoppingCart.GetCart(this.HttpContext);
 ViewData["CartCount"] = cart.GetCount();
 return PartialView("CartSummary");
}
```

代码说明：第 1 行定义 ActionResult 返回类型的动作方法 CartSummary。第 2 行通过购物车操作类的 GetCart 获得购物车对象 cart。第 3 行通过 cart 对象的 getCount 属性获得购物车对象中唱片的数量，并保存到 ViewData 中。第 4 行将 ViewData 对象返回到用户控件 GenreMenu.ascx 中。

(5) 母版页中的另一个控制器动作方法 `GenreMenu` 定义在 `Controllers` 文件夹下的 `StoreController.cs` 文件夹中, 关键代码如下。

```
public ActionResult GenreMenu() {
 var genres = storeDB.Genres.ToList();
 return View(genres);
}
```

代码说明：第 1 行定义 ActionResult 返回类型的动作方法 GenreMenu。第 2 行数据实体模型 MusicStoreEntities 的上下文对象 storeDB 中产品类型对象 Genres 的 ToList 方法获得所有唱片类型的列表。第 3 行将该列表返回到用户控件 GenreMenu.ascx 中。

(6) 母版页中 HTML 元素的外观样式呈现由 Content 文件夹下的 Site.css 样式表文件控制,

关键的 CSS 代码如下。

```

1. body {
2. font-family: Arial, Helvetica, sans-serif;
3. font-size: 14px;padding: 0px 6%;
4. font-family: @Adobe 黑体 Std R; background-position:center top ;
5. background-color:#ebebeb;
6. }
7. #container{
8. float: left;
9. }
10. #header{
11. float: left;width: 100%;border-bottom: 1px dotted #5D5A53;
12. margin-bottom: 10px;
13. }
14. #header h1{
15. font-size: 18px;float: left;width :400px;
16. background: url(/content/Images/ssss.gif) no-repeat;
17. padding: 44px 0px 0px 0px;
18. }
19. #footer{
20. clear: both;padding: 10px;text-align: right;color:#a20202;
21. border-top: 1px dotted #8A8575;border-bottom: 1px dotted #8A8575;
22. font-family: Constantia, Georgia, serif;
23. }
24. ul#navlist{
25. float: right;
26. }
27. ul#navlist li{
28. display: inline;
29. }
30. ul#navlist li a{
31. border-left: 1px dotted #8A8575; padding: 10px;
32. margin-top: 10px; color: #312323;
33. text-decoration: none; float: left;
34. }
35. ul#navlist li:first-child a{
36. border: none;
37. }
38. ul#navlist li a:hover{
39. color: #a20202;
40. }
```

代码说明：第 1~6 行定义 body 元素的各文本属性和背景颜色。第 7~9 行定义 id 选择器为 container 的 div 的样式。第 10~13 行定义 id 选择器为 header 的 div 的位置、宽度、边框和边距的样式。第 14~18 行定义 div 中 h1 标记的文本大小、宽度、位置、背景图片和内边距的样式。第 19~23 行定义 id 选择器为 footer 的 div 的内边距、边框和文本的位置、颜色和字体的样式。第 24~26 行定义 id 选择器为 navlist 的无序列表位置样式。第 27~29 行定义无序列表中 li 标记的显示样式。第 30~34 行定义 li 标记中 a 标记的文本、内外边距、颜色和位置的样式。第 35~37 行定义 li 标记中第一个 a 标记的边框样式。第 38~40 行定义 li 标记中 a 标记被单击时显示的颜色。

母版页 Site.Master 最后设计完毕的效果如图 18-20 所示。

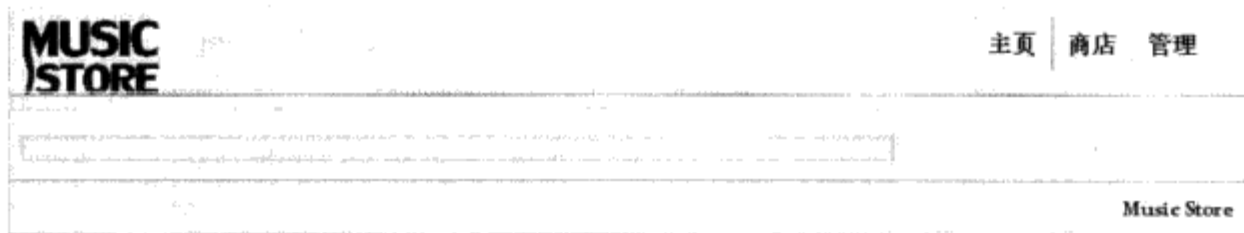


图 18-20 母版页界面

### 18.4.2 首页

首页作为内容页面被包含在母版页 Site.master 的占位符控件中显示一张图片和最新热点的唱片。

(1) 首页的控制器动作方法 Index 定义在 Controllers 文件夹下的 HomeController.cs 文件夹中，具体代码如下。

```

1. public ActionResult Index(){
2. MusicStoreEntities storeDB = new MusicStoreEntities();
3. var albums = GetTopSellingAlbums(5);
4. return View(albums);
5. }
6. private List<Album> GetTopSellingAlbums(int count){
7. return storeDB.Albums
8. OrderByDescending(a => a.OrderDetails.Count())
9. .Take(count)
10. .ToList();
11. }

```

代码说明：第 1 行定义 ActionResult 返回类型的动作方法 Index。第 2 行实例化数据实体模型 MusicStoreEntities 的上下文对象 storeDB。第 3 行调用 GetTopSellingAlbums 方法获得 albums 唱片对象集合。第 4 行将 albums 返回到首页视图。

第 6 行定义唱片获得热门唱片列表的方法 GetTopSellingAlbums，参数是显示唱片的张数。第 7~10 行使用 storeDB.Albums 的 OrderByDescending 方法获得排序后热门唱片的列表对象。

(2) 设计对应与首页动作方法 Index 的视图，该视图定义在 Views 文件夹下 Home 子文件夹下的 Indexd.aspx 文件中。关键的 HTML 代码如下。

```

1. <%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master" Inherits="System.Web.Mvc.ViewPage
2. <IEnumerable<MvcMusicStore.Models.Album>>" %>
3. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
4. <div id="promotion"></div>
5. <h3>最新热点</h3>
6. <ul id="album-list">
7. <% foreach (var album in Model){ %>
8.
9. <a href="<%= Url.Action("Details", "Store", new { id = album.AlbumId }) %>">
10. <img alt="<%= album.Title %>" src="<%= album.AlbumArtUrl %>" />

11. <%= album.Title %>
12.
13.
14. <% } %>
15.
16. </asp:Content>

```

代码说明: 第 1 行使用 @Page 指令设置视图页面的母版页文件属性 MasterPageFile 和页面类继承的属性。第 2~15 行定义 Content 控件关联母版页中占位符控件的 ID 属性 MainContent。其中定义了视图页面的界面元素, 第 3 行定义 div。第 4 行定义标题标签 h3 显示文本。第 5~14 行定义无序列表。第 6~13 行使用 foreach 循环显示列表的内容。第 7~12 行是显示的具体内容, 第 8 行通过 Url.Action 方法设置路由, 参数为动作方法、控制器和传递的参数。第 9 行设置显示图片的路径。第 10 行设置图片的标题。

(3) 首页视图中 HTML 元素的外观样式同样呈现由 Content 文件夹下的 Site.css 样式表文件控制, 关键的 css 代码如下。

```

1. #promotion{
2. height: 300px; width: 700px;
3. background: url(/content/Images/EN.jpg) no-repeat;
4. }
5. ul#album-list{
6. list-style: none; margin-left: 0px;
7. }
8. ul#album-list li{
9. height: 130px; width: 100px; float: left; margin: 10px;
10. text-align: center;
11. }
12. ul#album-list li a, ul#album-list li .button{
13. font-size: 13px; float: left;
14. }
15. ul#album-list li a span{
16. font-family: @Adobe 黑体 Std R ;text-decoration: underline;
17. color :black ;
18. }
```

代码说明: 第 1~4 行定义 id 选择器名为 promotion 的 div 的大小和背景图片。第 5~7 行定义 id 选择器名为 album 的无序列表的样式和左边距。第 8~11 行定义无序列表中 li 标记的大小、位置、边距和文字的样式。第 12~14 行定义列 li 标记中的 a 标记的文本的大小和位置。第 15~18 行定义 li 标记中 span 标记的中字体、文字下划线和颜色的样式。

设计后首页的界面如图 18-21 所示。

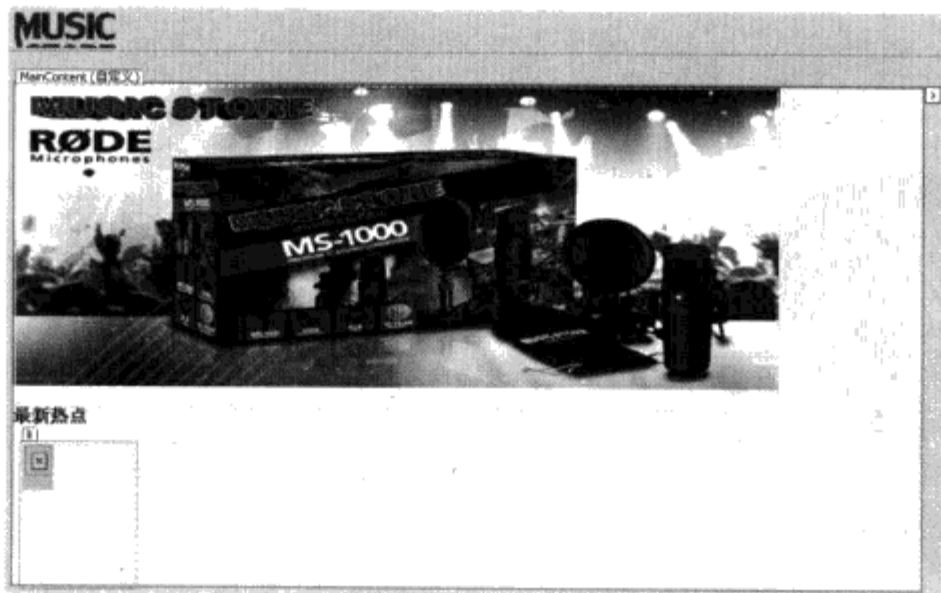


图 18-21 设计后的首页

### 18.4.3 登录页面

登录页面的控制器动作方法 LogOn 定义在 Controllers 文件夹下的 AccountController.cs 文件夹中, 关键代码如下。

```

1. [HttpPost]
2. public ActionResult LogOn(LogOnModel model, string returnUrl){
3. if (ModelState.IsValid){
4. if (MembershipService.ValidateUser(model.UserName, model.Password)){
5. MigrateShoppingCart(model.UserName);
6. FormsService.SignIn(model.UserName, model.RememberMe);
7. if (!String.IsNullOrEmpty(returnUrl)){
8. return Redirect(returnUrl);
9. }
10. else{
11. return RedirectToAction("Index", "Home");
12. }
13. }
14. else{
15. ModelState.AddModelError("", "用户名或密码提供的不正确.");
16. }
17. }
18. return View(model);
19. }
```

代码说明: 第 1 行表示下面的方法只接受用户通过 Post 方法发送表单数据。第 2 行定义 ActionResult 返回类型的动作方法 LogOn, 参数是登录对象和返回的路径。第 3 行判断如果请求对象模型字典的数据通过验证则第 4 行继续判断用户身份在数据库中是否存在。第 5 行调用迁移购物车的方法 MigrateShoppingCart 将用户的名字作为购物车类编号属性的值。第 6 行调用 FormsService 类的 SignIn 方法将用户的信息保存到 Cookie 中。第 7 行判断如果返回路径参数字符串不为空, 第 8 行重新定向到指定的 URL。否则第 11 行重新定向到 HomeController 控制器的 Index 动作方法。如果第 14 行判断用户在数据库中并不存在, 则第 15 行调用 ModelState 类的 AddModelError 方法显示“用户名或密码不正确”的错误提示。如果请求对象模型字典的数据没有通过验证则在第 18 行将登录对象传回登录视图页面。

设计对应与登录动作方法 LogOn 的视图, 该视图定义在 Views 文件夹下 Account 子文件夹下的 LogOn.aspx 文件中, 关键的 HTML 代码如下。

```

1. <asp:Content ID="loginContent" ContentPlaceHolderID="MainContent" runat="server">
2. <h2>登录</h2>
3. <p>请输入您的用户名和密码。 <%= Html.ActionLink("注册", "Register") %> 如果你没有一个帐户.</p>
4. <%= Html.ValidationSummary(true, "登录失败。请更正错误, 然后再试一次.") %>
5. <% using (Html.BeginForm()) { %>
6. <div>
7. <fieldset>
8. <legend>账户信息</legend>
9. <div class="editor-label">
10. <%= Html.LabelFor(m => m.UserName) %>
11. </div>
12. <div class="editor-field">
13. <%= Html.TextBoxFor(m => m.UserName) %>
14. <%= Html.ValidationMessageFor(m => m.UserName) %>
15. </div>

```

```

16. <div class="editor-label">
17. <%= Html.LabelFor(m => m.Password) %>
18. </div>
19. <div class="editor-field">
20. <%= Html.PasswordFor(m => m.Password) %>
21. <%= Html.ValidationMessageFor(m => m.Password) %>
22. </div>
23. <div class="editor-label">
24. <%= Html.CheckBoxFor(m => m.RememberMe) %>
25. <%= Html.LabelFor(m => m.RememberMe) %>
26. </div>
27. <p> <input type="submit" value="登录" /> </p>
28. </fieldset>
29. </div>
30. <% } %>
31. </asp:Content>

```

代码说明：第 1~31 行定义 Content 控件关联母版页中占位符控件的 ID 属性 MainContent。第 2 行创建标题。第 3 行定义段落标记显示登录提示，使用 Html 的 ActionLink 方法显示注册链接。第 4 行使用 Html 的 ValidationSummary 方法，当出现登录失败时显示提示信息。第 5~30 行定义一个表单，其中，第 7~28 行定义一个边框。第 8 行创建边框的名称。第 9~28 行定义了 5 个 div，分别显示标签、文本框、密码框和多选框，并对文本框和密码框设置验证。第 27 行在一个段落中定义一个登录按钮。

登录视图中 HTML 元素的外观样式同样呈现由 Content 文件夹下的 Site.css 样式表文件控制，关键的 CSS 代码如下。

```

1. legend{
2. padding: 10px;font-weight: bold; color :#a20202;
3. }
4. fieldset{
5. border: #9b9993 1px solid;padding: 0 10px; margin-bottom: 10px;
6. clear: left;
7. }
8. div.editor-field{
9. margin-bottom: 10px;
10. }

```

代码说明：第 1~3 行定义边框标题的内边距和颜色的演示。第 4~7 行定义边框标签的样式，外边距。第 8~10 行定义 div 中类选择器名为 editor-field 元素的边距样式。

设计后登录视图的界面如图 18-22 所示。

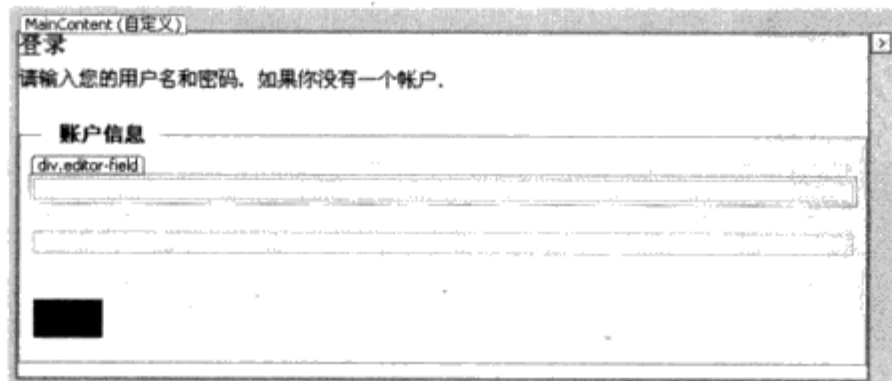


图 18-22 设计后登录界面

本模块中完成注册功能的控制器和视图与用户登录功能类似，这里就不再详细说明了，读者可参考光盘中的源代码进行学习。

## 18.5 购物车模块

购物车模块是本系统中逻辑最为复杂的模块。它由根据类型浏览唱片功能、唱片类型浏览功能、唱片详情浏览功能、购物车功能、填写订单功能和完成结账功能组成。每个功能由控制器和视图共同来实现。其中，使用了 ASP.NET AJAX 的控件。

### 18.5.1 根据类型浏览唱片页面

根据类型浏览唱片页面的控制器动作方法 Browse 定义在 Controllers 文件夹下的 StoreController.cs 文件夹中，关键代码如下。

```
1. public ActionResult Browse(string genre){
2. var genreModel = storeDB.Genres.Include("Albums")
3. .Single(g => g.Name == genre);
4. var viewModel = new StoreBrowseViewModel(){
5. Genre = genreModel,
6. Albums = genreModel.Albums.ToList()
7. };
8. return View(viewModel);
9. }
```

以上代码中，第 1 行定义返回值为 ActionResult 类型的动作方法 Browse，参数是唱片类型的名称。通过数据实体模型 MusicStoreEntities 的上下文对象 storeDB 中的唱片类型对象 Genres 的 Include 方法将获得指定唱片类型的包含唱片对象 Albums 信息的产品类型。第 4~7 行实例化唱片浏览视图模型类的对象 viewModel，第 5 行将获得的产品类型列表 genreModel 为该对象的 Genre 属性的值，第 6 行通过 genreModel 中唱片对象 Albums 的 ToList 方法获得唱片对象的列表并作为 viewModel 对象属性 Albums 的值。第 8 行将 viewModel 对象返回到 Browse 视图。

设计对应与动作方法 Browse 的视图，该视图定义在 Views 文件夹下 Store 子文件夹下的 Browse.aspx 文件中，关键的 HTML 代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <div class="genre">
3. <h3><%: Model.Genre.Name %> 专辑</h3>
4. <ul id="album-list">
5. <% foreach (var album in Model.Albums) { %>
6.
7. <a href="<%: Url.Action("Details", new { id = album.AlbumId }) %>">
8. <img alt="<%: album.Title %>" src="<%: album.AlbumArtUrl %>" style="border:2px solid black; width:87px;
height:78px" />

9. <%: album.Title %>
10.
11. <% } %>
12.
13. </div>
14. </asp:Content>
```

代码说明：第 1~14 行定义 Content 控件关联母版页中占位符控件的 ID 属性 MainContent。其中，第 3 行定义页面标题。第 4~12 行定义一个无序列表。第 5 行通过循环遍历从控制器传递的 Model 对象中的元素。第 6~10 行使用列表内容标签显示唱片详情的链接和唱片的图片。第 9 行在段落标记中显示唱片的标题。

设计后根据类型浏览唱片视图的界面如图 18-23 所示。

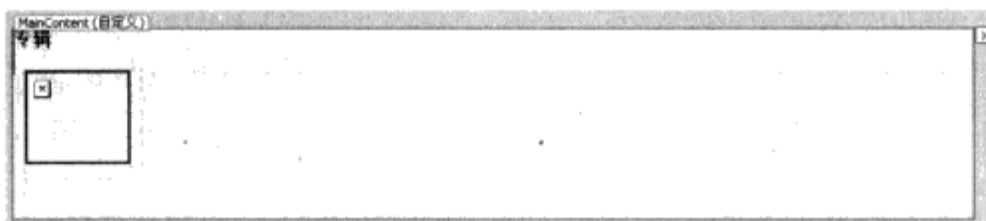


图 18-23 设计后根据类型浏览唱片界面

## 18.5.2 唱片详情浏览页面

唱片详情浏览页面的控制器动作方法 Details.aspx 定义在 Controllers 文件夹下的 StoreController.cs 文件夹中，关键代码如下。

```
1. public ActionResult Details(int id) {
2. var album = storeDB.Albums.Single(a => a.AlbumId == id);
3. return View(album);
4. }
```

代码说明：第 1 行定义返回值为 ActionResult 类型的动作方法 Details，参数是唱片的编号。第 2 行通过数据实体模型 MusicStoreEntities 的上下文对象 storeDB 中唱片对象 Albums 的 Single 查询获得指定唱片编号的唱片对象。第 3 行返回该对象到视图。

设计对应与动作方法 Details 的视图，该视图定义在 Views 文件夹下 Store 子文件夹下的 Details.aspx 文件中，关键的 HTML 代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <h2><%= Model.Title %></h2>
3. <p>
4. <img alt="<%= Model.Title %>" src="<%= Model.AlbumArtUrl %>" style="border:2px solid white; width:87px;
height:78px"/>
5. </p>
6. <div id="album-details">
7. <p>类 型: <%= Model.Genre.Name %></p>
8. <p>艺术家: <%= Model.Artist.Name %></p>
9. <p>价 格:
10. ¥ <%= String.Format("{0:F}", Model.Price) %>
11. </p>
12.

13. <p class="button">
14. <%= Html.ActionLink("添加到购物车", "AddToCart", "ShoppingCart", new { id = Model.AlbumId }, "")%>
15. </p>
16. </div>
17. </asp:Content>
```

代码说明：第 1~17 行定义 Content 控件关联母版页中占位符控件的 ID 属性 MainContent。其中，第 2 行定义页面标题用于显示产品的标题。第 3~5 行在段落标记中显示唱片的图片。第 6~16

行在一个 div 中, 第 7~11 行使用 3 个段落标记来显示产品的类型、艺术家、价格。第 14~15 行在段落标记中显示一个“添加到购物车”的超链接并指定了显示的文本、动作方法、控制器和传递的参数。

唱片详情浏览视图中 HTML 元素的外观样式呈现由 Content 文件夹下的 Site.css 样式表文件控制, 关键的 CSS 代码如下。

```

1. div#album-details p{
2. margin-bottom: 5px; color: #5e5b54;
3. font-family:@Adobe 黑体 Std R; width: 321px;
4. }
5. .button, input[type=submit]{
6. clear: both; display: inline-block; padding: 5px;
7. margin-top: 10px; border: 1px; background: red;
8. color: #a20202; font-weight: bold;
9. }
```

代码说明: 第 1~4 行定义 div 中 id 选择器名为 album-details 标签中的段落的边距、颜色、字体和宽度的样式。第 5~9 行定义 id 选择器名为 button 标记的显示方式、内外边距、边框、颜色和字重的样式。

设计后唱片详情浏览视图的界面如图 18-24 所示。

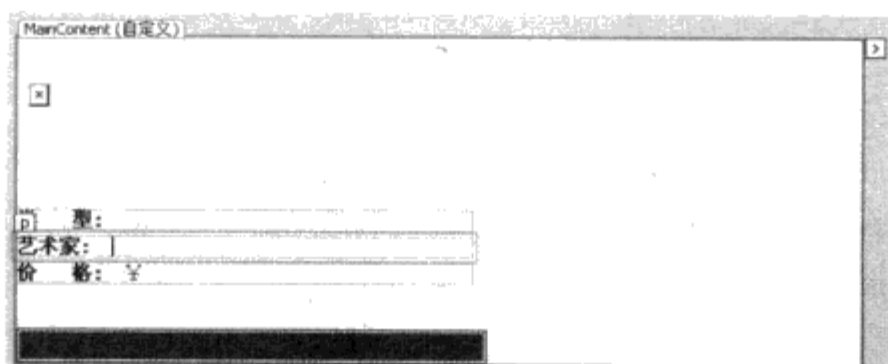


图 18-24 设计后唱片详情浏览界面

### 18.5.3 购物车页面

购物车页面的控制器动作方法 Index、AddToCart 和 RemoveFromCart 定义在 Controllers 文件夹下的 ShoppingCartController.cs 文件夹中, 关键代码如下。

```

1. public ActionResult Index(){
2. var cart = ShoppingCart.GetCart(this.HttpContext);
3. var viewModel = new ShoppingCartViewModel{
4. CartItems = cart.GetCartItems(),
5. CartTotal = cart.GetTotal()
6. };
7. return View(viewModel);
8. }
9. public ActionResult AddToCart(int id){
10. var addedAlbum = storeDB.Albums
11. .Single(album => album.AlbumId == id);
12. var cart = ShoppingCart.GetCart(this.HttpContext);
13. cart.AddToCart(addedAlbum);
14. return RedirectToAction("Index");
15. }
16. public ActionResult RemoveFromCart(int id){
```

```

17. var cart = ShoppingCart.GetCart(this.HttpContext);
18. string albumName = storeDB.Carts
19. .Single(item => item.RecordId == id).Album.Title;
20. cart.RemoveFromCart(id);
21. var results = new ShoppingCartRemoveViewModel {
22. Message = Server.HtmlEncode(albumName) + " 已从您的购物车移除.",
23. CartTotal = cart.GetTotal(),
24. CartCount = cart.GetCount(),
25. DeleteId = id
26. };
27. return Json(results);
28. }

```

代码说明：第 1 行定义返回值为 `ActionResult` 类型的动作方法 `Index`。第 2 行通过购物车操作类的 `GetCart` 获得购物车对象 `cart`。第 3~6 行实例化购物车视图模型类的对象 `viewModel`，第 4 行通过 `cart` 对象的 `GetCartItems` 方法获得购物车中所有对象并赋给 `viewModel` 对象属性 `CartItems`。第 5 行通过 `cart` 对象的 `GetTotal` 方法获得购物车的总价并赋给 `viewModel` 对象属性 `CartTotal`。第 7 行返回 `viewModel` 购物车视图。

第 9 行定义返回值为 `ActionResult` 类型的动作方法 `AddToCart`，参数是唱片的编号。第 10~11 行获取指定唱片编号的唱片对象 `addedAlbum`。第 12 行通过购物车操作类的 `GetCart` 获得购物车对象 `cart`。第 13 行调用 `cart` 对象的 `AddToCart` 方法将 `addedAlbum` 唱片添加到购物车。第 14 行重新定向到购物车视图。

第 15 行定义动作方法 `RemoveFromCart`，参数是购物车的记录编号。第 17 行通过购物车操作类的 `GetCart` 获得购物车对象 `cart`。第 18~19 行获取要移除的购物车内唱片的标题。第 20 行调用 `cart` 对象的 `RemoveFromCart` 从购物车中移除指定的唱片对象。第 21~26 行实例化一个移除购物车模型视图类的对象 `results`。第 22 行设置其显示文本属性的值。第 23 行调用 `cart` 对象的 `GetTotal` 方法设置其购物车总价属性的值。第 24 行调用 `cart` 对象的 `GetCount` 方法设置其购物车唱片数量属性的值。第 25 行设置其移除编号属性的值。第 27 行将 `viewModel` 对象以 `Json` 的数据类型返回到视图购物车视图。

设计对应与动作方法 `Index.aspx` 的视图，该视图定义在 `Views` 文件夹下 `ShoppingCart` 子文件夹下的 `Index.aspx` 文件中。在该视图中使用 `ASP.NET AJAX` 和 `JQuery` 技术实现异步调用移除购物车中唱片的方法来执行页面的局部刷新，关键的 `HTML` 代码如下。

```

1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <script src="/Scripts/MicrosoftAjax.js" type="text/javascript"></script>
3. <script src="/Scripts/MicrosoftMvcAjax.js" type="text/javascript"></script>
4. <script src="/Scripts/jquery-1.4.1.min.js" type="text/javascript"></script>
5. <script type="text/javascript">
6. function handleUpdate(context) {
7. var json = context.get_data();
8. var data = Sys.Serialization.JavaScriptSerializer.deserialize(json);
9. $('#row-' + data.DeleteId).fadeOut('slow');
10. $('#cart-status').text('Cart (' + data.CartCount + ')');
11. $('#update-message').text(data.Message);
12. $('#cart-total').text(data.CartTotal);
13. }
14. </script>
15. <h3>回顾 你的购物车: </h3>
16. <p class="button">

```

```

17. <%: Html.ActionLink("结账 >>", "AddressAndPayment", "Checkout")%>
18. </p>
19. <div id="update-message"></div>
20. <table>
21. <tr>
22. <th>专辑名称</th><th>价格(每个)</th><th>数量</th><th></th>
23. </tr>
24. <% foreach (var item in Model.CartItems) { %>
25. <tr id="row-%: item.RecordId %>">
26. <td>
27. <%: Html.ActionLink(item.Album.Title, "Details", "Store",
28. new { id = item.AlbumId }, null)%>
29. </td>
30. <td><%: item.Album.Price %></td>
31. <td><%: item.Count %></td>
32. <td>
33. <%: Ajax.ActionLink("移除商品", "RemoveFromCart",
34. new { id = item.RecordId },
35. new AjaxOptions { OnSuccess = "handleUpdate" })%>
36. </td>
37. </tr>
38. <% } %>
39. <tr>
40. <td>合计</td><td></td><td></td>
41. <td id="cart-total">¥ <%: Model.CartTotal %></td>
42. </tr>
43. </table>
44. </asp:Content>

```

代码说明:第2~4行添加对ASP.NET AJAX和JQuery脚本库的引用。第5~14行编写JavaScript脚本代码。第6行定义函数handleUpdate处理移除唱片后的数据显示,参数是页面的请求对象context。第7行调用context对象的get\_data获取控制器传递的Json数据。第8行将JSON字符串转换为ECMAScript(JavaScript)对象data。第9~12行使用JQuery对象\$获得移除唱片编号、唱片数量、提示的文本和购物车总价。

第16~18行使用段落标签显示结账的超链接,设置其显示文本、动作方法和控制器。第20~43行定义一个表格。第22行定义表格的标题。第24~38行使用循环动态生成表格要显示的数据内容,每行分为4列。第26~29行为第1列,显示一个唱片对象的超链接。第30行是第2列显示唱片的价格。第31行是第3列显示唱片的数量。第32~36行是第4列,使用Ajax.ActionLink的方法定义了一个实现Ajax的超链接,前三个参数是显示的文本、动作方法、传递的参数。第35行是该方法的第4个参数使用了创建new AjaxOptions对象用于Ajax脚本的选项设置,这里的OnSuccess表示成功更新页面之后,要调用的JavaScript函数handleUpdate。第41行显示购物车的总价。

设计后购物车视图的界面如图18-25所示。

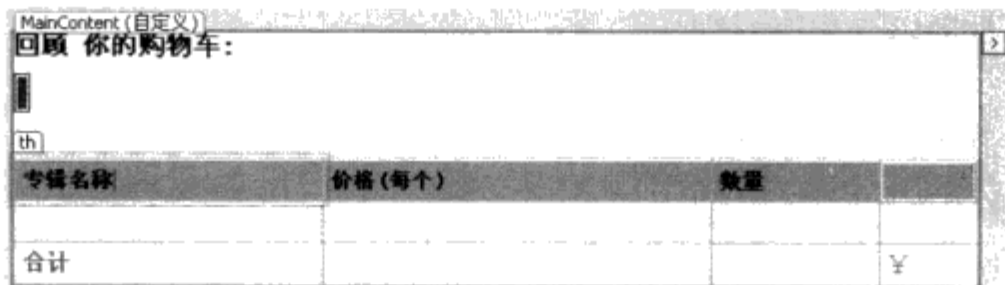


图 18-25 设计后购物车视图界面

### 18.5.4 填写订单页面

填写订单页面控制器动作方法 `AddressAndPayment` 定义在 `Controllers` 文件夹下的 `CheckoutController.cs` 文件夹中, 关键代码如下。

```

1. [HttpPost]
2. public ActionResult AddressAndPayment(FormCollection values){
3. var order = new Order();
4. TryUpdateModel(order);
5. try{
6. order.Username = User.Identity.Name;
7. order.OrderDate = DateTime.Now;
8. storeDB.AddToOrders(order);
9. storeDB.SaveChanges();
10. var cart = ShoppingCart.GetCart(this.HttpContext);
11. cart.CreateOrder(order);
12. return RedirectToAction("Complete",
13. new { id = order.OrderId});
14. }
15. catch{
16. return View(order);
17. }
18. }
```

代码说明: 第 1 行属性表示下面的方法只接受用户通过 `Post` 方法发送表单数据。第 2 行定义动作方法 `AddressAndPayment`, 参数是表单集合对象。第 3 行实例化一个订单对象 `order`。第 4 行调用 `TryUpdateModel` 方法更新订单对象。第 6 行获取用户名作为订单中用户名属性的值。第 7 行将当前系统时间作为订单中订单时间属性的值。第 8 行调用实体数据模型上下文对象 `storeDB` 的 `AddToOrders` 将订单对象 `order` 添加到订单表中。第 9 行保存数据库的更新。第 10 行通过购物车操作类 `ShoppingCart` 的 `GetCart` 方法得到购物车对象 `cart`。第 11 行调用 `CreateOrder` 方法根据 `order` 对象创建订单详情。第 12 行重新定向到订单完成页面。如果以上操作失败, 在第 16 行返回填写订单页面。

设计对应与动作方法 `AddressAndPayment` 的视图, 该视图定义在 `Views` 文件夹下 `Checkout` 子文件夹下的 `AddressAndPayment.aspx` 文件中, 关键的 HTML 代码如下。

```

1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <script src="/Scripts/MicrosoftAjax.js" type="text/javascript"></script>
3. <script src="/Scripts/MicrosoftMvcAjax.js" type="text/javascript"></script>
4. <script src="/Scripts/MicrosoftMvcValidation.js" type="text/javascript"></script>
5. <script src="/Scripts/jquery-1.4.1.min.js" type="text/javascript"></script>
6. <% Html.EnableClientValidation(); %>
7. <% using (Html.BeginForm()) {%>
8. <fieldset>
9. <legend>送货地址信息</legend>
10. <%: Html.EditorForModel() %>
11. </fieldset>
12. <input type="submit" value="提交订单" />
13. <% } %>
14. </asp:Content>
```

代码说明: 第 2~5 行导入 ASP.NET AJAX 和 JQuery 脚本库用来实现客户端验证功能。第 6 行指定使用客户端验证用户输入的数据。第 7 行定义一个表单。第 8~11 行定义一个边框。第 9 行定

义边框的标题。第 10 行使用 Html 的 EditorFor 方法以自定义的 Order 订单类的数据类型为模板, 在该模板中定义了显示填写订单的 HTML 代码。第 12 行定义“提交订单”的按钮。

设计后填写订单视图的界面如图 18-26 所示。

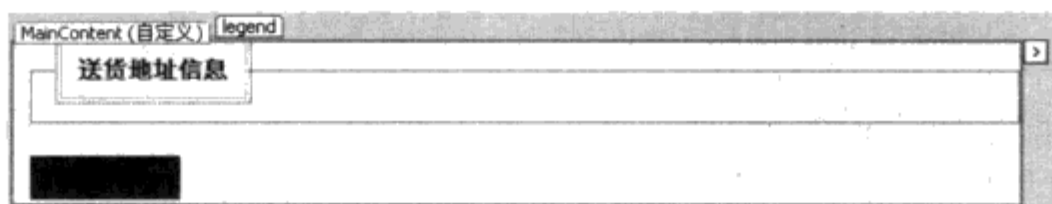


图 18-26 设计后填写订单界面

本模块中完成结账功能的控制器和视图上面的功能类似, 这里就不再详细说明了, 读者可参考光盘中的源代码进行学习。

## 18.6 后台管理模块

后台管理模块由管理唱片、编辑唱片、创建唱片、删除唱片和删除成功这 5 个功能组成。每个功能由控制器和视图共同来实现。

### 18.6.1 管理唱片页面

管理唱片页面控制器动作方法 Index 定义在 Controllers 文件夹下的 StoreManagerController.cs 文件夹中, 关键代码如下。

```
1. public ActionResult Index(){
2. var albums = storeDB.Albums.Include("Genre").Include("Artist")
3. .ToList();
4. return View(albums);
5. }
```

以上代码中, 第 1 行定义动作方法 Index。第 2 行获取包含演唱者对象数据的唱片类型的列表集合。第 4 行将该对象返回到视图。

设计对应与动作方法 Index 的视图, 该视图定义在 Views 文件夹下 StoreManager 子文件夹下的 Index.aspx 文件中, 关键的 HTML 代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <table>
3. <tr> <th></th><th>标题</th><th>艺术家</th><th>类型</th></tr>
4. <% foreach (var item in Model) { %>
5. <tr>
6. <td>
7. <%: Html.ActionLink("编辑", "Edit", new { id=item.AlbumId }) %> |
8. <%: Html.ActionLink("删除", "Delete", new { id=item.AlbumId })%>
9. </td>
10. <td><%: Html.Truncate(item.Title, 125) %></td>
11. <td><%: Html.Truncate(item.Artist.Name, 125) %></td>
12. <td><%: item.Genre.Name %></td>
13. </tr>
14. <% } %>
15. </table>

```

```

16. <p><%: Html.ActionLink("添加新唱片", "Create") %></p>
17. </asp:Content>

```

代码说明：第 2~15 行用一个表格来显示唱片的数据。第 3 行设置表格的标题。第 4 行通过常用的 `foreach` 循环语句，在从控制器传递到视图的 `Model` 数据中，分别获取唱片的标题、演唱者和唱片类型。其中第 7~8 行使用了 HTML 中的扩展方法 `ActionLink`，设置“编辑”和“删除”链接，用于编辑指定记录或删除该记录。第 16 行设置“添加新唱片”链接，用于添加新的唱片数据。

设计后唱片管理视图的界面如图 18-27 所示。

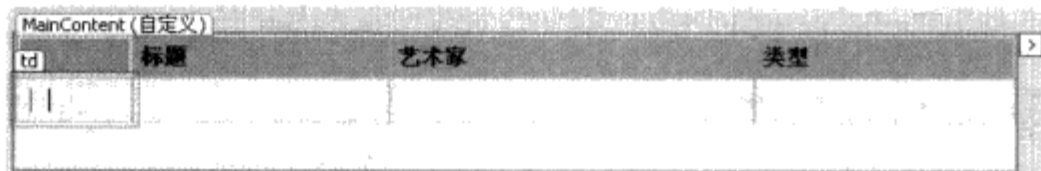


图 18-27 设计后唱片管理界面

### 18.6.2 编辑唱片页面

编辑唱片页面控制器动作方法 `Edit` 定义在 `Controllers` 文件夹下的 `StoreManagerController.cs` 文件夹中，关键代码如下。

```

1. public ActionResult Edit(int id){
2. var viewModel = new StoreManagerViewModel{
3. Album = storeDB.Albums.Single(a => a.AlbumId == id),
4. Genres = storeDB.Genres.ToList(),
5. Artists = storeDB.Artists.ToList()
6. };
7. return View(viewModel);
8. }
9. public ActionResult Edit(int id, FormCollection formValues){
10. var album = storeDB.Albums.Single(a => a.AlbumId == id);
11. try{
12. UpdateModel(album, "Album");
13. storeDB.SaveChanges();
14. return RedirectToAction("Index");
15. }
16. catch {
17. var viewModel = new StoreManagerViewModel{
18. Album = album,
19. Genres = storeDB.Genres.ToList(),
20. Artists = storeDB.Artists.ToList()
21. };
22. return View(viewModel);
23. }
24. }

```

代码说明：第 1 行定义动作方法 `Edit`，参数是唱片的编号。第 2~6 行实例化唱片管理模型视图类的对象 `viewModel`，其中，第 3 行获得指定产品编号的唱片对象作为 `viewModel` 对象 `Album` 属性的值。第 4 行获得唱片类型列表集合作为 `viewModel` 对象 `Genres` 属性的值。第 5 行获得演唱者对象列表作为 `viewModel` 对象 `Artists` 属性的值。第 7 行返回 `viewModel` 对象到视图。

第 9 行定义一个重载的动作方法 `Edit`，参数是唱片的编号和表单集合对象。第 10 行定义指定唱片编号的唱片对象 `album`。第 12 行调用 `UpdateModel` 方法更新唱片对象 `album`。第 13 行保存数

数据库的更改。第 14 行重定向到管理产品的视图。如果第 12 行更新失败，则第 17~21 行实例化唱片管理模型视图类的对象 `viewModel`，其中，第 18 行获得指定产品编号的唱片对象作为 `viewModel` 对象 `Album` 属性的值。第 19 行获得唱片类型列表集合作为 `viewModel` 对象 `Genres` 属性的值。第 20 行获得演唱者对象列表作为 `viewModel` 对象 `Artists` 属性的值。第 22 行返回 `viewModel` 对象到编辑唱片视图。

设计对应与动作方法 `Edit` 的视图，该视图定义在 `Views` 文件夹下 `StoreManager` 子文件夹下的 `Edit.aspx` 文件中，关键的 HTML 代码如下。

```
1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <% Html.EnableClientValidation(); %>
3. <% using (Html.BeginForm()) {%>
4. <%: Html.ValidationSummary(true) %>
5. <fieldset>
6. <legend>编辑专辑</legend>
7. <%: Html.EditorFor(model => model.Album,
8. new { Artists = Model.Artists, Genres = Model.Genres}) %>
9. <p> <input type="submit" value="保存" /></p>
10. </fieldset>
11. <% } %>
12. <div><%: Html.ActionLink("返回目录", "Index") %></div>
13. </asp:Content>
```

代码说明：第 2 行指定启用客户端脚本来执行输入验证。第 3 行定义使用一个表单。第 4 行启用显示验证错误的列表。第 5~10 行定义一个边框。第 7~8 行使用 `Html` 的 `EditorFor` 方法显示 `EditorTemplates` 文件夹下的模板 `Album.ascx` 用户控件模板，在该模板中定义了显示唱片对象的 HTML 代码。第 9 行定义一个“保存”按钮。第 12 行定义一个“返回列表”的超链接。

设计后编辑唱片视图的界面如图 18-28 所示。

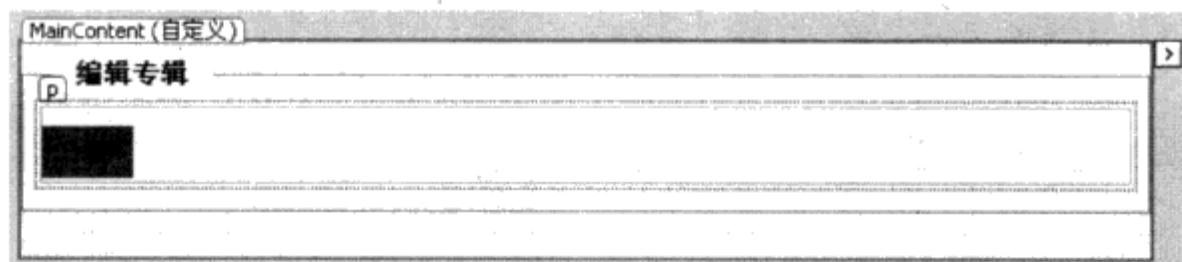


图 18-28 设计后编辑唱片界面

### 18.6.3 删除唱片页面

删除唱片页面控制器动作方法 `Delete` 定义在 `Controllers` 文件夹下的 `StoreManagerController.cs` 文件夹中，关键代码如下。

```
1. public ActionResult Delete(int id){
2. var album = storeDB.Albums.Single(a => a.AlbumId == id);
3. return View(album);
4. }
5. [HttpPost]
6. public ActionResult Delete(int id, string confirmButton){
7. var album = storeDB.Albums
8. .Include("OrderDetails").Include("Carts")
9. .Single(a => a.AlbumId == id);
10. storeDB.DeleteObject(album);
```

```

11. storeDB.SaveChanges();
12. return View("Deleted");
13. }

```

代码说明：第 1 行定义动作方法 Delete，方法的参数是唱片的编号 id。第 2 行调用实体数据模型上下文对象 storeDB 中唱片对象 Albums 的 Single 方法，获得指定唱片编号的唱片对象 album。第 3 行将该对象传递到对应的 Delete 视图中。

第 5 行属性表示下面的方法只接受用户通过 Post 方法发送表单数据。第 6 行定义了一个重载的 Delete 动作方法，方法的参数是唱片编号 id 和 FormCollection 窗体传递值的集合对象。第 7~9 行获取包含订单详情和购物车数据的指定唱片编号的唱片对象 album。第 10 行调用 DeleteObject 方法删除该唱片对象。第 11 行调用 SaveChanges 方法将修改的数据保存到数据库。通过第 12 行将数据传递到对应的 Deleted 视图。

设计对应与动作方法 Delete 的视图，该视图定义在 Views 文件夹下 StoreManager 子文件夹下的 Delete.aspx 文件中，关键的 HTML 代码如下。

```

1. <asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
2. <p> 你确定要删除该专辑名为
3. <%: Model.Title %>?
4. </p>
5. <div>
6. <% using (Html.BeginForm()) {%>
7. <input type="submit" value="删除" />
8. <% } %>
9. </div>
10. </asp:Content>

```

代码说明：第 2~4 行在段落标记中显示删除确认唱片的标题。第 6 行定义一个表单。第 7 行显示一个“删除”按钮。

设计后删除唱片视图的界面如图 18-29 所示。

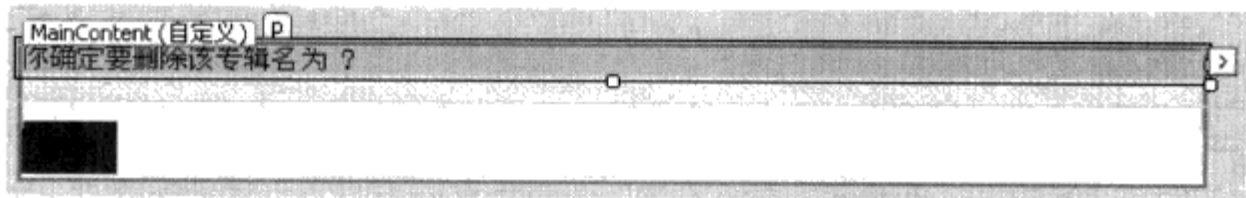


图 18-29 设计后删除唱片界面

本模块中创建唱片、删除唱片这两个功能的控制器和视图与以上介绍的三个功能类似，这里不再进行详细的说明，读者可以参考光盘中的源代码进行学习。

# 第 19 章 项目实例

## 19.1 项目 01：图书管理系统

图书是人类知识传承的载体，也是人们获得知识的一种主要途径，而图书馆在知识传播的过程中起了极大的作用。它作为大量图书流动的場所，对图书管理的好坏，直接影响到知识的传播问题。过去的图书管理模式从新书的购买、编码、入库、上架，到借阅、续借、归还、查询，全部是手工处理，需要大量的工作量与劳动力。而且，在此过程中由人为的因素造成的失误，也是不可避免的。

图书馆要做到顺利而有效地运转，就必须有信息管理系统的支持和帮助。通过软件系统实现对图书馆的便捷、高效、合理的管理。本节将介绍一个具备基本功能的图书管理系统。

### 19.1.1 系统分析与设计

#### 1. 系统需求分析

根据图书管理的功能要求，结合图书馆的实际情况，本系统需要实现的用户需求描述如下。

- ① 系统管理员从登录界面进入系统，在登录页面输入用户名和密码，通过身份验证后，方可进入系统的首页。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ② 通过身份验证的系统管理员进入系统的首页。在首页中可以进行图书管理的操作：包括图书信息的管理和图书类型的管理。对这两个操作都具备添加、查看、修改和删除的功能。
- ③ 对于图书馆而言，最常用的操作就是借阅图书和归还图书的操作。
- ④ 系统管理员可以根据需要，选择不同的条件：图书编号、图书名称、图书类别、图书作者、图书的出版社和所在书架，对图书信息进行查询。
- ⑤ 系统管理员可以根据图书编号、图书名称、读者编号、读者名称和借阅时间对图书的借阅信息进行查询。
- ⑥ 系统管理员可以对读者进行统一管理，包括对读者信息的管理和读者类型的管理。对这两个操作都具备了添加、查看、修改和删除的功能。
- ⑦ 系统管理员可以对系统进行设置。在系统设置中，可以添加管理员和设置管理员的权限，能够对书架进行添加、修改和删除的操作。
- ⑧ 系统管理员还能够对自己的密码进行重新设定。

#### 2. 系统模块设计

根据上述的系统需求分析，我们对本系统的模块进行划分，系统分为六大模块：管理产品模块、管理入库模块、管理出库模块、管理盘存模块、管理用户和系统管理模块。各模块所包含的文件及其功能如表 19-1 所示。

表 19-1 仓库管理模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/DataBase.cs	数据库公共访问类代码文件
	App_Code/OperatorClass.cs	公共操作类代码文件
	App_Code/AdminManage.cs	管理系统管理员代码文件
	App_Code/AuthorityManage.cs	权限管理代码文件
	App_Code/BookManage.cs	图书管理代码文件
	App_Code/BookShelfManage.cs	书架管理代码文件
	App_Code/BookTypeManage.cs	图书类别管理代码文件
	App_Code/BorrowBackMange.cs	图书借还管理代码文件
	App_Code/ReaderManage.cs	读者管理代码文件
	App_Code/ReaderTypeManage.cs	读者类型管理代码文件
	App_Code/ValidateClass.cs	验证类代码文件
图书借还管理模块	BookBackManage/BorrowBook.aspx	借阅图书界面设计文件
	BookBackManage/BorrowBook.aspx.cs	实现借阅图书界面的代码文件
	BookBackManage/ReturnBook.aspx	归还图书界面设计文件
	BookBackManage/ReturnBook.aspx.cs	实现归还图书界面的代码文件
图书信息管理模块	BookManage/AddBook.aspx	添加图书界面设计文件
	BookManage/AddBook.aspx.cs	实现添加图书界面的代码文件
	BookManage/AddBookType.aspx	添加图书类型界面设计文件
	BookManage/AddBookType.aspx.cs	实现添加图书类型界面的代码文件
	BookManage/BookTypeManage.aspx	图书类型管理界面设计文件
	BookManage/BookTypeManage.aspx.cs	实现图书类型管理界面的代码文件
	BookManage/BookManage.aspx	图书管理界面设计文件
	BookManage/BookManage.aspx.cs	实现图书管理界面的代码文件
系统查询模块	SysemQuery/BorrowBackQuery.aspx	图书借阅查询界面设计文件
	SysemQuery/BorrowBackQuery.aspx.cs	实现图书借阅查询界面的代码文件
	SysemQuery/BookQuery.aspx	图书查询界面设计文件
	BookQuery.aspx	实现图书查询界面的代码文件
系统设置模块	SystemSet/AddAdmin.aspx	添加管理员界面设计文件
	SystemSet/AddAdmin.aspx.cs	实现添加管理员界面的代码文件
	SystemSet/AddBookShelf.aspx	添加书架界面设计文件
	SystemSet/AddBookShelf.aspx.cs	实现添加书架界面的代码文件
	SystemSet/AdminManage.aspx	管理管理员界面设计文件
	SystemSet/AdminManage.aspx.cs	实现管理管理员界面的代码文件
	Password/ChanagePwd.aspx	更新密码界面设计文件
	Password/ChanagePwd.aspx.cs	实现更新密码界面的代码文件
	SystemSet/BookShelfManage.aspx	书架管理界面设计文件
	SystemSet/BookShelfManage.aspx.cs	实现书架管理界面的代码文件

(续表)

模块名	文件名	功能描述
读者管理模块	ReaderManage/AddReader.aspx	添加读者界面设计文件
	ReaderManage/AddReader.aspx.cs	实现添加读者界面的代码文件
	ReaderManage/AddReaderType.aspx	添加读者类型界面设计文件
	ReaderManage/AddReaderType.aspx.cs	实现添加读者类型界面的代码文件
	ReaderManage/ReaderManage.aspx	读者管理界面设计文件
	ReaderManage/ReaderManage.aspx.cs	实现读者管理界面的代码文件
	ReaderManage/ReaderTypeManage.aspx	读者类型管理界面设计文件
	ReaderManage/ReaderTypeManage.aspx.cs	实现读者类型管理界面的代码文件

19.1.2 系统数据库设计

根据系统的需求分析，我们开始对数据库进行合理的设计。首先在 Sql Server 2005 中建立一个名为“LibraryManage”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [LibraryManage] //创建数据库
USE [LibraryManage] //使用数据库
```

根据系统的需求分析和模块设计，至少需要的数据表包括：用户信息表、用户权限表、书架信息表、图书类别表、图书借还信息表、读者信息表、读者类别表。

(1) 用户信息表 (tb\_admin)，用来记录使用本系统用户的信息，使用用户编号 id 作为表的主键。该表的字段结构如表 19-2 所示。

表 19-2 tb\_admin 表结构

字段	中文描述	数据类型	是否为空	备注
id	用户编号	varchar(50)	否	主键
name	用户姓名	varchar(50)	是	.
pwd	用户密码	varchar(30)	是	

(2) 用户权限表 (tb\_authority)，用来记录使用本系统用户拥有的权限信息，id 用户权限编号是该表的主键。该表的字段结构如表 19-3 所示。

表 19-3 tb\_authority 表结构

字段	中文描述	数据类型	是否为空	备注
id	用户权限编号	varchar(50)	否	主键
systemset	系统管理的权限	bit	是	
readset	读者管理权限	bit	是	
bookset	图书管理权限	bit	是	
borrowback	图书借还管理权限	bit	是	
systemquery	系统查询权限	bit	是	

（3）图书信息表（tb\_bookinfo），用来记录所有图书的详细信息，我们选择图书编号 bookcode 作为主键。该表的字段结构如表 19-4 所示。

表 19-4 tb\_bookinfo 表结构

字段	中文描述	数据类型	是否为空	备注
bookcode	图书编号	varchar(30)	否	主键
bookname	图书名称	varchar(50)	是	
type	图书类型	varchar(50)	是	
author	作者	varchar(50)	是	
translator	译者	varchar(50)	是	
press	出版社	varchar(100)	是	
price	价格	money	是	
page	页数	int	是	
bookshelf	所在书架	varchar(50)	是	
storage	库存总数	bigint	是	
inTime	购买时间	smalldatetime	是	
operater	操作人	varchar(30)	是	
borrownum	可借数量	int	是	

（4）书架信息表（tb\_bookshelf），用来记录放置图书的所有书架信息，书架编号 id 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-5 所示。

表 19-5 tb\_bookshelf 表结构

字段	中文描述	数据类型	是否为空	备注
id	书架编号	varchar(30)	否	主键
name	书架名称	varchar(50)	是	

（5）图书类别表（tb\_booktype），用来记录所有图书的类别信息，图书类型名称 typename 是主键。该表的字段结构如表 19-6 所示。

表 19-6 tb\_booktype 表结构

字段	中文描述	数据类型	是否为空	备注
id	图书类型编号	int	否	主键
typename	图书类型名称	varchar(30)	否	
days	可借天数	int	是	

（6）图书借还表（tb\_borrowback），用来记录图书借阅和归还的详细信息，根据主键唯一性原则，设定图书借还编号 id 为主键。该表的字段结构如表 19-7 所示。

表 19-7 tb\_borrowback 表结构

字段	中文描述	数据类型	是否为空	备注
id	图书借还编号	varchar(30)	否	主键
readerid	读者编号	varchar(20)	是	
bookcode	图书编号	varchar(30)	是	
borrowTime	借阅时间	smalldatetime	是	
shouldbackTime	应还时间	smalldatetime	是	
ActualbackTime	实际还书时间	smalldatetime	是	
borrowoperator	借书操作人	varchar(30)	是	
backoperator	还书操作人	varchar(30)	是	
isback	是否已还	bit	是	

（7）读者信息表（tb\_reader），用来记录所有读者的详细信息，其中，用户姓名可以重复，而用户编号是不可重复的，所以用读者编号 id 作为该表的主键。该表的字段结构如表 19-8 所示。

表 19-8 tb\_reader 表结构

字段	中文描述	数据类型	是否为空	备注
id	读者编号	varchar(30)	否	主键
name	读者姓名	varchar(50)	是	
sex	读者性别	char(4)	是	
type	读者类型	varchar(50)	是	
birthday	读者生日	smalldatetime	是	
paperType	证件类型	varchar(20)	是	
paperNum	证件编号	varchar(30)	是	
tel	读者电话	varchar(20)	是	
email	读者电子邮件	varchar(50)	是	
createDate	创建日期	smalldatetime	是	
operator	操作人	varchar(30)	是	
remark	备注	text	是	
borrownum	借阅编号	int	是	
num	可借数量	int	是	

（8）读者类型表（tb\_readertype），用来记录所有读者的类型信息。该表的字段结构如表 19-9 所示。

表 19-9 tb\_readertype 表结构

字段	中文描述	数据类型	是否为空	备注
id	读者类型编号	int	否	
name	读者姓名	varchar(50)	否	主键
number	可借数量	int	是	

19.1.3 系统运行示例

系统运行后，首先出现的是登录页面，如图 19-1 所示。



图 19-1 用户登录页面

在该页面中，输入用户名和密码，单击“登录”按钮，通过身份验证后进入系统首页。如图 19-2 所示。

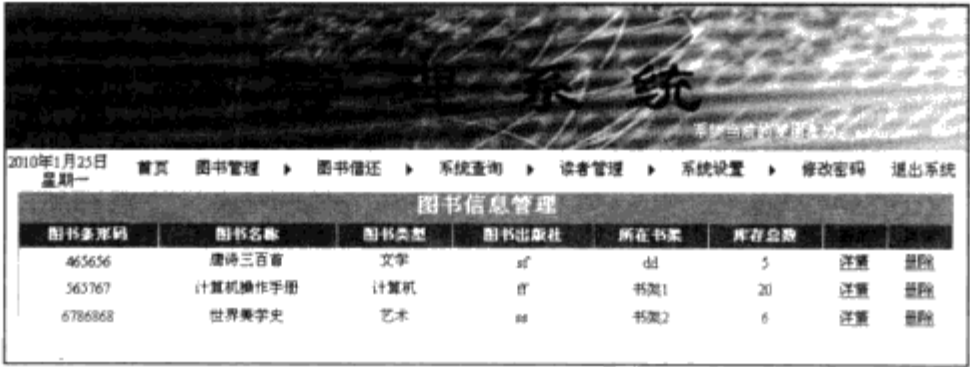


图 19-2 系统首页

在首页中，可以对图书进行管理。单击图书列表中“详情”的链接，进入修改图书信息的页面，如图 19-3 所示。

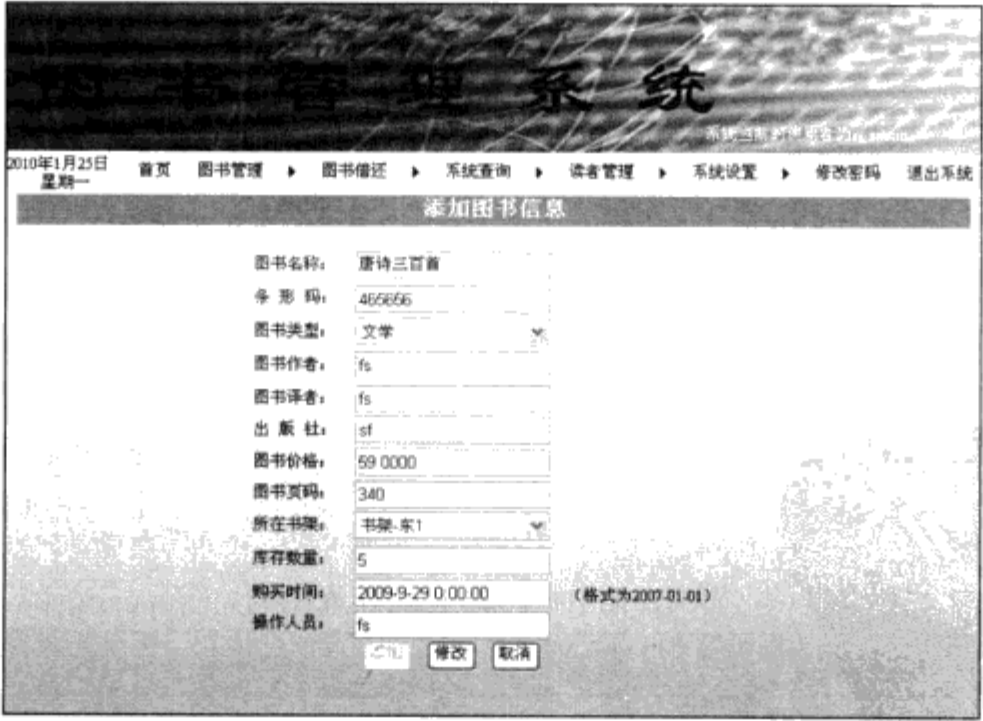


图 19-3 修改图书信息页面

在首页中，把鼠标放到菜单栏上的“图书借还”上，在弹出的二级菜单中的选择“图书借阅”子菜单，进入图书借阅的界面，如图 19-4 所示。

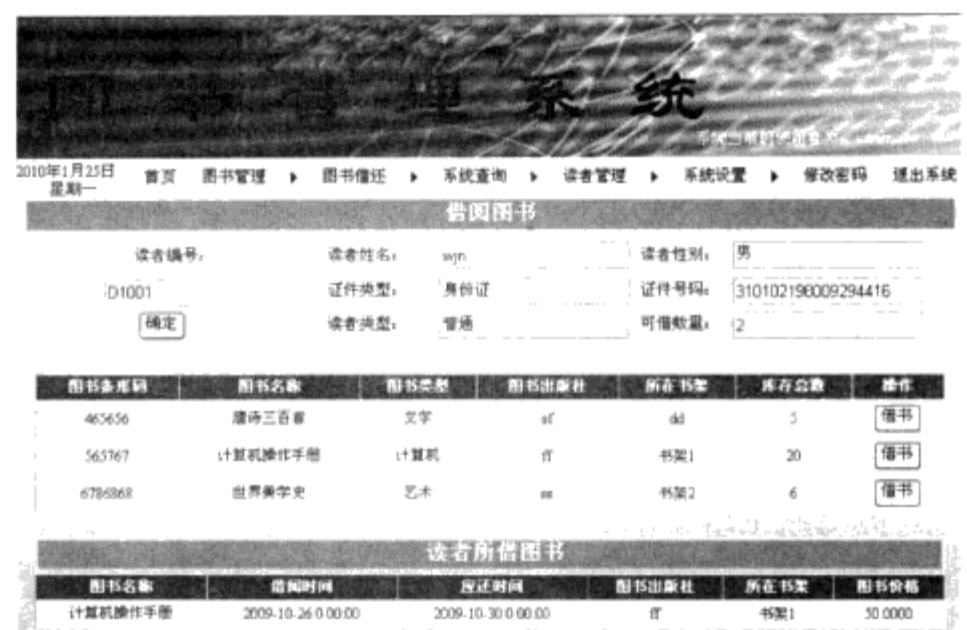


图 19-4 图书借阅界面

在页面中输入读者编号，读者的信息会出现在右边的文本框中。当单击页面中间图书信息列表中的借书按钮，可以进行借书的操作。

在首页中，把鼠标放到菜单栏上的“图书借还”上，在弹出的二级菜单中选择“图书归还”子菜单，进入图书归还界面，如图 19-5 所示。

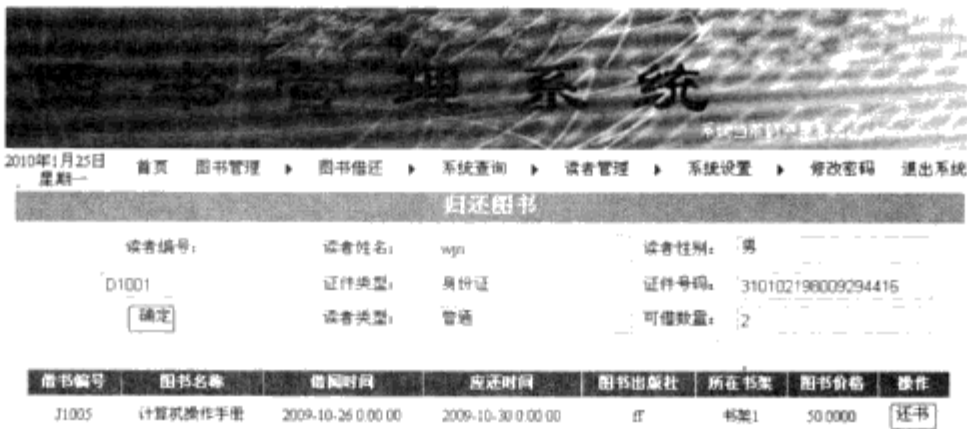


图 19-5 图书归还界面

在页面中输入读者的编号，读者的信息会出现在右边的文本框中。同时，所借的图书信息会显示页面下部，当单击页面中图书信息列表中的“还书”按钮，可以进行还书的操作。

在首页中，把鼠标放到菜单栏上的“系统查询”上，在弹出的二级菜单中选择“图书借阅查询”子菜单，进入查询界面，如图 19-6 所示。

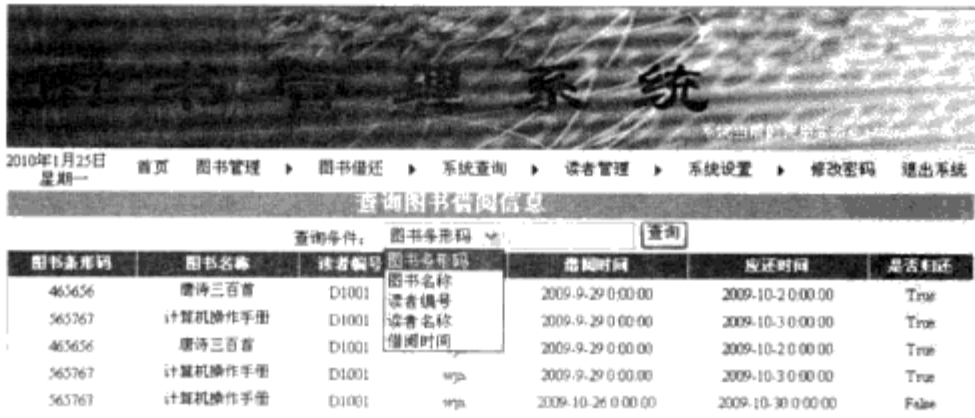


图 19-6 图书借阅查询界面

在页面中,可以根据图书条形码、图书名称、读者编号、读者名称和借阅时间来查询借阅图书的信息。

上面做了系统中主要页面的演示,其他页面基本相似,这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.2 项目 02: 网上校友录

计算机技术的快速发展,特别是计算机网络的发展,越来越深刻地改变了人们生活的方方面面。各种在线服务系统,更是深刻地影响了人们的联系方式,使得人们可以在远隔千里之遥随时通信。过去的种种陈旧的联系方式,已经不能满足现代生活的需要,校友录作为一种方便校友之间联系的实用系统便应运而生。校友录,是一种为用户提供网上交流、聚会的网络工具,它可以使你你的朋友、同学、同事、老师与亲人等在网上有一个相互交流的机会。本节将介绍一个包含最基本功能的校友录。

### 19.2.1 系统分析与设计

本系统设计的目标以信息网络为依托,建立基于网络的校友录管理系统,使校友录成为校友之间进行交流和联系的一个平台。利用网络资源优势和技术优势,通过提供完善的校友录服务和规范校友录的管理,以达到增进校友之间、校友与母校之间的感情,方便校友联系的目的。

#### 1. 系统需求分析

本系统需要实现的用户需求说明如下。

- ① 用户首先必须在系统中进行注册,输入自己的相关信息。
- ② 用户必须在登录页面输入用户名、密码,通过身份验证后,才可以进入操作界面使用各项功能。如果未能通过系统的身份验证,系统自动给出登录错误的提示信息。
- ③ 用户可以创建和查询学校。
- ④ 用户也能够创建班级和查询班级的信息。
- ⑤ 用户可以在班级中浏览班级的通信录。
- ⑥ 用户可以在班级中浏览留言、发表留言以及编辑和删除留言。
- ⑦ 允许用户将重要的留言置于留言列表的顶部。

#### 2. 系统模块设计

根据系统的需求分析,我们对本系统的模块进行划分,系统分为 4 大模块:用户注册模块、用户登录模块、学校班级管理模块和留言管理模块。各模块所包含的文件及其功能如表 19-10 所示。

表 19-10 校友录系统模块一览表

模块名	文件名	功能描述
用户注册模块	Register.aspx	新用户注册界面设计的代码文件
	Register.aspx.cs	实现新用户注册界面的代码文件
用户登录模块	Default.aspx	用户登录界面设计的代码文件
	Default.aspx.cs	实现用户登录界面的代码文件
学校班级管理模块	Welcome.aspx	注册班级界面设计的代码文件
	Welcome.aspx.cs	实现注册班级界面的代码文件
	FindSchool.aspx	确定学校所在地区界面设计的代码文件
	FindSchool.aspx.cs	实现确定学校所在地区界面的代码文件
	ChooseSchool.aspx	查询具体学校界面设计的代码文件
	ChooseSchool.aspx.cs	实现查询具体学校界面的代码文件
	FindClass.aspx	查询具体班级界面设计的代码文件
	FindClass.aspx.cs	实现查询具体班级界面的代码文件
	Register.aspx	注册管理员界面设计的代码文件
	Register.aspx.cs	实现注册管理员界面的代码文件
留言管理模块	WinForm.aspx	管理留言界面设计的代码文件
	WinForm.aspx.cs	实现管理留言界面的代码文件
留言管理模块	EditMessage.aspx	修改留言界面设计的代码文件
	EditMessage.aspx.cs	实现修改留言界面的代码文件
	DeleteMessage.aspx	删除留言界面设计的代码文件
	DeleteMessage.aspx.cs	实现删除留言界面的代码文件
	LockMessage.aspx	留言固定界面设计的代码文件
	LockMessage.aspx.cs	实现留言固定界面的代码文件
	AddressList.aspx	显示班级通信录界面设计的代码文件
	AddressList.aspx.cs	实现显示班级通信录界面的代码文件
	UnLockMessage.aspx	取消留言固顶界面设计的代码文件
	UnLockMessage.aspx.cs	实现取消留言固顶界面界面的代码文件

19.2.2 系统数据库设计

根据系统的需求分析，我们对数据库进行设计。首先在 Sql Server 2005 中建立一个名为 “AlumniRecordSys” 的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [AlumniRecordSys] //创建数据库
USE [AlumniRecordSys] //使用数据库
```

根据前面的系统需求分析和模块设计，所需要的数据表包括：留言信息表、城市信息表、班级信息表、班级类型信息表、省份信息表、学校信息表 and 用户信息表。

(1) 留言信息表（board），用来记录用户在班级中的留言信息，使用留言编号 b\_id 作为表的主键。该表的字段结构如表 19-11 所示。

表 19-11 board 表结构

字段	中文描述	数据类型	是否为空	备注
b_id	留言编号	int	否	主键
b_theme	主题	nvarchar(40)	是	
b_cnt	内容	ntext	是	
b_clid	班级编号	int	是	
b_uid	留言人	int	是	
b_date	留言时间	smalldatetime	是	
b_delflag	删除标志	nvarchar(1)	是	
b_editer	编辑人	int	是	
b_edittime	编辑时间	smalldatetime	是	
b_top	是否固顶	smallint	是	

(2) 城市信息表 (city)，用来记录全国城市的信息，c\_id 城市编号是该表的主键。该表的字段结构如表 19-12 所示。

表 19-12 city 表结构

字段	中文描述	数据类型	是否为空	备注
c_id	城市编号	int	否	主键
c_name	城市名称	nvarchar(50)	是	
c_pid	省份编号	int	是	

(3) 班级信息表 (classinfo)，用来记录所有班级的信息，我们选择班级编号 l\_id 作为主键。该表的字段结构如表 19-13 所示。

表 19-13 classinfo 表结构

字段	中文描述	数据类型	是否为空	备注
cl_id	班级编号	int	否	主键
cl_name	班级名称	nvarchar(40)	是	
cl_pid	省份编号	int	是	
cl_cid	城市编号	int	是	
cl_sid	学校编号	int	是	
cl_delflag	删除标志	nvarchar(1)	是	
cl_m1	主管理员	int	是	
cl_m2	副管理员	int	是	
cl_type	班级类型	int	是	
cl_num	成员数量	smallint	是	
cl_grad	年级	smallint	是	

(4) 班级类型信息表 (classtype)，用来记录班级类型的信息，类型编号 t\_id 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-14 所示。

表 19-14 classtype 表结构

字段	中文描述	数据类型	是否为空	备注
t_id	班级类型编号	int	否	主键
t_name	班级类型名称	nvarchar(30)	是	

(5) 省份信息表（prove），用来记录全国省、市、自治区的信息，省份编号 p\_id 是主键。该表的字段结构如表 19-15 所示。

表 19-15 prove 表结构

字段	中文描述	数据类型	是否为空	备注
p_id	省份编号	int	否	主键
p_name	省份名称	nvarchar(10)	是	

(6) 学校信息表（school），用来记录所有学校的详细信息，学校编号 s\_id 是主键。该表的字段结构如表 19-16 所示。

表 19-16 school 表结构

字段	中文描述	数据类型	是否为空	备注
s_id	学校编号	int	否	主键
s_name	学校名称	nvarchar(40)	是	
s_pid	省份编号	int	是	
s_cid	城市编号	int	是	
s_delflag	删除标志	nvarchar(1)	是	
s_erea	地区	nvarchar(16)	是	
s_type	学校类型	int	是	
s_num	班级数量	int	是	
s_zip	邮政编码	nvarchar(16)	是	
s_http	网址	nvarchar(50)		
s_who	管理员	int	是	

(7) 用户信息表（userinfo），用来记录所有使用校友录用户的详细信息，用户编号 u\_id 是主键。该表的字段结构如表 19-17 所示。

表 19-17 school 表结构

字段	中文描述	数据类型	是否为空	备注
u_id	用户编号	int	否	主键
u_name	用户名称	nvarchar(8)	是	
u_account	账户	nvarchar(12)	是	
u_pwd	密码	nvarchar(20)	是	

(续表)

字段	中文描述	数据类型	是否为空	备注
u_grp	用户组	nvarchar(1)	是	
u_class	班级编号	nvarchar(16)	是	
u_sex	性别	nvarchar(2)	是	
u_bth	生日	smalldatetime	是	
u_regdate	注册日期	smalldatetime	是	
u_tell	电话	nvarchar(16)	是	
u_addr	地址	nvarchar(60)	是	
u_zip	邮政编码	nvarchar(6)	是	
u_email	电子邮件	nvarchar(30)	是	
u_job	工作单位	nvarchar(60)	是	
u_icq	ICQ	nvarchar(12)	是	
u_qq	QICQ	nvarchar(8)	是	

19.2.3 系统运行示例

系统运行后，出现登录页面，如图 19-7 所示。

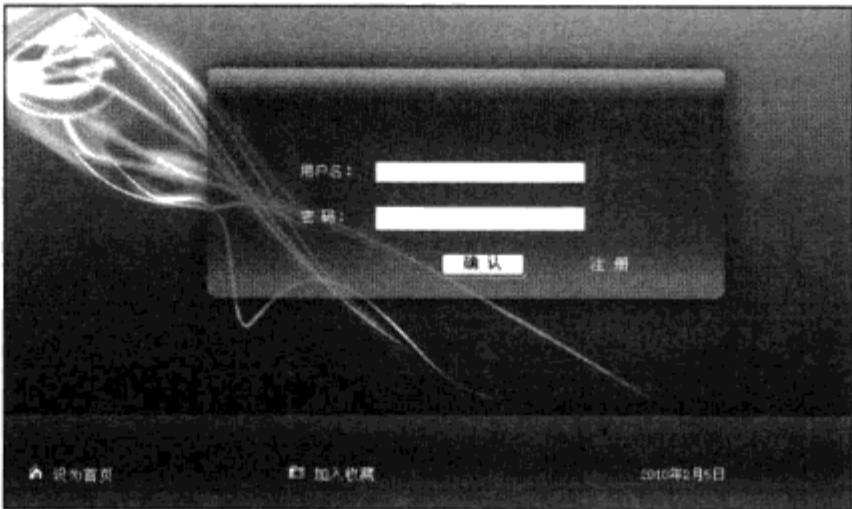


图 19-7 用户登录页面

在登录界面输入用户名和密码，单击“登录”按钮，进入欢迎页面，如图 19-8 所示。

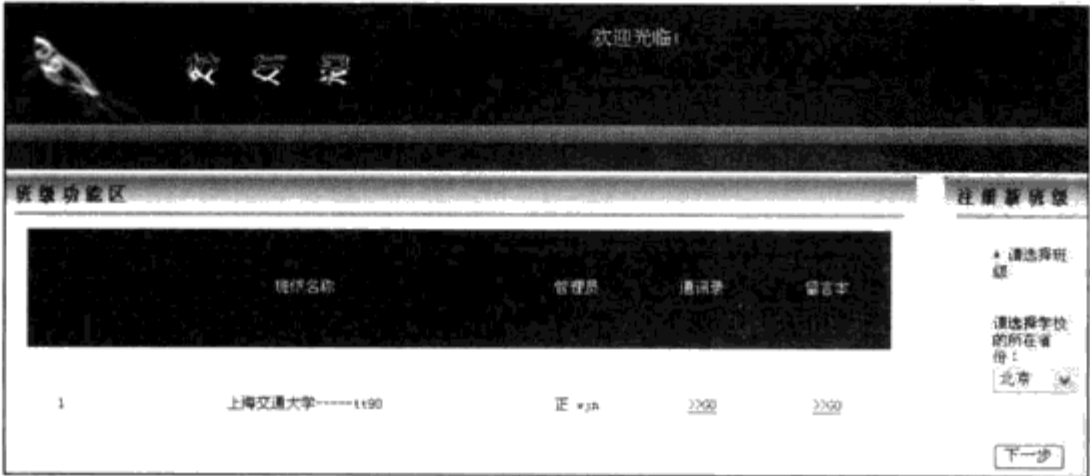


图 19-8 欢迎页面

在该页面，如果用户还没有设置自己的班级，选择班级所在学校的省份。单击“下一步”，进入班级搜索页面，如图 19-9 所示。



图 19-9 班级搜索页面

在该页面，可以选择学校的所在地、学校的类型，单击“下一步”，进入注册学校页面，如图 19-10 所示。



图 19-10 注册学校页面

我们可以在页面中输入自己学校的名称、所在地、邮编和网址信息，单击“下一步”，进入添加班级的页面，如 19-10 所示。



图 19-11 添加班级页面

在页面中输入班级名称和入学时间，单击“下一步”按钮，返回到欢迎页面，这时刚刚注册的学校和班级的信息出现信息列表中。可以单击“通信录”链接，进入通信录页面，如图 19-12 所示。

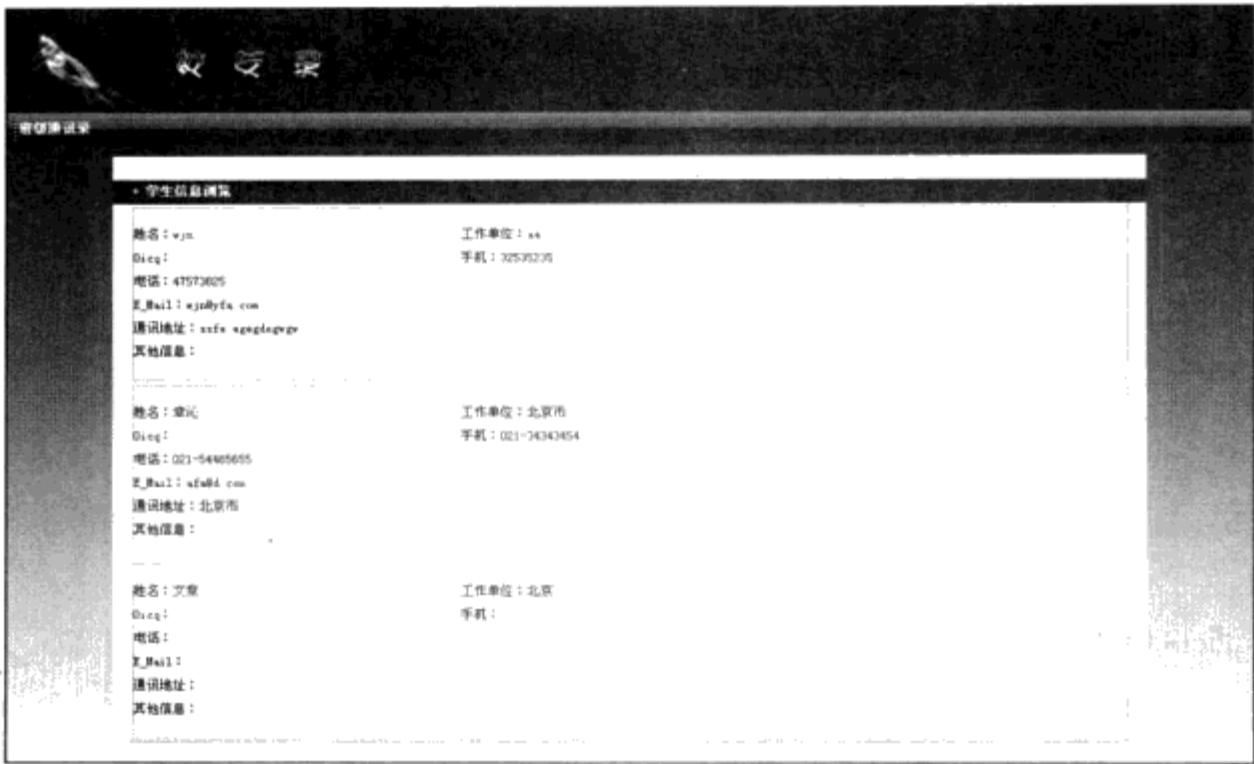


图 19-12 通信录页面

在欢迎页面中，单击“留言本”，进入留言本页面，如图 19-13 所示。

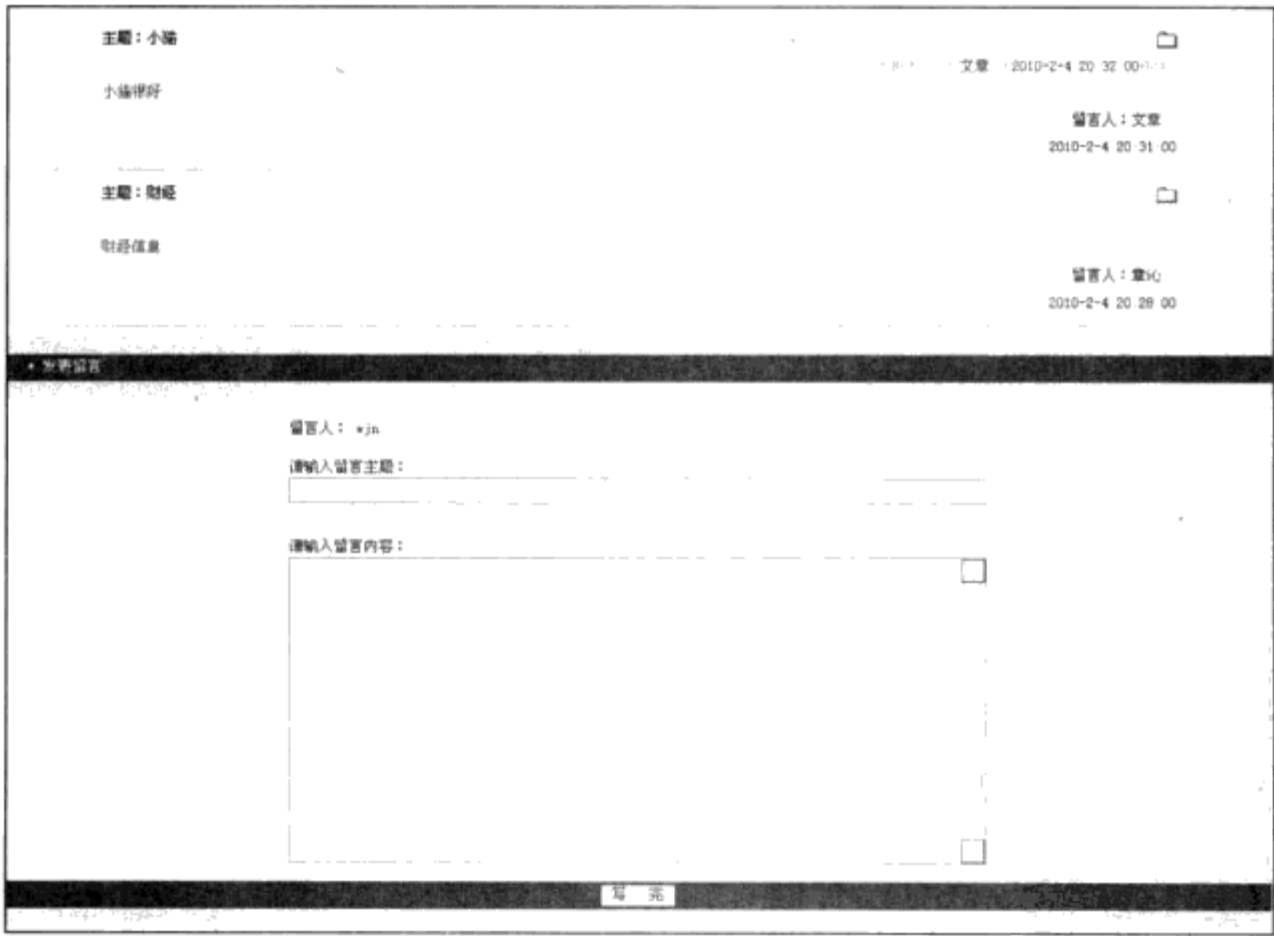


图 19-13 留言本页面

我们可以在留言本中发表、编辑和删除自己的留言。

上面做了系统中主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.3 项目 03：考勤管理系统

考勤是一个企业的最基本的管理内容，是企业对员工工作评定的基本依据。在实际的管理中需要快速获得每一个员工每一个工作日的考勤，以便及时向管理者反映员工的出勤，缺勤情况。为此，本章介绍的考勤管理系统基本实现了企业考勤的智能化管理，提高了考勤管理的效率，每个员工的工作状态能得到及时的反应。同时，增强员工管理的透明度以及约束员工自觉遵守出勤制度。

### 19.3.1 系统分析与设计

本系统是基于 B/S 结构的网络考勤系统，实现网上考勤任务，以减轻考勤人员的工作量。

#### 1. 系统需求分析

本系统设计的目的是：规范企业管理，实现对企业所有员工监控管理，建立比较完善的考勤管理制度，从而为企业领导决策提供依据。依据这样的设计思路，本系统需要实现的需求功能综述如下。

本系统的用户主要有两种，一种是考勤管理人员，一种是被考勤的员工。

① 对于考勤管理人员而言，从登录界面进入系统，在登录页面输入用户名和密码，通过身份验证后，才可以进入系统的首页。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。

② 通过身份验证的考勤管理人员进入系统的首页。在首页中可以通过员工编号、考勤年度和考勤月度的不同条件对员工的考勤记录进行查询。

③ 考勤管理人员能够选择员工编号、员工姓名、员工职位和员工所属部门的条件对员工的信息进行查询。同时，也可以将新员工的信息进行添加和对选择的员工的信息进行修改。

④ 考勤管理员可以根据需要，添加新的部门信息和修改原来的部门信息。

⑤ 考勤管理员能够添加新的职位信息并对该职位的上下班打卡时间进行设置。同样的，也可以对职位信息进行修改操作。

⑥ 考勤管理员也可以随时更新自己的登录密码。

⑦ 对于普通员工来说，也必须从登录界面进入系统，在登录页面输入用户名和密码，通过身份验证后，才可以进入系统的首页。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。

⑧ 普通员工在上班和下班时通过本系统进行打卡考勤。

⑨ 员工可以修改自己的登录密码。

#### 2. 系统模块设计

根据系统的需求分析，我们对本系统的模块进行划分，系统分为六大模块：数据库管理模块、实体类模块、考勤管理模块、员工管理模块和系统设置模块。各模块所包含的文件及其功能如表 19-18 所示。

表 19-18 考勤管理模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/DBL/Log.cs	数据库公共访问类代码文件
	App_Code/Logic/AttendanceLogic.cs	考勤类业务逻辑代码文件
	App_Code/ Logic/ DepartmentLogic.cs	部门类业务逻辑代码文件
	App_Code/ Logic/ EducationLogic.cs	学历类业务逻辑代码文件
	App_Code/ Logic/ EmployeeLogic.cs	员工信息类业务逻辑代码文件
	App_Code/ Logic/ LogLogic.cs	系统日志类业务逻辑代码文件
	App_Code/Logic/ PositionLogic.cs	职位类业务逻辑代码文件
	App_Code/Utility/ DBOperation.cs	数据库公共操作类代码文件
	App_Code/ Utility/ Common.cs	验证类业务逻辑代码文件
实体类模块	App_Code/Model/ AttendanceModel.cs	考勤实体类代码文件
	App_Code/Model/ DepartmentModel.cs	部门信息实体类代码文件
	App_Code/Model/ EducationModel.cs	学历信息实体类代码文件
	App_Code/Model/ EmployeeModel.cs	员工信息实体类代码文件
	App_Code/Model/LogModel.cs	系统日志实体类代码文件
	App_Code/Model/PositionModel.cs	职位信息实体类代码文件
考勤管理模块	Attendance/ AttendanceQuery.aspx	查询考勤信息界面设计文件
	Attendance/ AttendanceQuery.aspx.cs	实现查询考勤信息界面的代码文件
	Attendance/ ComeAttendance.aspx	上班打卡界面设计文件
	Attendance/ ComeAttendance.aspx.cs	实现下班打卡界面设计的文件
	Attendance/ LeaveAttendance.aspx	下班打卡界面设计文件
	Attendance/ LeaveAttendance.aspx.cs	实现下班打卡界面设计的文件
员工管理模块	Employee/EmployeeInfoAdd.aspx	添加员工信息界面设计文件
	Employee/EmployeeInfoAdd.aspx.cs	实现添加员工信息界面的代码文件
	Employee/ EmployeeInfoUpdate.aspx	修改员工信息界面设计文件
	Employee/ EmployeeInfoUpdate.aspx.cs	实现修改员工信息界面的代码文件
	Employee/ EmployeeManage.aspx	员工信息管理界面设计文件
	Employee/ EmployeeManage.aspx.cs	实现员工信息管理界面的代码文件
系统设置模块	Department/DepartmentDelete.aspx	删除部门信息界面设计文件
	Department/DepartmentDelete.aspx.cs	实现删除部门信息界面的代码文件
	Department/DepartmentManage.aspx	管理部门信息界面设计文件
	Department/DepartmentManage.aspx.cs	实现管理部门信息界面的代码文件
	Position/PositionManage.aspx	职位信息管理界面设计文件
	Position/PositionManage.aspx.cs	实现职位信息管理界面的代码文件
	Position/PositionUpdate.aspx	更新职位信息界面设计文件
	Position/PositionUpdate.aspx.cs	实现更新职位信息界面的代码文件
	ChangePassword.aspx	更新密码界面设计文件
	ChangePassword.aspx.cs	实现更新密码界面的代码文件

19.3.2 系统数据库设计

根据以上的需求分析，我们开始对数据库进行合理的设计。首先在 Sql Server 2005 中建立一个名为“AttendanceManage”的数据库来存放本系统所必须的数据表，创建上表的语句如下：

```
CREATE DATABASE [AttendanceManage] //创建数据库
USE [AttendanceManage] //使用数据库
```

根据前面的系统需求分析和模块设计，至少需要的数据表包括：系统管理员表、考勤信息表、部门信息表、员工信息表、学历类别表、职位类别表。

（1）管理员信息表（admin），用来记录所有使用本系统的管理员信息，使用管理员的用户名 adminUsername 作为表的主键。该表的字段结构如表 19-19 所示。

表 19-19 admin 表结构

字段	中文描述	数据类型	是否为空	备注
adminUsername	管理员用户名	varchar(50)	否	主键
adminPassword	管理员密码	varchar(50)	是	

（2）考勤信息表（attendanceInfo），用来记录所有员工的考勤信息，考勤编号 attendanceId 是该表的主键。该表的字段结构如表 19-20 所示。

表 19-20 attendanceInfo 表结构

字段	中文描述	数据类型	是否为空	备注
attendanceId	考勤编号	int	否	主键
attendanceEmployeeNo	员工编号	varchar(20)	是	
attendanceYear	考勤年度	int	是	
attendanceMonth	考勤月份	int	是	
attendanceDay	考勤日	int	是	
attendanceStartTime	上班时间	datetime	是	
attendanceEndTime	下班时间	datetime	是	
attendanceStartStatus	上班状态	smallint	是	
attendanceEndStatus	下班状态	smallint	是	

（3）部门信息表（departmentInfo），用来记录所有部门的详细信息，我们选择部门编号 departmentId 作为主键。该表的字段结构如表 19-21 所示。

表 19-21 tb\_bookinfo 表结构

字段	中文描述	数据类型	是否为空	备注
departmentId	部门编号	int	否	主键
departmentName	部门名称	nvarchar(20)	是	

（4）员工信息表（employeeInfo）用来记录所有员工的信息，员工编号 employeeNo 具有唯一

性，所以，可以设置成为主键字段。该表的字段结构如表 19-22 所示。

表 19-22 employeeInfo 表结构

字段	中文描述	数据类型	是否为空	备注
employeeNo	员工编号	varchar(20)	否	主键
employeeName	员工姓名	nvarchar(20)	是	
employeePassword	员工密码	varchar(30)	是	
employeeSex	员工性别	nchar(1)	是	
employeeBirthday	员工生日	datetime	是	
employeeDepartmentId	员工所属部门编号	int	是	
employeePositionId	员工职位编号	int	是	
employeeEducationId	员工学历编号	int	是	
employeeHomeTel	家庭电话	varchar(20)	是	
employeeMobile	员工手机	varchar(20)	是	
employeeCard	员工身份证号	varchar(20)	是	
employeeEmail	员工电子邮件	varchar(30)	是	
employeeAddress	员工住址	nvarchar(80)	是	

（5）学历信息表（educationInfo），用来记录所有员工的学历信息，学历信息编号 educationId 是主键。该表的字段结构如表 19-23 所示。

表 19-23 educationInfo 表结构

字段	中文描述	数据类型	是否为空	备注
educationId	学历编号	int	否	主键
educationName	学历名称	nvarchar(20)	否	

（6）职位类别表（positionInfo）用来记录职位的详细信息，根据主键唯一性原则，设定职位编号 positionId 为主键。该表的字段结构如表 19-24 所示。

表 19-24 positionInfo 表结构

字段	中文描述	数据类型	是否为空	备注
positionId	职位编号	int	否	主键
positionName	职位名称	nvarchar(50)	是	
positionStartHour	上班时间中的小时	int	是	
positionStartMinute	上班时间中的分钟	int	是	
positionEndHour	下班时间中的小时	int	是	
positionEndMinute	下班时间中的分钟	int	是	

19.3.3 系统运行示例

用户运行系统后，出现的是登录页面，如图 19-14 所示。



图 19-14 用户登录页面

在该页面中，输入用户名和密码，单击“登录”按钮，通过验证后，进入系统首页，如图 19-15 所示。

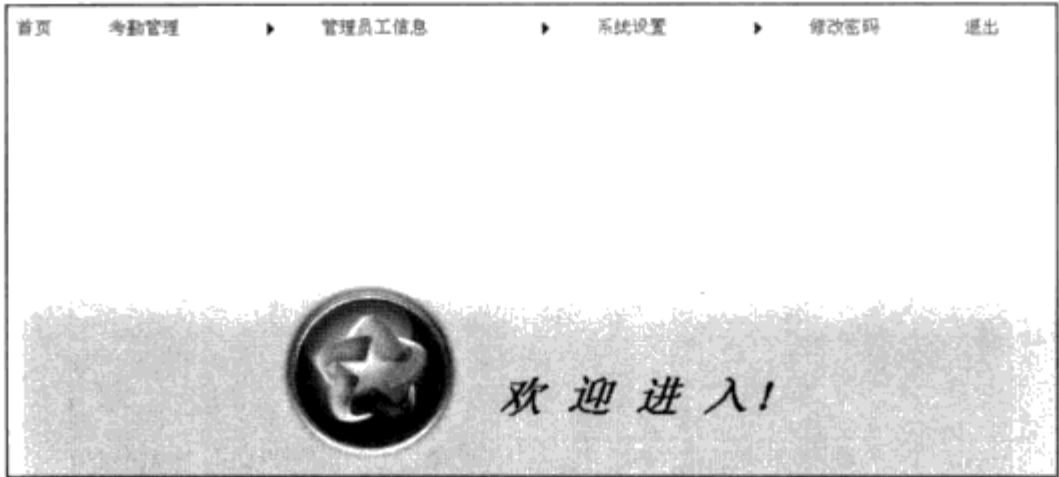


图 19-15 系统首页

在首页中，用户将鼠标放到菜单栏的“考勤管理”菜单上，在弹出的二级菜单中选择“查询考勤记录”，进入查询考勤信息的页面，如图 19-16 所示。

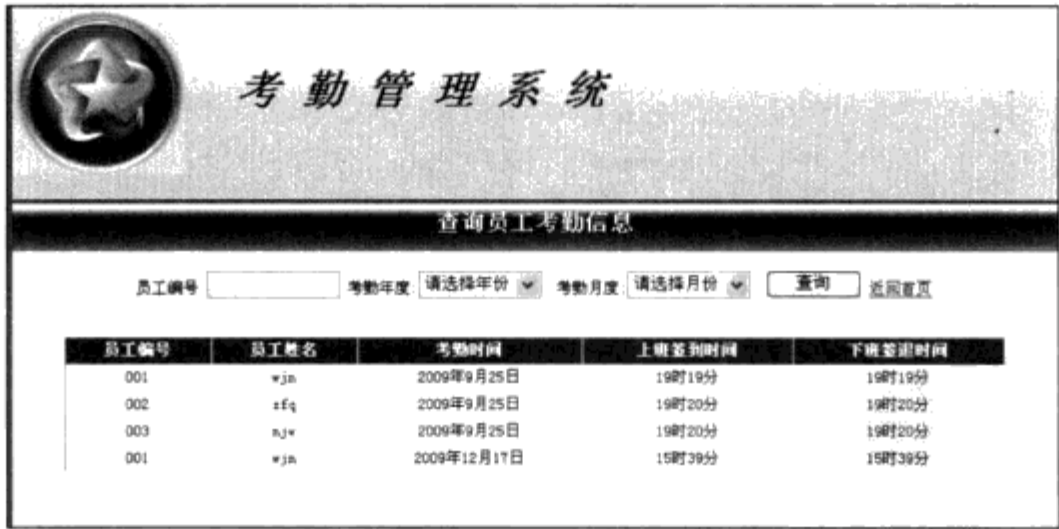



图 19-16 查询考勤信息页面

在该页面中，可以根据员工编号、考勤年度和考勤月度查询相应的考勤信息并显示在下面的数据列表中。

在首页中，如果用户将鼠标放到菜单栏的“管理员工信息”菜单上，在弹出的二级菜单中选择“添加员工信息”，进入添加员工信息的页面，如图 19-17 所示。



考勤管理系统

添加员工信息

编号

姓名

性别

最高学历

部门

职位类别

登陆密码

出生日期

手机

电话

E-mail

身份证

家庭地址

提交

取消

返回首页

图 19-17 添加员工信息

用户在该页面中，填写员工的各种详细的个人信息后，单击“提交”按钮，完成操作。

在首页中，用户将鼠标放到菜单栏的“系统设置”菜单上，在弹出的二级菜单中选择“添加职位设置”，进入设置职位的页面，如图 19-18 所示。



考勤管理系统

添加职位信息

职位类型	上班时间	下班时间	更新
部门经理	9点30分	15点30分	<a href="#">详情</a>
普通员工	9点0分	18点0分	<a href="#">详情</a>
部门主管	8点30分	18点0分	<a href="#">详情</a>
总经理	9点0分	18点0分	<a href="#">详情</a>

添加职位类型信息

职位类型名称

规定上班时间

规定下班时间

0

时

0

分

0

时

0


分

添加

返回首页

图 19-18 添加职位页面

用户在该页面中，单击列表中的“详情”按钮，可以进入更新职位信息的页面，如图 19-19 所示。



考勤管理系统

更新职位信息

职位类型名称

规定上班时间

规定下班时间

部门经理

9

时

30

分

15

时

30

分

更新

返回

图 19-19 更新职位信息页面

在该页面中，用户修改职位名称和上下班的时间后，单击“更新”按钮完成修改操作。

用户在“添加职位信息”页面，填写职位信息和设置好上下班时间后，单击“添加按钮”，可以完成添加职位信息的操作。

在首页中，用户将鼠标放到菜单栏的“系统设置”菜单上，在弹出的二级菜单中选择“部门信息设置”，进入设置部门信息的页面，如图 19-20 所示。

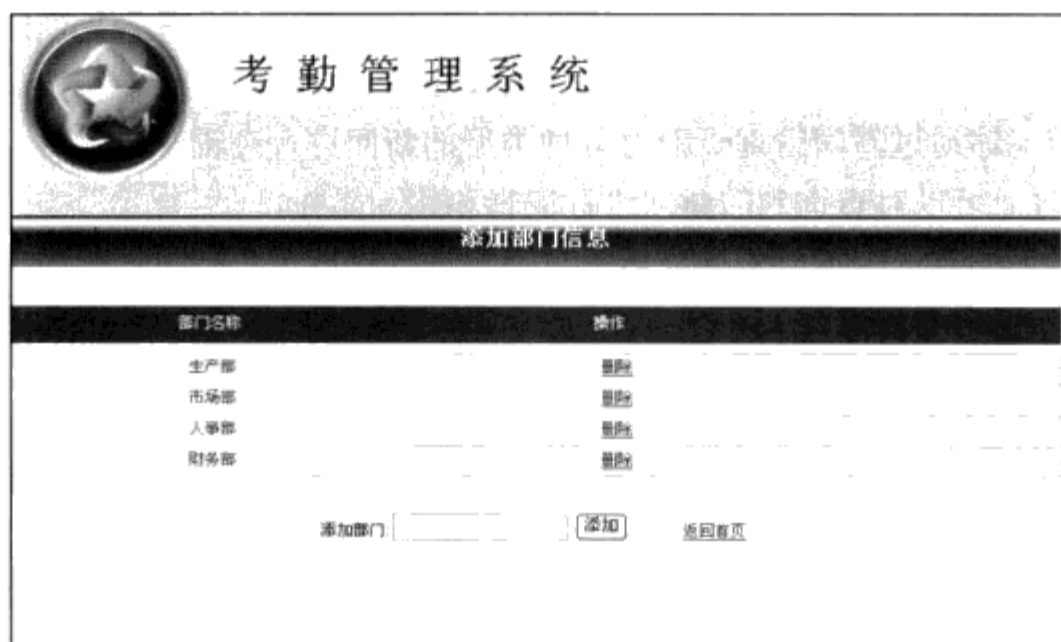


图 19-20 添加部门信息

用户在该页面中，单击“删除”按钮能够对部门信息进行删除操作。在页面下部的文本框中填写部门的名称，单击“添加”按钮可以完成添加部门的操作。

上面做了系统主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.4 项目 04：新闻发布系统

新闻发布系统是对新闻信息进行综合管理的平台，它将网页上的某些需要经常变动的新闻信息集中管理，并通过对其进行分类，最后发布到网站上的一种网站应用程序。新闻内容通过一个操作简单的界面加入数据库，然后通过设计的网页模板格式与审核流程发布到网站上。它的出现大大减轻了网站更新维护的工作量。本系统能够提高新闻的更新速度，大大加快了信息的传播速度，保持网站的活动力和影响力。

### 19.4.1 系统分析与设计

本系统设计的目标是实现网站新闻的动态管理，能高效、及时地对新闻信息进行发布和管理。

#### 1. 系统需求分析

依据本系统的设计思路，需要实现的需求功能综述如下。

① 本系统的用户主要有二类：一类是在网站浏览新闻的普通用户，他们无需经过身份验证就可以在网站浏览各种类型的新闻信息。

- ② 普通用户在页面可以将网站添加到收藏夹，也能够将本网站设为自己的首页。
- ③ 普通用户能够在页面通过选择不同类型的新闻的标题关键字阅读新闻。
- ④ 本系统还有一类用户是系统管理员，系统管理员可以打开系统的后台管理模块的登录界面。在登录页面输入用户名和密码，通过身份验证后，方可进入后台管理的页面。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ⑤ 通过身份验证的系统管理员进入后台管理页面。在该页面中可以对所有的新闻和用户进行管理。
- ⑥ 系统管理员可以对各种类型的新闻进行添加操作。
- ⑦ 系统管理员能够通过新闻管理页面对各种类型的新闻进行查询，或者对关键字进行单条的新闻进行查询。同时，可以对查询到的新闻进行编辑和删除操作。
- ⑧ 系统管理员另外有一个重要的操作就是对用户进行管理，包括添加新的用户信息、编辑原来用户的信息和删除用户的信息。

2. 系统模块设计

根据系统的需求分析，我们对本系统的模块进行划分，系统分为四大模块：数据库管理模块、前台管理模块、后台管理模块和登录模块。各模块所包含的文件及其功能如表 19-25 所示。

表 19-25 新闻发布系统模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/SqlData.cs	数据库公共访问类代码文件
后台管理模块	BackDesk/AddNews.aspx	添加新闻界面设计文件
	BackDesk/ AddNews.aspx.cs	实现添加新闻界面的代码文件
	BackDesk/ AddUser.aspx	添加用户界面设计文件
	BackDesk/ AddUser.aspx.cs	实现添加用户界面的代码文件
	BackDesk/ Index.aspx	后台首页框架设计界面代码文件
	BackDesk/ Index.aspx.cs	实现首页框架设计界面的代码文件
	BackDesk/ Main.aspx	后台首页设计界面的代码文件
	BackDesk/ Main.aspx.cs	实现首页设计界面的代码文件
	BackDesk/ Menu.aspx	首页菜单界面设计文件
	BackDesk/ Menu.aspx.cs	实现首页菜单界面的代码文件
	BackDesk/ NewsManage.aspx	新闻管理界面设计文件
	BackDesk/ NewsManage.aspx.cs	实现新闻管理界面的代码文件
	BackDesk/ UpdateNews.aspx	编辑新闻界面设计文件
	BackDesk/ UpdateNews.aspx.cs	实现编辑新闻界面的代码文件
	BackDesk/ UserManage.aspx	管理用户界面设计文件
	BackDesk/ UserManage.aspx.cs	实现管理用户界面的代码文件
前台管理模块	Default.aspx	前台首页界面设计文件
	Default.aspx.cs	实现添加图书界面的代码文件
	NewsDetail.aspx	显示新闻详情界面设计文件

(续表)

模块名	文件名	功能描述
前台管理模块	NewsDetail.aspx.cs	实现显示新闻详情界面的代码文件
	SelectNews.aspx	查询新闻界面设计文件
	SelectNews.aspx.cs	实现查询新闻界面的代码文件
	ShowNewsInformation.aspx	显示新闻信息界面设计文件
	ShowNewsInformation.aspx.cs	实现显示新闻信息界面的代码文件
登录模块	BackDesk/ Login.aspx	后台登录界面设计文件
	BackDesk/ Login.aspx.cs	实现后台登录界面的代码文件

19.4.2 系统数据库设计

根据系统的需求分析，我们开始对数据库进行合理的设计。首先在 Sql Server 2005 中建立一个名为“db\_news”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [db_news] //创建数据库
USE [db_news] //使用数据库
```

根据前面的系统需求分析和模块设计，需要的数据表包括：新闻信息表和用户信息表。

(1) 新闻信息表 (tb\_News)，用来记录所有新闻的详细信息，使用新闻编号 ID 作为表的主键。该表的字段结构如表 19-26 所示。

表 19-26 tb\_News 表结构

字段	中文描述	数据类型	是否为空	备注
ID	新闻编号	Int	否	主键
Title	新闻标题	varchar(50)	否	
Content	新闻内容	varchar(2000)	否	
Categories	新闻种类	varchar(50)	否	
Type	新闻标题	varchar(50)	否	
IssueDate	发布日期	datetime	否	

(2) 用户信息表 (tb\_User)，用来记录所有使用本系统用户的详细信息，用户编号 ID 是该表的主键。该表的字段结构如表 19-27 所示。

表 19-27 tb\_User 表结构

字段	中文描述	数据类型	是否为空	备注
ID	用户编号	int	否	主键
Name	用户姓名	varchar(50)	否	
PassWord	用户密码	varchar(50)	否	
Date	用户创建日期	datetime	否	

19.4.3 系统运行示例

系统运行后，出现系统首页，如图 19-21 所示。

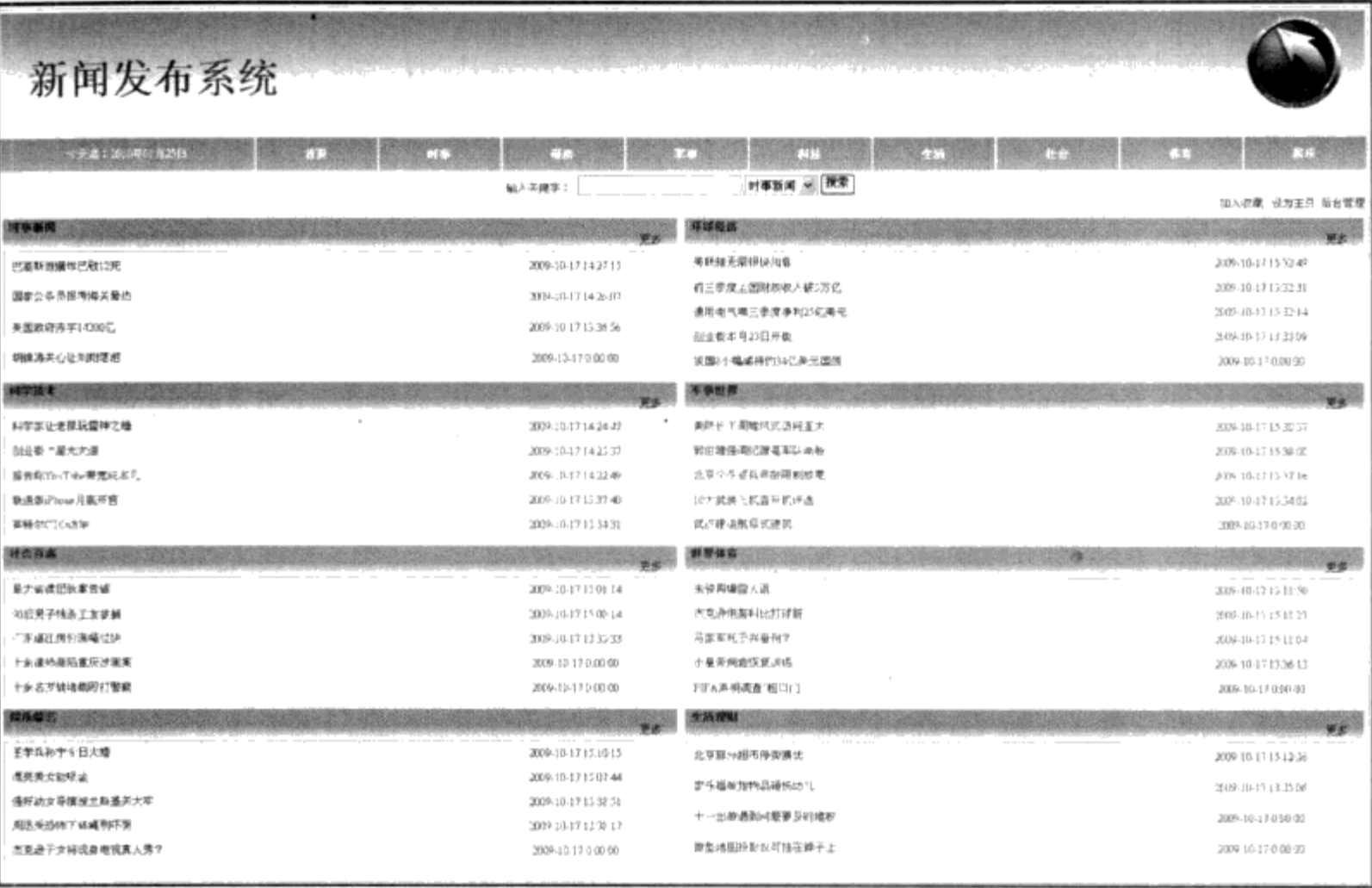


图 19-21 系统首页

在首页中，单击菜单栏中的各种不同的类别的新闻，进入分类新闻页面，如图 19-22 所示。

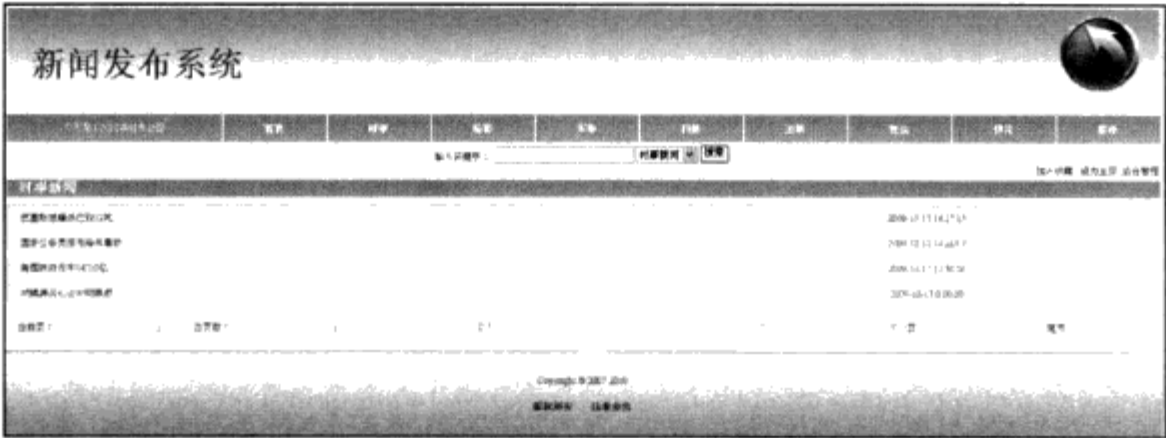


图 19-22 分类新闻显示页面

在页面中，单击表中新闻的标题。进入具体新闻的阅读页面，如图 19-23 所示。

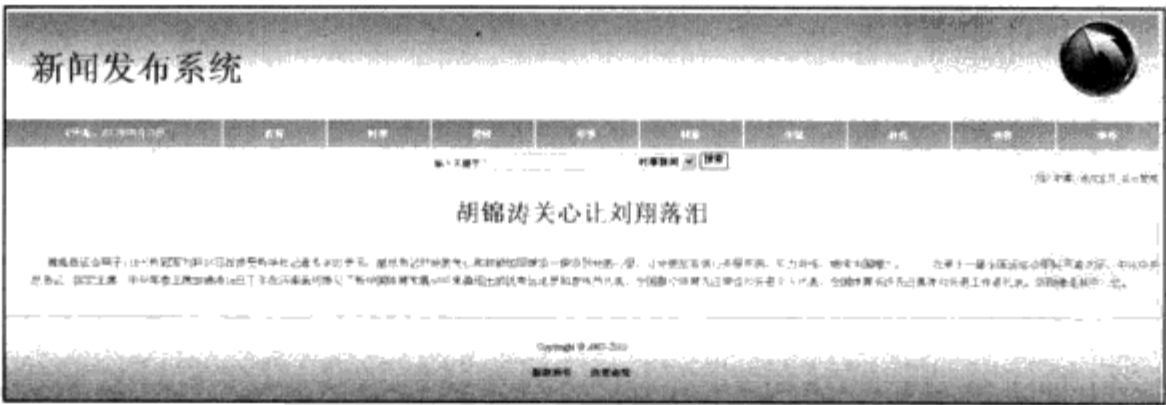


图 19-23 阅读新闻页面

在首页中，单击“后退管理”链接，进入后台登录页面，如图 19-24 所示。



图 19-24 后台登录页面

在页面中，输入用户名、密码和验证码。单击“登录”按钮，通过身份验证后，进入后台管理页面，如图 19-25 所示。



图 19-25 后台管理页面

单击表中的“编辑”按钮，进入新闻修改的界面，如图 19-26 所示。

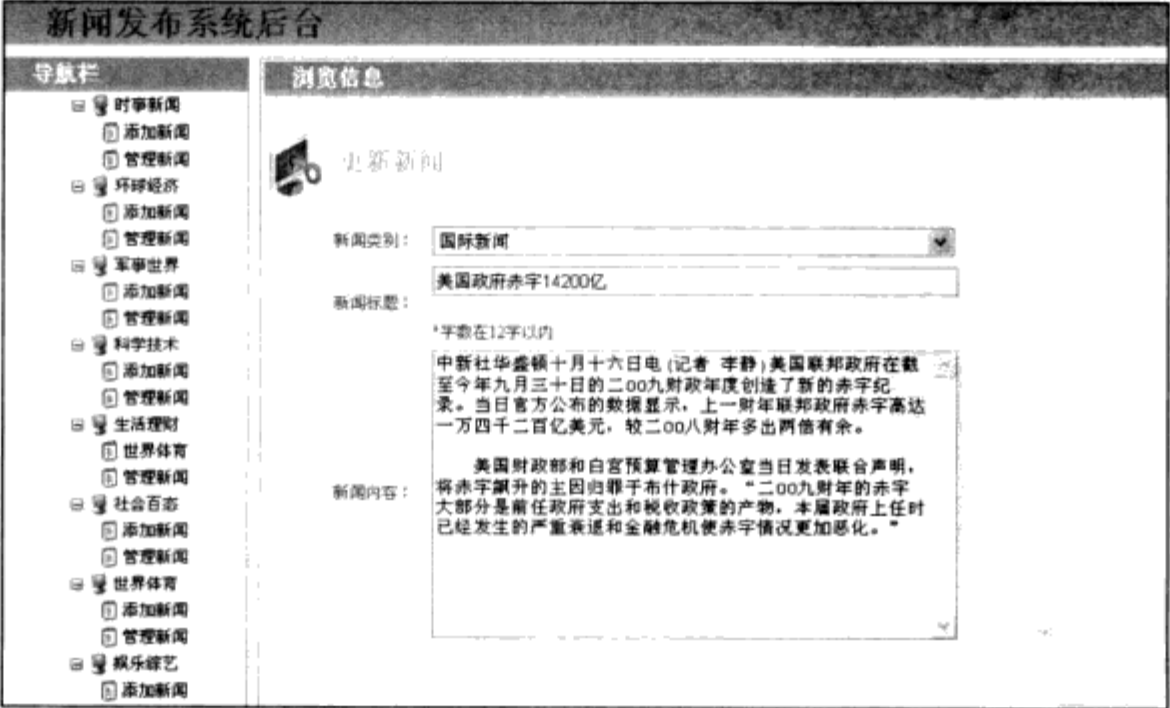


图 19-26 修改新闻页面

在页面中，单击“更新”按钮可以对新闻进行修改。  
在后台管理页面的导航菜单中单击“添加新闻”按钮，可以进入添加新闻页面，如图 19-27 所示。



图 19-27 添加新闻页面

在该页面中，可以对新闻进行添加的操作。  
上面做了系统中主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.5 项目 05：绩效管理系统

在知识经济时代，人力资源是企业最重要的资源，关系到企业的长久发展、企业既定目标的实现。其中，员工的能力是否能得到充分发挥将直接关系到企业的兴衰和成败。绩效管理作为人力资源的核心，是挖掘员工潜力的利器。绩效管理系统不仅是一个简单的评估工具，它还能在帮助企业实现战略目标分解与落实。它是将目标管理和绩效考核相结合的管理系统。本节介绍的是一个具备基本功能的绩效管理系统。

### 19.5.1 系统分析与设计

本系统设计的目的是：规范企业对员工日常工作的管理，实现对企业所有员工工作效率的控制。

#### 1. 系统需求分析

本系统的用户主要有三类，一类是系统管理人员，一类是普通员工，还有一类是部门经理。

- ① 对于系统管理员来说，从登录页面进入系统，在登录页面输入用户名和密码，通过身份验证后，才可以进入系统的首页。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ② 通过身份验证的系统管理员进入操作的首页。在首页中可以通过员工编号、员工姓名和所在部门对员工的信息进行查询。可以选择相应的员工来更新该员工的信息。同时，还可以对员工进行单一的或批量的删除操作。

③ 系统管理员可以对部门的信息和部门管理员的信息进行管理。包括对部门信息和部门管理人的信息进行添加和删除操作。

④ 系统管理员通过员工姓名和上传文件的时间对员工上传的文件进行查询。可以下载选择的文件，也可以对员工上传的文件进行删除。

⑤ 系统管理员还可以修改自己的登录密码。

⑥ 考勤管理员也可以随时修改自己的登录密码。

⑦ 普通员工要进入操作界面之前，也必须从登录页面进入系统，在登录页面输入用户名和密码，通过身份验证后，才可以进入操作页面。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。

⑧ 员工在操作界面可以填写当天的工作日志。还可以通过日志提交的时间，查询自己的历史日志记录和删除日志记录。

⑨ 员工可以查询当天安排的工作任务并根据任务的主题和时间查询历史的工作任务。

⑩ 员工能够上传自己的文件，并且可以通过上传时间查询到上传的文件进行下载。

⑪ 员工也可以修改自己的登录密码。

⑫ 部门管理人必须从登录页面进入操作页面。在该页面可以查询部门员工的日志内容、填写日志信息。

⑬ 部门管理人能够根据任务主题和任务的时间，查询历史的工作任务和当天的工作任务。

⑭ 部门管理人也可以查询所有上传的文件。同时，能够上传自己的文件。

⑮ 部门管理员可以修改自己的登录密码。

## 2. 系统模块设计

根据系统的需求分析，我们对本系统的模块进行划分，系统分为六大模块：数据库管理模块、实体类模块、部门管理模块、用户管理模块、工作日志管理模块和工作任务管理模块。各模块所包含的文件及其功能如表 19-28 所示。

表 19-28 绩效管理系统模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/Logic/ AdminLogic.cs	管理员类业务逻辑代码文件
	App_Code/Logic/ DepartmentLogic.cs	部门类业务逻辑代码文件
	App_Code/Logic/ EducationLogic.cs	学历类业务逻辑代码文件
	App_Code/ Logic/ EmployeeLogic.cs	员工类业务逻辑代码文件
	App_Code/ Logic/ LogLogic.cs	日志类业务逻辑代码文件
	App_Code/ Logic/ ManagerLogic.cs	经理类业务逻辑代码文件
	App_Code/ Logic/ TaskLogic.cs	任务类业务逻辑代码文件
	App_Code/Logic/ UpFileLogic.cs	上传文件类业务逻辑代码文件
	App_Code/Utility/ SQLString.cs	Sql 字符串格式化类代码文件
	App_Code/Utility/ DBOperation.cs	数据库公共操作类代码文件
	App_Code/ Utility/ Common.cs	数据库公共访问类代码文件

(续表)

模块名	文件名	功能描述
实体类模块	App_Code/Model/ AdminModel.cs	管理员实体类代码文件
	App_Code/Model/ DepartmentModel.cs	部门信息实体类代码文件
	App_Code/Model/ EducationModel.cs	学历信息实体类代码文件
	App_Code/Model/ EmployeeModel.cs	员工信息实体类代码文件
	App_Code/Model/LogModel.cs	日志实体类代码文件
	App_Code/Model/ ManagerModel.cs	经理信息实体类代码文件
	App_Code/Model/ TaskModel.cs	任务实体类代码文件
	App_Code/Model/ UpfileModel.cs	上传文件实体类代码文件
部门管理模块	Department/ DepartmentDelete.aspx	删除部门信息界面设计文件
	Department/ DepartmentDelete.aspx.cs	实现删除部门信息界面的代码文件
	Department/ DeppartmentAdd..aspx	添加部门信息界面设计文件
	Department/ DeppartmentAdd.aspx.cs	实现添加部门信息界面设计的文件
	Department/ ManagerCharge.aspx	添加部门经理界面设计文件
	Department/ManagerCharge.aspx.cs	实现添加部门经理界面设计的文件
	Department/ManagerDelete.aspx	删除部门经理界面设计文件
	Department/ ManagerDelete.aspx.cs	实现删除部门经理界面设计的文件
员工管理模块	Employee/File/FileManage.aspx	上传文件管理界面设计文件
	Employee/File/FileManage.aspx.cs	实现上传文件管理界面的代码文件
	Employee/File/FileUpload.aspx.aspx	文件上传界面设计文件
	Employee/File/ FileUpload.aspx.aspx.cs	实现文件上传界面的代码文件
	Employee/ EmployeeInfoAdd.aspx	添加员工信息界面设计文件
	Employee/ EmployeeInfoAdd.aspx.cs	实现添加员工信息界面的代码文件
员工管理模块	Employee/ EmployeeInfoUpdate.aspx	修改员工信息界面设计文件
	Employee/ EmployeeInfoUpdate.aspx.cs	实现修改员工信息界面的代码文件
	Employee/ EmployeeManage.aspx	管理员工界面设计文件
	Employee/ EmployeeManage.aspx.cs	实现管理员工界面的代码文件
	Employee/File/ PersonalFileManage.aspx	查询上传文件界面设计文件
	Employee/File/PersonalFileManage.aspx.cs	实现查询上传文件界面的代码文件
日志管理模块	Log/ LogAdd.aspx	添加日志界面设计文件
	Log/ LogAdd.aspx.cs	实现添加日志界面的代码文件
	Log/ LogInfoUpdate.aspx	日志更新界面设计文件
	Log/ LogInfoUpdate.aspx.cs	实现日志更新界面的代码文件
	Log/ LogManage.aspx	日志管理界面设计文件
	Log/ LogManage.aspx.cs	实现日志管理界面的代码文件

(续表)

模块名	文件名	功能描述
日志管理模块	Log/ PersonalLogInfoUpdate.aspx	更新日志界面设计文件
	Log/ PersonalLogInfoUpdate.aspx.cs	实现更新日志界面的代码文件
	Log/ PersonalLogManage.aspx	查询日志界面设计文件
	Log/ PersonalLogManage.aspx.cs	实现查询日志界面的代码文件
任务管理模块	Task/DayTask.aspx	查询任务界面的代码文件
	Task/DayTask.aspx.cs	实现查询任务界面的代码文件
	Task/TaskAdd.aspx	添加工作任务界面的代码文件
	Task/TaskAdd.aspx.cs	实现添加工作任务界面的代码文件
	Task/TaskInfo.aspx	查询任务详情界面的代码文件
	Task/TaskInfo.aspx.cs	实现查询任务详情界面的代码文件
	Task/TaskInfoUpdate.aspx	更新任务信息界面的代码文件
	Task/TaskInfoUpdate.aspx.cs	实现更新任务信息界面的代码文件
	Task/TaskManage.aspx	查询员工任务界面的代码文件
	Task/TaskManage.aspx.cs	实现查询员工任务界面的代码文件
	Task/TaskQuery.aspx	查询历史任务界面的代码文件
	Task/TaskQuery.aspx.cs	实现查询历史任务界面的代码文件

### 19.5.2 系统数据库设计

根据系统的需求分析，我们开始对数据库进行设计。首先在 Sql Server 2005 中建立一个名为“Efficiency”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [Efficiency] //创建数据库
USE [Efficiency] //使用数据库
```

根据系统的需求分析和模块设计，至少需要的数据表包括：系统管理员表、每日工作任务表、部门信息表、员工信息表、学历类别表、工作日志表、部门经理信息表和上传文件信息表。

(1) 系统管理员表（admin）用来记录使用本系统的管理员信息，使用管理员的用户名 adminUsername 作为表的主键。该表的字段结构如表 19-29 所示。

表 19-29 admin 表结构

字段	中文描述	数据类型	是否为空	备注
adminUsername	管理员用户名	varchar(50)	否	主键
adminPassword	管理员密码	varchar(50)	是	

(2) 每日工作任务表（dayTaskInfo），用来记录所有员工的每天的工作任务信息，任务编号 taskId 是该表的主键。该表的字段结构如表 19-30 所示。

表 19-30 dayTaskInfo 表结构

字段	中文描述	数据类型	是否为空	备注
taskId	任务编号	int	否	主键
EmployeeNo	员工编号	varchar(20)	是	
taskTitle	任务标题	nvarchar(30)	是	
taskContent	任务内容	text	是	
taskDate	任务日期	datetime	是	
taskStatus	任务状态	nchar(1)	是	
taskAddTime	任务添加时间	datetime	是	
taskUpdateTime	任务更新时间	datetime	是	

（3）部门信息表（departmentInfo），用来记录所有部门的详细信息，我们选择部门编号 departmentId 作为主键。该表的字段结构如表 19-31 所示。

表 19-31 departmentInfo 表结构

字段	中文描述	数据类型	是否为空	备注
departmentId	部门编号	int	否	主键
departmentName	部门名称	nvarchar(20)	是	

（4）员工信息表（employeeInfo），用来记录所有员工的信息，员工编号 employeeNo 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-32 所示。

表 19-32 employeeInfo 表结构

字段	中文描述	数据类型	是否为空	备注
employeeNo	员工编号	varchar(20)	否	主键
employeeName	员工姓名	nvarchar(20)	是	
employeePassword	员工密码	varchar(30)	是	
employeeSex	员工性别	nchar(1)	是	
employeeBirthday	员工生日	datetime	是	
employeeDepartmentId	员工所属部门编号	int	是	
employeePositionId	员工职位编号	int	是	
employeeEducationId	员工学历编号	int	是	
employeeHomeTel	家庭电话	varchar(20)	是	
employeeMobile	员工手机	varchar(20)	是	
employeeCard	员工身份证号	varchar(20)	是	
employeeEmail	员工电子邮件	varchar(30)	是	
employeeAddress	员工住址	nvarchar(80)	是	

（5）学历信息表（educationInfo），用来记录所有员工的学历信息，学历信息编号 educationId

是主键。该表的字段结构如表 19-33 所示。

表 19-33 educationInfo 表结构

字段	中文描述	数据类型	是否为空	备注
educationId	学历编号	int	否	主键
educationName	学历名称	nvarchar(20)	否	

(6) 工作日志表 (logInfo)，用来记录员工每日工作日志的详细信息，根据主键唯一性原则，设定日志编号 logId 为主键。该表的字段结构如表 19-34 所示。

表 19-34 logInfo 表结构

字段	中文描述	数据类型	是否为空	备注
logId	日志编号	int	否	主键
employeeNo	员工编号	varchar(20)	是	
logTitle	日志标题	nvarchar(30)	是	
logContent	日志内容	text	是	
logEvaluate	日志评估	varchar(100)	是	
logDate	日志日期	datetime	是	
logAddTime	添加日志时间	datetime	是	
logUpdateTime	日志更新时间	datetime	是	

(7) 部门经理表 (managerInfo)，用于记录每个部门有权限操作本系统的经理信息。我们用经理的用户名 managerUsername 作为该表的主键。该表的字段结构如表 19-35 所示。

表 19-35 managerInfo 表结构

字段	中文描述	数据类型	是否为空	备注
managerUsername	经理用户名	varchar(50)	否	主键
managerName	经理姓名	nvarchar(20)	是	
managerPassword	经理登录密码	varchar(50)	是	
departmentId	部门编号	Int	是	

(8) 上传文件信息表 (upfileInfo)，用于记录系统中用户上传文件的信息。上传文件编号 upfileId 是此表的主键。该表的字段结构如表 19-36 所示。

表 19-36 upfileInfo 表结构

字段	中文描述	数据类型	是否为空	备注
upfileId	上传文件编号	int	否	主键
employeeNo	用户编号	varchar(30)	是	
upfileTitle	上传文件标题	nvarchar(50)	是	
upfilePath	上传文件路径	varchar(50)	是	
upfileSize	上传文件的路径	varchar(50)	是	
uploadTime	上传时间	datetime	是	

### 19.5.3 系统运行示例

我们运行系统后，出现登录页面，如图 19-28 所示。



图 19-28 用户登录页面

在该页面中，输入用户名、密码和选择身份，单击“登录”按钮，通过验证进入系统首页，如图 19-29 所示。

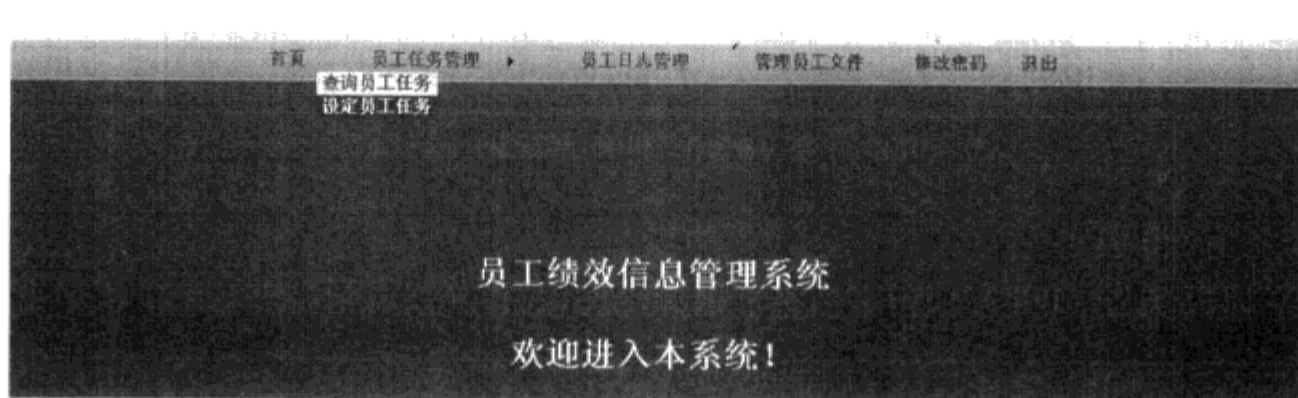


图 19-29 系统首页

在首页中，我们将鼠标放到菜单栏的“员工任务管理”菜单上，在弹出的二级菜单中选择“设定员工任务”，进入员工设定任务的页面，如图 19-30 所示。

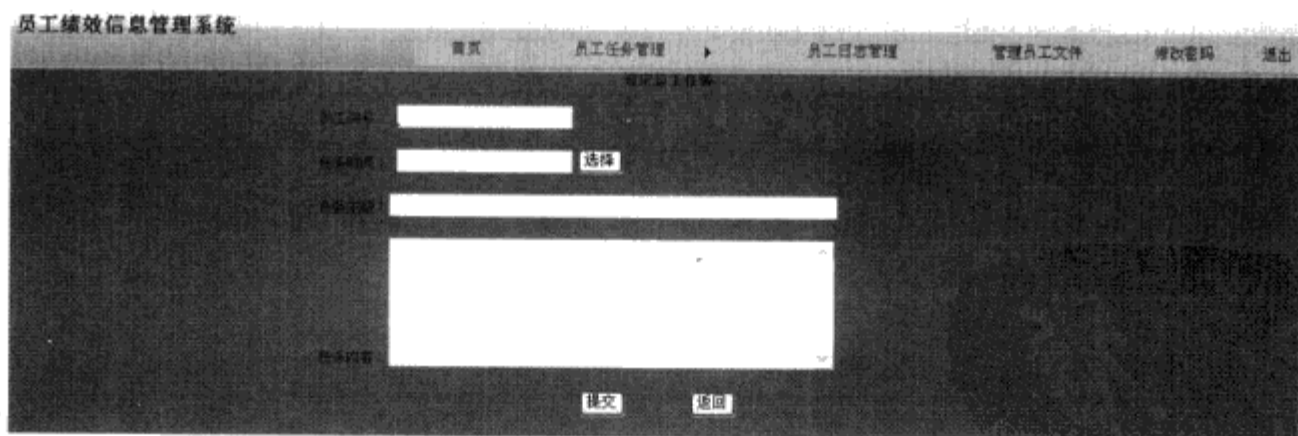


图 19-30 设定员工任务

在该页面中，部门管理员填写员工编号、任务时间、任务主题和任务内容后，单击“提交”按钮，完成员工任务设置。当我们将鼠标放到菜单栏的“员工任务管理”菜单上，在弹出的二级菜单中选择“查询员工任务”，进入员工任务查询页面，如图 19-31 所示。

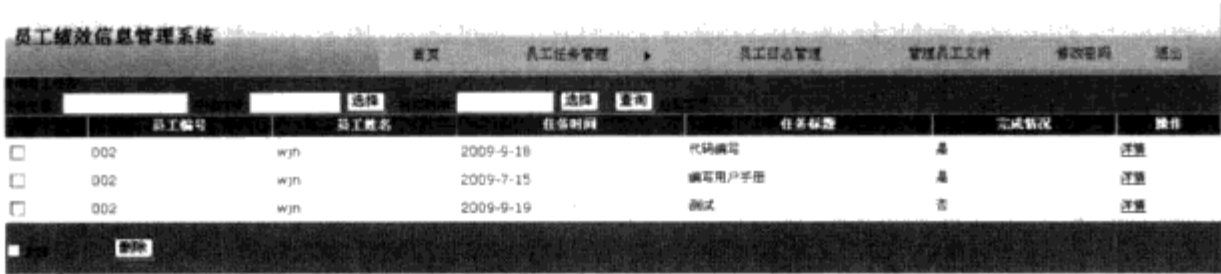


图 19-31 查询员工任务

在该页面中，输入任务编号、任务时间可以查询出所需要的任务信息，并在页面下方的列表中显示。同时，可以单选或全选后对工作任务进行删除操作。单击工作任务列表中的“详情”链接，能够进入修改任务页面，如图 19-32 所示。

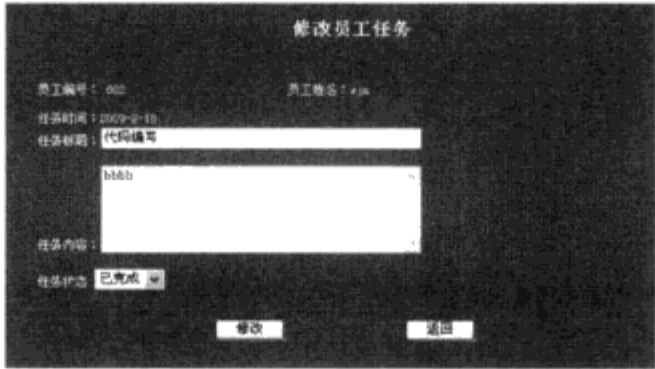


图 19-32 修改任务页面

在该页面修改任务标题、任务内容和任务状态后，单击“修改”按钮，完成任务的修改操作。单击页面菜单栏上的“管理员工文件”，进入员工文件下载页面，如图 19-33 所示。

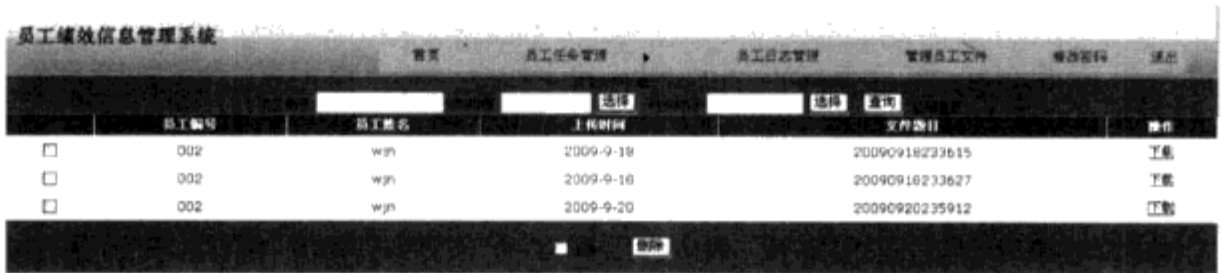


图 19-33 员工文件下载页面

我们在该页面中，输入员工编号、文件上传的时间可以查询出所需要的文件信息并在页面下方的列表中显示。同时，可以单选或全选后对员工文件进行删除操作。单击员工文件列表中的“下载”链接，会弹出文件下载对话框，如图 19-34 所示。

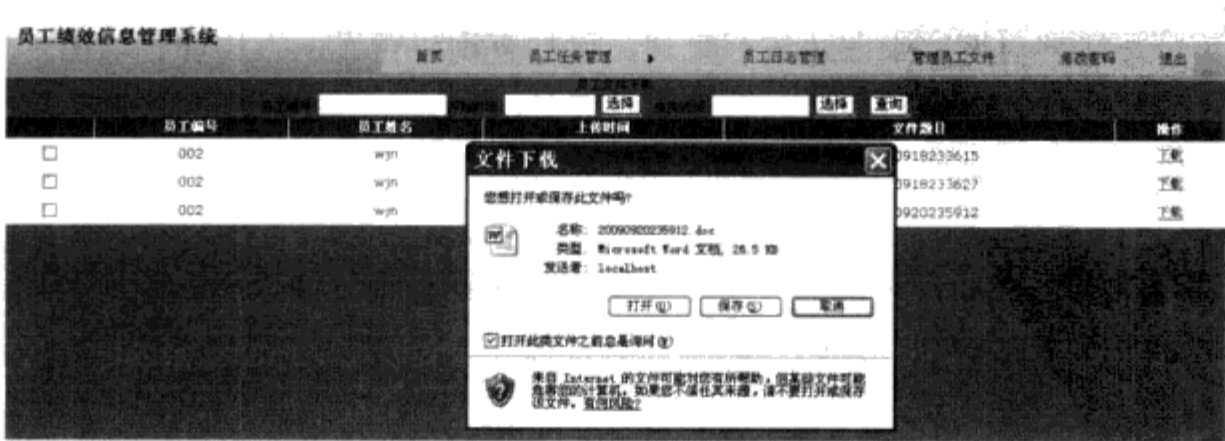


图 19-34 文件下载对话框

单击“保存”按钮，选择保存文件的路径，单击“确定”按钮，完成下载的操作。

上面做了系统中主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.6 项目 06：博客管理系统

博客一词是从英文单词 Blog 翻译而来。Blog 是 Weblog 的简称，而 Weblog 则是由“Web”和“Log”两个英文单词组合而成。Weblog 就是在网络上发布和阅读的流水记录，通常称为网络日志，简称为网志。它是继 Email、BBS、IM 之后出现的第四种全新的网络交流方式。它绝不仅仅是一种单向的发布系统，而是以网络作为载体，迅速、便捷地发布自己的心得，及时有效地与他人进行交流，再集丰富多彩的个性化展示于一体的综合性平台。本节介绍的就是一个具备基本功能的网上博客管理系统。

### 19.6.1 系统分析与设计

本系统设计的目标是：使用方便，操作简单，效率较高，实现对用户发表文章，发表评论，发表留言，上传照片，修改个人信息等功能。

#### 1. 系统需求分析

本系统的用户包括普通游客、博客用户和系统管理员。系统需要实现的需求说明如下。

- ① 普通游客进入网站后可以浏览网站。浏览文章、发表阅读文章后的感想和给博客的主人留言。
- ② 普通游客可以通过注册，成为博客用户。在网站上建立自己的博客并管理自己的博客。
- ③ 博客用户必须在网站首页输入用户名、密码和验证码，通过身份验证后，才可以进入个人的博客管理界面。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ④ 在个人博客管理界面，博客用户可以对自己的文章类型进行管理，包括添加、查询、修改和删除自己的文章类型。
- ⑤ 博客用户能够对网友的留言进行管理，包括删除和回复操作。
- ⑥ 博客用户可以对自己的文章进行管理，包括添加、查询、修改和删除自己文章内容。
- ⑦ 博客用户能够管理网友回复的信息，包括单个或批量删除网友的回复信息。
- ⑧ 博客用户可以对留言的网友进行管理，包括添加和查找网友的操作。
- ⑨ 博客用户还能够对博客中图片进行管理，包括查询、查看、上传和删除图片信息。
- ⑩ 系统管理员必须在网站首页进入管理员登录界面，在该界面中输入用户名、密码和验证码，通过身份验证后，才可以进入后台管理界面。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ⑪ 系统管理员可以在后台对博客用户进行管理，包括查询、修改和删除博客用户。
- ⑫ 可以对管理员进行添加、修改、删除和查询操作。

#### 2. 系统模块设计

根据上面的系统需求分析，我们对本系统的模块进行划分。我们将系统分为 4 大模块：数据

库管理模块、前台显示模块、后台系统管理模块和博客管理模块。各模块所包含的文件及其功能如表 19-37 所示。

表 19-37 宿舍管理模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/SqlData.cs	数据库公共访问类代码文件
前台显示模块	BlogIndex.aspx	显示文章界面设计的代码文件
	BlogIndex.aspx.cs	实现显示文章界面的代码文件
	showMainPage.aspx	系统首页界面设计的代码文件
	showMainPage.aspx.cs	实现系统首页界面的代码文件
	showMessage.aspx	显示留言界面设计的代码文件
	showMessage.aspx.cs	实现显示留言界面的代码文件
后台系统管理模块	AdminInfo.aspx	修改管理员信息界面设计的代码文件
	AdminInfo.aspx.cs	实现修改管理员信息界面的代码文件
	AdminManage.aspx	查询管理员信息界面设计的代码文件
	AdminManage.aspx.cs	实现查询管理员信息界面的代码文件
	BlogManage.aspx	用户管理界面设计的代码文件
	BlogManage.aspx.cs	实现用户管理界面的代码文件
	Index.aspx	管理员登陆界面设计的代码文件
	Index.aspx.cs	实现管理员登陆界面的代码文件
	Register.aspx	注册管理员界面设计的代码文件
	Register.aspx.cs	实现注册管理员界面的代码文件
	BlogInfo.aspx	修改用户信息界面设计的代码文件
	BlogInfo.aspx.cs	实现修改用户信息界面的文件
博客管理模块	AddArt.aspx	添加文章界面设计的代码文件
	AddArt.aspx.cs	实现添加文章界面的代码文件
	AddLink.aspx	添加友情链接界面设计的代码文件
	AddLink.aspx.cs	实现添加友情链接界面的代码文件
	AddLinkMan.aspx	添加留言人界面设计的代码文件
	AddLinkMan.aspx.cs	实现添加留言人界面的代码文件
	AddPhoto.aspx	添加图片界面设计的代码文件
	AddPhoto.aspx.cs	实现添加图片界面的代码文件
	admin_message.aspx	留言信息管理界面设计的代码文件
	admin_message.aspx.cs	实现留言信息管理界面界面的代码文件
	admin_replay.aspx	回复信息管理界面设计的代码文件
	admin_replay.aspx.cs	实现回复信息管理界面的代码文件
	AmendLink.aspx	修改友情链接界面设计的代码文件

(续表)

模块名	文件名	功能描述
博客管理模块	AmendLink.aspx.cs	实现修改友情链接界面界面的代码文件
	ArticleManage.aspx	文章管理界面设计的代码文件
	ArticleManage.aspx.cs	实现文章管理界面的代码文件
	BlogType.aspx	文章类型管理界面设计的代码文件
	BlogType.aspx.cs	实现文章类型管理界面的代码文件
	LinkManage.aspx	友情链接管理界面设计的代码文件
	LinkManage.aspx.cs	实现友情链接管理界面的代码文件
	LinkManInfo.aspx	修改留言人信息界面设计的代码文件
	LinkManInfo.aspx.cs	实现修改留言人信息界面的代码文件
	LinkManManage.aspx	管理留言人界面设计的代码文件
	LinkManManage.aspx.cs	实现管理留言人界面的代码文件
	PhotoManage.aspx	图片管理界面设计的代码文件
	PhotoManage.aspx.cs	实现图片管理界面的代码文件
	Register.aspx	用户注册界面设计的代码文件
	Register.aspx.cs	实现用户注册界面的代码文件
	ViewContent.aspx	查看文章界面设计的代码文件
	ViewContent.aspx.cs	实现查看文章界面的代码文件

19.6.2 系统数据库设计

根据系统的需求分析，我们对数据库进行设计。首先，在 Sql Server 2005 中建立一个名为“db\_Blog”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [db_Blog] //创建数据库
USE [db_Blog] //使用数据库
```

根据系统的需求分析和模块设计，所需要的数据表包括：系统管理员表、用户信息表、文章种类表、友情链接表、留言信息表、文章信息表、图片信息表、回复信息表和留言人信息表。

(1) 系统管理员表（Admin），用来记录使用本系统的管理员的基本信息，使用管理员的编号 ID 作为表的主键。该表的字段结构如表 19-38 所示。

表 19-38 Admin 表结构

字段	中文描述	数据类型	是否为空	备注
ID	管理员用户名	int	否	主键
UserName	用户名	nvarchar(50)	是	
PassWord	密码	nvarchar(50)	是	
Question	取回密码问题	nvarchar(100)	是	
Answer	问题回答	nvarchar(100)	是	
ReallyName	真实姓名	nvarchar(50)	是	
Birthday	生日	nvarchar(50)	是	

(续表)

字段	中文描述	数据类型	是否为空	备注
Address	家庭住址	nvarchar(100)	是	
PostCode	邮政编码	nvarchar(50)	是	
Email	电子邮件	nvarchar(50)	是	
HomePhone	家庭电话	nvarchar(50)	是	
MobilePhone	手机号码	nvarchar(50)	是	
QQ	QQ 号码	nvarchar(50)	是	
RegTime	注册时间	datetime	是	
Sex	性别	nvarchar(4)	是	
IP	IP 地址	nvarchar(20)	是	
BlogID	博客编号	int	是	
SuperAdmin	是否是超级管理员	nvarchar(4)	是	

(2) 文章种类表 (category)，用来记录博客中文章种类的详细信息，种类编号 c\_id 是该表的主键。该表的字段结构如表 19-39 所示。

表 19-39 category 表结构

字段	中文描述	数据类型	是否为空	备注
c_id	种类编号	bigint	否	主键
c_name	种类名称	nvarchar(50)	是	
BlogID	用户编号	int	是	

(3) 友情链接表 (link)，用来记录友情链接的信息，选择链接编号 l\_id 作为主键。该表的字段结构如表 19-40 所示。

表 19-40 link 表结构

字段	中文描述	数据类型	是否为空	备注
l_id	链接编号	bigint	否	主键
l_name	链接名称	nvarchar(50)	是	
l_url	链接地址	nvarchar(50)	是	

(4) 留言信息表 (message)，用来记录博客中用户留言的信息，留言编号 id 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-41 所示。

表 19-41 message 表结构

字段	中文描述	数据类型	是否为空	备注
id	留言编号	bigint	否	主键
nickname	留言人昵称	nvarchar(50)	是	
title	留言标题	nvarchar(50)	是	
content	留言内容	ntext	是	
mdate	发表日期	datetime	是	

（5）文章信息表（news），用来记录文章的详细信息，文章编号 n\_id 是主键。该表的字段结构如表 19-42 所示。

表 19-42 news 表结构

字段	中文描述	数据类型	是否为空	备注
n_id	文章编号	int	否	主键
n_author	文章作者	nvarchar(50)	是	
n_title	文章标题	nvarchar(200)	是	
n_key	文章摘要	nvarchar(200)	是	
n_content	文章内容	ntext	是	
n_date	发表日期	datetime	是	
n_hit	单击次数	bigint	是	
c_id	文章类型编号	bigint	是	外键
c_name	类型名称	nvarchar(50)	是	
n_iscmd	是否推荐	int	是	
BlogID	用户编号	int	是	

（6）图片信息表（Picture），用来记录所有图片的信息，图片编号 PictureID 是主键。该表的字段结构如表 19-43 所示。

表 19-43 Picture 表结构

字段	中文描述	数据类型	是否为空	备注
PictureID	图片编号	int	否	主键
ImageUrl	图片存放地址	nvarchar(20)	是	
Subject	图片主题	nvarchar(50)	是	
BlogID	用户编号	int	是	

（7）用户信息表（Blog），用来记录发表博客的用户信息，用户编号 BlogID 是主键。该表的字段和结构与系统管理员表（Admin）几乎一致，请读者参考系统管理员表。

（8）留言人信息表（tb\_Message），用来记录博客中留言人的详细信息，留言人编号 MessageID 是主键。该表的字段和结构与用户信息表（Blog）的十分相似，读者可以参考用户信息表。

19.6.3 系统运行示例

运行系统后，最先进入的是系统首页，如图 19-35 所示。



图 19-35 系统首页

在该页面中，单击文章的标题或“点击阅览”链接，进入文章查看页面，如图 19-36 所示。

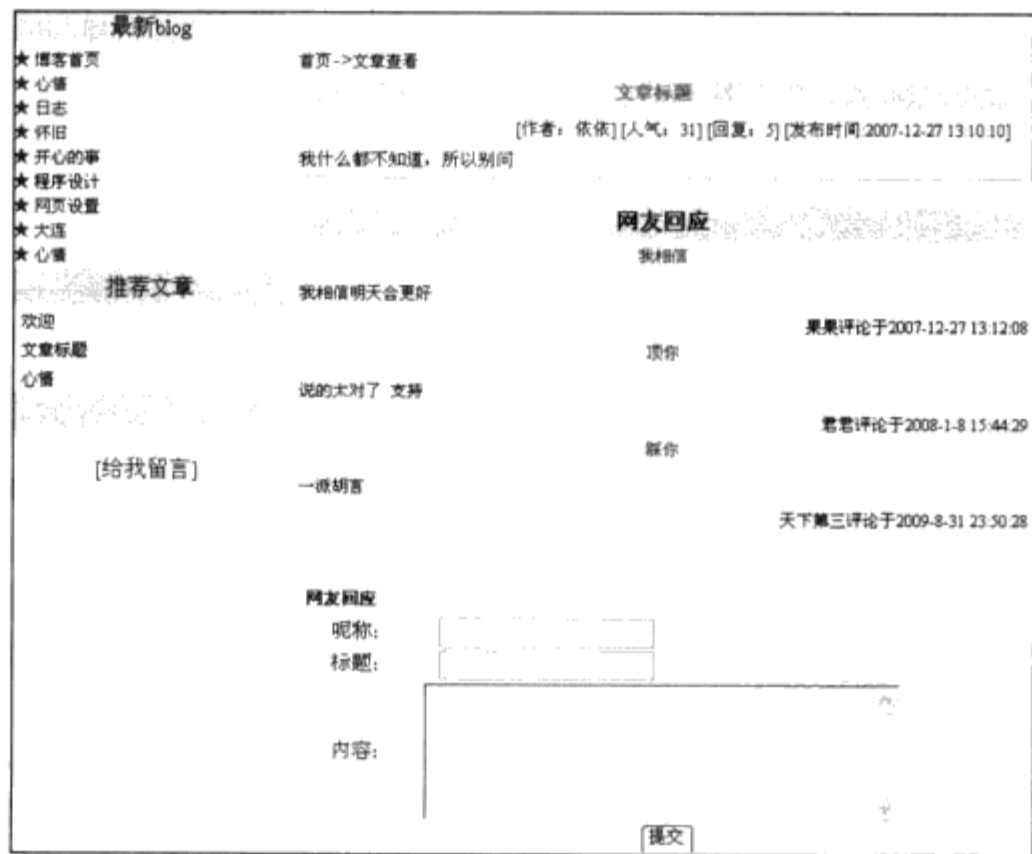


图 19-36 查看文章页面

在该页面中，可以浏览文章的同时，能够填写昵称、标题和内容单击“提交”按钮，对文章进行评论回应。

博客用户在首页，输入用户名、密码和验证码后，单击“登录”按钮。可进入博客管理界面，如图 19-37 所示。

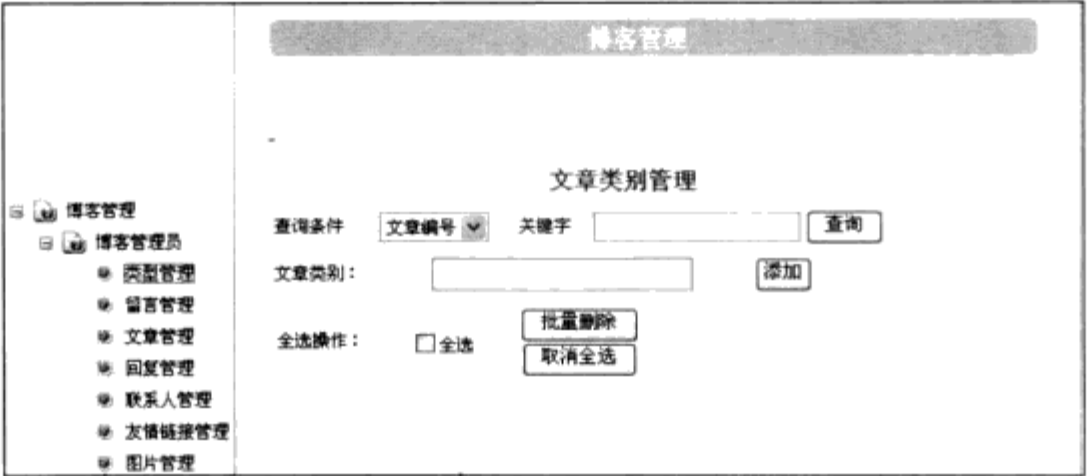


图 19-37 管理博客界面

在左面的树形菜单中单击“文章管理”链接，进入“文章管理”页面，如图 19-38 所示。



图 19-38 文章信息管理页面

用户在该页面中，可以通过文章编号查询文章。单选或多选文章进行删除。同时，当用户单击“添加新文章”链接后。进入文章修改界面，如图 19-39 所示。

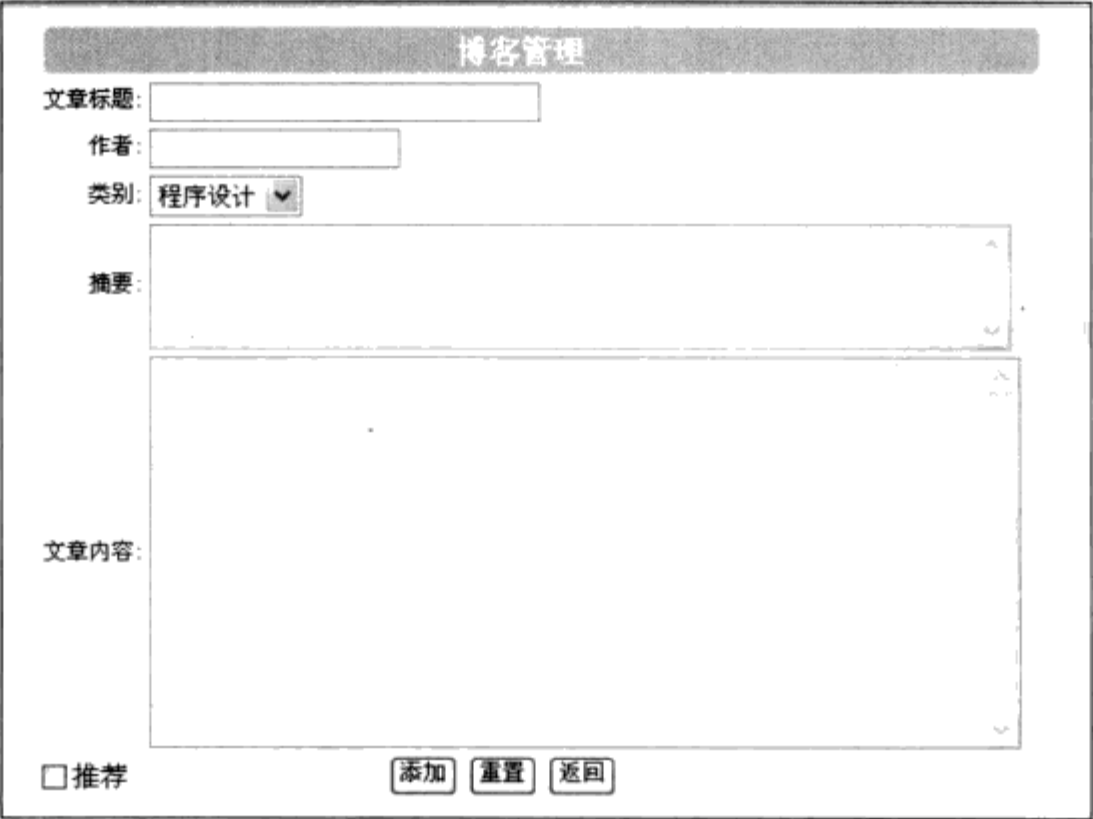


图 19-39 修改文章页面

用户在该页面，添加修改的各项内容后，单击“添加”按钮，完成添加的操作。博客管理员在首页中，单击“管理员”链接，进入管理员登录页面，如图 19-40 所示。



图 19-40 管理员登录界面

管理员在页面中输入用户名、密码和验证码，单击“确定”按钮进入后台管理页面，如图 19-41 所示。

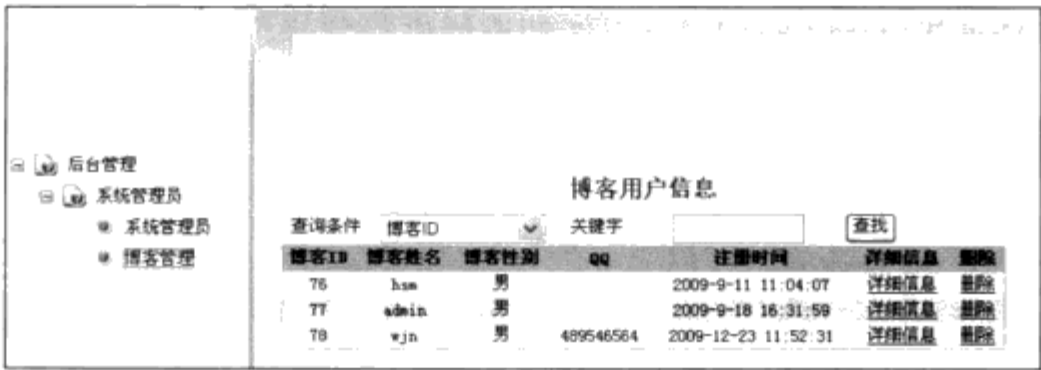


图 19-41 后台管理页面

管理员在该页面中能够进行对博客和管理员的进行查询和删除操作。  
上面做了系统中主要页面的演示，其他页面和功能基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.7 项目 07：医院管理系统

为了适应社会的发展、提高医院服务水平和工作效率，更好地服务于病患，我国的医院已经进入数字化和信息化的时代。以医疗费用为中心的医疗体制改革的开展，促使医院正在由二元化向多元化发展。医院管理系统的出现将提高医院各项工作的效率和质量，促进和改善医院的经营管理，保证病人和医院的经济利益。本节介绍的医院管理系统可以进一步加强医院管理，提高医院工作的效率和质量。

### 19.7.1 系统分析与设计

系统设计上采用的 B/S 结构，支持 SQL Server 数据库。系统的安全性较高，操作简便、快速、界面通俗易懂，是帮助传统的医院管理向数字化医院管理转变的有力工具。

#### 1. 系统需求分析

本系统的使用者根据自己所拥有的权限进行相应的操作，没有权限的用户无法进入相应的操作页面。

- ① 操作人员从登录界面进入系统，在登录页面输入用户名、密码和验证码，通过身份验证后，才可以进入系统的首页。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ② 通过身份验证的操作人员进入首页后，可以通过病人的姓名和挂号的时间，查询挂号信息。
- ③ 在挂号登记页面能够进行对病人的挂号登记。
- ④ 操作人员可以添加新的药品。并通过药品名称查询药品的信息。同时，能够修改原有的药品的详细信息。
- ⑤ 操作人员能够对病人的住院信息进行登记。并根据病人姓名、病房号和住院状态对病人的住院信息进行查询。对还未出院的病人可以进行出院登记和缴付住院费操作。
- ⑥ 操作人员在系统管理的菜单中，还能够对用户的操作权限进行设定。同时，可以修改自己的登录密码。

2. 系统模块设计

根据上面的系统需求分析，我们对本系统的模块进行划分，可以分为六大模块：数据库管理模块、实体类模块、住院管理模块、药品管理模块、挂号管理模块和用户管理模块。各模块所包含的文件及其功能如表 19-44 所示。

表 19-44 医院管理模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/Logic/ AdminLogic.cs	管理员类业务逻辑代码文件
	App_Code/Logic/ InhospitalLogic.cs	住院类业务逻辑代码文件
	App_Code/Logic/ MedicineLogic.cs	药品类业务逻辑代码文件
	App_Code/ Logic/ RegisteredLogic.cs	挂号类业务逻辑代码文件
	App_Code/ Logic/ SubjectLogic.cs	科室类业务逻辑代码文件
	App_Code/ DB / SQLString.cs	Sql 字符串格式化类代码文件
	App_Code/DB/ DataBase.cs	数据库公共访问类代码文件
实体类模块	App_Code/Model/ AdminModel.cs	管理员实体类代码文件
	App_Code/Model/ InhospitalModel.cs	住院信息实体类代码文件
	App_Code/Model/ MedicineModel.cs	药品信息实体类代码文件
	App_Code/Model/ RegisteredModel.cs	挂号登记信息实体类代码文件
	App_Code/Model/ SubjectModel.cs	科室信息实体类代码文件
住院管理模块	PatientAdd.aspx	住院登记界面设计文件
	PatientAdd.aspx.cs	实现住院登记界面的代码文件
	PatientQuery.aspx	管理住院信息界面设计文件
	PatientQuery.aspx.cs	实现管理住院信息界面设计的文件
	InhospitalInfoUpdate.aspx	出院登记界面设计文件
	InhospitalInfoUpdate.aspx.cs	实现出院登记界面设计的文件
药品管理模块	MedicineAdd.aspx	添加药品界面设计文件
	MedicineAdd.aspx.cs	实现添加药品界面的代码文件
	MedicineDetails.aspx	修改药品信息界面设计文件
	MedicineDetails.aspx.cs	实现修改药品信息界面的代码文件

(续表)

模块名	文件名	功能描述
药品管理模块	MedicineManage.aspx	药品信息管理界面设计文件
	MedicineManage.aspx.cs	实现药品信息管理界面的代码文件
挂号管理模块	RegisteredAdd.aspx	挂号登记界面设计文件
	RegisteredAdd.aspx.cs	实现挂号登记界面的代码文件
	RegisteredQuery.aspx	查询挂号信息界面设计文件
	RegisteredQuery.aspx.cs	实现查询挂号信息界面的代码文件
用户管理模块	UserDetails.aspx	用户权限设置界面的代码文件
	UserDetails.aspx.cs	实现用户权限设置界面的代码文件
	UserManage.aspx	用户管理界面的代码文件
	UserManage.aspx.cs	实现用户管理界面的代码文件
	login.aspx	登录界面的代码文件
	login.aspx.cs	实现登录界面的代码文件
	ChangePassword.aspx	更新密码界面的代码文件
	ChangePassword.aspx.cs	实现更新密码界面的代码文件

19.7.2 系统数据库设计

根据系统的需求分析，我们开始对数据库进行设计。首先在 Sql Server 2005 中建立一个名为“HospitalManage”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [HospitalManage] //创建数据库
USE [HospitalManage] //使用数据库
```

根据系统的需求分析和模块设计，所需要的数据表包括：系统管理员表、住院信息表、药品信息表、挂号登记信息表和医院科室表。

(1) 系统管理员表（admin），用来记录使用本系统的管理员基本信息和对系统所拥有的权限信息，使用管理员的用户名 username 作为表的主键。该表的字段结构如表 19-45 所示。

表 19-45 admin 表结构

字段	中文描述	数据类型	是否为空	备注
username	管理员用户名	varchar(20)	否	主键
password	管理员密码	varchar(32)	是	
registeredAdd	挂号登记权限	int	是	
registeredQuery	挂号查询权限	int	是	
patientAdd	住院登记权限	int	是	
patientManage	住院管理权限	int	是	
medicineAdd	添加药品权限	int	是	
medicineManage	管理药品权限	int	是	
userManage	管理用户权限	int	是	

(2) 住院信息表（inhospital），用来记录医院住院病人的信息，id 住院信息编号是该表的主键。该表的字段结构如表 19-46 所示。

表 19-46 inhospital 表结构

字段	中文描述	数据类型	是否为空	备注
id	住院信息编号	int	否	主键
name	病人姓名	varchar(50)	是	
sex	病人性别	varchar(5)	是	
age	病人年龄	int	是	
roomNo	病房号	varchar(50)	是	
arrivelTime	入院时间	datetime	是	
isLeave	是否出院	int	是	
leaveTime	出院时间	datetime	是	
money	住院费	float	是	
arrivelOperator	入院操作人	varchar(50)	是	
leaveOperator	出院操作人	varchar(50)	是	

(3) 药品信息表 (medicine)，用来记录医院药品的详细信息，我们选择药品编号 medicineId 作为主键。该表的字段结构如表 19-47 所示。

表 19-47 medicine 表结构

字段	中文描述	数据类型	是否为空	备注
medicineId	药品编号	int	否	主键
medicineName	药品名称	nvarchar(30)	是	
price	药品单价	float	是	
count	药品数量	int	是	
unit	药品单位	nvarchar(5)	是	
approvalNumber	批文编号	nvarchar(50)	是	
ingredient	药品成分	nvarchar(50)	是	
efficacy	药品功效	nvarchar(100)	是	
usage	服用方法	nvarchar(50)	是	

(4) 挂号登记信息表 (registered)，用来记录所有病人的挂号登记的信息，挂号登记编号 id 具有唯一性，所以可以设置成为主键。该表的字段结构如表 19-48 所示。

表 19-48 registered 表结构

字段	中文描述	数据类型	是否为空	备注
id	挂号登记编号	int	否	主键
name	病人姓名	nvarchar(10)	是	
sex	病人性别	nchar(1)	是	
age	病人年龄	int	是	
subjectId	挂号科	int	是	
operateTime	挂号时间	datetime	是	
operator	操作人	varchar(20)	是	

（5）医院科室表（subject），用来记录医院所有看病的科室信息，科室编号 subjectId 是主键。该表的字段结构如表 19-49 所示。

表 19-49 subject 表结构

字段	中文描述	数据类型	是否为空	备注
subjectId	科室编号	int	否	主键
subjectName	科室名称	nvarchar(50)	是	
registeredMoney	挂号金额	float	是	

19.7.3 系统运行示例

本系统运行后，首先出现的是登录页面，如图 19-42 所示。

用户在该页面中，输入用户名、密码和验证码，单击“登录”按钮，通过验证进入系统首页，如图 19-43 所示。

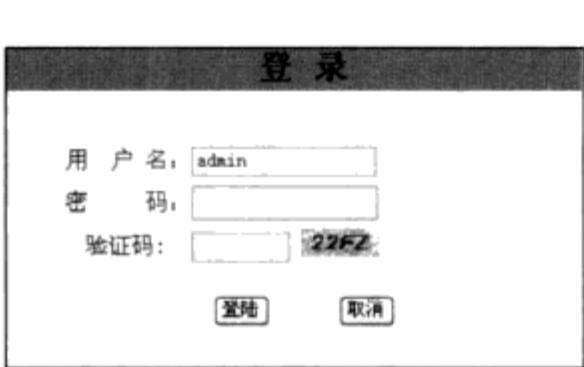


图 19-42 用户登录页面

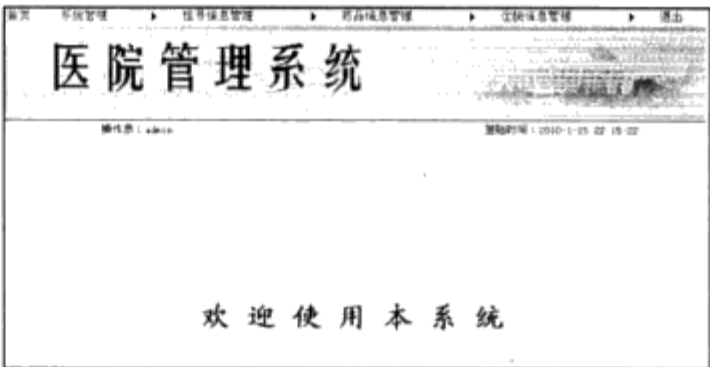


图 19-43 系统首页

在首页中，将鼠标放到菜单栏的“挂号信息管理”上，选择弹出的二级菜单中的“挂号信息登记”，进入挂号信息登记页面，如图 19-44 所示。



图 19-44 挂号信息登记页面

在该页面中输入姓名、性别、年龄、挂号科室和挂号费等信息，单击“挂号”按钮，完成挂号登记操作。

在首页中，用户将鼠标放到菜单栏的“挂号信息管理”上，选择弹出的二级菜单中的“查询挂号信息”，进入查询挂号信息页面，如图 19-45 所示。



图 19-45 查询挂号信息页面

在该页面中输入姓名、挂号时间的条件能够查询出相应的挂号信息。

用户在首页中，将鼠标放到菜单栏的“系统管理”上，选择弹出的二级菜单中的“用户权限管理”，进入用户列表页面，如图 19-46 所示。



图 19-46 用户列表页面

在用户列表页面中，用户单击“设置权限”链接，进入权限设置的页面，如图 19-47 所示。

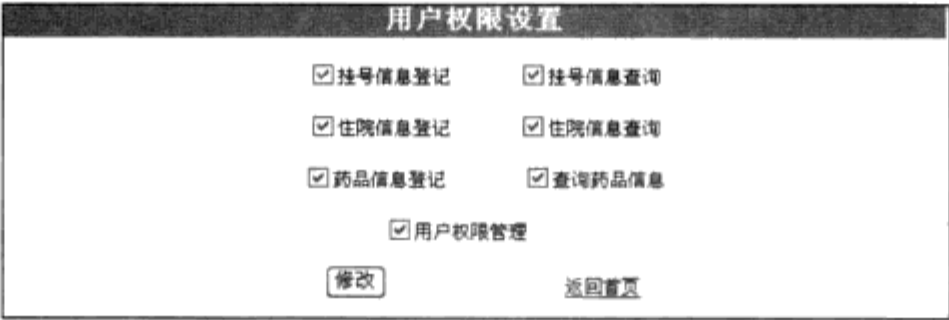


图 19-47 设置权限页面

用户在该页面，可以通过选择各种不同权限，对用户的权限进行设置和修改。

上面做了系统中主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.8 项目 08：仓库管理系统

目前，随着市场经济发展不断的深入，企业间形成了激烈的甚至是残酷的多元化竞争。当今的企业要想在这样的市场环境中生存并立于不败之地，就必须降低企业生产过程中的各种成本。而企业的采购、库存等环节是企业成本控制的关键。依靠传统的方法对这些环节进行管理，不能使企业的采购、库存等环节的信息数据得到及时沟通和适时共享，以至于造成采购过量和库存积压等现象，不仅降低了效率，大大增加企业不必要的成本开支。因此，使用一个功能齐全、操作方便的仓库管理系统已经是企业的一个必然的选择。

### 19.8.1 系统分析与设计

仓库管理系统的开发目的是提高仓库的使用效率，减少仓库管理的缺失和遗漏。具有对仓库信息、产品信息和相关经手人以及供货单位和收货单位进行管理和维护的功能。

#### 1. 系统需求分析

根据系统开发的目的，需要实现的用户需求总述如下。

- ① 系统操作人员从登录界面进入系统，在登录页面输入用户名和密码，通过身份验证后，方可进入系统的首页。如果未能通过系统的身份验证，系统自动给出无权操作的警告。
- ② 通过身份验证的操作人员进入系统的首页。在首页中可以进行各项仓库管理模块的操作。
- ③ 首先，可以对仓库中的产品进行管理。当有新的产品出现时添加新的产品。同时，可以对原来的产品信息进行修改。当一种产品从仓库中去除时，可以对该产品进行删除信息的操作。
- ④ 当采购人员采购来新的产品时，可对产品进行登记入库的工作，当操作中出现失误时，能够对录入的产品入库信息加以修改或者删除，并重新进行输入。在日常的入库管理工作中可以通过关键字对入库的产品进行查询，并且可以实现对入库产品的年度统计的功能。
- ⑤ 有了对产品的入库管理，相对应的必然会有对产品的出库管理。当产品获得销售从仓库中被提出时，必须及时对产品进行出库管理。其中，包括对出库产品进行登记、修改和删除。与入库产品管理一样，能够对出库产品的信息进行查询和年度统计。
- ⑥ 为了更好地对仓库中的产品进行管理，定期盘点库存是必须的。操作人员可以借助本系统对产品进行盘存操作：添加新的盘存信息和对盘存新信息的修改和删除。
- ⑦ 操作人员可以对系统进行管理，在系统管理中有对收货单位、供货单位、仓库和经手人的操作，包括删除、修改和添加。
- ⑧ 操作人员还能够对自己的密码进行重新设定。

#### 2. 系统模块设计

根据系统的需求分析，我们对本系统的模块进行划分，系统分为六大模块：管理产品模块、管理入库模块、管理出库模块、管理盘存模块、管理用户和系统管理模块。各模块所包含的文件及其功能如表 19-50 所示。

表 19-50 仓库管理模块一览表

模块名	文件名	功能描述
管理产品模块	ManagerProducts.aspx	管理产品界面设计文件
	ManagerProducts.aspx.cs	实现管理产品界面的代码文件
	AddProductInfo.aspx	添加产品界面设计文件
	AddProductInfo.aspx.cs	实现添加产品界面的代码文件
管理入库模块	ManagerInWarehouse.aspx	管理产品入库界面设计文件
	ManagerInWarehouse.aspx.cs	实现管理产品入库界面的代码文件
	AddInWarehouse.aspx	登记产品入库界面设计文件

(续表)

模块名	文件名	功能描述
管理入库模块	AddInWarehouse.aspx.cs	实现登记产品入库界面的代码文件
	InWarehouseStatistics.aspx	年度入库统计界面设计文件
	InWarehouseStatistics.aspx.cs	实现年度入库统计界面的代码文件
管理出库模块	ManagerOutWarehouse.aspx	管理产品出库界面设计文件
	ManagerOutWarehouse.aspx.cs	实现管理产品出库界面的代码文件
	AddOutWarehouse.aspx	登记产品出库界面设计文件
	AddOutWarehouse.aspx.cs	实现登记产品出库界面的代码文件
	OutWarehouseStatistics.aspx	年度出库统计界面设计文件
	OutWarehouseStatistics.aspx.cs	实现年度出库统计界面的代码文件
管理用户模块	UpdatePwd.aspx	修改密码界面设计文件
	UpdatePwd.aspx.cs	实现修改密码界面的代码文件
系统管理模块	AddReceivingUnit.aspx	添加收货单位界面设计文件
	AddReceivingUnit.aspx.cs	实现添加收货单位界面的代码文件
系统管理模块	ManagerReceivingUnit.aspx	管理收货单位界面设计文件
	ManagerReceivingUnit.aspx.cs	实现管理收货单位界面的代码文件
	ManagerSupplyUnits.aspx	管理供货单位界面设计文件
	ManagerSupplyUnits.aspx.cs	实现管理供货单位界面的代码文件
	AddWarehouse.aspx	添加仓库界面设计文件
	AddWarehouse.aspx.cs	实现添加仓库界面的代码文件
	ManagerWarehouse.aspx	管理仓库界面设计文件
	ManagerWarehouse.aspx.cs	实现管理仓库界面的代码文件
	AddPeople.aspx	添加经手人界面设计文件
	AddPeople.aspx.cs	实现添加经手人界面的代码文件
	ManagerPeople.aspx	管理经手人界面设计文件
	ManagerPeople.aspx.cs	实现管理经手人界面的代码文件
	AddSupply.aspx	添加供货货单位界面设计文件
	AddSupply.aspx	实现添加供货单位界面的代码文件

19.8.2 系统数据库设计

根据系统的需求分析，我们开始对数据库进行合理的设计。首先在 Sql Server 2005 中建立一个名为“ManagerWarehouse”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [ManagerWarehouse] //创建数据库
USE [ManagerWarehouse] //使用数据库
```

根据系统的需求分析和模块设计，至少需要的数据表包括：产品信息表、入库产品信息表、出库产品信息表、供货单位信息表、收货单位信息表、产品库存表、用户信息表、经手人信息表和仓库信息表。

1. 数据库表设计

（1）产品信息表（Product），用来记录仓库中所有产品的详细信息，使用产品编号 HID 作为表的主键。该表的字段结构如表 19-51 所示。

表 19-51 Product 表结构

字段	中文描述	数据类型	是否为空	备注
HID	产品编号	int	否	主键
ProductName	产品名称	char(10)	否	
Unit	产品单位	char(10)	否	
Warehouse	产品存放的仓库	char(10)	否	
InPrice	入库单价	decimal(18,6)	是	
OutPrice	出库单价	decimal(18,6)	是	
Original	原始库存	decimal(18,6)	是	
LowLine	库存下限数量	int	否	
UpLine	库存上限数量	int	否	
Number	产品数量	int	否	
Result	库存数量	int	否	

（2）入库产品信息表（InWarehouse），用来记录进入仓库产品的详细信息，入库编号 IId 是该表的主键。该表的字段结构如表 19-52 所示。

表 19-52 InWarehouse 表结构

字段	中文描述	数据类型	是否为空	备注
IId	入库编号	int	否	主键
IDate	产品入库日期	datetime	否	
Number	产品编号	int	否	
ProductName	产品名称	char(10)	否	
Count	产品数量	int	否	
InPrice	产品入库单价	decimal(19,4)	否	
Total	产品入库总价	decimal(19,4)	否	
SupplyUnit	供货单位	char(10)	否	
Person	经手人	char(10)	否	
Notes	备注	char(10)	是	

（3）出库产品信息表（OutWarehouse），用来记录出库产品的详细信息，我们选择出库编号 OId 作为主键。该表的字段结构如表 19-53 所示。

表 19-53 OutWarehouse 表结构

字段	中文描述	数据类型	是否为空	备注
Old	出库编号	int	否	主键
ODate	产品出库日期	datetime	否	
Number	产品编号	int	否	
ProductName	产品名称	char(10)	否	
Count	产品数量	int	否	
InPrice	产品出库单价	decimal(19,4)	否	
Total	产品出库总价	decimal(19,4)	否	
SupplyUnit	收货单位	char(10)	否	
Person	经手人	char(10)	否	
Notes	备注	char(10)	是	
GetPrice	收到的货款	decimal(18,0)	否	

（4）供货单位信息表（SupplyUnit），用来记录供货单位的详细信息，供货编号 GID 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-54 所示。

表 19-54 SupplyUnit 表结构

字段	中文描述	数据类型	是否为空	备注
GId	供货编号	int	否	主键
Number	供货单位编号	Int	否	
Name	供货单位名称	char(10)	否	
Address	供货单位地址	varchar(50)	是	
person	供货单位联系人	varchar(50)	是	
tel	供货单位联系电话	varchar(50)	是	

（5）收货单位信息表（ReceivingUnit），用来记录收货单位的详细信息，与供货单位信息表相似，收货编号 RId 是主键。该表的字段结构如表 19-55 所示。

表 19-55 ReceivingUnit 表结构

字段	中文描述	数据类型	是否为空	备注
RId	收货编号	int	否	主键
Number	收货单位编号	int	否	
Name	收货单位名称	varchar(50)	否	
Address	收货单位地址	varchar(50)	是	
person	收货单位联系人	varchar(50)	是	
tel	收货单位联系电话	varchar(50)	是	

（6）产品库存表（Result），用来记录产品库存的详细信息，根据主键唯一性原则，设定库存编号 HId 为主键。该表的字段结构如表 19-56 所示。

表 19-56 Result 表结构

字段	中文描述	数据类型	是否为空	备注
HId	库存编号	int	否	主键
Name	产品名称	char(10)	否	
Unit	产品单位	char(10)	否	
Warehouse	存放的仓库	char(10)	否	
Number	产品编号	int	否	
Result	库存数量	int	否	

(7) 用户信息表 (Users)，用来记录使用本系统的用户信息，其中，用户姓名可以重复，而用户编号 userID 是不可重复的，主键非用户编号莫属。该表的字段结构如表 19-57 所示。

表 19-57 Users 表结构

字段	中文描述	数据类型	是否为空	备注
userID	用户编号	int	否	主键
UserName	用户姓名	nvarchar(50)	否	
Pwd	用户密码	nvarchar(50)	否	

(8) 经手人信息表 (Person)，用来记录各种操作的经办人信息。该表的字段结构如表 19-58 所示。

表 19-58 Person 表结构

字段	中文描述	数据类型	是否为空	备注
userID	用户编号	int	否	主键
Name	经手人姓名	char(10)	否	
tel	经手人电话	char(10)	是	
Number	经手人编号	int	否	

(9) 仓库信息表 (Warehouse)，用来存放产品的各个仓库的信息。该表的字段结构如表 19-59 所示。

表 19-59 Warehouse 表结构

字段	中文描述	数据类型	是否为空	备注
WId	自增编号	int	否	主键
Number	仓库编号	int	否	
Name	仓库名称	char(10)	是	
Notes	备注	char(10)	否	

19.8.3 系统运行示例

本系统运行后，首先出现的是登录页面，如图 19-48 所示。

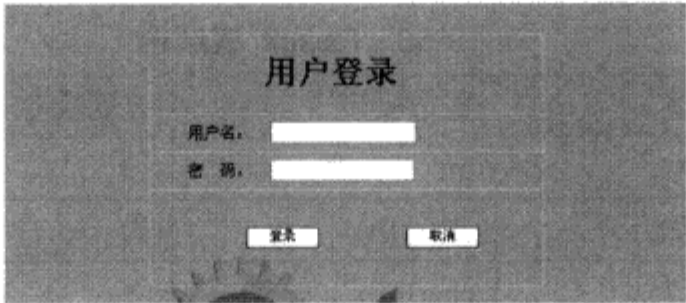


图 19-48 用户登录页面

用户在该页面中，输入用户名和密码，单击“登录”按钮，通过验证进入到系统首页，如图 19-49 所示。



图 19-49 系统首页

在首页中，用户可以对供货单位进行管理。单击表中“修改”的链接，进入修改供货单位的页面，如图 19-50 所示。

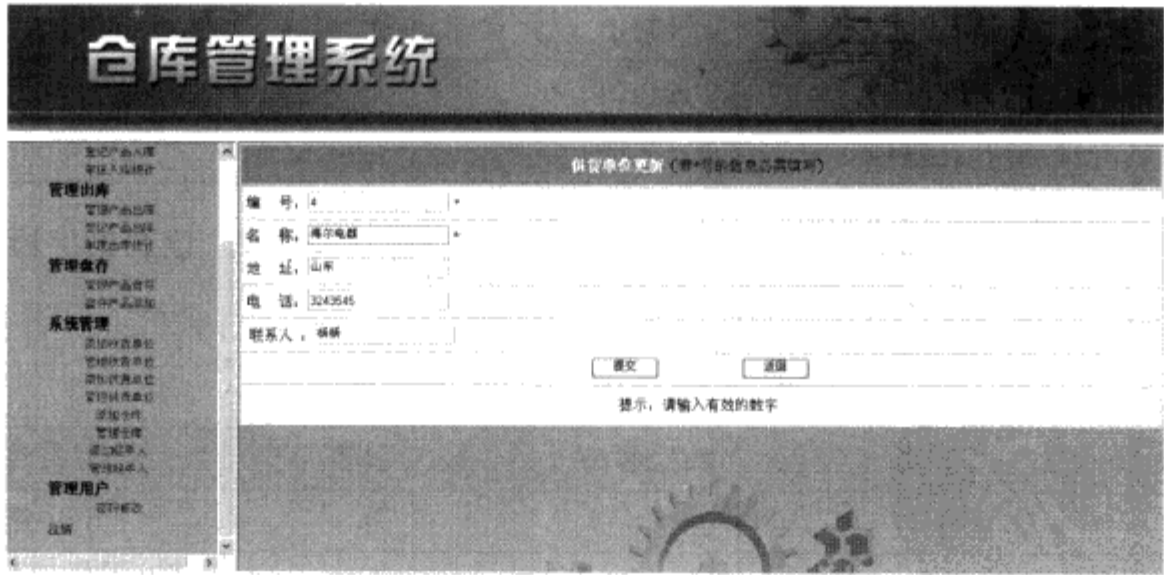


图 19-50 修改供货单位的页面

用户在此页面中，填写供货单位的名称、地址、电话和联系人，单击“提交”按钮，完成修改供货单位的操作。

在首页中，用户单击列表中的“删除”链接，可以对所选信息进行删除操作。当用户单击页面左侧的树形菜单中“管理入库”下的子菜单“年度入库统计”，就进入年度入库统计页面，如图 19-51 所示。

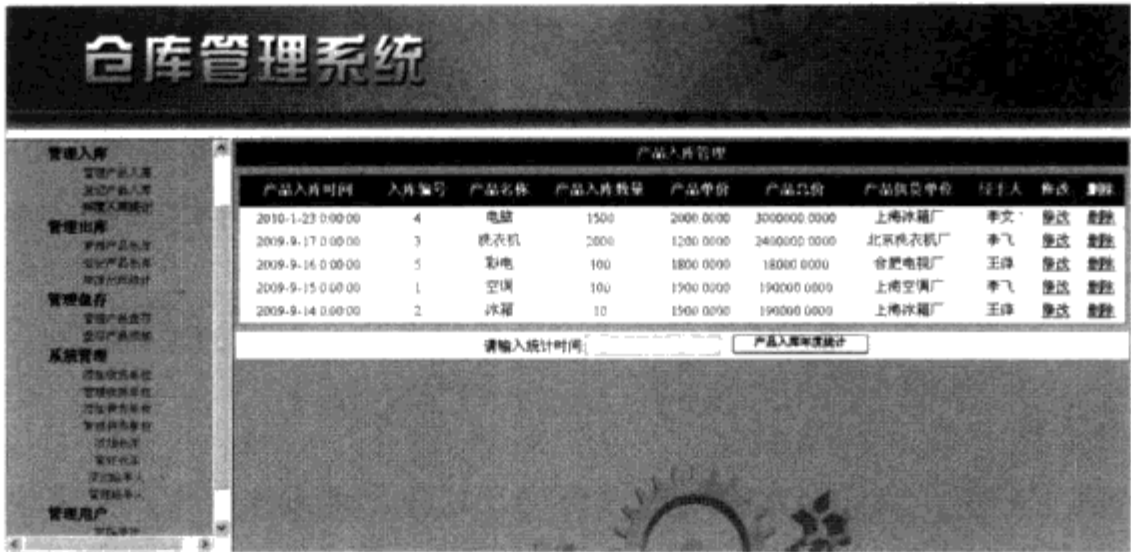


图 19-51 年度入库统计页面

在该页面中，用户除了可以对入库产品进行修改和删除操作，还可以输入统计的时间，查询每一年度的入库产品信息。

以上演示了系统中主要页面，其他的页面和功能操作基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行学习。

## 19.9 项目 09：学生宿舍管理系统

当今社会是飞速发展的世界，原始的数据记录方式已经逐渐被社会所淘汰，计算机化管理是适应时代发展的产物。21 世纪的今天，信息社会占着主流地位，计算机在各行各业中的使用已经相当普及，自动化、信息化的管理越来越广泛地应用于各个领域。学生宿舍管理系统对于一个学校来说是必不可少的组成部分。它采用的是计算机化管理，管理人员需要做的就是将数据输入到系统的数据库中去。由于数据库存储容量相当大，而且比较稳定，适合较长时间的保存，也不容易丢失。这无疑为信息存储量比较大的学校提供了一个方便、快捷的操作方式。这些优点能够极大地提高效率，也是学校科学化、正规化管理的重要条件。本节将介绍的学生管理系统就能满足上述的要求。

### 19.9.1 系统分析与设计

本系统设计的目标是：操作尽量人性化、运行速度快、安全性高、稳定性好，并且具备完善的修改功能，能够快速查询学校所需的住宿信息。

#### 1. 系统需求分析

根据系统开发的目的，需要实现的用户需求总述如下。

- ① 系统管理员必须先从登录界面进入系统，在登录页面输入用户名和密码，通过身份验证后，才可以进入系统的首页。如果未能通过系统的身份验证，系统自动给出登录错误的提示信息。
- ② 通过身份验证的系统管理员进系统首页。在首页中可以通过房间编号、所在楼名和房间类型对宿舍的房间信息进行查询。可以选择相应的房间来更新该房间的信息。同时，还可以对房间信息进行单一的或批量的删除操作。当然，添加房间信息的功能也是必不可少的。
- ③ 系统管理员能够对学生进行入住宿舍信息的登记。也使用学生学号、学生姓名、所在宿舍

楼名和房间的编号对宿舍信息进行查询。并且可以对学生换宿舍房间和退出宿舍房间进行操作。

④ 系统管理员通过学生学号、学生姓名和缴费类别查询学生的缴费信息。可以选择相应的缴费信息来更新。同时，还可以对学生缴费的信息进行删除操作和登记新的宿舍的缴费信息。

⑤ 系统管理员还能够通过学生学号、学生姓名和学生的专业查询学生的信息。同样，具有删除和修改功能。

⑥ 在系统设置的菜单中，能够设置学生的专业信息和修改登录的密码。

2. 系统模块设计

根据系统的需求分析。我们对本系统的模块进行划分，将系统分为六大模块：数据库管理模块、实体类模块、管理宿舍房间模块、管理住宿信息模块、管理缴费信息模块、系统设置模块和学生信息管理模块。各模块所包含的文件及其功能如表 19-60 所示。

表 19-60 宿舍管理模块一览表

模块名	文件名	功能描述
数据库管理模块	App_Code/DataAccessHelper/SQLString.cs	Sql 字符串格式化类代码文件
	App_Code/DataAccessLayer/ DataBase.cs	数据库公共访问类代码文件
实体类模块	App_Code/BusinessLogicLayer/Accommodation.cs	宿舍实体类代码文件
	App_Code/BusinessLogicLayer/Admin.cs	管理员实体类代码文件
	App_Code/BusinessLogicLayer/GiveMoneyInfo.cs	缴费实体类代码文件
	App_Code/BusinessLogicLayer/Payment.cs	付费实体类代码文件
	App_Code/BusinessLogicLayer/Room.cs	房间实体类代码文件
	App_Code/BusinessLogicLayer/Student.cs	学生实体类代码文件
管理宿舍房间模块	roomInfoAdd.aspx	添加房间信息界面设计文件
	roomInfoAdd.aspx.cd	实现添加房间信息界面的代码文件
	roomInfoManage.aspx	管理房间信息界面设计文件
	roomInfoManage.aspx.cs	实现管理房间信息界面设计的文件
	roomInfoUpdate.aspx	更新房间信息界面设计文件
	roomInfoUpdate.aspx.cs	实现更新房间信息界面设计的文件
管理缴费信息模块	PaymentInfoAdd.aspx	登记缴费界面设计文件
	PaymentInfoAdd.aspx.cs	实现登记缴费界面的代码文件
	PaymentInfoManage.aspx	管理缴费信息界面设计文件
	PaymentInfoManage.aspx.cs	实现管理缴费信息界面的代码文件
	PaymentInfoUpdate.aspx	更新缴费信息管理界面设计文件
	PaymentInfoUpdate.aspx.cs	实现更新缴费信息界面的代码文件
系统设置模块	ProfessionSet.aspx	设置专业界面设计文件
	ProfessionSet.aspx.cs	实现设置专业界面的代码文件
	login.aspx	登录界面设计文件
	login.aspx.cs	实现登录界面的代码文件
	Password.aspx	修改密码界面设计文件
	Password.aspx.cs	实现修改密码界面的代码文件

(续表)

模块名	文件名	功能描述
学生信息管理模块	studentInfoAdd.aspx	添加学生信息界面的代码文件
	studentInfoAdd.aspx.cs	实现添加学生信息界面的代码文件
	studentInfoManage.aspx	学生信息管理界面的代码文件
	studentInfoManage.aspx.cs	实现学生信息管理界面的代码文件
	studentInfoUpdate.aspx	更新学生信息界面的代码文件
	studentInfoUpdate.aspx.cs	实现更新学生信息界面的代码文件

19.9.2 系统数据库设计

根据系统的需求分析，我们开始对数据库进行设计。首先在 Sql Server 2005 中建立一个名为“StudentBuildingManage”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [StudentBuildingManage] //创建数据库
USE [StudentBuildingManage] //使用数据库
```

根据前面的系统需求分析和模块设计，所需要的数据表包括：系统管理员表、宿舍信息表、楼栋信息表、学院信息表、住宿费用表、费用类型表、专业信息表、房间信息表、房间类型表和学生信息表。

(1) 系统管理员表（admin），用来记录使用本系统的管理员的基本信息，使用管理员的用户名 adminUsername 作为表的主键。该表的字段结构如表 19-61 所示。

表 19-61 admin 表结构

字段	中文描述	数据类型	是否为空	备注
adminUsername	管理员用户名	nvarchar(20)	否	主键
adminPassword	管理员密码	nvarchar(20)	是	

(2) 宿舍信息表（accommodation），用来记录学生宿舍的详细信息，学生编号 studentNumber 是该表的主键。该表的字段结构如表 19-62 所示。

表 19-62 accommodation 表结构

字段	中文描述	数据类型	是否为空	备注
studentNumber	学生编号	nvarchar(20)	否	主键
buildingName	楼栋名称	nvarchar(20)	是	
roomNo	房间号	nvarchar(20)	是	
accommodationinYear	入住年份	smallint	是	
accommodationinMonth	入住月份	smallint	是	
accommodationinDay	入住日期	smallint	是	
accommodationNotes	备注	ntext	是	

(3) 楼栋信息表 (buildingInfo)，用来记录宿舍房间所属楼栋信息，我们选择楼栋名称 buildingName 作为主键。该表的字段结构如表 19-63 所示。

表 19-63 buildingInfo 表结构

字段	中文描述	数据类型	是否为空	备注
buildingName	楼栋名称	nvarchar(20)	否	主键

(4) 学院信息表 (collegeInfo)，用来记录学生就读的学院信息，学院名称 collegeName 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-64 所示。

表 19-64 collegeInfo 表结构

字段	中文描述	数据类型	是否为空	备注
collegeName	学院名称	nvarchar(50)	否	主键

(5) 住宿费用表 (paymentInfo)，用来记录所有住宿费用的信息，费用编号 paymentId 是主键。该表的字段结构如表 19-65 所示。

表 19-65 paymentInfo 表结构

字段	中文描述	数据类型	是否为空	备注
paymentId	费用编号	int	否	主键
studentNumber	学生编号	nvarchar(20)	是	
paymentType	费用类型	nvarchar(10)	是	
paymentYear	缴费年份	smallint	是	
paymentMonth	缴费月份	smallint	是	
paymentDay	缴费日期	smallint	是	
paymentMoney	缴费金额	float	是	

(6) 费用类型表 (paymentTypeInfo)，用来记录所有费用的类型信息，费用类型 paymentType 是主键。该表的字段结构如表 19-66 所示。

表 19-66 paymentTypeInfo 表结构

字段	中文描述	数据类型	是否为空	备注
paymentType	费用类型	nvarchar(50)	否	主键

(7) 专业信息表 (professionInfo)，用来记录学院专业的信息，专业名称 professionName 是主键。该表的字段结构如表 19-67 所示。

表 19-67 professionInfo 表结构

字段	中文描述	数据类型	是否为空	备注
professionName	专业名称	nvarchar(50)	否	主键
collegeName	学院名称	nvarchar(50)	是	主键

（8）房间信息表（roomInfo），用来记录宿舍中房间的详细信息，房号 roomNo 是主键。该表的字段结构如表 19-68 所示。

表 19-68 roomInfo 表结构

字段	中文描述	数据类型	是否为空	备注
roomNo	房间编号	nvarchar(20)	否	主键
buildingName	楼栋名称	nvarchar(20)	是	
roomType	房间类型	nvarchar(10)	是	
roomPrice	房费	float	是	
numberOfBed	床位数	smallint	是	
leftNumberOfBed	剩余床位	smallint	是	
roomTelephone	房间电话	nvarchar(20)	是	
roomNotes	备注	ntext	是	

（9）房间类型表（roomTypeInfo），用来记录宿舍中房间类型的信息，房间类型名称 roomTypeName 是主键。该表的字段结构如表 19-69 所示。

表 19-69 roomTypeInfo 表结构

字段	中文描述	数据类型	是否为空	备注
roomTypeName	房间类型名称	nvarchar(50)	否	主键

（10）学生信息表（studentInfo）用来记录学生的详细信息，学生编号 studentNumber 是主键。该表的字段结构如表 19-70 所示。

表 19-70 studentInfo 表结构

字段	中文描述	数据类型	是否为空	备注
studentNumber	学生编号	nvarchar(20)	否	主键
studentName	学生姓名	nvarchar(20)	是	
studentSex	学术性别	nvarchar(10)	是	
studentState	学生状态	float	是	
collegeName	学院名称	smallint	是	
studentProfession	学生专业	smallint	是	
studentBirthday	学生生日	nvarchar(20)	是	
studentAddress	学生住址	nvarchar(50)	是	
studentNotes	备注	ntext	是	

19.9.3 系统运行示例

运行本系统后，首先出现的是登录页面，如图 19-52 所示。

用户在该页面中，输入用户名和密码，单击“登录”按钮，通过身份验证后，进入系统首页，如图 19-53 所示。

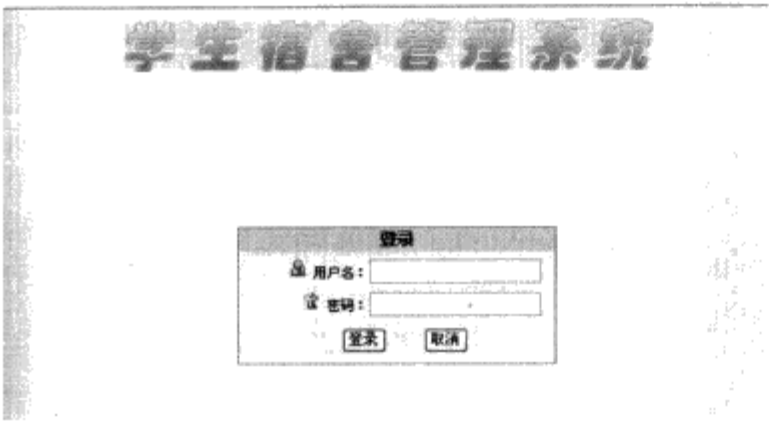


图 19-52 用户登录页面

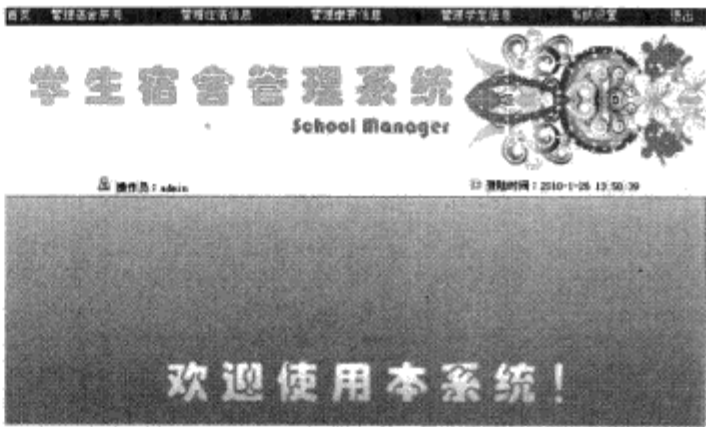


图 19-53 系统首页

用户在首页中，将鼠标放到菜单栏的“管理宿舍信息”菜单上，从弹出的二级菜单中选择“查询宿舍信息”子菜单，进入查询宿舍信息的页面，如图 19-54 所示。

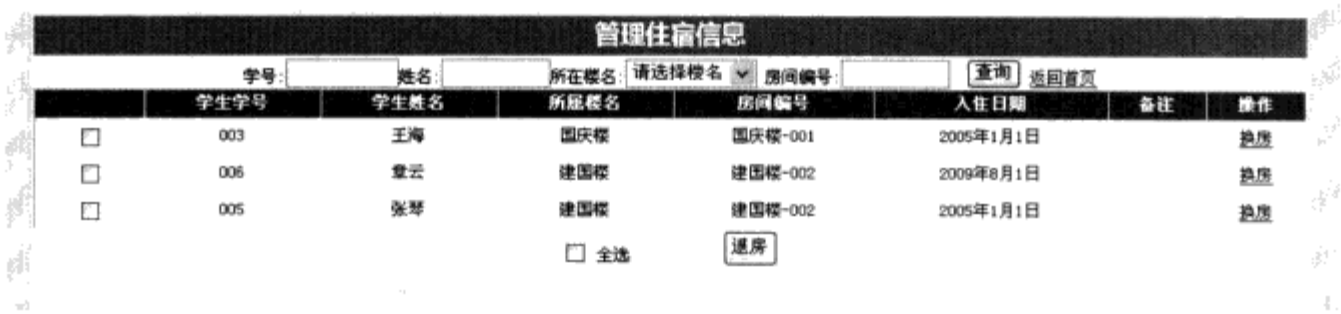


图 19-54 查询宿舍信息页面

在该页面中，用户通过学号、姓名和房间编号可以查询相关的宿舍信息。单击宿舍信息列表中的“换房”链接，可以进入换房页面，如图 19-55 所示。

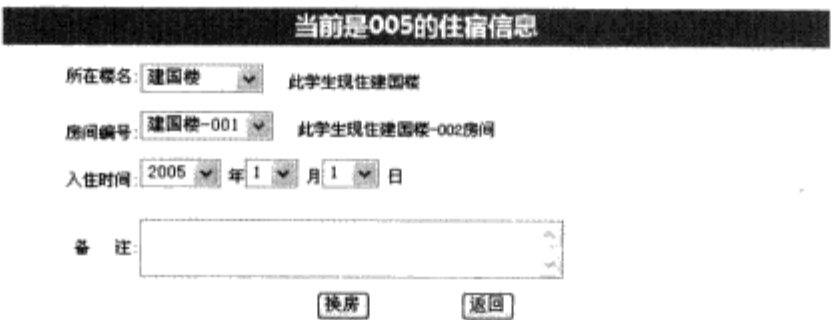


图 19-55 换房页面

用户在此页面中，选择所在楼名、房间编号和入住时间，单击“换房”按钮，完成换房的操作。

在首页中，用户将鼠标放到菜单栏的“管理宿舍信息”菜单上，从弹出的二级菜单中选择“入住宿舍登记”子菜单，进入入住宿舍登记的页面，如图 19-56 所示。



图 19-56 入住宿舍登记页面

在该页面中，填写学生编号、所在楼名、房间编号和入住时间后，单击“添加”按钮，完成学生入住宿舍的操作。

用户在首页中，将鼠标放到菜单栏的“系统设置”菜单上，从弹出的二级菜单中选择“专业信息设置”子菜单，进入专业信息设置的页面，如图 19-57 所示。

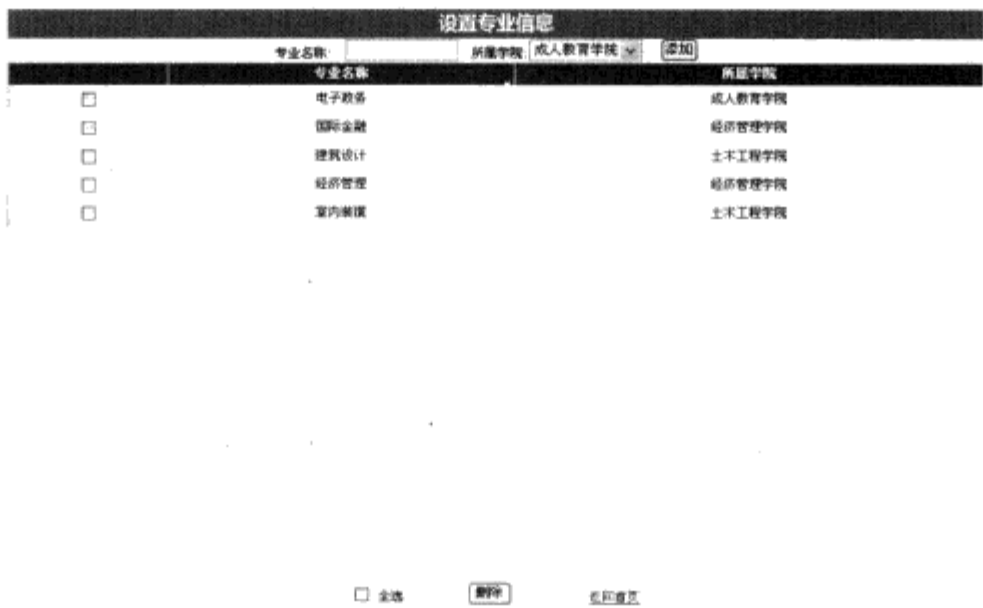


图 19-57 设置专业信息页面

在该页面中，用户可以单击“添加”按钮设置新的专业信息。也可以单选或多选已经存在的专业，单击“删除”的按钮进行删除的操作。

上面做了系统中主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

## 19.10 项目 10：机票预订系统

信息科技的快速发展，使计算机的应用已普及到经济和社会生活的各个领域。机票销售领域自然也不例外。传统的纸质机票要经历印刷、销售、运输、存档等环节，除销售渠道建设和维护成本外，每张机票大约花费航空公司人民币 50~60 元的成本；而在网上的售票预定系统可以提供航空乘客在网上查询、预定、支付后，在机场进出港系统确认了其行程、姓名等基本信息后，仅凭一张身份证就可办理一整套登机手续，降低了销售成本。开发本系统是为了提高机票预定的效率，减少错误的发生，方便用户预定和查询航班信息。

### 19.10.1 系统分析与设计

#### 1. 系统需求分析

本系统的用户有两类：一类是订票用户，一类是系统管理员。

- ① 系统管理员必须先从登录界面进入系统，在登录页面输入用户名和密码，通过身份验证后，才可以进入系统后台管理界面。
- ② 系统管理员在后台中可以对用户登录进行管理，包括添加用户、修改密码和删除用户等操作。

- ③ 系统管理员能够对系统的数据进行管理，包括添加航班信息、更新库存机票和查看库存票量。
- ④ 系统管理员还可以对订单进行管理，包括查看客户订单、处理订单状态、删除订单等操作。
- ⑤ 系统管理员能够对客户资料进行备份操作。
- ⑥ 系统管理员可以查看客户的投诉和意见，并能删除投诉和意见。
- ⑦ 订票用户可以在网站查询国内和国外机票后进行机票预订。
- ⑧ 用户可以在网站发布对网站的投诉和建议。

2. 系统模块设计

根据系统的需求分析，我们对本系统的模块进行划分，系统分为前台模块和后台模块。这两个模块所包含的文件及其功能如表 19-71 所示。

表 19-71 机票预订系统模块一览表

模块名	文件名	功能描述
前台模块	DomesticSelecet.aspx	首页界面设计的代码文件
	DomesticSelecet.aspx.cs	实现首页界面的代码文件
	DetailedInfor.aspx	显示航班查询结果界面设计的代码文件
	DetailedInfor.aspx.cs	实现显示航班查询结果界面的代码文件
	ShowDomestic.aspx	国际机票查询界面设计的代码文件
	ShowDomestic.aspx.cs	实现国际机票查询界面的代码文件
	ShowOversea.aspx	国际机票查询结果界面设计的代码文件
	ShowOversea.aspx.cs	实现国际机票查询结果界面的代码文件
	DetaildeSuggestion.aspx	提交建议界面设计的代码文件
	DetaildeSuggestion.aspx.cs	实现提交建议界面的代码文件
	ManageOrder.aspx	航班订单填写界面设计的代码文件
	ManageOrder.aspx.cs	实现航班订单填写界面的代码文件
	TimeDifference.aspx	显示时差界面设计的代码文件
	TimeDifference.aspx.cs	实现显示时差界面的代码文件
	OrderForm.aspx	提交订单详情界面设计的代码文件
	OrderForm.aspx.cs	实现提交订单详情界面的代码文件
	Payment.aspx	显示付款方式界面设计的代码文件
	Payment.aspx.cs	实现显示付款方式界面设计的文件
后台管理模块	AddFlightInfor.aspx	添加航班信息界面设计文件
	AddFlightInfor.aspx.cs	实现添加航班信息界面的代码文件
	AddUser.aspx	添加用户信息界面设计文件
	AddUser.aspx.cs	实现添加用户信息界面的代码文件
	DeleteUser.aspx	删除用户信息界面设计文件
	DeleteUser.aspx.cs	实现删除用户信息界面的代码文件
	DetaildeSuggestion.aspx	显示建议详情界面设计的代码文件

(续表)

模块名	文件名	功能描述
后台管理模块	DetaildeSuggestion.aspx.cs	实现显示建议详情界面的代码文件
	DetailedInfor.aspx	显示订单详情界面设计的代码文件
	DetailedInfor.aspx.cs	实现显示订单详情界面的代码文件
	Login.aspx	管理员登录界面设计的代码文件
	Login.aspx.cs	实现管理员登录界面的代码文件
	LookOrder.aspx	查看订单界面设计的代码文件
	LookOrder.aspx.cs	实现查看订单界面的代码文件
	LookSug.aspx	查看建议界面设计的代码文件
	LookSug.aspx.cs	实现查看建议界面的代码文件
	UpDateFlight.aspx	更新库存机票数界面设计的代码文件
	UpDateFlight.aspx.cs	实现更新库存机票数界面的代码文件
	UpDateStock.aspx	更新航班信息界面设计的代码文件
	UpDateStock.aspx.cs	实现更新航班信息界面的代码文件
	AlterPW.aspx	更新用户密码界面设计文件
	AlterPW.aspx.cs	实现更新用户密码界面的代码文件

19.10.2 系统数据库设计

1. 数据库表设计

首先在 Sql Server 2005 中建立一个名为“TicketSaler”的数据库来存放本系统所必须的数据表，创建数据库的语句如下：

```
CREATE DATABASE [TicketSaler] //创建数据库
USE [TicketSaler] //使用数据库
```

根据系统的需求分析和模块设计，所需要的数据表包括：订单用户表、用户建议表、航班信息表、航班订单信息表和管理员表。

(1) 管理员表（SystemUser），用来记录使用本系统的管理员的基本信息，使用管理员的编号 id 作为表的主键。该表的字段结构如表 19-72 所示。

表 19-72 SystemUser 表结构

字段	中文描述	数据类型	是否为空	备注
id	管理员编号	Int	否	主键
user_name	管理员用户名	nvarchar(20)	是	
pass_word	管理员密码	nvarchar(20)	是	

(2) 订单用户信息表（ConsumerOrder），用来记录订单用户的详细信息，订单客户编号 id 是该表的主键。该表的字段结构如表 19-73 所示。

表 19-73 ConsumerOrder 表结构

字段	中文描述	数据类型	是否为空	备注
id	订单客户编号	int	否	主键
address	地址	nvarchar(max)	是	
code	身份证号	nvarchar(50)	是	
relationer	联系人	nvarchar(50)	是	
email	电子邮件	nvarchar(50)	是	
mobile_phone	手机	nchar(20)	是	
telephone	电话	nchar(20)	是	
personal_id	个人编号	int	是	
try	订单状态	nvarchar(50)	是	

（3）用户建议表（ConsumerSuggestion），用来记录用户对网站的建议信息，我们选择建议编号 id 作为主键。该表的字段结构如表 19-74 所示。

表 19-74 ConsumerSuggestion 表结构

字段	中文描述	数据类型	是否为空	备注
id	建议编号	lint	否	主键
consumer_name	用户名称	nvarchar(50)	是	
suggestion	建议内容	varchar(max)	是	
email	电子邮件	nvarchar(50)	是	
mobile_phone	手机	varchar(50)	是	
telephone	电话	varchar(50)	是	

（4）航班信息表（FlightInformation），用来记录所有航班的详细信息，航班编号 id 具有唯一性，所以，可以设置成为主键字段。该表的字段结构如表 19-75 所示。

表 19-75 FlightInformation 表结构

字段	中文描述	数据类型	是否为空	备注
id	航班编号	int	否	主键
company_type	航空公司	varchar(50)	是	
starting_city	起飞城市	varchar(50)	是	
terminus_city	降落城市	varchar(50)	是	
OneWay_price	单程票价	money	是	
GoAndBack_price	往返票价	money	是	
flight_type	飞机类型	varchar(50)	是	
starting_date	起飞日期	varchar(50)	是	
arriving_date	降落日期	varchar(50)	是	
starting_time	起飞时间	varchar(50)	是	
arriving_time	降落时间	varchar(50)	是	
num	数量	int	是	
bunk_type	机舱类型	varchar(50)	是	
starting_airport	起飞机场	varchar(50)	是	
arriving_airport	降落机场	varchar(50)	是	

（5）航班订单表（ManageFlightInfor）用来记录订单的航班信息，航班订单编号 id 是主键。

该表的字段结构如表 19-76 所示。

表 19-76 ManageFlightInfor 表结构

字段	中文描述	数据类型	是否为空	备注
id	航班编号	int	否	主键
company	航空公司	varchar(50)	是	
flighttype	飞机类型	varchar(50)	是	
stacity	起飞城市	varchar(50)	是	
arrcity	降落城市	varchar(50)	是	
startairport	起飞城市	varchar(50)	是	
arrivingairport	起飞机场	varchar(50)	是	
onewayprice	单程票价	money	是	
backprice	回程票价	money	是	
startingtime	起飞时间	varchar(50)	是	
arrivingtime	降落时间	varchar(50)	是	
cusmnum	预订数量	int	是	
bunktype	机舱类型	varchar(50)	是	
tdate	起飞日期	varchar(50)	是	
flight_id	航班编号	int	是	

2. 设计系统视图

当开发中遇到经常重复的查询和相同数据的复杂 SQL 语句时，可以将它们建立为视图。本数据库建立了 1 张视图：订单管理视图（ManageOrder），由 ConsumerOrder 表、FlightInformation 表和 ManageFlightInfor 表连接组成。创建视图的语句如下：

```
SELECT dbo.ConsumerOrder.id, dbo.ConsumerOrder.address, dbo.ConsumerOrder.code, dbo.ConsumerOrder.relationer,
 dbo.ConsumerOrder.email, dbo.ConsumerOrder.mobile_phone, dbo.ConsumerOrder.telephone, dbo.ConsumerOrder.try,
 dbo.ManageFlightInfor.company, dbo.ManageFlightInfor.bunktype, dbo.ManageFlightInfor.flighttype, dbo.ManageFlightInfor.stacity,
 dbo.ManageFlightInfor.startairport, dbo.ManageFlightInfor.arrcity, dbo.ManageFlightInfor.arrivingairport,
 dbo.ManageFlightInfor.onewayprice, dbo.ManageFlightInfor.startingtime, dbo.ManageFlightInfor.arrivingtime,
 dbo.ManageFlightInfor.tdate, dbo.ManageFlightInfor.flight_id, dbo.ManageFlightInfor.cusmnum, dbo.FlightInformation.num
FROM dbo.ConsumerOrder INNER JOIN dbo.ManageFlightInfor ON dbo.ConsumerOrder.id = dbo.ManageFlightInfor.id INNER
JOIN dbo.FlightInformation ON dbo.ManageFlightInfor.flight_id = dbo.FlightInformation.id
```

19.10.3 系统运行示例

本系统运行后，出现的是系统的首页，如图 19-58 所示。



图 19-58 系统页面

用户在该页面中，输入出发城市、出发日期、返程日期和到达城市，单击“查询”按钮，机票信息会显示在右面的列表中。单击菜单上的“国际机票”，进入国际机票查询页面，如图 19-59 所示。

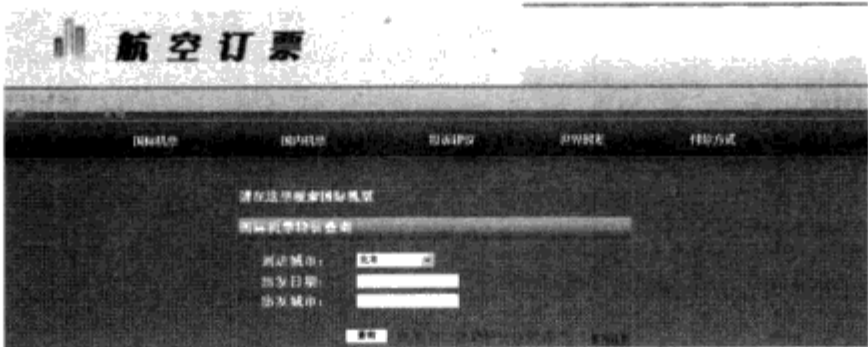


图 19-59 国际机票查询页面

在该页面上，用户输入到达城市、出发日期和出发城市，单击“查询”按钮，进入机票信息的显示页面，如图 19-60 所示。

航空订票								
国际机票								
国内机票								
付款方式								
机票预订								
世界时差								
航空公司	出发城市	到达城市	单程票价	往返票价	飞机型号	起飞时间	到达时间	
美大陆航空	北京	纽约	5010.0000	0.0000	MU271/MU587	21:20	10:10	预订
东方航空	北京	纽约	4500.0000	0.0000	空客330 (大)	13:20	19:40	预订
美联航	北京	纽约	3920.0000	0.0000	麦道90 (中)	08:10	19:40	预订

图 19-60 显示机票信息界面

用户选择显示机票信息的列表中，单击“预订”按钮，进入预订页面，如图 19-61 所示。

航空订票

2010年2月5日 星期五 Friday

国际机票

国内机票

航空公司: 吉祥航空

飞机型号: 空客319 (中)

起飞机场: 首都机场

降落机场: 江北机场

机舱类型: 经济舱

单程票价: 720.0000

往返票价: 0.0000

起飞时间: 13:20

降落时间: 18:00

订票数量:

填写详单

关闭窗口

图 19-61 预订页面

用户在该页中,选择机舱类型和订票的数量,然后,单击“填写详单”的链接,进入填写订单详情的页面,如图 19-62 所示。

在填写订单页面,填写联系人、电子邮件、座机、手机、身份证号和地址信息,单击“预订”按钮,完成机票预订的操作。

系统管理员使用本系统时,在首页中单击“管理员登录”链接,出现登录界面,如图 19-63 所示。

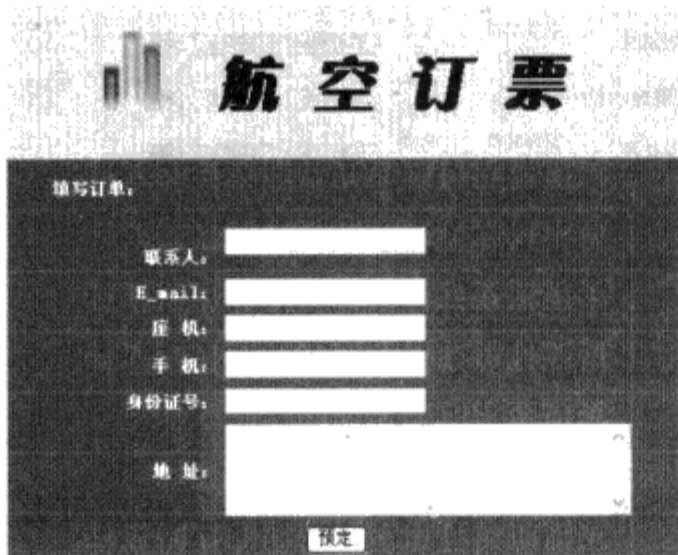
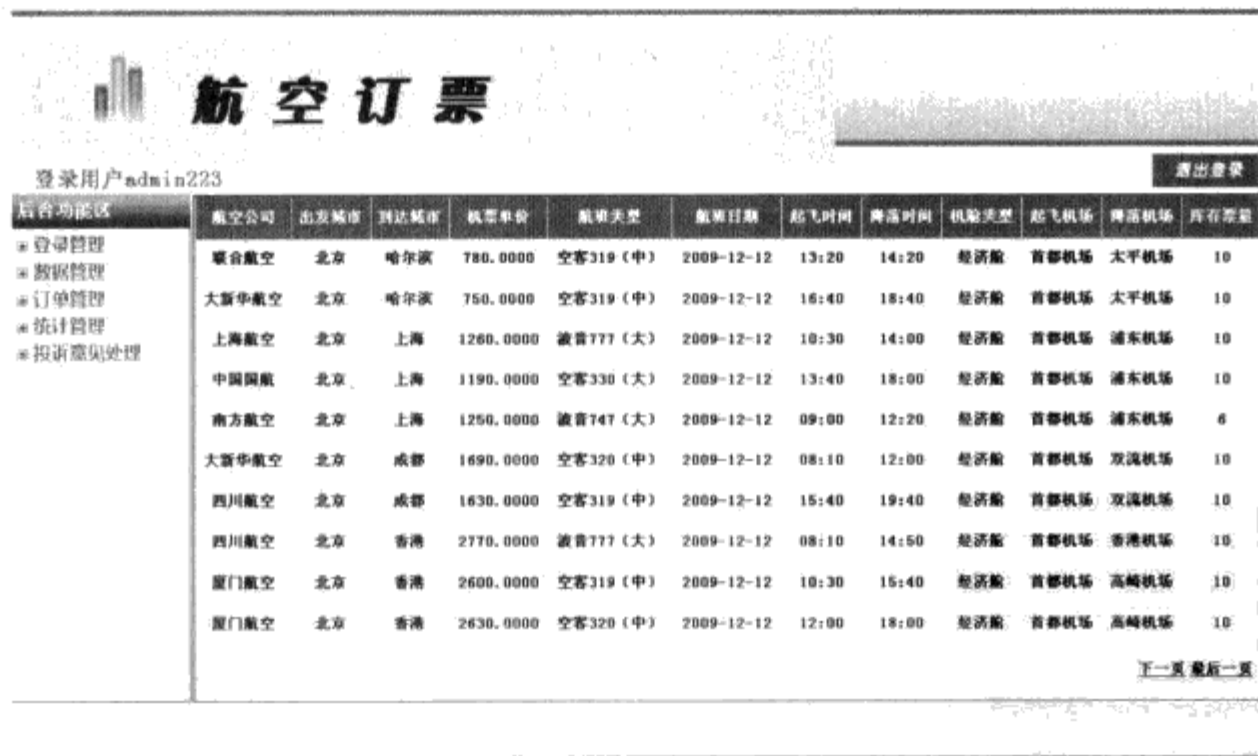


图 19-62 填写订单页面



图 19-63 管理员登录页面

在页面中填写用户名、密码和验证码,单击“登录”按钮,进入后台管理页面,如图 19-64 所示。



航空公司	出发城市	到达城市	机票单价	航班类型	航班日期	起飞时间	降落时间	机舱类型	起飞机场	降落机场	库存数量
联合航空	北京	哈尔滨	780.0000	空客319(中)	2009-12-12	13:20	14:20	经济舱	首都机场	太平机场	10
大新华航空	北京	哈尔滨	750.0000	空客319(中)	2009-12-12	16:40	18:40	经济舱	首都机场	太平机场	10
上海航空	北京	上海	1260.0000	波音777(大)	2009-12-12	10:30	14:00	经济舱	首都机场	浦东机场	10
中国国航	北京	上海	1190.0000	空客330(大)	2009-12-12	13:40	18:00	经济舱	首都机场	浦东机场	10
南方航空	北京	上海	1250.0000	波音747(大)	2009-12-12	09:00	12:20	经济舱	首都机场	浦东机场	6
大新华航空	北京	成都	1690.0000	空客320(中)	2009-12-12	08:10	12:00	经济舱	首都机场	双流机场	10
四川航空	北京	成都	1630.0000	空客319(中)	2009-12-12	15:40	19:40	经济舱	首都机场	双流机场	10
四川航空	北京	香港	2770.0000	波音777(大)	2009-12-12	08:10	14:50	经济舱	首都机场	香港机场	10
厦门航空	北京	香港	2600.0000	空客319(中)	2009-12-12	10:30	15:40	经济舱	首都机场	高崎机场	10
厦门航空	北京	香港	2630.0000	空客320(中)	2009-12-12	12:00	18:00	经济舱	首都机场	高崎机场	10

图 19-64 后台管理页面

管理员在该页面中,单击订单管理的二级菜单“查看客户订单”,进入订单管理页面,如图 19-65 所示。

航空订票

登录用户admin223

退出登录

后台功能 * 登录管理 * 数据管理 * 订单管理 查看客户订单 * 统计管理 * 投诉意见处理	客户姓名	移动电话	出发城市	前往城市	所属航空公司	航班号	状态	审核	查看	操作
	wangyang	13823984654	北京	上海	南方航空	波音747(大)	已处理	处理/取消	详细信息	删除
	xiaogang	13656069765	北京	成都	大新华航空	空客320(中)	未处理	处理/取消	详细信息	删除
	libin	13823984654	北京	哈尔滨	大新华航空	空客319(中)	未处理	处理/取消	详细信息	删除
	hanxu	13675346828	北京	成都	四川航空	空客319(中)	未处理	处理/取消	详细信息	删除
	李斌	13766952675	北京	洛杉矶	美联航	EQV	已处理	处理/取消	详细信息	删除
	李剑	13656069765	北京	首尔	国泰航空	空客330(大)	已处理	处理/取消	详细信息	删除
	孙伟	15946053856	北京	成都	四川航空	空客319(中)	未处理	处理/取消	详细信息	删除
	高洋	15155676573	北京	香港	厦门航空	空客319	未处理	处理/取消	详细信息	删除

图 19-65 管理订单页面

在页面中，管理员可以对订单进行删除、查看详细信息和状态处理等操作。

上面做了系统中主要页面的演示，其他页面基本相似，这里不再一一演示。读者可以运行随书光盘中的源代码进行系统学习。

[ G e n e r a l   I n f o r m a t i o n ]

书名= A S P . N E T   4 . 0从入门到精通

作者= 张正礼 , 王坚宁编著

页数= 6 1 2

出版社= 北京市：清华大学出版社

出版日期= 2 0 1 1 . 0 7

S S 号= 1 2 8 4 9 1 8 2

D X 号= 0 0 0 0 0 8 1 2 6 5 2 2

U R L = h t t p : / / b o o k . s z d n e t . o r g . c n / b o o k D e t a i l . j s  
p ? d x N u m b e r = 0 0 0 0 0 8 1 2 6 5 2 2 & d = E F B 5 5 5 C 7 7 7 0 1 3 B 0 9 D  
2 A B 5 0 F E 3 0 0 A 2 3 5 D