

**.NET** 精选系列

SAMS

# ASP.NET

从

# 入门到精通

Chris Payne 著

赵斌 张滨义 董清波 译

人民邮电出版社  
[www.pptph.com.cn](http://www.pptph.com.cn)

Microsoft  
**.net**

美术编辑：胡平利

# ASP.NET



## 入门到精通

ISBN 7-115-09770-4



9 787115 097705 >

ISBN7-115-09770-4/TP·2531

定价：75.00 元

人民邮电出版社  
[www.pptph.com.cn](http://www.pptph.com.cn)

NET 精选系列

# ASP.NET 从入门到精通

Chris Payne 著

赵斌 张滨义 董清波 译

人民邮电出版社

## 图书在版编目 (CIP) 数据

ASP.NET 从入门到精通 / ( ) 佩恩 (Payne, C.) 著; 赵斌, 张滨义, 董清波译.

—北京: 人民邮电出版社, 2002.1

(.NET 精选系列)

ISBN 7-115-09770-4

I. A... II. ①佩... ②赵... ③张... ④董... III. 主页制作—应用软件, ASP.Net  
IV. TN393.092

中国版本图书馆 CIP 数据核字 (2001) 第 080466 号

## 版 权 声 明

Chris Payne: Teach Yourself ASP.NET in 21 Days

Authorized translation from English language edition published by SAMS Publishing.

Copyright © 2002 by SAMS Publishing.

All rights reserved. For sale in mainland China only.

本书中文简体字版由美国 SAMS 出版公司授权人民邮电出版社出版, 未经出版者书面许可, 对书的任何部分不得以任何方式复制或抄袭。

版权所有, 侵权必究。

.NET 精选系列

### ASP.NET 从入门到精通

◆ 著 Chris Payne

译 赵 斌 张滨义 董清波

责任编辑 俞 彬

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号

邮编 100061 电子函件 315@pptph.com.cn

网址 <http://www.pptph.com.cn>

读者热线 010-67180876

北京汉魂图文设计有限公司制作

北京顺义振华印刷厂印刷

新华书店总店北京发行所经销

◆ 开本: 787 × 1092 1/16

印张: 48.75

字数: 1 336 千字

2002 年 1 月第 1 版

印数: 1 ~ 4 000 册

2002 年 1 月北京第 1 次印刷

著作权合同登记 图字: 01 - 2001 - 2038 号

ISBN 7-115-09770-4/TP·2531

定价: 75.00 元

本书如有印装质量问题, 请与本社联系 电话: (010) 67129223



## 内容提要

ASP.NET 是微软公司开发的动态 Web 编程技术活动服务器网页技术 (ASP) 的最新版本, 它不是传统 ASP 的简单升级, 而是一种全新的、令人振奋的 Web 开发技术, 对传统 ASP 做了大量的改进, 以充分利用最新的技术。本书以循序渐进的方式全面介绍了 ASP.NET 的内容, 指导读者从新手逐步成为 ASP.NET 高手。

本书分五部分, 共 23 章。第一部分介绍了 ASP.NET 的基础知识, 包括 ASP.NET 初步、创建 ASP.NET 页面、VB.NET 和 C#、Web 表单、验证 ASP.NET 页面等内容。第二部分介绍了数据操纵技术, 包括创建数据库、使用 ADO.NET 和 XML 文件、文件的读写以及缓存技术等内容。第三部分介绍了应用程序级的主题, 包括业务对象、Web 服务、配置、部署和调试应用程序、确保应用程序的安全等内容。第四部分创建了一个完整的 ASP.NET 应用程序, 并介绍了移动 Web 页方面的知识。最后一部分是附录, 介绍了各种控件的属性、方法和事件以及一些避免错误的技巧。

本书是为新手精通 ASP.NET 而编写的, 通过阅读本书, 初学者将全面掌握 ASP.NET 中的各种技术, 独立地开发出功能强大的应用程序。对于传统 ASP 开发人员, 每章最后的“这不是 ASP”一节将让您了解 ASP.NET 不同于传统 ASP 的地方以及 ASP.NET 的新特性, 从而平滑地从传统 ASP 过渡到 ASP.NET。

# 序 言

ASP.NET 是微软流行的动态 Web 编程技术活动服务器网页 (ASP) 的最新版本,但它远不是传统 ASP 的简单升级。ASP.NET 包括许多新特性、如全新的编程模型和大量全新的工具。传统 ASP 和 ASP.NET 之间有天壤之别,这使得开发人员从传统 ASP 转移到 ASP.NET 时存在很陡的学习曲线,本书将使这种学习曲线变得平坦。

虽然拥有传统 ASP 或其他动态 Web 编程技术方面的经验,将有助于学习 ASP.NET,但这不是必需的。作者在解释每一章中的概念时,都假设读者没有 ASP 方面的经验。但如果读者以前使用传统 ASP 创建过网页,则肯定会对每章的最后一节“这不是 ASP”感兴趣,这些章节重点介绍传统 ASP 和 ASP.NET 之间的差别,非常适合于有传统 ASP 背景的开发人员阅读。

ASP.NET 中的新特性使得设计网页的速度比以前任何时候都更快、更容易。例如,ASP.NET 给开发人员提供了大量的功能强大的 Web 控件,这些控件类似于 HTML 标记,它们提供了非常有用的功能、显示日历、随机显示横幅广告以及显示 HTML 表格(其中包含来自数据库中的数据)。这些 Web 控件使得开发人员只需编写少量的代码,便可以提供丰富的、与 W3C 兼容的 HTML。要了解 Web 控件,请参考第 5 章“Web Forms 初步”。

本书共 23 章,介绍了 ASP.NET 中的许多复杂内容。每一章都包括一些很有用的信息,读者可以立刻将它们用于 Web 应用程序中。鉴于 ASP.NET 包括大量新特性,这种循序渐进的教学方式将有助于 ASP.NET 新手快速进入这种激动人心的新技术的殿堂。

ASP.NET 是一种新的、令人振奋的 Web 开发技术,将给世界带来震撼。其下一代 Web 开发技术使得设计让人印象深刻的网站更容易,设计过程更有趣。如果您打算使用 ASP.NET,您将发现本书提供了宝贵的资源。

祝您编程愉快!

Scott Mitchell

# 前 言

欢迎使用本书。您之所以选择本书，很可能是因为您想学习编写健壮的 Internet 应用程序。您真是选对了。在本书的 23 章中，您将学习 ASP.NET（微软公司开发的基于 Web 的编程框架）各个重要方面的知识：从这些网页的外观到它们如何与操作系统结合在一起。

ASP.NET 是一种让开发人员能够轻松地创建并控制动态网页的技术，它是微软公司的下一代 ASP（作者将 ASP 称之为传统 ASP），对传统 ASP 做了大量的改进，以充分利用最新的技术。使用它，您可以与数据库交互、为访问者定制网页、在移动设备（如手机）上显示网页，从空白开始创建整个电子商务购物网站。

当您阅读本书中以教程方式编写的各章时，您将发现完成这些任务是多么地容易。您不但能够看到范例，了解如何创建这些应用程序（以及其他的应用程序），同时确信这样一点，即在 ASP.NET 方面付出的努力将肯定会得到回报。

## 本书针对的读者

本书是为新手精通 ASP.NET 而编写的，覆盖的读者群非常广。要完全理解本书的内容，读者必须熟悉计算机技术和术语（例如，如何浏览硬盘和 Internet 上的网页）。本书不要求读者有过编程经验（但以前的编程经验肯定会大有帮助），但必须拥有基本的 HTML 知识。

本书同样能让传统 ASP 开发人员受益匪浅。书中不但介绍了如何使用 ASP.NET 完成一些复杂的任务，同时每一章都包括特殊的一节——“这不是 ASP”，这是专门为有 ASP 经验的读者准备的。这些章节讨论了 ASP.NET 与 ASP 之间的区别，并提供了一些例子，帮助读者更为平滑地过渡到 ASP.NET。

最为重要的是，本书是为那些想使用 ASP.NET 的强大功能创建强壮网站（乃至简单的主页）的用户编写的。如果以前只创建过 HTML 网页，则 ASP.NET 将是您的理想选择。它功能强大、易于学习。ASP.NET 的强大功能定将让您大为惊奇。

## 阅读本书必备的知识

阅读本书的唯一要求是了解 HTML 基础知识，并熟悉操作系统。书中的每一章都将提出、定义并解释一些新的主题，因此读者将能够掌握它们，并继续阅读下去。

虽然如此，但编程方面的知识肯定有助于读者阅读本书。如果读者具备开发传统应用程序乃至 Web 应用程序方面的经验，将对书中的许多概念耳熟能详，这肯定会加快学习速度。

另外，读者也不需要具备传统 ASP 方面的经验。对于该旧框架中的一些概念，书中将其视为全新的东西进行解释，以便读者能够学习它们或对它们有进一步的认识。

### 为实现书中的例子，需要安装哪些软件

为完成书中的大部分实例，需要使用 Windows 2000 或带 Service Pack 6 的 Windows NT。有些 ASP.NET 组件可以在 Windows Me/98 或更早的 Windows 版本上运行，但可能存在问题，因此不推荐在这些操作系统上安装 ASP.NET。

另外，还需要安装一个 Web 服务器。该服务器可以是 IIS 5 到 Personal Web Server 的任何 Web 服务器。第 1 章将介绍如何正确地安装该服务器。另外，还需要从微软公司获得 .NET Framework SDK，同样第 1 章也将介绍如何安装该软件开发工具包。

最后，为使用书中的数据库范例，还需要安装某种 OLE DB 兼容的数据库系统，如 Microsoft Access 或 SQL Server。没有这种数据库系统，将无法实现相关的范例。

## 本书的组织方式

本书分五部分，共 23 章。第一部分介绍 ASP.NET 基础知识：ASP.NET 简介、如何创建 ASP.NET 页面以及如何使用 VB.NET 改进页面。另外，还将深入讨论 Web 表单框架（ASP.NET 的有机组成部分，让开发人员的工作更为轻松）。该部分为读者阅读后面的内容打下了基础。

第二部分介绍数据操纵技术（毕竟，大部分人进入 Web 编程领域主要是为了与数据库交互）。读者将学习如何创建数据库、如何在 ASP.NET 中检索和显示数据库以及如何修改数据。读者将学习传统数据库如何与常见的文件、XML 以及高速缓存交互。

第三部分重点介绍大型的、应用程序级的主题，如调试、组件化（Componentizing）和配置。对于创建基于 Web 的完整应用程序而言，这些知识是必不可少的。

第四部分包括两章，讨论了一些对学习 ASP.NET 而言并不太重要的主题，但这些知识对您很有帮助。这部分对一个基于 Web 的应用程序做了全面的分析，同时介绍了如何将 ASP.NET 用于诸如手机和 PDA 等移动设备。

最后一部分是附录，提供了全书涵盖的所有主题的参考，并介绍了一些避免错误的技巧。

本书是按教程的方式组织的，每一章都以前面的章节为基础，因此循序渐进地阅读将会有所帮助，虽然并不一定要这样做。一般而言，首先对主题进行讨论，然后介绍一些范例（其中包括对范例的分析）。在每章的最后都列出了一些常见的问题，同时包含一个小测验（测试您对新知识的掌握程度，请别作弊）和一些自我练习题。它们

都是经过仔细挑选的，以帮助读者巩固所学的知识，丰富使用所学技术的经验，但并不要求读者一定要完成这些测验和练习

## 结束语

祝您在学习这种激动人心的技术时，度过一段美好的时光。使用这种技术，您可以在 Internet 上完成一些了不起的工作，而其中的一些在技术市场上的需求非常旺盛。废话少说，让我们开始学习吧！

作 者

## 目 录

## 第一部分 基础知识

第1章 ASP.NET 初步	3
1.1 Web 的工作原理	3
1.1.1 动态处理技术	4
1.1.2 ASP.NET 的不同之处	5
1.1.3 客户端处理技术	5
1.1.4 ASP.NET 如何将客户机和服务器结合起来	6
1.2 .NET 框架	6
1.2.1 运行阶段通用语言	7
1.2.2 .NET 框架类	8
1.3 安装 ASP.NET	8
1.3.1 安装 Internet 信息服务器	9
1.3.2 安装 .NET 框架 SDK	11
1.4 创建 ASP.NET 页面	12
1.4.1 开发环境	13
1.5 ASP.NET 页面中的元素	14
1.6 ASP 和 ASP.NET 之比较	16
1.6.1 与 ASP 的根本差异	16
1.6.2 编程方面的改进	16
1.6.3 编程方法学方面的差异	17
1.7 总 结	17
1.8 问与答	18
1.9 作 业	18
1.9.1 小测验	18
1.9.2 练习	18
第2章 创建 ASP.NET 页面	20
2.1 一个简单的 ASP.NET 应用程序	20

2.1.1	Web 表单	22
2.1.2	代码声明块	23
2.1.3	代码交付块	24
2.1.4	页面编译指令	25
2.1.5	流程	26
2.1.6	视图状态	27
2.2	编写 ASP.NET 代码和 HTML 代码	28
2.2.1	对代码进行注释	29
2.2.2	跨越多行的代码	30
2.3	应用程序的其他方面	31
2.3.1	再谈 ASP.NET 编译	31
2.3.2	导入名称空间	31
2.4	CLR 和 ASP.NET	33
2.4.1	中间语言	33
2.4.2	执行	33
2.4.3	处理	33
2.4.4	组合体	34
2.4.5	并行执行	34
2.4.6	对 ASP.NET 而言, CLR 意味着什么	34
2.5	ASP.NET 编程语言	35
2.6	重新审视前面的代码	35
2.7	这不是 ASP	36
2.8	总 结	37
2.9	问与答	37
2.10	作 业	38
2.10.1	小测验	38
2.10.2	练习	38
第 3 章	使用 Visual Basic.NET	39
3.1	Visual Basic.NET 简介	39
3.2	变 量	39
3.2.1	数据类型	40
3.2.2	变量的声明	41
3.2.3	变量的命名	42
3.2.4	数据类型转换	43
3.3	数 组	45
3.4	操作符	47
3.5	条件逻辑	48
3.5.1	If 语句	48
3.5.2	Case 语句	50

3.6 循环逻辑	52
3.6.1 While 循环	52
3.6.2 For 循环	53
3.6.3 死循环	54
3.7 分支逻辑	55
3.7.1 子程序	55
3.7.2 函数	57
3.7.3 可选参数	58
3.7.4 事件处理程序	58
3.8 类	61
3.9 使用VB.NET 函数	64
3.10 给未来的VB.NET 高手：到哪里查找参考资料	65
3.11 这不是ASP	65
3.12 总 结	66
3.13 问与答	66
3.14 作 业	67
3.14.1 小测验	67
3.14.2 练习	67
第4章 在C#和VB.NET 中使用ASP.NET 对象	68
4.1 C#简介	68
4.1.1 C#语法范例	68
4.2 对象概述	71
4.2.1 属性	72
4.2.2 方法	72
4.2.3 对象实例	72
4.2.4 静态成员	73
4.3 ASP.NET 对象	74
4.3.1 Response 对象	74
4.3.2 Request 对象	78
4.3.3 HttpCookie 对象	79
4.3.4 Page 对象	82
4.3.5 Session 对象	86
4.3.6 HttpApplication 对象	91
4.3.7 HttpServerUtility 对象	92
4.4 深入学习C#的资源指南	93
4.5 这不是ASP	93
4.6 总 结	94
4.7 问与答	94
4.8 作 业	95



4.8.1 小测验	95
4.8.2 练习	95
<b>第5章 Web 表单初步</b>	<b>96</b>
5.1 表单简介	96
5.2 Web 表单简介	97
5.3 Web 表单编程模型	98
5.3.1 服务器控件	99
5.3.2 服务器控件事件	99
5.3.3 发送 Web 表单	102
5.3.4 保存状态	103
5.3.5 Web 表单的处理顺序	105
5.4 HTML 服务器控件	105
5.5 Web 服务器控件	109
5.5.1 使用 Web 控件	110
5.5.2 即时发送数据	113
5.5.3 Web 服务器控件与 HTML 服务器控件之比较	115
5.6 这不是 ASP	116
5.7 总结	116
5.8 问与答	117
5.9 作业	117
5.9.1 小测验	117
5.9.2 练习	118
<b>第6章 再谈 Web 表单</b>	<b>119</b>
6.1 Web 表单的扩展性	119
6.2 用户控件	119
6.2.1 创建用户控件	120
6.2.2 使用用户控件	124
6.2.3 改进用户控件	127
6.3 自定义控件	128
6.3.1 创建自定义控件	129
6.3.2 使用自定义控件	130
6.3.3 使用属性和状态	131
6.3.4 加入事件	135
6.4 在运行阶段创建控件	139
6.5 这不是 ASP	142
6.6 总结	143
6.7 问与答	143
6.8 作业	144

6.8.1 小测验	144
6.8.2 练习	144
<b>第7章 验证 ASP.NET 页面</b>	<b>145</b>
7.1 有效性验证情形	145
7.2 ASP.NET 有效性验证	149
7.2.1 Validation 控件的工作原理	150
7.3 使用 Validation 控件	154
7.3.1 服务器上的有效性验证	160
7.3.2 禁用有效性验证	162
7.3.3 正则表达式	162
7.4 定制有效性验证	164
7.4.1 错误消息	164
7.4.2 显示有效性验证摘要	165
7.4.3 自定义 Validation 控件	168
7.5 这不是 ASP	171
7.6 总 结	171
7.7 问与答	171
7.8 作 业	172
7.8.1 小测验	172
7.8.2 练习	172
<b>第一部分 复习</b>	<b>173</b>
附加项目 I	173
一个银行业应用程序	173
用户控件	174
登录页面	174
账户页面	177
账单支付页面	180
总 结	182
<b>第二部分 数据存取和处理</b>	
<b>第8章 创建数据库</b>	<b>185</b>
8.1 什么是数据库	185
8.1.1 关键字 (keys) 和约束 (constraints)	187
8.1.2 数据库通信标准	188
8.1.3 何时应使用数据库	188
8.2 创建数据库	188

8.3 结构化查询语言 (SQL)	192
8.3.1 SELECT 语句	193
8.3.2 INSERT 语句	196
8.3.3 UPDATE 语句	196
8.3.4 DELETE 语句	197
8.4 在 ASP.NET 中存取数据	197
8.4.1 存取数据	197
8.5 这不是 ASP	200
8.6 总 结	201
8.7 问与答	201
8.8 作 业	202
8.8.1 小测验	202
8.8.2 练习	202
第 9 章 在 ASP.NET 中使用数据库	203
9.1 ASP.NET 访问数据库简介	203
9.2 DataSet	204
9.2.1 使用 DataSet	205
9.2.2 关系	207
9.2.3 填充 DataSets	208
9.3 数据绑定	209
9.3.1 使用数据绑定	211
9.4 数据绑定控件	215
9.4.1 Repeater 服务器控件	215
9.4.2 DataList 服务器控件	219
9.4.3 DataGrid 服务器控件	224
9.4.4 数据绑定控件小结	230
9.5 这不是 ASP	239
9.6 总 结	240
9.7 问与答	240
9.8 作 业	240
9.8.1 小测验	240
9.8.2 练习	240
第 10 章 与 ASP.NET 通信	242
10.1 ADO.NET 简介	242
10.1.1 ADO.NET 和 ADO 的比较	242
10.1.2 ADO.NET 和 XML	243
10.1.3 ADO.NET 对象模型	244
10.2 再谈 DataSet	245

10.2.1 修改 DataRow 中的数据	247
10.2.2 查看 DataTable 中的数据	248
10.2.3 并发	251
10.3 数据库和 ADO.NET 的交互	251
10.3.1 连接信息	251
10.3.2 OleDbConnection 对象	253
10.3.3 OleDbCommand 对象	253
10.3.4 OleDbDataReader 对象	254
10.3.5 Update、Insert 和 Delete	256
10.3.6 OleDbDataAdapter 对象	257
10.4 在 ASP.NET 中使用 ADO.NET	262
10.5 这不是 ASP	272
10.6 总 结	273
10.7 问与答	273
10.8 作 业	274
10.8.1 小测验	274
10.8.2 练习	274
<b>第 11 章 在 ASP.NET 中使用 XML</b>	275
11.1 XML 简介	275
11.1.1 XML 数据模型	276
11.1.2 XML 模式	278
11.2 在 ASP.NET 中存取 XML	279
11.2.1 读取 XML	279
11.2.2 写 XML	283
11.2.3 验证 XML	285
11.3 XML 文档对象模型	289
11.3.1 装载 XML 数据	290
11.3.2 修改 XML 数据	293
11.4 XML 和 DataSet	296
11.5 这不是 ASP	300
11.6 总 结	300
11.7 问与答	301
11.8 作 业	301
11.8.1 小测验	301
11.8.2 练习	301
<b>第 12 章 应用高级数据技术</b>	302
12.1 高级数据库技术	302
12.1.1 参数化查询	303

12.1.2 存储过程 .....	307
12.1.3 事务 .....	314
12.2 高级 XML 技术 .....	316
12.2.1 XpathDocument .....	317
12.2.2 Xpath .....	320
12.2.3 XslTransforms .....	322
12.3 这不是 ASP .....	326
12.4 总 结 .....	326
12.5 问与答 .....	327
12.6 作 业 .....	327
12.6.1 小测验 .....	327
12.6.2 练习 .....	328
<b>第 13 章 Web 服务器上的文件读写 .....</b>	<b>329</b>
13.1 在 ASP.NET 中使用文件 .....	329
13.2 包含外部文件 .....	329
13.2.1 服务器端包含 .....	330
13.2.2 其他包含 .....	332
13.3 文件访问 .....	332
13.3.1 文件、流、Reader 和 Writer .....	332
13.3.2 查看文件和目录 .....	333
13.3.3 打开文件 .....	340
13.3.4 读文件 .....	342
13.3.5 写文件 .....	346
13.3.6 其他文件和目录操作 .....	346
13.3.7 文件对象小结 .....	347
13.4 隔离存储区 .....	348
13.4.1 创建隔离存储区域 .....	348
13.4.2 访问隔离存储区 .....	349
13.5 这不是 ASP .....	352
13.6 总 结 .....	353
13.7 问与答 .....	354
13.8 作 业 .....	354
13.8.1 小测验 .....	354
13.8.2 练习 .....	354
<b>第 14 章 使用 ASP.NET 改良后的缓存功能 .....</b>	<b>355</b>
14.1 什么是缓存技术 .....	355
14.2 ASP.NET 如何使用缓存技术 .....	356
14.2.1 页面缓存 .....	356

14.2.2	配置缓存 .....	357
14.2.3	输出和数据缓存 .....	357
14.3	如何使用缓存 .....	357
14.3.1	缓存页面输出 .....	357
14.3.2	缓存对象 .....	363
14.3.3	缓存依存关系 .....	368
14.3.4	使用 HttpCachePolicy 类 .....	371
14.4	高效地使用缓存技术 .....	374
14.5	这不是 ASP.NET .....	375
14.6	总 结 .....	375
14.7	问与答 .....	376
14.8	作 业 .....	376
14.8.1	小测验 .....	376
14.8.2	练习 .....	376
第二部分	复习 .....	377
附加项目 2	.....	377
数据添加功能	.....	377
数据库	.....	377
ASP.NET 页面	.....	380
总 结	.....	390

### 第三部分 应用程序级主题

第 15 章	使用业务对象 .....	393
15.1	组件简介 .....	393
15.1.1	业务对象是什么 .....	394
15.1.2	为何使用组件 .....	394
15.1.3	ASP.NET 如何使用组件 .....	395
15.2	创建业务对象 .....	395
15.2.1	为何需要编译 Database 对象 .....	398
15.3	开发业务对象 .....	398
15.4	一个实用的例子 .....	402
15.4.1	一些需要考虑的因素 .....	409
15.5	使用非.NET 组件 .....	409
15.6	这不是 ASP .....	412
15.7	总 结 .....	413
15.8	问与答 .....	413
15.9	作 业 .....	414

15.9.1 小测验 .....	414
15.9.2 练习 .....	414
<b>第 16 章 创建 Web 服务 .....</b>	<b>415</b>
16.1 Web 的工作方式——再访问 .....	415
16.2 Web 服务简介 .....	416
16.2.1 Web 服务方案 .....	417
16.2.2 Web 服务的编程模型 .....	418
16.2.3 用于访问 Web 服务的协议 .....	419
16.2.4 为何使用 Web 服务 .....	420
16.3 创建 Web 服务 .....	421
16.3.1 创建功能 .....	421
16.3.2 启用发现功能 .....	424
16.3.3 WebMethod 属性 .....	424
16.3.4 部署 Web 服务 .....	426
16.4 使用已有的业务对象创建 Web 服务 .....	426
16.5 从服务返回数据 .....	429
16.6 这不是 ASP .....	430
16.7 总 结 .....	430
16.8 问与答 .....	431
16.9 作 业 .....	431
16.9.1 小测验 .....	431
16.9.2 练习 .....	432
<b>第 17 章 使用 Web 服务并确保其安全 .....</b>	<b>433</b>
17.1 使用 Web 服务 .....	433
17.2 通过 ASP.NET 页面使用 Web 服务 .....	435
17.2.1 发现 .....	435
17.2.2 创建代理类 .....	437
17.2.3 实现代理类 .....	440
17.2.4 另一个使用 Web 服务的例子 .....	442
17.3 关于使用 Web 服务的建议 .....	445
17.4 确保 Web 服务的安全 .....	445
17.5 这不是 ASP .....	453
17.6 总 结 .....	453
17.7 问与答 .....	454
17.8 作 业 .....	454
17.8.1 小测验 .....	454
17.8.2 练习 .....	454

第 18 章 配置和部署 ASP.NET 应用程序 .....	455
18.1 ASP.NET 应用程序简介 .....	455
18.1.1 bin 目录 .....	456
18.2 Global.asax .....	456
18.2.1 HttpApplication 类 .....	458
18.2.2 编写 global.asax .....	458
18.3 配置 ASP.NET .....	463
18.3.1 Web.config .....	463
18.3.2 配置段 .....	467
18.3.3 自定义配置 .....	471
18.4 部署应用程序 .....	475
18.4.1 组合体仓库 .....	475
18.4.2 影子组合体 .....	476
18.5 这不是 ASP .....	477
18.6 总 结 .....	477
18.7 问与答 .....	478
18.8 作 业 .....	478
18.8.1 小测验 .....	478
18.8.2 练习 .....	479
第 19 章 将内容和代码分开 .....	480
19.1 为何要将代码和内容分开 .....	480
19.2 Code-behind 表单 .....	481
19.2.1 在 ASP.NET 页面中使用 code-behind 表单 .....	483
19.2.2 在用户控件中使用 code-behind 表单 .....	490
19.3 资源文件和本地化 .....	493
19.3.1 应用程序的本地化 .....	493
19.3.2 将资源包装到文件中 .....	500
19.4 这不是 ASP .....	505
19.5 总 结 .....	506
19.6 问与答 .....	506
19.7 作 业 .....	507
19.7.1 小测验 .....	507
19.7.2 练习 .....	507
第 20 章 调试 ASP.NET 页面 .....	508
20.1 调试简介 .....	508
20.2 Try 和 Catch 语句 .....	511
20.2.1 引发异常 .....	517
20.2.2 何时使用 try 语句 .....	518



20.3 跟 踪 .....	518
20.3.1 页面级跟踪 .....	520
20.3.2 应用程序级跟踪 .....	525
20.4 CLR 调试器 .....	527
20.4.1 使用 CLR 调试器 .....	527
20.5 有关调试方面的建议 .....	530
20.6 这不是 ASP .....	530
20.7 总 结 .....	530
20.8 问与答 .....	531
20.9 作 业 .....	531
20.9.1 小测验 .....	531
20.9.2 练习 .....	532
<b>第 21 章 确保 ASP.NET 应用程序的安全</b> .....	533
21.1 安全基础 .....	533
21.1 Windows 中的安全性 .....	534
21.2 认 证 .....	535
21.2.1 Windows 认证 .....	536
21.2.2 表单认证 .....	539
21.2.3 Passport 认证 .....	545
21.3 授 权 .....	546
21.4 模 拟 .....	549
21.5 这不是 ASP .....	551
21.6 总 结 .....	551
21.7 问与答 .....	552
21.8 作 业 .....	552
21.8.1 小测验 .....	552
21.8.2 练习 .....	552
<b>第三部分 复习</b> .....	553
附加项目 3 .....	553
添加业务对象 .....	553
Web 服务 .....	563
总 结 .....	565

## 第四部分 完整的应用程序和移动 Web 页面

<b>第 22 章 创建一个完整的应用程序</b> .....	569
22.1 需求简介: BananaMobiles .....	569

22.2 设计应用程序 .....	570
22.2.1 数据层 .....	570
22.2.2 前端 .....	571
22.2.3 业务对象 .....	572
22.3 构建 BananaMobile 站点 .....	572
22.3.1 创建数据库 .....	572
22.3.2 业务对象 .....	580
22.3.3 ASP.NET 页面 .....	591
22.3.4 Web 服务 .....	610
22.4 应用程序中还可以改进的地方 .....	614
22.5 这不是 ASP .....	615
22.6 总 结 .....	615
22.7 问与答 .....	616
22.8 作 业 .....	616
22.8.1 小测验 .....	616
第 23 章 创建移动 Web 页 .....	617
23.1 移动 Web 表单是什么 .....	617
23.1.1 移动 Web 表单和 ASP.NET Web 表单的区别何在 .....	618
23.1.2 为何需要使用移动 Web 表单 .....	618
23.2 安装移动 Web SDK .....	619
23.3 移动 Web 表单初步 .....	620
23.3.1 移动表单的工作原理 .....	622
23.3.2 创建一个更合适的界面 .....	623
23.4 开发移动页面 .....	624
23.4.1 添加分页功能 .....	629
23.4.2 设备特定的输出和模板集 .....	630
23.4.3 使用移动设备的功能 .....	633
23.5 这不是 ASP .....	636
23.6 总 结 .....	636
23.7 问与答 .....	637
23.8 作 业 .....	637
23.8.1 小测验 .....	638
23.8.2 练习 .....	638

## 第五部分 附录

附录 A 作业答案 .....	641
第 1 章答案 .....	641

---

小测验 .....	641
练习 .....	641
第2章答案 .....	642
小测验 .....	642
练习 .....	642
第3章答案 .....	644
小测验 .....	644
练习 .....	645
第4章答案 .....	646
小测验 .....	646
练习 .....	647
第5章答案 .....	649
小测验 .....	649
练习 .....	649
第6章答案 .....	653
小测验 .....	653
练习 .....	653
第7章答案 .....	663
小测验 .....	663
练习 .....	663
第8章答案 .....	664
小测验 .....	664
练习 .....	664
第9章答案 .....	665
小测验 .....	665
练习 .....	665
第10章答案 .....	665
小测验 .....	665
练习 .....	666
第11章答案 .....	670
小测验 .....	670
练习 .....	671
第12章答案 .....	674
小测验 .....	674
练习 .....	675
第13章答案 .....	678
小测验 .....	678
练习 .....	679
第14章答案 .....	681
小测验 .....	681

14.8.2 练习 .....	682
第 15 章答案 .....	684
小测验 .....	684
练习 .....	684
第 16 章答案 .....	687
小测验 .....	687
练习 .....	688
第 17 章答案 .....	690
小测验 .....	690
练习 .....	691
第 18 章答案 .....	692
小测验 .....	692
练习 .....	693
第 19 章答案 .....	694
小测验 .....	694
练习 .....	695
第 20 章答案 .....	698
小测验 .....	698
练习 .....	698
第 21 章答案 .....	699
小测验 .....	699
练习 .....	699
第 22 章答案 .....	701
小测验 .....	701
第 23 章答案 .....	702
小测验 .....	702
练习 .....	702
附录 B 常犯的 ASP.NET 错误 .....	704
B.1 ASP.NET 特有的问题 .....	704
B.1.1 Web 表单的问题 .....	704
B.1.2 其他问题 .....	706
B.2 与传统 ASP 不同的地方 .....	706
B.2.1 VBScript 方面的错误 .....	706
B.2.2 传统 ASP.NET 方面的问题 .....	708
附录 C ASP.NET 控件: 属性和方法 .....	709
C.1 HTML 服务器控件 .....	710
C.1.1 HTML 服务器控件都有的属性 .....	710
C.1.2 HTML 服务器控件 .....	711

---

C.2	Web 服务器控件.....	717
C.2.1	通用 Web 服务器控件属性.....	718
C.2.2	ASP.NET Web 服务器控件 .....	720
C.3	有效性验证服务器控件 .....	733
C.3.1	有效性验证服务器控件都有的属性 .....	733
C.3.2	有效性验证服务器控件 .....	734
附录 D	ADO.NET 控件：属性和方法 .....	736
D.1	DataSet 及相关对象 .....	736
D.2	管理提供程序.....	747

# 第一部分

ASP.NET 网络编程案例解密

## 基础知识

第 1 章 ASP.NET 初步

第 2 章 创建 ASP.NET 页面

第 3 章 使用 Visual Basic.NET

第 4 章 在 C# 和 VB.NET 中使用 ASP.NET 对象

第 5 章 Web 表单初步

第 6 章 再谈 Web 表单

第 7 章 验证 ASP.NET 页面



# 第 1 章

## ASP.NET 初步

欢迎参加 ASP.NET 学习之旅。ASP.NET 是微软公司用于创建功能强大的网站（超过了常规的 HTML 页面）的工具。如果您要创建 Internet 应用程序，您真是选对了。

对于开发复杂的网站而言，ASP.NET 是一种健壮的、易于使用的解决方案。使用它，您将能够创建访问者能够以愉快的方式与之交互的网页，而不是仅仅用于显示一些信息。ASP.NET 页面将给访问者带来全新的体验。

本章将介绍以下内容：

- 在客户/服务器模型中，Web 是如何运作的；
- ASP.NET 是什么；
- ASP.NET 如何将客户和服务器组合起来；
- .NET 框架是什么，它与 ASP.NET 有何关系；
- 要创建 ASP.NET 页面，需要什么？如何安装所需的软件；
- 如何创建 ASP.NET 页面；
- ASP.NET 与传统 ASP 之间的差别。

### 1.1 Web 的工作原理

Internet 真是神奇，它让世界各地的人能够通过其计算机相互交流。这种技术带来了许多全新的东西，其中包括电子邮件、即时信息传送和万维网。

最初的网站非常简单。在网络上，关于任何想象得到的主题的 HTML 页面都有，并且这些页面总是有读者。早期的页面是静态的——访问者无法以任何方式与之交互。

Web 快速地演进，新的功能被加入，其中包括图像、表格和表单等。这最终使得访问者能够与网站交互，从而出现了来客登记簿和用户调查等应用。网站开发人员开始在站点中加入一些精巧的花样，如图像翻转和下拉式菜单等。

这实现了交互性，但仍然缺乏真正动态的内容。随后，服务器处理技术被推出，从而可以与数据库交互、对内容进行处理以及通过 Web 确定访问者的分布情况。

ASP.NET 是一种将各种 Web 元素组合在一起的服务器技术，给网站开发人员提供了比以前任何时候都更强大的技术支持。但在深入介绍 ASP.NET 之前，让我们首先介绍一些动态处理技术的工作原理。



### 1.1.1 动态处理技术

**新术语：**Internet使用的是客户/服务器模型。两台计算机协同工作，相互发送信息，以完成一项任务。最常见的情况是服务器（存储了信息的计算机）和客户（需要使用信息的计算机）之间的通信。

客户机向服务器发送请求，要求获得信息；然后服务器将信息发送给请求的客户机，以进行应答。这种方案被称为“请求/应答模型”，它是客户/服务器模型的有机组成部分。

Web 服务器是存储了关于 Web 站点信息（HTML 页面、图像等）的计算机。客户是网站的访问者（具体地说，是访问者的 Web 浏览器）。图 1.1 说明了这一概念。

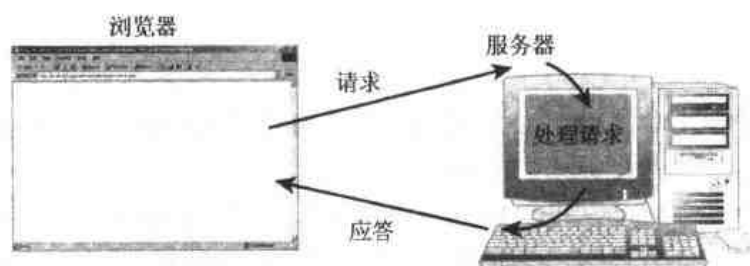


图 1.1 请求/应答模型

虽然这是一种非常神奇的通信和信息分发方式，但它非常简单，也是静态的。它不能提供动态信息，也不能进行动态处理。服务器只是静静地等待客户来请求信息，然后返回存储在其硬盘中的数据，而对发送的东西漠不关心。通常，静态 Web 请求可分为以下 4 步：

1. 客户（Web 浏览器）使用 Web 服务器的 URL（如 [www.microsoft.com](http://www.microsoft.com)）找到它。
2. 客户请求一个页面（如 [Index.htm](#)）。
3. 服务器发送被请求的文档。
4. 客户接收并显示文档。

客户收到信息后，这一过程就结束了。服务器对客户机上发生的情况一无所知，因此服务器和客户机是两台独立的计算机，它们之间只是在请求/响应的过程中进行通信。一旦页面被递送出去，服务器便对所发生的情况毫不关心。

至于服务器处理技术，则有多种方式，包括通用网关接口（CGI）和微软的 ASP，后者现在被称为传统 ASP。在现在的方案中，服务器在发送信息前对其进行检查，它可以接受来自客户的订单。服务器可以返回动态数据，如来自数据库的数据、计算得到的结果以及客户请求的其他任何数据。修改后的工作流程如下：

1. 客户（Web 浏览器）使用 Web 服务器的 URL（如 [www.microsoft.com](http://www.microsoft.com)）找到它。
2. 客户请求一个页面（如 [Index.htm](#)）。
3. 服务器检查被请求的文件，并对其中的代码进行处理。
4. 服务器将处理结果转换为 HTML（如果需要的话）并将请求的文档发送给客户。
5. 客户接收并显示文档。

即使在这种方案中，一旦客户收到页面，这一过程也就结束了。服务器对客户所做的操作一无所知，除非客户再次做出请求。

### 1.1.2 ASP.NET 的不同之处

**新术语：**在服务器和客户之间，存在另一种模型：事件驱动模型。服务器等待客户机发生事件，一旦该事件发生后，服务器将采取行动，执行某些功能。

想象一下图书馆的情况。按照客户/服务器模型，您将向管理员询问信息，后者要么直接给您提供答案或给您指明方向。在事件驱动模型中，图书管理员知道您要做什么，如果您要撰写一份报告，需要获得关于本杰明·富兰克林的信息，管理员将自动给您提供合适的东西；如果您口渴，管理员将给您一杯水；如果您摔倒，管理员将给您包扎。

当然，Web 服务器无法知道您在想什么，但能够对您的行动做出响应。如果您在网页上输入文本，服务器将做出响应。如果您单击图像，服务器也将做出响应。对于创建应用程序而言，这种模型使用请求/响应模型容易得多。ASP.NET 就是以这种方式工作的——它检测用户的操作，并做出响应。

等等！ASP.NET 如何知道访问者计算机上发生的情况呢？服务器如何对客户机上发生的情况做出响应呢？ASP.NET 依赖更为聪明的客户端处理技术来仿真事件驱动模型。

### 1.1.3 客户端处理技术

将一些客户能够理解的编程代码放置在 HTML 页面时，便会进行客户端处理。这些代码只是浏览器能够执行的 HTML。例如，请看清单 1.1。

清单 1.1 可执行的 JavaScript 客户端代码

```
1 <html>
2 <head>
3   <script language="JavaScript">
4     <!--
5       alert("Hello World!");
6     -->
7   </script>
8 </head>
9 <body>
10   Welcome to my page!
11 </body>
12 </html>
```

如果你熟悉客户端脚本语言或 JavaScript，则不会对它感到陌生。当浏览器收到该页面时，将把它视为 HTML。标记<script>向客户指出了页面中包括命令的部分，这被称为脚本。

如果浏览器支持客户端脚本编程语言，它将知道第 5 行是命令它向用户显示一个消息框，其中包括“Hello World!”，如图 1.2 所示。

因此，现在可以在两个地方执行代码：服务器（所有的信息都将以 HTML 的方式返回给客户）和客户。这两个地方执行的代码截然不同，无法交互。表 1.1 列出了客户端代码和服务器端代码之间的区别。



图 1.2 客户端脚本编程语言让您能够与客户进行交互

表 1.1 客户端代码和服务端代码之间的区别

位 置	描 述
客户端	<p>服务器根本不处理这种代码，那是客户的职责所在。这种代码被编写为脚本，命令客户执行某种操作的纯文本命令。</p> <p>通常用于实现动态客户效果，如图像翻转或显示消息框。</p>
服务器端	<p>这种代码只在服务器上执行，在被发送到客户端之前，这种代码生成的任何内容和信息都必须转换为 HTML。</p> <p>这种代码也可以使用脚本语言编写，但 ASP.NET 使用编译后的语言（后面将更详细地介绍）。</p> <p>这种代码用于处理内容并返回数据。</p>

#### 1.1.4 ASP.NET 如何将客户机和服务器结合起来

那么 ASP.NET 如何知道客户机上发生的情况呢？客户端的脚本无法与服务器端的代码交互，但 ASP.NET 机智地避开了这种问题。只有在请求期间，客户才能与服务器通信。

使用客户端脚本，ASP.NET 提供了关于客户端在请求期间发生的情况的信息。回到前面关于图书馆的例子，这是管理员看起来像具有魔力似的，知道您需要什么的原因所在。它拥有一个“间谍网”，在监视着您，虽然您不知道。当您有所行动时，“间谍”将飞快地将发生的情况告知图书管理员。这样，图书管理员便能确定应采取何种行动。

ASP.NET 的“间谍”是客户端脚本，每当客户端发生情况时，将执行客户端脚本，并将信息发送给服务器，就像递交表单将信息发送给服务器一样。浏览器只是一个对情况一无所知的同伙而已，它认为只是在完成自己份内的工作：显示 HTML。因此，客户端虽然不能与服务器端交互，但能够将消息转发给服务器。

因此，ASP.NET 将服务器和客户连接起来，这使得开发人员能够在网页中完成以前不可能完成的工作。开发人员不必将重点放在传递请求和应答，而可以将精力放在构建逻辑上。您可以立刻对用户事件做出响应，而不用等到表单提交后。同时，您能够知道用户界面的结构，并预先对其进行处理。ASP.NET 确实使得开发人员的工作更为轻松。

## 1.2 .NET 框架

ASP.NET 中的“.NET”从何而来呢？它代表.NET 框架：微软公司用于创建应用程序的一组对象和蓝图（blueprint）。.NET 框架提供了 ASP.NET 的底层功能。

在.NET 框架下开发的所有应用程序（包括 ASP.NET 应用程序）都包含一些关键特性，用于确

保其兼容性、安全性和稳定性。让我们来分别看看这些关键特性。

### 1.2.1 运行阶段通用语言

运行阶段通用语言 (Common Language Runtime, CLR) 是一种管理代码执行情况的环境。换句话说, 它运行并维护您编写的任何代码。

以前, 创建应用程序时, 您使用编程语言 (如 Visual Basic) 编写一些代码, 将其编译成计算机能够理解的格式 (由 0 和 1 组成), 然后执行它。由于不同的计算机 (如 PC 和 Macintosh) 使用不同的语言, 因此当您在其他类型的计算机上使用应用程序时, 必须将其重新编译为新计算机使用的语言。在 .NET 框架中, 情况稍微有些不同。

**新术语:** 有了 .NET 框架和 CLR 后, 仍需要编写代码并对其进行编译。不过您将代码编译为一种叫作微软中间语言 (Microsoft Intermediate Language, MSIL) 的语言, 而不是编译成某种计算机能够理解的语言。这种语言以简写方式表示您编写的所有代码。ASP.NET 页面也被编译为 MSIL。编译为 MSIL 时, 应用程序将生成一些元数据 (metadata), 它们是关于应用程序的描述性信息, 指出应用程序能做什么, 归属于哪里等等。

**注意:** 除了 MSIL 和元数据外, 人们还创建了一类新的编程语言 C#、Cobol、Perl 等等。这些语言与已有的语言类似, 但除了可以输出编译后的代码外, 还可以输出 MSIL。

这样, 当您要运行程序时, CLR 将接管工作, 进一步将代码编译成计算机的本机语言。这样, MSIL 便可以用于任何类型的计算机。CLR 懂得许多不同的计算机语言, 并为您完成所有的编译工作。应用程序编译后, 便可以在任何计算机上执行。图 1.3 说明了传统处理过程和 .NET 框架之间的区别。

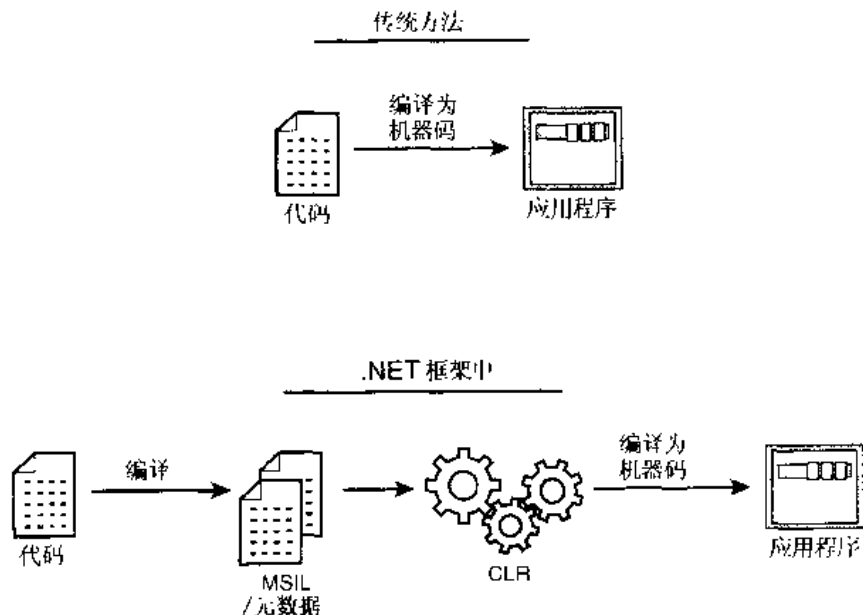


图 1.3 传统应用程序框架和 .NET 框架

**注意:** 如果您熟悉 Java 平台, 便可以发现相似的地方。Java 代码也是由一种被称为 Java 虚拟机 (JVM) 的运行阶段环境翻译和执行的。这使得开发人员只需编写并编译代码, 任何跨平台的问题则由 JVM 来处理。

CLR 使用元数据来确定如何运行应用程序,这使得安装程序非常容易。传统的方法要求将关于应用程序的信息存储在注册表(或一个应用程序信息的中央仓库)中。

不幸的是,每当应用程序被修改(移动了其目录、安装了新组件等)时,注册表将无效,因此应用程序将无法正常运行。使用元数据,根本不需要注册表,所有的应用程序信息都随应用程序文件一起存储,因此所做的任何修改都将自动生效。可以将安装新应用程序看作只是复制一些文件而已。

**新术语:** 在 CLR 中运行的代码被称为管理代码(managed code)。这是因为 CLR 将管理代码的执行,开发人员无需手工联编,因此具有一些优点(如资源管理)。在 CLR 之外运行的代码被称为不可管理的代码(unmanaged code)。

CLR 的功能并不止这些,它还提供了诸如错误处理、安全特性、版本和部署支持等服务以及跨语言集成等服务。这意味着可以选择任何语言来编写 .NET 应用程序,包括 ASP.NET 应用程序。

### 1.2.2 .NET 框架类

**新术语:** .NET 框架中有描述编程对象的蓝图。.NET 框架中的任何东西——ASP.NET 页面、消息框等等——都被视为对象。这些对象被放置在叫作名称空间(namespaces)的逻辑分组中。例如,所有处理数据库的对象都位于名称空间 System.Data 中,所有的 XML 对象都位于名称空间 System.Xml 中,等等。以这种方式对对象进行分组,对构建对象库很有帮助。创建 ASP.NET 应用程序时,将使用名称空间。

**注意:** 同样,这与 Java 也有些类似。.NET 中的名称空间与 Java 中的包(package)类似。

## 1.3 安装 ASP.NET

要运行 ASP.NET 页面,需要在计算机上安装两种软件: Internet 信息服务器和 .NET 框架软件开发工具包(SDK)。这些软件可运行在 Windows 2000 和带 Service Pack 6a 的 Windows NT 4 上。

不安装这些软件,计算机将不知道如何处理 ASP.NET 文件。它将把这种文件看作是无法识别的文件,并询问您应使用哪种应用程序来打开它们,如图 1.4 所示。

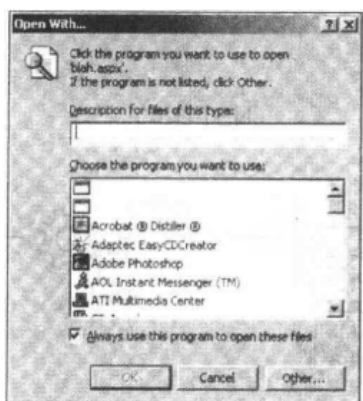


图 1.4 不安装 IIS 和 .NET 框架 SDK, 计算机将不知道如何处理 ASP.NET 文件

还记得吗, ASP.NET 是一种服务器端技术,这意味着需要在计算机上安装 Web 服务器。一旦您安装并运行 Web 服务器后,访问者便可以通过 Internet 向您的计算机请求网页。

然而, ASP.NET 页面还有一些其他的工作需要处理, 因为 Web 服务器无法理解它们。另外, 您还需要 ASP.NET 框架 SDK, 它提供运行 ASP.NET 页面的功能, 并提供 .NET 对象和类。

首先介绍如何安装 Web 服务器, 然后再讨论 .NET SDK。

### 1.3.1 安装 Internet 信息服务器

Internet 信息服务器 (IIS) 是微软公司的专业 Web 服务器, Windows 2000 捆绑了该服务器, 而对于 Windows NT Server, 则可以单独下载。如果您运行的是 NT Server, 可以从 <http://www.microsoft.com/msdownload/ntoptionpack/askwiz.asp> 免费下载 NT Option Pack, 这样便可以使用 IIS 4.0 了。如果运行的是 Windows 2000, 则其中已经自带了 IIS 5.0。

要安装 IIS 5.0, 请执行“开始/设置/控制面板/添加或删除应用程序”, 然后选择“添加/删除 Windows 组件”, 出现如图 1.5 所示的窗口, 让您添加可选的 Windows 组件。

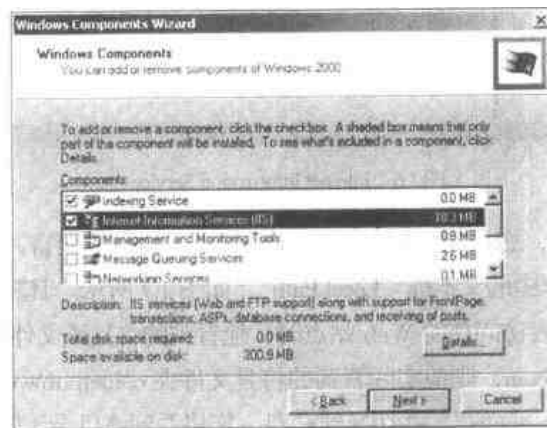


图 1.5 安装和删除 Windows 2000 组件

选中复选框“Internet Information Services (IIS)”, 然后单击“Next”按钮, 或者单击“Detail”按钮, 以选择 IIS 中的各个组件, 如 FTP 或 SMTP 服务。使用默认选项就可以了, 但全部安装它们也不会有什么害处。单击“Next”按钮后, Windows 2000 将收集一些信息并开始安装。

祝贺您! 您现在已经在计算机上安装了一个 Web 服务器! 打开浏览器, 并输入 <http://localhost>, 这样便可以打开您的网站。出现的页面是 IIS 创建的默认页面。也可以通过输入 <http://computename> 来访问您的服务器, 其中 `computename` 是您的计算机的名称。如果不知道您的计算机的名称, 使用 `localhost` 即可。

注意: 为网站申请域名 (如 [www.Mysite.com](http://www.Mysite.com)) 是一个完全不相关的过程, 它超出了本书的范围, 更详细的信息, 请访问 Network Solutions ([www.networksolutions.com](http://www.networksolutions.com))。

现在让我们进入 Internet 服务管理器 (Internet Services Manager, ISM), 以配置 IIS 的设置。选择“开始/设置/控制面板/管理工具/Internet 服务管理器”, 打开如图 1.6 所示的应用程序

提示: 有一个运行 ISM 的快捷方式, 它位于“Start/Programs/Administrative Tools”中, 如果找不到“Administrative Tools”选项, 可在任务栏上单击鼠标右键, 并选择“Properties/Advanced”, 在窗口底端弹出的菜单中, 可以选择显示“Administrative Tools”。

该应用程序显示 Web 站点以及 FTP 和 SMTP 站点 (如果安装了它们的话) 的目录结构。展开“Default Web Site”, 将可以看到其中的一些目录和文件。在“Default Web Site”上单击鼠标右键, 并选择“Properties”。

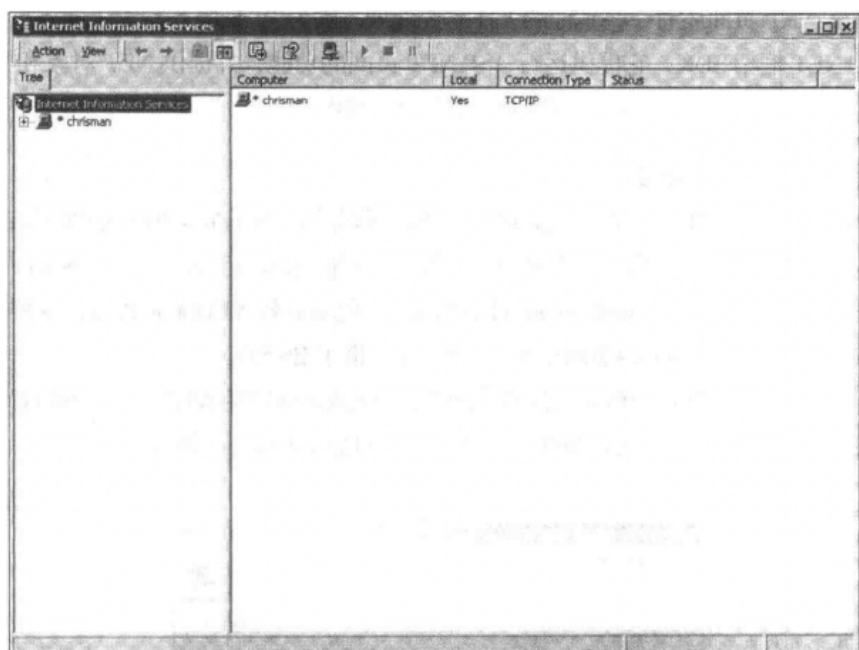


图 1.6 Internet Information Service

有许多选项可以设置，您可能需要花费大量的时间。就目前而言，您只关心选项卡“Home Directory”。请注意该对话框中的文本框“Local Path”，如图 1.7 所示。其默认值为 `c:\inetpub\wwwroot`（也叫根文件夹）。当访问者访问您的 Web 站点时，他看到的将是该文件夹中的内容。因此，如果访问者连接到 `www.yoursite.com`，则他实际看到的将是文件夹 `c:\inetpub\wwwroot` 中的所有内容。

事实上，进入“My Computer”，并切换到 C 盘，您将看到该目录（如果以前没有的话）。您所有的 ASP.NET 文件都将放置在这里。

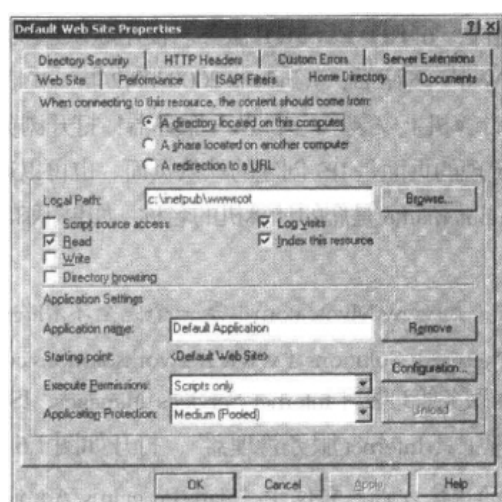


图 1.7 查看默认 Web 站点的“Home Directory”属性

**新术语：**单击“Cancel”按钮，返回到 ISM。其中的所有目录都被称为虚拟目录（virtual directories）。如果其中没有任何目录，也不用担心，因为您马上就会创建一个。虚拟目录是计算机上的一个文件夹，可以通过 Web 站点访问，就像它位于 `c:\inetpub\wwwroot` 下面一样。

要创建虚拟目录，请右击“Default Web Site”，然后选择“New/Virtual Directory”，打开如图 1.8 所示的向导。单击“Next”，并输入一个别名，这将是访问者访问该文件夹时使用的名称。输入别名后，单击“Next”按钮，然后在接下来的一个对话框中，选择计算机上的一个目录，这可以是计算机上任何地方的任何一个目录。接下来的一个对话框让您选择一些选项，但就目前而言，只需单击“Next”按钮，然后单击“Finish”按钮即可。

现在，应该可以在 ISM 中看到一个条目，其别名与您刚才输入的相同。假设您输入的别名是 images，则可以在 Web 浏览器中输入 <http://localhost/images>，来查看该目录的内容。

在大多数情况下，虚拟目录都将放置在根文件夹（`c:\inetpub\wwwroot`）下。有趣的是，实际的目录可以位于计算机的任何地方，但访问者访问它时，就像它位于根文件夹中一样。虚拟目录对 ASP.NET 而言至关重要，在接下来的几章中您将知道这一点。



图 1.8 虚拟目录向导让您轻松地创建虚拟目录

### 1.3.2 安装.NET 框架 SDK

.NET 框架 SDK 包含使得 ASP.NET 页面能够运行的工具和应用程序，其中包括本章前面的“.NET 框架”一节中介绍的 CLR。

SDK 可以从 [www.microsoft.com/.NET](http://www.microsoft.com/.NET) 免费下载，但它超过 100MB，使用 56K 的调制解调器下载，需要 6 个多小时。您也可以订购包括 SDK 的 CD-ROM，其费用非常低。

拥有 SDK 的拷贝后，便可以运行安装程序。这可能需要一段时间，因为 Windows 将对文件进行解压缩并收集信息（见图 1.9）。Windows 检查完您的计算机后，将打开一个与图 1.10 类似的窗口。

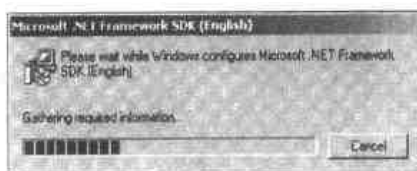


图 1.9 安装.NET 框架 SDK

单击“Next”按钮，并接受协议；然后再次单击“Next”按钮，并选择要安装的组件；再次单击“Next”按钮，并选择安装目录，然后单击“Next”按钮，以安装.NET 框架 SDK。安装好后，桌面上将出现一个打开.NET Framework SDK Overview 的快捷方式，单击它，以查看更多关于 SDK 的信息。该快捷方式包含一个到.NET 框架 SDK 文档的链接，因此务必检查它。



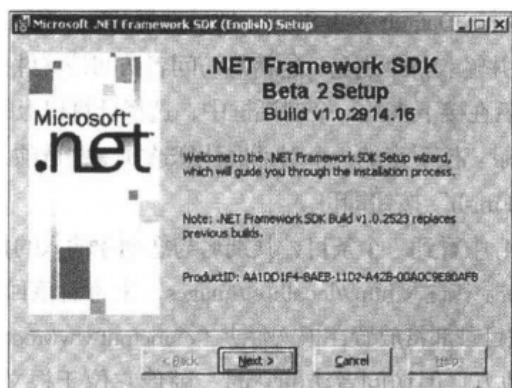


图 1.10 .NET 框架 SDK 安装屏幕

系统将在 `c:\inetpub\wwwroot` 文件夹中创建一个叫作 `Mypage.aspx` 的文档（`.aspx` 意味着这是一个 ASP.NET 文件，更详细的信息，请参考下一节）。双击该文件，将出现如图 1.4 所示的窗口，原因何在呢？

还记得吗，ASP.NET 是一种服务器技术，这意味着它需要通过 Web 服务器才能运行，因此 ASP.NET 文件必须通过服务器（如 IIS）才能运行。打开浏览器，并输入 `http://localhost/Mypage.aspx`，将可以看到该空白的 ASP.NET 页面。

**警告：**在浏览器中，单击“File”菜单，并选择“Open”选项，然后选择 ASP.NET 文件，同样不能打开这种页面。因为这是试图在浏览器中打开文件，而不是通过 Web 服务器来打开。必须使用 `http://localhost`，通过服务器来打开页面。这是初学者常犯的错误。

至此，我们已为运行 ASP.NET 页面做好了准备。

## 1.4 创建 ASP.NET 页面

和 HTML 文件一样，ASP.NET 页面也是纯文本的。安装并运行 Web 服务器和 .NET SDK 后，便可以在任何编辑器中轻松地创建 ASP.NET 页面。

ASP.NET 页面的扩展名为 `.aspx`，因此希望服务器将其作为 ASP.NET 页面解释的任何文件，都必须以 `.aspx` 结尾，如 `default.aspx`。让我们首先创建一个简单的文件。打开记事本（或其他编辑器），并输入清单 1.2 中的代码（首先不要考虑其含义，接下来将解释它们）。

### 清单 1.2 第一个 ASP.NET 页面

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         lblMessage.Text = "Welcome to ASP.NET!"
6     end sub
7 </script>
8
9 <html><body>
10     <asp:Label id="lblMessage" runat="server"/>
```

```
11 </body></html>
```

在文件夹 `c:\inetpub\wwwroot` 中新建一个目录 `day1`，并将该文件保存到该目录中，文件名为 `listing0102.aspx`。该页面只是向访问者显示欢迎信息。打开 Web 浏览器，并使用 `http://localhost/day1/listing0102.aspx` 来访问该文件。您看到的窗口将如图 1.11 所示。

提示：您将为本书的每一章新建一个目录，其中每个目录都位于文件夹 `c:\inetpub\wwwroot\tyaspnet21days` 中。第 2 章的 ASP.NET 页面将保存在文件夹 `c:\inetpub\wwwroot\tyaspnet21days\day2` 中，在浏览器中，可以使用 `http://localhost/tyaspnet21days/day2` 访问该目录。这样，文件将易于找到。

别忘了，浏览器只能理解 HTML 代码，在窗口中单击鼠标右键，并选择“View Source”，将看到如清单 1.3 所示的代码。

### 清单 1.3 ASP.NET 页面中的 HTML 代码

```
1 <html><body>
2   <span id="lblMessage">Welcome to ASP.NET!</span>
3 </body></html>
```

`listing0102.aspx` 中的其他代码（见清单 1.2）到哪里去了呢？ASP.NET 将该代码编译为 MSIL，然后被 CLR 编译为机器语言，并执行，执行结果如清单 1.3 所示。ASP.NET 将其所有的输出转换为 HTML，因为浏览器只懂 HTML 语言。

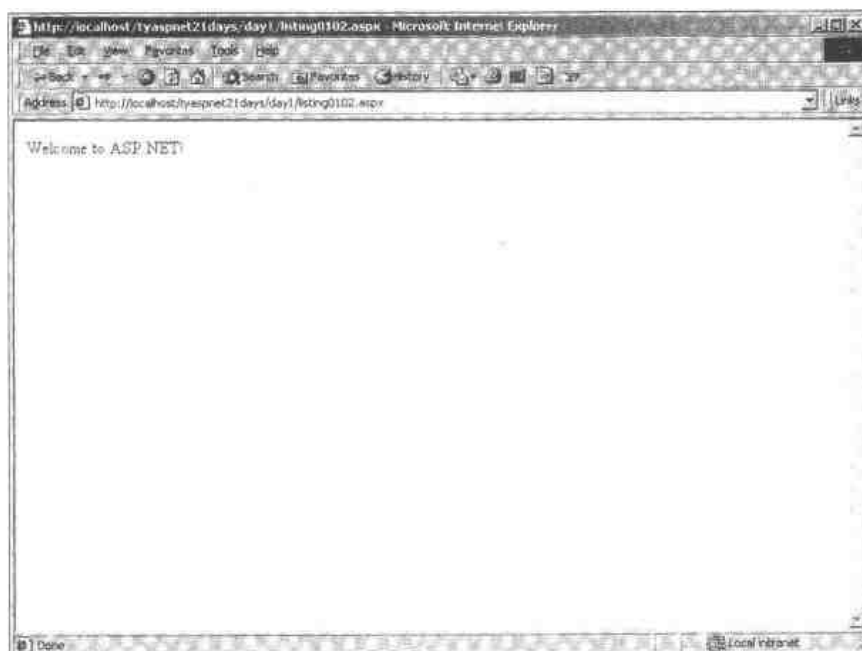


图 1.11 第一个 ASP.NET 欢迎页面

#### 1.4.1 开发环境

也许您很喜欢记事本，但对于创建 ASP.NET 页面而言，它并不是一种理想的应用程序。它简单，易于使用，但没有提供可简化 ASP.NET 开发工作的特性。

Microsoft Visual InterDev 是另一个常用的编辑器，它让用户能够管理整个 Web 站点，并提供了诸如创建和删除虚拟目录、使用数据库以及拖放 HTML 组件的特性。它甚至能够以不同的颜色显示

ASP.NET 代码，使之易于阅读。

**注意：**旧版本的 Visual InterDev (6.0 和以前的版本) 也可用，但它们不支持 .NET 框架，因此有些特性无法正确运行。例如，旧版本的 Visual InterDev 不会以不同的颜色显示 ASP.NET 代码。

另一种常用的环境是 Microsoft FrontPage。这是一种可视化工具，让您不用编写 HTML 代码，便可以创建网页。不幸的是，它不能为您编写 ASP.NET 代码，因此您必须手工编写。

就本书创建的网页而言，使用记事本便可完成，因此选择什么样的开发环境，完全取决于您。Visual InterDev 和 FrontPage 的一些特性也许很有用，但它们没有记事本简单易用。也有一些非微软公司的编辑器可以使用，如 HoTMetaL，但其中的许多编辑器还不支持 .NET 框架，因此可能不如微软公司的编辑器那样好用。

## 1.5 ASP.NET 页面中的元素

让我们来看一个典型的 ASP.NET 页面，如清单 1.4 所示。该页面向用户显示一条消息和一个表单，让用户输入其姓名。将用户单击“Submit”按钮后，将看到一条定制的欢迎消息。

**清单 1.4 与用户交互**

```

1  <%@Page Language="VB" %>
2
3  <script runat="server">
4      Sub tbMessage_Change(Sender As Object,E As EventArgs)
5          lblMessage.Text = "Hello" + lblMessage.Text
6      End Sub
7  / script>
8
9  <tml><body>
10 <font size="5">Sam's Teach Yourself ASP.NET in 21 Days:
11 Day 2</font><hr><p>
12 <%Response.Write("Our First Page<p>")%>
13
14 <form runat="server">
15     Please enter your name:
16     <asp:textbox id="tbMessage"
17         OnTextChanged="tbMessage_Change"
18         runat=server/>
19     <asp:button id="btSubmit" Text="Submit"
20         runat=server/><p>
21     <asp:label id="lblMessage" font size="20pt"
22         runat=server/>
23 </form>
24 </body></html>

```

将该页面保存到目录 `c:\inetpub\wwwroot\tyaspnet21days\day1` 中，文件名为 `listing0104.aspx`。在浏览器中输入 `http://localhost/tyaspnet21days/day1/listing0104.aspx` 以查看该文件。在文本框中输入您的姓名，并单击“Submit”按钮，将看到一条向您问好的消息，如图 1.12 所示

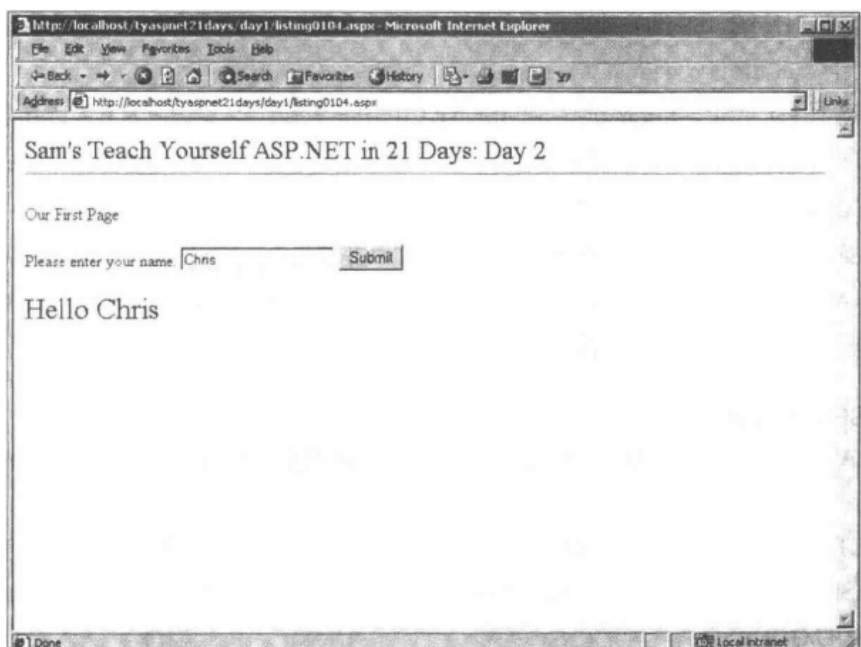


图 1.12 一个显示个性化欢迎消息的简单 ASP.NET 页面

**分析：**该页面包含了 ASP.NET 页面中最常见的元素。第 1 行中的编译指令 `<%@ Page %>` 为 ASP.NET 页面提供了编译时使用的特定信息。在这个例子中，是告诉 ASP.NET 您将使用 Visual Basic 来编写代码

**新术语：**第 3-7 行包含一个名叫代码声明块（code declaration block）的代码块，这与本章前面的“客户端处理”一节中介绍的客户端代码类似，但其中有一个新标记 `runat="server"`，这是 ASP.NET 用来处理其页面的代码，也是您控制所有功能的地方。这些代码也会被编译为 MSIL

**提示：**如果您熟悉客户端编程技术，您肯定知道：在 HTML 中，`<script>` 块通常被放置在标记 `<head>` 和 `</head>` 之间。这些代码块可以放置在任何位置，将其放置在最前面可以将 ASP.NET 代码和 HTML 代码分开。

**新术语：**从第 9 行开始是 HTML 代码。这是您要发送给浏览器的内容（和 ASP.NET 代码的输出一起）。第 12 行以 `<%` 开始，这被称为代码交付块（code render block），其中包含 ASP.NET 用来生成输出的其他指令。在这个例子中，是告诉 ASP.NET 将“`Our First Page`”写到浏览器中。代码交付块不会被编译，因此效率没有代码声明块高。在本书中，将尽可能少用。

**新术语：**在第 14 行，包括另一个传统的 HTML 元素，然后又是一个 `runat="server"`，指定该标记后，表单将是一个 Web 表单（Web form）。通过 ASP.NET，可以监视 Web 表单中的任何东西（还记得关于图书管理员及其间谍的比喻吗？）。

**新术语：**在第 16、19 和 21 行，包括一些与 HTML 元素类似的新元素。它们被称为 Web 控件，其功能与 HTML 元素类似，但具备一些只有 ASP.NET 才能使用的其他功能。注意，每一个元素中都包含 `runat="server"` 行。

这是您第一次查看 ASP.NET 页面，正如您看到的，其代码很容易理解

## 1.6 ASP 和 ASP.NET 之比较

ASP.NET 是对传统 ASP 彻底改造后的结果，因此在创建 Web 应用程序方面，它提供了完全不同的方法。

如果您熟悉传统 ASP，将高兴地看到 ASP.NET 的功能更强大得多，它为开发人员创建 Web 应用程序提供了更有效的方法。接下来的几节将对 ASP 和 ASP.NET 之间的差别做一概要性的讨论。

**提示：**后续章节中的“这不是ASP”将进一步讨论这些差别。

**注意：**虽然ASP.NET和ASP完全不同，但它们可以并肩运行。也就是说，Web服务器可以同时为传统ASP和ASP.NET网页提供服务。安装ASP.NET引擎后，不需要重写以前的ASP网页（虽然您可能想这样做，并利用ASP.NET中改进后的功能）。

### 1.6.1 与 ASP 的根本差异

传统的 ASP 是在 Windows 操作系统和 IIS 的基础上构建的，它总是一个独立的实体，因此功能有限。

而在 .NET 框架下，ASP.NET 是操作系统的一个有机组成部分。它共享了传统应用程序使用的许多对象，ASP.NET 还可以使用所有 .NET 对象。不像 ASP 那样只能使用 6 个继承对象，ASP.NET 可以使用大量的对象。

ASP 也使得客户和服务是两个独立的实体，一旦 ASP 在服务器上完成其工作后，便将 HTML 传递给客户，然后一切都结束了。ASP.NET 通过机智地使用客户端和服务端代码，将客户和服务端结合在一起，这一切对开发人员而言都是不可见的。与 ASP 开发使用的请求/响应模型相比，现在的 Web 开发与传统应用程序开发更类似。

另外，ASP.NET 代码是编译后的，而传统 ASP 使用的则是解释型脚本语言。使用编译代码意味着性能必然比 ASP 应用程序高。这使得 ASP.NET 页面更像传统应用程序，纵观本书，都将以这种方式看待 ASP.NET 页面。

由于上述（以及其他）方面的改进，ASP.NET 给开发人员提供了大量的功能。随着本书触及一些具体的主题，您将了解 ASP.NET 和 ASP 之间的更多差别。

### 1.6.2 编程方面的改进

除了上述这些根本性差别外，ASP.NET 还在编程方面做了许多改进，包括高速缓存技术、代码编译以及更简单、更安全等。

也许 ASP.NET 所做的最大改进是易于部署。正如前面讨论过的，元数据存储了应用程序的所有必要的信息，因此不再需要注册 Web 应用程序和 COM 对象。部署传统 ASP 应用程序时，需要复制合适的 DLL，并使用 REGSVR32.EXE 注册组件。对于 ASP.NET，您所需要做的只是复制 DLL 文件，其他的工作由 .NET 来完成。

在传统 ASP 中，会话状态（session state）是一个非常重要的概念，它指的是自动确定（主要是通过 cookie）请求是否来自同一个客户的能力。这种会话管理使得易于跟踪用户及其操作，因此易于构建的购物车和数据滚动技术应运而生。然而，随着 Web 站点使用多个服务器（一组服务器处理同一个 Web 站点），开发人员开始发现了基于 ASP 会话管理的局限性。也就是说，会话无法跨服务

器转移。

在 ASP.NET 中,会话管理更为容易,功能更为强大。ASP.NET 通过提供可跨服务器扩展的内置会话支持解决了这种问题。它还提高了可靠性,即使服务器崩溃,也不会遭到破坏,并能够与不支持 cookies 的浏览器协同工作。

在传统 ASP 中,几乎所有的代码都是在代码交付块中(即标记<%...%>内)执行的,在 ASP.NET 中,这类代码不会被编译,同时也不提倡频繁使用它。您可以使用代码声明块,这种代码将被编译,因此性能更高。使用这种代码块还可以避免代码和 HTML 交织在一起,这种情况使得 ASP 页面难以阅读。代码声明块可以放在页面的开始位置,使之与其他 HTML 分开,同时还可以控制页面的视觉效果(参见清单 1.4)。

在本书的后面,您将了解更多的编程方面的改进。您将发现,ASP 中缺少的任何东西,在 ASP.NET 中都实现了,并易于使用。

### 1.6.3 编程方法学方面的差异

由于 ASP.NET 以传统 ASP 无法实现的方式将服务器和客户结合在一起,因此开发 ASP.NET 应用程序需要一种更直观的方法。开发人员不再需要记住用户信息或请求输入变量——这些工作都由 ASP.NET 处理。开发人员可以将重点放在如何对用户的操作做出响应,而不用担心请求/响应模型的细节。

另外,ASP.NET 完全是面向对象的。传统 ASP 竭力想引进面向对象编程(OOP)的概念,但没有成功,因为它是一种完全不同的编程方法。习惯 OOP 特性的开发人员将乐于使用 ASP.NET,而那些不习惯 OOP 特性的开发人员将发现它易于学习,使用起来非常直观。

不过,ASP 开发人员也不必失望。虽然 ASP.NET 做了很大的修改,但以前您好不容易学会的一些技巧仍然管用。事实上,您将发现,在大多数时候,ASP.NET 使得工作更为简单。ASP.NET 向 Internet 编程迈出了合乎逻辑的一步。

## 1.7 总 结

本章介绍了 .NET 框架和 ASP .NET 的基本概念,后面的章节将介绍 ASP .NET 的强大功能及其如何使 Web 开发更高效、更健壮。

为处理 ASP.NET 页面,必须安装 Web 服务器(如 Internet 信息服务器)和 .NET 框架 SDK。它们提供了执行 ASP.NET 页面所需的工具和结构。ASP.NET 页面必须通过 Web 服务器进行处理,否则无法运行。

.NET 框架是一组蓝图和对象,在这种框架下开发的应用程序将被编译为 MSIL,并生成描述应用程序的元数据。CLR 将 MSIL 编译成机器代码,并使用元数据协助执行应用程序。

本章介绍了一些典型的 ASP.NET 页面,介绍了 ASP.NET 页面中一些最常用的元素,包括编译指令<%@ Page %>、代码声明块和代码交付块以及 Web 表单等。这些元素是构建复杂 ASP.NET 页面的基础。

最后,对 ASP.NET 和其前身 ASP 做了比较。ASP.NET 为构件 Web 应用程序提供了好得多的框架,使用起来更为容易、直观。另外,在编程方面也做了许多改进。但许多传统 ASP 的编程概念在 ASP.NET 中仍然适用,因此 ASP 开发人员向 ASP.NET 转移时不会遇到什么问题。

在本章中,最重要的一点是,ASP.NET 页面是一种服务器技术,让您能够创建动态 Web 页面。

下一章您将开发第一个 ASP.NET 应用程序，然后探讨每个元素正常运行的原因所在。另外，您将进一步了解.NET 框架及其如何与 ASP.NET 协同工作。

现在，先睡一会儿，做一个关于 ASP.NET 页面的美梦吧！

## 1.8 问与答

问：.NET 框架类是什么？

答：类是定义对象的蓝图。包含对象能够做什么，对象的属性为何等信息。对象是可以操纵的东西——可感知的，能够执行操作的东西。

.NET 框架类可以是.NET 中的任何类。例如，存在一个定义 ASP.NET 页面的行为和外观的类，而 ASP.NET 页面是可以操纵的对象。

问：ASP.NET 是一种编程语言吗？

答：不是。ASP.NET 只是一种用于创建交互式网页的框架。就像一本书只是容纳文字的容器，作者使用英语（或其他语言）来编书。同样，您使用诸如 Visual Basic 或 C#等语言在 ASP.NET 页面中编写代码，这些代码使事件得以发生，但 ASP.NET 提供了代码执行操作和构建网页的框架。

## 1.9 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 1.9.1 小测验

1. 将数据发送给浏览器之前，ASP.NET 必须执行何种操作？
2. CLR 表示的是什么？
3. 元数据是什么？
4. 与 ASP 相比，ASP.NET 在编程方面做了哪三方面的改进。
5. ASP.NET 如何与客户机和服务器协同工作？
6. 为何不能双击 ASP.NET 文件来运行它？
7. 虚拟目录是什么？
8. 如果创建一个名叫 hello.aspx 的文件，并将其保存到目录 c:\inetpub\wwwroot\tyaspnet21days\MyApplication 中，如何从 Web 浏览器中访问该文件。
9. 代码声明块和代码交付块之间有何区别？

### 1.9.2 练习

1. 下面的 ASP.NET 页面的输出将是什么？为使用户能够访问它，应该将它保存在什么位置？

```
1 <%@ Page Language='VB' %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         lblMessage.Text = "This is my first exercise!"
6     end sub
7 </script>
```

```
5
9 <html><body>
10   <asp:Label id="lb.Message" runat="server" >
11 </body></html>
```



## 第2章

# 创建 ASP.NET 页面

前一章为探索 ASP.NET 做好了准备。您了解了新的 ASP.NET 框架以及如何使之在服务器上运行，同时了解了 ASP.NET 的背景以及 Web 的历史。知道如何运行 ASP.NET 后，便可以着手学习开发技术了。

您已经知道 ASP.NET 页面的组成元素，本章将进一步剖析 ASP.NET 页面，并介绍其底层的東西。ASP.NET 到底是如何运行的，代码的含义是什么？同时，您还将动手创建一些代码，体验开发 ASP.NET 页面的工作。

本章重点介绍 .NET 框架在底层所做的工作，让您对这种新的体系结构的工作原理有一个全面的了解。这将使您在以后开发 ASP.NET 应用程序时容易得多。

如果对本章介绍的编程概念不熟悉，也不用担心。第3章“使用 Visual Basic.NET”和第4章“在 C#和 VB.NET 中使用 ASP.NET 对象”将深入介绍这些主题。本章只是热热身而已。

本章将介绍以下内容：

- 简单 ASP.NET 页面的组成部分；
- ASP.NET 页面的工作原理；
- 如何区分 ASP.NET 代码和 HTML；
- 一些 ASP.NET 代码编写语法——注释和换行符；
- CLR 是如何与 ASP.NET 协同工作的；
- 各种 ASP.NET 编程语言。

### 2.1 一个简单的 ASP.NET 应用程序

让我们再来看看上一章的代码清单。清单 2.1 列出了一个简单页面的代码，它获取用户的输入，并打印一条“Hello”的消息。该页面中的许多有趣的特性将在本书后面深入介绍，本章只对它做一大概的介绍。

**清单 2.1 第一个 ASP.NET 页面**

```
1 <%@ Page Language="VB" %>
2
3<script runat="server">
```

```
4 Sub tbMessage_Change(Sender As Object, E As EventArgs)
5     lblMessage.Text = "Hello " + tbMessage.Text
6 End Sub
7 </script>
8
9 <html><body>
10     <font size="5">Sam's Teach Yourself ASP.NET in 21 Days:
11     Day 2</font><hr><p>
12     <% Response.Write("Our First Page<p>") %>
13
14     <form runat="server">
15         Please enter your name:
16         <asp:textbox id="tbMessage"
17             OnTextChanged="tbMessage_Change"
18             runat=server/>
19         <asp:button id="btSubmit" Text="Submit"
20             runat=server/><p>
21         <asp:label id="lblMessage" font-size="20pt"
22             runat=server/>
23     </form>
24 </body></html>
```

详细讨论该页面之前，首先测试一下该页面。在文本框中输入您的姓名，并单击“Submit”按钮。在文本框的下面将以较大的字体显示“Hello[您的姓名]”，如图2.1所示。注意，单击“Submit”按钮后，您输入的文本仍然显示在文本框中，本章后面的“视图状态”一节将介绍其原因所在。

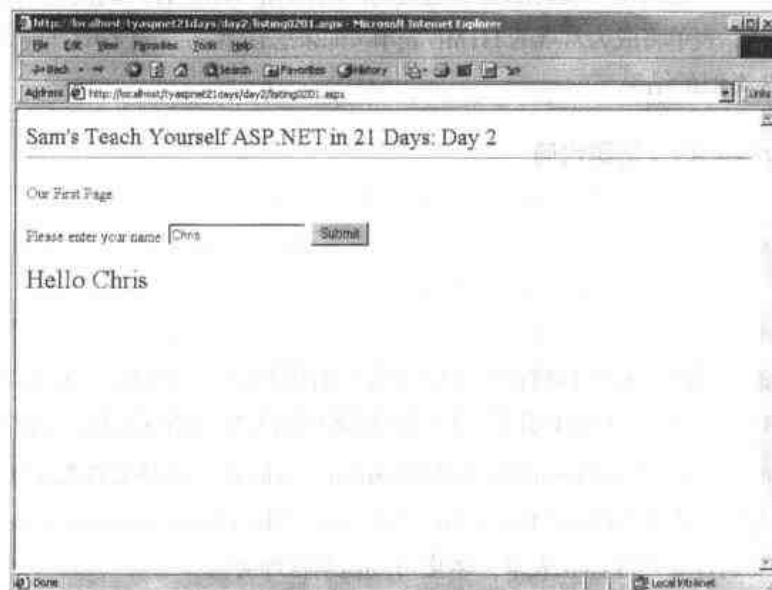


图2.1 清单2.1生成的页面

### 2.1.1 Web 表单

如果您熟悉常规的 HTML，将不会对该页面感到陌生。第 9 行包含开始标记<html>和<body>，其对应的结束标记位于清单的最后。表单中使用了一些新标记，让我们看看这一部分，如清单 2.2 所示。

**清单 2.2 清单 2.1 中的表单部分**

```

14 <form runat='server'>
15     Please enter your name:
16     <asp:textbox id='tbMessage'
17         OnTextChanged='tbMessage_Change'
18         runat=server/>
19     <asp:button id='btSubmit' Text='Submit'
20         runat=server/><p>
21     <asp:label id='lblMessage' font-size='20pt'
22         runat=server/>
23 </form>

```

**新术语：**注意，代码中并未指定表单的行为。通常应指定一种行为，告知 HTML 表单如何处理用户的输入——例如，对其进行处理或将其输入到数据库中。如果未指定行为，则表单将恢复到原来的情况。这种表单被称为回送表单（postback form），因为它回送（post back）给自己。这样，您便可以在页面中包含一些处理输入的 ASP.NET 代码。

ASP.NET 广泛地使用回送表单来处理表单中所有的用户输入。随着您编写更多的页面，您将对这种表单非常熟悉。

您注意到的第一个新东西是表单标记中的关键字“runat”，通过指定属性“server”，您告诉 ASP.NET，您希望在服务器上跟踪该表单，前一章介绍过，服务器拥有间谍，它们将客户端发生的情况报告给服务器。上述关键字使得服务器通过访问者的输入知道客户端发生的情况。

没有这项属性，表单将成为常规的 HTML 表单。如果您在 Web 浏览器中查看该文件的源代码，将得到与清单 2.3 类似的结果。

**清单 2.3 index.aspx 的源代码**

```

1 <form name="ctrl1" method="post" action="index.aspx"
2     id="ctrl1">
3     <input type="hidden" name="__VIEWSTATE"
4         value='YTB6LTEwNzAyOTU3Njc5fX1949e1355cb' />

```

是不是与您输入的不一样？这将在本章后面的“视图状态”一节中进一步进行解释。现在，只需知道这些 HTML 标记是 ASP.NET 引擎为帮助跟踪客户端发生的情况而插入的即可。

**警告：**假如您好奇，而在<form>标记中指定runat=“client”，这样做将无法生成常规的HTML表单。事实上，这样做将引起错误“Parser Error Message: The Runat attribute can only have the value “server””，如果要创建常规HTML表单，不使用runat属性即可。

在 ASP.NET 中，被指定为 runat=“server”的表单被称为 Web 表单，是 ASP.NET 用来为应用程序提供大部分功能的框架的一部分。在第 5 章“Web 表单初步”中将更详细地介绍它。

接下来几个有趣的标记是`<asp:textbox>`、`<asp:button>`、`<asp:label>`。它们是 ASP.NET 中的 Web 服务器控件，其中每一个控件都有属性，您可以使用这些属性为页面提供功能。同样，如果在 Web 浏览器中查看该文件的源代码，可以看到与这些 ASP.NET 控件等价的 HTML 对象分别是文本输入框、Submit 按钮和 span 标记，如清单 2.4 所示。

清单 2.4 index.aspx 的源代码

```
1 Please enter your name:
2 <input name='tbMessage' type='text' value='Chris'
3   id='tbMessage' />
4 <input type="submit" name="btSubmit" value='Submit'
5   id='btSubmit' /><p>
6 <span id="lblMessage" style="font-size:20pt;">Hello
7   Chris!</span>
```

**分析：**请注意每个控件的 id 属性，这是您给控件指定的唯一名称，以便可以在页面的其他地方引用它。例如，标签的 ID 为 lblMessage。在页面的后面，您可以编写一些代码，询问“lblMessage 中的文本是什么？”（事实上，您将在`<script>`块中完成类似的工作）。

**警告：**务必要指定`runat="server"`，对 ASP.NET 页面而言，`runat="server"`至关重要，它告诉 ASP.NET 这些项目将在服务器上进行处理。否则，ASP.NET 将不在服务器上做任何处理，直接将这些项目发送给客户。由于浏览器将忽略任何无法理解的标记，因此这些项目将不会被显示给用户。这是 ASP.NET 初学者常犯的错误。

**提示：**ASP.NET 要求开发人员编写严谨的代码。这意味着所有的标记都应该包括开始和结束标记，如`<body>... </body>`、`<form>... </form>`。因此，`<asp:textbox>`和`</asp:textbox>`也必须配对。

但也存在一些简写，如果在开始标记和结束标记之间没有指定任何内容，则可以在右括号之前加上斜杠即可，如`<asp:textbox />`。如果没有结束标签，ASP.NET 将出错。

**新术语：**您创建的文本框还有另一个属性：`OnTextChanged="tbMessage_Change"`。如果您使用过客户端脚本编写语言，则不会对此感到陌生。文本框有一个叫作 TextChanged 的事件。事件是应用程序中发生的情况，如鼠标单击或选择的内容发生变化。当文本框中的内容发生变化时，将引发 TextChange 事件。您通过添加 `OnTextChanged="tbMessage_Change"`，告诉 ASP.NET：发生事件 TextChanged 时，请执行过程 tbMessage\_Change。不过，与客户端脚本不同，这些事件是在服务器端进行处理的。

这是事件驱动模型的基础。通常，ASP.NET 应用程序只是对用户的输入做出响应。在 ASP.NET 页面中有大量的代码专门用于处理这些事件，您将经常遇到这种过程。

### 2.1.2 代码声明块

接下来看看`<script>`块，其中包含如清单 2.5 所示的过程。

清单 2.5 第一个 ASP.NET 页面中的`<script>`块

```
3 <script runat="server">
4   Sub tbMessage_Change(Sender As Object, E As EventArgs)
5       lblMessage.Text = "Hello " + tbMessage.Text
6   End Sub
```

```
7</script>
```

分析：同样，如果您熟悉脚本编写语言，肯定不会对这部分感到陌生。标记<script>通常用于编写客户端脚本。它定义了应用程序将动态处理的页面部分，在 ASP.NET 中，这被称为代码声明块。它不会像纯粹的 HTML 那样被交付给页面，而是包含应该由计算机执行的编程代码。别忘了加上 runat=“server”，否则 ASP.NET 将不会执行这些代码，而直接发送给浏览器。

第 4 行创建了一个 Visual Basic 子程序，以实现一些功能。从对清单 2.2 的分析可知，当其 TextChanged 事件发生时，文本框将使用该方法。这部分涉及大量的语法，这里只介绍一些基本知识。您需要知道的只是：每当该过程被调用时，它便将标签 lblMessage 的 Text 属性设置为“Hello”加上文本框中的文本，如第 5 行所示。结果，页面上将显示“Hello[您的姓名]”。第 2 行中的代码(Sender As Object, E As EventArgs)是大多数处理事件的方法都必不可少的。这将在第 5 章和第 6 章中详细讨论，而 Visual Basic 代码将在下一章中详细讨论。

同样，应该使用属性 runat=“server”来指定代码块应该在服务器上运行，这一点至关重要。否则，ASP.NET 将把它看作是客户端代码，因此无法得到预期的结果。就这个例子而言，ASP.NET 将引发错误，如图 2.2 所示。

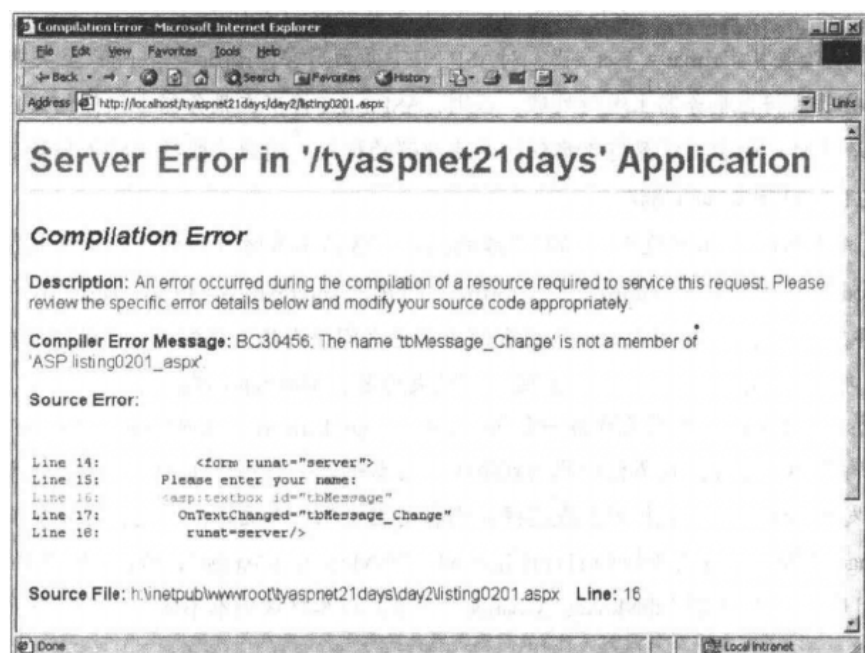


图 2.2 如果未使用属性 runat=“server”将发生的错误

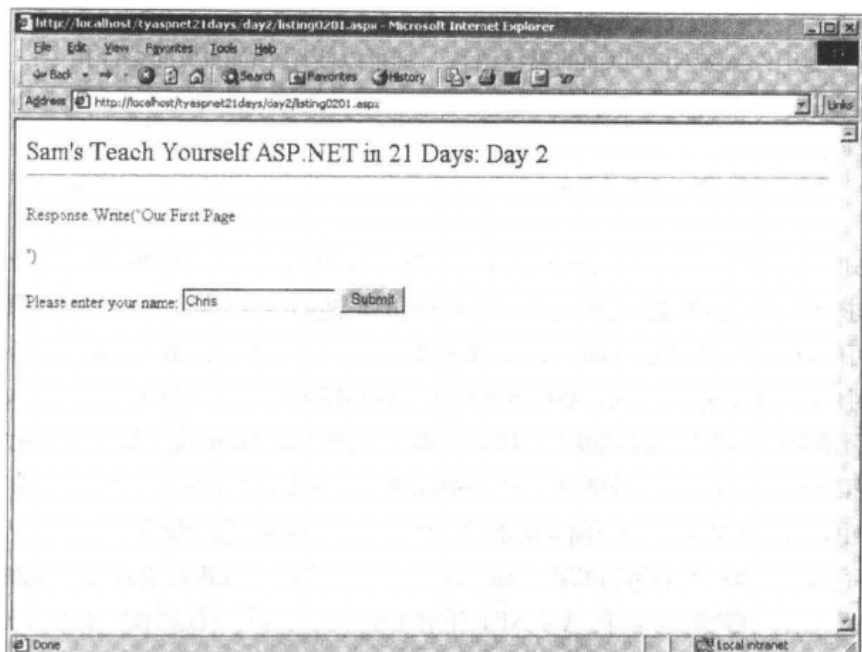
提示：实际上，可以将<script>放在页面的任何位置，但好的习惯是将其与HTML代码尽可能分开。这样，处理问题时，管理和调试代码将更为容易。以后您将知道如何将代码与页面中的其他代码完全分开。

### 2.1.3 代码交付块

最后，让我们来看看第 12 行：

```
12: <% Response.Write( "Our First Page<p>") %>
```

这是一个代码交付块，服务器将对<%和%>之间的 Visual Basic 代码进行处理，然后将结果发送给浏览器。如果删除这些标记，ASP.NET 将把这些代码视为纯粹的 HTML 代码，结果将类似图 2.3。

图 2.3 省略标记`<%和%>`后的结果

第 12 行使用对象 `Response` 的 `Write` 方法将文本显示到浏览器中。第 4 章将介绍该对象及其方法，但就现在而言，您只需要知道它提供了易于将文本动态写入到浏览器中的途径。

如果 `Response.Write` 的输出包括任何 HTML 标记，则浏览器将把它看作是纯粹的 HTML。例如，如果使用下面的代码，则“`Our First Page`”将显示为粗体。

```
<% Response.Write( "<b>Our First Page<b>") %>
```

所有要交付的字符都应该放在双引号中。这是一个简单的例子，但该方法能够生成一些有趣的输出，在第 4 章您将明白这一点。

**提示：**可以使用标记`<%=`来简写`Response.Write`，例如下面的两行代码是等价的：

```
<% Response.Write("Hello") %>
```

```
<% = "Hello" %>
```

ASP.NET 页面经常使用这种语法，因此您必须知道这一点。

代码交付块与代码声明块非常像。甚至可以将这些代码放在标记`<script>`中，它也将以类似的方式运行。但第 12 行和`<script>`块之间还是有一些不同的地方。首先代码声明块将被编译为机器码（首先被编译为 MSIL），这使得它比代码交付块的速度要快，稳定性更高。第二个差别是，处理的顺序不同。如果将第 12 行移到代码声明块中，则得到的结果将不同于分开的情况。

由于上述原因，在 ASP.NET 中，代码声明块使用频率将比代码交付块高得多。代码交付块在传统 ASP 中至关重要，但在 ASP.NET 中，已经被更好的机制所取代。

#### 2.1.4 页面编译指令

清单 2.1 的第 1 行包含如下代码：

```
1: <%@ Page Language="VB" %>
```

这被称为页面编译指令（page directive），让您能够包含特殊的指令，指示 ASP.NET 如何处理页面。该页面编译指令告诉 ASP.NET：在页面中，您将使用 Visual Basic 作为默认的编程语言。在本书的后面，您将学习如何使用其他的页面编译指令。在本章后面的“导入名称空间”一节中，您将学

习编译指令 `import`。

如果想将 C# 用作 ASP.NET 页面的默认语言，则可以使用下面的页面编译指令：

```
<%@ Page Language="C#" %>
```

第 4 章将更详细地介绍 C#。

### 2.1.5 流程

现在您知道了代码的大概情况，那么当用户在 Web 浏览器中请求页面时，将发生什么情况呢？

首次请求页面时，ASP.NET 将对代码声明块中的代码进行编译。如果浏览器显示页面时有一定的延迟，则可能就是因为编译引起的。不对代码做任何修改，并再次请求页面时，将不会有任何延迟，但即使对代码做非常小的改动，ASP.NET 都将重新编译页面。虽然编译将导致首次浏览时有些延迟，但以后请求时，性能将得到极大的提高。那么，引擎如何知道代码发生了变化呢？

还记得第 1 章介绍的管理代码吗？上述 ASP.NET 页面正是这样的——一些由 CLR 处理的管理代码。该页面被编译为 MSIL，当页面被请求时，MSIL 将被编译为机器语言。然而，您创建的源文件是被单独存储的，CLR 对其进行监视。如果该文件发生变化，CLR 将重新编译页面。

表单被提交后，代码将被编译。ASP.NET 开始处理您创建的所有代码和发生的所有事件。在这个例子中，由于您在文本框中输入了文本，所以将发生 `TextChanged` 事件。ASP.NET 查看该事件，确定它应该怎么做，并这样做。

然后，ASP.NET 将所有服务器控件转换为 HTML 元素。例如，它将控件 `<asp:TextBox>` 转换为 HTML 文本输入框。然后查看代码交付块，如果需要，则输出相应的 HTML。

最后，生成的 HTML 将被发送给浏览器。浏览器只接收标准的、合法的 HTML——ASP.NET 不会向浏览器发送任何代码或服务器控件。

因此，任何 Web 浏览器都能够访问并正确地交付 ASP.NET 页面。在 Web 浏览器看来，ASP.NET 页面只是做了一些扩展的 HTML 页面而已。例如，图 2.4 显示了浏览器收到的 HTML。

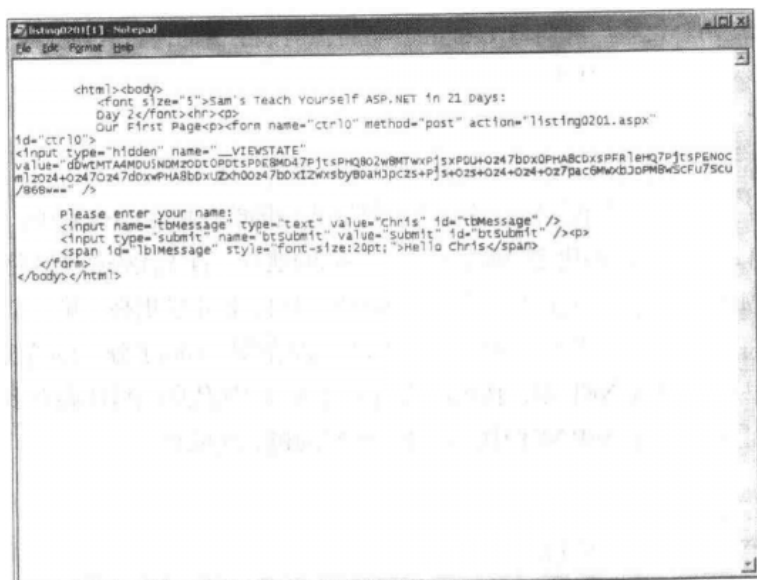


图 2.4 浏览器从清单 2.1 收到的 HTML 源代码

这一概念至关重要，必须掌握。从某种程度上说，浏览器是一个很笨的应用程序，它只能解释 HTML，仅此而已。ASP.NET 知道这一点，因此任何需要发送给浏览器的东西都被转换为 HTML。

幸运的是, ASP.NET 很聪明, 知道如何让 HTML 也执行一些处理工作。这将在第 5 章做更详细的介绍。

图 2.5 说明了典型的 ASP.NET 页面的工作流程。

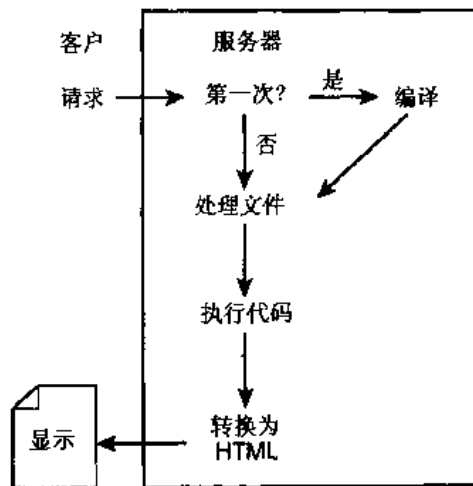


图 2.5 ASP.NET 的工作流程：从请求到显示

这简单地说明了 ASP.NET 页面的处理和递送过程。

### 2.1.6 视图状态

视图状态 (viewstate) 描述了对对象在特定时间的外观。例如, 图 2.1 所示的文本框的视图状态包含文本 “Chris”, 按钮的视图状态指出了它是否被单击, 等等。跟踪这种信息的应用程序被称为维护状态 (maintain state)。

如果您填写 HTML 表单, 并过一会儿再切换回来, 很可能您已经填写过的字段又变成了空的。这是因为 Web 是无状态的 (stateless) ——它不允许您跟踪视图状态或其他诸如此类的信息。对于传统 ASP 开发人员来说, 这很讨厌, 因为需要使用某种机制来维护和检索这种信息。ASP.NET 使得这种工作容易得多。

ASP.NET 自动替您跟踪视图状态。这意味着如果您填写 HTML 表单, 并单击 Submit 按钮, 页面刷新后, 其中仍将显示填写的值。这是 ASP.NET 中非常重要的组成部分, 它是许多机制的有机组成部分。

当您指出表单将在服务器上运行时, ASP.NET 将生成隐藏 HTML 表单字段, 以自动跟踪视图状态。还记得清单 2.3 中第 3 行的隐藏字段吗? 那个看起来好像由一些随机字符组成的字符串, 便是 ASP.NET 用来说明控件外观的。表单被提交后, ASP.NET 将自动检索该字符串, 并使用它来重新填写表单信息。ASP.NET 知道浏览器只能理解 HTML, 因此将自己记录下来的内容写入到发送给客户的页面中。

例如, 服务器上的一行代码:

```
<form runat="server">
```

将把如下 HTML 代码发送给浏览器:

For example, look at the following line written on the server:

```
<form runat="server">
```



This sends the following HTML code to the browser:

```
<form name='ctrl12' method='post' action='listing0201.aspx'
  id='ctrl12'>
```

```
<input type='hidden' name='_VIEWSTATE'
  value='YTB6LPEwNzAyOTU3NjJfXl549e1355cb'
```

Viewstate management showcases ASP.NET's focus on making the Web a more traditional application environment.

视图状态管理使得 ASP.NET 呈现出这样的优点,即它致力于使 Web 更像传统的应用程序环境。

## 2.2 编写 ASP.NET 代码和 HTML 代码

鉴于代码和服务器的交替出现,您可能难以弄清楚 ASP.NET 页面到底包含哪些东西,应该将它们放在什么位置。这些令人眼花缭乱的代码是传统 ASP 开发人员面临的挑战之一。

ASP.NET 使之尽可能简单。编写 ASP.NET 代码的方式有两种:在代码声明块中编写或在代码交付块中编写。因此,您必须能够轻松地对页面中的 ASP.NET 代码进行分类。第一种方法更佳,原因很多:不同于代码交付块,它将被编译;消除了代码令人眼花缭乱的问题,是一种更好的设计应用程序的方法。

其他任何东西几乎都是 HTML,在页面中,您在清单 2.1 中使用的服务器控件也是使用 HTML 编写的(这些控件确实是服务器端的对象,但在浏览器中,其界面是使用 HTML 描述的)。即使是事件说明符(见清单 2.2 中的第 17 行)也是 HTML。使用 ASP.NET,也有许多方法可以编写纯粹的 HTML 代码,如 `Response.Write` 或其简写 `<%=` 标记。

图 2.6 说明了构造 ASP.NET 页面时应该使用的方法,即将 ASP.NET 代码和 HTML 代码分开。如果在 HTML 中使用代码交付块,则维护 ASP.NET 代码的工作将变得非常困难,因为 HTML 代码将分散在各个地方。

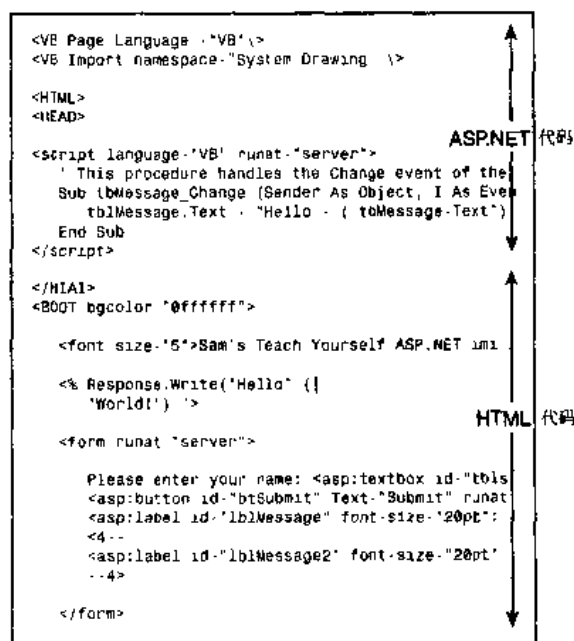


图 2.6 构造 ASP.NET 页面的最佳方式

应 该	不 应该
一定要使用代码声明块将 HTML 代码和 ASP.NET 代码尽可能分开	当能够找到其他方法时（通常能够），不要使用代码交付块，使代码与 HTML 输出（使用 <code>Response.Write</code> ）交织在一起

### 2.2.1 对代码进行注释

注释是代码中不影响代码执行的行。插入这些内容主要是为了帮助开发人员和其他人理解代码，虽然注释也可用来阻止某些代码被执行。最终用户看不到注释行，因为它们不会被发送给浏览器。

对代码进行注释对开发复杂应用程序的帮助非常大。注释让您能够使用自然语言阐述代码的逻辑，当您忘记了代码的功能时，这将非常有帮助。

在 ASP.NET 中，对代码进行注释的方法有三种。首先如果您熟悉 HTML，则应该知道 HTML 中的注释标记 `<!-- 和 -->`。这些标记只能用于注释 HTML。

在 ASP.NET 代码中，可以使用您所使用的编程语言的注释格式。例如，在 Visual Basic.NET 中，可以使用单引号，一次注释一行。下述代码片段的第 3 行和第 4 行便是一个注释。

```

1  <script language="VB"runat="server">
2      sub Page_Load(obj as object,e as eventargs)
3          ` this method executes as Soon as the ASP.NET page
4          ` loads
5          some code
6      end sub
7  </script>
```

C#则使用两个斜杠进行注释：

```

1  <script language="C#"runat="server">
2      function Page_Load(object obj,eventargs e){
3          // this method executes as soon as the ASP.NET page
4          // loads
5          ...
```

最后，除了代码声明块，在其他任何地方都可以使用服务器端注释标记 `<% 和 --%>`。像 HTML 注释标记一样，上述标记可一次注释多行。清单 2.6 说明了这三种注释风格。

**清单 2.6 各种不同的注释风格**

```

1  <%
2      ` this is a Visual Basic comment
3      ` on multiple lines
4  %>
5  <!--This is an HTML comment
6  on multiple lines-->
7  <%--This is a server-side comment
8  on multiple lines--%>
```

**警告：**如果试图在代码声明块中使用服务器端注释标记，将发生错误

### 2.2.2 跨越多行的代码

在 HTML 中，编写跨越多行的代码很容易。例如，下述两个代码块是等价的：

```
<b>Hello World!</b>

<b>Hello
World!</b>
```

在 ASP.NET 和 Visual Basic.NET 中，则没有这样简单。如果使用下述方式编写代码，将导致错误：

```
<%
    Response.Write
        ('Hello World')
%>
```

如图 2.7 所示：

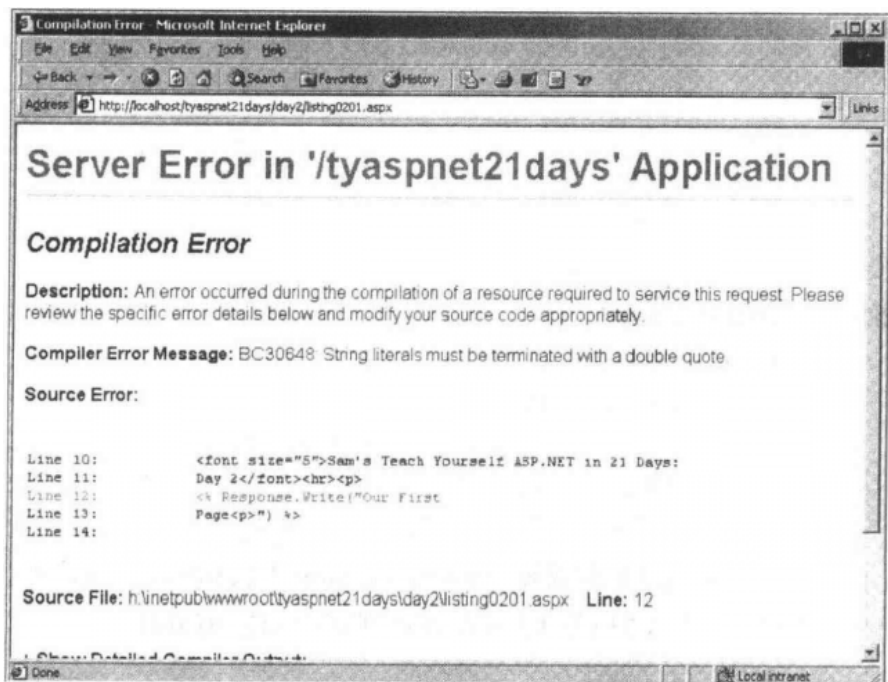


图 2.7 跨行代码引起的错误

Visual Basic.NET 引入了使用下划线 ( \_ ) 表示的连行符。让我们对前面的代码做一些修改：

```
<%
    Response.Write_
        ("Hello World")
%>
```

现在便能得到正确的输出，没有任何错误。在本书中，您将遇到大量的连行符。

使用连行符时，还有一个问题需要注意：不能在字符串中间使用连行符。例如，下面的代码将

导致另一个错误:

```
<%
    Response.Write("Hello_
        World")
%>
```

ASP.NET 将把该字符视为字符串中的一个下划线,并期望后面有其他代码。要在字符串中间进行分行,应使用双引号将前一部分括起,然后使用字符串连接字符(在 VB.NET 中为“和”符号)和连行符:

```
<%
    Response.Write('Hello' &_
        "World")
%>
```

注意:诸如C++和C#等语言不会出现如图2.7所示的错误,因为它们有特殊的行终止符来告诉应用程序是否到了行尾。因此,当代码跨越多行时,只要不输入换行符即可。

## 2.3 应用程序的其他方面

这个小型的应用程序好像不过如此而已,但它让您知道创建其他任何 ASP.NET 应用程序的基础所在。您学习了有关事件驱动模型和 Web 控件的知识,它们是 ASP.NET 的有机组成部分。该应用程序将让您知道 ASP.NET 是如何与.NET 框架的其他部分交互的,这将让您领略到该新框架的强大功能。

### 2.3.1 再谈 ASP.NET 编译

可以将.NET 框架看作是一组对象蓝图。例如, System.Web.UIPage 对象蓝图告诉您 ASP.NET 页面的基础所在及其将如何运行。

创建和编译 ASP.NET 页面时,ASP.NET 页面将成为一个真正基于蓝图 System.Web.UIPage 的对象。它包括自己的属性和方法,这些都是从 System.Web.UIPage 那里继承而来的。每个 ASP.NET 页面都是一个基于该蓝图的对象。这样每个 ASP.NET 页面都可以通过提供一些自定义的功能来扩展.NET 框架。应用程序将成为框架的一部分。

您将开始看到该层次式框架的强大功能,通过添加东西,可以使其更为强大得多。如果发现原来的框架中缺少某种特性,您只需构建该特性,并将其加进去,就像它早就存在一样。

另外,也有纯 HTML 蓝图,它叫 System.Web.UILiteralControl 类。因此,HTML 也是.NET 框架的一部分。基于 System.Web.UILiteralControl 蓝图的对象将包含所有的文字值,并被作为 HTML 文本交付给浏览器。这意味着 HTML 像其他对象一样,也有属性、方法和事件。

### 2.3.2 导入名称空间

在.NET 框架中,每个名称空间都是一个蓝图集。ASP.NET 有自己的蓝图,但有时候这些蓝图不够用。因此必须在 ASP.NET 页面中引用其他的蓝图集,以创建各种类型的对象。可以使用关键字 Import 来访问这些对象:

```
<%@ Import Namespace='System.Drawing' %>
```

上述代码将导入名称空间 `System.Drawing` 中的所有类，如 `font` 类和 `image` 类。现在您可以在自定义对象中使用这些类了。您还可以导入自己的用户自定义名称空间。默认情况下，下述名称空间将自动导入到每一个 ASP.NET 页面中：

- `System`
- `System.Collections`
- `System.IO`
- `System.Web`
- `System.Web.UI`
- `System.Web.UI.HtmlControls`
- `System.Web.UI.WebControls`

这些名称空间是 ASP.NET 的一部分，不用明确地导入它们，您也看不到用于导入它们的命令。ASP.NET 知道，这些名称空间肯定是要的，所以将它们准备好了，供您使用。图 2.8 列出了一小部分 .NET 框架中可用的名称空间。

但是，正如您从图 2.8 中可以知道的，导入一个名称空间并不能导入其底层的名称空间，只有属于该接口的类才会被导入。

```

System
System.CodeDOM
System.CodeDOM.Compiler
System.Collections
System.Collections.Bases
System.ComponentModel
System.ComponentModel.Design
System.ComponentModel.Design.CodeModel
System.Configuration
System.Configuration.Assemblies
System.Configuration.Core
System.Configuration.Install
System.Configuration.Interceptors
System.Configuration.Schema
System.Configuration.Web
System.Core
System.Data
System.Data.ADO
System.Data.Internal
System.Data.SQL
System.Data.SQLTypes
System.Diagnostics
System.Diagnostics.SymbolStore
System.DirectoryServices
System.Drawing
System.Drawing.Design
System.Drawing.Drawing2D
System.Drawing.Imaging
System.Drawing.Printing
System.Drawing.Text
System.Globalization
System.IO
System.IO.IsolatedStorage
System.Management
System.Messaging
System.Net
System.Net.Sockets
System.Reflection

```

图 2.8 一小部分 .NET 名称空间

**注意：**要使用名称空间中的对象，并不需要导入该名称空间，有一种更简单的方法。例如，您可以在 ASP.NET 页面中使用下面的代码：

```
dim objFile as File
```

（现在不用担心其中的语法，重要的是名称空间部分）。

File位于名称空间System.IO中，但不用导入该名称空间，便可以使用上述对象，方法是使用类似于下面的代码：

```
dir objFile as System.IO.File
```

换句话说，可以通过使用完整的名称空间名称来引用对象，这告诉ASP.NET到哪里去找该蓝图。导入名称空间仅仅让您能够使用简短的蓝图名称而已。

有关.NET 名称空间的完整列表及其属性，请参考 ASP.NET 环境自带的.NET 框架文档。

## 2.4 CLR 和 ASP.NET

让我们回过头来再次简要地讨论一下 CLR 是如何处理 ASP.NET 应用程序的。您需要知道 CLR 所做的工作，以便可以预先知道它将如何处理代码。

### 2.4.1 中间语言

ASP.NET 页面被编译时，将被转换为微软中间语言（MSIL）。这是一个独立于 CPU 的高效指令集，用于控制应用程序。基本上，它是您所编写的代码的简洁格式。MSIL 本身并不能运行，但可以轻松地将其转换为机器本机语言，以便执行。MSIL 代码被存储在可移植的可执行（PE）文件中。

### 2.4.2 执行

应用程序被执行前，MSIL 被即时（Just-in-time, JIT）编译器转换为机器本机语言，该编译器是 CLR 的一部分。与非编译代码相比，机器本机代码的运行效率要高得多。由于 MSIL 是独立于 CPU 的，因此 JIT 编译器必须将其转换为用户机器使用的语言。因此，需要非常多的 JIT 编译器。

**新术语：**PE 文件被编译为本机代码时，JIT 编译器将验证其类型安全（type-safe）。即它禁止对象引用其不可以引用的内存单元。换句话说，如果您创建一个 X 类型的对象，JIT 编译器将禁止您像 Y 类型那样使用该对象。这确保每个应用程序都被隔离，不会干扰其他的应用程序，可以防止内存被破坏。JIT 编译器只允许类型安全代码通过验证。当您开发 ASP.NET 页面时，将遇到更多这样的例子，这种情况非常常见。

### 2.4.3 处理

**新术语：**操作系统处理应用程序时使用的传统方法涉及到进程（process）。一个进程运行一个应用程序，它决定了该应用程序可用的资源。进程边界确保了应用程序是相互隔离的——如果一个进程（或应用程序）出现问题，其他应用程序将继续运行，而不受影响。图 2.9 列出了 Windows 2000 中运行的典型进程。

可以将操作系统看作是一个蜂窝，蜜蜂各干各的，不相互干扰。如果蜜蜂相互干扰，它们将崩溃。可以将每只蜜蜂看作是一个进程，因为它们运行一个“采蜜”程序，并知道花园的边界在哪里。如果一只蜜蜂被烧伤，其他蜜蜂还能继续工作。

**新术语：**然而从性能和稳定性的角度看，这种方法的代价仍然是非常昂贵的。.NET 框架引入了应用程序域（application domain），它是全新的、更小的处理单元。因为 CLR 知道其运行的所有代码都是类型安全的，因此可以花更少的时间来监视进程。如果需要，单个应用程序可以有多个应

用程序域，这意味着容错能力更强。

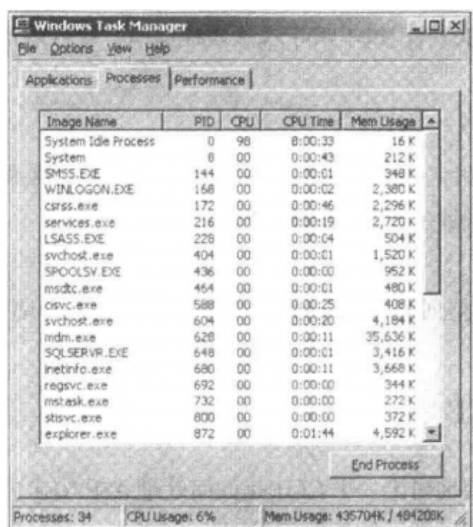


图 2.9 Windows 2000 上运行的典型进程列表

还是将其想象为一个蜂窝，只是每个采蜜程序使用多个蜜蜂而已。如果其中的某个蜜蜂受伤、迷路或被其他昆虫吃掉，则采蜜程序将继续下去，出现问题的蜜蜂将由另一只蜜蜂代替。这就是.NET 处理应用程序的方式。

#### 2.4.4 组合体

**新术语：**组合体（assembly）是 CLR 中基本的共享和重用单元。组合体包含代表其他常用单元的文件，如 ASP.NET 页面、PE 文件、图像或 VB.NET 源文件。在 Windows 框架中，与之类似的对象是动态链接库（DLL），但组合体使.NET 框架可以执行更为严格的安全型和应用程序版本管理，这反过来使代码更为稳定。因此，即使是 ASP.NET 页面，也被放置到页面被请求时动态创建的组合体中。

#### 2.4.5 并行执行

CLR 提供了同时运行同一个组合体的多个版本的功能，这就像在传统框架中同时运行 DLL 的多个版本（而这是不可能的，也是人们所渴望的）一样。在第三部分，当您创建更为复杂的 ASP.NET 应用程序时，这一主题将变得非常重要。

**注意：**并行执行并非可以同时运行 ASP 和 ASP.NET 页面的原因所在，其原因在于每种类型的页面是由不同的 IIS ISAPI 过滤器处理的。

当 Web 服务器收到 HTTP 请求时，ISAPI（Internet 服务器应用程序编程接口）程序进行响应。过滤器可用于任何 Web 服务器事件，如事件 Read 和 Write。对于 Web 应用程序来说，这些过滤器非常有用。

#### 2.4.6 对 ASP.NET 而言，CLR 意味着什么

从 ASP.NET 开发角度看，CLR 为开发人员承担了许多工作。与传统方法相比，它管理内存的能力要高得多，并使 ASP.NET 应用程序的性能更高，稳定性更强。由于具备更强的故障隔离功能，当 Web 应用程序崩溃时，您再也不需要重新启动 IIS 或服务器。与以前的 Web 开发方法相比，ASP.NET

的效率高得多,也稳定得多。

## 2.5 ASP.NET 编程语言

您也许会感到迷惑,进而询问“我是要学习一种新的编程语言吗?”ASP.NET不是一种编程语言,而只是一个框架,让您能够构建 Web 上的应用程序。

前面讲过,您几乎可以使用任何编程语言编写 ASP.NET 页面,如 Visual Basic.NET、C++的.NET 管理扩展、C#或 JScript.NET。所有这些代码都将被编译为 MSIL,同时编译器必须生成描述应用程序的元数据。有了这种中间编译器语言,JIT 编译器只需理解 MSIL 即可。第3章和第4章将介绍 ASP.NET 编程中最常用的两种语言 VB.NET 和 C#。

由于 CLR 需要确保其所有的部件能够协同工作,因此它定义了一个每种编程语言都必须遵循的特性子集。否则,使用不同语言开发的对象将无法协同工作。该子集被称为通用语言规范(Common Language Specification, CLS)。只要应用程序只使用 CLS 中的特性,便可以在任何平台上运行,并能够被其他语言中编译的对象所使用。

## 2.6 重新审视前面的代码

对 ASP.NET 的工作原理有了进一步的了解后,再回过头来看看本章开头的代码,如清单 2.7 所示。

清单 2.7 重新审视第一个 ASP.NET 页面

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub tbMessage_Change(Sender As Object, E As EventArgs)
5         lblMessage.Text = "Hello " + tbMessage.Text
6     End Sub
7 </script>
8
9 <html><body>
10     <font size="5">Sam's Teach Yourself ASP.NET in 21 Days:
11     Day 2</font><br><p>
12     <% Response.Write("Our First Page<br>") %>
13
14     <form runat="server">
15         Please enter your name:
16         <asp:textbox id="tbMessage"
17             OnTextChanged="tbMessage_Change"
18             runat="server"/>
19         <asp:button id="btSubmit" Text="Submit"
```



```

20         runat=server/'><p>
21         <asp:label id="lblMessage" font size="20pt"
22         runat=server'>
23     </form>
24 </body></html>

```

**分析：**很容易辨别 ASP.NET 代码的不同部分——3-7 行的代码声明块和 12 行的代码交付块。同时，您知道每个代码段的功能。前一个代码段定义了一个方法，供文本框用于其 TextChanged 事件；而第二个代码段只是将“*Our First Page<P>*”写入到浏览器中而已。

首次请求该网页时，代码声明块将被编译为 MSIL 代码，然后 JIT 编译器将把 MSIL 代码转换为本机机器码。但页面的其他部分（包括代码交付块）将在运行阶段（页面被装载到浏览器中时）进行评估。

表单中还包括三个服务器控件，其行为与常规 HTML 控件类似，但为应用程序提供了其他的功能。第 5 章和第 6 章将介绍这些控件以及其他服务器控件。

您还认识到了属性 `runat="server"` 的重要性，没有它，ASP.NET 页面将无法正常运行。该属性是维护视图状态的关键所在，并将浏览器发生的情况与服务器应该采取的措施关联起来。

阅读本章后，您将能够理解该页面中各种机制的细节，无论是从 .NET 框架的角度还是用户的角度。通过这两章，您已经学到了很多的东西。

## 2.7 这不是 ASP

如果您熟悉传统 ASP，通过阅读本章，您将发现其与 ASP.NET 之间的许多差别。最大的变化是从代码交付块到代码声明块的转移。以前，代码交付块一直是标准的处理方法，比代码声明块更受到人们的青睐。

现在，代码声明块更佳，因为它们是被编译过的，并可以充分利用 CLR 的特性。这种代码块也使得将 MSIL 代码和 HTML 代码隔离开来更为容易。事实上，如果您想处理事件和表单提交，则代码声明块是必不可少的。现在，在许多情况下，代码交付块都将引发错误，因为处理模型不同了。

另一个重大变化是从脚本语言到编译型语言的转移。以前，所有人使用 VBScript 来编写 ASP 页面，偶尔也使用 JScript。ASP.NET 不再支持这种解释型语言，但您可以使用任何可以生成 MSIL 的编译型语言，如 VB.NET 或 C#。这使得您能够使用诸如早期联编和强类型变量等特性。向编译语言的转移绝对是值得的，如果您熟悉 VBScript，则可以很轻松地转移到 VB，因为这两种语言极其类似。更详细的信息，请访问 <http://www.getdotnet.com/languages.aspx>。

ASP.NET 现在是完全面向对象的，每个对象都适用于 .NET 框架，并完全支持 OOP 原则，如继承、重载和接口等。

在 ASP 和 ASP.NET 之间，一个受到欢迎的相似之处是 `Response.Write` 仍然可以像以前一样管用。可以在 ASP.NET 页面中使用它来生成输出，甚至提供调试功能（ASP 开发人员将证明这一点）。ASP.NET 还有一些更佳的机制，用于调试这将在第三部分介绍。

**警告：**尽管 `Response.Write` 仍然管用，但现在使用它时必须带括号，例如下述代码现在不再管用：

```
Response.Write 'Hello World'
```

必须将字符串放在括号内。附录B将介绍更多的可能导致常见错误的变化。

虽然不那么容易发现,但最大的变化是 ASP.NET 的处理模型。代码被编译为 MSIL,然后使用 JIT 编译器将其编译为机器语言。应用程序被划分为应用程序域而不是进程,而文件被组织成组合体和名称空间。对刚开始使用 ASP.NET 的开发人员而言,这些变化也许不重要,但随着您深入探讨 .NET 框架,这些变化将非常明显。

## 2.8 总 结

本章介绍了许多关于 ASP.NET 框架的知识,虽然目前这些知识可能还用不上,但当您开发更复杂的 ASP.NET 应用程序时,它们将派上用场。

ASP.NET 在服务器上处理事件和代码,浏览器只是查看 HTML,这意味着客户不需要添加额外的应用程序。这是 ASP.NET 的一个重要特征——将所有代码转换为 HTML,再发送给客户。

属性 `runat="server"` 为 ASP.NET 页面提供了大量的功能,让组件能够将服务器和维护视图状态程序链接起来。该属性对 ASP.NET 页面至关重要。

通过使用标记 `<script>... </script>` 和 `<%... %>`, ASP.NET 代码得以和 HTML 代码隔离开来。大多数代码都应该放在前一个标记中,因为这些代码段将被编译,并消除了代码令人眼花缭乱的问题。

注释由下述三组标记之一定义: `<!--...-->` 用于注释 HTML 代码; `'` 用于 VB; 而 `<%--...-->` 用于服务端。这些注释通常用于解释代码,它们不影响代码的执行。

使用连行符 (`_`) 可以将长语句分成多行,在字符串中使用该字符时,务必用双引号括起字符串的前半部分,并在后面加上“和”符号。

.NET 框架和 CLR 给 ASP.NET 应用程序引入了许多新概念,包括应用程序域(一种隔离应用程序的新方式)、组合体和名称空间。ASP.NET 是 .NET 框架的一个有机组成部分,您所创建的每一个页面都是对该框架的扩展。

最后,ASP.NET 让您能够使用诸如 VB、C++、C# 等编译型语言,并不再支持诸如 VBScript 等解释型语言。

接下来的两章将介绍如何编写 ASP.NET 页面。本章您了解了 ASP.NET 页面的外观及其各部分的工作原理,接下来应该开始学习如何编写处理这些页面的代码。第3章将重点介绍 VB.NET 的编程技术,而第4章将重点介绍 C#。它们是 ASP.NET 最常用的两种编程语言。

## 2.9 问与答

问: 导入名称空间会增加开销吗?

答: 导入名称空间与在其他编程语言中使用包含文件或访问引用类似。不存在处理开销,而且与其他语言不同,文件的大小也不一定会增大。另外, JIT 编译器一次只编译代码中要使用的一小部分,因此导入名称空间不一定会增加编译时间和处理时间。

问: ASP.NET 页面真的可以使用任何编程语言创建吗?

答: 理论上是这样的,虽然语言必须被 CLR 支持,并能够生成 MSIL。当前支持的只有 C#、VB.NET、JScript.NET 和 C++ 的管理扩展,但开发人员可以相对容易地在 CLR 中添加对任何语言的支持。

## 2.10 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 2.10.1 小测验

1. ASP.NET 如何维护服务器控件的状态？
2. ASP.NET 的 Page 和 LiteralControl 类属于哪个名称空间？
3. 自动被导入到每个 ASP.NET 页面中的名称空间是哪 7 个？
4. 类型安全的含义是什么？
5. 何为通用语言规范？

### 2.10.2 练习

1. 创建一个简单的 ASP.NET 页面，它从用户那里接受两个数字，Submit 按钮被单击后，显示它们的乘积。使用两个文本框控件、一个标签控件和一个 Submit 按钮控件。另外，在代码声明块中使用标准的事件处理程序代码。尝试使用函数 CInt() 将文本框中的文本转换为整数，以便相乘，如 CInt(tbNumber1, Text)。
2. 在前一个练习的基础上，加上多个 Submit 按钮，执行基本的算术运算，以创建一个计算器。



## 第3章

# 使用 Visual Basic.NET

在前两章，您知道了 ASP.NET 页面的组成部分及各部分的功能，并在 .NET 框架方面打下了坚实的基础，本章将学习如何使用 Visual Basic.NET 创建 ASP.NET 页面。

使用 VB.NET 能够创建出使用了 ASP.NET 强大功能的动态页面。本章介绍 ASP.NET 的语法、通用编程结构和编程方法学，讲述每一项内容时，都提供了一些实例。这决不是 VB.NET 的完整指南，而只是对后续章节中您需要知道的一些概念作一简介。

即使您熟悉 VB.NET，也应该阅读本章。因为它并非关于该语言的初级读本——它包含了关于如何使用 VB.NET 编写 ASP.NET 页面的信息。

本章将介绍以下内容：

- VB.NET 简介；
- 变量和数组；
- 条件逻辑、循环逻辑和分支逻辑；
- 如何编写事件处理程序；
- 类是什么；
- 一些很有用的 VB.NET 函数。

### 3.1 Visual Basic.NET 简介

Visual Basic (VB) 是一种人们使用了多年的编程语言。最初，VB 只是用于创建应用程序原型，但自从其问世以来，得到了极大的发展，现已成为一种功能强大的开发环境，可用于创建各种单机应用程序。

VB.NET 是该语言最新版本，得到了 .NET 框架和 CLR 的全面支持。它是最流行的 ASP.NET 开发语言之一，最重要的是，它易于学习。由于这些原因，本书将使用 VB.NET 来编写 ASP.NET 页面。

### 3.2 变 量

变量是一个通用性术语，指的是计算机内存中有名称的数据。例如，如果您将字符串“Hello World”赋给一个变量，并将其命名为 x，则该字符串将被放置在内存中，占用（大约）10 个字节。这样，便可以使用上述名称引用这些信息。

因为变量只不过是一个内存单元，因此可以通过修改、删除、移动来操纵它。但变量最为重要的部分是内存单元中包含的数据。

### 3.2.1 数据类型

您可以将许多不同类型的信息存储到变量中，如字符串、数字和日期等。每种类型都有一组规则管理其用法，当您开发 ASP.NET 页面时将发现这一点。在 Visual Basic.NET 中，有 10 种基本的数据类型，它们被称为原语（primitive）类型。它们是使用变量的基石，这也是术语原语的由来。这 10 种数据类型被划分为 5 个类别：整数、浮点数、字符串、日期和布尔型。表 3.1 对这些数据类型做了总结。

表 3.1 VB.NET 原语

类 型	类 别	描 述
Byte	整数	1 字节的整数（也叫 System.Int1）
Short	整数	2 字节的整数（也叫 System.Int16）
Integer	整数	4 字节的整数（也叫 System.Int32）
Long	整数	8 字节的整数（也叫 System.Int64）
Single	浮点数	4 字节的小数（也叫 System.Single）
Double	浮点数	8 字节的小数（也叫 System.Double）
Decimal	浮点数	12 字节的小数（也叫 System.Decimal）
Char	字符串	一个 Unicode 字符（也叫 System.Char）
Date	日期	日期和/或时间值（也叫 System.DateTime）
Boolean	布尔型	真或假（也叫 System.Boolean）

#### 1. 整数

整数指的是不包括分数或小数部分的数。如 36767 和 -1 都是整数，而 3.4 和  $-3\frac{1}{2}$  则不是。

整数是一种总称，您可以根据需要的内存多少来使用其子类型。一个 integer 变量使用 32 位的内存——4 字节。这意味着它可以存储从 -2147483648 到 2147483647 之间的任何整数，通常这足以满足您的需求。当然，还有 byte（8 位）、char（16 位）、short（16 位）、long（64 位）等子类型。对于这些数据类型，不必过分担心，之所以在这里介绍它们，只是以备不时之需。

#### 2. 浮点数

浮点数是带小数部分的数，如 4.5、-1.956445、3.0 等。

根据需要多少位小数，可以使用 single、double、decimal 等子类型。同样，您也不必对这些数据类型过分担心，因为默认的子类型通常够用。

#### 3. 字符串

字符串是一组字符，如“Hello”、“my name is”、“@\$\$@#&!”，乃至“234”等。在前两章，您已经使用过字符串。它们是 ASP.NET 中最常用的数据类型之一。在 VB.NET 中，字符串被括在双引号之间，如“Hello”。

#### 4. 日期

日期指的是日期和时间值, 实际的数据类型叫做 `DateTime`, 可以以许多不同的格式存储, 如“1/2/2001”、“Wednesday January 5th, 2001 8:09:30PM”等等, 可以很容易地从一种格式转换为另一种格式。

可以将日期表示为字符串, 但数据类型 `DateTime` 允许 VB.NET 执行一些字符串不可能的特殊操作, 如将小时、分、乃至天相加。VB.NET 中包含大量的日期函数, 可以在 ASP.NET 页面中使用它们。

#### 5. 布尔型

布尔型是真/假值的总称, 如 1/0、yes/no、on/off 等。在 VB.NET 中, 布尔数据类型只能是真或假。

#### 6. Object

`Object` 数据类型是对没有指定数据类型的变量总称。在 .NET 框架中, 它有特殊用途, 以后您将知道这一点。

### 3.2.2 变量的声明

那么如何使用变量或数据类型呢? 首先, 要告诉系统您要预留一块内存, 并给该内存块指定一个名称。这是通过类似下面的代码行来实现的:

```
Dim MyVariable
```

`Dim` 命令 VB.NET 创建一个名叫 `MyVariable` 的内存单元, 现在便可以在代码的其他地方使用该名称。但由于您没有告诉 VB.NET 该变量的数据类型, 因此它将创建一个 `Object` 类型的变量。要将变量声明为特定的数据类型, 可以使用类似下面的代码:

```
Dim MyVariable As String
```

**新术语:** 这被称为显式声明 (explicit declaration)。现在, VB.NET 便知道您要在该内存单元中存储一个字符串。

**提示:** 强烈建议您总是显式地声明变量。如果您明确告知 VB.NET 预留多少内存, 则它以后便无需为改变内存量而操心, 这样也使您只能对该变量执行其固有的操作。

现在, 您可以给该变量赋值:

```
MyVariable = "Hello World!"
```

您甚至可以在声明变量的同时给它赋值, 或在同一行代码中声明多个变量:

```
Dim MyVariable As String = "Hello World!"
```

```
Dim MyIntA, MyIntB, MyIntC as Integer
```

```
Dim MyIntA as Integer = 9, MyIntB as Integer = 7
```

第 1 行声明了一个名为 `MyVariable` 的 `String` 变量, 并将值 “Hello World!” 赋给它。第二行声明了三个 `Integer` 变量: `MyIntA`、`MyIntB` 和 `MyIntC`。最后, 第三行声明了两个 `Integer` 变量 `MyIntA` 和 `MyIntB`, 并将值 9 和 7 分别赋给它们。这些都是合法的声明变量的方式。

清单 3.1 列出了一个例子。

#### 清单 3.1 在 ASP.NET 中声明变量

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
```

```

4  dim MyIntA as integer = 8, MyIntB as Integer = 7
5
6  sub Page_Load(obj as object, e as EventArgs)
7      Response.Write(MyIntA * MyIntB)
8  end sub
9 </script>
10
11 <html><body>
12 </body></html>

```

在本章的后面，您将学习该清单中使用的语法。现在，您只需知道代码声明块的第 4 行声明了两个变量 `MyIntA` 和 `MyIntB`，然后在第 7 行打印它们的乘积。该清单的输出如图 3.1 所示。

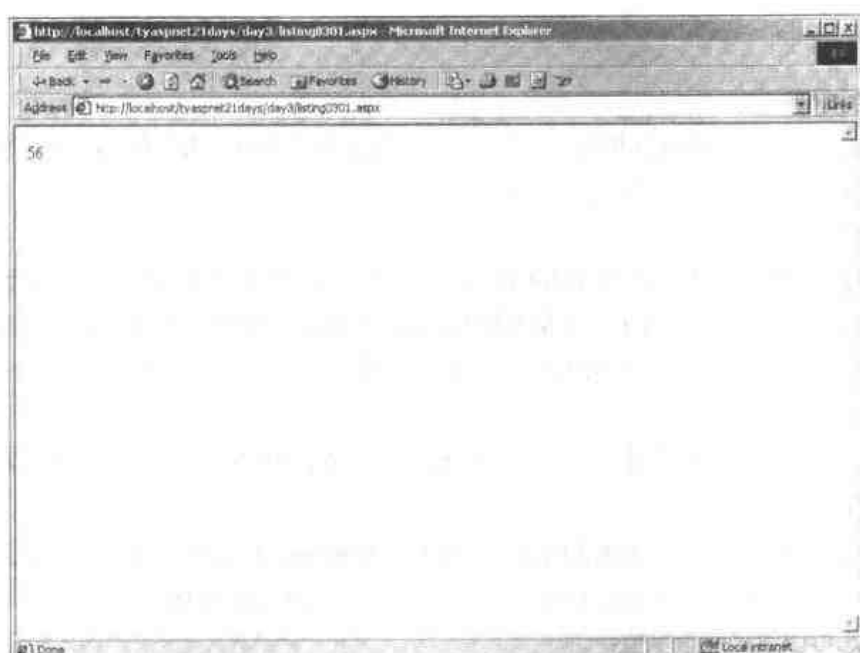


图 3.1 清单 3.1 生成的页面

### 3.2.3 变量的命名

对编程而言，正确地给变量命名至关重要。如果您有过编程的经验，则可能注意到了对变量名称的一些限制。下面对变量命名的规则做了总结：

- 不能使用空格、斜杠或句点，否则将导致应用程序出错；但可以使用下划线。
- 名称必须以字母或下划线打头。
- 名称不能是 VB.NET 中的关键字。
- 名称不能超过 255 个字符。

创建名称时，还需要遵循一些众所周知的风格，这些风格将使代码易于理解：

- 将变量的数据类型缩写作为名称的前缀，这将有助于您了解变量的用途：

```

Dim intMyInteger as Integer 'int for integer
Dim strName as String 'str for string

```

```
Dim blnGo as Boolean `bln for Boolean
```

这不需要您花多少功夫，但对您的帮助却是巨大的，让您很容易知道您处理的是哪类数据。

- 使用有意义的名称。诸如 I 或 term 等名称，在当时您也许知道其含义，但几个星期后，当您再来阅读这些代码时，将根本不知道这些变量的用途。

但也不要矫枉过正，使用诸如 intUsedToKeepTrackofMyLoopInThePage 这样的名称，只会降低您开发程序的速度。相反，应该使用诸如 intLoop 或 intIterator 这样的名称。

- 尽可能在一个地方声明所有的变量（通常是页面的开始位置），这样可以节省您查找东西的时间。

应 该	不 应 该
一定要使用能充分描述变量含义的名称	不要使用临时性的变量名称，也不要重用名称，这只会使代码混乱

### 3.2.4 数据类型转换

类型转换指的是将变量从一种数据类型变成另一种数据类型，这一过程也叫强制转换（casting）。VB.NET 能够自动转换一些数据类型（隐式转换），但对于其他类型必须明确地指定（显式转换）。

VB.NET 提供了几个将一种数据类型强制转换为另一种类型的函数，如表 3.2 所示。

表 3.2 转换函数

CBool	CByte	CChar	CDate
CDec	CDbl	CInt	CLng
CObj	CShort	CSng	CStr
CType	Asc		

例如，CByte 将某种数据类型转换为 byte，而 CStr 转换为 String。在编写 ASP.NET 页面时，这些函数很有用，您将经常使用它们。清单 3.2 列出了一个转换数据类型的例子。

清单 3.2 不强制转换数据类型的情况

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     dim strName as String = "a"
5     dim intNumber as integer = 4
6
7     sub Page_Load(obj as object, e as eventargs)
8         Response.Write("The value of strName is: ")
9         Response.Write(strName & "<p>")
10
11         Response.Write("The value of intNumber is: ")
```



```

12     Response.Write(intNumber & "<p>")
13
14     Response.Write("Their product is: ")
15     Response.Write(intNumber * strName & "<p>")
16 end sub
17 </script>
18
19 <html><body>
20
21 </body></html>

```

分析：在第 4 行和第 5 行声明了两个变量，一个变量的值为“a”，另一个变量的值为 4。前一个为 String 值，而后一个为 Integer。在第 15 行将它们相乘时，发生了错误（如图 3.2 所示），因此不能将 Integer 和 String 相乘。

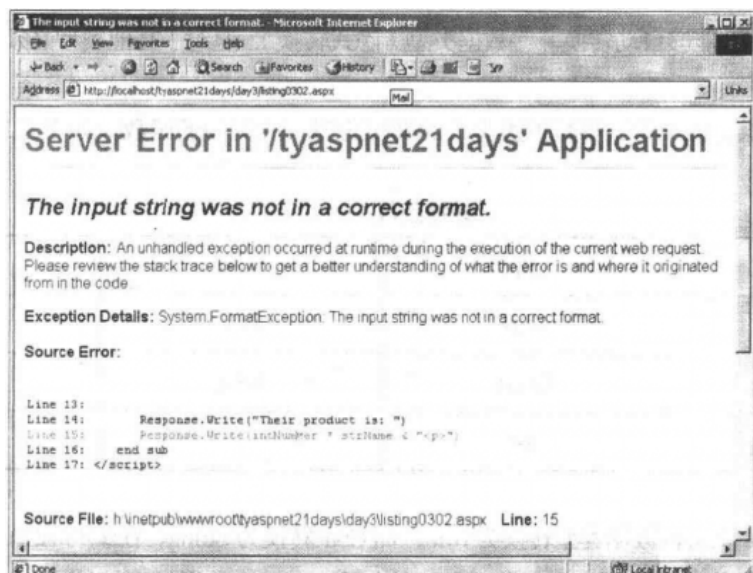


图 3.2 错误的数据类型相乘引发的错误

在 ASP.NET 中，这种情况很常见。在这个例子中，必须使用 Asc 函数将 String 值强制转换为 Integer 值，该函数将字符转换为其对应的 ASCII 码值。让我们对第 15 行进行修改：

```
Response.Write( intNumber * Asc(strName) & "<p>")
```

现在，页面将按预期的运行。

**警告：**要知道，某些转换将丢失数据。例如，如果将浮点数转换为一个整数，则小数部分将被丢失。

在 VB.NET 中，还有另一种转换数据类型的方式。许多数据类型都有让您能够将其转换为另一种特定类型的方法。这些方法通常以 To 开始，以目标类型结尾（我们将在“分支逻辑”一节中讨论这些方法）。

例如，要将 Integer 转换为 String，可以使用方法 ToString。

```

dim MyIntA as Integer = 4
dim MyString as String

```

```
MyString = MyIntA.ToString
```

但使用这些方法时一定要小心,因为有些转换是不允许的。例如,您不能使用方法 `ToInt32` 将一个 `String` 转换为 `Integer`。在本书的代码范例中,您将经常遇到这些方法。

### 3.3 数 组

在任何编程语言中,数组都是最有用的,也是最难掌握的。不同的编程语言使用不同的数组规则,因此很容易混淆。.NET 框架简化了这种处理方法,制定了一组适用于所有数组的规则。

数组是一组存储在一起的变量,可以使用其索引单独引用。设想一个蛋品盒,盒子本身是一个容器,而每个鸡蛋是变量。您可以使用索引引用其中的每一个鸡蛋:鸡蛋 1、鸡蛋 2 等等(图 3.3 说明了这一概念)。这让您能够将类似的元素存储在一起。

在 VB.NET 中,数组的索引是从 0 开始的,这意味着数组的第一个元素的索引为 0。因此,最后一个元素的索引总是比元素总数小 1。数组中所有元素都必须是同一种数据类型的——不可以混合。让我们来看一个简单的数组声明:

```
Dim MyArray(6) As Integer
Dim MyArray2() As String = {'dog','cat',"horse",_
    'elephant',"llama"}
```

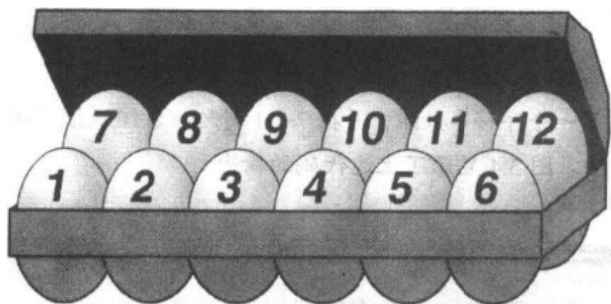


图 3.3 数组与蛋品盒很相似

第 1 行声明了一个包含 6 个 `Integer` 元素的数组(这些元素都为空)。这意味着数组最后一个元素的索引为 5。第一行的括号内的数字指出了数组中的元素数目,该数字也被称为数组长度。如果要显式地声明数组,则不能设置数组长度,并使用花括号来设置各元素的值,如第 2 行所示。第二个数组的长度为 5,“Dog”的索引号为 0,可以使用 `MyArray2(0)` 来引用它,而“llama”则可以使用 `MyArray2(4)` 来引用。

清单 3.3 列出了一个例子。

#### 清单 3.3 声明和引用数组元素

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     dim arrColors(5) as String
5
6     sub Page_Load(obj as object, e as EventArgs)
```

```
7   arrColors(0) = "green"
8   arrColors(1) = "red"
9   arrColors(2) = "yellow"
10  arrColors(3) = "blue"
11  arrColors(4) = "violet"
12
13  Response.Write("The first element is: ")
14  Response.Write(arrColors(0) & "<p>")
15
16  Response.Write("The third element is: ")
17  Response.Write(arrColors(2) & "<p>")
18
19  Response.Write("The 5-3 element is: ")
20  Response.Write(arrColors(5-3) & "<p>")
21  end sub
22 </script>
23
24 <html><body>
25
26 </body></html>
```

**分析：**第 4 行声明了一个字符串数组，并在 7-11 行给该数组赋值。第 14、17 和 20 行演示了各种引用数组元素的方法。图 3.4 显示了上述代码的输出。

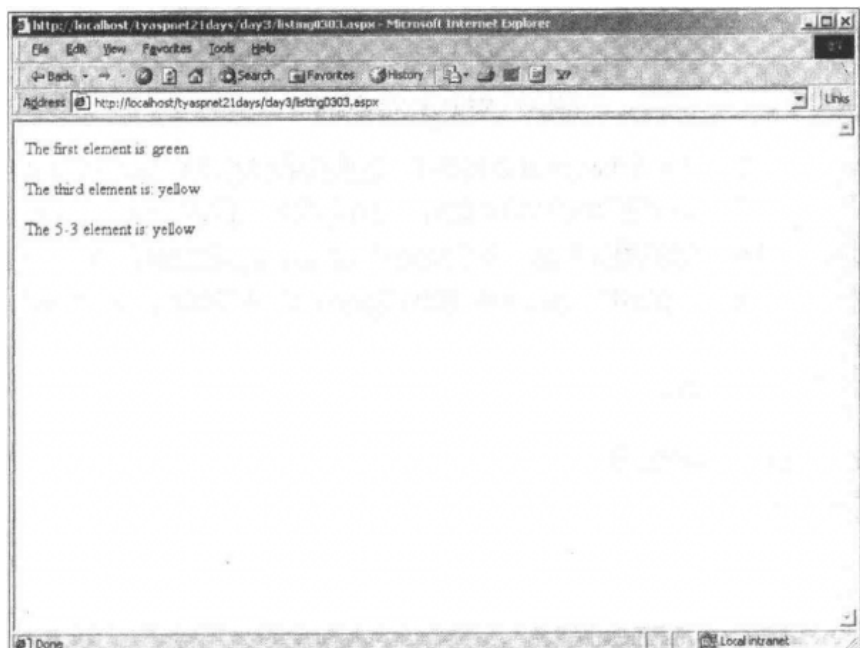


图 3.4 各种存取数组元素的方法

在VB.NET中，有两个函数专门用于操纵数组：Redim 和 Erase。

数组声明后，其长度便固定了，但可以使用 Redim 修改数组的长度，例如：

```
Redim arrColors(6)
```

现在，数组 arrColors 的长度变成了 6。但在上述过程中，原有的值将被破坏。可以使用关键字 preserve 来保留这些值：

```
Redim Preserve arrColors(6)
```

如果数组的长度缩短了，则超出新数组范围的值将被丢弃。如果数组被增长了，则新增元素的值将被初始化为默认值。

**新术语：**Erase 语句将数组中每个元素的值都设置为 Nothing，在 VB.NET 中，这指的是变量中没有保存任何值：

```
Erase arrColors
```

**注意：**VB.NET 中的 Array 类提供了更多的操作数组的函数。更详细的信息，请参考 .NET 框架 SDK 文档。

## 3.4 操作符

操作符是一种用于执行某种运算的符号。例如，= 操作符可用于赋值：

```
strName = "Hello"
```

您一定熟悉许多操作符，因为在日常生活中经常使用它们。表 3.3 按优先级次序列出了 VB.NET 中的所有操作符。

表 3.3 VB.NET 操作符

功 能	操 作 符
求幂	^
正负	+, -
乘除 (62=333)	*, \
除以 (62=3)	/
求模 (6 mod 4=2)	Mod
加减	+, -
按位 NOT、AND、OR 和 XOR	BitNot, BitAnd, BitOr, BitXor
连接	&, + (字符串)
等于、不等于、小于、大于	=, <>, <, >
小于等于、大于等于	<=, >=
关系运算	TypeOf, Is, Is, Like
赋值	=,  =, *=, /=, \=, +=, -=, &=
逻辑 NOT、AND、OR 和 XOR	NOT, AND, OR, XOR

可以使用括号改变优先级。例如， $4+5*3=19$ ，而 $(4+5)*3=27$ 。

## 3.5 条件逻辑

条件逻辑让您能够根据满足的条件执行相应的代码。这是一种非常强大的机制，几乎在所有的应用程序中都是必须可少的。本节将讨论几种处理逻辑条件的方法：if 语句和 case 语句。

### 3.5.1 If 语句

If 语句是最简单的条件逻辑，其处理流程非常简单：如果某件事情发生或某个条件满足，则执行某种操作。

下面是一个真实世界中的 If 语句的例子：假设您在一家钟表厂的装配线上工作，您的职责是将钟表组装起来。老板告诉您“如果时针断了，则丢弃该钟表”。

让我们来看看语法：

```
if (condition) Then
    some code
end if
```

如果条件满足，则执行 if 和 end if 之间的代码行。否则，继续执行后面的代码。清单 3.4 列出了一些例子。

**清单 3.4 简单的 If 语句**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         dim MyMessage As String = 'Hello'
6         dim MyBool As Boolean = True
7
8         'if-statement 1
9         if MyMessage = 'Hello' then
10             Response.Write("True")
11         end if
12
13         'if-statement 2
14         if MyBool then
15             Response.Write("True")
16         end if
17     end sub
18 </script>
19
20 <html><body>
21
```

```
22</body></html>
```

分析: 第5行和第6行声明了两个变量 MyMessage 和 MyBool。在第一个 if 语句 (第9-11行) 中, 对一个简单的表达式进行评估: 字符串变量 MyMessage 的值是否为 “Hello”? , 如果是, 则使用 Response.Write 将 “True” 写入到 Web 浏览器中。

第2个 If 语句 (第14-16行) 演示了一种简写方式, 当您只想知道某个变量是否为 true 时, 可以写作 If variable then, 而不是 if variable = TRUE then。相反, 当您想确定一个值是否为 false 时, 可以使用关键字 not, 如 if not variable then。

回到关于钟表厂的例子, 由于工厂在毁坏的钟表上浪费的钱过多, 老板决定改变规则: “如果时针断了, 则丢弃钟表; 如果分针断了, 则修复它; 如果其他元件出了问题, 则将其放在一个盒子中。”

关键字 elseif 让您能够增加一个 “或者” 条件, 如 “或者如果分针断了”。关键字 else 让您处理其他的情况, 如 “如果其他东西出了问题...” 清单3.5列出了一些例子

### 清单3.5 If...Then....Else 语句

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         dim MyMessage As String = "Hello"
6         dim MyBool As Boolean = True
7
8         if-statement 3
9         if MyMessage = "Hi" AND MyBool then
10             Response.Write(MyMessage)
11         elseif MyMessage = "Hello" then
12             Response.Write("True")
13         else
14             Response.Write ("False")
15         end if
16
17         'if-statement 4
18         if not MyBool Then Response.Write( "False")
19     end sub
20 </script>
21
22 <html><body>
23
24 </body></html>
```

分析: 第三个 If 语句 (第9-15行) 使用关键字 elseif 和 else 评估其他条件。第9行使用关键字 AND 判断 MyMessage 是否为 “Hi”, Mybool 是否为 true。如果上述两个条件都满足, 则将 MyMessage 写入浏览器中。或者, 如果 MyMessage 为 “Hello”, 则将 “True” 写到浏览器中, 如第11-12行所

示。最后,如果上述两种情况都不满足,则将“False”写入浏览器中。这使得您能够实现非常复杂的条件逻辑。

第四个 If 语句(第 18 行)是 If 语句的另一种简写方式。可以将整个 If 语句放在一行中,并省略 end if。

使用 If 语句可以完成很多工作,但有时候使用它很不方便,甚至根本无法完成您要实现的工作。在这种情况下,可使用 case 语句。

### 3.5.2 Case 语句

Case 语句有时候也叫选择(Select)语句,基本上与带 elseif 子句的 If 语句相同。Case 语句检查一个变量,您可以指定变量满足某种条件时应执行的操作。让我们来看看其语法:

```
Select Case variable
    Case option 1
        code
    Case Else
        code
End Select
```

您可以根据需要,指定任意多种情况(case),并使用 case else 来捕获其他情况。例如:

```
Select Case FirstName
    Case "Ringo"
        Response.Write('Drummer')
    Case "Paul"
        Response.Write('Not the drummer')
    Case Else
        Response.Write('Another Beetle')
End Select
```

也可以在一行中指定多种情况。让我们再次回到钟表厂的例子。现在,老板规定,如果钟表的时针、分针或秒针断了,则丢弃它;如果发条坏了,则替换它即可;如果是其他东西坏了,则将钟表放进盒子。对于这种情况,其代码与清单 3.6 类似。

#### 清单 3.6 处理钟表的 case 语句

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         dim strClockStatus As String
6         strClockStatus = "MinuteHandBroken"
7
8         select Case strClockStatus
9             case "MinuteHandBroken", "HourHandBroken", _
10                "SecondHandBroken"
11                 Response.Write("Throw clock away")
```

```

12         case "SpringBroken"
13             Response.Write("Replace spring!")
14         case else
15             Response.Write("Put clock in box.")
16     end select
17 end sub
18 </script>
19
20 <html><body>
21
22 </body></html>

```

该清单打印“Throw clock away.”，清单 3.7 的打印结果是什么呢？

#### 清单 3.7 case 语句的执行顺序

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as EventArgs)
5         dim intAge As Integer = 7
6
7         select Case intAge
8             case "7"
9                 Response.Write("That's a string!")
10            case 7
11                Response.Write("You're 7 years old.")
12            case <10
13                Response.Write("Wow, you're young!")
14        end select
15    end sub
16</script>
17
18 <html><body>
19
20 </body></html>

```

**分析：**清单 3.7 的打印结果为“*That's a string!*”，其中的每种条件都满足，但只有第一条语句被执行。因此，一定要确保条件不相互重叠，并且将最具体的放在前面。

应 该	不 应 该
当需要检查变量是否满足多个条件时，请使用 case 语句	不要使用大量的 if 语句来检查一个变量



## 3.6 循环逻辑

循环让您不断执行某个操作，直到条件满足为止。VB.NET 中有三种循环：While 循环、Do 循环和 For 循环。

### 3.6.1 While 循环

While 循环在您不知道要循环多少次时很有用，如当您想查找小汽车，直到发现奔驰为止。这些循环是基于条件表达式的，它们将不断运行，直到条件为假为止。While 循环的语法如下：

```
While condition
```

```
code
```

```
End While
```

例如，下述代码从 1 数到 10，并将这些数字打印到浏览器中。

```
dim intCount as Integer = 1
```

```
While intCount < 10
```

```
    Response.Write(intCount & "<br>")
```

```
    intCount = intCount + 1
```

```
End While
```

在第 2 行，程序检查 intCount 是否大于 10。该条件不成立，因为您刚把它设置为 1。程序将执行 While 循环中的代码，将数字写到浏览器中，并将 intCount 的值加 1。然后，返回到第二行，并检查 intCount 是否大于 10。现在，intCount 的值为 2，因此循环将继续进行。循环结束后，数字 1-9 将被写入到浏览器中。图 3.5 说明了这种过程。

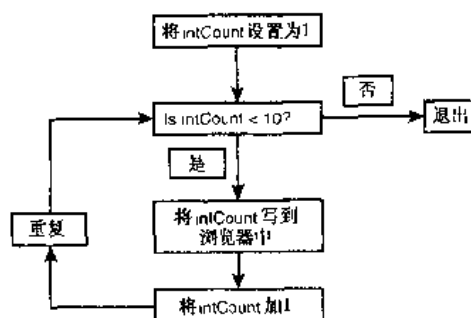


图 3.5 循环过程

为何最后写入的数字为 9，而不是 10 呢？因为 intCount 的值为 10 后，第 2 行的条件便不再为 true，因此将退出循环，并继续执行第 5 行的代码。

Do 循环与 While 循环相同，其语法稍微有些不同。Do 循环有多种格式，例如：

```
Do
```

```
code
```

```
Loop While condition
```

下面是另一个格式：

```
Do While condition
```

```
    code
```

```
Loop
```

不同格式得到的结果也不同，这取决于您想执行循环多少次。让我们来看看每种格式的例子：

```
dim intCount as Integer = 10
```

```
Do
```

```
    Response.Write(intCount & "<br>")
```

```
    intCount = intCount + 1
```

```
Loop While intCount < 10
```

```
Do While intCount < 10
```

```
    Response.Write(intCount & "<br>")
```

```
    intCount = intCount + 1
```

```
Loop
```

第一个循环打印数字 10，而第二个循环不打印任何东西。原因何在呢？第一个循环在最后一行才判断条件，因此，不管情况如何，该循环将至少执行一次；而第二个循环在开始就判断条件，在这个例子中，后面的代码将不会被执行。

当您希望循环块中的代码至少执行一次时，应使用 Do 循环；相反，如果根本不关心循环块中的代码是否执行，则应该使用 Do While 循环。也可以使用 until 代替 While，如下所示：

```
dim intCount as Integer = 1
```

```
Do
```

```
    Response.Write(intCount & "<br>")
```

```
    intCount = intCount + 1
```

```
Loop Until intCount >= 10
```

该循环的效果与前述清单中的第一个循环完全相同，区别在于，使用 until 时，循环将一直执行下去，直到条件为真；而使用 While 时，当条件为假时，循环将结束。在这个例子中，当 intCount 的值大于或等于 10 时，循环将结束。

### 3.6.2 For 循环

当您知道代码的执行次数时，可以使用 For 循环。这种循环将逐渐增加计数器的值，以告诉自己何时退出循环。For 循环的语法如下：

```
For variable = startvalue To stopvalue [Step step-size]
```

```
    code
```

```
Next [variable]
```

该循环每次将 Variable 的量递增 step-size，直到到达 stopvalue 为止，至此循环将结束（如果愿意，可以省略 step 参数，其默认值为 1）。例如：

```
For intCount = 1 to 10
```

```
    Response.Write(intCount & "<br>")
```

```
Next
```

或者：

```
For intCount = 10 to 1 step -1
```

```
Response.Write(intCount & "<br>")
Next
```

第一个循环打印 1-10 的数字，而第二个循环以相反的次序打印这些数字

另一种格式的 For 循环是 for...each 循环。该循环遍历集合（如数组）中的所有元素。例如：

```
dim arrWeekDays() as String = {"Monday", "Tuesday", _
    "Wednesday", "Thursday", "Friday"}
For Each strDay in arrWeekDays
    Response.Write(strDay & "<br>")
Next
```

第 1 行声明了一个字符串数组，其中包含星期一到星期天的名称。第四行遍历数组中的每个元素。变量 strDay 只是循环使用的一个计数器——您可以使用任何变量。在第五行循环将数组中的每个变量赋给 strDay，并将其打印出来。结果，从星期一到星期天的名称被显示在浏览器中，如图 3.6 所示。

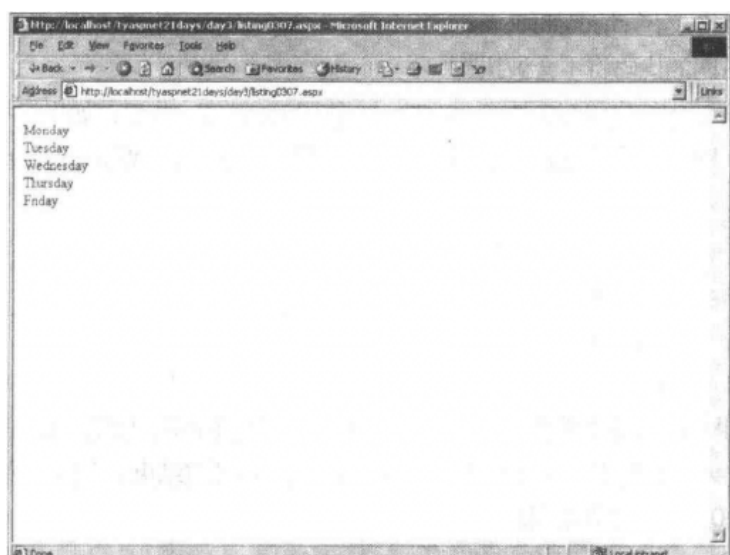


图 3.6 使用 For...each 循环遍历数组中的每一个元素

### 3.6.3 死循环

**新术语：**在 While 循环和 Do 循环中，需要编写增加计数器值的代码，而在 for 循环中，这是自动完成的。在循环中，让计数器递增至关重要，否则将导致死循环（infinite loop）——循环永远不会结束。这不但对于网站的访问者是一种痛苦，而且可能很快耗尽系统资源，导致整个站点崩溃。

如果要中途退出循环，可以使用关键字 exit 来结束循环。该关键字的语法根据其使用位置而异。例如：

```
Do While intCount < 10
    Response.Write(intCount & "<br>")
    intCount = intCount + 1
    if intCount = 7 then
        exit do
    end if
End Do
```

```

    end if
Loop
For intCount = 1 to 10
    Response.Write(intCount & "<br>")
    if intCount = 7 then
        exit for
    end if
Next

```

第一个循环从 1 到 10，您使用 `exit do` 在数字为 7（在这个例子中，肯定会出现这样的情况）时结束循环（注意该循环只打印 1-6）。第二个循环与第一个循环完全相同，只不过使用的是 `exit for`（它打印 1-7）。使用何种 `exit` 语句取决于要退出的是哪种循环。

### 3.7 分支逻辑

分支逻辑使代码转移到其他地方。通常，当需要在不同的地方重复使用相同的代码时，将使用这种逻辑。

在钟表厂的例子中，每当钟表组装好后，都要将其放到盒子中。而装箱机位于地下室，钟表组装好后，您将其送到地下室，进行装箱，然后您再计算其邮资。装箱机就是一种分支逻辑——可以在工厂的任何位置使用它，而不仅仅是在装配线上。将装箱机放在地下室，比在每一根装配线附近都放置一台装箱机的效率要高得多。

在 VB.NET 中有两种分支逻辑控件：函数和子程序，它们之间的区别在于函数能返回信息，而子程序不能。换句话说，函数计算值，而子程序执行操作。对于 ASP.NET 开发而言，这两种控件是必不可少的，没有它们，将无法完成任何工作。我们首先介绍子程序。

#### 3.7.1 子程序

子程序（有时候也叫过程）的格式如下：

```

Sub name (parameters)
    code
End Sub

```

您已经在 ASP.NET 页面中见过这种语法，还记得以下代码行吗？

```

sub Page_Load(obj as object,e as eventargs)
    code
end sub

```

清单 3.8 包含一些将数字相乘的代码。

**清单 3.8 在子程序中将数字相乘**

```

1 <%@Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object,e as eventargs)
5         MultiplyNumbers(8,9)

```

```
6
7     MultiplyNumbers(4,12)
8
9     MultiplyNumbers(348,23)
10 end sub
11
12 sub MultiplyNumbers(intA as integer,intB as integer)
13     Response.Write(intA *intB &"br>")
14 end sub
15 </script>
16
17 <html><body>
18
19</body></html>
```

**分析：**在代码声明块的第 12 行，声明了一个名为 MultiplyNumbers 的子程序。括号中的变量被称为参数，它们是调用该子程序的代码传递给子程序的值，然后子程序使用这些值执行所需的操作。一组参数被称为参数列表。指定这些参数与声明变量类似，您在这里声明的参数必须是由调用子程序的代码传递过来的，否则将出错。

第 13 行使用 Response.Write 将参数的乘积写入到浏览器中，并在第 14 行结束子程序。创建子程序后，便可以在页面的任何地方使用它。除了明确指示，否则子程序将不会执行——子程序不会因为您创建了它便自动执行。

**新术语：**第 5 行调用子程序。它命令程序执行子程序中的代码，完成后再回到第 5 行（在这里，您可以看到分支逻辑是管用的）。调用 MultiplyNumbers 时，指定了子程序定义的两个参数——两个 Integer。MultiplyNumbers 使用这两个整数执行其操作。

第 7 行和第 9 行完成相同的工作，只是传递给子程序的参数不同。这是分支逻辑控件的好处之一，即可以在任何地方调用它们，并且每次传递不同的参数。上述代码清单的输出如图 3.7 所示。

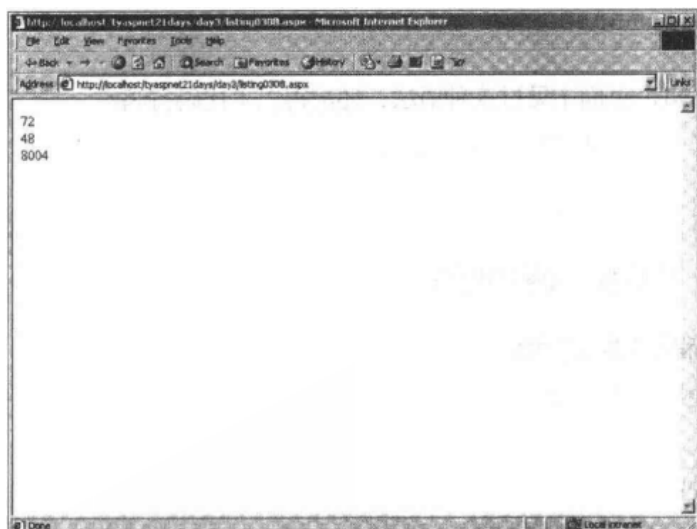


图 3.7 使用子程序执行封装的逻辑

**警告：**如果您熟悉VBScript和传统ASP，则可能还记得，调用子程序和函数的语法如下：

```
MultiplyNumber 8, 9
```

也就是说没有括号。在VB.NET和ASP.NET中，调用函数时，必须使用括号，否则将出错。

### 3.7.2 函数

函数与子程序很相似——它们的语法几乎相同，而且可以执行相同的操作。但函数将给调用它的代码返回一个值。我们将清单 3.8 进行修改，以使用函数而不是子程序。

**清单 3.9 在函数中将数字相乘**

```
1 <%@Page Language='VB' %>
2
3 <script runat="server">
4     sub Page_Load(obj as object,e as eventargs)
5         Response.Write(MultiplyNumbers(8,9)&"<br>")
6
7         Response.Write(MultiplyNumbers(4,12)&"<br>")
8
9         Response.Write(MultiplyNumbers(348,23)&"<br>")
10    end sub
11
12    function MultiplyNumbers(intA as integer,intB as integer) _
13        as Integer
14        Return intA *intB
15    end function
16 </script>
17
18 <html><body>
19
20 </body></html>
```

**分析：**上述代码与清单 3.8 稍微有些不同。首先第 12 行的函数没有将两个参数的乘积写入到浏览器中，而是使用关键字 `return` 将乘积返回给调用它的代码。注意，该函数在参数列表的后面还指定了 `As Integer`，它用于告诉调用代码返回值属于什么数据类型。该清单的结果与前一清单相同。

第 5 行在 `Response.Write` 中调用函数 `MultiplyNumbers`，乍一看，这好像有些古怪，让我们仔细看看。首先，第 5 行使用参数 8 和 9 调用 `MultiplyNumbers`，在第 14 行，该函数将这两个参数相乘，并将结果返回到第 5 行。现在，第 5 行的 `Response.Write` 看到的只是乘积 72，等价于 `Response.Write(72)`。

这让您可以在许多不同的地方调用函数。例如，您可以使用下面的任何一行代码。

```
MultiplyNumbers(8,9)
intCount = MultiplyNumbers(8,9) * 72
if MultiplyNumbers(8,9) < 80 then
    ...
```

这些方法都是有效的，因为函数将首先执行，最终看到的是返回值。在 VB.NET 中，返回值的方式有两种，第一种方法您已经知道了——使用关键字 `return`；第二种方法是将返回值赋给函数本身，例如下述代码的功能与清单 3.9 中的函数相同：

```
12 function MultiplyNumbers(intA as integer, intB as integer) _
13     as Integer
14     Return intA * intB
15 end function
```

函数并不一定要返回一个值，但如果需要，也可以返回。可以像清单 3.8 中那样，将 `Response.Write` 放在函数中，清单 3.9 的做法只是为了演示说明而已。另外，还可以指定返回值的数据类型：

```
function MultiplyNumbers(intA as integer, intB as integer) _
    as Integer
```

上述代码指出函数将返回一个 `integer`。

**提示：**声明函数时一定要指定返回类型，这可以提高代码的可读性和类型安全。

### 3.7.3 可选参数

如果需要在函数或子程序中使用一个参数，但不要求调用时必须传递该参数，怎么办呢？可以将该参数指定为可选的。幸运的是，VB.NET 使用关键字 `Optional` 来指定可选参数。来看一看下面的例子：

```
function MultiplyNumbers(intA as integer, optional intB as _
    integer = 3) as Integer
    return intA * intB
end function
```

第一行使用关键字 `optional` 告诉 VB.NET，调用该函数时，可以不指定 `intB` 的值，但也可以指定。这样，便可以以下面任何一种方式调用该函数：

```
MultiplyNumbers(8)
MultiplyNumbers(8,9)
```

如果没有指定值，情况将如何呢？第三行将两个值相乘，如果 `intB` 的值没有被指定，则将发生错误。

将参数指定为可选的时，必须同时指定默认值。第一行的 `optional intB as integer = 3` 告诉 VB.NET，如果未指定该参数的值，则其值为 3。

使用 `optional` 时需要注意的一点是，将某个参数指定为可选的后，其后面的所有参数都必须是可选的。下述代码将发生错误：

```
function MultiplyNumber(intA as integer, optional intB as _
    integer = 3, intC as integer) as Integer
```

正确的编写方式是：

```
function MultiplyNumber(intA as integer, optional intB as _
    integer = 3, optional intC as integer = 4) as Integer
```

### 3.7.4 事件处理程序

正如在第 1 章和第 2 章介绍的，事件是在应用程序中执行的操作——如单击鼠标或单击按钮。

通常，每当事件发生时，都需要进行一些处理。例如，当 Submit 按钮被单击时，应该将表单中的信息加入到数据库中；当用户单击超连接时，应跳到相应的页面。这种功能不是自动完成的——您必须告诉程序如何做。

**新术语：**这是通过创建事件处理程序（event handler）实现的。事件处理程序的语法与子程序相同，因此您应该很熟悉。不同之处在于参数列表。当事件被引发（raised）时——意味着事件已经发生——将生成描述事件的变量。事件处理程序可以通过使用这些变量弄清楚应如何做。

在 ASP.NET 中，几乎所有的事件生成的参数列表都一样。清单 3.10 显示了一个例子。

**清单 3.10 处理 ASP.NET 中的事件**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub Button_Click(obj As Object, e As EventArgs)
5         Response.Write("You clicked the button!")
6     End Sub
7 </script>
8
9 <html><body>
10     <form runat="server">
11         <asp:button id="btSubmit" Text="Submit"
12             runat="server"
13             OnClick="Button_Click" /><p>
14     </form>
15 </body></html>
```

**分析：**第 4 行的代码像一个常规的子程序，但参数列表表明它是一个事件处理程序。第一个参数的数据类型为 Object，它指出了引发该事件的对象。第二个参数是一种您没有见过的新类型 EventArgs，该参数包含发生的事件特有的所有信息，通常该变量为空。这是标准的事件处理程序参数列表，虽然在几章后，您将看到例外的情况。

**注意：**记住可以将参数指定为任何一个名称，这里使用 Obj 和 e 是出于简单和标准化考虑的。

第 11 行创建了一个 ASP.NET 按钮控件。您可能还记得，第 1 章和第 2 章介绍过该控件，第 5 章将深入介绍它。现在，您只需知道您将其 ID——唯一名称——设置为 btSubmit，将其显示的文本设置为 Submit，并指定它将在服务器上运行。每当该按钮被单击时，都将引发一个名为 Click 的事件。第 13 行告诉按钮：当 Click 事件发生时，执行子程序 Button\_Click。然后定义其事件处理程序。

这样，每当按钮被单击时，都将执行第 4~6 行的子程序，在浏览器中显示“You clicked the button!”。将上述代码保存到文件 listing0310.aspx 中，并在浏览器中查看它。单击按钮后，将看到如图 3.8 所示的画面。

上面介绍的是处理 ASP.NET 页面中所有操作的基本原理。一旦您知道如何处理事件后，ASP.NET 便会强大得多。让我们对清单 3.10 稍作修改，以使用提供给事件处理程序的参数。



## 清单 3.11 在 ASP.NET 中使用事件参数

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub Button_Click(obj As Object, e As EventArgs)
5         Response.Write(obj.Text)
6     End Sub
7 </script>
8
9 <html><body>
10     <form runat="server">
11         <asp:button id="btSubmit" Text="Submit"
12             runat=server
13             OnClick="Button_Click"/><p>
14     </form>
15 </body></html>

```

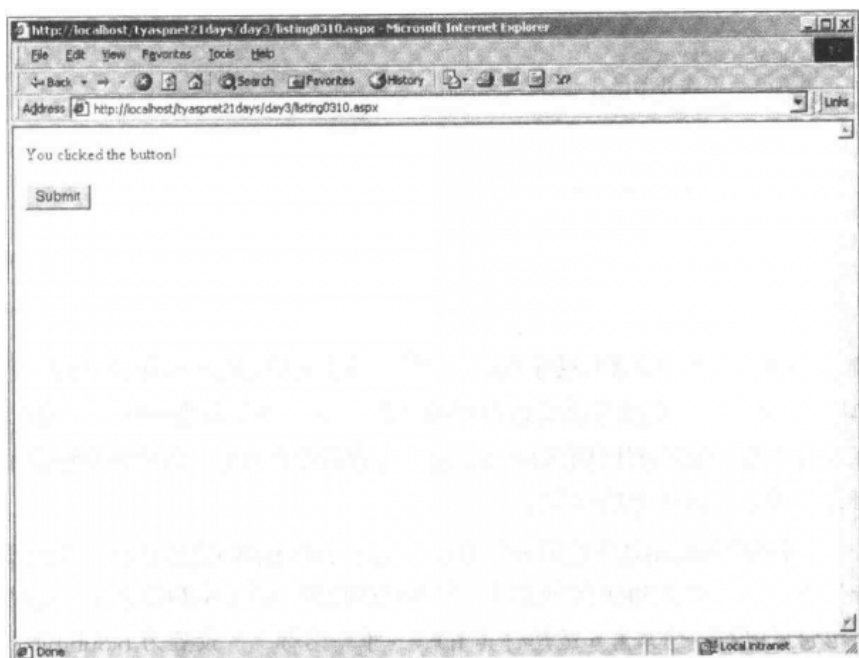


图 3.8 在 ASP.NET 中处理事件

**分析:** 当您查看该页面, 并单击按钮后, 将在“*You Clicked the button!*”的位置出现单词“*Submit*”。让我们看看第 5 行的代码。第一个参数代表的是引发事件的对象, 在这个例子中是 *Submit* 按钮。因此, *obj* 是可以用来引用该按钮的名称之一, 另一个可用来引用该按钮的名称是 *id* 值 *btSubmit*。第 5 行将按钮的 *Text* 属性写入到浏览器中。在第 11 行, *Text* 属性被设置为“*Submit*”, 可以将第 5 行改成下面的样子, 其结果相同。

```
Response.Write(btSubmit.Text)
```

我们有些超前了，在第5章“Web 表单初步”将做更详细的介绍。

**警告：**通常只有当引发代码位于标记<form>... </form>中时，ASP.NET事件才会被处理，这是因为事件必须发送给服务器，这将在第5章“Web表单初步”中做更详细地讨论。

在本章的例子中，您使用得较多的另一个事件是 Page\_Load。例如，在清单 3.9 中，第 4 行是一个名叫 Page\_Load 的方法，其中包含标准的事件参数列表：obj as object, e as eventargs。当 ASP.NET 页面装载时，Page\_Load 事件将被引发，可以使用它来执行一些处理工作。通常，您可以使用该事件处理程序在页面装载时将某些信息显示给用户。第 4 章“在 C#和 VB.NET 中使用 ASP.NET 对象”将更详细地讨论该事件。

### 3.8 类

类是对象的定义。例如，钟表的定义如下：包含时针、分针和秒针，整点报时，并可以设置在其他时间报时。同样，类通过将对象包含的内容及其功能告诉其他代码来定义对象。

在面向对象编程（和 ASP.NET）中，类是必不可少的。定义类是为了将一些通用代码和属性包装起来，就像将钟表零件组装成钟表一样。事实上，.NET 框架包含许多定义对象的类，虽然您还没学习它们。使用 Response.Write 相当于调用 Response 对象的子程序 Write，该对象是由 Response 类定义的。更值得注意的是，当您创建 ASP.NET 页面时，实际上是创建了一个在.NET 框架中基于 Page 类的对象。

**提示：**类是定义，而对象是基于这些定义的、真正可用的东西，知道这种区别非常重要，虽然有时候这两个术语被交替使用。

在.NET 框架和 VB.NET 中，任何东西都是用类表示的，因此必须习惯用这种方式引用它们，这至关重要。

那么如何创建自己的类呢？其语法很简单：

```
Class name
    subroutines, functions, and properties
End Class
```

创建类的关键是将通用的代码组合起来。清单 3.12 包含一个关于钟表的简单例子。

**清单 3.12 一个 ASP.NET 钟表类**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Class Clock
5         public Second as integer
6         public Minute as integer
7         public Hour as integer
8
9         sub SetTime(intSec as integer, intMin as integer,
10             intHour as integer)
11             Second = intSec
```

```

12         Minute = intMin
13         Hour = intHour
14     end sub
15 End Class
16
17 sub Page_Load(obj as object, e as EventArgs)
18     dim objClock as new Clock
19
20     objClock.Second = 60
21
22 end sub
23
24 </script>
25
26 <html><body>
27
28 </body></html>

```

**分析：**在类中声明的所有变量（如第5-7行所示）都被称为类的属性。这个类包含属性 second、minute 和 hour 以及一个设置时间的子程序。第20行将 Second 属性的值设置为 60。

关键字 Public 告诉 VB.NET，对任何代码而言，该条目都是可见的、可修改的，无论这些代码是位于类的内部还是外部，这被称为全局（global）变量。相反，如果使用关键字 private，则相应的变量或子程序只能在类中使用。

**新术语：**这对于防止其他代码修改您依赖的变量很有帮助，但这方面的内容不在本书的讨论范围之内——更详细的信息，请参考面向对象编程方面的指南。

方法 SetTime 将类的 second、minute 和 hour 属性设置为相应的输入参数。在 ASP.NET 页面中，可以这样做：

```

Dim objClock as new Clock
objClock.SetTime(60,4,12)
objClock.Second = 59

```

第一行声明了变量 objclock，其数据类型是基于 Clock 类的。然后便可以使用变量 objClock 设置类的属性或调用类的方法，如第2-3行所示。通过像前面那样包装通用代码，可以将编程工作看作仅仅是操作对象。

**注意：**这里是将类放在代码声明块中，但通常将其放在另一个文件中，以使代码更容易理解。对于 VB.NET 类，文件的扩展名通常为 .vb，这将在第15章“使用业务对象”中做更详细的介绍。

#### New 的功能

关键字 new 至关重要——它用于初始化新对象。例如，下述代码声明了一个数据类型为 Clock 的变量，但未初始化该变量（也就是说，没有给该变量赋值）。

```
Dim objClock as Clock
```

关键字 new 给变量赋值，其功能与下面的语句类似：

```
dim intNumber as Integer = 7
```

要初始化对象变量，也可以使用类似下面的代码：

```
dim objClock as Clock = New Clock
```

这与只使用关键字 `new` 的效果完全相同。

### 继承

继承是类和面向对象编程的重要组成部分。这里不打算详细讨论它，但您应该知道一些 ASP.NET 的基础知识。

**新术语：**假设您创建了一个 `Clock` 类，随后您发现有两种不同的钟表：模拟的和数字的。您可以创建两个基于 `Clock` 的新类，而不是将 `Clock` 类放在一边，重新创建两个新类。原来的 `Clock` 类包括所有钟表共有的属性和方法，因此只需在新类中加上其需要的属性和方法即可，这被称为继承。

在开发过程中，继承省去了许多麻烦。当存在功能与您所需的类似的类时，您不用创建一个全新的类。让我们来看一个使用原来的 `Clock` 类的例子：

```
Class AnalogClock : Inherits Clock
    private ClockWound as Boolean = false
    sub WindClock()
        ClockWound = true
    end sub
End Class
```

您创建了一个新类 `AnalogClock`，它是从原来的 `Clock` 类继承而来的。这个新类自动继承了属性 `second`、`minute` 和 `hour` 以及方法 `SetTime`，因此您不必重新创建它们。您只需添加这个类所特有的项目即可：一个布尔值 `ClockWound` 和一个给钟表上发条的方法。ASP.NET 页面将与下面类似：

```
dim objAnalogClock as new AnalogClock
objAnalogClock.SetTime(50, 4, 12)
objAnalogClock.WindClock
```

随继承而来的是覆盖父类方法的能力。如果由于某种原因，`AnalogClock` 类的方法 `SetTime` 应该不同于父类，则只需使用相同的名称提供一个不同的方法即可。

```
Class AnalogClock : Inherits Clock
    private ClockWound as Boolean = false
    sub WindClock()
        ClockWound = true
    end sub
    sub SetTime(intSec as integer, intMin as integer, _
        intHour as integer)
        Second = intSec
        Minute = intMin
        Hour = intHour - 12
    end sub
End Class
```

这样，代码 `objAnalogClock.SetTime` 将使用修改后的子程序。

继承是一种奇妙的机制，但它非常复杂，本书并不打算详细介绍它。您将在 ASP.NET 页面（特别是在 `code-behind` 表单，这将在第 6 章中进行介绍）中使用它。

### 3.9 使用 VB.NET 函数

本节简要描述一些常用的、并很有用的 VB.NET 函数，贯穿本书，您都将使用它们。本书决不是函数的完整参考，而只是提供一些概要性的信息。

表 3.4、3.5 和 3.6 将许多常见的函数做了简介。

**表 3.4** 日期/时间函数

函 数	描 述
<code>dateDiff(dateinterval,date1, date2[,firstdayofweek[,firstdayofyear]])</code>	返回一个数字,指出 date1 和 date2 之间的 dateinterval。dateinterval 可以是 yyyy(年)、q(季度)、y(每年的天数)、d(天)、w(工作日)、ww(星期)、h(小时)、n(分钟)、s(秒)
<code>day(datetime)</code>	返回 1-31 的整数,指出是某个月的哪一天
<code>dayofweek</code>	返回一个整数,指出是某个星期的哪一天(星期天为 0,星期六为 6);这是数据类型 datetime 的一个属性
<code>hour(time)</code>	返回一个 0-23 的整数,指出某天中的哪个小时
<code>isdate(datetime)</code>	返回一个布尔值,指出所提供的 datetime 是否是一个有效的日期
<code>minute(time)</code>	返回一个 0-59 的整数,指出时间中的分钟部分
<code>month(datetime)</code>	返回一个 1-12 的整数,指出是几个月
<code>now()</code>	返回一个 datetime 值,指出计算机当前的日期和时间
<code>second(time)</code>	返回一个 0-59 的整数,指出时间中的秒部分
<code>year(datetime)</code>	返回一个代表年份的整数(1-9999)

**表 3.5** 数学函数

函 数	描 述
<code>abs(value)</code>	value 的绝对值
<code>atan(value)</code>	value 的反余切
<code>cos(value)</code>	value 的余弦
<code>exp(value)</code>	e <sup>value</sup>
<code>fix(value)</code>	返回整数部分,对于负数,则向上舍入
<code>hex(value)</code>	返回 value 对应的 16 进制数
<code>int(value)</code>	返回整数部分,对于负数,则向下舍入
<code>log(value)</code>	返回 value 的自然对数

续表

函 数	描 述
oct(value)	返回 value 对应的 8 进制数
rnd	返回随机数
round(value [, dec])	返回整数部分, 或保留 dec 位小数
sin(value)	value 的正弦
sqr(value)	value 的平方根
tan(value)	value 的余切

表 3.6 字符串函数

函 数	描 述
instr(start, string1, string2[, compare])	返回 一个数字, 指出 string2 第一次出现在 string1 中的位置, 如果没有出现, 则返回 0。Compare 可以是 0 (二进制比较) 或 1 (文本比较)
left(string, length)	返回一个字符串, 其中包含从 string 的左边开始的 length 个字符。
len(string   variable)	返回一个数字, 指出字符串的长度或存储变量需要的空间 (字节数)。
mid(string, start[, length])	返回 一个字符串, 其中包含 string 中指定数目的字符: <code>dim strstring as string = mid("hello", 3)</code> 上述代码将返回 "llo"
replace(expression, find, replace[, start[, count[, compare]])	从 start 开始, 将 expression 中的 find 替换为 replace, count 是替换次数, 默认为 -1 (全部替换), compare 与 instr 中的情况相同
right(string, length)	返回一个字符串, 其中包含从 string 的右边开始的 length 个字符

### 3.10 给未来的 VB.NET 高手：到哪里查找参考资料

Visual Basic.NET 是一种功能强大、对开发人员友好的编程语言, 它让您能够创建复杂的企业级 ASP.NET 应用程序。虽然本章没有介绍得很深, 但您应该有能力编写出这样的 ASP.NET 页面。关于 VB.NET 的更详细的信息, 请查看以下资料:

- MSDN 开发人员参考 ([msdn.Microsoft.com](http://msdn.Microsoft.com));
- .NET 框架在线参考 ([www.Microsoft.com/net](http://www.Microsoft.com/net));
- .NET SDK 文档中的 VB.NET 语言规范和参考。

### 3.11 这不是 ASP

ASP 和 ASP.NET 之间的最大区别在于从 VBScript 到 VB.NET 的迁移。虽然大部分语法是相似的, 但一些语法上的变化可能会给 ASP.NET 新手带来麻烦。

VB.NET 本身便有很大的变化。它现在是完全面向对象的,对于 VBScript 开发人员来说,这可能需要一段时间来习惯。.NET 框架广泛地使用类,这在传统 ASP 中很少见。尽早学习如何使用它们是值得的。

从 ASP 到 ASP.NET 的另一个较大的变化是,后者使用的是事件驱动模型。虽然这种模型的底层机制依赖于请求/响应机制,但添加了一个新的抽象层,让开发人员可以将更多的时间放在创建应用程序,而不是发送的数据上。第 5 章和第 6 章将更详细地介绍这一点。

附录 B “常犯的 ASP.NET 错误”列出了 ASP 开发人员迁移到 ASP.NET 时常犯的一些错误。

## 3.12 总 结

本章介绍了使用 VB.NET 创建 ASP.NET 页面,学习本章后,您已牢固地掌握了编程基础知识,让您能够创建更为复杂的页面。

本章首先介绍了 VB.NET 中的变量。变量是被命名的内存单元,您可以使用其名称操纵它。对于 VB.NET 变量,有 10 种基本的数据类型,它们被划分为 5 类:整数、浮点数、布尔型、字符串和日期时间。

数组是变量可以使用索引引用的变量的集合,对将类似的信息存储在同一个地方很有帮助。在 VB.NET 中,所有数组的索引都是从 0 开始的,不能创建索引不从 0 开始的数组。

本章介绍了三种逻辑:条件逻辑、循环逻辑和分支逻辑。条件逻辑使用 if 语句和 case 语句对条件进行判断;循环逻辑使用 While、Do 和 for 循环来多次执行代码块;分支逻辑使用子程序和函数,它们都执行某种操作,但函数可以返回值。

本章还介绍了事件和事件处理程序,前者是在应用程序中可能发生的情况,而后者则是事件发生时执行某种操作的方法。它们与子程序很相似,不同之处在于其参数列表。

最后本章介绍了类和继承。类是对象的定义,它们将通用的代码放在一起,以使用属性和方法表示一个实体。类是 .NET 框架和 ASP.NET 的重要组成部分。继承让您能够扩展已有的类,以满足您的需要,并覆盖那些不能满足需要的方法。

本章介绍了 ASP.NET 编程中最难的知识,一旦您能轻松地创建方法、事件处理程序和类,您便有能力创建任何 ASP.NET 页面。

下一章将介绍如何使用 C#(一种设计用来方便开发企业级 Web-enabled 应用程序的语言)。C# 是从 C 和 C++ 派生而来的,在很多方面与 Java 类似,如果您使用过这些语言编程,则肯定不会对其中的大部分语法感到陌生。本章还将介绍一些 ASP.NET 中最常用的对象。

## 3.13 问与答

问: VB.NET 区分大小写吗?

答: 不。例如,类名 MyClass 和 myclass 是等价的。

问: variant 类型现在情况如何?

答: 如果您熟悉以前的 VB 版本,肯定还记得,variant 类型是一种通用数据类型,用于表示任何没有声明为特定类型的变量。在 VB.NET 中,variant 已经被 object 类型所取代。

## 3.14 作业

下面的作业帮助您巩固本章介绍的概念，答案见附录 A。

### 3.14.1 小测验

1. 何时应使用 for 循环，何时应使用 while 循环？
2. 下述代码段的输出是什么？

```
Dim I as Integer = 0
Do
    Response.Write(I & " ")
    I = I + 2
Loop Until I > 10
```

3. 下述代码段的输出是什么？

```
Dim I as Integer = 5
Do
    I = I + 2
    Response.Write(I & " ")
Loop Until I > 10
```

4. 对于 ASP.NET 页面，标准事件处理程序参数是什么？

### 3.14.2 练习

使用代表您的类创建一个 ASP.NET 页面，该类包含描述您的头发颜色、眼睛颜色和生日的属性。使用数据类型 `datetime` 的 `dayofweek` 属性来确定您的生日是星期几。`dayofweek` 属性返回一个整数——将其转换为对应的星期几。每当用户单击 Submit 按钮时，便调用该方法，将输出打印到浏览器中。



## 第4章 在C#和VB.NET中使用ASP.NET对象

上一章介绍了如何使用 VB.NET 编写 ASP.NET 页面。您学习了各种控制页面流程的结构，包括循环、条件语句、函数和事件。使用这些结构可以创建出功能强大的 ASP.NET 页面。

然而，如果 ASP.NET 的功能仅此而已，则它将不会是一种好的 Web 开发工具。作为 .NET 框架的一部分，ASP.NET 能够利用成百上千的内置类和对象，扩展页面的功能。本章将介绍一些最常用的对象，本书中将不断使用它们。

您将学习 C#（读作“see-sharp”），该语言是由微软公司推出的，它与 VB.NET 类似，着重于创建 Web 服务应用程序。C# 在 ASP.NET 开发中很流行，因此您必须了解。

本章的大部分例子都将使用 C# 和 VB.NET 编写。这两种语言的许多编程概念很相似，因此，在阅读了前一章内容后，您只需了解 C# 的语法。但在本章之后，我们将把重点放在使用 VB.NET 编写页面上。

本章介绍以下内容：

- 如何在 ASP.NET 页面中使用 C#；
- 一些对象及其属性；
- 如何使用 ASP.NET 中最常用的对象；
- 如何使用会话和 cookies。

### 4.1 C#简介

C# 是微软公司推出的一种全新的编程语言，得到了 .NET 框架的全面支持。由于 C# 是从 C 和 C++ 派生而来的，因此熟悉这些语言的编程人员不用为编写 ASP.NET 页面而麻烦地学些 VB.NET。

C# 也与 Java 编程语言很相似，因此即使是 Java 开发人员也可以很轻松地转移到这种新语言上来。

本章介绍的很多概念与前一章介绍的 VB.NET 很相似，VB.NET 和 C# 都是强大的 ASP.NET 开发语言，因此您可以使用自己最熟悉的语言来开发页面。

#### 4.1.1 C#语法范例

本节粗略地介绍 C# 概念的基本语法，其许多底层概念和 VB.NET 相同，因此您只需了解编写代码时的差别即可。

## 1. 变量和语句

C#的变量类型基本上和VB.NET相同，但声明方式稍微有些差别。在VB.NET中，要声明一个整型变量，可以使用下面的代码：

```
Dim MyInt as Integer
```

在C#中，则使用类似下面的代码：

```
int MyInt;
```

这里有三个不同的地方。首先，在C#中，数据类型在变量名称的前面，Int表示integer；其次，没有使用关键字dim和as；最后，所有的C#语句都必须以分号结束。

让我们来看一些其他的例子：

```
public int Hour; //create an integer variable named Hour
bool binGo = true; //create a Boolean value named binGo
                    //initialized to true
string strWord = "Hello"; //create a string variable named
                           //strWord initialized to "Hello"
```

由于每条语句都必须以分号结束，因此语句可以跨越多行，而不必像VB.NET那样使用连行符。例如：

```
string
    strWord =
        "Hello";
```

一定不要忘了分号！

表4.1列出了C#中可用的数据类型。

表4.1 C#中的数据类型

类 别	类 型
符号整型	sbyte, short, int, long
无符号整型	byte, ushort, uint, ulong
浮点型	float, double
布尔值	bool
16位Unicode字符	char
28个有效位的精确小数	decimal
字符串	string

## 2. 控制结构

控制结构（如if语句、循环、类和函数）的语法都与VB.NET有些不同，但存在一些通用的规则。首先，这些结构以花括号{...}开始和结束。例如，在VB.NET中，您可以使用下面的代码：

```
if x then code
end if
```

而在 C# 中，则使用下面的代码：

```
if (x) {
    code
}
```

在 C# 中，使用下面的方式声明函数：

```
void MyFunction(){
    code
}
```

其中使用花括号，而不是关键字 `function` 和 `end function`。关键字 `void` 表示该函数不返回任何值，这与 VB.NET 中的子程序类似。要返回一个整数，可以使用下面的代码：

```
int MyFunction(){
    code
}
```

而 `if` 语句则与下面类似：

```
if (strWord == "Hello"){
    return true;
}
else {
    return false;
}
```

在 C# 中，通常使用花括号代替 VB.NET 中的关键字。例如，在第一行，使用 `{` 而不是关键字 `then`，而在最后一行使用 `}` 而不是 `end if`。如果要执行的代码位于一行中，则不需要花括号。下述代码与前一个 `if` 语句等价：

```
if (strWord == "Hello")
    return true;
else
    return false;
```

请注意第 1 行的恒等操作符“`==`”，该操作符与在 VB.NET 中的情况不同，经常会让初学者混淆。在 VB.NET 中，可以使用 `=` 来进行比较和赋值，但 C# 将这两个操作符区分开来，`=` 用于赋值，而 `==` 用于比较。

For 循环与下面类似：

```
for (int i = 1; i <= 10; i++){
    Response.Write(i);
}
```

该循环使用 `int` 计数器 `i`，它从 1 开始，当 `i` 不大于 10 时，将一直循环下去，并使用语法 `i++`，每循环一次，将 `i` 的值加 1。同样使用的是花括号和分号。

### 3. 操作符

表 4.2 按优先级次序列出了 C# 中使用的操作符。其中的大部分操作符与 VB.NET 相同，但也有几个新的。

表 4.2 按优先级序列出的 C#操作符

类 型	操 作 符
主要操作符	(x),x.y,f(x),a[x],x++,x--,new,typeof,sizeof,checked,unchecked
单目操作符	+, -,!, ~, ++X, --X, (T)X
乘除	*, /, %
加減	+, -
移位	<(<), >(>)
关系	<, >, <=, >=, is
相等	==, !=
逻辑 AND	&
逻辑 XOR	^
逻辑 OR	
条件 AND	&&
条件 OR	
条件	?:
赋值	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=,  =

#### 4. 强制类型转换

和 VB.NET 一样, C#也允许将变量从一种类型强制转换为另一种数据类型, 只是语法上有些不同。在 C#中, 只需将目标类型放在变量的前面, 并使用括号括起即可。例如:

```
double j = 6.7;
int i = (int)j;
Response.Write(int.ToString(i));
```

第1行声明了一个值为6.7的 double 变量, 第二行将j强制转换为 int 型。现在, 它的值为6(自动向下舍入)。

在强制转换变量数据类型方面, C#比 VB.NET 要严格得多。例如, C#不允许将 int 变量强制转换为 string, 而在 VB.NET 中是可以的。这也是在上述代码段的最后一行使用 int 对象的 ToString 方法的原因所在。它将 int 转换为字符串, 因为 Response.Write 只接受字符串参数。

C#和 VB.NET 之间的区别主要在于语法方面。了解这些区别后, 编写代码时将容易得多。阅读本章后, 您将能够区分 C#代码和 VB.NET 代码。

## 4.2 对象概述

正如前一章讨论过的, 对象是可重用的代码片段, 而类是对象的定义。上一章您创建了一个名为 clock 的简单对象。

.NET 框架由许多定义对象的类组成，ASP.NET 可以随意使用它们。当然，有些对象可以永远不会在真实的 ASP.NET 页面中用到，因为它们不是为 ASP.NET 设计的，但您确实可以使用它们。让我们在上一章的基础上再次简要地介绍一下对象。

#### 4.2.1 属性

属性是描述对象的变量，清单 4.1 列出了前一章中关于钟表的例子，但这次是使用 C# 编写的。

**清单 4.1 使用 C# 编写的 clock 类**

```
1 <script language="C#" runat="server">
2     public class Clock {
3         public int Second;
4         public int Minute;
5         public int Hour;
6         public static int ClockCounter = 0;
7
8         public void SetTime(int intSec, int intMin,
9             int intHour)
10        {
11            Second = intSec;
12            Minute = intMin;
13            Hour = intHour;
14        }
15    }
16
17 </script>
```

分析：第 3-5 行为类指定了三个属性：Second、Minute 和 Hour。如果要创建的是 car 类，则属性可能是 color、make 和 model。

同样，语法上也有些不同。首先，第一行将 language 属性指定为 C#，同时您使用的是本章前面的“C#语法范例”一节中讨论过的 C# 语法。注意，第 8 行没有连行符，而在 VB.NET 中，这是不可少的。

#### 4.2.2 方法

方法是可以用对象执行的操作，如给钟表上发条或设置时间。在清单 4.1 中，第 8 行的方法 SetTime 设置了对象的属性。对于小汽车，其方法可能包括 Accelerate 和 Stop 等。

#### 4.2.3 对象实例

**新术语：**您必须创建一个对象实例，才能使用该对象。也就是说，您必须创建一个在内存中保存该对象的变量。理解实际对象和对象实例之间的差别至关重要。

以钟表为例，您知道钟表是一个通用对象。而我的 clock 对象是一个实例，可以设置其时间。您的 clock 对象则是另一个实例，有自己的一组属性。图 4.1 显示了两个钟表实例，其属性不同。

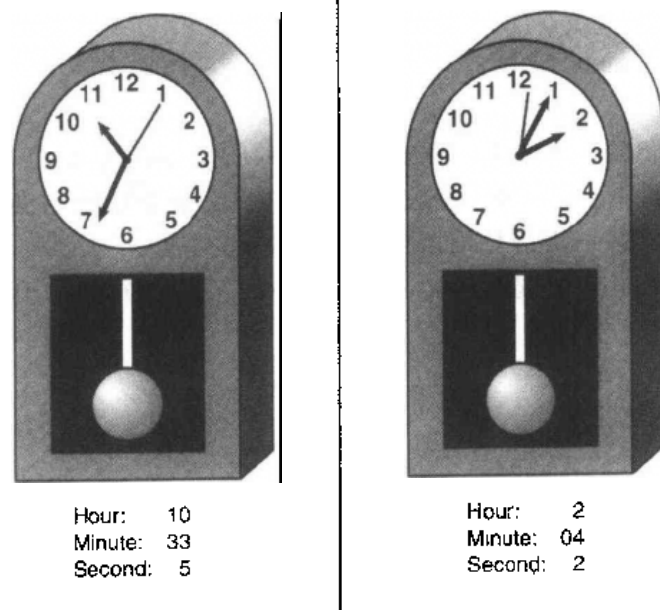


图4.1 两个属性不同的钟表对象的实例

设置通用钟表对象的分针是毫无意义的。这会影响所有的钟表吗？设置的是哪个钟表？通常只能设置对象实例的属性和调用其方法。让我们来看一个C#的例子：

```
clock.Second = 41; //would produce an error because we can't
                  //modify the properties of an object

Clock objClock = new Clock();
objClock.Second = 41; //works because objClock is an instance
```

第二种命令是正确的设置属性的方式。首先创建一个对象实例，然后设置其属性。相应的VB.NET代码如下：

```
Dim objClock as New Clock
objClock.Second = 41
```

#### 4.2.4 静态成员

注意，我前面是这样说的：“通常，只能设置实例的属性”，有时候也可以直接从对象中设置属性或调用方法。例如，假设clock类中有一个计数器，用于指出创建的clock实例数目——这是一个通用属性，不属于任何一个特定的实例——它一直都存在，类中不属于任何特定实例的属性和方法被称为静态成员，是使用关键字static来声明的。

```
public static int ClockCounter = 0;
public static int AddClock()
{
    ClockCounter += 1;
    return ClockCounter;
}
```

在ASP.NET中，可以以下述方式引用它们：

```
<%
    Clock objClock = new Clock();
```

```
objClock.SetTime(4,6,7);

Clock.AddClock();

Response.Write(Clock.ClockCounter);

%>
```

在上述代码段的倒数第三行调用了静态方法 `AddClock`，它将钟表总数加 1。该方法不用在实例中进行调用。接下来的一行使用了两个静态成员。`ToString` 是 `int` 对象的一个静态方法，它将 `int` 转换为字符串；而 `ClockCounter` 是静态属性，告诉您有多少座钟表。

您不会经常在自己的类中创建静态成员，但知道它的情况将很有帮助，因为 ASP.NET 中有大量的静态成员。

## 4.3 ASP.NET 对象

本节并非要介绍 ASP.NET 中的所有对象——那可能需要一部百科全书，而只是介绍一些最常用的对象，您将经常在 ASP.NET 页面中使用它们。您应该对下面要介绍的第一个对象（`Response`）比较熟悉了。

**注意：**下面要介绍的所有对象都不是 ASP.NET 固有的，也就是说它们都属于 .NET 框架（而不是 ASP.NET），该框架下的任何应用程序都可以使用它们。

### 4.3.1 Response 对象

`Response` 对象让服务器能够与客户通信，例如，您可以使用 `Response` 对象的 `Write` 方法将 HTTP 输出发送给浏览器。您已经见过该方法的例子：

```
Response.Write("Hello World!");
```

使用该对象时，情况到底是如何的呢？用户请求页面时，ASP.NET 创建 `HttpResponse` 对象的一个实例，其中包含与客户通信所需的信息（属性和方法）。该实例名为 `Response`，因此可以使用该名称访问 `HttpResponse` 对象的属性和方法。您可能已经猜到了，其中的一个方法便是 `Write`，它将一个字符串写到浏览器中。

#### 1. Write 方法

让我们来创建一个 ASP.NET 页面，更详细地探讨 `Write` 方法，如清单 4.2 所示。

**清单 4.2 使用 Response.Write**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4
5     sub Page_Load(obj as object, e as EventArgs)
6
7         dim i as integer
8
9         Response.Write('This is an example')
10        Response.Write('<HR width=100%>')
11
12        for i = 1 to 5
13
14            Response.Write('<font size=" & i & ">Hi!<br></font>')

```

```

12     next
13 end sub
14 </script>
15
16 <html><body>
17
18 </body></html>

```

**分析：**Write 方法接受一个字符串参数，因此需要将输入括在双引号中。更详细的细节请看下面的提醒，让我们仔细地看看第8行：

```
Response.Write("<HR width=100%>")
```

**新术语：**特别要注意行尾的反斜杠，它好像有点不得其所。没有该字符，该行将包括字符串 %>，它告诉 ASP.NET：脚本块到此结束。因此程序将在字符串的中间停止执行，从而引发错误 %> 是一个特殊的字符序列，因此必须以某种方式将其分开，以免 ASP.NET 搞不清楚。因此，使用了转义字符 (\)

HTML 输出中不会出现反斜杠。如果要输出双引号，则必须使用另一对引号将其转义：

```
Response.Write("<HR width=""30%"">")
```

该清单的输出如图 4.2 所示。

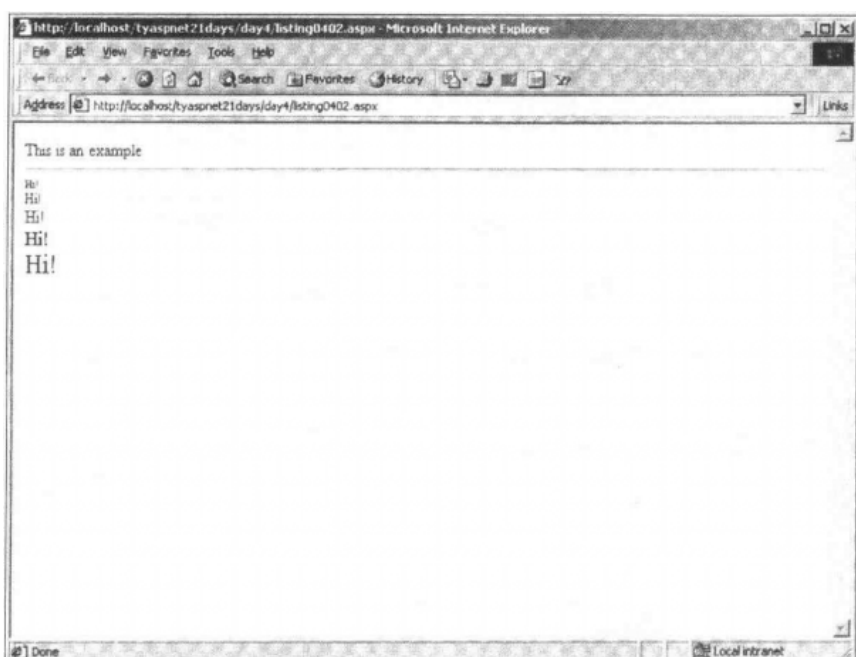


图 4.2 使用 Response.Write 将输出写入浏览器中

**警告：**如果可能，VB.NET 将自动执行一些数据类型强制转换。其中的一个例子是将 integer 强制转换为 string，这也是可以在 VB.NET 编写的 ASP.NET 页面中使用 Response.Write(6) 的原因所在

但 C# 不自动执行这类强制转换，因此使用 C# 编写 ASP.NET 页面时，使用 Response.Write(6) 将引发错误。在 C# 中，应该使用下面的代码：

```
Response.Write(int.ToString(6))
```



请务必以合适的方式对变量进行转换。

## 2. 缓冲页面

缓冲技术让您能够控制何时将输出发送到浏览器中。当您浏览网页时,可能遇到过这样的情况,只是没有意识到而已。

当输出被缓冲时,在所有的代码都执行完之前,不会向浏览器发送任何信息,这是默认情况下 ASP.NET 采用的缓冲方式。如果不缓冲,则输出将立刻被发送到浏览器。

例如,假设页面中使用了两次 `Response.Write` 方法。如果缓冲,则输出将被存储在内存中,当执行完所有的 ASP.NET 代码后,这些输出将被一次性发送给浏览器。如果不缓冲,则每个方法的输出都将立刻被发送给浏览器。图 4.3 说明了这个概念。

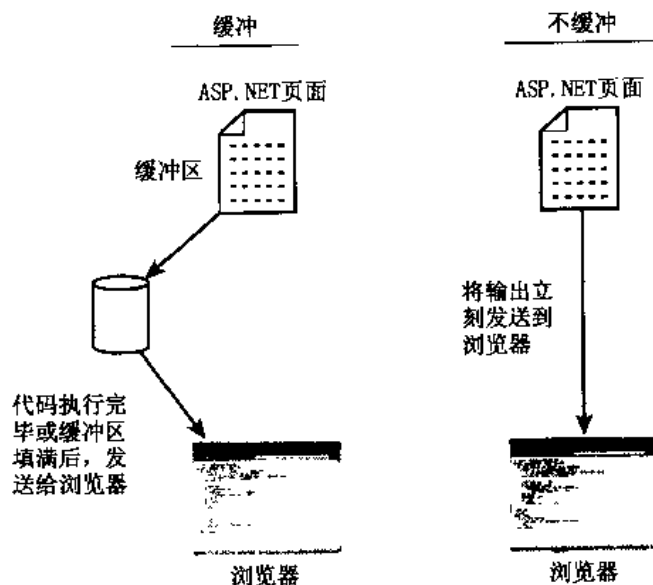


图 4.3 缓冲和不缓冲输出之间的差别

缓冲技术可以极大地提高性能,因此通常应该启用。要关闭缓冲,只需在给浏览器发送输出之前,使用 `BufferOutput = false` 即可。

下述代码将引发错误,因为您已经将输出发送给浏览器了。

```
<html><body>
  <% Response.Buffer = False %>
</body></html>
```

相反,应该使用下面的代码:

```
<% Response.Buffer = False %>
<html><body>
  blah blah
</body></html>
```

您可以使用 `Clear`、`Flush` 和 `End` 方法操纵缓冲区。`Clear` 将缓冲区清空,因此将丢失存储的所有信息; `Flush` 立刻将缓冲区中的所有信息发送给浏览器; `End` 阻止 `Response` 对象将新的输出发送给浏览器,而缓冲区中的所有信息将被发送给浏览器。清单 4.3 给出了一个例子。

## 清单 4.3 测试输出缓冲技术

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as EventArgs)
5         dim i as integer
6     {
7         Response.Write("Before flush<br>")
8         Response.Flush()
9         for i = 0 to 5000
10             'just wasting time
11         next
12
13         Response.Write("After flush, before clear<br>")
14         Response.Clear()
15         for i = 0 to 5000
16             'just wasting time
17         next
18
19         Response.Write("After clear, before end<br>")
20         Response.End
21         for i = 0 to 5000
22             'just wasting time
23         next
24
25         Response.Write("After end<br>")
26     end sub
27 </script>
28
29 <html><body>
30
31 </body></html>
```

**分析：**默认情况下，输出将被缓冲，不用显式地指定。第7行将“Before flush”存储到缓冲区中，而第8行则将其立刻发送给浏览器。您使用 for 循环来等待一段时间，以便能够更明显地看到缓冲的效果。第13行将另一个字符串存储到缓冲区中，然后立刻将其清除。第19行又存储了一个字符串，并在第20行调用 End。结果，缓冲区的内容被发送给浏览器，而以后的输出则没有被发送给浏览器。因此，在浏览器中看不到“After clear, before end”以后的内容。

上述代码的输出如图4.4所示。

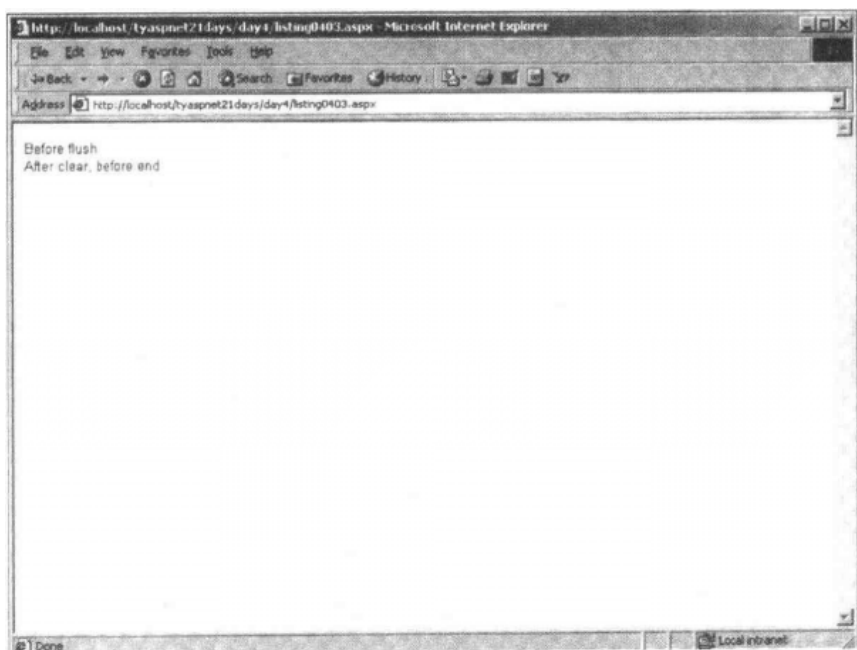


图 4.4 操纵 Response 缓冲区

### 3. 重定向用户

Response 对象还让您能够使用 Redirect 方法, 将其他页面发送给用户, 而用户却对此一无所知。例如:

```
<%@ Page Language="VB"%>
<script runat="server">
    sub Page_Load(obj as Object,e as EventArgs)
        Response.Redirect("http://www.microsoft.com")
    end sub
</script>
<html><body>
    Hello!
</body></html>
```

当用户请求该页面时, 将立刻被重定向到 [www.microsoft.com](http://www.microsoft.com), 而看不到您提供的问候消息。例如, 如果想根据用户在表单中输入的用户名和密码, 将其重定向到某个地方, 则该方法很有用。

### 4.3.2 Request 对象

与 Response 对象相反, Request 对象允许浏览器与服务器进行通信。请求页面时, 浏览器给服务器发送大量的信息。每当出现这种情况时, 将创建一个 HttpRequest 对象, 来处理所有这些信息, 该对象被称为 Request。因此, Request 将代表客户请求网页, 而 Web 服务器则使用 Request 和 Response 对象将网页发送回去。

对于 ASP.NET 开发而言, 该对象的功能并非必不可少, 因为 ASP.NET 将替您处理其大部分功能, 但还是有一些东西您必须知道。

#### 发现客户信息

**新术语:** 该对象的一个主要用途是从浏览器那里收集客户信息, 如用户在表单中的输入或查询

字符串 (querystring) 值。查询字符串是 URL 之后的信息, 例如:

```
http://www.microsoft.com?id=chris&sex=male
```

查询字符串 “?id=Chris&sex=male” 以关键字/值对的方式表示数据。id 是第一个关键字, 而 chris 是第一个值; sex 是第二个关键字, 而 male 是第二个值。第一个关键字/值对之前总是有一个问号, 而关键字/值对之间以 “和” 符号分开。

查询字符串用于在页面之间传递信息, 例如:

```
Response.Redirect("MyPage.aspx?ID=Chris")
```

这样, MyPage.aspx 在需要时可以使用该查询字符串。

**警告:** 不要在查询字符串中存储过多的信息。老式浏览器要求这种字符串不能超过255个字符, 而您无法知道访问者是否会使用老式浏览器。另外, 将大量的文本放在这种地方, 可能引发老式 IIS 版本中的 bug。

ASP.NET 提供了一个抽象层, 因此无需使用 Request 对象, 便可以收集信息 (在第 5 章您将知道这一点), 但仍然提供了该对象, 让您使用。以前面的 URL 为例, 可以使用下列代码来检索查询字符串的值:

```
Request.QueryString["id"] returns "id=Chris&sex=male"
```

```
Request.QueryString["id"] returns "Chris"
```

如果用户提交了表单, 则可以使用下面的代码:

```
Request.Form["name"] //returns all form values
```

```
Request.Form[name] //returns a single value specified by name
```

属性 QueryString 和 Form 都是信息集, 这些信息通常来自用户的输入, ASP.NET 则使用 Request 对象进行检索。本书中使用该对象的情况不多, 因为许多请求是由 Web 表单框架为您处理的。

和 Request 一起使用的另两个常用的集合是 ServerVariable 和 Cookies。前者返回关于服务器的各种信息, 如 IP 地址或 HTTP 协议; 后者返回关于 cookie 的信息, cookie 是客户机上的小型文件 (请参考本章后面的 “HttpCookie 对象”)

表 4.3 列出了 ServerVariable 集合中一些常用的环境变量。

表 4.3 常用的环境变量

变 量	描 述
URL	ASP.NET 页面的 URL, 位于服务器和域名的后面 (也就是 http://www.server.com/ 的后面)
PATH_INFO	和 URL 相同
PATH_TRANSLATED	ASP.NET 在服务器上的完整物理路径
SERVER_NAME	Web 服务器的名称
SERVER_SOFTWARE	Web 服务器软件名称, 如 Microsoft-IIS/5.0

#### 4.3.3 HttpCookie 对象

**新术语:** cookie 是用户计算机上的一个小文件, 其中包括关于某个 Web 站点特定的信息。该文件包括诸如用户名和密码等用于定制用户访问站点的信息。Cookie 可以包含任何简单数据类型,

如 string、integer、float、boolean 等。例如，许多显示新闻摘要的站点允许用户选择显示的新闻类型。这些信息可以保存在 cookie 中，以便用户下次访问时，站点可以读取这些值，并相应地进行定制。

HttpCookie 对象提供了存取和创建 cookie 的方法。您可以使用该对象来检查 cookie 的属性。但对于操纵 cookie 而言，最常用的方法是通过 Request 和 Response 对象，这两个对象都有 cookie 属性，返回 HttpCookie 对象的引用。

### 1. 创建 cookie

Response 对象让您能够很容易地创建 cookie。创建 cookie 的方法有两种：创建多个只有一个值的 cookie 或创建一个包含多个关键字/值对的 cookie。下面的代码片段演示了这两种方法：

```
'set up some cookie variables
Response.Cookies("MyCookie").Value = "Single Cookie"
Response.Cookies("21DaysCookie")("Username") = "Chris"
Response.Cookies("21DaysCookie")("Preference") = "800x640"
```

**分析：**第二行在客户机上创建一个名为 MyCookie 的 cookie，其值为 Single Cookie。使用这种方法可以创建任意数量的 cookie 和关键字/值对，但可能导致 cookie 数量非常多。第二组代码创建一个名为 21DaysCookie 的 cookie，其中包含两个关键字/值对。要指定关键字名称，只需在括号（在 C# 中为方括号）中加入另一个名称即可，如下所示：

```
Response.Cookies[CookieName][KeyName];
```

在第二行，必须使用 Value 属性将一个字符串赋给 cookie。如果使用 Response.Cookies("MyCookie") 来返回一个 HttpCookie，则无法将字符串写入到该对象中。属性 Value 返回一个您可以修改的字符串对象。但在第二组代码中，不用指定 Value，因为如果您指定一个关键字名称，ASP.NET 将知道您需要一个字符串。

**警告：**如果使用值创建 cookie，然后再添加关键字，则原来的值将被清除。不能同时使用关键字和值来创建 cookie。

假设某访问者来到您的站点，而您为他创建了一个 cookie。如果该访问者不打算以后再访问您的站点，则没有理由再保留该 cookie。当然，他可以手工删除该 cookie，但有一种更简单的方式。可以使用属性 Expires 设置 cookie 作废的时间，例如：

```
Response.Cookies("21DaysCookie").Expires = _
    DateTime.FromString("1/1/2002")
```

或：

```
Response.Cookies("21DaysCookie").Expires = _
    DateTime.Now.AddMonths(1)
```

前一个例子将 cookie 的作废时间设置为 2002 年 1 月 1 日。第二个例子将 cookie 的作废时间设置为代码执行后一个月。默认的 Expires 值将 cookie 的作废时间设置为 1000 分钟以后。如果只是想维护当前会话（参见后面的“Session 对象”）的信息，则足够了。但 cookie 通常用于将信息保存更长的时间——几周、几个月，甚至几年。

要删除客户端的 cookie，必须将 cookie 的 Expires 值设置为以前的时间或 0。这样一旦用户关闭其浏览器，该 cookie 便会被删除。

您还应该了解 HttpCookie 对象的另外四个属性。Domain 限制 cookie 只能在指定的域中使用，如 www.myserver.com，这给您提供了更大的对 cookie 存取控制的能力，但通常使用默认值即可。Path 与 Domain 类似，但它将对 cookie 的存取限制为服务器上特定路径中的 ASP.NET 页面。HasKeys 指

出 cookie 有关键字, 还是只是一个单值 cookie。最后 Secure 告诉 ASP.NET: 是否应以安全的方式(也就是说, 只通过 HTTPS 协议)传输 cookie, 其默认值为 false。

## 2. Cookie 的存取

发出请求时, 浏览器将所有的 cookie 信息发送给服务器。因此, 可以使用 Request 对象来收集这些信息。存取 cookie 的语法与创建 cookie 完全相同, 在下述清单中, 使用 Response.Write 将 cookie 的值写到浏览器中:

```
'set up some cookie variables
Response.Write( _
    Request.Cookies("MyCookie").Value)
Response.Write( _
    Request.Cookies("21DaysCookie")("Username"))
Response.Write( _
    Request.Cookies("21DaysCookie")("Preference"))
```

请注意存取值和存取关键字/值对之间的差别, 特别是 Value 属性的用法。由于关键字/值对只是数据集合而已, 因此可以很容易地遍历它们, 如清单 4.4 所示:

**清单 4.4 遍历 cookie 中的关键字**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as EventArgs)
5         dim strVariable as string
6
7         'set up some cookie variables
8         Response.Cookies("temp").Value = "HI"
9         Response.Cookies("21DaysCookie")("Username") = "Chris"
10        Response.Cookies("21DaysCookie")("Preference") = _
11            '800x640'
12        Response.Cookies("21DaysCookie")("Password") = _
13            'CookiesRock'
14        Response.Cookies("21DaysCookie")("LastVisit") = _
15            DateTime.Now.ToString
16        Response.Cookies("21DaysCookie")("UserAgent") = _
17            Request.ServerVariables("HTTP_USER_AGENT")
18
19        for each strVariable in Response.Cookies("21DaysCookie").Values
20            Label1.Text += "<b>" & strVariable & "</b>: " & _
21                Request.Cookies("21DaysCookie")(strVariable) & "<br>"
22        next
23    end sub
```

```

24 </script>
25
26 <html><body>
27   <form runat="server">
28     <asp:Label id="Label1" runat="server"/>
29   </form>
30 </body></html>

```

分析: 第 28 行创建了一个 ASP.NET Label 服务器控件 (将在第 5 章讨论), 现在您只需知道该控件将 HTML 文本显示为 `<span>` 元素即可。

注意, 其中大部分代码都位于事件处理程序 `Page_Load` (将在下一节讨论) 中。第 8-17 行创建了一个 cookie, 并放置了一些关键字和值。第 17 行使用 `Request.ServerVariables` 集合来返回 `HTTP_USER_AGENT` 的设置。

第 19 行返回一个由 cookie 的关键字组成的集合, 使用 `for...each` 循环, 将关键字名称及其值显示在标签中。`for...each` 循环将从集合中返回的值 (这里是关键字名称) 赋给计数器变量 `strVariable`。您可以使用下面的代码来存取关键字的值:

```
Request.Cookies("21DaysCookie")(strVariable)
```

上述清单的输出与图 4.5 类似。

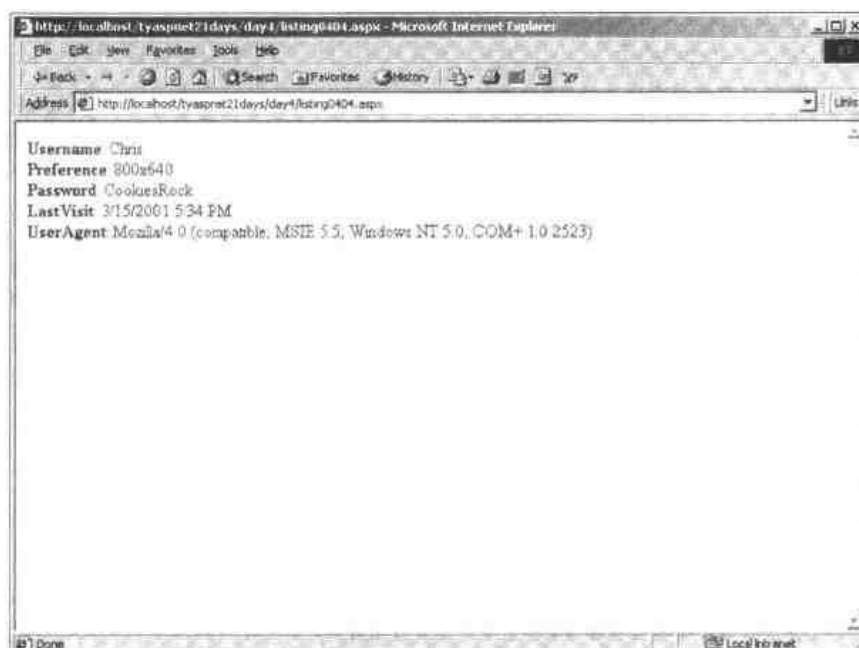


图 4.5 遍历 cookie 集合

cookie 是非常棒的维护用户信息的工具, 其唯一的缺点是, 不是所有的浏览器都支持 cookie。在本章后面的“Session 对象”一节中, 将介绍另一种维护信息的方法。

#### 4.3.4 Page 对象

`Page` 对象包含用于所有 ASP.NET 页面的方法和属性, 执行 ASP.NET 页面时, 它是从 .NET 框架中的 `Page` 类派生而来的。

还记得前一章关于继承的讨论吗? 类定义了基于该类的对象的参数和方法, 当您以继承的方式创建类时, 它也继承了父类的成员, 同时可以为子类创建成员。

ASP.NET 页面是 `Page` 对象的子对象, 页面中的成员就是从 `Page` 那里继承而来的。您在 ASP.NET 页面中定义的属性和方法, 都将成为基于该页面的对象中的成员。这意味着如果您创建另一个页面, 它将能够访问第一个页面的方法和属性! 这都是面向对象编程的功劳。

该对象只有几个有用的内置成员。 `IsPostBack` 指出页面上的表单是否回送给同一个页面 (关于回送表单的讨论, 请参考第2章)。在第5章和第6章学习过 Web 表单后, 您将经常使用该属性。

`Databind` 将所有的数据表达式绑定到页面的控件上——我们有些超前了, 这将在第9章进行讨论。

最后, 该对象还有一个您将经常使用的事件 `Load`, 每当页面开始被装载到浏览器中 (对 ASP.NET 开发人员来说, 这是一个非常好地时机) 时, 该事件都将被激活。清单 4.5 给出了一个例子。

**清单 4.5 使用 `Page_Load` 事件**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server" >
4     sub Page_Load(obj as object, e as EventArgs)
5         tbMyText.Value = "This is the page load event!"
6     end sub
7 </script>
8 <html><body>
9     <form runat="server">
10         <input type="text" size="25"
11             id="tbMyText" runat="server" />
12     </form>
13 </body></html>
```

**分析:** 第10行创建了一个简单的 HTML 表单文本输入框, 名为 `tbMyText`, 并指定它将在服务器上运行。第4行为事件 `Load` 定义了一个事件处理程序 `Page_Load` (请注意标准的事件处理程序参数列表)。当激活该事件时 (即当该页面被装载到浏览器中时), 将 `tbMyText` 的 `Value` 属性设置为 "This is the page load event!"。图 4.6 显示了该清单的结果。

事实上, 您将在用户看到表单字段之前填写它。

**注意:** 请注意事件处理程序名称的格式, ASP.NET 要求按约定 `object_event` 给事件处理程序命名, 这里为 `Page_Load`。

这种约定是必须遵守的, 否则 ASP.NET 如何知道发生内置事件时, 执行哪个事件处理程序呢?

对您自定义的事件处理程序, 可以不遵守这种命名约定。该约定只适用于您不能通过编程来访问的事件。



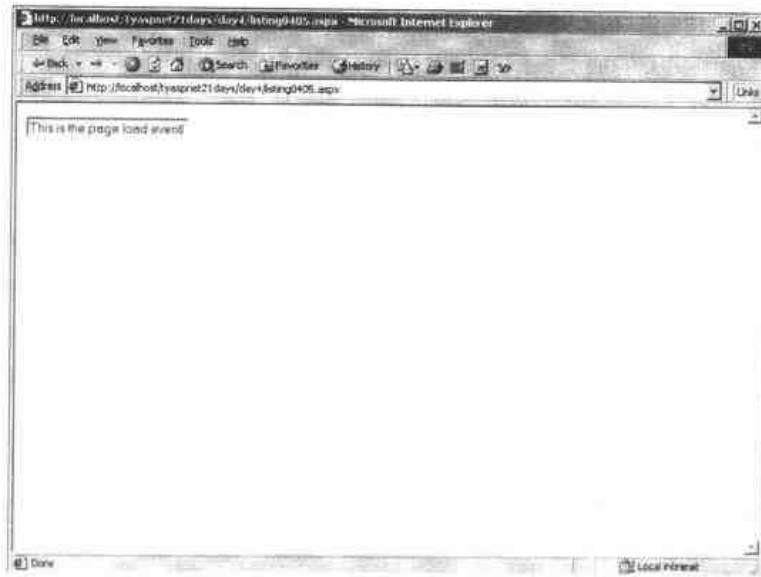


图 4.6 Page\_Load 事件的功效

对 ASP.NET 开发人员来说, Load 事件至关重要。您可以在该事件的处理程序中执行大量的工作,包括验证用户的身份、装载数据库中的数据以及将用户重定向等。事实上,以后的所有 ASP.NET 页面几乎都包含了该事件的处理程序。

让我们来看另一个例子。假设您想根据用户访问页面的时间,显示不同的消息,则可以像清单 4.6 那样实现这种目的。

#### 清单 4.6 使用 Load 事件显示不同的消息

```

1 <%@Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object,e as EventArgs)
5         dim Now as DateTime =DateTime.Now
6         dim intHour as integer =Now.Hour
7
8         Label1.Text = "The time is now  & _
9             Now.ToString("T")+ '<sp>'
10
11         if intHour <12 then
12             Label1.Text += 'Good morning!'
13         elseif intHour >12 & intHour <18 then
14             Label1.Text += "Good afternoon!"
15         else
16             Label1.Text += "Good evening!"
17         end if
18     end sub
19 </script>

```

```

20
21 <html><body>
22   <form runat="server">
23     <asp:Label id="Label1" runat="server"/>
24   </form>
25 </body></html>

```

**分析:** 第 23 行创建了另一个 Label 控件。在 Load 事件处理程序中, 您将变量 now 的值设置为当前的时间, 并使用 DateTime 对象的 Hour 属性返回其中的小时部分。

第 8 行将当前时间显示到 Label 控件中。ToString 方法返回一个字符串, 其中包括格式化后的时间; 该方法接收一个字符串参数, 告诉对象如何格式化日期。表 4.4 列出了有效的格式化字符串。

**表 4.4** 有效的 DateTime 格式字符串

字 符 串	例 子
"d"	2/01/2001
"D"	Thursday, February 01, 2001
"f"	Thursday, February 01, 2001 4:32 PM
"F"	Thursday, February 01, 2001 4:32:30 PM
"g"	2/01/2001 4:32 PM
"G"	2/01/2001 4:32:30 PM
"m"	February 01
"T"	Thu, 01 February 2001 16:32:30 GMT
"s"	2001-01-01T 16:32:30
"t"	4:32 PM
"T"	4:32:30 PM
"u"	2001-01-01 16:32:30Z
"U"	Thursday, February 01, 2001 16:32:30
"y"	February, 2001
"ddd, MMM dd yyyy"	Thursday, February 01, 2001
"ddd, MMM d"yy"	Thu, Feb 01
"ddd, MMM dd"	Thursday, February 01
"M/yy"	2/01
"dd-MM-yy"	01-02-01

从第 11 行开始, 您使用一系列的 if 语句来决定将合适的消息显示到标签中。图 4.7 显示了编写本书时该页面的输出。

几章后，当您学习了 Web 表单后，将使用 load 事件来完成更多有趣的任务。

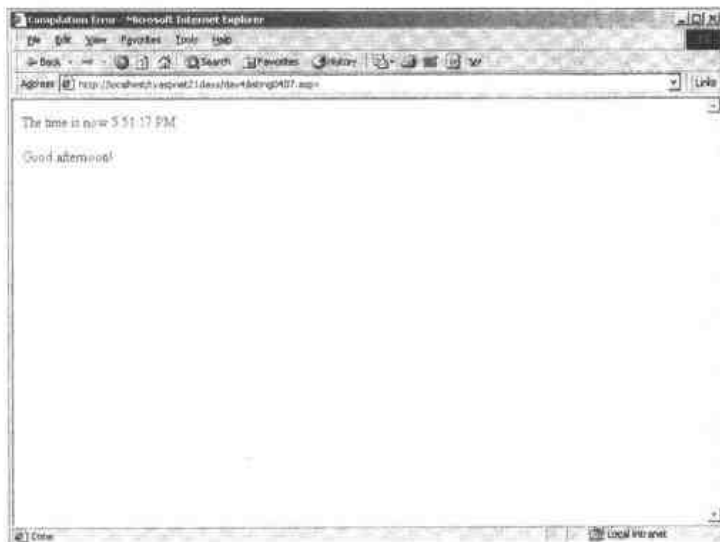


图 4.7 在 Web 事件处理程序中显示定制的消息

#### 4.3.5 Session 对象

Session 对象说明了一个非常有趣的概念。由于 Web 是一种无状态的介质（参考第 2 章），因此很难跟踪关于特定用户的信息。使用 HTTP 无法弄清楚一系列的请求是来自同一个用户还是大量不同的用户，这使得难以为用户定制 Web 站点。

Session 对象可以突破这种局限性，它让您能够将与特定用户相关的元素（如变量、对象、字符串或任何东西）存储在服务器的某个位置，它几乎是用户的个人信息存储箱。想象一下学校的储物箱——上课时，您将自己的东西存储在储物箱中，离校时再把东西取出。

**新术语：**Session 对象的工作机制与此相同。当用户访问站点时，将给他分配一个储物箱，开发人员可以根据需要将相应的信息保存到其中。用户在站点上停留的时间被称为会话（Session）。用户离开站点时，储物箱将作废，其中的信息将丢失，会话也将结束。

假设用户来到您的网站，并在表单中输入其姓名，而您想将其记录下来。您可以将姓名放到 Session 对象中，则只要该会话没有结束，您便可以在任何地方获取该姓名。其语法如下：

```
Session.Add(variablename, value)
```

或：

```
Session(variablename) = value;
```

您可以为每个用户存储任意数目的变量，只要内存够用。清单 4.7 是一个实现上述范例的一段代码。

#### 清单 4.7 在 Session 对象中存储变量

```
1 <%@Page Language="VB" %>
2
3 <script runat="server">
4     sub Submit_Click(obj as object,e as EventArgs)
5         if tbName.Value <> ""
```

```

6      Session("Name")=tbName.Value
7      Response.Write("Hi " & Session("Name")&"!")
8  else
9      Response.Write("You forgot to enter a name.")
10     end if
11 end sub
12 </script>
13
14 <html><body>
15     <form runat="server">
16         Please enter your name:
17         <input type="text" id="tbName"
18             runat="server" />
19
20         <p>
21             <asp:Button id="btSubmit"
22                 Text="Submit"
23                 runat="server"
24                 OnClick="Submit_Click" />
25         </form>
26 </body></html>

```

**分析：**让我们看看代码声明块中的函数。首先，第5行检查用户是否在文本框中输入了值，如果没有，则使用 Response.Write 显示一条错误消息，如第9行所示；如果输入了，则新建一个名叫 name 的 Session 变量，并将文本框中的值赋给它，如第6行所示。最后，第7行使用刚才存储在 Session 对象中的变量给用户显示一条欢迎消息。图4.8显示了该清单的结果。

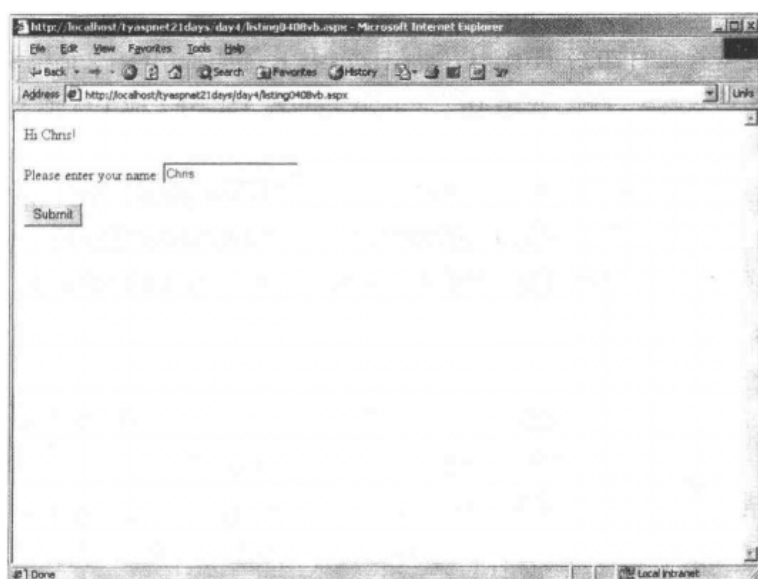


图 4.8 存储和检索 Session 变量

注意,输入的文本将一直保留在文本框中,即使单击 Submit 按钮后,这都是自动视图状态管理的功劳。

Session 对象使用起来非常简单,也很有用。让我们创建另一个页面,它使用了刚才创建的 Session 变量,如清单 4.8 所示。

**清单 4.8 检索 Session 对象中的变量**

```
1 <%@Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object,e as event\args)
5         Label1.Text = "Welcome back " & Session("Name ")&'!'
6     end sub
7 </script>
8
9 <html><body>
10     <form runat="server">
11         <asp:Label id="Label1" runat="server"/>
12     </form>
13 </body></html>
```

假设没有关闭浏览器(或让它持续运行很长时间),则现在浏览器上应该出现一条欢迎消息,它使用了在清单 4.7 中存储的姓名。您甚至可以像对其他任何变量一样,保存存储在 Session 中的数据和对象。

ASP.NET 如何跟踪会话呢?通常用户访问站点时,将开始一个会话,计算机将为用户生成一个唯一的 ID,然后将其作为一个 cookie 存储在客户机中。当用户从一个页面移动到另一个页面时,ASP.NET 将读取该 cookie 值,来确定该用户所属的会话。可以使用下述代码轻松地读取这个值:

```
Response.Write(Session.SessionID)
```

Session ID 是一个 120 位的字符串,以确保对每个用户都是唯一的。会话结束时,该 cookie 将被删除(虽然可以修改这种行为,见第 18 章“配置和部署 ASP.NET 应用程序”)

#### 1. 控制会话

控制会话在 ASP.NET 应用程序中的行为的方法很多。Timeout 值用于设置会话空闲多长时间后,ASP.NET 将删除它。换句话说,如果用户访问您的站点,但不执行任何操作或离开站点的时间等于 Timeout 的值后,则相应的会话及其信息将丢失。在 IIS 2.0 中,默认值为 20 分钟,但您可以修改这个值,如下所示:

```
Session.Timeout = x 'x is the number of minutes
```

想修改这个值的原因很多。让我们来看一个典型的 Web 站点:访问该站点的每个用户都将获得一个唯一的 Session 对象,这意味着每个用户都将在服务器内存中占有自己的空间。即使用户离开该站点 30 秒钟后,其会话仍将继续存活 19 分钟 30 秒。这可能占用大量的内存。

表 4.5 列出了当 timeout 为 120 分钟,每 30 分钟有 100 个访问者时,会话的增长情况

表 4.5 会话的增长情况

时 间	增加的用户数	会 话 数	描 述
0:00	100	100	为开始的 100 个用户创建了 100 个会话
0:30	100	200	最初的 100 个会话仍然存活, 不管用户是否离开站点。同时为新增的 100 个用户创建了 100 个会话
1:00	100	300	新增 100 个用户意味着 100 个会话, 而原来的会话还未过期
1:30	100	400	新增 100 个用户意味着 100 个会话, 而原来的会话还未过期
2:00	100	500	新增 100 个用户意味着 100 个会话, 而原来的会话还未过期
2:30	100	500	新增 100 个用户意味着 100 个会话, 最开始的 100 个会话过期了
3:00	100	500	会话数将一直保持不变

这已经浪费了大量的内存——100 个并行访问者使用了 500 个会话。这肯定将降低整个站点的运行速度。

但将 Timeout 值设置得太短也会带来一些问题。假设有一个电子商务站点, 用户可以将商品加入到购物车中。可以将商品保存到 Session 对象中, 但如果将 Timeout 值设置得太短, 则很可能用户在付账之前, 其购物车中的商品已经全部丢失了。

对于很多情况来说, 20 分钟是比较理想的, 但您可以根据实际情况调整这个值。一个安全的银行站点, 可能将这个值设置得较小, 而在线电子邮件站点则需要设置得较大。

另外, 也可以使用 Abandon 方法使会话立刻过期。假设有一个用户可以查看其电子邮件的站点, 用户查看后, 想立刻注销, 以防他离开计算机时, 其他人使用其账户。将会话作废是实现上述目的的方法之一, 只需调用下面的代码即可:

```
Session.Abandon
```

临时 cookie 和所有会话信息将被删除。

## 2. 使用 Session

可以像操纵数组那样操纵 Session 对象。需要时, 可以遍历并操纵所有的变量。清单 4.9 使用 VB.NET 中的 for... each 语句来遍历并显示会话的内容

### 清单 4.9 遍历会话变量

```
1 <%@Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as Object,e as EventArgs)
5         dim strVariable as string
6
7         'set up some session variables
8         Session("Name")="Chris "
9         Session("FavoriteColor")="Blue "
10        Session("EyeColor")="Brown "
```

```
11 Session('AMessage')="Welcome to my world "
12 Session('ASPNET')="rocks!"
13
14 for each strVariable in Session.Contents
15     Label1.Text += "<b>" & strVariable & "</b>:" & _
16         Session(strVariable) & "<br>"
17 next
18 end sub
19 </script>
20 <html><body>
21     <form runat= server'>
22         <asp:Label id="Label1" runat="server"/>
23     </form>
24 </body></html>
```

**分析：**第8-12行建立了一些 Session 变量，第14行使用 for... each 循环，遍历集合中的所有条目，这里是 Session Content 集合中的所有字符串。该集合中的每个条目代表一个关键字值——一个 Session 变量的名称——如“Name”、“FavoriteColor”等。您使用 Session(strVariable)来检索关键字的实际值，然后将关键字/值对显示在标签控件中，如图4.9所示。

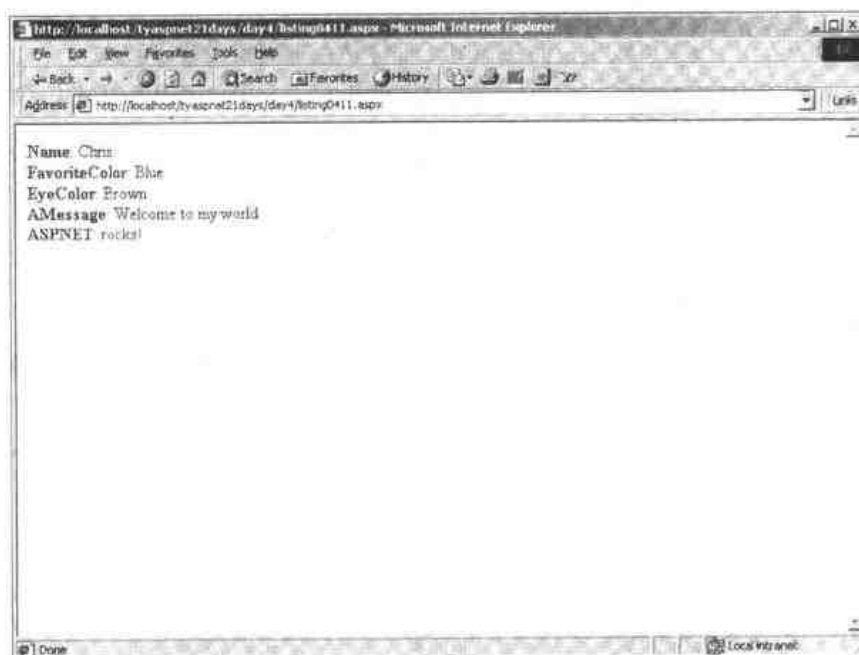


图4.9 使用循环来显示 Session 中的关键字和值

### 3. 不使用 cookie 的 Session

在默认情况下，ASP.NET 使用 cookie 来存储 Session ID 并跟踪用户。但如果用户的浏览器不支持 cookie 或用户不允许创建 cookie，情况将如何呢？幸运的是，ASP.NET 拥有另一种跟踪会话的方法。

**新术语:** cookie munging 是一种 ASP.NET 不使用 cookie 来跟踪会话的方式。在页面被发送给浏览器之前, ASP.NET 对 HTML 代码进行扫描, 以找到其中的超链接。ASP.NET 在每个链接的后面加上一个经过编码的 Session ID。当用户单击超链接时, ASP.NET 获取该字符串, 对其进行解码, 并将它传递给用户请求的页面, 该页面便可以使用该 ID 来存储或检索任何 Session 变量。ASP.NET 还在该页面的每个超链接中加上编码后的 Session ID。当 ASP.NET 发现访问者不支持 cookie 时, 将自动完成上述操作。图 4.10 说明了这一概念。

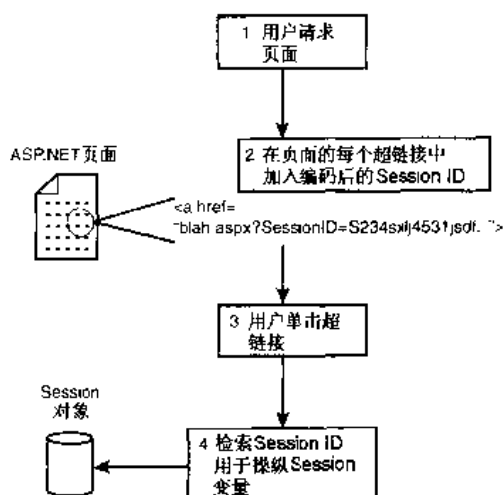


图 4.10 cookie munging 需要以查询字符串的方式将编码后的 Session ID 加入到超链接中

第 18 章将介绍一些更先进的配置 Session 的方法。

应 该	不 应 该
如果有少量用户信息需要存储, 以便维护当前会话, 如用户名或密码, 则使用 Session 变量	如果需要为每个用户存储大量的信息, 则不要使用 Session 变量。其他一些方法不会像 Session 变量那样飞快地耗尽服务器的内存, 如 cookie 和数据库

#### 4.3.6 HttpApplication 对象

HttpApplication 对象代表一个 ASP.NET 应用程序(在 ASP.NET 中, 应用程序指的是一个虚拟目录(包括子目录)中的所有文件)。我们将在第 18 章更为详细地介绍该对象, 本节只介绍一些初步知识。

**新术语:** 和 Response 对象一样, 每当应用程序开始运行时(即某个用户第一次向站点请求页面时), ASP.NET 将创建一个名为 Application 的 HttpApplication 对象。对于整个应用程序, 将只创建一个 Application 对象, 而不像 Session 对象那样为每个用户创建一个。但和 Session 对象一样, Application 对象也可以用来存储变量和对象。这些应用程序变量(Application variable)在整个应用程序中都是可用的, 而不是专门针对某个特定用户的。

例如, 假设您需要在每个页面中加入页脚或否认声明, 则可以将它存储在应用程序变量中, 如下所示:

```
Application("Disclaimer") = "Copyright 2001"
```

然后在每个页面中引用它:



```
Response.Write(Application("Disclaimer"))
```

如果将其存储在 Session 对象中, 则每个用户将在其会话中收到该字符串。如果有 100 名用户, 则需要存储该字符串 100 次, 这将浪费大量的内存。将其存储在 Application 对象中就可以消除这个问题, 因为它将被存储在一个地方, 占用的内存将少得多。

应 该	不 应 该
当变量不是针对某个用户时, 请使用应用程序变量; 存储个人信息时, 请使用 Session 变量	不要使用应用程序变量来存储个人信息, 如用户名、密码等; 当可以将其写入到 HTML 页面中时, 不要将大量的 HTML 存储在 Application 对象中

利用 Application 对象, 您可以处理许多事件, 这将在第 18 章介绍。

#### 4.3.7 HttpServerUtility 对象

本章要介绍的最后一个对象是 HttpServerUtility 对象, 该对象提供了几个用于处理请求的助手 (helper) 方法。可以使用名称 Server 来访问该对象的成员。

##### 1. 重定向用户

前面介绍了使用 Response 对象重定向用户的方法, 还有两种其他的方法: Response.Redirect 方法将 HTTP 信息发送给浏览器, 命令其跳到另一个页面。这是一个没有必要的往返过程, 因此 ASP.NET 也可以使用 HttpServerUtility 对象的方法 Execute 和 Transfer 来将用户带到其他页面。

Server.Transfer 只是转移到执行另一个页面, 这不用向用户发送任何信息——它完全是在服务器上进行的, 不需要客户知道。例如, 下述代码片段检查用户的密码 (假设他在表单中输入了一个):

```
if (strPassword == 'blablabla') {
    Server.Transfer('success.aspx')
```

如果密码与第 1 行的字符串匹配, 则将他带到另一个页面。这对于轻松地将用户重定向至关重要。

Server.Execute 让 ASP.NET 执行另一个页面, 但结束后重新返回到原来的页面。如果需要执行另一个页面中的代码 (例如处理数据), 并在执行后返回原来的页面, 则这种方法很有用。其工作原理与前一章讨论的分支逻辑很相似, 其格式为 Server.Transfer。

##### 2. 格式化字符串

通常, 将输出发送给浏览器时, 输出将被看作是 HTML。例如, 下面的代码在浏览器中换行:

```
Response.Write("<br>")
```

但有时候, 可能想打印实际的单词。您想在访问者的屏幕上显示<br>, 而不是换行符。为此, 可以使用 HttpServerUtility 对象的 HtmlEncode 方法。下述代码将输出<br>, 而用户将在其浏览器中看到<br>:

```
Response.Write(Server.HtmlEncode("<br>"))
```

在 HTML 编码中, &lt; 等同于<, 而&gt; 等同于>。HtmlEncode 将在文本中完成上述替换。

UrlEncode 的功能与此类似, 但使用 URL 的规则格式化指定的字符串。例如, 在 URL 中, & 和? 都有特殊的含义, 因此 Server.UrlEncode 将这些字符转换为相应的 URL 编码版本。如果要将字符串放到查询字符串中, 则该方法很有用。

Server.HtmlDecode 和 Server.UrlDecode 执行相反的操作, 它将编码后的字符序列转换为其代表的

字符（例如，将&lt;转换为<）。

MapPath 方法不用于格式化字符串，但有助于确定您需要的合适字符串。例如，下述代码：

```
Server.MapPath("~/TYASPNET/21Days/day1")
```

将得到下述结果：

```
C:\inetpub\wwwroot\TYASPNET\21Days\day1
```

当您需要知道物理路径，如当您需要将文件写入到服务器（见第13章）时，该方法很有用。

### 3. 控制脚本

HttpServerUtility 对象提供了一个用于控制 ASP.NET 脚本执行的方法：ScriptTimeout，这个值告诉服务器等待脚本结束的时间。例如，下述代码告诉 ASP.NET，对于执行时间超过 90 秒的脚本，都应该终止。

```
Server.ScriptTimeout = 90
```

有时候，编写的代码可能有 bug，如死循环，它将不断运行，直到服务器崩溃。使用上述属性，可以避免这种情况发生。对于执行时间超过 ScriptTime 的任何脚本，ASP.NET 都认为它无法正确运行，并将其终止。

另一方面，有些代码确实需要执行较长的时间（例如它可能非常复杂），而 ScriptTime 属性的值可能过小，此时您可以增大它的值。

### 4. 创建对象

最后 CreateObject 方法实例化一个有 progid 标识的 COM 对象。现在不讨论它，我们将在第三部分详细地进行讨论。

## 4.4 深入学习 C#的资源指南

C#是一种非常复杂的、功能强大的语言，本章并没有深入介绍它。但您有一个良好的开端，您马上就可以将所学到的 C#技巧用于 ASP.NET 页面中。更详细的关于 C#的参考资料请从以下资源中查找：

- 在线 MSDN (<http://msdn.Microsoft.com>)；
- .NET 框架 SDK 文档。

## 4.5 这不是 ASP

如果您熟悉传统的 ASP，您肯定已经注意到了许多不同的地方和相似的地方。许多您熟悉的 ASP 对象在.NET 框架中仍然可用，但格式稍微有些不同。例如，现在有一个名叫 HttpCookie 的对象，它提供了操纵 cookie 的属性和方法。Request、Response 和 Session 对象基本上变化不大。HttpServerUtility 对象代替了以前的 Server 对象，而现在有一个新的对象 Page，它表示页面本身。

这些对象提供了大量的功能，它们在 ASP.NET 中的概念及其实现方法与在 ASP 中的相同。例如，Session 变量的使用仍然遵守 ASP 中使用的内存限制。但该对象和其他对象在幕后提供了稳定得多、功能强大得多的环境。Session 对象现在可以跨服务器存储变量，甚至在服务器崩溃时，仍可以保存数据。第 18 章将更详细地介绍这些内容。

转移到面向对象编程可能需要适应许多东西，但一旦适应后，便可以简化工作。学会如何使用内置的.NET 对象至关重要，因为它们是开发强大的 ASP.NET 应用程序的关键。

本章和前一章介绍了两种新的 ASP.NET 编程语言：VB.NET 和 C#。幸运的是，就 VBScript 开发人员而言，并不需要做太多的调整。对 C 和 C++ 开发人员来说，这些语言打开了一个全新的世界，他们现在可以使用熟悉的语言来开发 ASP.NET 页面。

虽然一些 ASP.NET 特性只是表面上比传统 ASP 有所改进，但新的框架提供更强大的功能，您不久就可以使用到它。

## 4.6 总 结

本章简要地介绍了 C# 及其语法，其中的许多编程概念与 VB.NET 类似。另外，还介绍了一些最常用的 ASP.NET 对象，并使用 C# 和 VB.NET 实现了一些例子。这两种语言都非常强大，并可以相互转换。

C# 的编程方法与 VB.NET 极其相似，主要区别在于语法——代码行以分号结束，声明变量的方法不同（在变量名称前面指定数据类型）并使用花括号代替了大量的关键字。

对象包括描述自己的属性和方法，其中的许多属性和方法必须通过对象实例才能使用，但静态成员不需要。

Response 对象让服务器能够与浏览器进行通信，而 Request 对象则让浏览器能够与服务器进行通信。Response 对象对于向用户显示信息以及重定向用户很有用，而 Request 让您能够获得关于客户的信息。

HttpCookie 对象提供了读写 Cookie 的机制，这主要是通过 Response 和 Request 对象完成的。记住，cookie 存储在客户机上，因此它们不会占用宝贵的服务器资源。

Page 对象代表 ASP.NET 页面，该对象的 Load 事件很有用，以后您将经常使用它。

Session 对象存储了关于用户的当前会话的信息，可以将个人信息保存在 Session 变量中，如用户名和偏好。Session 变量将占用服务器资源，随着用户增多，将很快耗尽内存，因此对于这种维护用户信息的方法必须做出明智的衡量。

HttpApplication 对象代表 ASP.NET 应用程序，后者包括虚拟目录及其子目录中的所有文件和页面。该对象有一些很有用的事件，这将在第 18 章中进行讨论。它还可以用于存储应用程序级的变量。

最后，HttpServerUtility 对象提供了帮助开发人员处理用户请求的方法。这包括格式化字符串、重定向用户、控制脚本的执行以及创建对象等。

本章包括大量关于 ASP.NET 对象的内容，下一章将介绍 Web 表单，一种使用 ASP.NET 创建交互式用户界面的新框架。该框架提供了大量其他的对象，如 HTML 服务器控件和有效性验证控件。

## 4.7 问与答

问：C# 像 VB.NET 那样支持函数的可选参数吗？

答：不直接支持。但您可以提供接受不同参数列表的重载方法来模拟可选参数。

问：ASP.NET 支持缓存技术吗？

答：支持。ASP.NET 用于复杂的缓存系统（将在第二部分介绍），而不是像传统 ASP 那样使用 Response.Expires 方法。

## 4.8 作业

下面的作业帮助您巩固本章介绍的概念。答案见附录 A。

### 4.8.1 小测验

1. 何为静态成员?
2. 下面的代码片段能正常运行吗?

```
<html><body>  
  
    Hi there!  
  
    <%  
  
        Response.Redirect("/page2.aspx");  
  
    %>  
  
</body></html>
```

3. 下述代码片段是使用何种语言编写的?

```
Response.Write("Hello there!");
```

### 4.8.2 练习

1. 使用 C# 创建一个 ASP.NET 页面，将其作为安全登录页面。让用户在文本框中输入其用户名和密码，如果与您指定的字符串相同，则将用户重定向到一个“Success”页面；否则显示一个错误。如果用户是合法的，则将其用户名存储到一个 Session 变量中。

2. 使用 VB.NET 编写“Success”页面。使用 Page 对象的 Load 事件在一个 Label 控件中显示一条个性化的欢迎消息，并将当前时间和用户的 Session ID 告诉用户。检查 Page 对象的 IsPostBack 属性，以确保只在其首次打开该页面时才显示。提供一个让用户注销的按钮，同时当用户注销时，显示确认消息。

## 第 5 章

# Web 表单初步

在 Web 开发中，Web 表单是个有趣的概念，它使得 ASP.NET 能够通过使用驻留在服务器中的对象来控制用户界面。Web 表单是 ASP.NET 不可或缺的一部分。

本章介绍 Web 表单框架以及为何要在 ASP.NET 中使用它们。这些控件可以极大地提高 ASP.NET 页面的功能，并让您能够轻松地开发出复杂的用户界面。

下一章将继续讨论一些更高级的 Web 表单话题。

本章包括以下内容：

- 什么是 HTML 表单，它们是如何运作的；
- 为什么 Web 表单优于 HTML 表单；
- 如何使用服务器控件；
- 如何响应控件事件；
- 如何使用 HTML 服务器控件；
- 如何使用 WEB 服务器控件。

### 5.1 表单简介

我们再看一下 Web 的工作原理。第 1 章介绍过，用户向服务器请求信息，服务器通过发送相应的文件来进行响应，这就是请求/响应模型的基础。

客户也可以向服务器发送数据。HTML 表单让用户能够与 Web 页交互，即在用户与服务器之间提供了通信的途径。例如，清单 5.1 列出了一个典型的 HTML 表单，让用户能够输入自己的姓名，并单击 Submit 按钮。

**清单 5.1 一个典型的 HTML 表单**

```
1 <html><body>
2   <form method="post">
3     Please enter your name:
4     <input type="text" size="20">
5     <input type="Submit" value="Submit">
6   </form>
```

```
7 </body></html>
```

用户单击 Submit 按钮后, 表单将把用户输入的所有数据发送给服务器。这与客户请求相似, 但是浏览器同时发送了附加信息, 供服务器使用。HTML 表单由构成用户界面的元素组成, 是完全基于客户的。用户在各元素中输入数据后提交表单, 将数据发送给服务器。图 5.1 说明了这一概念。

服务器和客户之间的通信只能通过发送和请求进行, 因此所有基于 Web 的应用程序都依赖于这种模型, 这使得编程有时会很困难。由于客户和服务器之间的交流不足, 因此无法得到关于对方的详细信息。例如, 服务器不知道客户浏览器能干什么或要干什么, 这使得客户的行为完全不可预见。

服务器也对用户界面的外观或表单会发送什么类型的数据一无所知, 它所知道的仅是表单告知的信息, 并且通常很少。发送完成后, 服务器就忘了表单告知的信息, 因此无法知道接下来的情况。



图 5.1 服务器收到的只是用户输入的信息

## 5.2 Web 表单简介

Web 表单与传统的 HTML 表单很相似。不同之处在于, Web 表单是基于服务器的, 也就是说您将在服务器上创建用户元素。服务器对界面的外观、功能以及期望的数据等一清二楚。

**新术语:** 在服务器上, 您创建被称为服务器控件 (server controls) 的对象, 它们表示 UI 部分不同于 HTML 表单元素, 这些对象是完全可以控制的, 它们有属性、事件以及方法。当用户请求页面时, ASP.NET 把这些控件转换为 HTML, 后者可以显示在浏览器中。图 5.2 说明了这一过程。

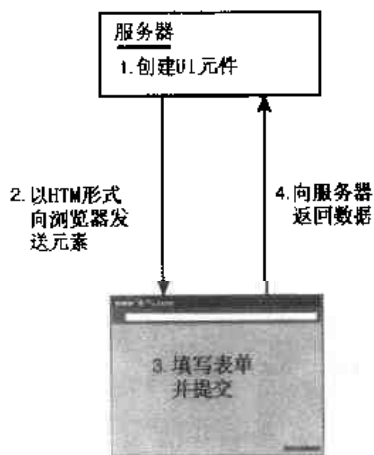


图 5.2 使用 Web 表单, 服务器知道表单的外观和功能

通过 ASP.NET 自动生成的客户端脚本, 这些控件可以将发生的所有事情 (例如单击按钮) 告知服务器。事件发生后, 客户端脚本将把信息发送给服务器, 而用户感觉不到这一点。因此, 服务器不断地被告知客户端发生的情况, 从而服务器和客户被紧密地联系在一起。

注意：上面说明了事件驱动模型的优点。但在底层，ASP.NET 仍依赖于请求/响应模型来收发信息。这种建立在请求/响应模型之上的事件驱动模型，为应用程序开发提供了更直观的环境。

同时，由于控件是由服务器创建的，因此它能记住每个控件的输入。第 2 章介绍过，ASP.NET 用隐藏表单字段来记录每个控件的视图状态，并相应地更新它们。

通过告诉服务器用户界面上发生的情况，Web 表单让我们工作得更容易。设想您的眼睛被蒙住，由乘客喊“左”、“右”来指挥您开车的情形。使用传统的 HTML 表单与此类似，它们不将客户端的情况告知服务器，仅偶尔提供一些数据。服务器怎么能够正确地控制应用程序呢？

Web 表单除掉了眼罩。服务器能够知道将要发生的情况并正确地驾驭程序。

### 5.3 Web 表单编程模型

Web 表单页面分为两部分：可视元素和 UI 逻辑。从概念上讲，这两部分是完全独立的，可以将它们放在任何物理位置，但通常将它们放在同一个.aspx 文件中。清单 5.2 是一个 Web 表单的例子。

清单 5.2 典型的 Web 表单页面

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub lblMessage_Change(Sender As Object, E As EventArgs)
5         lblMessage.Text = "Hello " + tbMessage.Text
6     End Sub
7 </script>
8
9 <html><body>
10     <font size="5">Sam's Teach Yourself ASP.NET in 21 Days:
11     Day 2</font><hr><p>
12     <% Response.Write('Our first Page<p>') %>
13
14     <form runat="server">
15         Please enter your name:
16         <asp:textbox id="tbMessage"
17             OnTextChanged="tbMessage_Change"
18             runat="server"/>
19         <asp:button id="btSubmit" Text="Submit"
20             runat="server"/><p>
21         <asp:label id="lblMessage" font-size="20pt"
22             runat="server"/>
23     </form>
24 </body></html>

```

分析：这个清单您应该看着眼熟——这是第 2 章的一个例子。UI 包含在页面的 HTML 部分中，

由 16、19、21 行的服务器控件组成。这些控件包含您可以控制的属性、方法和事件。UI 逻辑，即控制 UI 的代码，包含在 3-7 行的代码声明块中。

创建 Web 表单的工作并不神秘。包含服务器控件和 UI 逻辑的 ASP.NET 页面就是 Web 表单。我们来看一下组成 Web 表单的一些片段。

### 5.3.1 服务器控件

前面讨论过，服务器控件是 Web 表单的用户界面元素。ASP.NET 有四种服务器控件：HTML 服务器控件、Web 控件、有效性验证控件和用户控件。HTML 服务器控件代表常规的 HTML 表单元素，如文本输入框和按钮，但它们是在服务器上创建的，您可以在服务器上控制它们。Web 控件与此类似，但提供了更多的功能，可以提供更复杂的用户界面。有效性验证控件用于验证用户的输入。用户控件是自定义的控件，用于实现某些功能。除了有效性验证控件将在第 7 章介绍外，其他控件将在本章或下一章介绍。

所有服务器控件都有属性、方法和事件。它们提供的功能比传统 HTML 表单元素多，使开发人员创建用户界面更容易。

创建服务器控件时，不必为编写 HTML 代码而操心。页面被请求时，服务器控件将自动生成正确的 HTML 代码。例如下面的代码行将在服务器上创建一个按钮服务器控件：

```
<asp: Button text="Submit" runat="server" />
```

当该按钮被请求时，它将自动为客户生成下面的一行 HTML 代码：

```
<input type="Submit" name="ctl1" value="Submit" />
```

这两行几乎没有相象之处。前一行仅供服务器使用——客户看不见它（即使浏览器可以见到，也看不懂，因为它仅能识别 HTML 代码）。客户看到的是后一行。以这样的方式创建控件，您可以少花一些时间去考虑提供 UI 布局的 HTML 代码，而将更多的时间用来考虑 UI 应该实现什么功能。

**新术语：**另外，ASP.NET 也知道各种浏览器的功能，因此能够给每个浏览器发送恰当的 HTML 代码。例如，如果浏览器不支持动态 HTML (DHTML)，ASP.NET 就不发送动态 HTML。这就是所说的下限支持 (down-level support)，因为对于不支持高级功能的浏览器而言，ASP.NET 可以降低 HTML 输出级别。

**警告：**从理论上说，下限支持可以正确处理所有控件的每一个问题。但实际上，您并没有那么幸运。由于不同浏览器可能会对 HTML 进行不同的处理，有些元素在不同浏览器上显示时可能会出现问题。当然，您的开发用于主流浏览器时，下限支持会在大多数情况下正确地运行的。

### 5.3.2 服务器控件事件

服务器控件能引发多种事件。换句话说，用户可以对服务器控件执行许多操作：单击按钮、单击链接、填写文本框、选择列表框中的条目，等等。所有这些事件都要由服务器进行处理，所以每当事件发生时，客户都要向服务器发送信息。

**注意：**将被动用户事件与主动事件区别开来很重要。主动事件要求用户进行明确的操作，即用户必须进行有意识的操作，包括单击按钮或链接、填写文本框等。被动事件是用户无意中执行的操作，例如鼠标在图像上移动。

只有主动事件才会被在服务器上进行处理，因为被动事件太多，无法将其所有数据发送给服务器，不过可以通过客户端脚本来处理它们。

向服务器发送事件的方式有两种：发生时立即发送和积累一段时间后一起发送。后者的效率更



高，因为不用发送那么多次，这意味着在客户端和服务端之间传递的数据更少。

**新术语：**大多数情况下，您将使用后一种方法。例如，假设用户将其姓名输入到表单中，每键入一个字母，就发生了一个事件。您并不希望每更改一次字母就发送一次数据，而是希望在用户输入好数据或单击 Submit 按钮后再发送。这种事件被缓存 (cached)，即在用户决定发送数据前，它们被存在客户端，这样服务器可以一次性处理这些事件。

**注意：**实际上，这种事件被称为 TextChanged 事件，它只在用户输入文本并离开该 UI 元素时（即使用 Tab 键移动到另一个元素时）发生，但道理是一样的。

您也可以强行将这些事件立即发送给服务器，这将在本章后面的“立即发送数据”一节中介绍。

正如第 3 章介绍过的，ASP.NET 中的所有服务器控制事件都会向服务器发送两个对象：一个代表引发事件的控件，另一个描述关于此事件的特定信息。

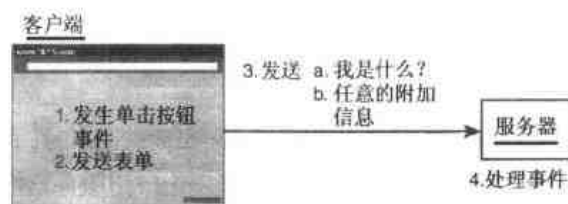


图 5.3 服务器控件发送描述其本身和关于该事件的特殊信息

有事情发生时，控件自动引发事件。为了在服务器上处理事件，需要告诉控件使用哪种方法。

例如，下面的代码告诉 ASP.NET，发生 Click 事件时，使用 ClickHandler 方法来进行处理：

```
<asp:Button runat="server" OnClick="ClickHandler" />
```

该方法与下面类似：

```
1: sub ClickHandler(obj as Object,e as EventArgs)
2:     do something...
3: end sub
```

正如第 3 章指出的，第 1 行是 ASP.NET 服务器控件的标准事件参数列表。第二个参数随发生的事件而异。大多数情况下，您都可以使用这种标准参数列表。

例如，请看清单 5.3 中的例子。

### 清单 5.3 事件处理程序

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub Button1_Click(obj as object, e as EventArgs)
5         Label1.Text="You clicked <b>' & obj.Text & "</b>"
6     end Sub
7 </script>
8
9 <html><body>
10     <font size="5">Sam's Teach Yourself ASP.NET in 21
11     Days: Day 5 </font><hr><p>
```

```

12
13 <form runat="server">
14     <asp:Button id=Button1 runat="server" Text="Button1"
15         onClick="Button1_Click" />
16     <p>
17         <asp:Label id=Label1 runat=server />
18 </form>
19 </body></html>

```

**分析：**单击按钮后，代码的输出如图 5.4 所示。

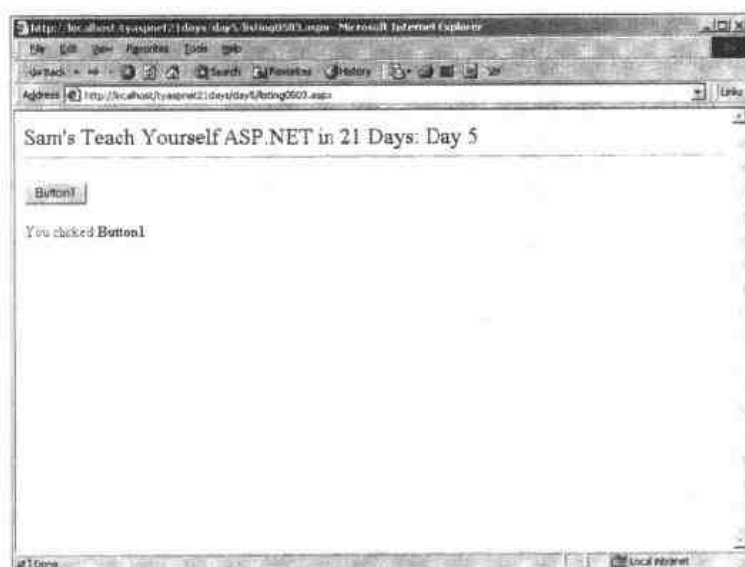


图 5.4 清单 5.3 生成的页面

尽管没看到服务器控件，但您知道第 14 行的按钮控件有一个 Click 事件。该事件向服务器发送信息，并执行代码声明块中 4-6 行定义的 Button1\_Click 方法。

**警告：**不要忘记加入 runat="server"，这样服务器控件才能正常运行。

另外，<form>标记在这里也是必不可少的。没有它，数据将不会被发送给服务器，页面也不能正常运行。

让我们来看看第 5 行：

```
5: Label1.Text='You clicked <b>' & obj.Text & '</b>'
```

该行将名为 Label1 控件（第 17 行的 Label）的 Text 属性设置为等号后面的值，obj 是其中一个参数的名称参数。因为您知道它代表一个按钮，因此知道它有 Text 属性。再看第 14 行，该按钮的 Text 属性被设置为 Button1。下面的代码行可以得到同样的输出：

```
Label1.Text="You clicked <b>' & Button1.Text & "</b>"
```

因为按钮控件没有发送其他的信息，因此方法的第 2 个参数 (e) 当前没有包含任何信息。然而，您还是要将它包括在内，因为它是标准事件参数列表的一部分。

这样，您就可以按照您的想法来处理事件了。事件将不断发生，您可以提供用于处理事件的方法。对于想忽略的事件，不提供相应的事件处理程序即可。另外，这种方法使您能够在 一个方法中

处理多种不同的事件。您完全可以使用同一个方法来处理 20 个按钮控件。

**警告：**假设在服务器控件中指定了用于处理某个事件的方法：

```
<asp:Button runat="server" OnClick="SomeMethod" />
```

而没有在代码声明块中创建该方法，将发生错误。

### 5.3.3 发送 Web 表单

设想一种典型的 ASP.NET 情形。当页面被加载到浏览器中时，在服务器控件上显示一条欢迎消息。这条消息将成为控件的视图状态的一部分，ASP.NET 将自动记录它。

随后，用户提交表单，或某一事件发生，导致它向服务器发送信息。服务器将对数据或事件进行处理，再把页面发回客户端，并显示出来。

由于 ASP.NET 记得控件的视图状态，因此它将自动再次填写欢迎消息，而无需您的干涉。该消息将与用户输入的内容一致。如果用户在文本框中输入了其姓名，则表单被提交后，其姓名将保留在文本框中。这与 HTML 表单不同，HTML 表单中的值在发送后将丢失。

正如第 4 章介绍过的，Page 对象有一个叫作 IsPostBack 的属性，指出表单是否已经发送过。您可以检查该属性，以决定是否需要重新填写服务器控件。例如，我们看清单 5.4。

#### 清单 5.4 检查 IsPostBack 属性

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub Page_Load(obj as Object, e as EventArgs)
5         if not Page.IsPostBack then
6             lblMessage.Text = "Hello world!!!!"
7         end if
8     end sub
9
10    Sub Submit(obj as Object, e as EventArgs)
11        lblMessage2.Text = "Form posted"
12    end Sub
13 </script>
14
15 <html><body>
16     <form runat="server">
17         <asp:Button id="btSubmit" runat="server" Text="Submit"
18             onClick="Submit" />
19     <p>
20         <asp:Label id="lblMessage" runat="server" />
21     <p>
22         <asp:Label id="lblMessage2" runat="server" />
23     </form>
24 </body></html>
```

**分析：**该页面被装载后，第 20 行的标签将显示“Hello World!!!”。用户单击 Submit 按钮后，第 22 行的第二个标签将显示“Form posted”。第一个标签仍然显示原来的内容，因为 ASP.NET 已将其保存在视图状态中，并自动填写它。

第 5 行检查 IsPostBack 属性，如果为真（即表单已被发送），则不执行第 6 行。不用在每次发送后都重新填写标签，因为 ASP.NET 都替您做了。图 5.5 是表单被发送后，这些代码生成的输出。

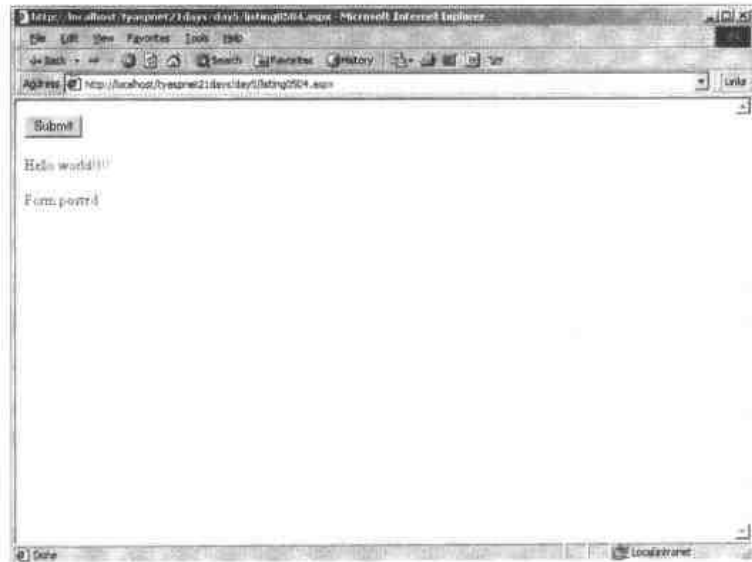


图 5.5 表单被发送后，文本仍然保留在标签中，您无需重新填写

这是一个简单的例子，不用运行第 6 行，并没有节省多少时间。但当命令很多、很复杂时，检查 IsPostBack 属性将非常有用。

#### 5.3.4 保存状态

您知道，Web 表单通过表单隐藏字段来保存表单中每个控件的视图状态。视图状态指出控件中输入的内容（如果有的话）、控件是否被选中，哪个列表项被选中等信息。视图状态提供了大量的信息，ASP.NET 可以用来维护每个服务器控件的状态。

还有另一种方法可用来保存 Web 表单的信息。您可以使用状态包（state bag），一个保存表单被发送时的值的对象。将信息加入状态包中，并提交表单时，服务器将保存这些信息，并在处理完毕后，将它们发回给客户。这是一种存储非用户输入的自定义信息（如计算）的简便方法。您可以通过 ViewState 变量访问状态包。

我们来看一个例子。清单 5.5 在页面被装载时将当前时间保存在状态包中。用户单击 Submit 按钮后将对开始时间和新时间进行比较。

##### 清单 5.5 将值保存到状态包中

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub Page_Load(obj as object, e as EventArgs)
5         if not Page.IsPostBack then
6             ViewState("StartTime") = DateTime.Now
```

```

7      lblMessage.Text = 'The time is now: ' & _
8          ViewState('StartTime')
9      end if
10 end sub
11
12 Sub Submit(obj as object, e as EventArgs)
13     lblMessage.Text = "The time is now: " & DateTime.Now & _
14         '<br>started at: ' & ViewState("StartTime")
15
16 end Sub
17 </script>
18
19 <html><body>
20     <font size="5">Sam's Teach Yourself ASP.NET in 21
21     Days: Day 5 </font><hr><p>
22
23     <form runat="server">
24         <asp:Button id="btSubmit" runat="server" Text="Submit!"
25             onClick="Submit" />
26     <p>
27         <asp:Label id="lblMessage" runat="server" />
28     </form>
29 </body></html>

```

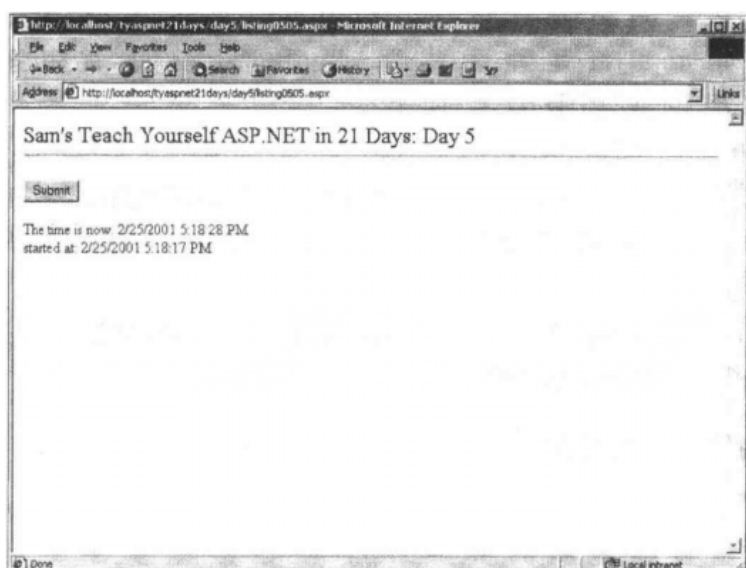


图 5.6 状态包保存了页面首次被查看时的时间

**分析:** 您想保存页面首次被浏览时的时间, 因此在 Page\_Load 处理程序中的第 5 行检查 IsPostBack

属性。如果是首次浏览，则将当前的时间（表示为 `DateTime.Now`）保存在 `ViewState` 中。`ViewState` 是 ASP.NET 创建的状态包的名称，与之交互的方法与上一章介绍的 `Session` 对象相同。然后，将该时间显示在标签控件中。

每当用户单击 `Submit` 按钮，`Submit` 方法便被执行。该方法始于第 12 行，它将当前时间和存储在状态包中的时间显示在标签上。图 5.6 是该清单的输出结果，状态包的数值与当前值不同。

**警告：**与 `Session` 或 `Application` 对象不同，用户离开页面时，状态包会被清空。只有在同一个表单被重复发送时，数值才会被保存。一旦加载新页面，状态包将被丢弃。

应 该	不 应 该
需要为特定表单保存自定义值时，应该使用状态包	需要长期保存信息时，不要使用状态包，而应使用 <code>Session</code> 或 <code>Application</code> 对象、 <code>cookie</code> 或数据库

### 5.3.5 Web 表单的处理顺序

有时候需要按特定顺序处理表单。例如您可能希望某个事件被处理后再显示消息，这一点在第二部分介绍数据库时非常重要，所以知道 Web 表单的处理顺序很有帮助：

1. 页面被请求（或发送）；
2. 存储所有控件的视图状态；
3. `Page_Load` 事件发生；
4. 处理事件（即处理事件的方法被调用）。首先处理被缓存的事件，随后是发送表单的事件。

如果多个事件同时被引发，则不以特定的顺序处理它们；

5. `Page_Unload` 事件发生。

该过程由用户请求页面引发。然后，ASP.NET 将恢复页面上的每个控件的状态，并引发 `Page_Load` 事件，该事件可能有也可能没有相应的事件处理程序。然后按照开发人员定义的方式处理发生的所有事件。最后引发 `Page_Unload` 事件，释放没必要再占用的内存。随着您开发更多的 Web 表单，将对该过程非常熟悉。

## 5.4 HTML 服务器控件

您已经熟悉了 HTML 元素：文本输入框、列表框、按钮、图像、表格等，这些元素都代表页面中某个可视的部分。例如可以用下面的 HTML 标记创建一个文本框：

```
<input type="text" id="MyTextBox" value="blah" />
```

HTML 元素完全是基于客户的，服务器对这些控件一无所知。浏览器知道 `<input type="text">` 的含义并能正确地进行处理。

正如本章前面的“Web 表单简介”一节指出的，HTML 服务器控件是服务器端元素，它们是在服务器中创建的对象，包含属性、方法和事件。它们生成 HTML，供浏览器进行显示。

创建 HTML 服务器控件非常容易——只要在 HTML 元素中加上 `runat="server"` 属性就可以了。每个 HTML 元素都有对应的 HTML 服务器控件。事实上，有一个控件您已经用过多次了：`HtmlForm` 控件，它有些像 `<Form runat="server">`。下面是一个 `HtmlInputText` 控件：

```
<input type="text" id="MyTextBox" runat="server" />
```

这里服务器创建了一个 `HtmlInputText` 控件。当客户请求页面时, ASP.NET 生成相应的 HTML 代码, 将控件显示为文本输入框。创建 HTML 服务器控件的代码与它们所代表的 HTML 元素相同, 只是需要加上 `runat="server"` 属性。

将 HTML 元素转换为 HTML 服务器控件后, 便可以通过代码来修改元素的每个属性。例如, 可以使用下面的代码来修改上述 `HtmlInputText` 控件中的文本:

```
MyTextBox.Value = "Text Changed"
```

我们来看一个例子, 清单 5.6 说明了如何修改 HTML 服务器控件的属性。

#### 清单 5.6 操纵 HTML 服务器控件

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     Sub Click(obj as object, e as EventArgs)
5         select case obj.Value
6             case "Left"
7                 word_image.align = 'left'
8             case "Right"
9                 word_image.align = 'right'
10            case "Center"
11                word_image.align = 'center'
12        end select
13
14        Left.Style("Border-Style") = 'notset'
15        Right.Style("Border-Style") = "notset"
16        Center.Style("Border-Style") = 'notset'
17
18        obj.Style("Border-Style") = 'inset'
19    end Sub
20 </script>
21
22 <html><body>
23     <font size="5">Sam's Teach Yourself ASP.NET in
24     21 Days: Day 5 </font><hr><p>
25
26     <form runat="server">
27         <input type="Button" id="Left" runat="server"
28             Value="Left" OnServerClick="Click" />
29         <input type="Button" id="Center" runat="server"
30             Value="Center" OnServerClick="Click" />
31         <input type="Button" id="Right" runat="server"
```

```

32     Value="Right" OnServerClick="Click" />
33 </form>
34
35 
36 <div id="Label1" runat="server">This is an example
37 of text. By pressing the buttons above, the image
38 will move around the text accordingly.<p> This example
39 demonstrates the HtmlImage and HtmlInputButton controls.
40 </div>
41
42 </BODY> </HTML>

```

分析：我们先来看看从第 22 行开始的 HTML 部分。这里有 6 个 HTML 元素：1 个表单、3 个输入按钮、1 个图像和 1 个<div>段（section），请注意所有元素都有属性 runat="server"，使之成为 HTML 服务器控件。

HtmlInputButton 控件（代表输入按钮）都有一个叫作 ServerClick 的事件，该事件在按钮被单击时发生。事件发生后，在服务器上执行 Click 方法，如第 28、30、32 行所示。

注意：如果您熟悉客户端脚本，就会知道 HTML 按钮也有 Click 事件，该事件发生在客户端。确保您的代码中使用了正确的事件：ServerClick 是用于服务器端的事件，Click 则用于客户端（也就是用于动态 HTML 的功能）。

我们回过头来看看代码声明块。这里唯一的方法是 Click 事件句柄，它使用 Case 语句确定调用自己的按钮。您知道，obj 代表按钮服务器控件，所以您检查它的值以确定哪个按钮被单击了，然后相应地设置图像的 align 属性。

第 14-16 行设置了按钮元素的样式（注意这些样式在常规 HTML 元素中也能设置）。具体地说，您希望被单击的按钮看起来好像是被压下去了，因此您首先通过将每个按钮控件的样式设置为 notset，使之被弹起，然后在第 18 行将 obj 代表的按钮的样式设置为 inset。图 5.7 和 5.8 显示了两个按钮被单击后的情况

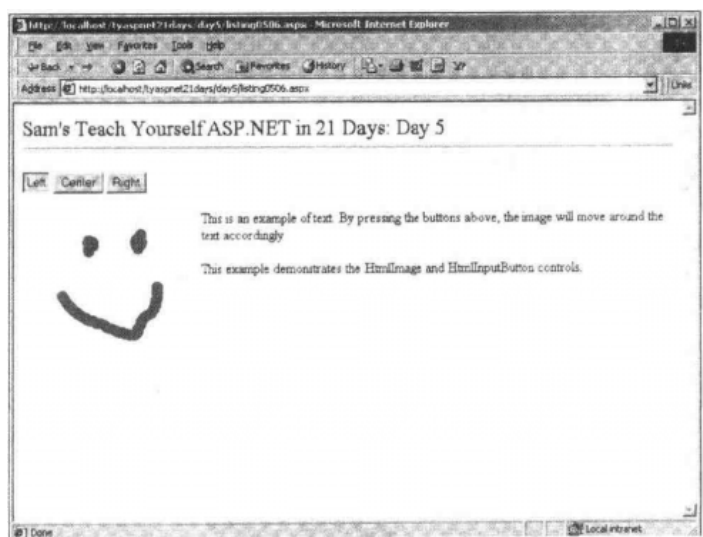


图 5.7 单击 left HTML 服务器按钮的结果



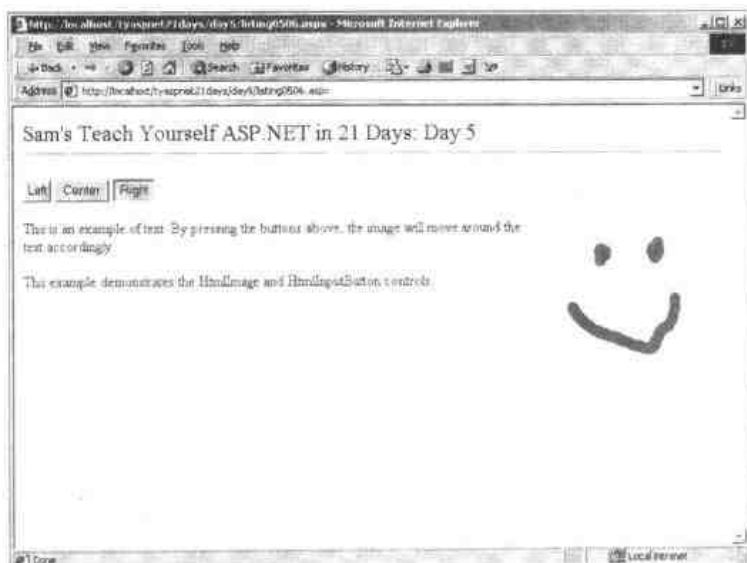


图 5.8 单击 HTML 服务器按钮 right 后的结果

注意：别忘了把图像源改为服务器上确实存在的图像。

被转换为 HTML 服务器控件后，HTML 元素的所有属性都将易于操作。

表 5.1 总结了预定义的 HTML 服务器控件。只要加上属性 `runat="server"`，便可以将没有包括在该表中的其他任何 HTML 元素转换为 HTML 服务器控件。所有这些控件都属于名字空间 `System.Web.UI.HtmlControls`。

表 5.1 HTML 服务器控件

HTML 控件	描 述
<code>HtmlAnchor</code>	创建一个 Web 导航链接。 <code>HtmlAnchor</code> 控件引发发送操作。 HTML 元素： <code>&lt;a&gt;</code>
<code>HtmlButton</code>	执行一项任务。这个控件可以包含任何 HTML，因此在外观上非常灵活，但它并不与所有浏览器兼容。 HTML 元素： <code>&lt;input type="button"&gt;</code>
<code>HtmlForm</code>	定义一个 HTML 表单。表单被提交时，表单中控件的值将被发送给服务器。 HTML 元素： <code>&lt;form&gt;</code>
<code>HtmlGenericControl</code>	为被转换为 HTML 元素的控件创建一个基本的对象模型（属性、方法、事件）
<code>HtmlImage</code>	显示图像。 HTML 元素： <code>&lt;img&gt;</code>
<code>HtmlInputButton</code>	执行一项任务。所有浏览器都支持这种按钮。 HTML 元素： <code>&lt;input type="button"&gt;</code>
<code>HtmlInputCheckBox</code>	创建一个复选框。用户可以单击它来选中或取消选中。 <code>CheckBox</code> 控件包括一个标签。 HTML 元素： <code>&lt;input type="checkbox"&gt;</code>
<code>HtmlInputFile</code>	让用户能够指定要上传到服务器的文档（服务器必须允许上传）。 HTML 元素： <code>&lt;input type="file"&gt;</code>
<code>HtmlInputHidden</code>	存储表单的状态信息（即在服务器和客户之间通信时都需要的信息） HTML 元素： <code>&lt;input type="hidden"&gt;</code>
<code>HtmlInputImage</code>	像一个按钮，但显示一个图像。 HTML 元素： <code>&lt;input type="image"&gt;</code>

续表

HTML 控件	描 述
HtmlInputRadioButton	显示一个可以开关的按钮。Radio 按钮通常用于让用户从一个简短的固定选项列表选择一个选项。HTML 元素: <code>&lt;input type="radio"&gt;</code>
HtmlInputText	显示设计阶段输入的文本。用户可以在运行阶段编辑或修改。该控件还可用于创建密码框, 其中的值被显示为星号。HTML 元素: <code>&lt;input type="text"&gt;</code>
HtmlSelect	显示文本和图标列表。HTML 元素: <code>&lt;select&gt;</code>
HtmlTable	创建表格。HTML 元素: <code>&lt;table&gt;</code>
HtmlTableCell	在表格行中创建一个单元格。HTML 元素: <code>&lt;td&gt;</code>
HtmlTableRow	在表格中创建一行。HTML 元素: <code>&lt;tr&gt;</code>
HtmlTextArea	显示大量的文本。用于输入和显示多行文本。HTML 元素: <code>&lt;textarea&gt;</code>

## 5.5 Web 服务器控件

Web 服务器控件与 HTML 服务器控件类似, 它们是在服务器上创建的, 让您能够轻松地创建复杂的用户界面。它们必须包含属性 `Runat="server"`, 才能正确地运行。它们还提供了丰富的编程功能。

然而, 不同于 HTML 服务器控件, Web 控件与 HTML 元素并非一一对应, 它们能代表更复杂的 UI 元素。例如, Calendar Web 控件代表一个显示日历的、复杂的 HTML 表格, 包含多行、多列。

前面介绍了几个 Web 控件, Label 和 Button 便是其中的两个, 本章一直在使用这两个控件。可以使用下面的语法来创建 Web 控件:

```
<asp:Control id="name" runat="server" />
```

例如, 创建 Label 控件的代码如下所示:

```
<asp:Label id="Label1" runat="server" />
```

在浏览器中被显示时, 该控件显示一个 `<div>` 段。知道控件的名称后, 创建它们就很容易。表 5.2 对 ASP.NET 控件做了总结。

表 5.2 Web 服务器控件

Web 控件	描 述
AdRotator	显示一个预定义的或随机的图像序列
Button	用于执行一项任务
Calendar	显示一个让用户选择日期的图形日历
CheckBox	显示一个复选框, 用户可以单击来选中或取消选中
CheckBoxList	创建一组 Check box, 该列表控件使得创建多个 Check box 更方便

续表

Web 控件	描 述
DataGrid	在包括多列的表格式表单中显示信息（通常是被绑定的数据）提供允许编辑和排序的机制
DataList	类似 Repeater 控件，但具有更多格式和布局选项，其中包括在表格中显示信息的功能，DataList 控件也让您能够限定编辑行为
DropDownList	允许用户选择列表中的选项或输入文本
HyperLink	创建 Web 导航链接
Image	显示图形
ImageButton	与 Button 控件相似，但包含的是图形而非文本
Label	显示用户不能直接编辑的文本
LinkButton	与 Button 控件相似，但显示的是超级链接
ListBox	显示选项列表，允许选择其中的多个选项
Panel	在表单上创建无边界部分，用于存放其他控件
RadioButton	显示一个单选按钮，它可以被选中或不被选中
RadioButtonList	创建一组 RadioButton，仅可从中选择一个按钮
Repeater	使用您指定的 HTML 元素和控件显示数据集中的信息，对于数据集中的每条记录，使用一个该元素
Table	创建表格
TableCell	在表格行中创建单个单元格
TableRow	在表格中创建一行
TextBox	显示设计阶段输入的文本，用户可以在运行阶段编辑修改它 注意，虽然其他控件也允许用户编辑文本（例如 DropDownList），但它们通常不是用于编辑文本

附录 C “ASP.NET 控件：属性和方法”列出了所有 ASP.NET 控件的属性、方法和事件。另外，DataList、DataGrid 和 Repeater 控件将在第 9 章介绍数据绑定时讨论。

### 5.5.1 使用 Web 控件

使用 Web 控件的方法和使用 HTML 服务器控件的方法一样。您在服务器上创建它们，指定其属性和处理其事件的方法。大部分 Web 控件将数据发送给服务器。您已经知道如何使用 Label 和 Button 控件了，这是两个最常见的 Web 控件。

我们来看一个使用某些 Web 控件的例子。清单 5.7 显示了一个典型的用户分析表单，它询问姓名、年龄和收入，但它取得用户的输入数据，将动态地输出一些信息。

**清单 5.7 用 Web 控件分析用户分布情况**

```
1 <%@ Page Language="VB" %>
2
```

```

3 <script runat="server">
4   sub Submit(obj as object, e as EventArgs)
5       dim strIncome as string = lblIncome.SelectedItem.Text
6       dim strAge as string = r1Age.SelectedItem.Text
7
8       lblMessage.Text = "Hello " & tbName.Text & "!<p>" & _
9           "Your income is: " & strIncome & "<br>" & _
10          "Your age is: " & strAge & "<br>"
11
12       if r1Age.SelectedIndex < 3 then
13           lblMessage.Text += "You're a young one!<p>"
14       else
15           lblMessage.Text += "You're a wise one!<p>"
16       end if
17
18       if cbNewsletter.Checked then
19           lblMessage.Text += "You will be receiving our" & _
20              " newsletter shortly."
21       end if
22   end sub
23 </script>
24
25 <html><body>
26   <form runat="server">
27       <asp:Label id="lblHeader" runat="server"
28           Height="25px" Width="100%" BackColor="#ddaa66"
29           ForeColor="white" Font-Bold="true"
30           Text="A Web Controls Example" />
31       <br>
32       <asp:Label id="lblMessage" runat="server" /><p>
33
34       Enter your name:
35       <asp:TextBox id="tbName" runat="server" /><p>
36
37       Choose your age:<br>
38       <asp:RadioButtonList id="r1Age" runat="server"
39           RepeatDirection="horizontal">
40           <asp:ListItem><18</asp:ListItem>
41           <asp:ListItem>19 24</asp:ListItem>
42           <asp:ListItem>25 34</asp:ListItem>

```

```

43      <asp:ListItem>35-49</asp:ListItem>
44      <asp:ListItem>50-65</asp:ListItem>
45  </asp:RadioButtonList><p>
46
47  Choose your income:<br>
48  <asp:ListBox id="lbIncome" runat="server"
49      size="1">
50      <asp:ListItem>< $999/year</asp:ListItem>
51      <asp:ListItem>$1000-$9999</asp:ListItem>
52      <asp:ListItem>$10000-$49999</asp:ListItem>
53      <asp:ListItem>> $50000</asp:ListItem>
54  </asp:ListBox><p>
55
56  Do you want to receive our newsletter?<br>
57  <asp:CheckBox id="cbNewsletter" runat="server"
58      Text="Yes!" /><p>
59
60  <asp:Button id="btSubmit" runat="server"
61      Text="Submit" OnClick="Submit" />
62  </form>
63 </body></html>

```

**分析：**首先来看一下该页面的 HTML 部分。这个例子使用了几个不同的 Web 控件。第 27 - 30 行显示了一个我们熟悉的 Label 控件，但额外指定了一些样式属性。这些样式属性几乎可以用于任何 Web 控件，所以请随意定制显示样式。第 32 行是另一个 Label，用于显示消息。第 35 行是个 TextBox 控件，让用户输入其姓名。第 38 - 45 行是一个 RadioButtonList 控件——一组 Radio 按钮，它包含 5 个由 ListItem 控件表示的选项。第 48 - 54 行是一个与 HTML 元素 select 类似的 ListBox，这个控件包含 4 种不同的选项。第 57 行是一个简单的 CheckBox 控件，而第 59 行是一个大家熟悉的 Button 控件。图 5.9 是在浏览器中查看该清单得到的结果。

每个 Web 控件对应一个已有的 HTML 元素，控件并没有被真正地发送给浏览器。查看该页面的源文件，您将发现其中没什么新东西——对于浏览器来说，HTML 代码足够了。

在代码声明块中，您根据用户的输入执行某些操作。第 5 - 6 行检索从 ListBox 和 RadioButtonList 控件中选中的值。这两个控件都有 SelectedItem 属性，该属性返回被选中的条目。被选中的条目的 Text 属性返回该条目中的文本（如 “<\$999/year,”）。

第 8 - 10 行将用户输入的信息返回，并显示给用户。TextBox 控件 TbName 的 Text 属性取得用户的姓名。

第 12 - 16 行的 if 语句根据用户的年龄显示一条消息。SelectedIndex 属性返回了 RadioButtonList 控件 rdAge 中被选中的条目的索引。例如，第一个条目的索引为 0，最后一个条目的索引为 4。如果索引小于 3，即用户的年龄小于 35，则显示一条消息；否则显示另一条消息。

最后，第 18 行的 if 语句检查 CheckBox 控件的 Checked 属性，以确定复选框是否被选中。如果是，则显示一条消息，提醒用户他将被加入到新闻稿列表中。图 5.10 显示了用户输入一些信息，并

提交了表格后的结果。

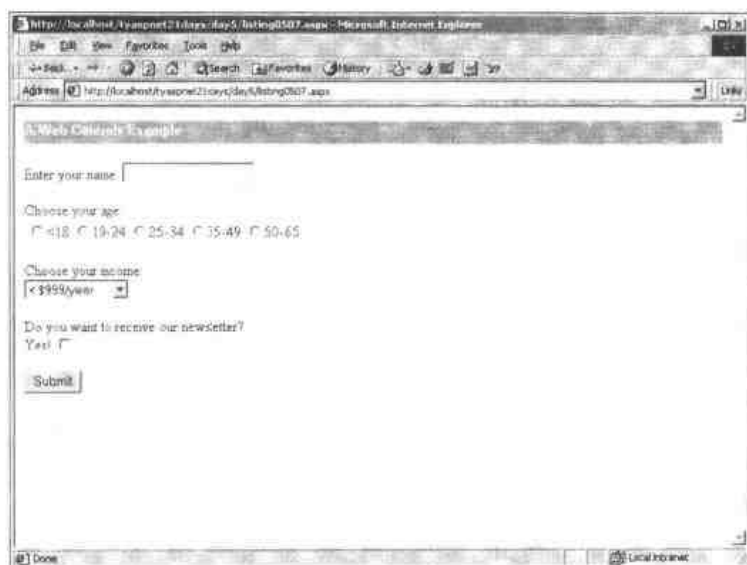


图 5.9 使用 Web 控件分析用户分布情况

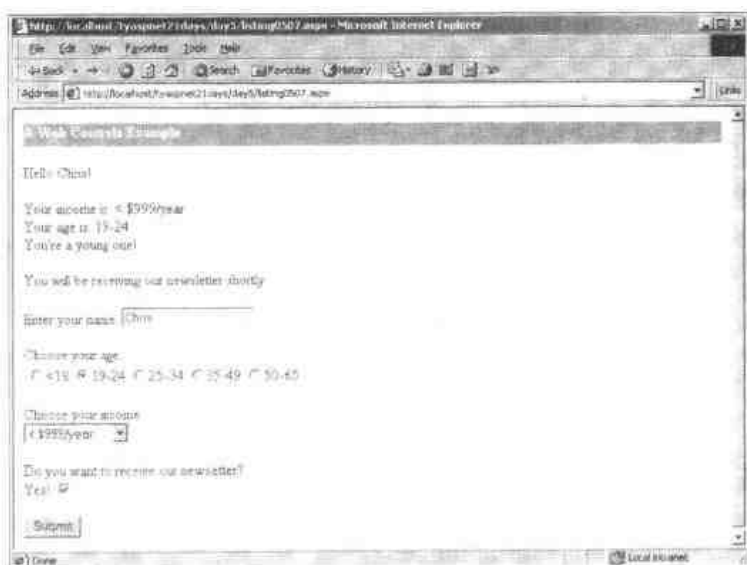


图 5.10 提交后的用户统计表单

**提示：**大多数Web控件都有一组通用的属性。例如，TextBox、Label、LinkButton、Hyperlink、Button和CheckBox控件都有Text属性，该属性返回或设置控件中的文本。ListBox、ComboBox、DataList和DataGrid控件都有SelectedItem属性。这种标准界面使得使用Web控件开发页面变得非常容易。

### 5.5.2 即时发送数据

正如本章前面指出的，您不一定要缓存事件，然后把它们一起发送到服务器。事实上，有时候您的确不想等待。

例如，假设您要创建一个显示某个州所有城市的表单。ListBox 控件提供了城市和州两个列表。一旦某个州被选中后，便发送表单，执行一些处理，然后对城市列表进行过滤，以便只显示属于该

州的城市。在这种情况下，您不想等到用户自己提交表单。

您可以使用 Web 控件的 `AutoPostBack` 属性立即发送数据。将该属性设置为 `true`，告诉 Web 控件，只要有事件发生，就发送数据。我们来看清单 5.8，它依赖两个事件来发送数据。

#### 清单 5.8 使用 `AutoPostBack` 属性

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Drawing" %>
3
4 <script runat="server">
5     sub NameHandler(obj as object, e as EventArgs)
6         lblMessage.Text = tbName.Text & ", select a color: "
7     end sub
8
9     sub ListHandler(obj as object, e as EventArgs)
10        lblColor.BackColor = Color.FromName(obj.SelectedItem.Text)
11    end sub
12 </script>
13
14 <html><body>
15     <form runat="server">
16         <asp:TextBox id="tbName" runat="server"
17             OnTextChanged="NameHandler"
18             AutoPostBack="true" /><p>
19
20         <asp:Label id="lblMessage" runat="server"
21             Text="Select a color: " /><p>
22         <asp:ListBox id="lbColor" runat="server"
23             OnSelectedIndexChanged="ListHandler"
24             AutoPostBack="true" >
25             <asp:ListItem>Red</asp:ListItem>
26             <asp:ListItem>blue</asp:ListItem>
27             <asp:ListItem>green</asp:Listitem>
28             <asp:ListItem>white</asp:Listitem>
29         </asp:ListBox>
30     </form>
31 </body></html>

```

**分析：**第 17 行声明了 `TextBox` Web 控件，当 `TextChanged` 事件发生时，将执行第 5 行的 `NameHandler` 方法。注意，该控件的 `AutoPostBack` 属性被设置为 `true`，这意味着一旦 `TextChanged` 事件发生，便进行发送。

**注意：**仅对您为之指定了句柄的事件，表单才会自动发送。对于前面的 TextBox，仅当 TextChanged 事件发生时，表单才被提交；对于其他事件则不会。

在第 21 行，Label1 控件将一条指令显示给用户。第 23 行的 ListBox 控件有 4 个选项，其 AutoPostBack 属性被设置为 true。用户选择列表中的其他选项时，便发送信息，并执行 ListHandler 方法。

注意，这里没有 Submit 按钮，所以用户不能手工提交表单。

NameHandler 方法读取文本框的值，并将其作为一条自定义消息加到标签中。

ListHandler 方法根据被选中的选项设置 ListBox 的背景颜色。设置颜色时，不能仅使用颜色名称本身（如“red”）或 16 进制的数值（如“#FF0000”），而必须使用 .NET 框架的 Color 对象。

Color.FromName 方法根据提供的颜色名称创建一个 Color 对象，如 Color.FromName(“red”)创建红色对象。然后您便可以使用该对象来设置 ListBox 的颜色（Color 对象属于名称空间 System.Drawing，因此需要使用第 2 行的 Import 编译指令）。图 5.11 显示了用户选择一种颜色并在文本框中输入了数值后的结果。

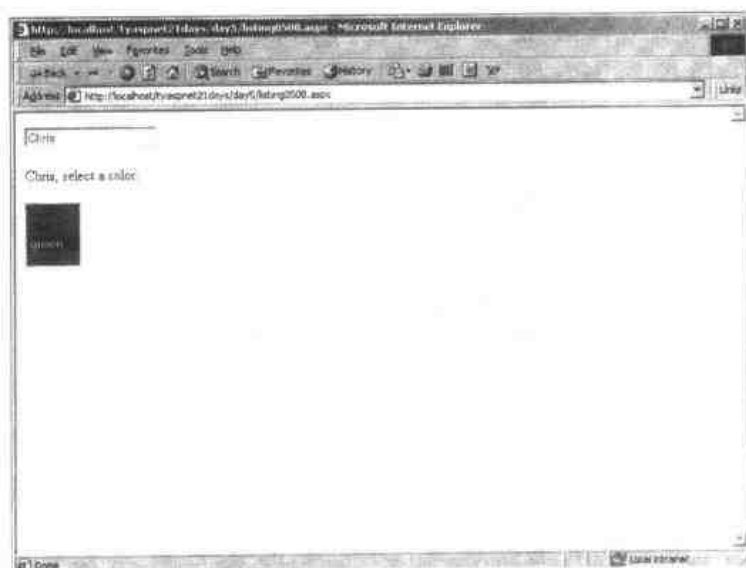


图 5.11 使用 AutoPostBack 提交表单

**注意：**要使该范例运行，必须使用支持级联式样式表的浏览器（IE 或 Netscape 4 以上版本）老版本的浏览器不支持列表框设置颜色的功能。

### 5.5.3 Web 服务器控件与 HTML 服务器控件之比较

许多服务器控件的功能好像是重叠的，例如 HTML 服务器控件 HtmlAnchor 和 Web 控件 HyperLink 的功能相同。在什么情况下应该使用其中一个，而不使用另一个呢？

通常，HTML 服务器控件使得将传统的 HTML 表单转换为 Web 表单变得很容易。只要在 HTML 元素中加上 runat="server"，便可以通过代码访问它。这使得转换已有的页面很容易，并增强了应用程序的功能。

HTML 服务器控件的缺点之一是不提供标准界面。HTML 元素提供许多不同属性和关键词来设置属性，这经常会引起混淆。此外 HTML 服务器控件本质上不提供下限支持，所以 ASP.NET 不能根据不同版本的浏览器自动提供不同的 HTML，这使得自动控制功能降低了，但它允许您手工定制页



面。

从头开始创建页面时应使用 Web 控件。与 HTML 服务器控件相比，它们提供的机制通常更为强大。同时，它们支持标准界面，并提供下限支持。

应 该	不 应 该
不需要通过编程进行操纵时，应使用常规的 HTML 元素。例如，通常不需要操纵链接，所以应该使用标准的 <a href="#">&lt;a href&gt;</a> 标签，而不是 HtmlAnchor 或 HyperLink 控件。	不要创建没必要的服务器控件（即不会通过编程进行方向的控件）。服务器需要创建的对象越多，需要的资源也越多。尽管性能降低的程度不大，但也不值得。
不想花太多的时间来将现有页面转换为 Web 表单时，应使用 HTML 服务器控件。	
创建需要通过编程进行访问的新 Web 表单时，应使用 Web 服务器控件。	
如果有时间，应将现有的元素转换为 Web 服务器控件，而不是 HTML 服务器控件。	

## 5.6 这不是 ASP

传统 ASP 中没有与 Web 表单模型类似的东西，传统 ASP 无法让服务器跟踪控件，也不能创建可以在客户端显示的 UI 控件。开发人员可以创建行为与 Web 表单相似的系统，但要花费大量时间，并需要丰富的经验。

传统的 ASP 依赖请求/响应模型来与用户交互。没有事件驱动模型来在用户界面和应用程序逻辑之间提供一个抽象层。开发人员必须在页面提交时线性地（一次性地）处理它们，而不是对用户事件做出响应。

过去，由于 Web 的无状态性，表单信息一旦提交就会丢失。除非开发人员明确地采取措施来防止这一点（通过检索表单变量）。例如，下面是传统 ASP 的情况：

```
<input type="text" name="tbName"
value="<% -Request.Form("tbName") %>
```

如果开发人员想实现状态不变的错觉，必须不断地重新填写表单元素。ASP.NET 为您自动保存视图状态，所以您不再需要为这种问题操心。

ASP.NET 中的 HTML 服务器控件和 Web 服务器控件是全新的。服务器创建 UI 元素并跟踪它们的状态的能力是革命性的。开发强大应用程序的最好办法是让代码知道 UI 中发生的情况——换句话说，服务器知道客户端的情况。

这些控件可以在每种浏览器中正确地显示，根据不同的版本输出相应的 HTML，这与传统的 ASP 大不相同。过去常常由于浏览器的版本繁多，而需要创建页面的多个版本，这简直是一场噩梦。

无论是在创建 UI 还是应用程序方面，Web 表单框架对 Web 开发人员来说都大有裨益。使用 Web 表单进行开发后，您就会觉得相见恨晚。

## 5.7 总 结

本章涉猎了许多领域。您学习了 ASP.NET Web 表单，学习了 HTML 服务器控件和 Web 控件。在关于 Web 表单的开始章节中就学习了这么多东西已经很不错。阅读本章后，您便能够使用这些控件的丰富功能来创建复杂的用户界面了。

Web 表单给 ASP.NET 页面带来了许多好处。它们使服务器可以跟踪 UI：用户执行了什么操作

以及各个元素都应该做什么。通过巧妙地运用客户端脚本，ASP.NET 页面将关于事件的信息发送给服务器，后者可以相应地处理它们。

Web 表单由四种不同类型的服务器控件组成：HTML 服务器控件、Web 服务器控件、有效性验证控件和用户控件。这些控件包含属性、方法和事件，使开发人员可以控制应用程序。Web 表单还可自动保存每个控件的状态。

只需加上 `runat="server"` 属性，便可以将任何已有的 HTML 元素转换为 HTML 服务器控件。这些控件是在服务器上创建的，然后将适当的 HTML 输出发送给浏览器。使用 HTML 服务器控件让您能够操纵 HTML 元素的属性。

Web 服务器控件比 HTML 服务器控件更复杂，它们经常用来提供更为复杂的用户界面元素。这些控件是在服务器上创建的，并提供了大量的属性和方法供您使用。

下一章将介绍另一种服务器控件：用户控件。这是自定义的控件，可以封装您想要的任何功能。同时，还将介绍更复杂的 Web 表单，如 code-behind 表单。您正在创建复杂 ASP.NET 页面的道路上顺利地前进。

## 5.8 问与答

问：维护视图状态是否会降低性能？有办法关闭这项功能吗？

答：维护视图状态对性能的影响很小。维护视图状态时，ASP.NET 必须额外完成一个步骤。在性能非常重要的情形下，可以用下面的语句关闭维护视图状态的功能：

```
EnableViewState=false
```

您可以为单个控件设置该属性，也可以使用编译指令 `<%Page%>` 设置整个页面。

问：有些操作对 Web 表单无效，是什么错误引起的？

答：别忘了把所有要处理其事件的控件放在 `HtmlForm` 控件中：`<form runat="server">`（这是最常见的错误）。

还要确保在声明事件处理程序时使用正确的语法：

```
<asp:ControlName id="name" runat="server"
    OnXControlName="eventHandler" />
```

最后，别忘了在服务器上，事件的处理并没有特定的次序。您可能无意间创建了阻止另一个事件发生的事件处理程序。

例如在 `Page_Load` 事件中，您可能清除了 `TextBox` 中的文本。这样，处理 `TextBox` 的 `TextChanged` 事件的方法将不能正常运行，因为 `Page_Load` 事件已经清除了文本。在这种情况下，应检查 `IsPostBack` 属性，以确保没有执行不需要或不想执行的代码。

## 5.9 作业

下面的作业帮助巩固本章介绍的概念。答案见附录 A。

### 5.9.1 小测验

1. Web 表单如何跟踪视图状态？
2. 状态包有何作用？如何访问它？

3. 判断正误: Web 表单事件在 Page\_Load 事件之前被处理。
4. 什么是标准事件参数列表?
5. Runat="server" 标记的重要性是什么?
6. 为何要使用 HTML 服务器控件?
7. 以下代码片段错在何处?  

```
<asp: Button id="btSubmit"
    Click="HandleThis" >
```
8. 哪个属性可用于使控件在事件发生后立即向服务器发送信息?
9. 何时应使用 Web 控件而不是 HTML 服务器控件?

### 5.9.2 练习

1. 创建一个选择婴儿名字的应用程序。用户选择婴儿的性别后, ListBox 将显示几个名字。当用户选择了名字后, 将使用被选中的名字来显示一条消息。
2. 创建一个包含如下字段的标用户注册页面: 姓名 (必须填写)、地址、电话 (必须填写)、传真、e-mail 地址 (必须填写)。漏填必须填写的字段时, 向用户显示一条错误消息。

## 第6章

# 再谈 Web 表单

上一章介绍了 Web 表单及其给 Web 开发人员提供的新功能,介绍了 Web 表单背后的机制以及 HTML 服务器控件和 Web 服务器控件。这些都是提高您的编程能力的工具。

本章将继续介绍 Web 表单。您将学习 Web 的扩展功能,学习如何轻松地通过用户控件和自定义控件在框架中加入自己的东西。用户控件让您能够将通用的 UI 功能封装起来,以便不断地重用它们。自定义控件包含您从头开始创建的 UI 功能。结合本章和上一章介绍的内容,您将能够在第一部分结束之前编写出非常棒的 ASP.NET 程序。

本章将介绍以下内容:

- Web 表单的扩展性;
- 什么是用户控件;
- 如何创建和使用用户控件;
- 如何创建和使用自定义控件;
- 如何使自定义控件的行为更接近常规服务器控件;
- 如何在运行阶段在代码中创建服务器控件。

### 6.1 Web 表单的扩展性

上一章介绍过,Web 表单的主要用途之一是让开发人员能够创建复杂的用户界面和相关的逻辑,这包括用自定义机制来扩展 Web 表单框架。

Web 表单提供 HTML 服务器控件和 Web 服务器控件,供创建 UI。每个控件都将某种功能封装到一个易于使用的模块中,并提供了属性、方法和事件,供创建应用程序以及对用户的操作做出响应。上一章介绍了如何使用这些控件创建用户界面,但这只是冰山一角,Web 表单允许您以任何方式使用这些控件,您甚至可以基于它们创建自定义控件!这是一项非常有用的特性。

### 6.2 用户控件

用户控件是 Web 表单的精彩特性,让您能够封装并重用任何 UI 功能。您需要在 Web 应用程序中使用复合控件 Calendar/DropDownList 吗?只需将这两个控件组合起来并封装到一个用户控件中即可。用户控件的工作原理与服务器控件的相同,所以您可以将它们放在任何地方,不用管它们如

何运行。

我们来看一个典型的用户控件。设想您创建了一个网站，它要求用户登录后才能访问某些部分。登录表单相当规范：包含两个文本框（用于输入用户名和密码信息）和一个提交按钮，如图 6.1 所示。您可以使用两个 TextBox 控件和一个 Button 控件，然后加上处理这些控件的方法。但如果要在多个页面上使用该表单，怎么办呢？不用重复创建这三个控件及其事件句柄，而只需使用一行代码便可以创建一个完成相同工作的用户控件。

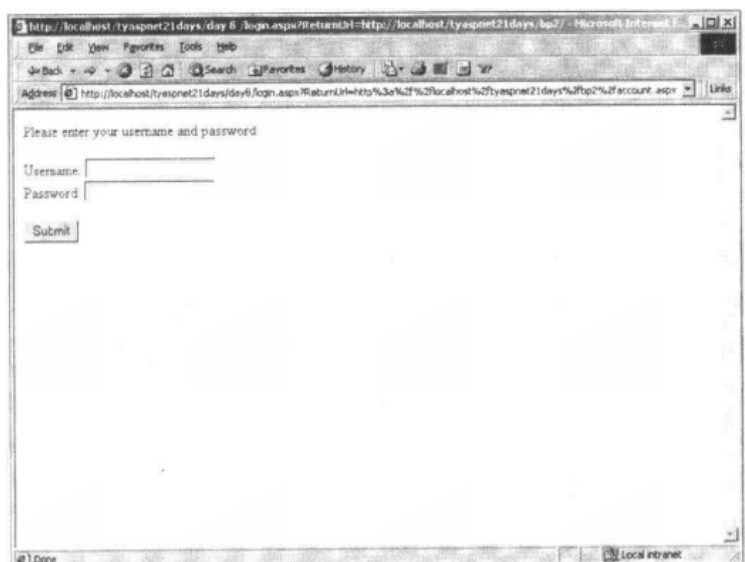


图 6.1 包含两个文本框和一个提交按钮的典型登录表单

一个更复杂的例子是新闻门户网站，它在主页的子元素上显示来自整个网站的新闻。由于这些子元素使用相同格式，却从不同来源采集数据，因此可以实现为一个用户控件。每个用户控件都封装了显示功能，您可以设置每个控件，使之检索不同的信息。事实上，可以使用一个用户控件，并创建该控件的多个实例，就像创建多个文本框控件的实例一样。

当然，用户控件不一定要封装任何实际的功能。您可以创建一个这样的用户控件，即只包含一个带链接的导航条或页面的页眉或页脚。各种可能性都是存在的。

用户控件完全支持 Web 表单框架，所以它们像内置控件一样，可以有属性和事件。另外，在 .NET 框架中，可以使用任何支持 MSIL 的编程语言创建各个用户控件（虽然在每个控件中只能使用一种语言）。

最为重要的是，创建用户控件很容易。只需修改几个地方，便可以把已创建的 ASP.NET 页面转换为用户控件。我们来看一下这是如何完成的。

### 6.2.1 创建用户控件

基本上，用户控件是一个有不同扩展名 .ascx 的 ASP.NET 文件。它包含 UI 元素和控制这些元素的代码。区别在于用户控件将被嵌入到其他的页面中，而不是独立存在的。想一下服务器控件（如 TextBox 控件），这种控件显示一个 HTML 文本框，并提供内置的属性和自己处理的方法。它不能独立存在，而必须用于 ASP.NET 页面。用户控件与此相同。

我们来看一个简单的用户控件，如清单 6.1 所示。这个控件封装了前一节介绍的用户登录表单。

清单 6.1 一个封装在用户控件中的典型的登录表单

```

1 <table style="background-color:<%=BackColor%>;
2   font: 10pt verdana;border-width:1px;border-style:solid;
3   border-color:black; cell-spacing=15>
4 <tr>
5   <td><b>Login: </b></td>
6   <td><ASP:TextBox id="User" runat="server"></td>
7 </tr>
8 <tr>
9   <td><b>Password: </b></td>
10  <td><ASP:TextBox id="Pass" TextMode="Password"
11      runat="server"/></td>
12 </tr>
13 <tr>
14   <td></td>
15   <td><ASP:Button Text="Submit" runat="server"
16      OnClick="Submit"></td>
17 </tr>
18 </table>
19 <p>
20 <ASP:Label id="lblMessage" runat="server"/>

```

**分析：**将该清单保存为 LoginForm.ascx。注意，它与 ASP.NET 页面的 HTML 部分非常相似。它包含两个文本框控件、一个按钮控件、一个标记控件和一个提供格式的表格，但是它没有包含任何<html>、<form>或<body>标记。您还无法知道这个控件是什么样的，因为它还没被放到 ASP.NET 页面中去。

另外，请注意第 1 行的<%@=BackColor%>，它看上去像一个代码交付块。它将表格的背景颜色设置为变量 BackColor 的值，这将在后面做更详细的讨论。现在我们来看看如何在 ASP.NET 中创建该用户控件，如清单 6.2 所示。

清单 6.2 基于该用户控件的 ASP.NET 页面

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4   dim Username as string
5   dim Password as string
6   dim BackColor as string
7
8   sub Submit(obj as object, e as eventargs)
9     Label1.Text = "Username: <b>" & User.Text & _
10        "<br>" & Password: <b>" & Pass.Text & _

```

```

11         <.b><p>"
12     end sub
13 </script>
14
15 <html><body>
16     <form runat="server">
17         <table style="background-color:<%-BackColor%>;
18             font: 10pt verdana;border-width:1;border-style:solid;
19             border-color:black;" cellspacing=15>
20             <tr>
21                 <td><b>Login: </b></td>
22                 <td><ASP:TextBox id="User" runat="server" /></td>
23             </tr>
24             <tr>
25                 <td><b>Password: </b></td>
26                 <td><ASP:TextBox id="Pass" TextMode="Password"
27                     runat="server" /></td>
28             </tr>
29             <tr>
30                 <td></td>
31                 <td><ASP:Button Text="Submit" runat="server"
32                     OnClick="Submit" /></td>
33             </tr>
34         </table>
35         <p>
36         <ASP:Label id="Label1" runat="server" />
37     </form>
38 </body></html>

```

**分析:** 第4—6行声明了变量 Username、Password 和 BackColor，用于存储相应的信息。第8行声明了 Submit 方法，它只是将提供的信息显示在标签中。这个页面的 HTML 部分始于第15行，您应该觉得眼熟，因为它们与清单 6.1 相似。图 6.2 是在浏览器中查看该页面时的结果。

将 ASP.NET 页面转换为用户控件包括三步。第一步是删除所有的<html>、<form>和<body>标记，保留其他 HTML 格式标记。包含用户控件的 ASP.NET 页面将拥有这些标记，所以没有理由在用户控件中再次使用它们。加上这些标记将导致格式错误，甚至导致应用程序崩溃。

其次，您必须修改文件名，使其扩展名为.ascx。例如，listing0602.aspx 可以改为 listing0602.ascx。要求修改扩展名是为了使文件被看作用户控件，而不会被看作一个残缺的 ASP.NET 页面。

最后，如果被转换的页面中包含@Page 编译指令，则必须把它改为@Control。除了追踪外，该指令支持@Page 的所有属性。例如下面的编译指令：

```
<%@ Page Language="VB" %>
```

将改为:

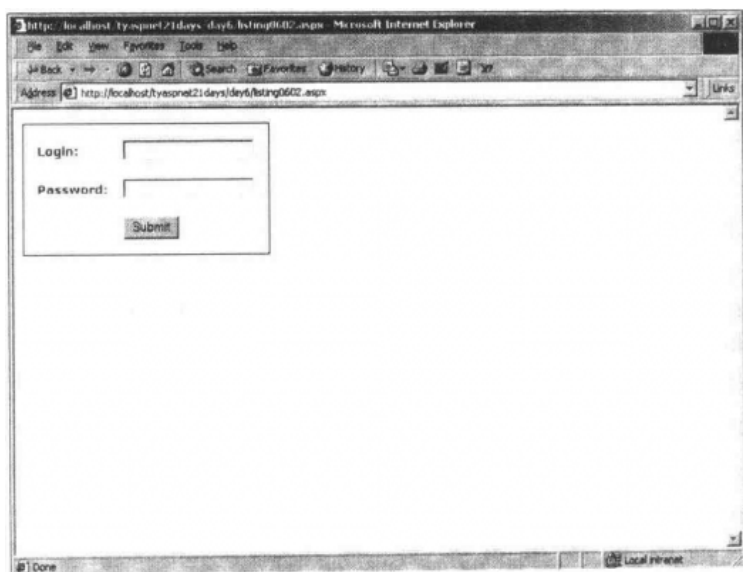


图 6.2 基于用户控件的 ASP.NET 页面

```
<%@ Control Language="VB" %>
```

对清单 6.2 做上述修改后, 将得到清单 6.1 所示的用户控件。这样便可以在任何 ASP.NET 页面中使用该控件了。

**注意:** 不一定非要用已有的 ASP.NET 页面来创建用户控件, 这里这么做是为了突出其不同的地方。按照上述指导方针, 您可以轻松地从头开始创建一个用户控件。

至此, 这个用户控件封装了登录表单的 UI 元素。如果您打算在 ASP.NET 页面中实现该模块, 则这可以节省时间和精力。但用户控件可以包含更多的功能, 它们也可以包含控制 UI 元素的代码。我们来创建一个代码声明块, 显示该控件的属性、事件和方法。清单 6.3 显示了清单 6.2 中的 Username、Password 和 BackColor 属性, 并定义了一个 Submit 方法来处理按钮的 OnClick 事件。

#### 清单 6.3 用户控件的代码声明块

```
1 <script language="VB" runat="server">
2   public BackColor as String = "White"
3   public UserName as string
4   public Password as string
5
6   public sub Submit(obj as object, e as eventargs)
7       Label1.Text = "Username: <b>" & User.Text & "</b><br>" & _
8           "Password: <b>" & Pass.Text & "</b><p>"
9   end sub
10 </script>
11
12 <table style="background-color:<% BackColor %>
13   font: 10pt verdana;border-width:1;
```



```

14 border-style:solid;border color:black;
15 cellpadding=15>
16 <tr>
17     <td><b>Login: </b></td>
18     <td><ASP:TextBox id='User' runat='server' /></td>
19 </tr>
20 <tr>
21     <td><b>Password: </b></td>
22     <td><ASP:TextBox id='Pass' textMode="Password"
23         runat="server" /></td>
24 </tr>
25 <tr>
26     <td></td>
27     <td><ASP:Button Text="Submit" runat="server"
28         OnClick="Submit" /></td>
29 </tr>
30 </table>
31 <p>
32 <ASP:Label id='Label1' runat="server" />

```

**分析：**用该清单中的代码替换前面的 LoginForm.aspx 文件，增加的内容只是第 1-10 行的代码声明块。它大致与清单 6.2 相似，仅有少量的改动。第 2 行的 BackColor 属性开始被设置为“white”，还定义了 UserName 和 Password 属性，供用户操作。注意关键词 public，它使其他 ASP.NET 页面可以使用这些属性来设置或读取 UI 中的值。

Submit 方法与清单 6.2 完全相同。由于用户控件与 ASP.NET 页面非常相似，因此只需复制并粘贴这些代码。使用扩展名.aspx 保存该文件后，便可以在 ASP.NET 中使用它。

## 6.2.2 使用用户控件

在 ASP.NET 页面中使用用户控件的方法与使用其他服务器控件的方法相同。在页面的 UI 部分创建其实例，并在代码中控制它。但由于这些控件是自定义的，因此需要使用一些额外的语法。要将用户控件放置在其他页面中，必须首先在该页面中使用页面编译指令@Register 注册该控件。注册控件相当于告诉 ASP.NET，您想使用一个新的服务器控件来扩展 Web 表单。这样，页面就知道该控件的有效路径和规定的名称了。我们来看一下该编译指令的语法：

```

<%@ Register TagPrefix="Prefix" TagName="ControlName"
    src="filepath" Namespace="name" %>

```

TagPrefix 属性定义了该控件所属的组。上一章介绍的所有 Web 服务器控件的 TagPrefix 属性都为 asp。例如：

```
<asp:Button id="btOne" runat="server" />
```

您可以根据需要将用户控件的 TagPrefix 属性设置为任何值，只需在@Register 编译指令指定即可。TagName 属性给用户控件指定名称，供页面引用。例如，登录表单用户控件可以使用名称 LoginForm。Src 属性指定用户控件要使用的资源的位置，页面必须知道该位置，才能知道用户控件

的功能。最后，Namespace 属性是一个可选元素，它指定了与 TagPrefix 相关的名称空间。这有助于进一步将用户控件分组和分类。

我们创建一个 ASP.NET 页面来使用新的登录表单用户控件。清单 6.4 显示了一个 ASP.NET 页面，它包含一个 @Register 编译指令，并创建了该用户控件的一个实例。

清单 6.4 将用户控件嵌入到 ASP.NET 中很容易

```

1  %& Page Language= 'VB' %>
2  <%@ Register TagPrefix="TYASPNET" TagName="LoginForm" src= .
3  'LoginForm.ascx' %>
4
5  <script runat="server">
6      sub Page_Load(obj as object, e as EventArgs)
7          lblMessage.Text = "Properties of the user control: " & _
8              "<br>id: " & LoginForm1.id & "<br>" & _
9              "BackColor: " & LoginForm1.BackColor & "<br>" & _
10             "Username: " & LoginForm1.Username & "<br>" & _
11             "Password: " & LoginForm1.Password
12      end sub
13 </script>
14
15 <html><body>
16     <form runat="server">
17         <TYASPNET:LoginForm id="LoginForm1" runat="server"
18             Password= MyPassword'
19             Username="Chris"
20             BackColor="white" />
21     </form>
22     <p>
23     <asp:Label id="lblMessage" runat="server" />
24 </body></html>

```

**分析：**该页面有几个有趣的特性。首先，请注意第 2 行的编译指令 @Register。您将前缀设置为 TYASPNET、控件名设置为 LoginForm。这样便可以在其他任何地方使用该名称来引用它，如第 17 行所示：

```
<TYASPNET:LoginForm>
```

Page\_Load 事件将用户控件的属性显示在第 23 行的标签中。最后，在第 17-20 行实现了该控件其语法与其他所有服务器控件的完全相同。runat="server" 同样是必不可少的。您在第 18-20 行设置的属性是清单 6.3 中第 2-4 行创建的公有变量。

另外，请注意第 23 行的标签 ID: lblMessage，这与您在用户控件中用于该标签的 ID 相同。通常以同一个页面中给两个控件指定相同的 ID 将导致错误，但该清单看起来却运行良好，原因是 ASP.NET 并不关心用户控件内部使用了什么控件以及它们的 ID 是什么，而只关心用户控件提供的

UI 功能将由该控件进行处理，而不管其细节。用户控件的公有属性也被显示在该页面中。同样，ASP.NET 不关心您如何使用它，而只是知道这些属性是可用的，其他的工作由用户控件负责。

因为 ASP.NET 不关心细节，因此无法在 ASP.NET 页面中设置用户控件内部的属性。例如，在清单 6.4 中加入下述代码将出错：

```
User.Text = "Clpayne"  
Pass.Text = "HelloWorld!"
```

User 和 Pass 引用的是用户控件内部的控件，而 ASP.NET 不知道也不关心它们。因而，您需要提供用于处理用户控件内部 UI 元素的代码。

图 6.3 是在浏览器查看该页面时得到的结果。

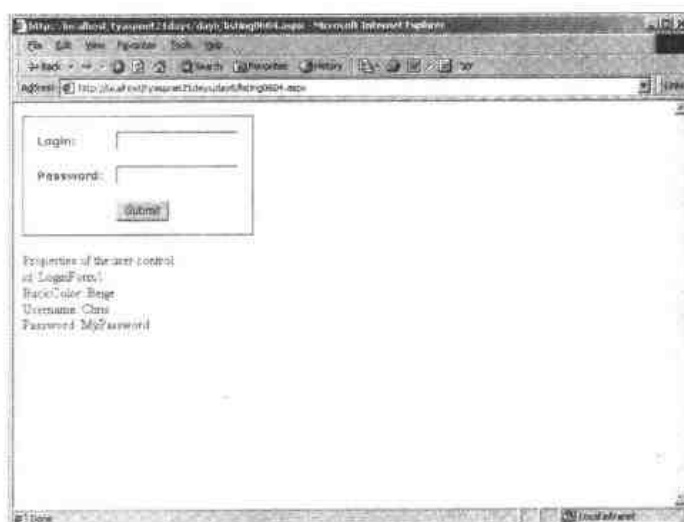


图 6.3 提供 UI 的登录表单用户控件

在登录表单中输入一些文本，然后单击 Submit 按钮，表单将被发送，即使清单 6.4 没有包含完成该工作的代码。这是由于该项功能被放置在用户控件中。

单击 Submit 按钮后，用户将看到如图 6.4 所示的结果。

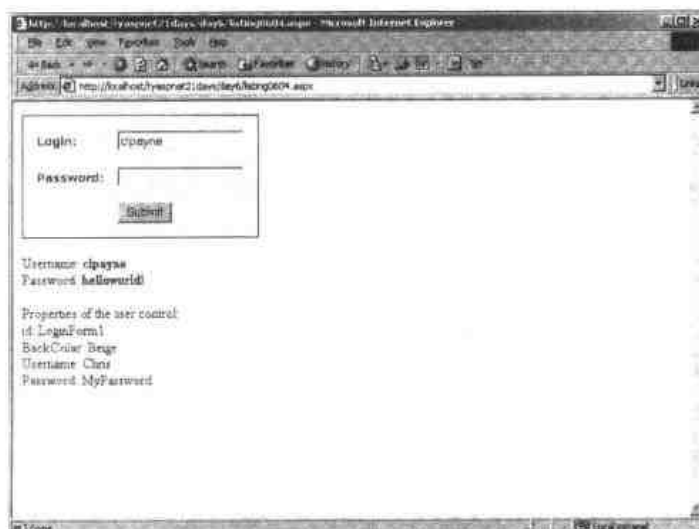


图 6.4 用户控件处理自己的表单提交

### 6.2.3 改进用户控件

到目前为止，我们仍未使用用户控件的 Username 和 Password 属性。当设置这些属性后，将预先使用这些值填写登录表单。不能像 BackColor 属性那样使用代码交付块。它们不允许出现在服务器控件标记中，否则将出错；而必须将公有变量 Username 和 Password 转换为实际的 VB.NET 属性元素。请使用清单 6.5 中的代码替换 LoginForm.aspx 中的第 3–4 行。

清单 6.5 使用 VB.NET 属性而不是公有变量

```

1:  public property UserName as string
2:
3:      Get
4:
5:          UserName = User.Text
6:
7:      End Get
8:
9:      Set
10:
11:         User.Text = value
12:
13:     End Set
14:
15: end property
16:
17: public property Password as string
18:
19:     Get
20:
21:         Password = Pass.Text
22:
23:     End Get
24:
25:     Set
26:
27:         Pass.Text = value
28:
29:     End Set
30:
31: end property

```

**分析：**这是一种新的 VB.NET 语法，相当于将公有变量转换为用户控件的属性。它们的区别在于，公有变量提供的功能很有限，您只能指定和检索值。用户控件的属性让您能够在属性被设置或存取时执行某种操作。

第 3 行与您前面看到的很相似：public UserName as string 再加上关键字 property。使用关键字 property 时，必须有对应的 end property，如第 10 行所示。在属性标记内，有另外两个元素：Get 和 Set。这些元素包含用于检索属性或给属性赋值的代码。例如，当开发人员编写下面的代码行时，实际上是调用 Get 语句：

```
dim strName as string = LoginForm1.UserName
```

而 set 语句是以下面的方式调用的：

```
LoginForm1.UserName="Chris"
```

在这些元素内部，您可以加入代码，以实现属性被存取时要实现的功能。当用户检索数据时将执行第 5 行，它通过将属性设置为 User 文本框控件中的值，来返回这个值。用户设置属性时将执行第 8 行，它将输入的值赋给 User 文本框控件。Value 关键字引用用户赋给该属性的值。例如，在下面的代码行中，value 指的是“Chris”：

```
LoginForm1.UserName="Chris"
```

**提示：**如果要创建只读或只写的属性，只需留下Set或Get元素，并加入关键词ReadOnly或WriteOnly。例如下面的代码段创建了一个只读的Username属性：

```
public ReadOnly property UserName as String
    Get
        UserName = User.Text;
    End Get
End property
```

第 12~19 行为 Password 的属性和 Pass 文本框控件提供了完全相同的功能。现在再在浏览器中查看清单 6.4，您将看到如图 6.5 所示的结果。

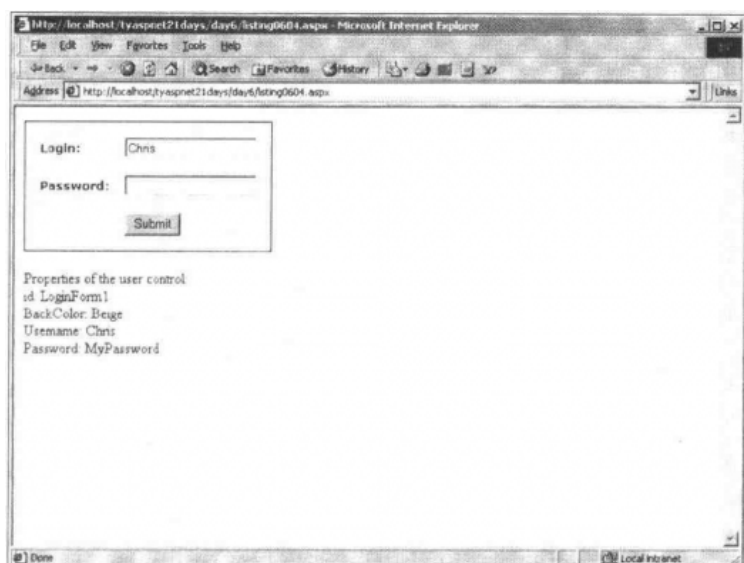


图 6.5 使用属性预先填写用户控件元素

由于您在清单 6.4 的第 19 行设置了相应的属性，因此用户名字段将被预先填写。注意密码字段还是空的，这是因为这个文本框控件的 TextMode 属性被设置为 Password，出于安全方面的考虑，ASP.NET 不会预先填写该文本框。但 Password 属性还是被指定了值，如用户控件下面的标签所示。如果删除该文本框控件中的代码 TextMode="Password"，该值将出现在该控件中。

现在该用户控件已经有了许多功能，包含可以从 ASP.NET 页面操纵的属性和处理自己的事件的方法。该控件可以放在任何 ASP.NET 页面中，其功能将与上面介绍的相同。

## 6.3 自定义控件

ASP.NET 框架还允许创建自定义控件。这是一个高级话题，但是因为您已经熟悉了 Web 表单框架，因此学习自定义控件将得心应手。

自定义控件不是用户控件。自定义控件不是封装一些预建的 UI 功能，而是定义自己的、完全原创的行为。例如，ASP.NET 不提供内置的交付行技术（render line art），开发人员可以通过建立自定义控件来提供这项功能，并可以很轻松地重用它或将其放置在任何 ASP.NET 应用程序中。

当您想把已有的控件功能组合在一起时，应使用用户控件；而当现有的控件无法满足您的要求

时,则需要使用自定义控件。Web 表单框架允许您使用自己的控件来扩展控件库。自定义控件可以是完全新建的,也可以是现有控件行为的扩展。从这个意义上说,它有助于更详细地学习 Web 表单框架。图 6.6 说明了 Web 表单框架中不同概念之间的关系。

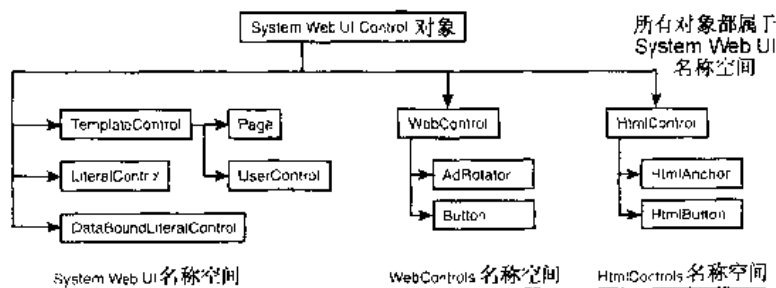


图 6.6 Web 表单框架中的对象层次结构

Web 表单框架中所有对象都是直接或间接从 System.Web.UI.Control 类派生而来的。这个类为所有的控件提供通用的基本方法和属性,例如 ID、GetType 和 ToString。Control 对象之下的各个对象都提供了额外的功能。

Web 表单框架允许您将自定义控件插入到该层次结构的任何位置,只要该自定义控件是从 Control 类派生而来的。您可以扩展现有功能,或干脆创建新的机制,将其置于图 6.6 所示的某个名称空间下。通常您从 WebControl 或 HtmlControl 类派生出自定义控件,因为它们在显示用户界面方面提供的预建功能最多。

### 6.3.1 创建自定义控件

我们来创建一个自定义控件。清单 6.6 显示了一个从 Control 派生而来的类,它向用户显示一条简单的消息。

清单 6.6 自定义控件只不过是一个 VB.NET 类而已

```

1 Imports System
2 Imports System.Web
3 Imports System.Web.UI
4
5 Namespace MyCustomControls
6     Public Class CustomControl1 : Inherits Control
7         Protected Overrides Sub Render(ByVal Output As HtmlTextWriter)
8             Output.Write(ToString & " The time is now: " & _
9                 DateTime.Now.ToString)
10        End Sub
11    End Class
12 End Namespace
  
```

**分析:** 将该文件保存为 CustomControl1.vb。注意,这是一个 VB.NET 源文件,而不是 ASP.NET 页面。因此,您必须完成额外的工作,如导入名称空间;而在.aspx 文件中,名称控件是自动导入的。

第 1-3 行导入了这些名称空间，Imports 关键词类似于 ASP.NET 页面中的 @Import 编译指令

接下来，第 5 行声明了该控件所属的名称空间，这可以是任何名称空间，只是别忘了加上 End Namespace，如第 11 行所示。第 6 行创建了一个从 Control 类派生而来的类（关于创建类的更详细的信息，请参阅第 3 章）。

第 7-9 行包含用户控件最重要的部分之一。Render 是允许现有控件在浏览器中显示 HTML 的方法，所有服务器控件都从 Control 类那里继承了这个方法。为使自定义控件能够在浏览器中显示 HTML，必须实现该方法。然而，因为该方法在 Controls 类中已经存在，因此必须用 Overrides 关键字来覆盖它（内置服务器控件也是这样做的）。不要忘记关键词 Protected！

Render 方法接受了一个参数：一个名为 Output 的 HtmlTextWriter 对象。该对象提供了帮助 Render 方法生成 HTML 输出的所有方法。第 8 行使用变量 Output 的 Write 方法将 HTML 发送给浏览器，这里是当前时间以及一条欢迎消息。

最后，为了能够在另一个 ASP.NET 页面中使用自定义控件，必须编译这些代码——ASP.NET 不能直接使用 VB.NET 源文件。打开命令提示窗口，切换到 CustomControl1.vb 所在的目录，并输入下面的命令：

```
vtx /r:library /out:..\bin\CustomControls.dll /r:System.dll
/r:System.Web.dll CustomControl1.vb
```

这将把源文件 CustomControl1.vb 编译为一个名为 CustomControls.dll（由 /out 参数指定）的 MSIL 文件中，供 ASP.NET 使用。您应该见到如下的输出：

```
Microsoft (R) Visual Basic .NET Compiler version 7.00.9114
for Microsoft (R) .NET CLR version 1.00.2523
Copyright (C) Microsoft Corp 2000. All rights reserved.
```

关于名字空间、类、编译命令、bin 目录和创建对象的更详细的信息，请参阅第 15 章。

### 6.3.2 使用自定义控件

创建并编译自定义控件后，便可以在任何页面中实现它，就像用户控件一样。同样，必须要使用 @Register 编译指令，但语法稍微有些不同：

```
<%@ Register TagPrefix="ACML" Namespace="MyCustomControls"
Assembly="CustomControls"%>
```

TagPrefix 属性与用户控件一样。但 TagName 和 src 已被 Namespace 和 Assembly 属性取代。Namespace 是在清单 6.6 的第 5 行创建的自定义控件的名字空间：MyCustomControls。Assembly 属性指定了编译后的源文件：CustomControls（注意不包含扩展名.dll，使用 Assembly 元素时，ASP.NET 认为扩展名为.dll）。然后，您就可以在 ASP.NET 页面中实现该控件，就像实现其他控件一样。清单 6.7 是一个使用该控件的简单例子。

#### 清单 6.7 实现自定义控件和实现用户（或服务器）控件完全相同

```
1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ACML" Namespace="MyCustomControls" _
   Assembly="CustomControls"%>
3
4 <html><body>
5   <form runat="server">
```

```

6      The custom control produces the following output:<p>
7
8      <ACME:CustomControl1 id='MyControl' runat=server
          Message="Hello world!" />
9  </form>
10 <.body:</html>

```

在浏览器中查看该清单得到的结果如图 6.7 所示。

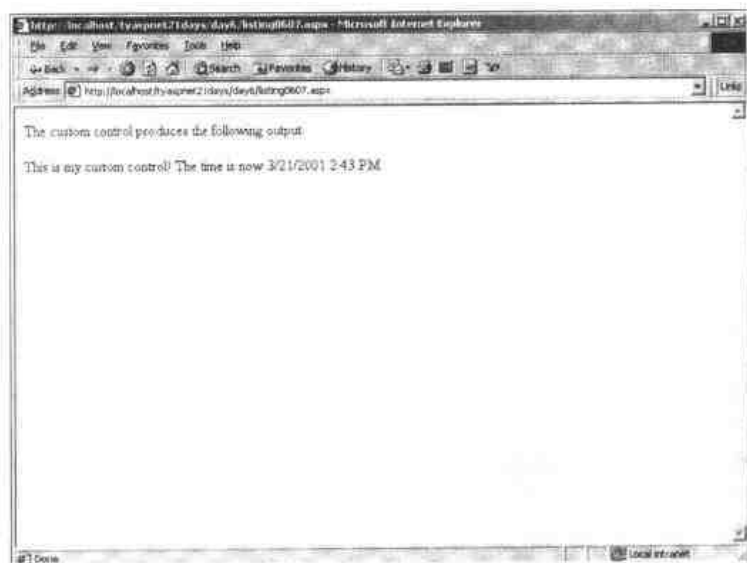


图 6.7 自定义控件输出 HtmlTextWriter 对象的 Write 方法指定 HTML

### 6.3.3 使用属性和状态

到目前为止,我们只介绍了使用自定义控件将 HTML 写入到浏览器中。像其他服务器控件一样,自定义控件也可以包括属性、方法和事件,可参与表单提交,甚至可以维护状态。

要给自定义控件加上属性,只需在自定义控件类中声明一个公有变量或 VB.NET 属性元素。例如,可以将清单 6.6 中的类修改成清单 6.8 那样。

#### 清单 6.8 给自定义控件加上属性

```

5 Namespace MyCustomControls
6     private Class CustomControl1 : Inherits Control
7         private strMessage as string = "This is my custom control!"
8         public property Message as string
9             Get
10                 Message = strMessage
11             End Get
12             Set
13                 strMessage = value
14             End Set
15         end property

```



```

16
17     Protected Overrides Sub Render(Output as HtmlTextWriter)
18         Output.Write(strMessage & " The time is now " & _
19             DateTime.Now.ToString)
20     End Sub
21 End Class

```

**分析：**第 7 行创建了一个私有变量 `strMessage`，它保存了要显示给用户的文本。第 8-15 行创建了一个公有属性，用于设置和检索 `strMessage` 的值（关于创建属性的信息，请参见本章前面的“改进用户控件”一节）。在 `Render` 方法中，您修改代码，以便显示包含在 `strMessage` 中的文本，而不是显示原来的静态文本。

在命令提示符下使用下面的命令重新编译该自定义控件：

```
vbc /r:library /out:..\bin\CustomControls.dll /r:System.dll
```

```
➡ /r:System.Web.dll CustomControl1.vb
```

修改清单 6.7，以使用新的 `Message` 属性：

```
<ACME:CustomControl1 id="MyControl" runat=server
    Message="Hello world!" />
```

清单 6.7 将显示新的消息，如图 6.8 所示。

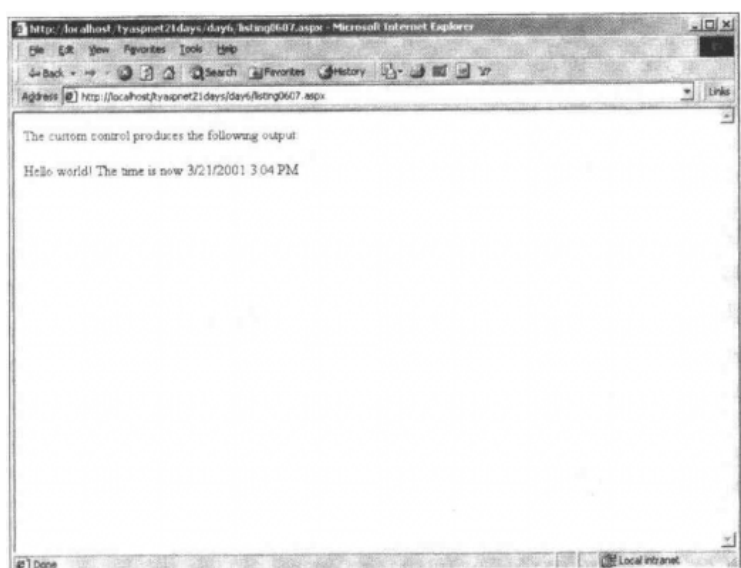


图 6.8 给自定义控件加上属性以提高其功能

如果希望该控件维护状态，只需将该项目加入到状态包中即可（参见第 4 章）。清单 6.9 是一个新的自定义控件，它实现了状态管理功能。

#### 清单 6.9 创建具备状态管理功能的自定义控件

```

1 Imports System
2 Imports System.Web
3 Imports System.Web.UI
4

```

```

5 Namespace MyCustomControls
6     Public Class CustomControl2 : Inherits Control
7         public property Message as string
8         Get
9             Message = ViewState("Message").ToString
10        End Get
11        Set
12            ViewState("Message") = value
13        End Set
14    end property
15
16    public property Size as integer
17    Get
18        Size = CType(ViewState("Size"), Integer)
19    End Get
20    Set
21        ViewState("Size") = value
22    End Set
23 end property
24
25 Protected Overrides Sub Render (Output as HtmlTextWriter)
26     Output.Write( <font size=" & Me.Size & "> &
27         Me.Message & "</font>" )
28 End Sub
29 End Class
30
31 End Namespace

```

**分析：**将该文件保存为 CustomControl2.vb。该控件向用户显示一条由 Message 属性指定的简单消息。它与 CustomControl1.vb 很相似，但现在 Message 属性保存在状态中。在第 12 行的 Set 元素中，您将提供的值保存到状态包中名为 Message 的变量中，使之永久化；而不是像清单 6.8 那样将其保存到一个私有变量中。第 9 行的 Get 语句则从状态包中检索这个值。别忘了使用 ToString 方法将从状态中返回的值转换为字符串，因为在默认情况下，ASP.NET 将它作为一个对象返回。

您添加了一个新的属性：Size，它指定了显示消息时使用的字号。该属性元素的工作原理与 Message 属性相似，只不过它是一个整数。同样，Get 和 Set 元素设置和检索状态包中的值。第 18 行的 CType 方法将从状态包返回的数据转换为整数。

使用下面的命令编译该文件：

```

vbc /t:library /out:..\bin\CustomControls.dll /r:System.dll
  /r:System.Web.dll CustomControl2.vb

```

清单 6.10 是一个使用该新控件的 ASP.NET 页面。

## 清单 6.10 在 ASP.NET 页面中实现可维护状态的自定义控件

```

1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ACME" Namespace="MyCustomControls"
   Assembly="CustomControls" %>
3
4 <script runat="server">
5     sub Submit(obj as object, e as EventArgs)
6         MyControl.Size = MyControl.Size + 1
7     end sub
8 </script>
9
10 <html><body>
11     <form runat="server">
12         The custom control produces the following output:<p>
13
14         <ACME:CustomControl2 id="MyControl" runat="server"
15             Message="Hello world!"
16             Size= />
17
18         <asp:Button runat="server"
19             Text="Increase size!"
20             OnClick="Submit"/>
21     </form>
22 </body></html>

```

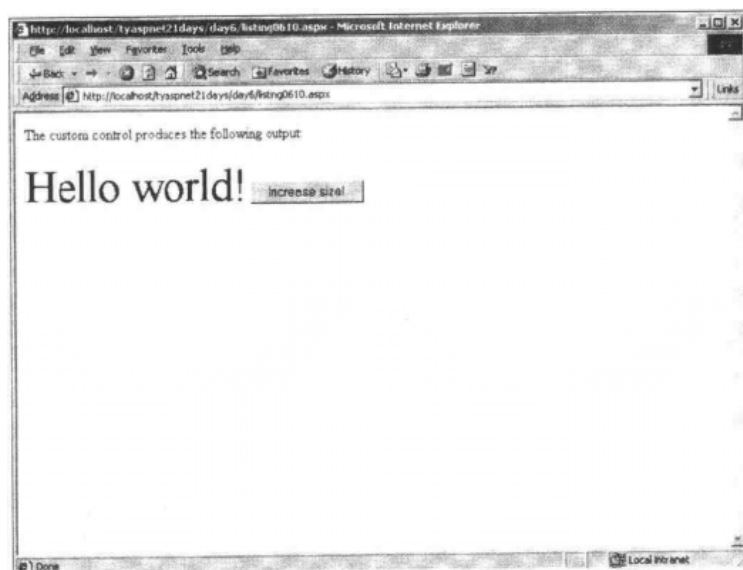


图 6.9 字号越来越大表明状态被保存

分析：自定义控件是在第 14 行实现的，并给 Message 和 Size 属性指定了值。当用户单击第 15 行所示的 Submit 按钮提交表单时，Submit 方法（第 5 行）使用自定义控件的 Size 属性将消息的字号加 1。

注意，仅当自定义控件的状态管理功能被启用时才会发生这种情况。ASP.NET 必须知道发送表单之前的字号，才能将字号加 1。如果状态未保存，MyControl.Size 将总是返回 1——第 16 行设置的值，但如果每次提交后，字号都加 1，则表明控件的状态被保存了。图 6.9 显示了单击按钮多次后的情况。

### 6.3.4 加入事件

服务器控件最重要的方面之一是对用户的操作做出响应。在自定义控件中加入这项功能并不难，但需要知道被客户请求时，服务器控件是如何运作的。所有的服务器控件都按下面的次序执行。注意，这里并没有列出所有的步骤，而只列出最重要的步骤：

1. 加载状态信息；
2. 处理回送数据：控件检验所有来自表单发送的数据；
3. 加载控件，类似于 Page\_Load 事件；
4. 如果表单数据发生了变化，引发事件，以便对其进行处理；
5. 处理导致回送的事件；
6. 保存控件状态；
7. 将输出交付给浏览器。

注意，处理与回送数据相关的事件时，有两个独立的步骤。首先，必须检测数据的变化情况，如第 2 步所述。然后要执行对变化进行处理的方法，如第 4 步所示。该两步过程直接转到必须完成的方法，以便自定义控件可以处理事件。您首先决定是否引发某事件，然后调用一个方法来引发它，实现该控件的 ASP.NET 页面便可以处理该事件了。图 6.10 说明了这种流程。

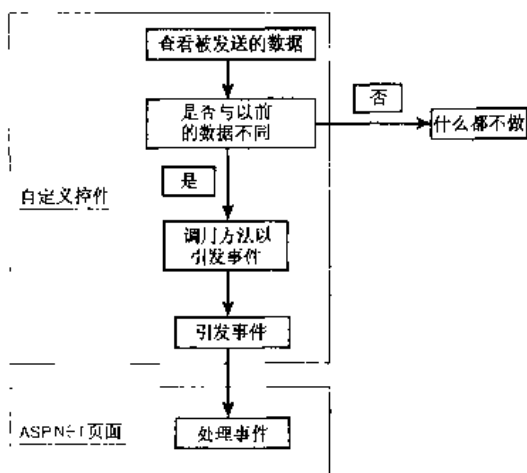


图 6.10 在自定义控件中处理事件的步骤

要为自定义控件创建事件，首先必须确定要处理哪些事件。您要“对文本框内容的变化还是按钮被单击做出响应”？事件生成的数据类型、对如何在控件中实现该功能有重要的影响。您只需实现第 2 步和第 4 步即可。幸运的是这项工作的很多内容是现成的，您只需编写少量的代码。我们来看一

一个新的、包含事件的自定义控件，如清单 6.11 所示。

**清单 6.11 一个事件可处理自定义控件**

```

1 Imports System
2 Imports System.Web
3 Imports System.Web.UI
4 Imports System.Collections.Specialized
5
6 Namespace MyCustomControls
7     Public Class CustomControl3 : Inherits Control : _
8         Implements IPostBackDataHandler
9         public Event TextChanged(obj as object, e as EventArgs)
10        protected sub OnTextChanged(e as EventArgs)
11            RaiseEvent TextChanged(Me, e)
12        end sub
13
14        Public Function LoadPostData(PostDataKey As String, _
15            Values As NameValueCollection) As Boolean Implements _
16            IPostBackDataHandler.LoadPostData
17            dim strOldValue as String = Me.Message
18            dim strNewValue as String = Values(postDataKey)
19            if not strOldValue = strNewValue
20                Me.Message = strNewValue
21                return true
22            end if
23            return false
24        End Function
25
26        Public Sub RaisePostDataChangedEvent() Implements _
27            IPostBackDataHandler.RaisePostDataChangedEvent
28            OnTextChanged,EventArgs.Empty)
29        end sub
30
31        public property Message as string
32        Get
33            Message = ViewState("Message").ToString
34        End Get
35        Set
36            ViewState("Message") = value
37        End Set

```

```

34     end property
35
36     Protected Overrides Sub Render(Output as HtmlTextWriter)
37         Output.Writer.WriteLine("<input name=' " & Me.UniqueID & _
38             " type='text' value=' " & Me.Message & " '">")
39     End Sub
40 End Class
41
42 End Namespace

```

分析：请将该文件保存为 CustomControl3.vb。这里好像有许多复杂的功能，但别着急！对其进行分析后，您就会发现它实际上很简单。首先要注意的是第 7 行的 Implements IPostBackDataHandler。IPostBackDataHandler 是一个定义控件如何与回送信息交互的接口，为您编写代码提供了蓝图。它是一个标准项，如果要处理事件，则是必不可少的，所以不用过多担心它。知道其语法后，其他的就很简单了。

紧接着，第 8 行声明了一个事件。这与声明变量相似，只是必须使用关键词 event。这里要处理自定义控件中的文本发生变化的情况，因此将该事件命名为 TextChanged。注意，这里也需要提供标准事件参数列表。

第 9-11 行声明了一个子程序——OnTextChanged，它引发 TextChanged 事件。还记得吗？在 ASP.NET 页面中，您使用 OnEventName 为控件指定事件句柄（如 OnTextChanged=“HandleChange”）。第 9-11 行定义了 On 事件。第 10 行调用 RaiseEvent 方法，后者引发事件 TextChanged，并带有指定的参数。同样，这些代码也是标准的。为控件定制这些代码时，需要做的只是修改事件的名称。

接下来是方法 LoadPostData，它对应于控件执行次序的第 2 步。该方法有两个参数，第一个指定了其数据将被检验的控件的名称（这里总是自定义控件），第二个指定了用户提交表单时发送的数据。如果希望该事件被处理，则 LoadPostData 方法应返回 true；否则返回 false。这个方法的声明相当长，不过是标准的。

该方法的第 14-20 行查看发送的数据。第 14 行访问控件的 Message 属性，取得控件原来的数值。然后您通过访问 LoadPostData 方法的第二个参数取得新的值。第 16 行对这两个值进行比较，以确定数据是否发生了变化。如果是，则返回 true，告诉 ASP.NET 必须执行一个方法来处理这一事件；否则返回 false。

最后，第 23 行的 RaisePostDataChangedEvent 调用方法来处理事件。该声明也很长，不过也是标准的。它只是调用 OnTextChanged 方法，该方法引发 TextChanged 事件，这样便可以在 ASP.NET 页面中处理该事件。

该清单的其他部分和清单 6.9 几乎完全相同。唯一的区别在于，现在必须为将引发事件的控件提供一个 name 属性，如第 37 行所示。使用 UniqueID 属性可以返回自定义控件的名称。没有这一步，控件将无法引发任何事件。

另外，别忘了导入名称空间 System.Collections.Specialized（如第 4 行所示），并将类的名称改为 CustomControl3，如第 7 行所示。

我们简要地重述一下：

- 首先用关键词 event 创建一个事件，然后用 OnEvent 语法创建一个方法，该方法使用 RaiseEvent 方法引发事件。

- 然后，用标准的 LoadPostData 方法检测数据是否发生了变化以及是否要引发事件。
  - RaisePostDataChangedEvent 方法负责调用另一个方法来引发在第 1 步中定义的事件
- 这里使用的大部分语法都是标准的，因此实现步骤也是标准的

编写好自定义控件后，用下面的命令编译它：

```
vb /t:library /out:..bin/CustomControls.dll /r:System.dll
➤ r:System.Web.dll CustomControls.vb
```

清单 6.12 是一个实现该自定义控件的 ASP.NET 页面。

**清单 6.12 在 ASP.NET 页面中处理自定义控件的事件**

```
1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ACME" Namespace="MyCustomControls" _
   Assembly="CustomControls"%>
3
4 <script runat="server">
5     sub Submit(obj as object, e as eventargs)
6         do nothing
7     end sub
8
9     sub ChangeIt(obj as object, e as eventargs)
10        Response.write("Event handled!")
11    end sub
12 </script>
13
14 <html><body>
15     <form runat="server">
16         the custom control produces the following output:<p>
17         <ACME:CustomControl id="MyControl" runat="server"
18             Message="Hello world!"
19             OnTextChanged="ChangeIt" />
20         <asp:Button runat="server"
21             Text="Submit"
22             OnClick="Submit" />
23     </form>
24 </body></html>
```

**分析：**现在可以轻松一下了！该页面和其他 ASP.NET 页面一样，所以与清单 6.10 相比，它应该是非常容易的。首先看第 17 行，它实现了自定义控件。看上去和实现前面创建的自定义控件一样，但现在有第 19 行的 OnTextChanged="ChangeIt"。根据清单 6.10 可知，这一行将导致这样的结果，即每当文本框中的数据发生变化时（即表单提交后），ChangeIt 方法都会执行。第 20 行创建了一个简单的按钮控件，它导致表单被提交。它调用第 5 行的 Submit 方法，这个方法只是一个占位符，不执行任何实际的工作。

第9行的 `ChangeIt` 方法将一条消息写入到浏览器中，指出发生了事件，并对其进行了处理。请在浏览器中查看该页面，尝试提交表单并修改文本数据。注意，如果没有修改文本框中的文本，则 `ChangeIt` 方法将不会被执行，也就是说该事件不会被引发。图 6.11 显示了修改文本并单击按钮后的结果。

该自定义控件还有许多可以改进的地方。例如，可以使正常情况下不发送的元素（如文本框或图像）在事件中自动发送。只要您愿意，也可以在自定义控件中创建已有的服务器控件。您甚至可以使自定义控件生成 JavaScript，以实现客户端事件处理。

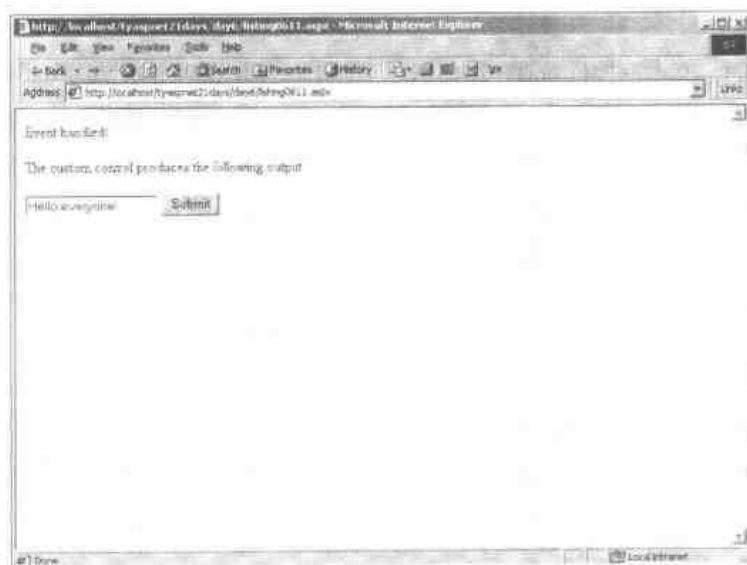


图 6.11 在 ASP.NET 页面中处理自定义控件的事件

这只是冰山一角，但是现在您已经知道了创建和使用功能齐全的自定义控件的基本知识。更详细的信息，请参见 .NET 框架文档或 .NET SDK 自带的 ASP.NET Quickstart 范例。

## 6.4 在运行阶段创建控件

到目前为止，您都是在设计阶段——即创建 ASP.NET 页面时——创建服务器控件。您也可以运行阶段——从浏览器请求页面时——动态地创建控件。

那么，区别何在呢？从技术上讲，所有的控件都是在运行阶段创建的。当用户请求页面时，控件被装载并显示。但是，设计阶段实现让您能够在控件被请求之前设置其属性。当页面被请求时，ASP.NET 已经知道必须创建该控件以及如何创建。而对于运行阶段创建来说，在代码命令 ASP.NET 创建控件之前，它并不知道需要创建控件。

既然在设计阶段创建控件可能提供同样的功能，而不要求学习额外的语法，为何还要在运行阶段创建控件呢？有时候，您可能事先并不知道需要多少控件，典型的例子是从数据库中检索数据的情况。假设您要显示数据，每条记录占一行。您很可能不知道数据库中有多少条记录，因而事先并不知道需要创建一个多少行的表格。运行阶段创建控件的方式解决了这种问题，它让您能够根据需要新建表格行。

创建内置控件和创建其他对象一样。例如，下面的代码创建了一个文本框控件，并在页面被装



载时设置其属性。您不需要在页面的 HTML 部分实现该控件：

```
<script language="VB" runat="server">
    sub Page_Load(obj as object,e as eventargs)
        dim objTB as TextBox
        objTB.ID = "tbOne"
        objTB.Text = "Hello there!"
        Page.Controls.Add(objTB)
    end sub
</script>
```

页面被装载时，文本框控件 objTB 将显示在页面上。使用这种方法，您可以根据需要创建任意数目的服务器控件。控件被创建后，必须将其加入到容器中——可以是另一个控件（如 Panel），也可以是 Page 本身。所有包含其他控件的控件都有 Controls 属性，而后者有 Add 方法，它接受要加入的控件作为参数。在浏览器中查看时，上述代码片段的输出如图 6.12 所示。

但是，要注意的是，您不能用这种方法指定事件句柄。下面的代码将出错：

```
objTB.OnTextChanged = "HandleIt"
```

而必须使用 AddHandler 方法给控件动态地添加事件句柄：

```
AddHandler objTB.TextChanged, addressOf HandleIt
```

AddHandler 方法有两个参数：要处理的事件（这里是事件 TextChanged，而不是 OnTextChanged）和用于处理事件的方法。这里必须使用操作符 addressOf，它提供一个执行相应方法的指针。现在，可以创建 HandleIt 方法，该方法在控件的 TextChanged 事件被引发时执行。

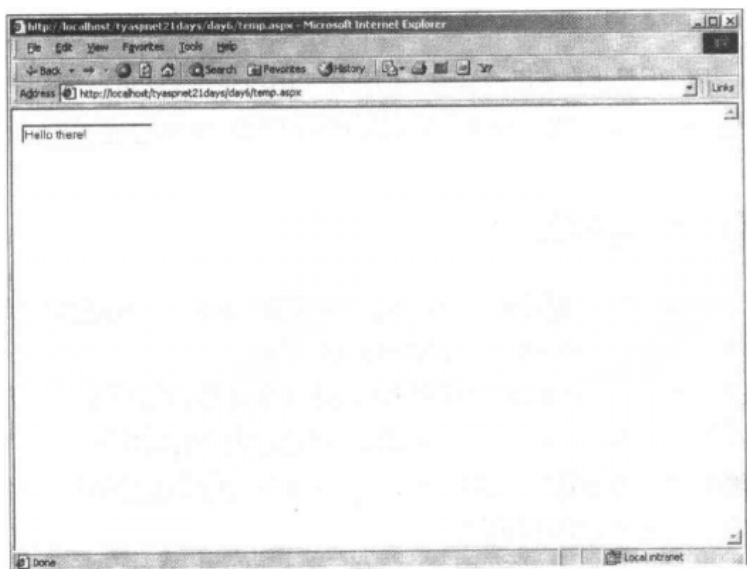


图 6.12 可以在运行阶段动态地将服务器控件加入到页面中

**提示：**如果要维护运行阶段创建的控件的状态，则必须将其加入到 ASP.NET 页面的表单中。使用 Page.Controls.Add 无法实现这项任务，而应该在 <form runat="server"> 标记中创建一个 Panel，并使用 Panel.ID.Controls.Add。

也可以使用这些方法动态地将自定义控件加入到页面中。下面的代码片段说明了如何在运行阶

段将第三个自定义控件加入到 ASP.NET 页面中:

```
<script runat="server">
    dim objCC as new MyCustomControls.CustomControl3
    sub Page_Load(obj as object, e as EventArgs)
        objCC.ID = "MyControl1"
        objCC.Message = "Hello world!"
        Page.Controls.Add(objCC)
    End sub
</script>
```

在名称空间后面加上控件名称来新建一个实例,如第2行所示(别忘了注册控件!)。但是,对于用户控件,前面介绍的方法不管用。没有名称空间的名称,如何引用控件呢? Page.LoadControl 方法可以用于动态地装载用户控件。这个方法从指定的源文件中装载用户控件,并返回一个反应文件名的对象。清单 6.13 是一个这样的例子。

**清单 6.13 LoadControl 方法返回一个基于用户控件文件名的对象**

```
1 <%@ Page Language="VB" %>
2 <%@ Register Prefix="TYASPNET" TagName="LoginForm" Src=" _
   LoginForm.ascx" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object, e as EventArgs)
6         dim objUC as LoginForm_ascx
7         objUC = Page.LoadControl("LoginForm.ascx")
8         objUC.ID = "Control1"
9         objUC.BackColor = "Beige"
10        Panel1.Controls.Add(objUC)
11
12        lblMessage.Text = "Properties of the user control: " & _
13            <br>id: " & objUC.id & "<br>" & _
14            "BackColor: " & objUC.BackColor & "<br>" & _
15            "Username: " & objUC.Username & "<br>" & _
16            "Password: " & objUC.Password
17    end sub
18 </script>
19
20 <html><body>
21     <form runat="server">
22         <asp:Panel id="Panel1" runat="server">
23     </form>
24 </p>
```

```

25 <asp:Label ID="lblMessage" runat="server" />
26 </body></html>

```

**分析:** 第 2 行注册了用户控件, 现在您应该熟悉这种语法了。我们来看一下第 6 行, 您创建了一个新的 `LoginForm.ascx` 变量 `objUC`, 这种新的数据类型是从哪里来的呢? 当您注册控件时, ASP.NET 依据用户控件的源文件名 (`LoginForm.ascx`) 创建了它, 仅是用下划线代替了句点。这种新的数据类型包含用户控件中声明的所有属性和方法, 因此可以使用它来设置属性。

在第 7 行, `LoadControl` 方法使用源文件创建了一个新的用户控件实例。该方法返回一个 `LoginForm.ascx` 对象, 所以变量 `objUC` 可以用于存储该数据。第 8 和第 9 行像前面动态装载控件那样给属性赋值 (只给属性 `ID` 和 `BackColor` 赋值), 第 10 行控件加入到第 22 行的 `Panel` 控件中。第 12-16 行在第 25 行的标签上显示控件的属性。在浏览器中查看该页面时, 得到的结果如图 6.13 所示。

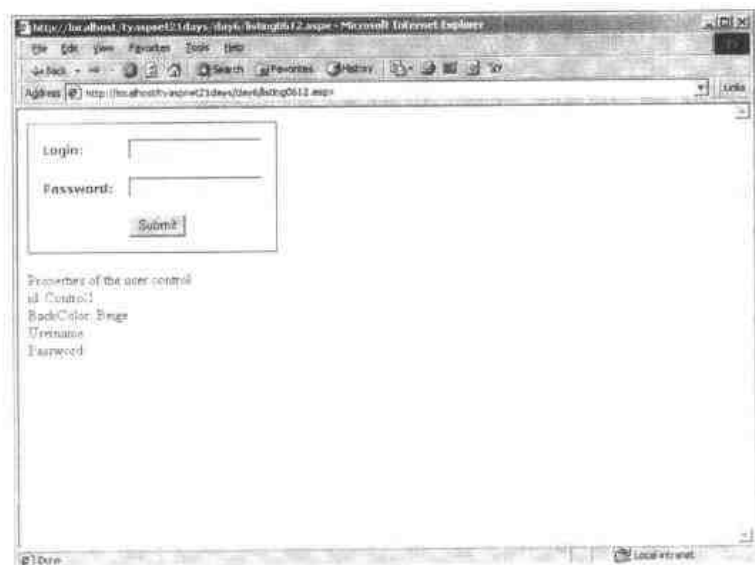


图 6.13 运行阶段创建的控制像设计阶段创建的控制那样运行

您可以像对待常规服务器控件那样给用户控件指定事件句柄。

## 6.5 这不是 ASP

本章的内容和传统 ASP.NET 之间没有多少可比性。Web 表单框架是全新的, 自定义控件和用户控件也是如此。但是您还是可以发现一些与旧技术类似的地方。

用户控件与服务器端包含, 后者常被用来封装用户界面部分, 供多个页面重用。尽管服务器端包含可以有处理 UI 的代码, 但无法从 ASP 页面访问它们。用户控件极大地扩展了服务器端包含, 给 ASP.NET 提供了关于封装的 UI 的完整信息。ASP.NET 给用户控件创建属性, 并给属性赋值, 而不用为实现操心。所有不必要的细节都被隐藏起来。下面的区别是非常重要的, 即用户控件完全是自包含的, 可以增强页面的功能, 而服务器端包含仅允许您将代码划分为独立的页面, 以便重用。

在传统 ASP 中, 与自定义控件最相似的是 COM 对象。这些对象是由开发人员从头开始创建的, 以提供可重用的功能。但是, COM 对象不提供 UI 信息 (虽然如果需要, 也可以提供)。应用于 UI

时,自定义控件提供了可重用的功能。通常不会创建不包含 UI 信息的自定义控件。ASP.NET 与 COM 对象和自定义控件交互的方式完全不同。有关 ASP.NET 中 COM 对象的更详细的信息,请参阅第 15 章。

## 6.6 总 结

本章介绍了许多关于 Web 表单框架以及如何通过创建自己的服务器控件对其进行扩展的知识。还介绍了在运行阶段和设计阶段创建的控件之间的差别,现在您应该熟悉这两种方法了。

用户控件只是一段用户界面代码,被包含在扩展名为.aspx 的文件中。任何已有的页面都可以被转换为用户控件,只需去掉所有的<html>、<body>和<form>标记,将@Page 编译指令改为@Control,并把文件扩展名从.aspx 改为.ascx 即可。用户控件可以也应该包含控制 UI 元素的逻辑。这让您能够完全封装 UI 功能,并易于在 ASP.NET 页面中实现它。自定义控件让您能够创建服务器控件,来实现所需要的任何功能。当现有的服务器控件无法满足需要时,可以使用这种控件。它们是在 VB.NET 源文件中创建的,并需要使用 VB.NET 编译器进行编译(更详细的信息,请参阅第 15 章)。

对于自定义控件,可以添加状态和事件功能。前者易于实现,只需使用状态包而不是私有变量存储控件的所有属性即可;后者较为复杂,但是幸运的是它是标准的。只需编写一次,然后将其复制并粘贴到其他控件中即可。

必须用@Register 编译指令来注册用户控件和自定义控件,然后才能在页面中使用它们。它们的语法稍有不同:

```
user control
<% Register TagPrefix="Prefix" TypeName="Name" %>
<!-- C# code -->
custom control
<% Register TagPrefix="Prefix" Namespace="Namespace"
Assembly="AssemblyName" %>
```

除了可以在设计阶段创建服务器控件外,还可以在运行阶段创建它们。这使您在无法预见的环境中也能控制 UI,例如数据太多,无法全部加入到 UI 中时。可以像 ASP.NET 中其他任何的对象那样,创建常规服务器控件和自定义控件,即使用 dim 语句。但对于用户控件,则必须使用 Page.LoadControl 方法来新建其实例。该方法返回的数据类型是基于用户控件源文件的文件名称的。可以使用这种数据类型的变量来给控件的属性赋值。

下一章将介绍有效性验证控件,从而完成对 Web 表单的介绍。这些控件让您能很容易地验证用户输入的有效性,防止应用程序发生错误。您将发现,强迫用户按您的要求输入信息非常容易。

## 6.7 问与答

问: 用户控件的处理方式与 ASP.NET 页面的处理方式有什么不同?

答: 事实上,用户控件的处理方式和 ASP.NET 页面的处理方式非常相似。首次被请求时,用户控件将被编译,就像.aspx 文件那样。用户控件也可以包含 Page\_Load 事件处理程序。当用户控件和 ASP.NET 页面中都包含 Page\_Load 事件处理程序时,后者被首先处理。其他事件的处理次序和 ASP.NET 页面相同。

问：用户控件和 code-behind 表单有什么不同？

答：如果您使用过 ASP.NET，则可能知道 code-behind 表单。初学者常混淆这两者之间的区别。用户控件封装用户界面及其功能，供其他应用程序重用。Code-behind 表单是一个类，它声明了对象及其方法，供.aspx 文件使用，但没有用户界面元素。

可以将用户控件看作积木，可以用于创建更大的 UI。它包含了部分 UI，并知道如何自己处理。而 code-behind 表单是.aspx 文件的父类，告诉.aspx 文件它能做什么以及如何做，而没有指出其外观。

## 6.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 6.8.1 小测验

1. 判断正误：用户控件本身便可运行。
2. 将.aspx 文件转换为用户控件的三个要求是什么？
3. 给这样一个用户控件创建一条@Register 编译指令，即其标记前缀为 ACMEVBCustomControls，名称为 TextWriter，该控件位于子目录 ucontrols 的 shared.ascx 文件中。
4. 判断正误：自定义控件必须通过编译后才能在 ASP.NET 页面中使用。
5. 判断正误：自定义控件必须直接或间接地从 Object 类派生而来。
6. 用 VB.NET 的属性元素的语法，为自定义控件创建一个名为 Color 的简单属性，用于存储视图状态信息。
7. 判断正误：LoadControl 方法可用于动态地装载自定义控件。

### 6.8.2 练习

1. 对第 2 章的计算器练习进行改进，使它类似于 Windows 中的计算器。包括记忆功能（使用 Session 变量）、清除和退位按钮以及其他操作符按钮。然后将其封装为一个用户控件。这个练习相当复杂，但别着急。您已经学习过需要的所有技术。

提示：创建 UI 的工作相当容易，只需使用一个 HTML 表格即可。另外，需要注意的是，许多功能是重复的。例如，每个数字按钮都执行相同的功能（在文本框中显示一个数字），所以它们可以引用同一个事件句柄。最后，在确定要处理的数字时，隐藏表单字段将会有所帮助。例如，用户依次按下 6、乘号和 7，则将 6 存储在隐藏字段中，防止以后提交表单时，该数字被丢失。

2. 创建一个会议安排程序。它包括一个日历、一个文本框（供用户输入）和一个标签（显示当前数据）。当用户向文本框中输入数据时，使用事件 AutoPostBack 和 TextChanged。把输入的数据存储会话变量中，以便用户以后可以使用它们。当会议通知被加入时，动态创建一个标签，来反馈信息。有关日历控件的事件和方法，请参考附录 C。

## 第 7 章

# 验证 ASP.NET 页面

对于开发人员来说，验证用户在 ASP.NET Web 表单中的输入是否有效是最重要的任务之一。在提供信息时，用户的行为是无法预测的，因此必须采取额外的预防措施，确保用户提供的数据是正确的。例如，假设您要求用户输入信用卡号码，如果用户输入的数据（如 ddfg934053）无效，将会发生什么情况呢？如果少了一位数字又将如何呢？显然这将在应用程序中引起问题，特别是当用户下订单时。

在 ASP.NET 中，这些有效性验证很容易实现。在验证用户输入的有效性方面（包括模式匹配和确保输入数据在给定的范围内），可选择的方法很多。Validation 控件提供了许多指出用户错误的方法，它们自动运行，无需提交给服务器。当内置的选项无法满足要求时，可创建自己的方法。

本章将介绍 ASP.NET 中各种验证用户输入有效性的方法。具体地说，将介绍 Validation 服务器控件，它使得验证输入数据的有效性比以前任何时候都更容易；还将介绍如何创建自己的有效性验证了程序。

本章包括以下内容：

- 在 ASP.NET 之前，为何要验证用户输入的有效性，如何验证；
- ASP.NET Validation 控件是什么？它是如何运行的；
- 如何在页面中实现 Validation 控件；
- 如何对显示给用户的错误消息进行定制；
- 如何创建自定义 Validation 控件。

### 7.1 有效性验证情形

假设您到学校注册课程，注册员询问您的姓名。如果您说自己的姓名是“56”，他会怀疑您在撒谎，而再询问一次。他必须确认您的姓名确实是“56”或得知您的真实姓名。

无论什么时候，您向别人询问问题，都必须有得到意外回答的准备。对于一个简单的问题，您也不知道他们会回答什么。对于计算机来说，这种行为是灾难性的。如果应用程序期待某种类型的数据，而提供的却是其他类型的数据，则应用程序可能会出错甚至崩溃（您可能在 ASP.NET 页面中遇到过）。输入的有效性验证是确保用户提供正确类型数据的过程。

当您允许用户手工在 ASP.NET 页面中输入信息时，有效性验证是必不可少的。大多数用户都会尽量按要求的格式输入信息，但他们也有出错的时候。还有些用户喜欢通过输入无效数据来破坏

应用程序。在这两种情况下，得到的数据通常不会是您所期望的。

例如，假设您创建了一个电子商务站点，接收办公用品订单。最后您需要从用户那里获取其个人信息，如姓名、e-mail、付款方式等。您必须确保这些信息的准确性，否则订单将无法完成。例如，如果用户输入的收货地点为“45”，应用程序不能立即识别它，那么货物就永远不会递送出去，您最后面对的是愤怒的客户。

那么，如果用户在应该填写姓名的地方填写的是一个数字，又将如何呢？或者对于 e-mail 地址，输入的是一个随机的字符串，又将如何呢？最终，输入到系统的数据将是错误的。检测无效输入对于实现稳定的应用程序至关重要。

输入有效性验证也给开发人员带来了好处——具体地说，是数据的一致性。设计应用程序时，总是期望数据符合特定的格式。例如，如果要将两个数字相加，则需要确定它们确实是数字。否则，加法运算将无法进行，应用程序将出现问题。如果预先对输入进行有效性验证，便可确保顺利地完成加法运算。

我们来看一种要求验证输入有效性的典型情况。清单 7.1 是一个 ASP.NET 页面的简单用户界面，它让用户能够输入一些个人信息。

**清单 7.1 一种典型的有效性验证情形**

```

1 <!DOCTYPE html>
2 <form runat="server">
3     <asp:Label id="lblHeader" runat="server"
4         Height="20px" Width="100%" BackColor="#d3aa66"
5         ForeColor="white" Font-Bold="true"
6         Text="A Validation Example" />
7     <asp:Label id="lblMessage" runat="server" />
8     <asp:Panel id="Panel1" runat="server">
9         <table>
10             <tr>
11                 <td width="100" valign="top">
12                     First and last name:
13                 </td>
14                 <td width="300" valign="top">
15                     <asp:TextBox id="txtFName" runat="server" />
16                     <asp:TextBox id="txtLName" runat="server" />
17                 </td>
18             </tr>
19             <tr>
20                 <td valign="top">Email:</td>
21                 <td valign="top">
22                     <asp:TextBox id="txtEmail"
23                         runat="server" />
24                 </td>

```

```

25         <td>
26             <tr>
27                 <td colspan="2" id="top">Address: </td>
28             </td colspan="2" id="top">
29                 <asp:TextBox id="tAddress"
30                     runat="server" />
31             </td>
32         </tr>
33         <tr>
34             <td colspan="2" id="top">City, State, ZIP: </td>
35             </td colspan="2" id="top">
36                 <asp:TextBox id="tCity"
37                     runat="server" />
38                 <asp:TextBox id="tState" runat="server"
39                     size="20" />
40                 <asp:TextBox id="tZip" runat="server"
41                     size="20" />
42             </td>
43         </tr>
44         <tr>
45             <td colspan="2" id="top">Phone: </td>
46             </td colspan="2" id="top">
47                 <asp:TextBox id="tPhone" runat="server"
48                     size="11" />
49             </td>
50         </tr>
51         <tr>
52             <td colspan="2" id="top"> </td>
53             <td colspan="2" id="top">
54                 <asp:Button id="bSubmit" runat="server"
55                     text="Add OnClick="Submit" />
56             </td>
57         </tr>
58     </table>
59 </form>
60 </body></html>

```

**分析：**该清单使用了几个 Web 控件来显示用户界面。用户界面让用户提供名、姓、e-mail 地址、街道地址、城市、州、邮政编码和电话号码。清单 7.1 的运行结果如图 7.1 所示。



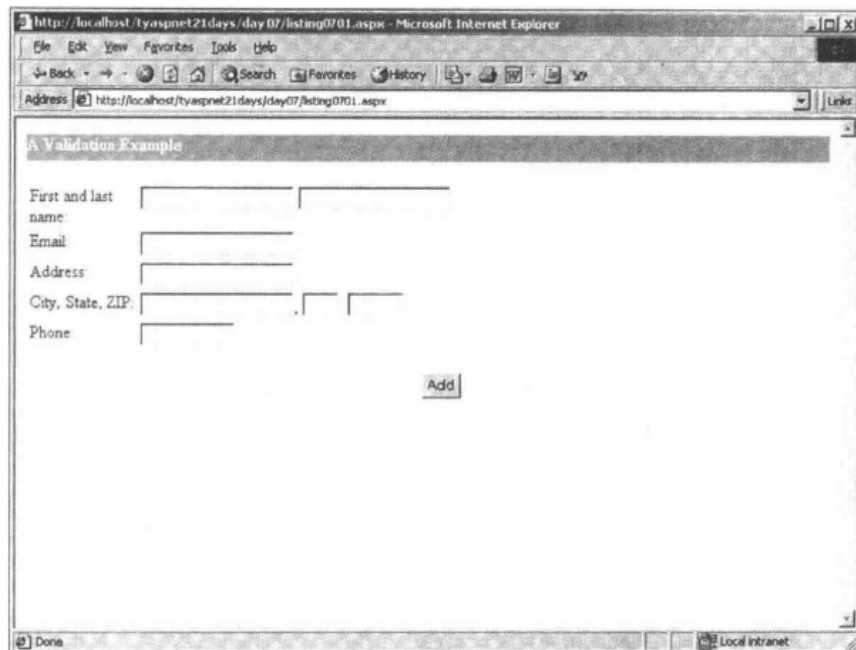


图 7.1 一个典型的用户输入页面

该表单被提交后,您必须在将数据输入到数据库之前,核实每个字段中的输入是否都是有效的。常用的办法是使用一系列 if 语句,如清单 7.2 所示。这些 if 语句验证清单 7.1 中的所有服务器控件中的数据。

#### 清单 7.2 使用 if 语句来验证清单 7.1 中用户输入的有效性

```

1 <script runat="server">
2   sub Submit(obj as Object, e as EventArgs)
3       if tbFName.Text <> "" and not IsNumeric(tbFName.Text) then
4           if tbLName.Text <> "" and not IsNumeric(tbLName.Text) then
5               if tbAddress.Text <> "" then
6                   if tbCity.Text <> "" and not IsNumeric(tbCity.Text) then
7                       if tbState.Text <> "" and not IsNumeric(tbState.Text) then
8                           if tbZIP.Text <> "" and IsNumeric(tbZIP.Text) then
9                               if tbPhone.Text <> "" then
10                                  lblMessage.Text = "Success!"
11                               else
12                                  lblMessage.Text = "Phone is incorrect!"
13                               end if
14                           else
15                                  lblMessage.Text = "ZIP is incorrect!"
16                               end if
17                       else
18                                  lblMessage.Text = "State is incorrect!"
19                       end if

```

```

20         else
21             lblMessage.Text = "City is incorrect!"
22         end if
23     else
24         lblMessage.Text = "Address is incorrect!"
25     end if
26 else
27     lblMessage.Text = "Last name is incorrect!"
28 end if
29 else
30     lblMessage.Text = "First name is incorrect!"
31 end if
32 end sub
33 </script>

```

**分析：**正如您看到的，以这样的方式验证所有控件需要的代码很长，也很麻烦。第 3 行的 `if` 语句核实用户是否在“姓”字段中输入了输入，并且不是一个数字。第 4 行有第二个 `if` 语句对“名”字段做同样的事情。如此继续下去，直到核实完包含用户输入的所有控件。别忘了 `else` 语句，它显示一条消息，指出错误何在。

以这种方式核实所有用户输入后，可采取的措施有两种。如果用户输入的所有信息的格式都是正确的，则可以将这些数据插入到数据源中；如果其中的某项信息的格式不正确，则需要提醒用户：输入不正确（使用 `else` 语句），并允许更正错误（只需重新显示页面即可），然后将数据插入到数据库中。

然而，对于每条输入信息，经常需要检查多种情况，因此有效性验证将非常复杂。例如对于“名”字段来说，必须核实该字段不为空、不包含数字、不包含空格，等等。这需要使用许多 `if` 语句。尽管这么麻烦，它仍然是一种典型的验证用户输入有效性的方法。这个过程包含以下步骤：

1. 向用户显示表单，让他输入数据；
2. 表单提交后，用 `if` 语句检查每项用户输入，包括是否为空、格式和数据类型是否正确、是否在合法的范围内（对于日期而言），等等；
3. 如果所有 `if` 语句都通过了，则执行相应的功能；
4. 如果未通过，则重新显示表单，最好其中包含用户原来输入的内容，这样用户只需更正错误字段，而保留其他的字段。然后回到第 2 步。

## 7.2 ASP.NET 有效性验证

ASP.NET Validation 服务器控件让您能够轻松地验证用户在 Web 表单中输入的任何数据的有效性。该控件支持诸如必须填写的字段和模式匹配等有效性验证，同时使得创建自定义有效性验证更为容易。此外，当数据未通过有效性验证时，Validation 控件让您能够定制向用户显示错误消息的方式。

Validation 控件与 Web 控件类似（参见第 5 章）。它们是在服务器上创建的，并将 HTML 输出到浏览器中，声明的语法也相同：

```
<asp:ValidationName lunat> 'server'
    ControlToValidate="ControlName"
    ErrorMessage="Descriptive Text" >
```

不同之处在于，除非输入未通过有效性验证，否则这种控件将不会显示任何东西，在这种情况下，这种控件对于用户来说是不可见的，用户无法与之交互。所以 Validation 控件的工作是监视另一个服务器控件，并验证其内容的有效性。ControlToValidate 属性指定要监视的用户输入服务器控件。当用户在被监视的控件中输入数据时，Validation 控件将检查这些数据，确保符合指定的所有规则（参见后面的“Validation 控件的工作原理”一节）。

表 7.1 对 ASP.NET 提供的预定义有效性验证类型做了总结。在本章的学习过程中，您将使用所有这些控件，它们都属于名称空间 System.Web.UI.WebControls。

表 7.1 ASP.NET Validation 控件类型

Validation 控件	描 述
RequiredFieldValidator	确保用户没有遗漏必须填写的字段
CompareValidator	使用比较操作符（如小于、等于、大于等）对用户输入与一个常量、另一个控件的属性值或数据库中的值进行比较
RangeValidator	检查用户输入是否位于指定的范围之内。上下限可以是一对数字、字母字符或日期。上下限可以用常量或来自其他控件的值来表示（来自.NETSDK 文档）
RegularExpressionValidator	检查输入是否与正则表示指定的模式匹配。这种有效性验证让您能够检查可预见的字符序列，例如社会保险号、e-mail 地址、邮政编码等中的字符序列（来自.NETSDK 文档）
CustomValidator	使用自己编写的有效性验证逻辑来检查用户的输入。这种有效性验证让您能够检查运行阶段生成的值

不幸的是，Validation 控件只能验证一部分 ASP.NET 服务器控件中输入的有效性。在大多数情况下，这些控件对于验证所有用户输入的有效性绰绰有余。可以用 ASP.NET 的 Validation 控件来验证其输入的有效性的控件包括：

- HtmlInputText;
- HtmlTextArea;
- HtmlSelect;
- HtmlInputFile;
- TextBox;
- ListBox;
- DropDownList;
- RadioButtonList。

注意，Validation 控件不属于上述范围，它们不许用户输入任何内容。

### 7.2.1 Validation 控件的工作原理

要在 ASP.NET 页面中创建 Validation 控件，只需在 Web 表单中加入合适的 Validation 控件标记

即可。所加入的每个 Validation 控件都有特定的任务：监视另一个服务器控件。当用户在该服务器控件中输入任何信息时，相应的有效性验证器将仔细检查输入的内容，看是否能通过指定的所有测试。这些工作不需要开发人员的干涉，也不用提供额外的代码。

Validation 控件同时在客户端和服务端运行。当用户在 Web 控件中输入的数据违反了指定的有效性验证规则时，用户会立即被提醒。事实上，ASP.NET 将不允许包含无效信息的表单被提交。这就是说，进行有效性验证时，无需在客户端和服务端之间发送信息——这项工作自动在客户端进行。这使得性能提高很多，用户的感觉也更好。

**注意：**仅当浏览器支持DHTML和 JavaScript时，才会进行客户端有效性验证。

当表单最后提交给服务器后，ASP.NET 将再次验证输入的有效性，这使您可以根据客户端无法使用的资源（如数据库）对数据再次进行核实。如果信任客户端有效性验证，也可以不在服务器上进行有效性验证，但当有效性验证流程更为复杂时，在服务器端进行验证常常很有帮助。

尽管这种功能大部分是自动完成的，但要使这些控件正常运行，还需要在幕后完成许多工作。清单 7.3 是一个简单的例子。

**清单 7.3 一个简单的有效性验证范例**

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub SubmitObj as object, e as EventArgs
5         if Page.IsValid then
6             lblMessage.Text = "You passed validation!"
7         end if
8     end sub
9 </script>
10
11 <html>
12 <form runat="server">
13     <asp:Label id="lblMessage" runat="server" />
14
15     Enter your name:
16     <asp:TextBox id="tbName" runat="server" />
17     <asp:RequiredFieldValidator runat="server"
18         ControlToValidate="tbName"
19         ErrorMessage="First name required" />
20
21     <asp:Button id="tbSubmit" runat="server"
22         Text="Submit"
23         OnClick="Submit" />
24 </form>
25 </body></html>

```

分析：将该文件保存为 listing0703.aspx，并在浏览器中查看它。在介绍后台发生的情况之前，先来看一下该页面本身。

该页面的 HTML 部分包含 4 个服务器控件：Label（第 13 行）、TextBox（第 16 行）、Validation 控件 `RequiredFieldValidator`（第 17-19 行）和 Button 控件（第 21-23 行）。`RequiredFieldValidator` 的声明方式与其他控件相同，包含控件名和 `runat="server"`。`ControlToValidate` 属性指定了该 Validation 控件要监视的服务器控件（这里是 `tbFName`）。

只要在 `tbFName` 文本框中输入了数据，`RequiredFieldValidator` 的要求便满足了。该控件只是检查字段中是否包含数据。每个 Validation 控件都有 `IsValid` 属性，指示是否通过有效性验证。在这个例子中，只要用户输入了数据，该属性就将被设置为 `true`；否则为 `false`。表单被提交后，第 4 行的 `Submit` 方法将执行。`Page.IsValid` 属性确保页面中所有 Validation 控件的要求都得到了满足，当然也可以检查各个控件的 `IsValid` 属性，但这样做更快。如果页面中所有的 Validation 控件对用户的输入都满意，则第 5-7 行向用户显示一条消息。

尝试不在文本框中输入任何内容就单击提交按钮，您会看到如图 7.2 所示的结果。

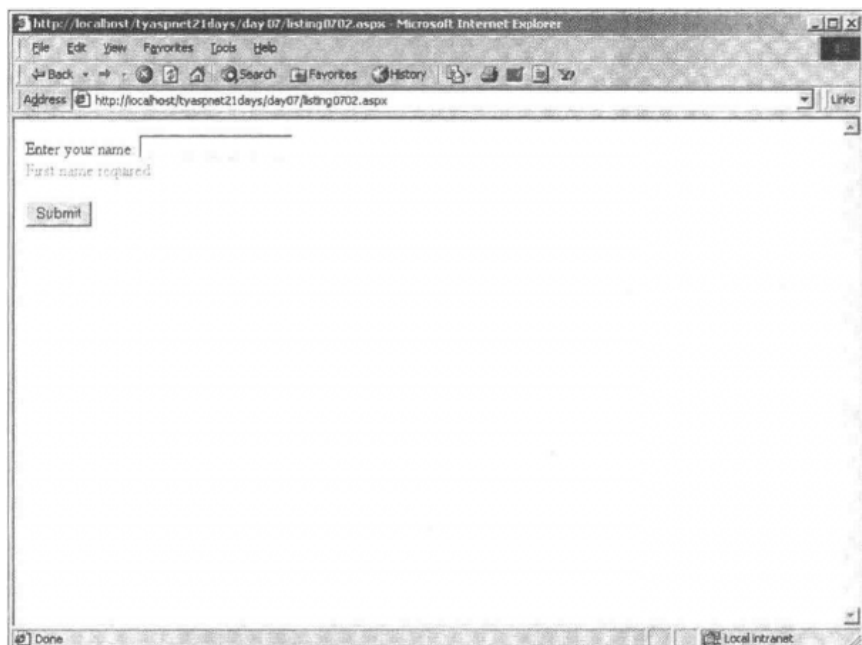


图 7.2 Validation 控件显示错误消息

怎么样？位于第 17 行的 Validation 控件检查其监视的控件是否包含数据。由于没输入任何信息，所以 Validation 控件禁用提交表单，并显示第 19 行的 `ErrorMessage` 属性指定的消息。服务器根本就不看用户的输入。

尝试在文本框中输入一些信息，然后离开该元素。错误消息自动消失了！删除其中的文本后，错误消息又出现了：当光标离开元素时，每个元素都将被验证。

注意：只有支持动态 HTML 的浏览器（Internet Explorer 和 Netscape 4+）才支持错误消息的动态显示。

我们来看一看 ASP.NET 页面生成的 HTML。在页面上单击鼠标右键，选择“查看源文件”。清单 7.4 显示了 HTML 源文件的一部分。

清单 7.4 从有效性验证页面中摘录的部分源代码

```

1  ...
2  <form name='ctrl1' method="post" action="listing0702.aspx
3      language="javascript" onsubmit="ValidatorOnSubmit();"
4      id= 'ctrl1">
5  ...
6  ...
7  <script language="javascript"
8      src="/_asp/1.0.2523/script/WebUIValidation.js">
9  </script>
10 ...
11 ...
12 <span id="ctrl16" controltovalidate="tb7Name"
13     errormessage="First name required" evaluationfunction="
14     RequiredFieldValidatorEvaluateIsValid" initialvalue=" "
15     style="color:Red;visibility:hidden;">
16     First name required</span><p>
17 ...
18 ...
19 <script language="javascript">
20 <!--
21 ...
22 ...
23     function ValidatorOnSubmit() {
24         if (Page_ValidationActive) {
25             ValidatorCommonOnSubmit();
26         }
27     }
28     /-->
29 </script>
30 ...
31 ...

```

**分析：**该页面的内容很多，所以我们仅看重要的部分。第2行包含一个标准的表单标签。但需要注意的是，该表单被提交时（即 Submit 方法被激发时），表单将执行 ValidatorOnSubmit 函数，而不是直接提交给服务器。该函数位于第25行，它确定页面是否启用了有效性验证（默认情况为启用），并执行另一个函数，来执行有效性验证处理工作（更详细的信息，参见本章后面的“禁用有效性验证”一节）。

第8行包含一个对 JavaScript 文件 WebUIValidation.js 的引用，该文件位于服务器上。该脚本是由 ASP.NET 自动生成的，包含动态显示错误消息、验证输入的有效性、向服务器发送数据所需要的所有客户端 DHTML 函数。该文件非常大，所以这里不打算介绍它——它通常位于

c:\inetpub\wwwroot\\_aspx\version\script 中。

第 12 行是 Validation 服务器控件的 HTML 输出——一个<span>元素，它包含一个自定义属性：evaluationfunction，该属性告诉 WebUIValidation.js 应执行何种有效性验证。该 span 被设置为“hidden”，即在验证器函数获得控制权之前，用户看不到它。

这个过程很复杂，但您无需做任何创建工作，因为 ASP.NET 自动为您处理所有的事情。

当该客户端脚本执行并评估页面中的每个 Validation 控件时，它基于有效性验证结果为每个 Validation 控件设置一个参数，这些参数被发送给服务器，再次进行有效性验证。图 7.3 说明了这个过程。

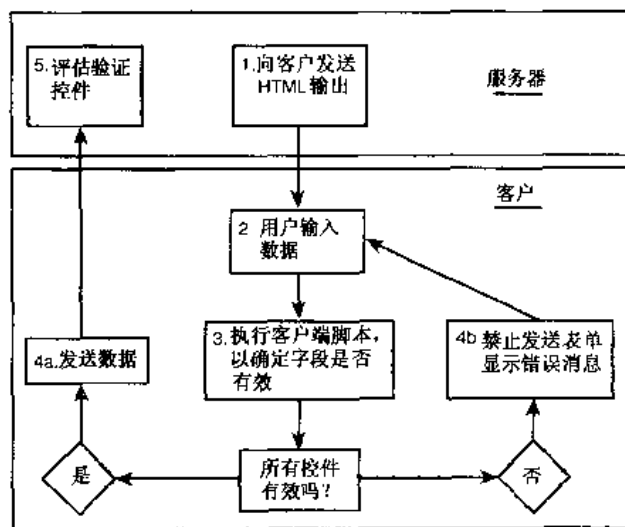


图 7.3 在服务器和客户端发生的有效性验证过程

**注意：**还记得吗？客户端有效性验证仅在支持 DHTML 的浏览器上进行。此外，如果客户端禁用了 JavaScript，则客户端有效性验证也不会进行。

为什么服务器需要再次验证信息的有效性呢？想象一下您需要根据服务器上的数据库来验证某些信息的有效性或想防止用户多次提交表单的情况。有时再次进行有效性验证是必须的，多加注意总没坏处。

收到表单数据后，服务器检查每一个 Validation 控件的状态（通过 IsValid 属性）。如果它们全被满足了，则 ASP.NET 将 Page 对象的 IsValid 属性设置为 true；否则设置为 false。即便属性为 false，代码也会继续执行。无论输入是否通过有效性验证，代码都执行，所以如果输入无效，是否终止代码运行将取决于您，这通常通过检查 Page.IsValid 属性来实现。

## 7.3 使用 Validation 控件

所有 Validation 控件都包含相似的属性。至少，每个控件都应指定两个属性（不包括 runat="server"）。首先，它们必须包括 ControlToValidate 属性，该属性指定了 Validation 控件要监视的服务器控件的名称。其次，每个控件必须有 ErrorMessage 属性，该属性告诉 ASP.NET：有效性验证失败时，应向用户显示什么消息。另外还有几个其他的属性，这将在本章后面的“定制有效性验证”一

节中介绍。

我们回过头来看一下清单 7.1，将它改为使用 Validation 控件。首先，考虑一下需要什么类型的有效性验证。

对于文本框“名”和“姓”，需要确保用户输入了数据——任何数据都行，只要文本框包含文本。这种有效性验证可以使用 RequiredFieldValidator 控件。为了更保险，还要确保名和姓不同。这可以防止用户在文本框中输入同一个字符串（如“asdf”）。可以使用 CompareValidator 来实现这一目标。

对于 e-mail 文本框，需要确保 e-mail 地址的格式有效，如 somename@some.com，这也是为了防止用户随机地输入字符串。RegularExpressionValidator 控件可以实现这种功能。对于邮政编码和电话号码文本框，也需要检查其内容的格式是否有效。例如，邮政编码必须是 5 位数，不包含字母，而电话号码的格式是 x x x - x x x x（就我们的目的而言）。

最后，假设您只允许来自美国特定地区（邮政编码位于特定范围内）的用户提交表单。这种有效性验证可以通过 RangeValidator 控件来实现。清单 7.5 列出了该页面的 UI 部分。

清单 7.5 改进后的用户表单的 UI 部分

```

1 <html ><body>
2   <form runat='server'>
3     <asp:Label id='lblHeader' runat='server'
4       Height='25px' Width='100%' BackColor='#d4aa66'
5       ForeColor='white' Font-Bold='true'
6       Text='A Validation Example' />
7     <asp:Label id='lblMessage' runat='server' /><br>
8     <asp:Panel id='Panel1' runat='server'>
9       <table>
10        <tr>
11          <td width='100' valign='top'>
12            First and last name:
13          </td>
14          <td width='300' valign='top'>
15            <asp:TextBox id='tbFName' runat='server' />
16            <asp:TextBox id='tbLName' runat='server' />
17            <br>
18            <asp:RequiredFieldValidator runat='server'
19              ControlToValidate='tbFName'
20              ErrorMessage='First name required' /><br>
21            <asp:RequiredFieldValidator runat='server'
22              ControlToValidate='tbLName'
23              ErrorMessage='Last name required' /><br>
24            <asp:CompareValidator runat='server'
25              ControlToValidate='tbFName'

```



```

26         ControlToCompare="tbLName"
27         Type='String'
28         Operator='NotEqual'
29         ErrorMessage="First and last name
30         cannot be the same" />
31     </td>
32 </tr>
33 <tr>
34     <td valign='top'>Email (.com's only):</td>
35     <td valign="top">
36         <asp:TextBox id="tbEmail"
37             runat="server" /><br>
38         <asp:RegularExpressionValidator
39             runat="server"
40             ControlToValidate="tbEmail"
41             ValidationExpression="\w+\.@\w+\.com"
42             ErrorMessage="That is not a valid email" />
43     </td>
44 </td>
45 </tr>
46 <tr>
47     <td valign='top'>Address:</td>
48     <td valign="top">
49         <asp:TextBox id="tbAddress"
50             runat="server" />
51     </td>
52 </tr>
53 <tr>
54     <td valign="top">City, State, ZIP (5-digit):</td>
55     <td valign="top">
56         <asp:TextBox id="tbCity"
57             runat="server" />,
58         <asp:TextBox id="tbState" runat="server"
59             size=2 />&nbsp;
60         <asp:TextBox id="tbZIP" runat="server"
61             size=5 /><br>
62         <asp:RegularExpressionValidator
63             runat="server"
64             ControlToValidate="tbZIP"
65             ValidationExpression="[0-9]{5}"

```

```

66         ErrorMessage="That is not a valid ZIP" />
67         <br>
68         <asp:RangeValidator runat="server"
69             ControlToValidate="tbZIP"
70             MinimumValue="00000" MaximumValue="22222"
71             Type="String"
72             ErrorMessage="You don't live in the
73             correct region" />
74     </td>
75 </tr>
76 <tr>
77     <td valign="top">Phone (<i>x1xxx-xxxx</i>);</td>
78     <td valign="top">
79         <asp:TextBox id="tbPhone" runat="server"
80             size=11 /><br>
81         <asp:RegularExpressionValidator
82             runat="server"
83             ControlToValidate="tbPhone"
84             ValidationExpression="[0-9]{3}-[0-9]{4}"
85             ErrorMessage="That is not a valid phone"
86         />
87     </td>
88 </tr>
89 <tr>
90     <td colspan="2" valign="top" align="right">
91         <asp:Button id="Submit" runat="server"
92             text="Add" />
93     </td>
94 </tr>
95 </table>
96 </asp:Panel>
97 </form>
98 </body></html>

```

**分析：**这段代码很长，但只有少量的代码需要介绍，其他的都是常规 HTML。首先，我们看看该页面没有做什么。E-mail、城市、州、邮政编码和电话号码控件都没有对应的 `RequireFieldValidators`，因此它们都是可选的。即使用户没有在这些字段中输入数据，也可通过有效性验证，因为没有 `Validation` 控件监视这些字段是否为空。即使这些字段为空，控件的 `IsValid` 属性也将为 `true`。其次，要注意的是，该表单只接受特定格式的 e-mail 地址、邮政编码和电话号码。E-mail 地址必须以 `.com` 结束；邮政编码必须是 5 位（有些邮政编码使用 9 位数，`xxxxx-xxxxx`）；而电话号码只能是 7 位，第 3 位和第 4 位之间有短划线。虽然这些有效性验证规则的功能有限，但

是对于学习本章的内容而言足够了。在实际情况中，规则可能需要更通用些。

我们来看第 18-30 行。这里有两个 RequiredFieldValidator 实例，分别在第 18 和第 22 行。这些控件分别确保“名”和“姓”文本框不为空。本章前面已经使用过这种控件。这些代码行只包含两个必不可少的属性：ControlToValidate 和 ErrorMessage。

第 24 行的 CompareValidator 将服务器控件与常量或另一个服务器控件进行比较。这里比较的是 tbFName 和 tbLName，防止它们相同。Type 属性告诉 ASP.NET，您要比较的是哪种类型的数据，如字符串、双精度、日期等。Operator 指定进行哪种类型的比较。这里比较的是两个文本框中的字符串，并确保它们不相等。同样，它们也包含标准的 ControlToValidate 和 ErrorMessage 属性。表 7.2 和 7.3 列出了 CompareValidator 可使用的操作符和类型。

表 7.2 CompareValidator 的 Operator 属性可使用的操作符

操 作 符	描 述
DataTypeCheck	检验数据是否为特定的数据类型（字符型、整型等）
Equal	检验是否相等
GreaterThan	检验是否大于
GreaterThanEqual	检验是否大于或等于
LessThan	检验是否小于
LessThanEqual	检验是否小于或等于
NotEqual	检验是否不等

表 7.3 用于 CompareValidator 的 Type 属性的类型

类 型	描 述
Currency	金额
Date	日期值
Double	双精度值（浮点）
Integer	整数值
String	字符串值

**注意：**如果要与常量进行比较，请将 ControlToCompare 改变为 ValueToCompare。例如，下面的 CompareValidator 检查输入的数据是否是字符串“Chris”。如果是，则有效性验证失败，并显示一条消息：

```
<asp:CompareValidator runat="server"
    ControlToValidate="tbFName"
    ValueToCompare="Chris"
    Type="String"
    Operator="NotEqual">
```

```
ErrorMessage: 'Your first name cannot be the same as mine' />
```

如果仅想检验用户输入的是否为有效的数据类型,则可以将 CompareValidator 的 Operator 属性设置为 DataTypeCheck,同时无需使用 ControlToCompare 属性,但使用该操作符时应小心。在 Web 表单中,所有的输入都被看作是字符串,因此可能没有实现正确的有效性验证。例如,如果用户输入 76 7878,则下面的 CompareValidator 将得到满足:

```
<asp:comparevalidator runat="server"
    Control="ctl_date" data-type="DataTypeCheck"
    type="String"
    operator="DataTypeCheck"
    ErrorMessage: "You must enter a string value" />
```

注意,使用两个或更多 Validation 控件监视同一个服务器控件时,只有当所有条件都满足后,输入才算有效。因此“名”和“姓”文本框都必须填写,同时它们不能相同。

第 38 行声明了一个 RegularExpressionValidator 控件,它检验用户的输入是否与特定的字符模式匹配。该控件的唯一一个新属性是第 41 行的 ValidationExpression,它指定了用于验证输入的有效性的正则表达式。字符串 \w+\@ \w+\.com 表示一个或多个单词字符(即字母),后接 @ 符号,再后面是一个或多个单词字符,然后是 .com。更详细的信息,请参阅本章后面“正则表达式”一节。

第 62 行使用另一个 RegularExpressionValidator 来检验输入的邮政编码是否正确。字符串 [0-9]{5} 指的是 5 个 0-9 的数字。

第 68 行包含一个 RangeValidator 控件,它检验指定的服务器控件的值是否位于指定的范围内,这样可限制只有美国特定地区的人才可以提交该表单,您甚至可以使用该控件比较日期和字符串。第 70 行指定的 MinimumValue 和 MaximumValue 属性设置了邮政编码的范围。将 Type 属性设置为 String,以便邮政编码被正确地检验(如果使用双精度型或整型,字符串 0001 将被认为是 1,这不是您所希望的)。

最后,第 81 行使用了另一个 RegularExpressionValidator,该控件被用于验证电话号码的有效性,它确保电话号码为 7 位。ValidationExpression 字符串 [0-9]{3}-[0-9]{4} 表示由 0-9 组成的 3 位数,后面跟短划线,再后面是由 0-9 组成的 4 位数。这是美国的 7 位电话号码。

**提示:**对于 RegularExpressionValidator,可以在 ValidationExpression 属性中使用 | 符号(意思是“或者”)来指定多种匹配模式。例如下面的字符串与带短划线的 7 位电话号码、带短划线的 10 位电话号码或不带短划线的 10 位电话号码匹配:

```
[0-9]{3}-[0-9]{4} | [0-9]{3}-[0-9]{4} | [0-9]{4} | [0-9]{10}
```

用户输入下面的任何一个号码都有效:

```
458-9865
625-458-6865
6254189865
```

使用该表单的试验获得不同的输出。输入模式匹配和不匹配的值来测试 RegularExpressionValidator 控件。尝试在一些字段中输入值而空着另一些,然后提交表单。图 7.4 是输入无效数据后得到的结果。注意数据无效的文本框旁边的错误消息。图 7.5 显示了正确输入所有数据后得到的结果。

**警告:**别忘了,只有 RequiredFieldValidator 控件才认为空字段无效,其他所有控件都接受空字段。这意味着虽然 e-mail 地址字段为空,仍可通过 RegularExpressValidator 的检查,同时该控件的 IsValid

属性将为true，即使该字段为空，验证器也不会终止运行。

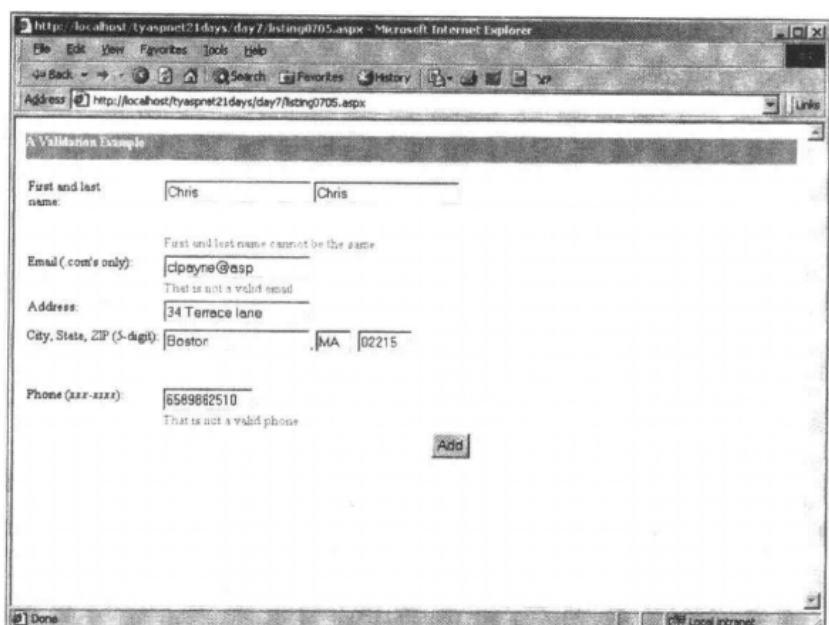


图 7.4 使用无效数据测试有效性验证表单

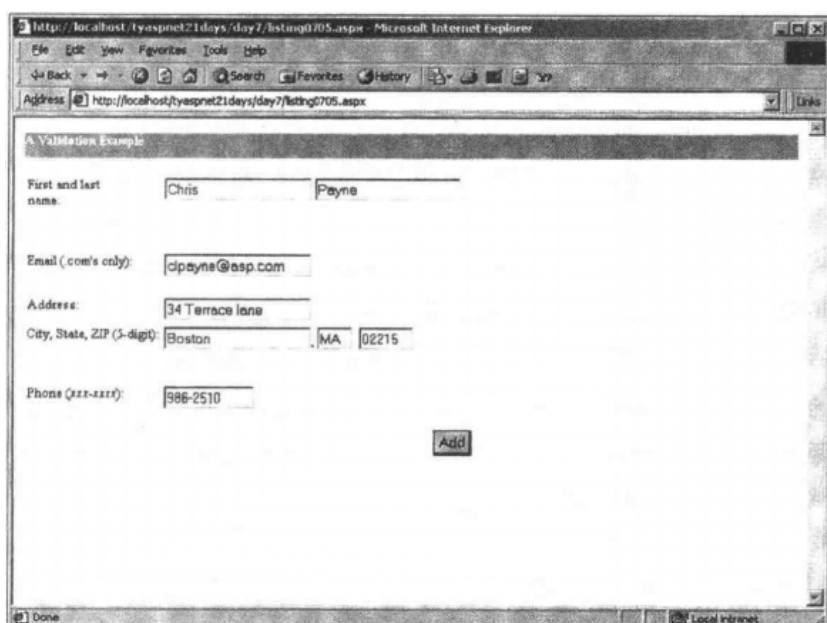


图 7.5 测试有效性验证表单，没有错误

所以，如果要检查实际值，除了使用其他Validation控件外，还必须使用RequireFieldValidator控件。

### 7.3.1 服务器上的有效性验证

客户端有效性验证是个很好的工具，解决了不少令人头疼的问题。但您仍然要在服务器端自己处理有效性验证，因为表单无效并不意味着 ASP.NET 不会执行代码，所以必须在方法中增加一些检

验步骤。

最简单的检验表单有效性的方法是检查 Page 对象的 IsValid 属性。如果所有的 Validation 控件都对输入满意，则该属性为 true；如果任何一个 Validation 控件未通过，则为 false。因此，只需检查该属性便可以决定是否执行代码。例如，清单 7.6 列出了可用于清单 7.5 的代码声明块。

**清单 7.6 测试表单的有效性**

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Submit(obj as object, e as eventargs)
5         if Page.IsValid then
6             lblMessage.Text = "Success!"
7         else
8             lblMessage.Text = "One of the fields is invalid!"
9         end if
10    end sub
11 </script>
12 ...
13 ...
14     <tr>
15         <td colspan="2" valign="top" align="right">
16             <asp:Button id="btSubmit" runat="server"
17                 Text="Add"
18                 OnClick="Submit" />
19         </td>
20     </tr>
21 </table>
22 </asp:Panel>
23 </form>
24 </body></html>

```

**分析：**第 4 行的 Submit 方法用一个属性来检查是否所有的 Validation 控件都有效。然后可能根据需要，执行或不执行代码。如果控件有效，可以将数据插入到数据库或给用户发 e-mail——可以做表单提交后您想做的任何事情。如果控件无效，则向用户显示一条消息，解释提交为什么失败——为何客户端有效性验证通过了，但服务器端没通过。

Validators 集合包含对页面中所有的 Validation 控件的引用。可以遍历该集合来确定各个控件的有效性：

```

1: sub Submit(obj as object,e as eventargs)
2:     if Page.IsValid then
3:         lblMessage.Text = "Success!"
4:     else

```

```

5:         dim objValidator as IValidator
6:         For Each objValidator in Validators
7:             if not objValidator.IsValid then
8:                 lblMessage.Text += objValidator.ErrorMessage
9:             end if
10:        Next
11:    end if
12: end sub

```

上述代码遍历 Validators 中的每个 Validation 控件，并为每一个无效控件显示错误消息。也可以使用控件的名称检查某个特定控件的 IsValid 属性。为此，必须为每个 Validation 控件指定 id 属性，前面还没有这样做过。例如，如果 e-mail 文本框的 RegularExpressionValidator 的 id 为 valEmail，则当 e-mail 没有通过有效性验证时，下面的代码将显示错误消息：

```

if not valEmail.IsValid then
    lblMessage.Text += valEmail.ErrorMessage
end if

```

### 7.3.2 禁用有效性验证

有几种方法可以禁用有效性验证，而无需删除 Validation 控件。第一种方法是将每个控件的 Enabled 属性设置为 false：

```

<asp:RegularExpressionValidator runat='server'
    ControlToValidate='txtZIP'
    Enabled='false' />

```

这样可以防止控件客户端发送任何输出而验证指定控件的有效性。

也可以将<%@ Page %>编译指令的 clienttarget 属性设置为 downlevel 来禁用客户端的所有有效性验证：

```

<%@ Page Language="VB" clienttarget="downlevel" %>

```

该命令使得 ASP.NET 认为与之交互的是一个不支持 DHTML 的老式浏览器，实际上，所有的 DHTML 事件都被禁用，包括客户端有效性验证。

**注意：**应将这种方案作为最后的手段，禁用所有的 DHTML 事件意味着根本就不会再有动态的客户端处理。

例如，ASP.NET 将不再使用<style>标记来改变 Web 控件的外观。所有的 CSS 属性也将被禁用。

### 7.3.3 正则表达式

正则表达式是一个包含特殊符号（或符号序列）的字符串，可用于匹配字符模式。例如，您可能见过使用星号进行搜索的情况。下面的字符串指的是显示当前目录下的所有文件：

```
dir *.*

```

这是一种正则表达式。星号是一个通配符，可用于表示其他任何字符。它与指定的文件名模式匹配——具体地说，是文件名为句点隔开的两个字符串的所有文件。学习正则表达式的关键在于弄清楚如何匹配文本模式。

正则表达式难于掌握，其语法非常复杂，所以本章不打算深入探讨。创建或解读表达式涉及到

学习在程序语言中哪个字符有特殊的意义以及如何指定常规字符。解读几个例子可以帮助您理解将会遇到的表达式。表 7.4 列出了正则表达式中最常用的字符。这些字符本身也是很有用的，然而只有当它们组合在一起时，正则表达式的威力才能真正地显示出来。

表 7.4 正则表达式的语言元素

字 符	含 义
常规字符	除、\$、^、[、]、(、)、*、+、?和.之外的其他所有字符都表示的是自己。换句话说，正则表达式 hello 与任何包含文本 hello 的字符串匹配，而上述特殊字符则有特殊含义。
.	与任何一个字符匹配
\$	与以字符串结尾的模式匹配。例如，Shello 与以 hello 结尾的字符串匹配。它与 I said hello 匹配，但与 Did you say hello? 不匹配。
^	与以字符串开始的模式匹配。类似于 \$。用于方括号中时，表示“非”。例如，[^aeiou]指的是除列出字符外，与其他的任何一个字符都匹配。
()	用于匹配字符串重复特定次数的情况。例如，hello(2)与字符串 hellohello 匹配，而 aye(2)与 ayeaye 匹配。
[]	用于匹配一组字符中的任何一个。例如，[aeiou]与 a、e、i、o 或 u 都匹配。也可以用短划线来指定范围，[a-z]让 a 到 z 的任一个小写字母匹配。
()	用于组合字符串，类似于用括号改变运算的优先级。
	逻辑 or。(hello) (HELLO)指的是“与 hello 或 HELLO 匹配”。
*	与 0 或若干个实例匹配。h*ello 指的是 h 后不带字符或带若干个字符，并以 ello 结尾。
+	匹配 1 个或多个前面的值。h+ello 与 hello 或 hhhhello 匹配，但与 ello 不匹配。
?	匹配 0 个或 1 个前面的值。h?ello 与 ello 或 hello 匹配，但与 hhello 不匹配。
\	转义字符。特殊字符前为反斜杠时，该字符表示的是原来的含义。例如，h*ello 与 h*ello 匹配，但与 hWello 或 hhhello 不匹配。 此外，前面为反斜杠时，某些普通字符也有特殊的意义。

解读正则表达式时，注意特殊字符将很有帮助。因为它们可以被组合，所以也应注意特殊字符之间的字符。例如表达式 href\s\*=\s\*(\"([^\"]\*)\"|(\s+))指的是 href 后面为 0 或若干个空格(\s\*)，然后是=、0 或若干个空格，然后是一个双引号(\")、0 或若干个双引号之外的字符([^\"]\*)、一个双引号或 1(或多)个空格之外的字符(s+)。该字符串可用于查找 HTML 页面中的锚点，如 href="blah: and href="hey"。

注意，在几个不必要的地方使用了转义字符(\)，例如“没有特殊意义-它只与字符匹配”。但在没有把握时，使用转义字符将更保险。

正则表达式[0-9]{5}-[0-9]{4}[0-9]{5}指的是 5 个 0 到 9 之间的字符、短划线、4 或 5 个 0 到 9 之间的任意字符。诸如数字 90210 或 83647-1422 都与之匹配，而数字 9021A 或 902 则不匹配。

正则表达式通常用于计算，以匹配文本模式。更详细的信息，请参考 .NET 框架 SDK 文档或下面的在线资源：



- <http://www.4guysfromrolla.com/webtech/regexexpressions.shtml>
- <http://www.asplists.com/asplists/aspregexp2.asp?tab=links>
- <http://www.amazon.com/exec/obidos/ASIN/1565922573/>

## 7.4 定制有效性验证

Validation 控件的可定制性非常强。您可以控制错误消息的显示方式，决定如何将错误告知用户，甚至创建自定义的 Validation 控件。ASP.NET 给开发人员提供了丰富的功能。

### 7.4.1 错误消息

本章的范例都是将 Validation 控件放在要验证的控件的后面。并不是非这样不可——Validation 控件可以放在页面上的任意地方。例如，如果想把错误消息放在一起，可将它们都放在页面的开始位置。

我们再来看一下图 7.4。由于 Validation 控件，页面的 UI 元素之间有许多空白。即使有效性验证器不显示任何消息，它们也要占用空间。可以删除每个控件后面的<br>，以消除空白，而错误消息依然按原来的方式放置，如图 7.6 所示。这种显示方式为静态的，但也可以允许动态显示，只需将 Display 属性设置为 Dynamic 即可。例如，下面的代码以动态对象的方式来显示“名”和“姓”字段的 RequiredFieldValidators：

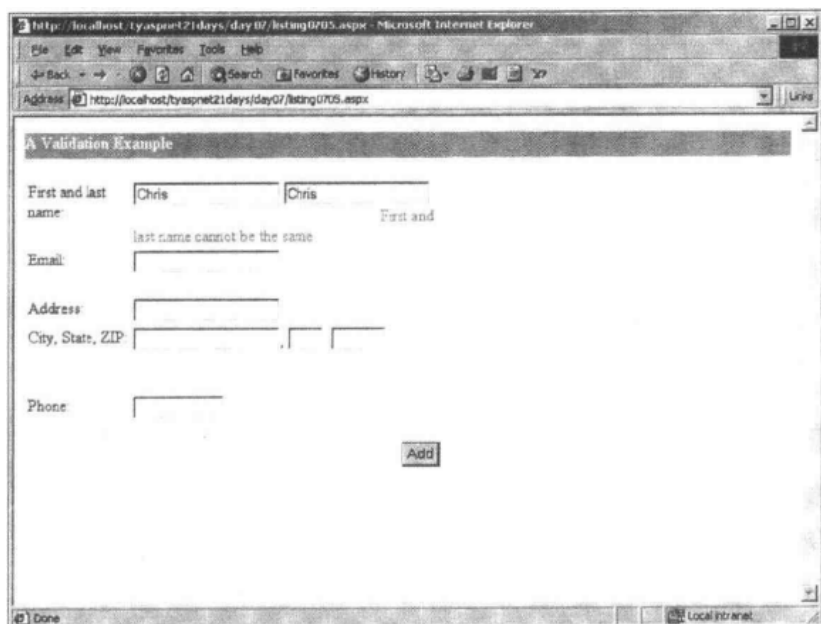


图 7.6 由于其他 Validation 控件的影响，错误消息好像不在合适的位置

```
<asp:RequiredFieldValidator runat='server'
    ControlToValidate='tbFName'
    ErrorMessage='First name required'
    Display='dynamic' />
<asp:RequiredFieldValidator runat='server'
    ControlToValidate='tbLName'
```

```

ErrorMessage="Last name required"
Display="dynamic" />
<asp:CompareValidator runat="server"
    ControlToValidate="tbFName"
    ControlToCompare="tbLName"
    Type="String"
    Operator="NotEqual"
    ErrorMessage="First and last name cannot be the same"
    Display="dynamic" />

```

只是增加了属性 `Display="Dynamic"`，给清单 7.4 中的每个控件加上该属性，其结果将如图 7.7 所示。

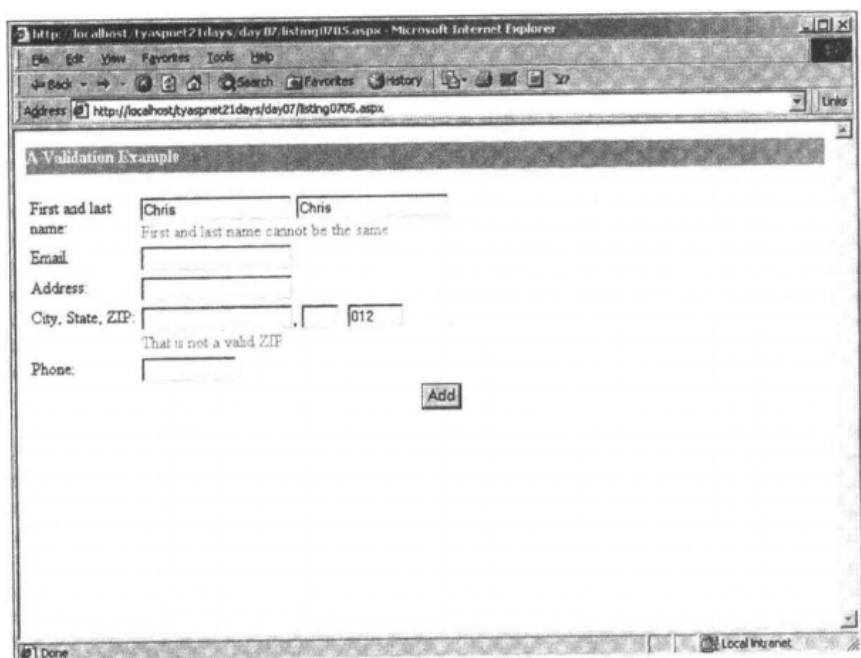


图 7.7 动态显示解决了空白的问题

空白问题现在解决了。当控件无效时，页面设置将发生变化，给错误消息腾出空间。

**警告：**这种方法可能导致页面设置上的意外变化，所以没有把握时，不要用它，或应该在完全测试过新的布局后再使用。

另外，只有支持 CSS 样式属性 `visible` 的浏览器（IE5+ 和 Netscape 4+）才支持 `Dynamic` 属性。该属性可用于选择性地对用户隐藏 UI 元素。

#### 7.4.2 显示有效性验证摘要

到目前为止，所有的例子都是将错误消息直接放在被验证的服务器控件后面（从技术上说，这些错误消息被放置在 `Validation` 控件的位置上）。在告知用户哪个控件有错方面，这很有帮助，尤其是客户端有效性验证，然而这并不总是错误消息的最佳放置位置。可以选择错误消息的放置位置：

- 嵌入（in-line）式：`Validation` 控件的位置；

- 摘要 (Summary) 式: 在一个单独的摘要中 (一个可以在支持 DHTML 的浏览器中弹出的窗口);
- 嵌入和摘要式: 各自的错误消息可以不同;
- 定制。

当您想在某个地方 (如页面的开始位置) 列出所有的错误消息时, 摘要方式非常有用。用户可能更喜欢这种方式, 而不是嵌入方式, 这取决于应用程序。

不用将所有的 Validation 控件放在一个地方, 而只需将它们留在原来的地方, 并使用 ValidationSummary 控件即可。该控件按指定的格式在页面的某一个位置显示错误消息。例如, 在清单 7.5 的顶部加上如下代码:

```
1: <asp:ValidationSummary runat="server"
2:     DisplayMode="BulletList" />
```

填写表单信息并单击 Submit 按钮后, 将得到如图 7.8 所示的结果。

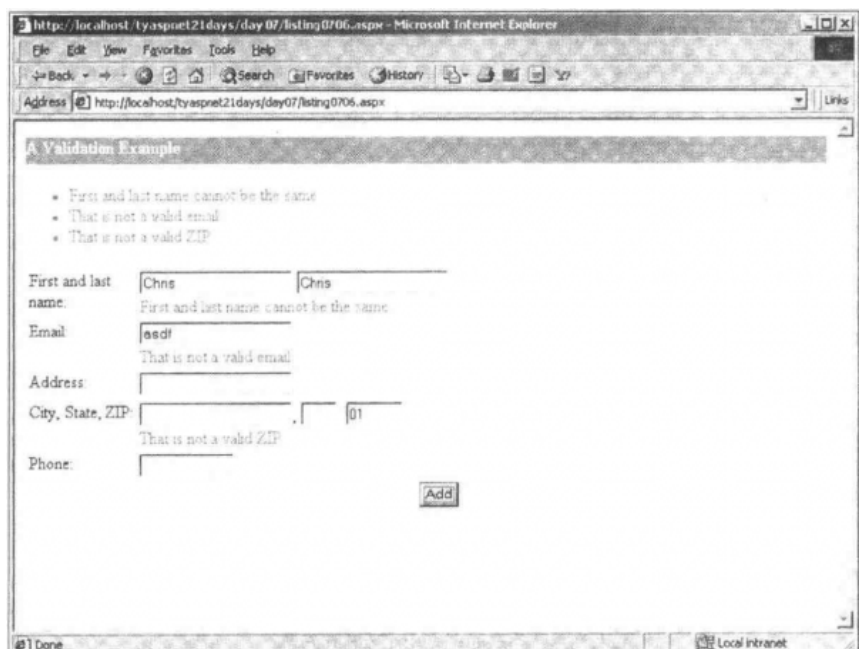


图 7.8 显示错误消息摘要

该控件与其他 Validation 控件有点不同, 它不使用属性 `ControlToValidate` 或 `ErrorMessage`, 而是取得页面中其他所有控件的错误消息, 并在一个地方显示它们。DisplayMode 可用于以项目符号列表的方式显示消息 (通过指定 `BulletList`, 如第 2 行所示), 或以段落的方式显示消息 (使用 `SingleParagraph`)。

ValidationSummary 控件显示的消息是 Validation 控件的 `ErrorMessage` 属性指定的内容。要修改嵌入信息, 可使用 `Text` 属性:

```
1: <asp:RequiredFieldValidator runat="server"
2:     ControlToValidate="tbFName"
3:     ErrorMessage="First name required"
4:     Text="Forgot first name!"
5:     Display="dynamic" />
```

现在，“姓”字段的有效性验证器将在“姓”文本框下面显示“Forgot first name!”，并在 ValidationSummary 控件中显示“First name required”。如果您不想显示嵌入错误消息，可将每个控件的 Display 属性设置为 none。

通过添加 ShowMessageBox 属性，可以用 ValidationSummary 控件创建弹出式消息框。例如，将前面的摘要代码修改为下面的样子：

```
1: <asp:ValidationSummary runat="server"
2:     ShowMessageBox="true"
3:     DisplayMode="BulletList" />
```

然后，输入无效信息并单击 Submit 按钮后，将看到如图 7.9 所示的对话框。

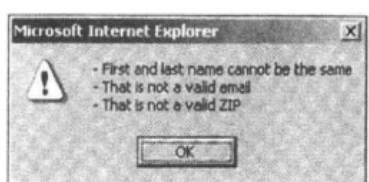


图 7.9 使用弹出式消息框提醒用户

表 7.5 列出了使用不同格式以及在不同位置显示 Validation 控件错误消息所需的参数。

表 7.5 错误消息布局参数

类 型	ValidationSummary	Validation 控件的设置
嵌入式	不要求	Display = Static 或 Dynamic ErrorMessage = 嵌入错误文本
摘要式	要求	Display = None ErrorMessage = 摘要错误文本
嵌入和摘要式	要求	Display = Static 或 Dynamic ErrorMessage = 摘要错误文本 Text = 嵌入错误之本

可以将 Validation 控件的 Text 属性设置为任何有效的 HTML 标记，这意味着可以使用图像来显示错误消息或提供到详细描述的连接！

最后，可以用所有服务器控件都有的典型样式属性来定制每条错误消息。例如：

- ForeColor：错误消息文本的颜色；
- BackColor：文本的背景颜色；
- Font：字体、字号、粗细等；
- BorderWidth、BorderColor 和 BorderStyle：错误消息周围边框的宽度和颜色；
- CSSStyle 和 CSSClass：级联式样式表中的样式设置。

更详细的信息，请参阅附录 C 中的通用属性列表。

### 7.4.3 自定义 Validation 控件

如果预定义的 Validation 控件无法满足需要, 可以创建自己的自定义 Validation 控件。例如, 有些 Web 站点要求用户注册时选择一个用户名。用户名的长度通常是固定的, 由于现有的 Validation 控件无法检验用户输入的长度, 因此必须自己动手来完成。

使用 CustomValidator 控件, 通过您熟悉的 Validation 控件语法, 可以根据需要轻松地定义任何类型的有效性验证。该控件只为自定义有效性验证提供了框架, 您必须自己创建实际的子程序。用户输入数据时, CustomValidator 将执行您创建的方法。

#### 清单 7.7 使用自定义验证器

```

1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4   sub Submit(obj as object, e as eventargs)
5       'do something
6   end sub
7
8   sub ValidateThis(obj as Object, args as _
9       ServerValidateEventArgs)
10      if len(args.Value) < 8 then
11          args.IsValid = false
12      else
13          args.IsValid = true
14      end if
15 end sub
16 </script>
17
18 <html><body>
19   <form runat="server">
20     <asp:Label id="lblMessage" runat="server" />
21     <table>
22       <tr>
23         <td valign="top">Username:</td>
24         <td valign="top">
25           <asp:Textbox id="tbUserName" runat="server"
26             /><br>
27           <asp:CustomValidator runat="server"
28             OnServerValidate="ValidateThis"
29             Display="Dynamic"
30             ControlToValidate="tbUserName"
31             ErrorMessage="The username must be 8 characters or longer"/>

```

```

32     </td>
33 </tr>
34 <tr>
35     <td valign="top">Password:</td>
36     <td valign="top">
37         <asp:Textbox id="tbPassword" runat="server"
38             TextMode="password" />
39     </td>
40 </tr>
41 <tr>
42     <td align="right" colspan="2">
43         <asp:Button id="tbSubmit" runat="server"
44             OnClick="Submit"
45             text="Submit" />
46     </td>
47 </tr>
48 </table>
49 </form>
50 </body></html>

```

**分析：**第 27 行包含一个 CustomValidator 控件，它与以前介绍的其他所有 Validation 控件类似，也有 ControlToValidate、ErrorMessage 和 Display 属性。但是它还有一个新属性：OnServerValidate，该属性用于指定处理有效性验证的事件句柄。

我们来看一下代码声明块。第 4 行创建了 Submit 方法，它不执行任何操作。第 5 章介绍过，Web 表单事件并不以特定的顺序处理。CustomValidator 控件的 ServerValidate 事件是该规则的唯一例外情况。该事件将总是首先被处理，以便其他对象可以检验页面对象的有效性。

第 8 行是 ServerValidate 事件的事件句柄。注意，其参数列表与以前介绍的不同。第二个参数 args 是一个 ServerValidateEventArgs 对象，包含相应的用户输入。可以使用 Value 属性来检索该对象包含的文本，并可以用 IsValid 属性来设置自定义 Validation 控件的有效性。

在这个函数中，您检查输入文本是否少于 8 个字符（使用 Len 函数）。如果是，则输入无效，并返回 false。填写该表单并单击 Submit 按钮后，将看到如图 7.10 所示的结果。

自定义验证器事件处理程序可按您的要求验证信息的有效性。您甚至可以用自定义有效性验证来根据服务器上的数据库验证值的有效性。

为了测试验证器，在第 4 行的 Submit 方法中加上如下代码：

```

if Page.IsValid then
    lblMessage.Text = "You passed validation!"
    'do some other processing
end if

```

当所有字段都有效时，该方法将对表单提交进行处理。但是，有效性验证处理程序将首先执行，并返回一个布尔值，指出输入是否有效。如果有效，CustomValidator 控件将其 IsValid 属性设置为 true，同时 Page.IsValid 属性也为 true。

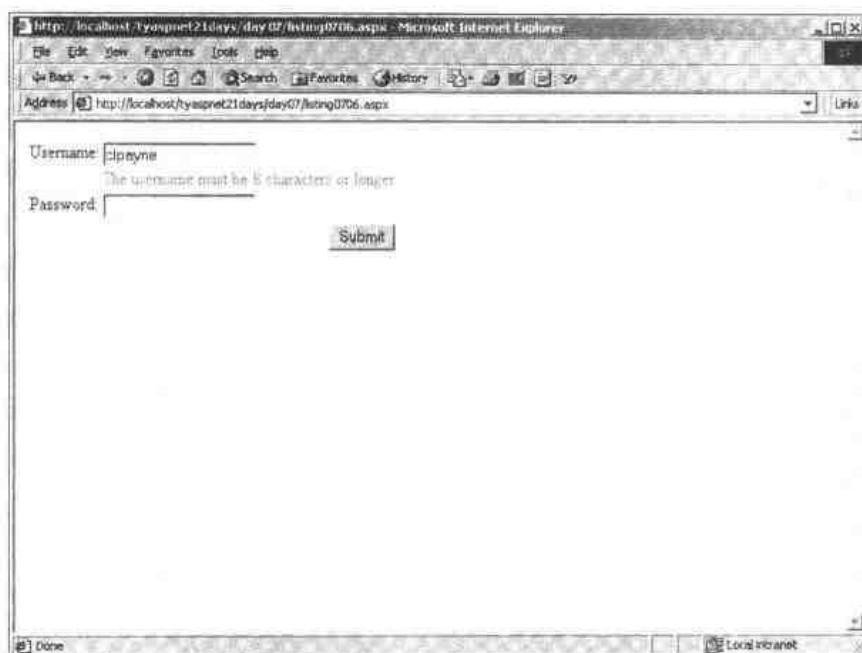


图 7.10 自定义 Validation 控件让您能够实现定制的有效性验证

CustomValidator 还让您能够在客户端进行有效性验证。为此，需要先创建一个 JavaScript 事件句柄，并将其保存在客户端。例如，下述脚本块的功能与清单 7.7 中的有效性验证程序相同，只是 JavaScript 位于客户端。

```

1 <script language="JavaScript">
2   function validateLength( oSrc, txtValue ){
3       var retValue = true; // assume ok
4       if(txtValue.length < 8){
5           retValue = false;}
6       return retValue
7   }
8 </script>

```

将该脚本块加入到清单 7.7 中的代码声明块的后面。然后，在 CustomValidator 控件中将 OnClientSideValidate 属性设置为 JavaScript 方法：

```

27 <asp:CustomValidator runat="server"
28   OnServerValidate="ValidateThis"
29   OnClientValidate="validateLength"
30   Display="dynamic"
31   ControlToValidate="tbUserName"
32   ErrorMessage="The username must be 8 characters or longer"/>

```

在浏览器中查看该页面，您将发现它和其他 Validation 控件一样，具备动态显示错误消息的功能。用户输入无效信息并移动到另一个控件后，JavaScript 方法将执行，并返回 false，同时将 CustomValidator 的 IsValid 属性也设置为 false。动态错误消息将自动显示，而不用向服务器发送信息！当然，还可以使用服务器有效性验证程序，将其作为另一次检验。

## 7.5 这不是 ASP

在传统 ASP 中，用户输入的有效性验证涉及到许多不同的检查，因此代码长而复杂，清单 7.1 列出了一个这样的例子，这通常需要许多 if 语句来针对每个控件评估多种情况。必须确保用户输入的数据（如日期、姓名和电话号码）使用了正确的格式，如果不是这样，则必须想办法使之有效。这很麻烦，而且容易出问题。

在 ASP.NET Validation 控件中，传统 ASP 中需要操心的有效性验证问题都没有了。它能够自动验证用户输入的有效性，并动态地向用户提供反馈，不再需要调试复杂的 if 语句。

本章介绍的所有功能都可以使用传统的 ASP 来实现，但要求开发人员创建所有的客户端脚本，并确保正确地显示错误消息。看一下清单 7.4 的 WebUIValidation.js 文件，便可以知道这项工作并不容易。这个文件包含许多功能和代码，而 ASP.NET 将为您完成所有这些工作，甚至可以根据客户浏览器的功能调整脚本。

Validation 控件是 ASP.NET 对 ASP 的重大改进之一，您应该充分地利用它。

## 7.6 总 结

本章介绍了使用 Validation 服务器控件验证 ASP.NET 页面的有效性的知识。这些强大的工具使得验证用户输入的有效性变得很容易。

ASP.NET Validation 控件与 Web 服务器控件非常相似。它们是在服务器上创建的，并给浏览器提供 HTML。每个 Validation 控件都监视另一个服务器控件的输入。它们依靠客户端 JavaScript 代码来提供客户端有效性验证，也可以在服务器上进行有效性验证。

每个 Validation 控件都至少应该有两个属性：ControlToValidate 和 ErrorMessage。前者指定要监视的服务器控件；后者包含未通过有效性验证时显示给用户的文本。您在“使用 Validation 控件”中学习了如何在 Web 表单中使用各种 Validation 控件。

Validation 控件还提供许多选项，供开发人员定制。可以使用 ValidationSummary 控件在同一个地方显示错误消息摘要，并有多种格式选项。该控件的 ShowMessageBox 属性还让您能够以弹出式消息框来提醒用户。

最后，本章介绍了如何使用 CustomValidator 控件来创建自定义有效性验证程序。该控件包含一个 OnServerValidate 属性，指定使用哪个服务器方法来处理有效性验证。该方法可根据需要以任何方式来验证有效性，甚至可以通过与数据库中的值进行比较来验证。

至此，第一部分便结束了！现在，您应该牢固地掌握了 ASP.NET 和 Web 表单框架的运行方式。您学习了各种可用的服务器控件，甚至可以自己创建几个。下一部分将介绍数据存取，介绍如何与数据库、XML 文件以及其他格式的数据交互，在应用程序中加入大量的功能。

## 7.7 问与答

问：必须同时在服务器端和客户端都进行有效性验证吗？

答：不。并不需要在表单提交后在服务器上测试有效性。通常，客户端的有效性验证将使所有



的用户输入都是正确的。但有时候为了更保险，尤其是客户端有效性验证依靠机制不同于 ASP.NET 的 JavaScript 时，在服务器上进行有效性验证是值得的。

问：动态错误消息没有显示出来，问题在哪里？

答：确保至少将表单提交了一次。错误消息将在首次提交表单时出现（即如果输入无效的话）。在首次提交后，动态消息将显示出来，并在离开该 UI 元素后自动消失。

另外，还应该确保使用较新的、支持 DHTML 的浏览器版本（IE 或 Netscape 4+）。

问：何时应该使用 Validation 控件？

答：需要时便使用！对于重要的应用程序而言，有效性验证都是必不可少的，而 Validation 控件使得 ASP.NET 中的有效性验证非常容易。

应该说，Validation 控件确实增加了页面的开销、使文件更大，因此在服务器和客户之前发送文件时需要的时间更长，用户可能会觉得性能有所下降。另外，要获得 Validation 控件的所有好处，客户端必须支持 JavaScript 和 DHTML。

当性能至关重要，而带宽又非常有限或知道有些客户使用的是老版本的浏览器时，则使用传统的有效性验证方式将是明智的。也就是说，在服务器端使用 if 和 case 语句。

## 7.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 7.8.1 小测验

1. 有哪 5 种 Validation 控件？
2. 判断正误：Validation 控件是在客户端创建的。
3. 每个 Validation 控件都必须包含哪两个属性？
4. 参数 Clienttarget=downlevel 起什么作用？
5. 假设有两个服务器控件：tbName 和 tbAge，下面的验证器有用吗？

```
<asp:CompareValidator runat="server"
    ControlToValidate="tbName"
    ValueToCompare="tbAge"
    ErrorMessage="Error!" />
```

6. 为给 CustomValidator 控件指定事件句柄，需要设置什么属性？

### 7.8.2 练习

创建两个列表框控件，它们包含相同的项目（如“red”、“green”、“blue”和“brown”）。然后创建一个 Validation 控件，对这两个控件中被选中的值进行比较，如果它们不同，则显示一条错误消息。



# 第一部分 复 习

祝贺您阅读完了第一部分的内容！至此，您已经学习了许多关于 ASP.NET 和 .NET 框架的知识。您还学习了如何使用 Visual Basic、HTML 和 ASP.NET 服务器控件来创建很有用的 ASP.NET 页面以及如何充分利用 Web 表单框架。您已经为创建自己的应用程序做好了准备。

## 附加项目 1

在本书前三部分的结尾，您将使用在相应部分学到的技能来创建一个应用程序，而且每一部分都将对同一个例子进行扩展，因此，最后将创建出一个功能完备的、复杂的应用程序。这里将创建该应用程序的基础部分，在“附加项目 2”中将添加数据功能，而在“附加项目 3”中将加入一些更高级的特性。

强烈建议您要完成这些附加项目，因为使用真实的范例将让您进一步熟悉 ASP.NET 的概念。我们开始创建该应用程序吧。

## 一个银行业应用程序

第一个附加项目将创建一个可在线使用的银行业应用程序。该程序允许用户登录并检查其账户余额、支付账单等。但是现在只使用学过的服务器控件来创建 UI。尽管这个程序在后边将会相当复杂，但它只依赖于这里架设的几个页面。

银行业应用程序应该具备下述功能：

- 让用户可以安全地访问其账户；
- 允许用户查看其交易记录和账户余额；
- 允许用户在线支付账单。

这些内容描述了应用程序的范围——要求它能够做什么。预先对目标进行规划有助于创建应用程序。

我们来确定需要创建的页面。首先，用户需要一个登录页面，该页面对用户进行认证并将它带到其账户。其次，需要创建一个账户摘要页面，列出交易记录和结余。最后，需要创建一个让用户在线支付账单的页面。因为您想为网站提供标准的用户界面，因此需要以用户控件的方式在每个页面中使用相同的页眉和导航栏。

- 登录页面: login.aspx;
- 账户摘要页面: account.aspx;
- 支付账单页面: bills.aspx;
- 两个用户控件: header.ascx 和 nav.ascx。

注意: 显然, 该应用程序可以包含更多的页面。但是因为您刚起步, 因此让它相对简单一些。

但是您应该可以随意地使用该应用程序进行练习。您可以给它添加学过的许多特性, 因此不要害怕对该应用程序进行改进。

## 用户控件

当前, 菜单或导航控件相当简单。我们先创建一个表格, 其中包含到其他页面的链接, 如清单 BP1.1 所示。

### 清单 BP1.1 基于用户控件的菜单

```
1 <table align="left" width="150" height="350"
2   bgcolor="#cccc99">
3 <tr>
4   <td align="right" valign="top" bgcolor="Black">
5       <a href="account.aspx"> Account Summary </a><p>
6       <a href="bills.aspx">Pay Bills</a><p>
7       <a href="Logout.aspx"> Sign Out</a><p>
8   </td>
9 </tr>
10 </table>
```

将该文件保存为 nav.ascx。再创建一个显示标准页眉的用户控件, 如清单 BP1.2 所示。该控件仅包含一个 ASP.NET 标签控件, 用来显示标准页眉。

### 清单 BP1.2 页眉用户控件

```
1 <asp:Label id="Header" runat="server"
2   text="ASP.NET Banking Center"
3   Height="25px"
4   Width="100%"
5   BackColor="#cccccc"
6   ForeColor="Black"
7   Font-Size=20
8   Font-Bold="False"
9   Font-Name="Arial"
10  BorderStyle="outset" />
```

将该文件保存为 header.ascx。接下来创建 ASP.NET 页面。

## 登录页面

登录页面非常简单, 只需给用户提供几个文本框控件, 让他们输入信息, 然后创建一个方法来

确定他们是否为合法用户。

有效性验证方法可以使用 ASP.NET 的安全系统非常方便地完成, 但是因为他还没有学习这方面的知识, 因此必须用其他方法来实现。将来学习 ASP.NET 安全系统后, 可以重新复习这部分内容。

另外, 还需要确保开始创建该应用程序时就使用一致的 UI, 因此应包含页眉用户控件, 而不要菜单用户控件, 因为还不能允许用户访问它列出的页面。我们来看清单 BP1.3 中的代码。

清单 BP1.3 login.aspx: 认证用户

```

1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ASPNETBank" TagName="Header" src="header.aspx" %>
3
4 <script runat="server">
5
6     *****
7
8     login.aspx: Logs users in
9
10    *****
11
12    *****
13    ' when user clicks submit button, verify that they are a
14    ' valid user. If they are, log them in, and set a cookie
15    ' with their user name, and redirect to account.aspx.
16    ' Otherwise display error message
17    *****
18    sub Submit(obj as Object, e as EventArgs)
19        if tbUsername.text = "clpayne" and _
20            tbPassword.text = "pass" then
21            Response.Cookies("FirstName").Value = tbUsername.text
22            Response.Redirect("account.aspx")
23        else
24            lblMessage.Text = "<font color=red>Sorry, invalid" & _
25                " username or password!</font><p>"
26        end if
27    end sub
28 </script>
29
30 <html><body>
31
32     <ASPNETBank:Header runat="server" />
33     <font face="arial">
34     <p>

```

```
34 Welcome to the ASP.NET Banking Center. Please enter your
35 username and password to gain access to your account
36 information.<p>
37
38 <form runat="server">
39     <asp:Label id="lblMessage" runat="server" />
40     <table>
41         <tr>
42             <td width="75" rowspan="3">&nbsp;&nbsp;&nbsp;</td>
43             <td width="50" valign="top">
44                 <font face="arial">Username:</font>
45             </td>
46             <td width="50" valign="top">
47                 <font face="arial">
48                     <asp:Textbox id="tbUsername" runat="server" />
49                 </font>
50             </td>
51         </tr>
52         <tr>
53             <td valign="top">
54                 <font face="arial">Password:</font>
55             </td>
56             <td valign="top">
57                 <font face="arial">
58                     <asp:Textbox id="tbPassword"
59                         TextMode="password" runat="server" /><p>
60                 </font>
61             </td>
62         </tr>
63         <tr>
64             <td align="right" colspan="2">
65                 <font face="arial">
66                     <ASP:Button id="btSubmit" runat="server"
67                         onClick="Submit"
68                         text="Submit" />
69                 </font>
70             </td>
71         </tr>
72     </table>
73 </form>
```

```
74 </font>
75 </body></html>
```

**分析：**第 2 行注册了页眉用户控件——您应该还记得在第 6 章介绍过的这种方式。现在可以像使用常规 ASP.NET 服务器控件那样，在页面中使用页眉用户控件。该页面有一个 Submit 方法，当用户单击第 66 行的 Submit 按钮时将调用它。

因为在这里没有使用数据库，您可以偷点懒，在第 19 和 20 行以硬编码的方式使用用户名和密码，在最后的应用程序中，可千万别这样做。实际上，在下一部分介绍过如何使用数据库后，将修改它。但是现在这样做还是可以的。

然后，第 19 行的 if 语句核实用户提供的证件是否正确。如果正确，则设置一个值为用户名的 cookie，供以后引用。然后把用户重定向到其账户页（接下来将创建）。如果用户不合法，则使用第 24 和 25 行的标签控件显示一条消息。

该页面是应用程序入口页。在进入银行中心之前，用户必须通过该页面。图 BP1.1 显示了用户输入错误的证件后出现的结果。

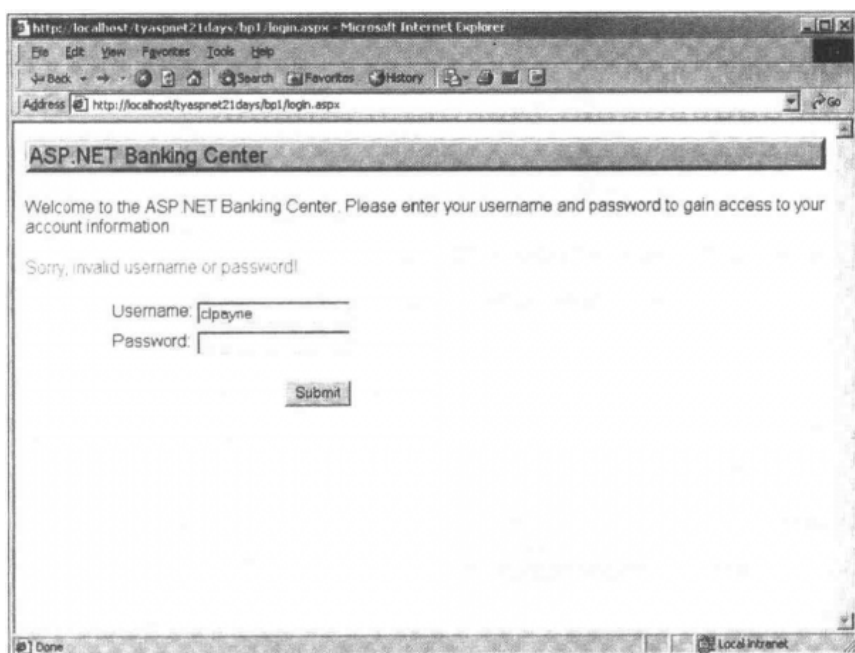


图 BP1.1 验证用户证件的有效性

## 账户页面

账户页面显示用户的交易记录及其当前的结余。交易记录将通过数据库和数据绑定服务器控件来处理（参见“附加项目 2”），所以现在先不理睬它们。留给以后做的工作很多，这里只完成了几个 UI 服务器控件以及标准页眉和菜单用户控件。

您可能想给用户显示定制的欢迎消息，因此需要访问登录页面中创建的 Cookie：

```
Request.Cookies("FirstName").Value
```

该命令检索了用户 cookie 变量 FirstName 中的值（将其作为一个字符串），这和使用会话变量来存储类似。

清单 BP1.4 列出了该页面的代码。

## 清单 BP1.4 account.aspx: 用户的账户摘要信息

```

1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ASPNETBank" TagName="Header" src="header.ascx" %>
3 <%@ Register TagPrefix="ASPNETBank" TagName="Menu" src="nav.ascx" %>
4
5 <script runat="server">
6     '*****
7     '
8     ' Account.aspx: lists account summaries and current balances
9     ' for the current user
10
11     '*****
12
13     '*****
14     ' When the page loads, display a welcome message in the
15     ' 'WelcomeMsg' label
16     '*****
17     sub Page_Load(obj as object, e as eventargs)
18         If (Request.Cookies("FirstName") Is Nothing) Then
19             lblWelcomeMsg.Text = "Welcome <b>" & _
20                 Request.Cookies("FirstName").Value & "!</b>"
21         End If
22     end sub
23 </script>
24
25 <html><body>
26     <ASPNETBank:Header runat="server" />
27     <table>
28     <tr>
29         <td valign="top" width="150">
30             <ASPNETBank:Menu runat="server" />
31         </td>
32         <td width="550" valign="top">
33             <font face="arial">
34
35                 <form runat="server">
36                     <table>
37                     <tr>
38                         <td width="10">
39                             &nbsp;

```

```
40         </td>
41         <td width="100%">
42             <font face='arial'>
43                 &nbsp;&nbsp;&nbsp;<p>
44                 <asp:Label id="lblWelcomeMsg" runat="server"/><p>
45
46                 Your account balance is <b>$945.31</b><p>
47             </font>
48         </td>
49     </tr>
50 </table>
51 </form>
52 </font>
53 </td>
54 </tr>
55 </table>
56 </font>
</body></html>
```

分析：第 2 和第 3 行注册了两个用户控件。然后在 Page\_Load 事件中设置了第 44 行的标签的 Text 属性。您使用了在 login.aspx 中设置的 cookie 中的值，它保存的是用户的登录名。然后使用标签向用户显示一条个性化的问候消息。图 BP1.2 显示了该页面。

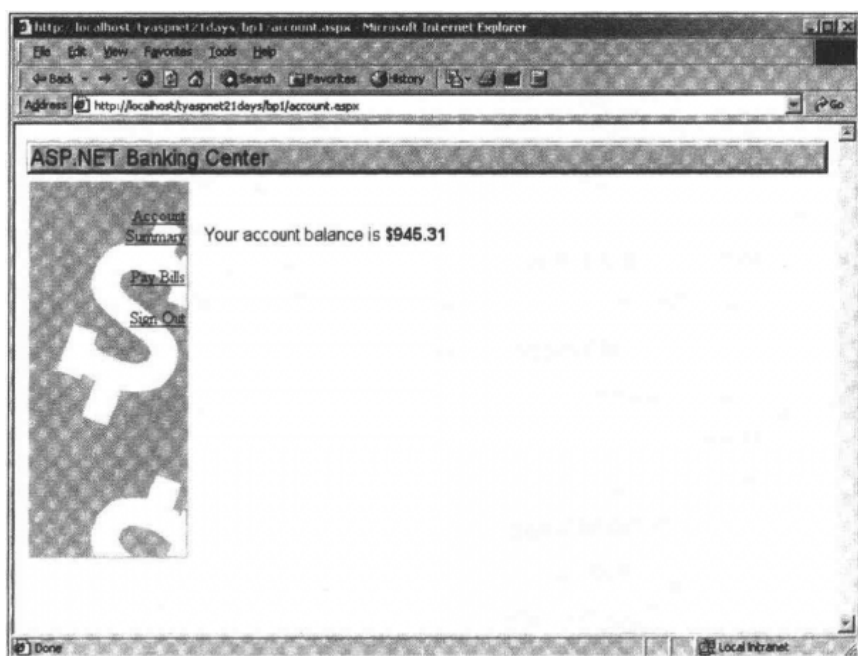


图 BP1.2 账户摘要页面

该页面完成的其他工作很少。第 46 行包含一个临时值，在“附加项目 2”中将使用数据库中



的值将它替换掉，而该区域中的交易记录也将是动态信息。

### 账单支付页面

该应用程序将允许用户在线支付账单（当然不是真的，因为这里全是假钱）。用户将提供收款人的信息，应用程序将自动转账。最后，您还要加上一个功能，使付款人可以从任何位置完成交易（从其账户中划出资金）！这有些超前了，我们来看一下清单 BP1.5 包含的该页面的代码。

#### 清单 BP1.5 bills.aspx: 在线支付

```

1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ASPNETBank" TagName="Header" src="header.ascx" %>
3 <%@ Register TagPrefix="ASPNETBank" TagName="Menu" src="nav.ascx" %>
4
5 <script runat="server">
6     '*****
7     '
8     ' Bills.aspx: Allows user to pay bills online
9
10    '*****
11 </script>
12
13 <html><body>
14     <ASPNETBank:Header runat="server" />
15
16     <table>
17     <tr>
18         <td valign="top" width="150">
19             <ASPNETBank:Menu runat="server" />
20         </td>
21         <td width="550" valign="top">
22             <font face="arial">
23                 <asp:Label id="lblMessage" runat="server"/>
24                 <form runat="server">
25                     <table>
26                     <tr>
27                         <td width="10">&nbsp;</td>
28                         <td width="100%">
29                             <font face="arial">
30                                 &nbsp;<br><p>
31                                 Welcome to the bill paying service!
32                                 Please fill out the information below
33                                 and hit "Pay that Bill!" to automatically

```

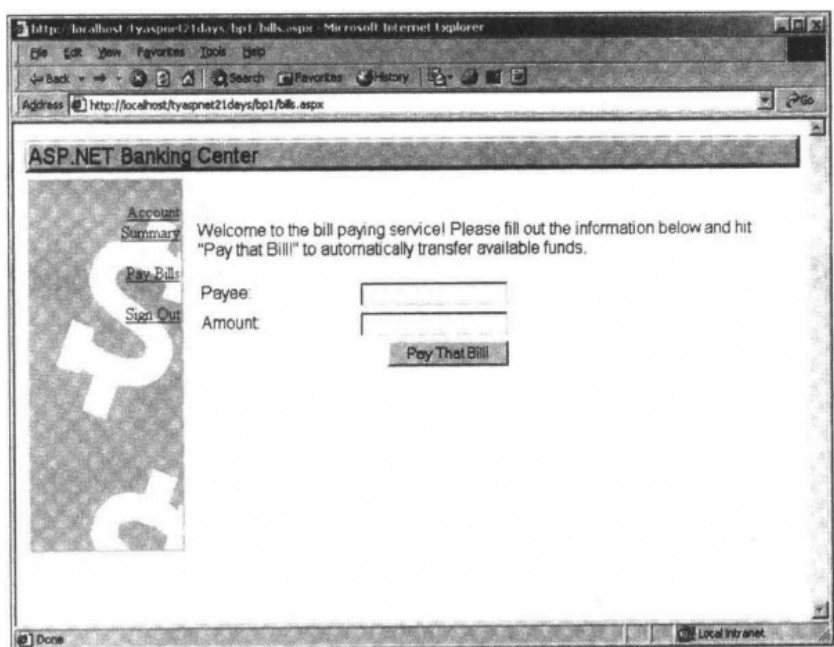
```

34         transfer available funds.<p>
35
36     <table>
37     <tr>
38         <td width='150' valign="top">
39             <font face="arial">Payee:
40         </td>
41         <td width="100" valign="top">
42             <font face="arial">
43                 <asp:textbox id="tbPayee" runat="server" />
44             </td>
45         </tr>
46     <tr>
47         <td valign="top">
48             <font face="arial">Amount:
49         </td>
50         <td valign="top">
51             <font face="arial">
52                 <asp:textbox id="tbAmount" runat="server" />
53             </td>
54         </tr>
55     <tr>
56         <td valign="top" colspan="2" align="right">
57             <font face="arial">
58                 <asp:button id="btSubmit" runat="server
59                     text="Pay That Bill!"
60                 />
61             </td>
62         </tr>
63     </table>
64 </font>
65 </td>
66 </tr>
67 </table>
68 </form>
69 </font>
70 </td>
71 </tr>
72 </table>
73 </font>

```

```
74 </body></html>
```

**分析：**该文件与清单 BP1.4 中的 `account.aspx` 类似，它们的用户控件和基本布局相同。您只是添加了几个文本框（供用户输入数据）以及一个 Submit 按钮（供用户提交数据）。目前，Submit 按钮不做什么事情。该清单的输出如图 BP1.3 所示。



图BP1.3 用户可以在线支付账单

## 总 结

祝贺您创建了第一个完整的 ASP.NET 应用程序！通过这个项目，您懂得了如何应用这一部分学到的知识以及如何为完成复杂的 ASP.NET 应用程序做好基础工作。

该附加项目旨在介绍开发应用程序的各个阶段。首先是简单的设计阶段，然后规划所有的页面和功能。该项目也有助于提供 ASP.NET 技能，以便开发其他项目时能够得心应手。

后面的附加项目将对该应用程序进行改进。在“附加项目 2”中，将在应用程序中加入数据库功能，使之真正是动态的。在最后一个附加项目中，将介绍如何利用“组件化技术”和 Web 服务创建功能强大的、健壮的、真实的应用程序。但是首先您需要阅读第二部分的内容。

## 第二部分

第 8 章 创建数据库



# 数据存取和处理

第 8 章 创建数据库

第 9 章 在 ASP.NET 中使用数据库

第 10 章 与 ASP.NET 通信

第 11 章 在 ASP.NET 中使用 XML

第 12 章 应用高级数据技术

第 13 章 Web 服务器上的文件读写

第 14 章 使用 ASP.NET 改良后的缓存功能



## 第 8 章

# 创建数据库

本章介绍数据库的基础知识,包括什么是数据库、如何创建数据库以及如何使用数据库。本章不是有关数据库的参考大全,但给您提供了数据库的背景知识,有助于在以后的章节中创建应用程序。

数据库能够以高效的方式存储大量的数据,因此是一个极为有用的工具。数据库让您能够查看、修改和使用各种类型的信息,并使 ASP.NET 应用程序的用途更为广泛,在接下来的几章中,您将体会到这一点。

本章学习以下内容:

- 什么是数据库;
- 关系型数据库和平面文件 (flat-file) 的区别;
- 如何使用 Access 2000 和 SQL Server 2000 来创建数据库;
- 如何使用结构化查询语言 (SQL) 来选择、插入、更新和删除数据;
- ASP.NET 与数据库的交互。

### 8.1 什么是数据库

**新术语:** 数据库是存储数据的仓库,仅此而已。我们不深入研究数据库,因为数据库可以是任何一种数据存储,从简单的文本文件到分布在多台计算机中的产品目录清单。数据库是计算的最重要的方面之一,这也解释了数据库在 Web 开发中的重要性。

ASP.NET 的一个很重要的应用是向数据库进行交互。Microsoft 认识到了这一点,因此使用 ASP.NET 可以很轻松地与数据库交互。

当前市面上有很多商业数据库应用程序,包括 Microsoft Access、Oracle、Informix 以及 DB2。这些应用程序的数据存储机制各不相同,但数据库概念却是相同的。

**新术语:** 在数据库的术语中,表 (table) 是对数据的一种二维表示。另外,数据库的形状和大小各不相同,所以表也是形形色色,不尽相同。图 8.1 是一个在 Microsoft Notepad 中创建的简单数据库表。

图 8.1 中的表有两列: Cars 和 Manufacturers,前者包含汽车的型号,后者包含制造商的名称。因此每列描述了信息的一种属性。而行含有真正的数据,对应于一列或多列——每一行称为一条数据记录,它详细地描述了一个条目。行中各列被称为字段,例如,图 8.1 有 11 行 (或记录),每一

行含有两个字段：Cars 和 Manufacturers。



图 8.1 数据库表的形式各种各样，包括在 Notepad 中创建的简单的文本文件

**提示：**为便于理解，可以将字段看作是行和列的交叉。

每条记录都是一个唯一性的条目，可以根据需要使用任意多个字段来描述该条目。例如，可以在图 8.1 中添加以下几列：miles（里程）、the year（制造年度）、the last oil change（上次换油的时间）

图 8.1 的数据库只包含了一个表，但是数据库通常含有多个表，每个表描述一种对象。多个表有助于将信息更加逻辑、有效地联系起来。例如，假设您想将以下有关汽车和制造商的信息存储在表中：

- Make;
- Model;
- Mileage;
- Manufacturer;
- Manufacturer's address;
- Manufacturer's phone.

虽然可以在一个表中存储上述所有信息，但将重复很多信息。例如，Geo 只有一个地址和一个电话号码，那么为什么要在每种汽车中重复这些信息呢？如图 8.2 所示：

Cars				
Make	Year	Mileage	Manufacturer	ManufacturerAddress
Prizm	1993	193,000	Geo	Dayton, OH
Metro	1995	34,005	Geo	Dayton, OH
Metro	1990	340,507	Geo	Dayton, OH

图 8.2 表中的重复信息

**新术语：**通常，这种信息存储方式称为平面文件（flat-file）数据库，其中对应于多个表的信息被存储在一个大型表中。这种数据库中有很多重复的信息，占用了大量的空间和时间，而且平面文

件数据库容易出现错误。假设您不得不手工重复输入同一个地址,这时很可能出错。对于这些数据,一种更有效的存储方式是使用两个表,如图8.3所示。

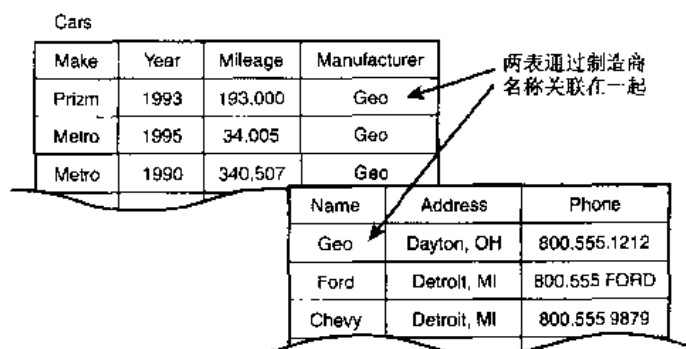


图8.3 将冗余信息移到另一个表中

**新术语:** 使用这种方式, Cars表只需记录制造商的名称,然后这个表的 name 字段再同制造商表中的 name 字段关联起来,Manufacturers表存储了制造商的相关信息。这种表之间通过关系连接起来的数据库称为关系型数据库 (related database)。

**新术语:** 同平面文件数据库相比,关系型数据库有许多优点,包括存储效率更高、搜索速度更快以及分组更为合理等。本书不详细介绍关系型数据库的设计方法,而对被称为规范化 (normalization) 的技术做全面探讨,这种技术致力于如何正确地将数据库表分成不同的对象。通常,如果发现表中的某种数据重复多次,则应考虑为这些数据创建一个单独的表,并在这两个表之间建立一种关系。

### 8.1.1 关键字 (keys) 和约束 (constraints)

**新术语:** 正如前面所述,将数据库中重复的数据降低到最少十分重要。最重要的是,您不希望出现完全相同的记录,因为这完全是浪费空间。因此,我们引入主关键字 (primary key) 来防止出现相同的记录。假设有一个关于家庭成员的数据库,包括 name、address 和 birthdate 三个字段,则可以将 name 作为主关键字,这样就不会出现两条记录的姓名相同的情况。

在控制数据库方面,这十分有用,但这种方案并不太现实,因为许多人的姓名相同。此外,还可以将多个字段定义为主关键字,这将在本章后面的“创建数据库”一节中介绍。在这种情况下,如果主关键字包括 name、address 及 birthdate 三个字段,则数据库将不允许在表中输入三个字段的信息都完全相同的两条记录。

**新术语:** 同前面的例子相比,使用多个关键字的作用更好,但这仍不能防止错误。或许您找到了很久前失散的堂妹,其姓名和出生日期与您妹妹完全相同,并且她搬到了您家里来往。这样数据库中就有两条姓名、住址、出生日期相同的记录。要保证数据库中不会有两条相同记录,一种更好的方法是再加入一个字段,该字段与现有字段毫无关系,并能保证记录的唯一性。这种字段称为标识字段 (identity column),因为它可以用来唯一地、精确地标识数据库中的记录。最常用的标识字段是递增的数字——每条新记录的标识数字都比前一条记录大 1。

**新术语:** 确保表中的每条记录都是唯一的后,需要创建到另一个表的链接。这是通过在不同表的字段之间建立某种关系来实现的,其中每个字段都包含相同的信息。在图8.3中,两个表通过 Cars表中 Manufacturer 字段和 Manufacturers表中的 Name 字段连接起来。尽管这两个字段的名称不同,



但包含的信息相同——都是制造商的名称。用于链接两个表的字段称为外关键字（foreign keys）。

Cars 表有一个与 Manufacturers 表链接的外关键字，这是一种多对一的关系。也就是说 Cars 表中可以有很多条记录与 Manufacturers 表中的同一条记录相关。例如，Geo 公司生产的汽车很多，但是只有一个公司叫 Geo。同理，也存在一对多和一对一的关系。

### 8.1.2 数据库通信标准

**新术语：**现在市面上有很多种不同的数据库系统，如 Microsoft Access、Oracle 及 Informix。如果每一种数据库系统都使用一种独特的协议来访问数据库信息，则开发人员在开发适用于各种数据库系统的数据驱动应用程序时将非常困难。幸运的是，大多数商用数据库系统遵循开放数据库互连标准（Open Database Connection standard, ODBC）。该标准是 Microsoft 制定的，为访问数据库提供了一个通用接口，这样开发人员就无需为访问不同的数据库编写不同的代码了。遵循 ODBC 标准的数据库称为 ODBC 顺应（ODBC-compliant）数据库。

不幸的是，ODBC 有其局限性。开发人员指责 ODBC 不易使用，因为 ODBC 需要低级应用程序代码，而这些代码随数据库而异。

为解决这个问题，Microsoft 随后开发了 OLE DB——一个基于 COM 的 Windows 数据访问对象，可用于访问各种不同类型的数据。OLE DB 现在是数据库连接的标准。OLE DB 也包含 ODBC 标准，所以几乎任何商用数据库都支持该标准。

### 8.1.3 何时应使用数据库

数据库是一种高效的、存储大量永久型数据的方法。当然管理这些信息是以降低性能为代价的。数据库在存储大量复杂的相关数据方面的效率非常高，但如果只想临时存储少量的数据，如访问者的姓名或 e-mail，则使用 cookie 或 session 变量更合适。

如果在所存储的信息中，不同的数据间具有复杂的关系，则数据库是最佳选择。数据库的另一个优势是它在一个地方集中存储了大量的数据。大多数需要存储数据的大型软件都使用了某种数据库。

在 ASP.NET 页面中应用数据库既简单又自然。稍后您将知道在 ASP.NET 页面中访问数据库是多么的简单。

## 8.2 创建数据库

掌握数据库的基础知识后，我们开始创建第一个数据库。首先，我们将使用 Microsoft Access 2000 来创建一个数据库，然后再使用 Microsoft SQL Server 2000 来创建一个数据库。您将发现这两种数据库系统之间有许多相似（不同）的地方。本书中的大多数例子都使用 Access，因为 Access 更易于学习。尽管在实际应用中，SQL Server 更常用，但由于它比 Access 复杂，所以这里只对它做简要的讨论。

在后面的章节中，将使用这里创建的数据库，因此我们来设计一些有用的东西。为方便初学者，这里创建一个 user（用户）表来存储以下信息：

- First name（姓）；
- Last name（名）；
- Address（地址）；
- City（城市）；

- State (州);
- Zip code (邮政编码);
- Phone number (电话).

这是一个典型的用户数据库，可用于很多不同的应用程序中。要在 Access 中创建数据库，启动 Access，并从 File 菜单中选择 New，打开如图 8.4 所示的窗口。选择 Blank Database (新数据库)，取名为 banking.mdb，然后保存到 C:\ASPNET\data 目录下（可将其保存在任何目录下，只是要知道本书的范例将引用该目录）

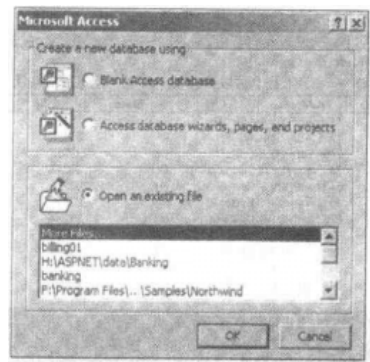


图 8.4 新建一个数据库

您会看到下面的选项：在 Design View (设计视图) 中创建表、使用向导创建表、通过输入数据来创建表。选择 Design View，并输入字段。在 Field Name (字段名) 字段中输入 FirstName，然后在数据类型字段中选择 Text (文本)。Description 字段是可选项，可以在该字段中输入新建字段的描述信息，如“User's first name”输入前面的项目符号列表描述的其他字段后，将得到类似图 8.5 的结果

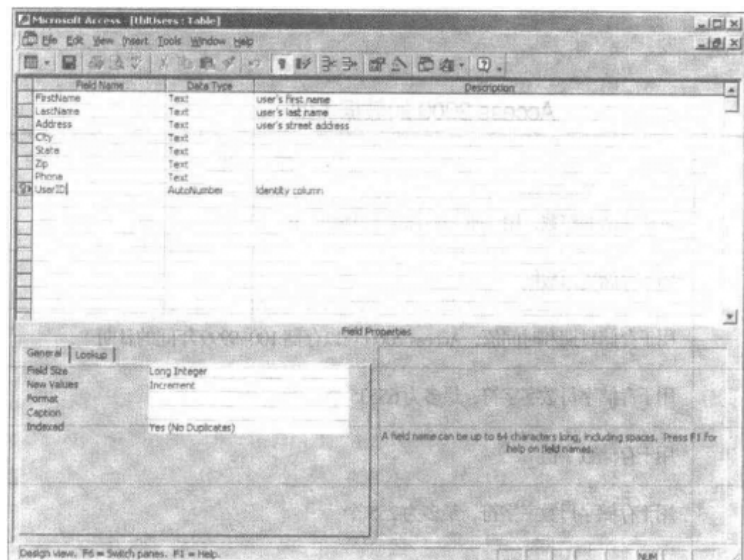


图 8.5 输入表字段的信息

**注意：**可用的选项随您所使用的 Access 版本而异。

最后再加入一个名叫 UserID 的字段，这是标识字段。将其数据类型设置为 Autonumber，并在

其名称旁边的黑色箭头上单击鼠标右键。从菜单中选择 **Primary Key**，请注意这时主关键字的标志将出现在名称的旁边。接着从 **File** 菜单中选择 **Save** 命令，保存这个表，并命名为 **tblUsers**。选择 **File** 菜单中的 **Close** 命令，关闭 **tblUsers**。

**提示：**要将多个字段定义为主关键字，可按下 **Ctrl** 键，选中各个字段，然后单击鼠标右键，并选择 **Primary Key**。

所有的工作便完成了！现在，只需双击菜单中的 **tblUsers** 选项，便可以添加数据。注意，在输入数据时，无需为 **UserID** 字段输入值。该字段的值是自动递增的，如图 8.6 所示。

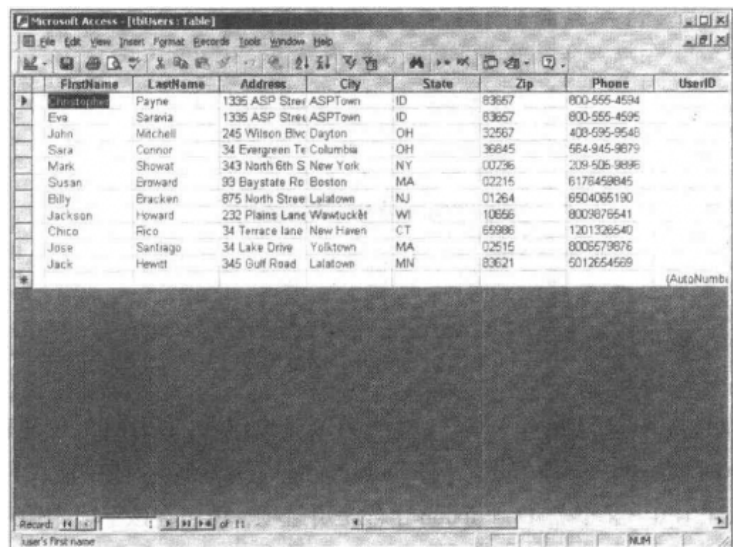


图 8.6 在新表中输入数据

表 8.1 列出了 Access 2000 提供的各种数据类型，所有数据库系统都提供类似的数据类型，虽然名称可能不同。

表 8.1 Access 2000 的数据类型

数据类型	描述
Autonumber	自动递增的整数，用于唯一地标识表中的记录
Currency	用于存储货币数据
Date/Time	用于存储日期和时间值，Access 2000 可以存储 100~9999 年间的日期
Memo	用于存储字母数字字符，最多为 65535 个
Number	用于存储数字值
Text	用于存储字母数字字符，最多为 255 个
Yes/No	用于那些只可能是两种值之一的字段（类似于 bit 或 boolean 类型）

SQL Server 2000 更复杂些，但是步骤基本相同。Access 能够满足要求，但 SQL Server 通常用于那些对安全性和稳定性要求更高的大型数据库。

现在让我们在 SQL Server 2000 中创建 dbUser。首先,从 Start (开始) 菜单下的 SQL Server 2000 菜单组中打开 Enterprise Manager。如果一切设置正确,将看到一个 Windows 资源管理器(类似于导航系统),其中最上方是 Microsoft SQL Server,服务器的名称位于接下来的第二行。展开 databases 文件夹,如图 8.7 所示,并从 Action 菜单中选择 New Database。在弹出的对话框中输入名称 Banking。现在,只需要接受 General 选项卡中的默认选项,单击 OK 即可。SQL Server 将为您创建数据库和事务日志。这时可以在 Database 文件夹中看到 Banking,展开 Banking 文件夹,选择 Tables 节点。

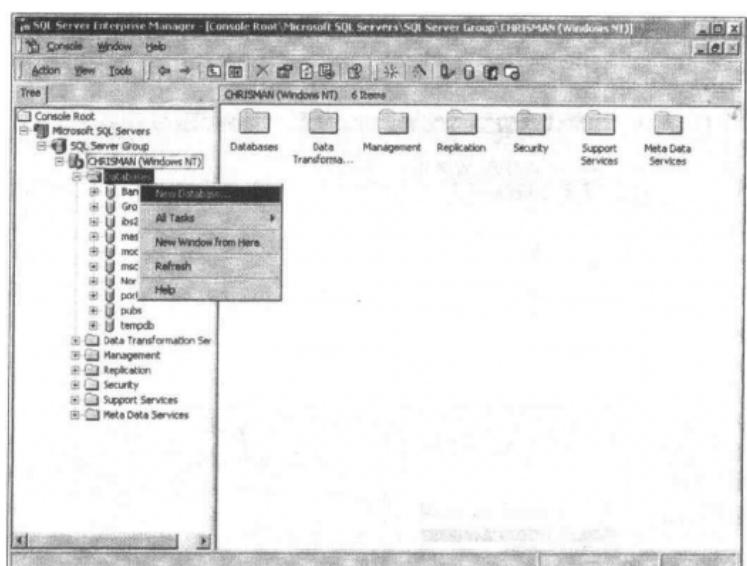


图 8.7 展开 Database 文件夹,从 Action 菜单中选择 New Database

SQL Server 已为您创建了很多表。这些是系统表,SQL Server 使用它们来记录您在数据库中加入的条目,现在不用理会这些表。从 Action 菜单中选择 New Table,出现一个与 Access 中类似的表格。再次输入信息,如图 8.8 所示,只是应使用数据类型 varchar 代替 Text (Text 在 SQL Server 中有不同的含义)。

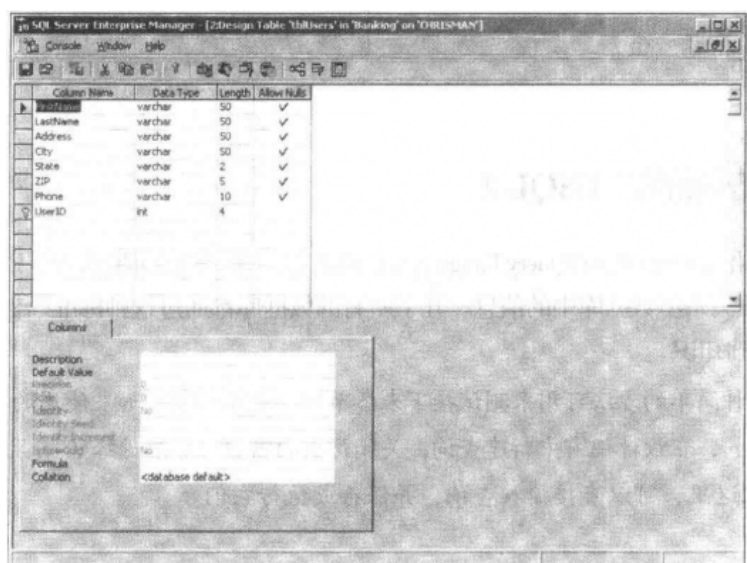


图 8.8 输入字段名称和数据类型

注意, SQL Server 提供的数据类型比 Access 提供的要多。SQL Server 功能更加强大, 让开发人员能够更细致地设置数据库选项。SQL Server 不要求首先设置主关键字——但是现在让我们来定义一个。新建一个名为 UserID 的字段, 将其数据类型设置为 int。在下面的 Column 框架中将 Identity 的属性设为 yes (见图 8.9)。然后在该字段上单击鼠标右键, 选择 Set Primary Key (设置主关键字)。接下来, 将该表保存为 tblUsers, 并单击右上角的“x”按钮, 来关闭该设计视图。

要在新表中输入数据, 在该表上单击鼠标右键, 并选择 Open Table/Return All Rows, 出现一个类似于 Access 中的数据输入表, 可输入几个姓名和住址来体验一下其中的乐趣。(记住不要在 UserID 字段中输入数据)。单击菜单条上的惊叹号使 UserID 字段自动递增, 然后单击右上角的“x”按钮, 关闭这个表。

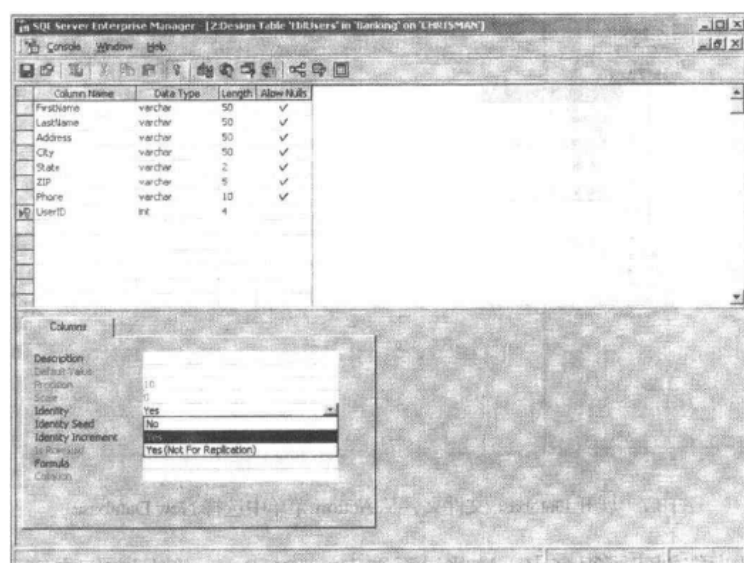


图 8.9 设置标识字段和主关键字

应 该	不 应 该
如果开发的是需要使用高级数据库功能的大型、性能关键的数据 库方案, 应该使用 SQL Server	对于只有几个表的简单数据库, 不要使用 SQL Server, 否则会 增加不必要的复杂性

### 8.3 结构化查询语言 (SQL)

结构化查询语言 (Structured Query Language, SQL), 是一种高效的语言, 被开发人员用于检索、增加、删除或修改关系型数据库中的信息。几乎所有的数据库都通过这种标准语言来通信, 因此必须学习 SQL 的基础知识。

提示: 有几种简单的方法可用来测试接下来各节中的 SQL 语句。如果使用的是 Access, 请打开数据库, 单击 Query, 在设计视图中新建查询, 关闭弹出的窗口, 然后选择 View 菜单中的 SQL View (见图 8.10)。在这里, 可以直接输入查询, 并保存或执行它们。

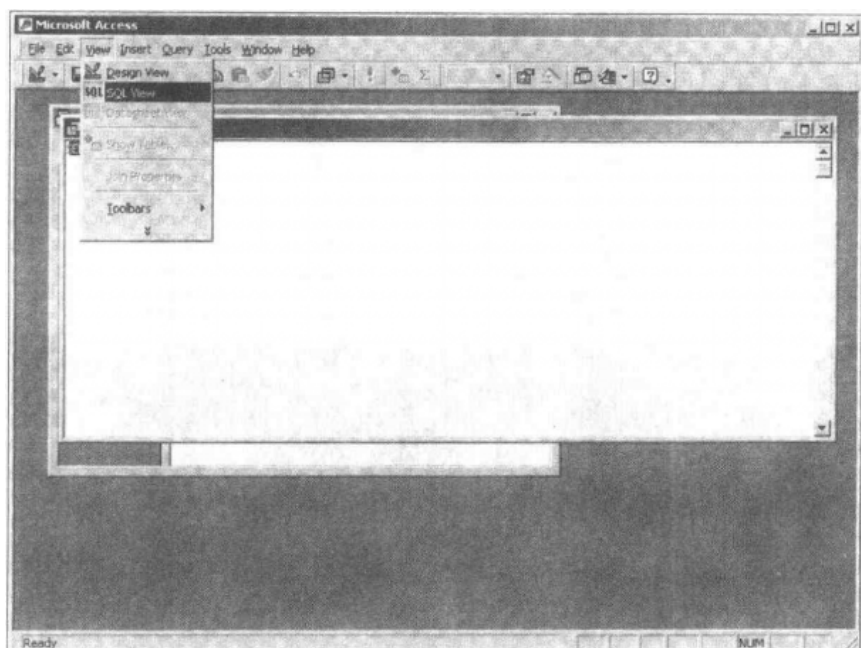


图 8.10 在 Access 中输入 SQL 查询

对于 SQL Server，则从 Start 菜单中打开 Query Analyzer

### 8.3.1 SELECT 语句

SELECT 可能是最常用的 SQL 语句，其主要用途是从数据库表中获得一个数据集。SELECT 语句的语法如下：

```
SELECT selectlist
FROM tablename
[WHERE searchCondition]
[ORDER BY orderByExpression [ASC|DESC]]
```

这种语法结构符合英语语法模式。SelectList 通常是要取回的字段列表，字段间用逗号分隔。例如，Firstname、Lastname 将返回这两个字段中的数据。另外，“\*”可用于返回表中的所有字段。Tablename 是所要查询的数据表的名称。下面是一个简单的 SELECT 语句，可将它用于您创建的数据库中。

```
SELECT * FROM tblUsers
```

该语句返回 tblUsers 表中所有的记录，然后您可以对其执行任何操作。

下面的语句与前一个语句等效：

```
SELECT FirstName, LastName, Address, City, State, Zip, Phone FROM tblUsers
```

**提示：**应尽可能避免使用 SELECT \*，因为该语句返回的信息通常多于您需要的，这样会降低性能，因此应明确指定要返回的字段，虽然这样做时，编写代码的时间要长一些，但得到的回报是性能更高。

SELECT 语句经常会返回数据库中的所有记录（除非使用 WHERE 子句进行限定，这将在下面进行讨论）。图 8.11 和 8.12 分别是下面两个 SELECT 语句从您创建的 Access 数据库中返回的数据：

```
SELECT * FROM tblUsers
```

```
SELECT FirstName, Phone FROM tblUsers
```

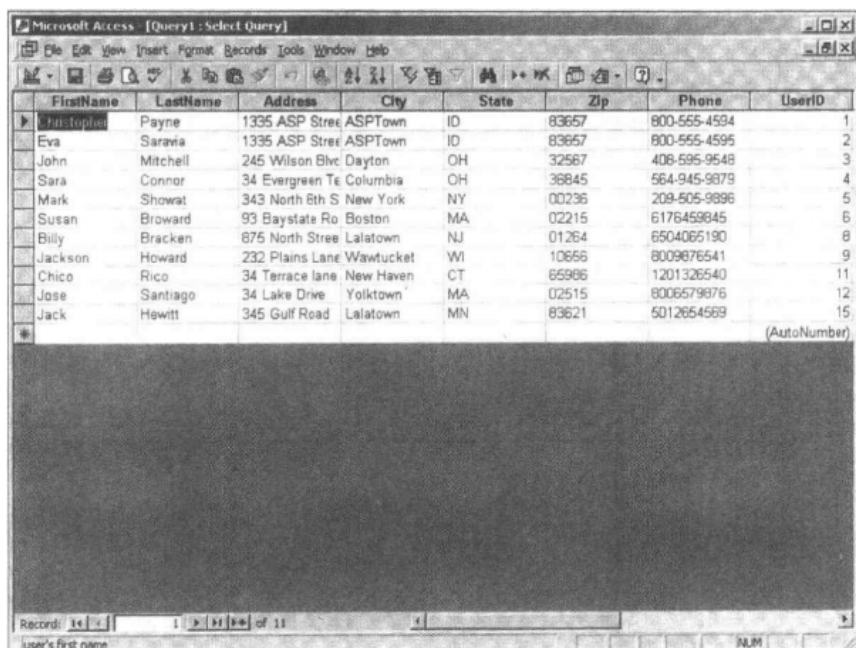


Figure 8.11 shows a Microsoft Access window titled "Microsoft Access - [Query1: Select Query]". The window displays a table with 11 records and 8 columns: FirstName, LastName, Address, City, State, Zip, Phone, and UserID. The records are as follows:

FirstName	LastName	Address	City	State	Zip	Phone	UserID
Christopher	Payne	1335 ASP Street	ASPTown	ID	83657	800-555-4594	1
Eva	Saravia	1335 ASP Street	ASPTown	ID	83657	800-555-4595	2
John	Mitchell	245 Wilson Blvd	Dayton	OH	32587	408-595-9548	3
Sara	Connor	34 Evergreen Ter	Columbia	OH	36845	564-945-9879	4
Mark	Showrat	343 North 8th St	New York	NY	00236	209-505-9896	5
Susan	Broward	93 Baystate Road	Boston	MA	02215	6176459845	6
Billy	Bracken	875 North Street	Lalatown	NJ	01264	6504065190	8
Jackson	Howard	232 Plains Lane	Wawtucket	WI	10666	8009876541	9
Chico	Rico	34 Terrace Lane	New Haven	CT	65986	1201326540	11
Jose	Santiago	34 Lake Drive	Yolktown	MA	02515	8006579876	12
Jack	Hewitt	345 Gulf Road	Lalatown	MIN	83621	5012654569	15

The status bar at the bottom indicates "Record: 11 of 11" and "user's first name".

图 8.11 SELECT 语句返回的所有记录

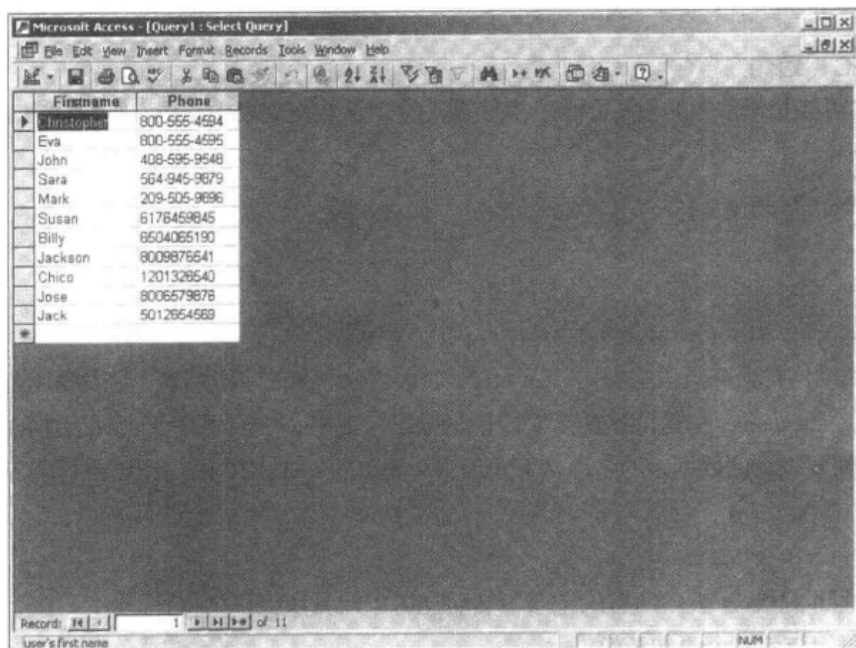


Figure 8.12 shows a Microsoft Access window titled "Microsoft Access - [Query1: Select Query]". The window displays a table with 11 records and 2 columns: FirstName and Phone. The records are as follows:

FirstName	Phone
Christopher	800-555-4594
Eva	800-555-4595
John	408-595-9548
Sara	564-945-9879
Mark	209-505-9896
Susan	6176459845
Billy	6504065190
Jackson	8009876541
Chico	1201326540
Jose	8006579876
Jack	5012654569

The status bar at the bottom indicates "Record: 11 of 11" and "user's first name".

图 8.12 SELECT 语句返回的指定字段的所有记录

可以使用 WHERE 子句指定其他的条件，以限制返回的数据。WHERE 子句通常可以是任何一种逻辑语句，类似于常规编程语言中的 IF 语句，例如：

```
SELECT * FROM tblUsers WHERE FirstName='Chris'
```

```
SELECT * FROM tblUsers WHERE City='ASP TOWN' and LastName='Mitchell'
```

第一个语句只返回那些 FirstName 字段值为 Chris 的记录；第二个语句只返回那些 City 字段值

为 ASP Town 并且 LastName 字段值为 Mitchell 的记录。

要在 WHERE 子句条件中使用通配符，可在关键字 LIKE 后面加上通配符操作符。表 8.2 列出了 LIKE 关键字接受的通配符操作符。

表 8.2 LIKE 关键字接受的通配符操作符

操作符	描述
*	含有 0 个或多个字符的字符串（注意，在 SQL Server 中，对应的通配符为 %）
_	一个字符
[]	给定字符集中的指定字符

下面语句是几个使用 LIKE 的例子：

```
1 SELECT * FROM tblUsers WHERE FirstName LIKE '*S*'
2 SELECT * FROM tblUsers WHERE FirstName LIKE '_hris*'
3 SELECT * FROM tblUsers WHERE FirstName LIKE '[abcd]hris'
```

第 1 行的 SQL 语句返回 FirstName 字段包含“s”的所有记录。在 SQL 中，\*操作符的作用与在 Windows 中的作用类似，表示 0 或多个字符。

第 2 行的 SQL 语句返回 FirstName 字段中以任何一个字符开头，其后面为 hris 的所有记录。对于前面输入的数据而言，该语句只返回一条记录，但您知道了这种通配符的含义。最后第 3 行的 SQL 语句返回 FirstName 字段以 a、b、c 或 d 打头，后面为 hris 的所有记录。

ORDER BY 子句指定返回的数据的排列顺序，例如按字母排序。没有该语句，将无法知道数据库将以何种次序返回记录。尽管数据是按字母顺序输入的，但这并不意味着数据会按字母顺序返回。ORDER BY 让您能够指定数据的返回次序以哪个字段为准。如果 ORDER BY 被用于包含字符串的字段，则其中的字符将自动转换为对应的 ASCII 码，然后按 ASCII 码排序。例如：

```
SELECT * FROM tblUsers
ORDER BY FirstName ASC
```

该语句返回所有的记录，并按 FirstName 字段的字母顺序进行排序。同样，如果将 ASC 换为 DESC，则返回的记录按字母降序排列。还可在 ORDER BY 子句中指定多个字段，字段间用逗号分隔，每个字段具有自己的排序方式。如果第一个字段含有相同的数据，则 SQL 将其他字段进行排序，只是千万不要指定表中不存在的字段！

例如，下面的语句返回如图 8.13 所示的结果。

```
SELECT * FROM tblUsers
ORDER BY FirstName, Phone ASC
```

注意，当 FirstName 相同时（有两个 Jack），记录将按电话号码排序。

也可使用 SELECT 语句一次返回多个表中的数据。这需要将 tablename 参数设置为逗号分隔多个表的名称。通常，需要以某种方式将这些表关联起来，这将在下一章进一步讨论。

本章将要学习的其他 SQL 语句的基本格式与 SELECT 查询相同。因此，介绍接下来的三个语句时将不再花太大的篇幅。



FirstName	LastName	Address	City	State	Zip	Phone	UserID
Bracken		675 North Street	Lalatown	NJ	01264	6804065190	8
Chico	Rico	34 Terrace Lane	New Haven	CT	65986	1201326540	11
Christopher	Payne	1335 ASP Street	ASPTown	ID	83657	800-555-4594	1
Eva	Saravia	1335 ASP Street	ASPTown	ID	83657	800-555-4595	2
Jack	Hewitt	345 Gulf Road	Lalatown	MN	83621	5012654569	15
Jack	Cross	208 Yipawow	Shonville	IN	45678	5648781005	16
Jackson	Howard	232 Plains Lane	Wawtucket	WI	10656	8009876541	9
John	Mitchell	245 Wilson Blvd	Dayton	OH	32567	438-595-9548	3
Jose	Santiago	34 Lake Drive	Yolkdown	MA	02515	8006579876	12
Mark	Showat	343 North 8th S	New York	NY	00236	209-535-9096	5
Sara	Connor	34 Evergreen Te	Columbia	OH	36645	584-945-9879	4
Susan	Broward	93 Baystate Rd	Boston	MA	02215	6176458945	6

图 8.13 SELECT 语句中用于排序的字段

### 8.3.2 INSERT 语句

另一个常用的 SQL 语句是 INSERT，用于向数据库表中插入新记录（数据）。其基本语法如下：

```
INSERT [INTO] tablename
[(column list)]
values (DEFAULT | NULL | expression)
```

该语句很简单，其语法类似于 SELECT。下面让我们使用用户数据库来创建一个例子：

```
INSERT INTO tblUsers (FirstName, LastName, Address, City,
State, Zip, Phone)
VALUES ('Chris', 'Payne', '135 ASP Street', 'ASPTown', 'FL',
36844, '8006596598')
```

该语句将一条新记录插入表中，其字段值在 VALUES 子句中指定。在 VALUES 中提供的数据类型必须和相应字段的数据类型相匹配，否则将出错。另外，如果指定了一个字段，就必须指定该字段的值，否则也会出错。

### 8.3.3 UPDATE 语句

UPDATE 语句用于更新数据库表中的记录。可以在 UPDATE 语句中使用 WHERE 子句，以便只更新数据库中的某些值。其语法如下：

```
UPDATE tablename
SET column name = (DEFAULT | NULL | expression)
[WHERE searchCondition]
```

如果省去 WHERE 子句，则数据库中所有记录的指定字段都将被更新。使用该语句时一定要小心，因为它很容易破坏数据！下面再次使用 tblUser 作为例子：

```
UPDATE tblUser
SET Address = '136 ASP Street', Town = 'ASPville'
WHERE FirstName = 'Chris' AND LastName = 'Payne'
```

该语句修改 FirstName 字段为 Chris，LastName 字段为 Payne 的记录中的 Address 和 Town，这样

的记录只有一条。可以在 column name 中只指定要修改的字段，其中 WHERE 子句的使用方法同 SELECT 语句完全相同——也可以使用通配符。

#### 8.3.4 DELETE 语句

本章要介绍的最后一种 SQL 语句是 DELETE，它用于删除数据库中的记录。其语法如下：

```
DELETE FROM table_name
[WHERE searchCondition]
```

DELETE 语句极为简单，唯一要记住的是 WHERE 子句，它十分重要。如果没有指定 WHERE 子句，DELETE 语句将删除数据库中的所有记录——后果将是灾难性的。WHERE 子句在此处的用法与在 SELECT 语句中的用法完全相同。例如，下面的例子只删除 FirstName 字段为 Chris 的记录：

```
DELETE FROM tblUsers
WHERE FirstName = 'Chris'
```

### 8.4 在 ASP.NET 中存取数据

本章余下的内容将介绍在 ASP.NET 页面中存取数据库的基础知识。下一章将使用这些知识在页面中访问真正的数据。

通常，在 ASP.NET Web 页面中存取数据库分 5 步：

1. 建立数据库连接；
2. 打开数据库连接；
3. 使用需要的数据填充 DataSet；
4. 建立 DataView 以显示数据；
5. 通过数据绑定将服务器控件绑定到 DataView。

可将存取数据看作排队吃自助餐。建立和打开数据库连接相当于找到并走向餐台。挑选自己喜欢的食品时，您将土豆泥放在食品盘的一端，盘的另一端放大块肉，等等。DataSet 相当于盘子，用来盛放各种食品或数据。前面介绍了 SQL SELECT 语句，使用该语句可以从数据库表中返回一个记录集记录。在 ASP.NET 中，DataSet 可用来保存这些结果。

以这种方式划分数据时，使用的是 DataView。DataView 提供了一个数据视图，可以显示给用户。

数据绑定更复杂一些，但可以将其看作在盘中的食品和自助餐柜中的食品之间建立一种神奇的联系。您在盘中添加一些通心粉时，自助餐柜中也多了一些通心粉。

#### 8.4.1 存取数据

清单 8.1 给出了一个简单的 ASP.NET 页面，可用于访问已创建的数据库。具体地说，访问和显示了 tblUsers 表中的所有字段和行。让我们来看看源代码！

清单 8.1 在 ASP.NET 中存取数据

```
1 <%@ Import Namespace="System.Data" %>
2 <%@ Import Namespace="System.Data.OleDb" %>
3
4 <script language="VB" runat="server">
5     sub Page_Load(obj as Object, e as EventArgs)
```

```

6      'set up connection
7      dim objConn as new OleDbConnection _
8          ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
9          "Data Source=H:\ASPNET\data\banking.mdb")
10
11      'open connection
12      dim objCmd as OleDbDataAdapter = new OleDbDataAdapter _
13          ("select * from tblUsers", objConn)
14
15      'fill dataset
16      dim ds as DataSet = new DataSet()
17      objCmd.Fill(ds, "tblUsers")
18
19      'select data view and bind to server control
20      MyDataList.DataSource = ds.Tables("tblUsers"). _
21      DefaultView
22      MyDataList.DataBind()
23  end sub
24 </script>
25
26 <html><body>
27 <ASP:DataList id="MyDataList" RepeatColumns="2"
28     RepeatDirection="Vertical" runat="server">
29 <itemTemplate>
30     <div style="padding:15,15,15,15;font-size:10pt;
31         font-family:Verdana">
32     <div style="font:12pt verdana;color:darkred">
33         <i><b><%# DataBinder.Eval(Container. _
34             DataItem, "FirstName")%>&nbsp;&nbsp;&
35         <%# DataBinder.Eval(Container. _
36             DataItem, "LastName")%>
37         </i></b>
38     </div>
39     <br>
40     <b>Address: </b><%# DataBinder.Eval(Container.DataItem, _
41         "Address") %><br>
42     <b>City: </b><%# DataBinder.Eval(Container.DataItem, _
43         "City")%><br>
44     <b>State: </b><%# DataBinder.Eval _
45         (Container.DataItem, "State") %><br>

```

```

46      <b>ZIP: </b><%# DataBinder.Eval _
47          (Container.DataItem, "Zip"). %><br>
48      <b>Phone: </b><%# DataBinder.Eval _
49          (Container.DataItem, "Phone") %><br>
50  </div>
51  </ItemTemplate>
52  </ASP:DataList>
53 </body></html>

```

**分析:** 在访问数据之前, 必须导入数据名称空间, 如第 1-2 行所示(如果数据库是使用 Microsoft SQL Server 创建的, 则数据名称空间为 System.Data.SqlClient)。

第 7-9 行建立了到数据库的连接。数据源被设置为本章前面创建的 Access 文件。第 12-13 行打开到数据库的连接, 并执行了一个您熟悉的 SQL 语句, 以返回表中的所有数据。这些代码行对应于数据访问过程中的第 1 步和第 2 步。

第 16-17 行创建了一个 DataSet 对象, 并使用返回的数据填充它。这是第 3 步。

第 20-22 行为第 4 步和第 5 步。您将数据绑定到一个 DataList Web 控件, 该控件可自动显示数据。余下的唯一一项工作是创建 DataList 控件。

DataList 服务器控件使用模板来按您的要求格式化数据, 并自动遍历数据库记录。它还能确定浏览器的形状和尺寸, 并创建列以便在一个窗口中显示所有的数据, 这为您省了不少麻烦。您将发现模板中的大多数语法只是 HTML 格式标签。

然而, 这里有一些奇怪的<%# and %>标签, 它看起来有点像代码交付块, 这是数据绑定表达式。使用数据绑定可以明确地将数据源链接到服务器控件。数据源发生的任何变化都会反映到控件上, 这使得数据操纵很容易(如果现在搞不清楚, 也不要担心, 下一章将花大量的篇幅介绍这方面的内容)。

图 8.14 是 DataList 控件的输出。

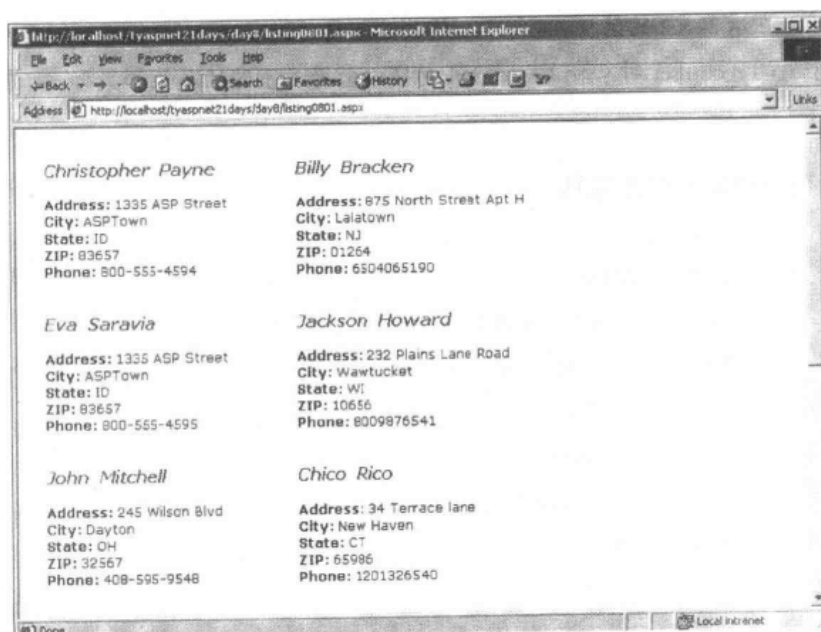


图 8.14 tblUsers 表的内容

## 8.5 这不是 ASP

熟悉传统 ASP 的读者将发现 ASP.NET 与传统 ASP 有很大的不同(即使不熟悉传统 ASP 的读者也应该阅读本节)。建立并打开数据库连接的过程有些眼熟——只是语法有一些变化,但格式相同。除了必须导入名称空间外,第一个主要差别在于要填充一个 DataSet,而不是返回一个记录集(recordset)。清单 8.2 显示了使用传统 ASP 和 VBScript 从数据库获取数据的过程。

清单 8.2 使用传统 ASP 存取数据

```

1:  <%
2:      Dim objConn
3:      Set objConn = Server.CreateObject("ADODB.Connection")
4:      objConn.ConnectionString = "Provider=" & _
5:      "Microsoft.Jet.OLEDB.4.0;Data Source=" & _
6:      "C:\ASP\NET\data\banking.mdb"
7:      objConn.Open
8:
9:      Dim strSQL
10:     strSQL = "SELECT * FROM tblUsers"
11:
12:     Dim objRS
13:     Set objRS = Server.CreateObject("ADODB.Recordset")
14:     objRS.Open strSQL, objConn
15:  %>

```

**分析:** 第 2~7 行相当于清单 8.1 中的第 7~9 行,您创建一个 ADO 连接,指定连接字符串,并打开数据库连接。在清单 8.2 的第 9~14 行中,您创建了自己的 SQL 语句,并使用该 SQL 语句返回的数据填充一个新的 recordset 对象。该 recordset 对象包含很多不同的属性和集合,可用来显示数据,如清单 8.3 所示。

清单 8.3 在传统 ASP 中显示数据

```

1  <%
2      Do While Not objRS.EOF
3          Response.write "<B>" & objRS("FirstName") & " " & _
4              objRS("LastName") & "</B><BR>" & _
5              objRS("Address") & "<BR>" & objRS("City") & _
6              "<BR>" & objRS("State") & "<BR>" & objRS("Zip") & _
7              objRS("Phone") & "<P><HR><P>"
8              objRS.MoveNext
9      Loop
10
11      objRS.Close

```

```

12      Set objRS = Nothing
13
14      objConn.Close
15      Set objConn = Nothing
16  %>

```

在该清单中，您遍历 recordset 中返回的记录，并使用 Response.Write 输出各个字段。您使用 objRS.MoveNext 移动到下一条记录，否则将导致死循环。最后，您关闭并释放 recordset 及 connection 对象。

在清单 8.1 中，清单 8.3 中很多内容是不必要的。清单 8.1 的第 17 行填充了一个 dataset，后者包含一个通用数据集。然后，在第 22 行将 DataSet 绑定到服务器控件，这样整个过程便完成了。数据绑定表达式自动遍历所有记录，并显示各个字段。任务完成后，CLR 自动地清除并释放这些对象。

ASP.NET 的方法更加强大大，实施起来更轻松，运行速度更快，效率更高。在 ASP.NET 中，您无需为遍历数据操心，从而可以腾出更多的精力来规划如何显示数据。DataSet 还提供了比 RecordSet 更多的功能，包括完全支持 XML 以及重新与数据源同步。

幸运的是，数据存取后面的很多概念仍相同，也就是说您现有的技能仍适用。学习本章和下一章介绍的机制后，您将发现新的 ASP.NET 框架的功能是多么的强大！

## 8.6 总 结

对所有应用程序来说，数据库都十分重要，尤其是基于 Web 的应用程序。通过 Internet 存取数据的价值无法衡量，而 ASP.NET 使得与数据库交互异常简单。

本章介绍了数据库的基础知识——平面文件数据库和关系型数据库的差别以及约束和连接数据库的方法。然后使用 Access 2000 创建了一个将在后面的应用程序中使用的数据库。

接下来，简要地介绍了 SQL 和几个重要的语句：SELECT、INSERT、UPDATE 及 DELETE，并介绍了如何在数据库表中使用这些语句。最后介绍了如何通过一个简单的 ASP.NET 页面来存取数据，并对 ASP.NET 和传统 ASP 存取数据的方式做了比较。

下一章将介绍如何在 ASP.NET 中访问及修改数据库，同时将介绍第 6 章略过的以数据为中心的服务器控件。至此，我们只介绍了 ASP.NET 强大数据库功能的很小一部分。阅读完本部分的内容后，您将能熟练地使用数据库。

## 8.7 问与答

问：是否应该设计关系型数据库？

答：绝对应该！刚开始时或许会有些笨手笨脚，但现在几乎所有的数据库应用程序都建立在关系型数据库的基础之上。要有效地设计关系型数据库，您需做大量的练习，但是如果不涉入的话永远都学不会！

本章设计了一个只包含一个表的数据库（这很难明白关系型设计方法），但是几章后将加入更多的表，创建更复杂的关系。相信我，使用关系型数据库您将得到更多的乐趣！

问：什么样的数据类型是 XML？

答：XML 本身是一种规范，让您能够为数据创建定义良好的结构。也就是说可以使用 XML 存

储任何类型的数据。

XML 有三大优点：首先，它是基于文本的，便于用户阅读和修改；其次，它是一种标准：最后它能够很容易地通过 Web 进行传输。因此，可以将创建的数据存储为 XML 格式，而任何人都可以轻松地访问这些数据。试着让其他人访问您的 SQL Server 2000 数据库，您将发现这要复杂得多。

当然，ASP.NET 使得其他人可以很轻松地访问您的数据库，但是想象一下，如果没有 ASP.NET，您的生活将会是什么样子。第 11 章将介绍 XML 和 ASP.NET。

问：在 ASP.NET 中使用数据库有没有什么缺点？

答：在 Web 中应用数据库的主要缺点是需要很长的连接时间（相对而言）。在这一点上，ASP.NET 比传统 ASP 的性能要好，但是仍旧有缺陷。

一般说来，如果数据量很大且要无限期使用，则使用数据库是种不错的选择。如果系统配置良好，性能问题将小一些。而数据库提供了很多好处，例如易于操纵数据，这使其有别于其他数据存储方式。

## 8.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 8.8.1 小测验

1. 什么是关系型数据库？
2. SQL 代表什么？它有什么作用？
3. ODBC 代是什么？
4. 判断正误：可以通过 ASP.NET 访问任何 ODBC 顺应的数据库。
5. 在 ASP.NET 中访问数据库包括哪 5 步？

### 8.8.2 练习

1. 从头开始创建一个数据库，将其用来存储宠物的信息（如果您没有宠物，试着虚构一些！）。需要哪些字段？
2. 对于清单 8.1（具体地说，是 `dataList` 控件），修改模板中的格式标签，查看更改后的显示情况，并应用 ASP.NET 服务器控件的知识来修改 `dataList` 控件的属性。

## 第9章

# 在 ASP.NET 中使用数据库

前一章介绍了有关数据库的基础知识——何时使用数据库、如何创建数据库、（最重要的是）如何在 ASP.NET Web 页面中访问数据库。前一章对这一过程做了简要性的概述，本章将更详细地介绍如何在 ASP.NET 中使用数据库。

本章还将介绍数据绑定和第6章略过的服务器控件。这些特性使得 ASP.NET 非常适合编写数据库驱动应用程序。

本章包括以下内容：

- 新 DataSet 对象；
- 什么是数据绑定，ASP.NET 如何应用数据绑定；
- 三个 ASP.NET 服务器控件：Repeater、DataList 和 DataGrid；
- 一个数据绑定的例子。

### 9.1 ASP.NET 访问数据库简介

通过 Web 访问数据库近年来取得了重大进展。数据库访问已从访问简单的文本文件转向访问大型企业的全部在线数据系统——其中一些含有几万亿字节的数据。甚至股票经纪人和订单处理系统也已转到网上，它们每天产生大量的数据。幸运的是，现在 ASP.NET 可以帮助您处理所有这些事情。

传统的 ASP 页面使用 ActiveX Data Object (ADO) 来访问和修改数据库。ADO 是一种用于存取数据的编程接口。ADO 方法的效率高，便于开发人员学习和实现。然而，由于其数据存取模型过时了，因此存在很多局限性，例如，ADO 没有数据传输能力，因此用通用的方法便可以很容易地访问它。随着从标准 SQL 数据库到分布式数据库（例如 XML）的转移，Microsoft 推出了 ADO.NET——下一代 ADO。

ADO.NET 在 ADO 的基础上做了重大改进，使得 ASP.NET 页面能够以一种不同的、效率更高的方式提供数据。例如，ADO.NET 全面支持 XML，能够轻松地同 XML 顺应的应用程序通信。ADO.NET 提供了众多令人激动的新特性，这将使开发人员的工作更为轻松。

ADO.NET 涉及的内容很多，所以将在下一章专门讨论。现在，您只需要知道 ASP.NET 页面总是通过 ASP.ADO 同数据源通信的。图 9.1 是使用 ADO.NET 和 ASP.NET 存取数据的模型。ADO.NET 与 OLE-DB 顺应数据源完全兼容，如 SQL 和 Jet（Microsoft Access 的数据库引擎）。



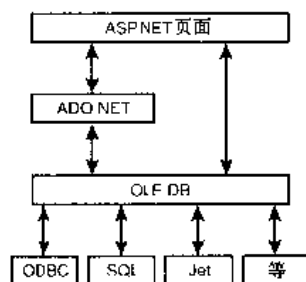


图 9.1 ADO.NET 和 ASP.NET 的数据存取模型

## 9.2 DataSet

ADO.NET 是围绕着 DataSet 运行的。DataSet 是一个全新的概念，它取代了 ADO 中的 RecordSet 对象。RecordSet 提供了读取及显示数据库记录的方法。RecordSet 是一种十分有用的读取数据的方法，但存在一些局限性。具体地说，RecordSet 表示数据的方式很简单：它只包含一个数据集，不包含关于数据间关系的信息。

DataSet 是一种简单的、驻留在内存中的数据存储方式，为数据存取提供了一致的编程模型，而不管包含的数据是何种类型。同 RecordSet 不同，DataSet 包含了全套数据，包括约束、关系，甚至多表。图 9.2 是 DataSet 对象模型的抽象图。

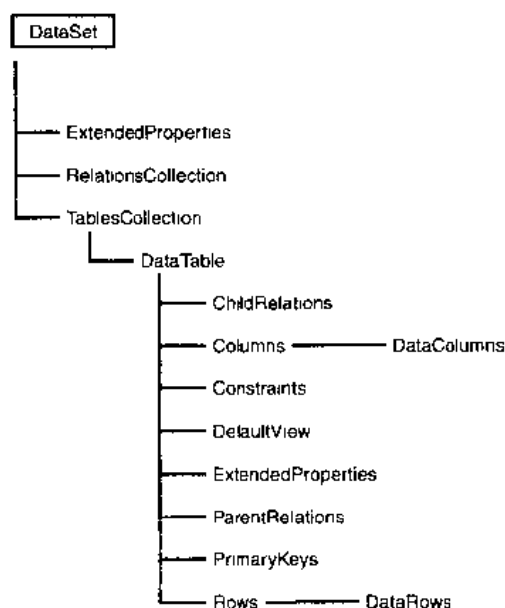


图 9.2 DataSet 对象模型

想象一个包含多格的盒子。您可以将任何喜欢的物体放进每个格子，只要盒子能够容得下。可以查看或操纵盒子里的任何物体——取出、添加或只是查看等。DataSet 本质上也是如此。建立到数据库的连接后，您给数据库分配一个盒子，并命令数据库将一些东西放到里面。您可以使用数据表、其他地方的数据、其他对象——任何数据——来填充盒子。不管放入的是什么物体，盒子都允许您

对物体做相同的操作，如查看、添加、删除等。盒子是动态的，可以根据物体的多少扩大或缩小。

图 9.2 包含了一个 `DataTable` 对象，后者表示一个数据库表。（`DataSet` 在 `TablesCollection` 对象中维护了这些表的集合）。`DataTable` 代表相应的表，包括其关系和关键字约束。它还包含另外两个集合：`Rows` 和 `Columns`，它们分别表示表的数据和模式。

现在再回到盒子上。其中每一格是一个 `DataTable`。盒子周围是 LCD，可以自动显示里面的物体（如果电冰箱有这种功能就太棒了！）。盒子现在已颇具功能！图 9.3 说明了这种盒子。

`RelationsCollection` 对象可以让您能够通过表的关系在表之间导航。现在没必要再在 SQL 查询中使用连接和合并（一种将表关联起来的老式方法）了，因为 ADO.NET 使得这一切简单得多。表之间的实际关系由 `DataRelation` 对象表示，该对象含有被连接的两个表的信息、主关键字和外关键字关系以及关系的名称。

您还可以使用 `DataRelation` 对象添加关系。ADO.NET 还自动执行关键字约束——禁止对表执行破坏表关系的修改。

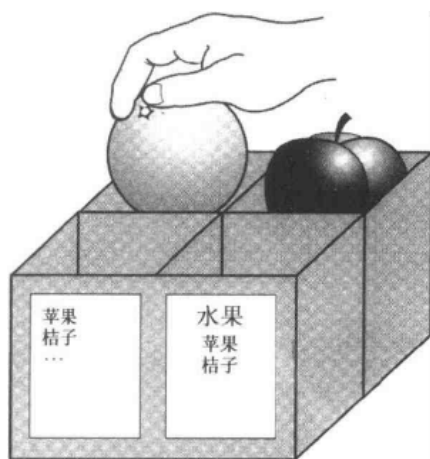


图 9.3 DataSet-Box 模型

盒子再次得到改进。现在您知道哪些对象是相关的以及它们是如何被关联在一起的。例如，假设有一个苹果和一个桔子，您告诉盒子：它们是相关的，因为它们都是水果。以后见到盒子时，您可以说“给我看看水果”，盒子将列出苹果和桔子。假设您要将胡萝卜加入到盒子中，并且您告诉盒子：胡萝卜也与苹果、桔子相关。这时盒子将拒绝您的要求，因为这违反了关系规则。

`ExtendedProperties` 对象包含其他所有的信息，包括用户名和密码。

盒子中水果的范例虽有些简单，但它解释了 `DataSet` 的概念。`DataSet` 完全不关心数据的来源或去向——它完全是独立于数据存储方式的。因此，可将 `DataSet` 作为一个独立的实体。下一节将详细介绍 `DataSet` 实际的工作原理。

### 9.2.1 使用 DataSet

使用 `DataSet` 的方法有两种：使用数据源中的数据填充 `DataSet` 或从头开始创建一个空的 `DataSet`。首先介绍后一种方法。

由于 `DataSet` 是驻留在内存的数据存储器，因此可以通过编写代码创建 `DataSet`，并将表加入到其中。如果想在 ASP.NET 应用程序中使用动态数据，而无需直接操作数据库，则这是一种很有用的特性。

**清单 9.1 通过编写代码创建一个 DataSet**

```

1:  'create an empty dataset
2:  dim ds as new DataSet("MyDataSet")
3:
4:  'create a new table and columns
5:  dim dTable as New DataTable("Users")
6:  dTable.Columns.Add("FirstName",System.Type. _
7:      GetType("System.String"))
8:  dTable.Columns.Add("LastName",System.Type. _
9:      GetType("System.String"))
10: dTable.Columns.Add("UserID",System.Type. _
11:     GetType("System.Int32"))
12: dTable.Columns("UserID").AutoIncrement = true
13:
14: 'add table
15: ds.Tables.Add(dTable)
16:
17: 'define primary key
18: dim keys() as DataColumn = {ds.Tables("Users"). _
19:     Columns("UserID")}
20:ds.Tables("Users").PrimaryKey = keys
21:
22: 'add a sample row
23: dim dr as DataRow = dTable.NewRow()
24: dr(0) = "Chris"
25: dr(1) = "Payne"
26: dTable.Rows.Add(dr)

```

**分析:** 清单 9.1 创建一个 DataSet, 在其中加入一个表, 并使用一些数据填充这个表。您甚至不需要数据库——现在可以在 ASP.NET 页面的任何地方使用该 DataSet。(注意, 对于在这个清单中使用的对象, 必须导入名称空间 System.Data)。

第 2 行实例化了一个新的、名为“MyDataSet”的 DataSet 对象。然后在第 5 行新建了名为“Users”的表, 并加入了三个字段“FirstName”、“LastName”和“UserID”。Add 方法有两个参数——字段的名称和类型。GetType 方法返回对象的类型, 如第 7、9、11 行所示。第 12 行将“UserID”字段的 AutoIncrement 属性设置为 True, 以便新添记录时该字段的值被自动设置, 这确保每条记录都有一个唯一的标识符。第 15 行将新表加入到 DataSet 的表集合中。

第 20 行的 PrimaryKey 属性可被设置为一个 DataColumn 对象数组, 这样便将多个字段定义为主关键字, 使记录是唯一的。例如, 在 User 数据库中, 姓可能不是唯一的, 但姓名可能是唯一的。您可以定义一个由这两个字段组成的数组, 并将其设置为主关键字。幸运的是, 您有一个唯一性的标识字段——UserID, 因此在第 18 行创建了只包含该字段的数组, 并将其设置为主关键字。

现在可以使用该 DataSet 来存储数据了, 如果愿意, 您甚至可以将这些数据写入到文件中或插

入到数据库中。第 23 行新建了一条记录存放数据, 而第 24–25 行将 FirstName 字段设置为 Chris, 将 LastName 字段设置为 Payne。您无需为字段 UserID 设置值, 因为 DataSet 将自动设置该字段。最后, 第 26 行将该记录加入到表中。可以使用表 9.1 列出的方法之一将记录加入到 DataSet 中。

表 9.1 将对象加入到 DataSet 中的步骤

方 法 1	方 法 2
1. 实例化一个对象	1. 在 DataSet 中添加一个对象
2. 设置该对象的属性	2. 使用 DataSet 对象模型访问该对象, 以设置其属性
3. 将该对象添加到 DataSet 中	

下面是方法 1 的例子:

```
set properties for dTable
or add columns to dTable
dTable.MinimumCapacity = 25
...
ds.Tables.Add(dTable)
```

下面是方法 2 的例子:

```
ds.Tables.Add(new DataTable("Users"))
'set properties for dTable
'or add columns to dTable
ds.Tables("Users").MinimumCapacity = 25
```

这两种方法的性能差别不大, 只是第一种方法的可读性强, 而第二种方法更紧凑一些。

然后, 使用熟悉的 “.” 语法可以设置每个 DataSet 的属性和值。要引用特定的字段或记录, 可以使用下面的方法:

```
ds.Tables(tablename).Columns(column index or name)
ds.Tables(tablename).Rows(row index)
```

这提供了一种非常强大的机制, 用于与 DataSet 交互。同样, 可以使用下面的方法来访问或修改字段的值:

```
strValue = ds.Tables(tablename).Rows(row index)(field name).ToString
ds.Tables(tablename).Rows(row index)(field name) = strValue
```

## 9.2.2 关系

现在假设以前面的方式创建了两个表, 并想将它们关联起来。第一个是用户表, 第二个是 Books 表, 后者存储了每个用户阅读的图书列表。这两个表通过用户的 UserID 被关联起来。假设这两个表都有 UserID 字段, 清单 9.2 列出了一些代码。

清单 9.2 使用 DataSet 创建关系

```
1: create a new relation
2: dim dr as DataRelation = New DataRelation("UserBooks",
3: ds.Tables("Users").Columns("UserID"), _
```

```

4:      ds.Tables("Books").Columns("UserID")
5:
6:      \add the relation
7:      ds.Relations.Add(dr)

```

您通过指定关系名称新建了一个名为 `UserBooks` 的关系，并指定了构成该关系的字段。这种关系让您能够通过关联的字段在这两个表之间导航。

**提示：**还可以在每个表中使用多个字段来创建关系，只需将后两个参数设置为 `DataColumn` 对象数组即可。例如，下面的代码在 `Users` 表和 `Books` 表的 `UserID` 和 `FirstName` 字段之间建立了一种关系

```

dim dcUsers(2) as DataColumn
dcUsers(0) = ds.Tables("Users").Columns("UserID")
dcUsers(1) = ds.Tables("Users").Columns("FirstName")
dim dcBooks(2) as DataColumn
dcBooks(0) = ds.Tables("Books").Columns("UserID")
dcBooks(1) = ds.Tables("Books").Columns("FirstName")
dim dr as DataRelation = New DataRelation("UserBooks", _
    dcUsers, dcBooks)
ds.Relations.Add(dr)

```

但要注意的是，指定 `DataColumns` 数组时，第二个参数必须是长度相同的数组。

### 9.2.3 填充 DataSets

知道 `DataSet` 是什么及其功能后，让我们使用前一章创建的数据库中的数据来填充它。这里将使用几个将在下一章介绍的新对象，所以如果您不能理解所有的代码，也不用担心。

清单 9.3 是一个在 ASP.NET 页面中使用 SQL 查询返回的数据来填充 `DataSet` 的例子。

#### 清单 9.3 使用 DataSet 检索数据库中的记录

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     sub Page_Load(obj as Object, e as EventArgs)
7         'set up connection
8         dim myConnection as OleDbConnection = new OleDbConnection _
9             ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
10             "Data Source=H:\ASPNET\data\banking.mdb")
11
12         open connection
13         dim myCommand as OleDbDataAdapter = new OleDbDataAdapter _
14             ("select * from tblUsers", myConnection)
15
16         fill dataset

```

```

17         dim ds as DataSet = new DataSet()
18         myCommand.Fill(ds, 'tblUsers')
19     end sub
20 </script>
21
22 <html><body>
23 </body></html>

```

分析：第2行和第3行包含两个新的名称空间。要以这种方式填充 DataSet 时，必须导入它们（事实上，对于 DataSet 来说，只有 System.Data 是必需的，但 System.Data.OleDb 提供了其他一些助手对象）这里所有的操作都是在 Page\_Load 事件处理程序中进行的。需要重申的是，如果不了解这里的对象或命令，也不要担心——我们将在下一章详细讨论它们。

第8-10行使用 OleDbConnection 对象建立了到数据库的连接。OleDbConnection 对象告诉 ASP.NET 页面到哪里去取得所需要的数据，具体地说，是前一章创建的 banking 数据库。第13行新建了一个 OleDbDataAdapter 对象，用于在 ASP.NET 中执行 SQL 语句。该对象的参数是一条 SQL 语句和第8行创建的 OleDbConnection 对象。这里的 SQL 语句只是检索 tblUsers 表存储的数据。

第17行新建了一个 DataSet。第18行使用 OleDbDataAdapter 对象的 Fill 方法将 SQL 语句返回的数据填充到 DataSet 对象中。第二个参数指定了数据适配器（data adapter）应将数据存放到 DataSet 的哪个表中。注意，不必专门创建该表，Fill 方法将自动为您创建它。

现在可以操纵 DataSet 中的数据以及本章前面介绍的其他所有属性了。另外，还可以将 DataSet 中的数据绑定到 ASP.NET 页面的控件中。数据绑定将在下一节介绍。

使用数据库中的数据填充 DataSet 甚至比自己创建 DataSet 对象还要简单。正如您将在下一节看到的，检索数据只需要4行代码，再使用几行代码来显示数据。

### 9.3 数据绑定

数据绑定让您能够充分地控制数据。几乎可以将任何类型的数据绑定到 ASP.NET 页面中的任何控件或控件属性中，这让您能够完全控制数据在数据库和页面之间的移动方式。您可以将数据显示给用户、设置控件上的样式属性，或让用户直接修改或更新数据库。

将数据绑定到 ASP.NET 页面的方法有两种：使用控件的 DataSource 属性或使用数据绑定表达式。第一种方法常用于较复杂的 ASP.NET 服务器控件，稍后将介绍。第二种方法可用于任何地方。数据绑定表达式的语法如下：

```
<%# property or collection %>
```

注意：虽然这种语法看起来与传统ASP中的代码交付块很相像，但情况并非如此。代码交付块总是在页面运行时被评估，而数据绑定表达式只能使用 DataBind() 方法进行评估。

就这么简单。这种表达式的功能随使用场合而异。以清单9.4为例，它声明了几个变量，然后将其绑定到页面中的不同位置。

**清单 9.4 将数据绑定到页面中的不同位置**

```

1 <script runat="server">
2     dim strName as String = 'Chris'

```

```

3  dim myArray() as String = {"Hello", "World"}
4  dim myString as String = 'Chris'
5
6  sub Page_Load(obj as Object, e as EventArgs)
7      Page.DataBind()
8  end sub
9  </script>
10
11 <html>
12
13
14  My Name is <%# strName %>
15
16  <asp:Listbox datasource='<%# myArray %>' runat="server" />
17
18  <asp:Textbox text='<%# myString.ToString %>' runat="server" />
19
20
21 </html>

```

第 14 行在 HTML 输出 “My Name is” 后显示 strName 的值。在第 16 行中，Listbox 接受一个集合，并自动遍历集合中的数据，将 myArray 值输出为列表项。第 18 行创建一个文本框，用于显示 myString 的值。因此，数据绑定提供了很大的灵活性。

第 14 行和第 18 行并没有体现数据绑定的优势，毕竟您可以在变量的位置输出文本。但第 16 行的 Listbox 则不同，数据绑定使得 Listbox 可以遍历数组的所有元素，并用字符串填充它们。如果不使用数据绑定，则必须使用某种循环手动填充 Listbox。

**警告：**数据绑定表达式必须返回控件期望的数据类型。例如，第 18 行的文本框控件期望一个字符串，如果数据绑定表达式返回的不是字符串将出错。强制转换数据绑定表达式的数据类型通常很有帮助。例如，将第 18 行改为如下代码后，可以保证数据格式的正确性，虽然这里并没有必要：

```
<asp:Textbox text='<%# myString.ToString %>' runat="Server" />
```

仅当数据绑定方法被调用后，数据绑定表达式才会被评估——ASP.NET 不会自动进行处理，所以就何处及何时调用该方法而言，您有很大的选择余地。如果在页面级调用数据绑定，则页面中的所有数据绑定表达式都将被评估，这通常是在 Page\_Load 事件处理程序中完成的。

```

Sub Page_Load(obj as Object,e as EventArgs)
    DataBind()
End sub

```

也可以针对各个控件调用数据绑定，这让您能够更好地控制应用程序如何使用数据。例如，假设允许用户查看和修改用户保存在 Web 站点的个人信息，则可以将用户的全部信息显示在一个页面中，但只允许一次修改一个字段。这样便不用再收集 and 验证每个字段中的信息。清单 9.5 列出了一个例子。

**清单 9.5 各种使用数据绑定的方法**

```

1: Sub Page_Load(obj as Object, e as EventArgs)
2:     'do some stuff
3:     DataBind()
4: End Sub
5:
6: Sub Submit_Click(obj as Object, e as EventArgs)
7:     If Text1.TextChanged Then
8:         Table1.DataBind()
9:     End If
10: End Sub

```

页面被装载后，第 1-4 行中的子程序将数据绑定到页面。这种情况很常见，也常常是我们要实现的。第 6-10 行中的子程序是 Submit 按钮的 Click 事件句柄。当该子程序发现 Text1 控件中的文本发生了变化，则只评估 Table1 的数据绑定表达式，对于用户修改数据字段来说这很有用。

**9.3.1 使用数据绑定**

数据绑定的语法十分灵活，您可以自由地决定如何在页面中使用数据。但如何创建和使用数据源呢？

最简单的方法是创建一个页面级变量（page-level variable），这种变量不包含在任何方法中，这样便可以在页面的任何绑定表达式中使用该变量：

```

<script language="VB" runat="server">
    dim strName as string = "My Name"

    sub Page_Load(obj as Object, e as EventArgs)
        DataBind()
    end sub
</script>
...
...
<asp:Label id="lblName" runat="server" text="<%# strName %>" />
<%# strName %>

```

最后两行提供了两个绑定该页面属性的例子，就这么简单，但这也没帮您多少忙。事实上，可以在 Page\_Load 事件中使用下面的命令来填写标签：

```
lblName.Text = "My Name"
```

数据绑定的真正威力在于在服务器控件中使用动态值。

我们来看一下清单 9.6 中的例子。

**清单 9.6 将一个服务器控件绑定到另一个**

```

1 <script language="VB" runat="server">
2 sub Index_Changed(obj as Object, e as EventArgs)
3     DataBind()
4 end sub

```



```
5 </script>
6
7<html><body>
8  <form runat="server">
9
10     <asp:Listbox runat="server" id="lbColors"
11         width="150"
12         AutoPostBack=true
13         rows="1"
14         SelectionMode="Single"
15         OnSelectedIndexChanged="Index_Changed" >
16         <asp:Listitem value="1">Red</asp:Listitem>
17         <asp:Listitem value="2">Blue</asp:Listitem>
18         <asp:Listitem value="3">Green</asp:Listitem>
19         <asp:Listitem value="4">Yellow</asp:Listitem>
20     </asp:Listbox><p>
21
22     <asp:Label id="lblMessage" runat="server"
23         Text="<%# lbColors.SelectedItem.Text %>" />
24 </form>
25 </body></html>
```

第 22 行的标签使用了一个数据绑定表达式，绑定到 ListBox 的 SelectedItem 属性。每当被选中的索引改变时，再次调用 DataBind，将新值绑定到标签，结果是标签将显示被选中的项目，这只是一个如何绑定到页面公有属性的例子。图 9.4 是该页面的输出。

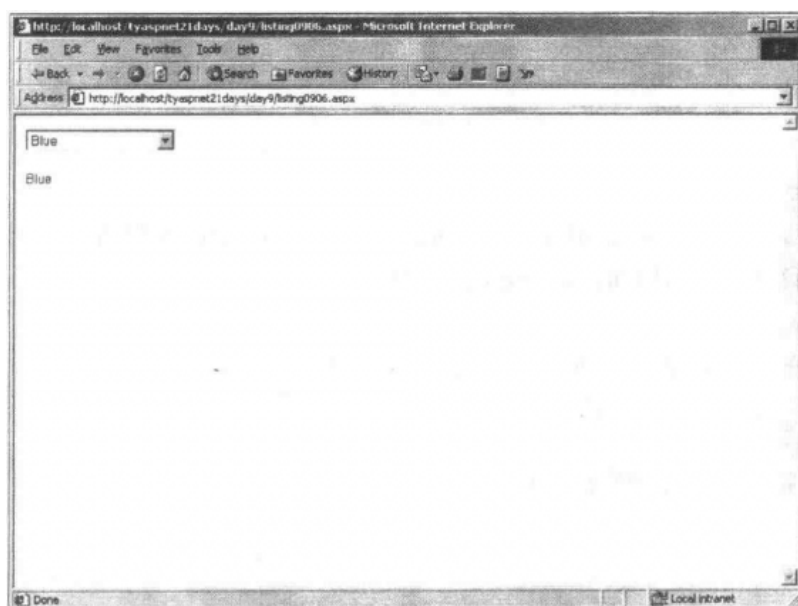


图 9.4 使用数据绑定显示被选中的数据

一些服务器控件还可以绑定到数据类（而不仅仅是属性），以便更细致地操纵数据。这些控件有一个只能在设计阶段才能访问的 `DataSource` 属性。只要将该属性设置为数据类（如数组或数据视图）并调用 `DataBind`，这些控件将为您完成大部分工作。清单 9.7 扩展了前一个例子。

清单 9.7 使用 `DataSource` 属性

```

1 <script runat="server">
2   sub Page_Load(obj as Object, e as EventArgs)
3       if not Page.IsPostBack then
4           'create an array of colors
5           dim arrColors() as string = _
6               {"red", "orange", "yellow", "green", _
7               "blue", "indigo", "violet"}
8
9           lblColors.DataSource = arrColors
10      end if
11      DataBind()
12  end sub
13 </script>
14
15 <html><body>
16   <form runat="server">
17       <asp:Listbox runat="server" id= lblColors
18           width="150"
19           AutoPostBack="true"
20           SelectionMode="Single" >
21       </asp:Listbox><p>
22
23       <asp:Label id="lblMessage" runat="server"
24           Text= <%= lblColors.SelectedItem.Text %> >
25   </form>
26 </body></html>

```

分析：第 5 行创建了一个数据类（一个数组），而第 9 行设置了列表框的 `DataSource` 属性。这样列表框可以自动使用该数组填充自己。标签的功能和前一个清单完全相同——被选择的项目改变时，自动更新自己。图 9.5 显示了该页面。

第 19 行检查 `PostBack`。您只需要在页面首次被查看时创建数组，以后列表框将使用内置的视图状态管理功能自动地填充。因此，对 `PostBack` 进行检查，来确定表单是否被提交过。如果提交过，则无需再重新填充列表框。

这种检查还有另一个用途——如果删掉它，标签将引发“Attempted to dereference a null object reference”的错误。

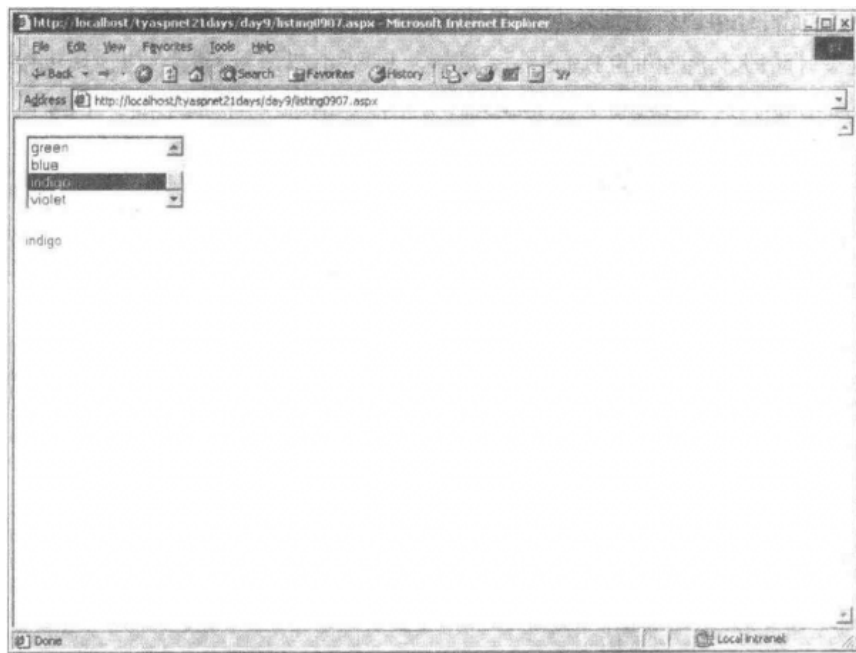


图 9.5 在设计阶段绑定数据

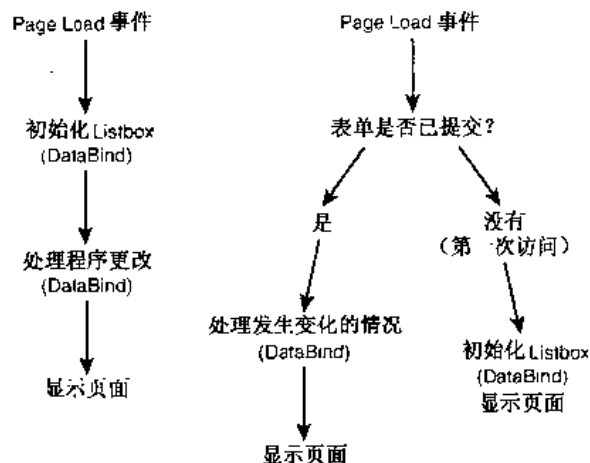


图 9.6 左边是不检查 Postback 时的流程; 右边是检查 Postback 时的流程

假设页面已被提交。在 Page\_Load 事件中设置了列表框的 DataSource 属性, 并绑定了它。这实际上是重新初始化列表框, 并释放状态信息。然后, Index\_Changed 事件被引发, 它评估页面中所有的数据绑定表达式。标签的数据绑定表达式试图使用列表框的 SelectedItem 属性。由于状态信息被删除了, 因此对 SelectedItem 的引用返回一个空对象, 从而引发错误。因此, 对 Postback 进行检查, 防止页面被提交时重新初始化列表框。

要注意这样的细节。鉴于事件驱动模型的工作方式, 在页面中绑定数据时必须小心。

知道数据绑定的工作原理后, 您可能会问, 这对 ASP.NET 页面用户有何益处。答案是, 除了可以缩短程序的开发时间, 更早地交付网站外, 对用户并没有太多的好处。数据绑定最大的好处在于, 它给开发人员提供特性: 缩短开发时间、易于使用以及标准化方法等。

到此您便为学习更复杂的、使用数据绑定的 ASP.NET 服务器控件做好了准备。

## 9.4 数据绑定控件

第5章介绍了各种 ASP.NET 服务器控件，但不包括下面三个复杂的列表控件：服务器控件 Repeater、DataList 和 DataGrid。之所以被称为列表控件，是因为它们能够取得数据集，并自动遍历它们。这三个控件与其他控件（如 DropDownList 及 ListBox）类似，但它们提供的功能要复杂得多。它们用作其他实际显示数据的控件（如标签）的容器（container）。这三个控件的功能非常强大，为开发人员省去了大量的工作。

正如您将在接下来的几节看到的，这些服务器控件各有其优缺点。实际上是功能和复杂性的折衷，即控件提供的功能越多，则使用起来越复杂。下面按复杂程度依次介绍这些控件。

### 9.4.1 Repeater 服务器控件

Repeater 服务器控件是一个可以遍历数据的容器。它没有预先设置好的显示方式，必须使用模板控件来指定其布局。Repeater 提供了一个可高度定制的界面，如果不指定其布局，控件将不会显示。

模板是一种控件，让您能够使用 HTML 标签、文本及其他控件来控制数据的显示。Repeater 控件必须和模板一起使用，该控件支持下列类型的模板：

- **ItemTemplate**: Repeater 控件必须使用该模板。ItemTemplate 针对每条记录显示在一行输出。可以使用其他控件来显示数据，方法是将合适的字段绑定到这些控件。
- **AlternatingItemTemplate**: 该控件同 ItemTemplate 相同，但它隔行针对每两条记录显示一次，这让您可以设置不同的样式，如记录的轮换颜色。
- **HeaderTemplate** 和 **FooterTemplate**: 这两个模板在所有数据记录的前后显示 HTML，其典型用法是使用<table>或</table>标签来开始或结束表格。
- **SeparatorTemplate**: 该模板在数据记录之间显示条目，如 HTML 标签<br>、<p>或<HR>。

上述模板没有特定的格式或语法，您只是将它们用作容器。我们来看一个典型的例子，如清单 9.8 所示。您将使用前一章创建的数据库表 tblUser。

**清单 9.8 一个典型的 Repeater 控件**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6   sub Page_Load(obj as Object, e as EventArgs)
7
8     'set up connection
9     dim myConnection as new OleDbConnection _
10         , "Provider=Microsoft.Jet.OLEDB.4.0;" & _
11         "Data Source=H:\ASPNET\data\banking.mdb"
12
13     'open connection
14     dim myCommand as new OleDbDataAdapter _

```

```

15      ('select * from tblUsers', myConnection)
16
17      'fill dataset
18      dim ds as DataSet = new DataSet()
19      myCommand.Fill(ds, "tblUsers")
20
21      'select data view and bind to server control
22      Repeater1.DataSource = ds.Tables("tblUsers").
23      DefaultView
24      DataBind()
25      end sub
26</script>
27
28<html><body>
29 <ASP:Repeater id="Repeater1" runat="server" >
30   <HeaderTemplate>
31     <table>
32       <tr>
33         <td bgcolor="#cccc99" width="200"><b>Name</b></td>
34         <td bgcolor="#cccc99" width="200"><b>Phone</b></td>
35       </tr>
36     </HeaderTemplate>
37
38     <ItemTemplate>
39       <tr>
40         <td> <%# Container.DataItem("FirstName") %>&nbsp;
41         <%# Container.DataItem("LastName") %>
42       </td>
43       <td> <%# Container.DataItem("Phone") %> </td>
44     </tr>
45   </ItemTemplate>
46
47   <AlternatingItemTemplate>
48     <tr>
49       <td bgcolor="#cccc99">
50         <%# Container.DataItem("FirstName") %>&nbsp;
51         <%# Container.DataItem("LastName") %>
52       </td>
53       <td bgcolor="#cccc99">
54         <%# Container.DataItem("Phone") %>

```

```

55         </td>
56     </tr>
57 </AlternatingItemTemplate>
58
59 <SeparatorTemplate>
60     <tr>
61         <td colspan="2" align="center">
62             - - -
63         </td>
64     </tr>
65 </SeparatorTemplate>
66
67 <FooterTemplate>
68     </table>
69 </FooterTemplate>
70 </ASP:Repeater>
71</body></html>

```

图 9.7 是在浏览器中查看清单 9.8 得到的结果。

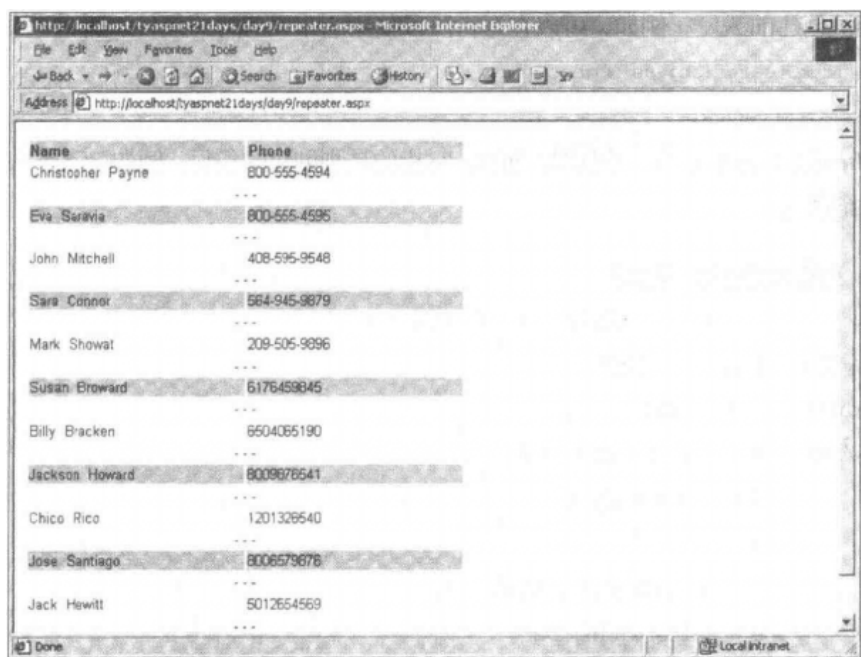


图 9.7 Repeater 控件 (使用模板)

分析: Page\_Load 事件处理程序是直接从前一章的清单 8.1 中复制过来的, 其中包含几个将在下一章介绍的对象。第 8—15 行建立并打开数据库连接, 第 19 行填充 DataSet 对象, 第 22 行将 DataView 绑定到 Repeater。

将数据绑定到 Repeater 控件的方法有两种 (本章后面将介绍的 DataList 和 DataGrid 控件也是如

此) 第一种方法是这里使用的: 将控件的 `DataSource` 属性设置为 `DataView`, 后者告知控件要显示的表和数据。另一种方法是将控件的 `DataSource` 属性设置为 `DataSet`, 并使用控件的 `DataMember` 属性指定要显示的数据。例如, 可以使用下列代码替代第 22-23 行:

```
Repeater1.DataSource = ds
Repeater1.DataMember = "tblUsers"
```

然后只需调用 `DataBind` 即可。这两种方法的功能相同, 本章将交替使用它们。之所以有两种方法, 是因为第二种方法让您能够轻松地使用 `DataSet` 来绑定数据, 而无需创建 `DataView`; 而第一种方法让您可以使用不同类型的对象来绑定数据。

接下来, 第 29 行声明了一个 `Repeater` 控件。标题模板只是创建了一个 HTML 表, 并输出字段标题 “Name” 和 “Phone”。接下来的条目模板将三个数据字段绑定到页面。`Container.DataItem` 是当前控件的父控件 (这里为 `Repeater`) 的数据字段集合。您将 “FirstName”、“LastName” 和 “Phone” 等三个字段绑定到该模板, 这样 `Repeater` 控件将自动遍历记录, 并将这些字段显示到页面上。`AlternatingItemTemplate` 模板的作用完全相同, 只是给表中的记录设置了不同的背景颜色。

第 59 行中的 `SeparatorTemplate` 只是使用几个短划线创建了一个新的 HTML 行, 以分隔记录 (如果轮换颜色不足以分隔的话)。最后, `FooterTemplate` 只是结束 HTML 表格, 之后是结束标记 `</asp:Repeater>`。

仅此而已! 从中可以知道, `Repeater` 控件使得显示数据是多么的容易, 并让您能够完全控制数据的显示方式。

`Repeater` 控件有两个您可以对其进行响应的事件: `ItemCreated` 和 `ItemCommand`。`ItemCreated` 在新条目或模板被创建前激发。例如, 可以在运行阶段使用它来设置样式属性。其语法如下:

```
sub_name_ItemCreated(obj as Object, e as _
    RepeaterItemEventArgs)
```

`RepeaterItemEventArgs` 包含一个属性: `Item`, 它指的是刚创建的条目。例如, 清单 9.9 列出了处理该事件的代码片段。

#### 清单 9.9 处理 Repeater 的事件

```
1: sub Repeater1_ItemCreated(obj as Object, e as _
2:     RepeaterItemEventArgs)
3:     dim type as String
4:     select case (e.Item.ItemType)
5:         case ListItemType.Item
6:             type = "Item"
7:         case ListItemType.AlternatingItem
8:             type = "AlternatingItem"
9:         case ListItemType.Header
10:            type = "Header"
11:         case ListItemType.Footer
12:            type = "Footer"
13:         case ListItemType.Separator
14:            type = "Separator"
```

```

15:         end select
16:
17:     ...
18:     Label1.Text = Label1.Text & " / " & Repeater1.Items.Item(i).Text
19:     " has been created successfully"
20: end sub
21: ...
22: <asp:Page_Load runat="server"
23: OnItemCreated="Repeater1_ItemCreated">
24: ...
25: </asp:Repeater>
26: <asp:Label id="Label1" runat="server">

```

标签将按顺序列出创建的模板类型。

当 Repeater 控件内的服务器控件引发事件时，ItemCommand 事件便被引发。内部控件将命令传给 Repeater。例如，如果在模板中放置了按钮或链接控件，则当用户单击该控件时，您可以检测到，并做出相应的反应。在每一个控件中都可以设置 Command 属性（包含一个字符串），该属性将被上传，这样处理程序便可以确定要采取何种措施。

也可以直接响应控件的事件，而无需通过 Repeater 的 ItemCommand 事件。然而，由于 Repeater 动态地创建这些控件，所以必须以某种方式获知各个控件的名称或通过集合来引用它们。

**注意：** Repeater 控件只是显示数据——它不允许用户编辑或修改实际的数据源。如果希望用户能够编辑或修改数据，可以使用 Web 控件 DataList 或 DataGrid，这些控件将在下一节介绍。

#### 9.4.2 DataList 服务器控件

DataList 控件与 Repeater 控件非常类似，只是它具有交互性，允许用户修改数据。可以在该控件中使用模板来像 Repeater 那样列出条目，但 DataList 还支持另外两种模板：

**SelectedItemTemplate：**该模板包含仅当用户选择 DataList 控件中的项目后才显示的元素。SelectedItemTemplate 通常用来修改样式属性以反映某记录被选中或扩展诸如层次列表（父子关系）这样的项目。

**EditItemTemplate：**该模板指定处于编辑模式的条目的布局。

我们来看一个例子。这里使用的 Page\_Load 事件与清单 9.8 相同，所以没有列出。清单 9.10 只列出 DataList 本身（注意，需要修改第 22 行，以引用 DataList 控件而不是 Repeater 控件）。

**清单 9.10 一个典型的 DataList 控件**

```

1  <asp:DataList id="DataList1" runat="server"
2      SelectItemStyle-BackColor="#cccc99"
3      RepeatLayout="Table"
4      RepeatDirection="horizontal"
5      DataKeyField="UserID">
6
7  <ItemTemplate>
8      <asp:LinkButton id="button1" runat="server"

```



```

9         Text= <%# Container.DataItem('FirstName') & " " & _
10             Container.DataItem("LastName") %>'
11         Command="select" />
12     <p>
13 </ItemTemplate>
14
15 <SelectedItemTemplate>
16     <%# Container.DataItem('FirstName') & " " & _
17         Container.DataItem("LastName") %><br>
18     Phone:
19     <%# Container.DataItem('Phone') %>
20     <br>
21     Address:
22     <%# Container.DataItem('Address') %>
23     <br>
24     <%# Container.DataItem('City') %>,
25     <%# Container.DataItem('State') %>
26     &nbsp;&nbsp;&nbsp;<%# Container.DataItem("ZIP") %>
27     <br>
28 </SelectedItemTemplate>
29 </asp:DataList>

```

**注意：**别忘了将DataList包含在<form>标记内！否则您为其定义的事件不会起作用。

第 1-5 行使用第 5 章讨论的很多服务器控件的属性声明 DataList 控件。SelectedItemStyle-BackColor 指定 SelectedItemTemplate 的背景颜色。DataKeyField 是用作主关键字的字段的名称。该主关键字将是唯一性标识符，在标识列表中的记录方面很有帮助。

从第 7 行开始的条目模板中，您只创建了一个 LinkButton 控件(第 8 行)，并被绑定到 FirstName 和 LastName 字段。被单击时，该连接将生成 Select 命令，并将其传递给 DataList。

最后，被选中的条目模板只是显示一些数据绑定信息。每当 DataList 中的条目(或数据记录)被选中时，该模板便显示该记录。但您首先需要定义用于处理链接被单击的方法(见清单 9.11)。

#### 清单 9.11 处理 DataList 的 Click 事件

```

1 sub DataList1_ItemCommand(obj as object, e as _
2     DataListCommandEventArgs)
3     DataList1.SelectedIndex = e.Item.ItemIndex
4     DataList1.DataBind()
5 end sub

```

将该过程加入到清单 9.10 的代码声明块中，并将下列属性添加到 DataList 中：

```
OnItemCommand="DataList1_ItemCommand"
```

该过程处理任何被传递给容器控件 DataList 的事件，包括 LinkButton 的 Click 事件，它接受一个特殊类型的事件参数：DataListCommandEventArgs，该参数包含以下属性：

- **CommandArgument**: 命令的参数属性;
- **CommandName**: 命令的名称;
- **CommandSource**: 命令的来源;
- **Item**: **DataList** 中选中的条目。

您定义的方法将 **DataList** 控件的 **SelectedIndex** 属性设置为用户选中的条目,并调用了 **DataBind** 方法。用户单击 **DataList** 中的条目时,该条目对应的 **SelectedItemTemplate** 将被显示,如图 9.8 所示。

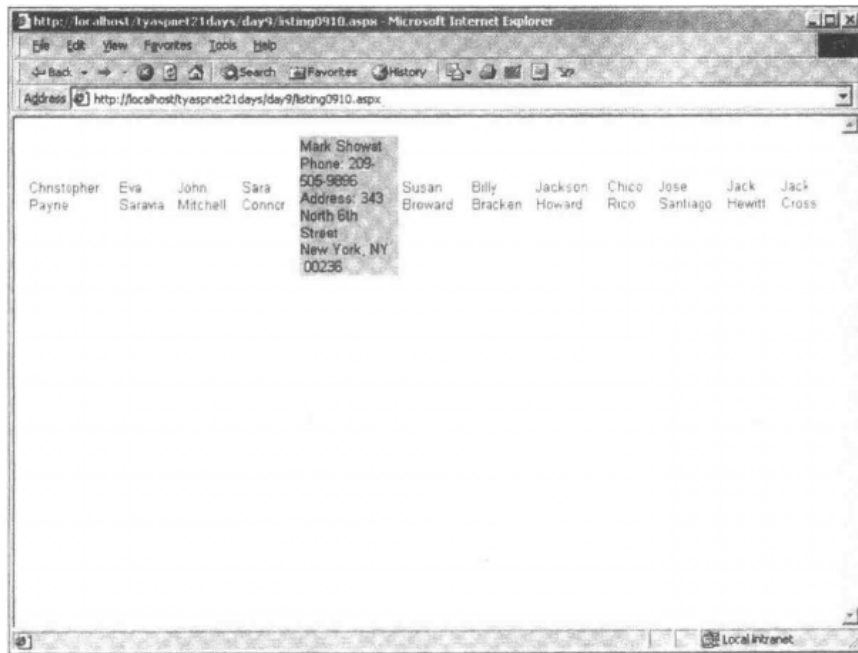


图 9.8 **DataList** 控件的运行结果

**注意:** **SelectedItemTemplate** 将取代任何为该条目显示的任何东西。这里显示的是用户的姓名。当上述模板被显示时,将覆盖这两个字段,因此只是在模板中重复这些信息,如第 16-17 行所示。

#### 编辑条目

**DataList** 控件还允许编辑所显示的条目,它含有特殊的命令,可以用来命令 ASP.NET 将某个条目切换到编辑模式。清单 9.12 在清单 9.10 的基础上进行了修改,列出所有可用的特殊命令。

#### 清单 9.12 在 **DataList** 中编辑条目

```

1  <asp:DataList id="DataList1" runat="server"
2  SelectedItemStyle-BackColor="#cccc99"
3  repeatlayout="table"
4  repeatdirection="horizontal"
5  OnItemCommand="DataList1_ItemCommand"
6  OnEditCommand="DataList1_EditCommand"
7  OnCancelCommand="DataList1_CancelCommand"
8  OnUpdateCommand="DataList1_UpdateCommand"

```

```

9   OnDeleteCommand="DataList1_DeleteCommand"
10  DataKeyField="UserID" >
11
12  <ItemTemplate>
13      <asp:LinkButton id="button1" runat="server"
14          Text=" <%= Container.DataItem("FirstName") & " " & _
15              Container.DataItem("LastName") %>
16          CommandName="Edit" />
17      <p>
18  </ItemTemplate>
19
20  <EditItemTemplate>
21      <asp:LinkButton id="lbtCancel" runat="server"
22          CommandName="Cancel"
23          Text="Cancel" />
24      <asp:LinkButton id="lbtUpdate" runat="server"
25          CommandName="Update"
26          Text="Update" />
27      <asp:LinkButton id="lbtDelete" runat="server"
28          CommandName="Delete"
29          Text="Delete" />
30  </EditItemTemplate>
31 </asp:DataList>

```

**分析：**该清单还不能运行，还必须首先定义方法来处理其事件——稍后将介绍。当用户单击条目时，将出现一个选择菜单，这要归功于第 20 行的 EditItemTemplate。我们先来看看该清单中的代码。

第 16 行将 LinkButton 的 Select 命令改为 Edit。这是 ASP.NET 保留的一个特殊的命令，它自动引发 DataList 的 EditCommand 事件。然后，您必须声明该事件的处理程序，如下所示：

```

sub DataList1_EditCommand(obj as object, e as _
    DataListCommandEventArgs)
    DataList1.EditItemIndex = e.Item.ItemIndex
    DataList1.DataBind()
end sub

```

将该方法加入到代码声明块中，它只是将 DataList 的 EditItemIndex 属性设置为被选中的条目，然后 DataList 显示（从第 20 行开始的）EditItemTemplate 模板，并将所选条目切换到编辑状态。通常，将条目置为编辑状态后，用户不但可以修改该条目，还可以更新数据库。下一章将更详细地介绍这方面的内容。您新建了链接按钮，它们包含三个特殊的命令：Cancel、Update 和 Delete。这些链接按钮如第 21-29 行所示。当这些命令被上传时，将分别引发 DataList 的 CancelCommand、UpdateCommand 和 DeleteCommand 命令。可以像为 EditCommand 命令创建处理程序那样为这三个命令创建处理程序：

```

sub DataList1_CancelCommand(obj as object, e as _
    DataListCommandEventArgs)
    DataList1.EditItemIndex = -1
    DataList1.DataBind()
end sub

sub DataList1_UpdateCommand(obj as object, e as _
    DataListCommandEventArgs)
    ' update data store
    DataList1.DataBind()
end sub

sub DataList1_DeleteCommand(obj as object, e as _
    DataListCommandEventArgs)
    ' delete from data store
    DataList1.DataBind()
end sub

```

将上述方法添加到代码声明块中。要让条目退出编辑状态，只需将 `EditItemIndex` 设置为 -1 即可。要真正地更新或删除数据库中的数据，必须手工创建方法。记住一定要再次调用数据绑定方法，否则 `DataList` 不会被更新，从而无法反映发生的事件或变化。最后，定义好所有的事件处理程序后，可以在浏览器中查看该清单。图 9.9 显示了单击条目，并打开编辑状态时的结果。

当前，单击 `Update` 和 `Delete` 按钮不会完成任何工作。但是，单击 `Cancel` 可从编辑模式切换到选择模式，其结果类似于图 9.8。

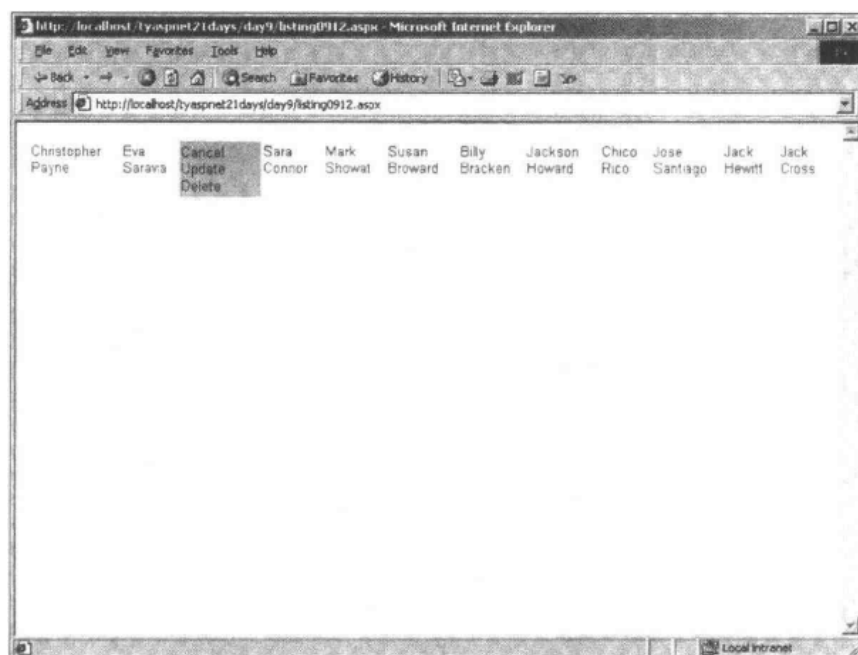


图 9.9 将条目切换到编辑模式

### 9.4.3 DataGrid 服务器控件

DataGrid 控件类似于 DataList 和 Repeater 控件，只是 DataGrid 具有更多的功能。该控件使用列将数据显示在网格中。默认情况下，DataGrid 为数据库中的每个字段都创建一列，但可以指定要显示的字段及其显示方式。您可以定义下列类型的列：

- **绑定列：**让您能够指定要显示的列、以何种顺序显示这些列以及格式化样式属性。默认情况下，DataGrid 使用的是绑定列。
- **按钮列：**将网格中的条目显示为按钮，供您指定自定义功能。一个典型的例子是 Add to Shopping Cart 按钮。
- **编辑命令列：**让您能够编辑条目，使用可修改的列代替绑定列。
- **超链接列：**将数据显示为超链接。
- **模板列：**与 Repeater 和 DataList 控件相同，可使用模板给字段定义自定义格式。

DataGrid 控件根据要显示的数据自动选择列的类型，但可以很容易地修改默认行为。清单 9.13 给出一个例子。

**清单 9.13 DataGrid 范例**

```

1  <asp:DataGrid id="DataGrid1" runat="server"
2      BorderColor="black"
3      GridLines="Vertical"
4      cellpadding="4"
5      cellspacing="0"
6      width="450"
7      Font-Names="Arial"
8      Font-Size="8pt"
9      ShowFooter="True"
10     HeaderStyle-BackColor="#cccc99"
11     FooterStyle-BackColor="#cccc99"
12     ItemStyle-BackColor="#ffffff"
13     AlternatingItemStyle-BackColor="#cccccc"
14     AutoGenerateColumns="false">
15
16     <Columns>
17
18         <asp:TemplateColumn HeaderText="Name">
19             <ItemTemplate>
20                 <asp:Label id="Name" runat="server"
21                     Text="<%# Container.DataItem("FirstName") & _
22                         " " & Container.DataItem("LastName") %>" />
23             </ItemTemplate>
24         </asp:TemplateColumn>
25

```

```

26 <asp:BoundColumn HeaderText="Address"
27     DataField='Address' />
28
29 <asp:BoundColumn HeaderText="City" DataField="City" />
30
31 <asp:BoundColumn HeaderText="State"
32     DataField="State" />
33
34 <asp:BoundColumn HeaderText="Zip" DataField="Zip" />
35
36 <asp:HyperlinkColumn HeaderText="Edit" text="Edit"
37     NavigateURL="edit.aspx" />
38
39 <asp:ButtonColumn HeaderText="Delete?" text="X"
40     CommandName="delete"
41     ButtonType="PushButton" />
42
43 </Columns>
44
45 </asp:DataGrid>

```

清单 9.13 列出了 DataGrid 的各种属性以及如何为数据创建列。清单 9.13 使用的 Page\_Load 事件和清单 9.10 相同（只是名称从 ListBox 改为 DataGrid），其运行结果如图 9.10 所示。

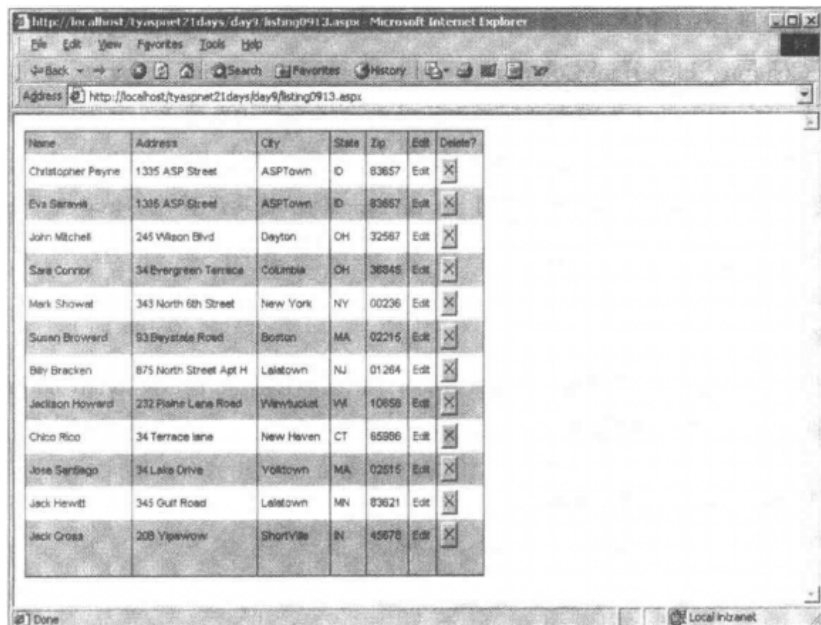


图 9.10 DataGrid 控件的运行结果

**分析：**该控件拥有很多可操纵的属性。第 2-13 行设置了一些您熟悉的属性。第 14 行告知 DataGrid：您想设置自己的列。将在下一节处理这些列。如果将 AutoGenerateColumns 设置为 true，DataGrid 将自动生成列，您不必指定自定义的列。AutoGenerateColumns 使得 DataGrid 使用数据源中所有可用的字段，因此可能更愿意自己定义列，这样可以限制用户可以看到哪些数据。

**注意：**如果将 AutoGenerateColumns 设置为 true，并提供自定义列定义，则 DataGrid 将同时显示它们，这将显示所有的字段以及您定义的字段。这意味着将出现多个重复的字段。例如，在清单 9.13 的第 14 行将 AutoGenerateColumns 设置为 true 将得到如图 9.11 所示的结果。

Name	Address	City	State	Zip	Edit	Delete	First Name	Last Name	Address	City	State	Zip	Phone	User ID
Christopher Payne	1335 ASP Street	ASPTown	D	83657	Edit	X	Christopher	Payne	1335 ASP Street	ASPTown	D	83657	800-655-4594	1
Eve Saravia	1335 ASP Street	ASPTown	D	83657	Edit	X	Eve	Saravia	1335 ASP Street	ASPTown	D	83657	800-655-4595	2
John Mitchell	245 Wilson Blvd	Dayton	OH	32597	Edit	X	John	Mitchell	245 Wilson Blvd	Dayton	OH	32597	408-595-9548	3
Sara Connor	34 Evergreen Terrace	Columbus	OH	39845	Edit	X	Sara	Connor	34 Evergreen Terrace	Columbus	OH	39845	984-945-9878	4
Mark Showat	343 North 8th Street	New York	NY	00236	Edit	X	Mark	Showat	343 North 8th Street	New York	NY	00236	209-505-8599	5
Susan Broward	93 Beverly Road	Boston	MA	02215	Edit	X	Susan	Broward	93 Beverly Road	Boston	MA	02215	617-949-6645	6
Billy Bracken	675 North Street Apt H	Lakewood	NJ	01204	Edit	X	Billy	Bracken	675 North Street Apt H	Lakewood	NJ	01204	650-4060-90	8
Jackson Howard	232 Plains Lane Road	Waukegan	WI	10695	Edit	X	Jackson	Howard	232 Plains Lane Road	Waukegan	WI	10695	800-987-8841	9
Chico Rico	34 Terrace Lane	New Haven	CT	65986	Edit	X	Chico	Rico	34 Terrace Lane	New Haven	CT	65986	1201326540	11

图 9.11 AutoGenerateColumns=true 将导致显示所有的列以及自定义列

要指定自定义列，必须将其加入到 DataGrid 控件的 Columns 集合中，这可以在运行阶段实现，也可以在设计阶段进行，这里使用的是后一种方法。第 16 行使用 <Columns> 标记开始定义自定义列。

TemplateColumn、HyperLinkColumn、BoundColumn 和 ButtonColumn 是专用于 DataGrid 的 ASP.NET 控件。图 9.10 包含不同类型列的输出结果。这些列相当简单，只有几个属性是您没有见过的。BoundColumns 控件使用 DataField 属性来将列绑定到数据库字段。默认情况下，ButtonColumn 控件显示的是 LinkButton 控件，但可以使用 ButtonType 属性将其设置为其他类型，如第 41 行所示。

请随便设定 DataGrid 的设置。您将发现对显示数据库数据，该控件很有用，同时它允许您定制显示方式的各个方面。

#### 1. 编辑条目

DataGrid 控件也使得允许用户编辑条目更简单。您只需创建一个 EditCommandColumn，ASP.NET 将完成大部分的显示工作。让我们对清单 9.13 稍做修改，加入一个编辑命令列，如清单 9.14 所示。

#### 清单 9.14 包含可编辑列的 DataGrid

```
1 <asp:DataGrid id="DataGrid1" runat="server">
2 ...
3 ...
4     AutoGenerateColumns="false"
5     OnEditCommand="DataGrid1_Edit"
```

```

6   OnCancelCommand="DataGrid1_Cancel"
7   OnUpdateCommand="DataGrid1_Update"   >
8
9   <Columns>
10
11   ...
12
13   ...
14   <asp:EditCommandColumn
15       EditText="Edit"
16       CancelText="Cancel"
17       UpdateText="Update"
18       ItemStyle-Wrap="false"
19       HeaderText="Edit" />
20
21   </Columns>
22
23 </asp:DataGrid>

```

分析：由于新建了 `EditCommandColumn`，因此可以删除清单 9.13 中第 36 行的 `HyperLinkColumn`，还可以删除第 39 行中的 `ButtonColumn`，因为 `EditCommandColumn` 将为您自动添加一个。其他地方保持不变，第 5-7 行代码告知 `DataGrid` 使用哪个处理程序来处理 `Edit`、`Cancel` 和 `Update` 事件。第 14 声明了 `EditCommandColumn`，它显示一个 `LinkButton`，其中包含文本 `Edit`。用户单击该链接后，被选中的条目将进入编辑模式，而每个绑定的列则变为一个用户可修改的文本框。图 9.12 是单击“Edit”按钮后得到的结果。

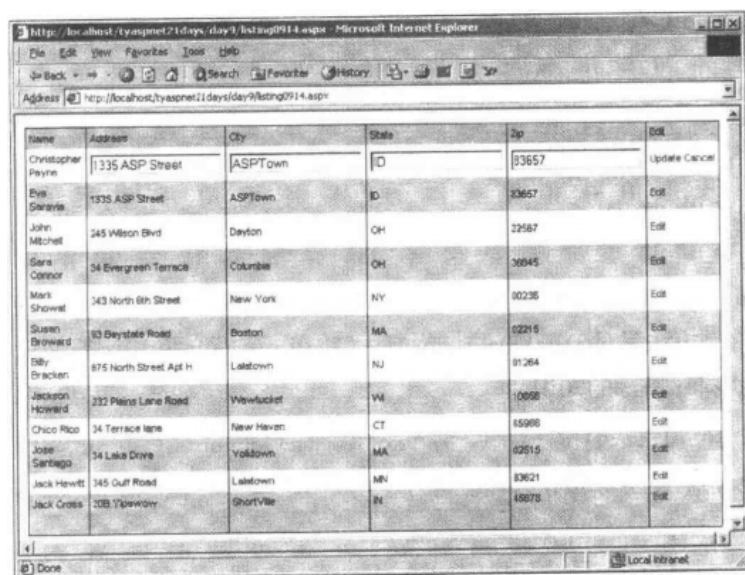


图 9.12 `EditCommandColumn` 让用户可以修改 `DataGrid` 中的绑定列



**注意：**只有BoundColumns会变为可修改的字段！其他列（如TemplateColumns）仍保持原有的状况。一定要确保您希望用户可以编辑的每个类要么是BoundColumn，要么包含用于编辑的文本框。EditCommandColumn 还自动显示链接 Update 和 Cancel。您也可以定义方法来处理这些事件：

```
sub DataGrid1_Edit(obj as object, e as DataGridCommandEventArgs)
    DataGrid1.EditItemIndex = e.Item.ItemIndex
    DataGrid1.DataBind()
end sub

sub DataGrid1_Update(obj as object, e as DataGridCommandEventArgs)
    'do updates
    DataGrid1.DataBind()
end sub

sub DataGrid1_Cancel(obj as object, e as DataGridCommandEventArgs)
    DataGrid1.EditItemIndex = -1
    DataGrid1.DataBind()
end sub
```

这些方法与 DataList 类似。

## 2. 排序

DataGrid 控件本身不支持对记录进行排序，但它让您能够使用内置的事件或显示功能来对记录进行排序。启用排序功能后，默认情况下，DataGrid 将每个列标题转换为 LinkButton，用户可单击链接按钮，按相应的列进行排序。虽然您必须自己创建排序机制，但 ASP.NET 为您提供了基础。您还可以自定义排序方式，从中可定义用户用于排序的列，您甚至还可以提供自定义的排序链接。

要启用排序功能，只要在 DataGrid 声明中加入 AllowSorting="true" 即可。然后创建一个名为 SortCommand 的方法来处理排序事件：

```
sub DataGrid1_SortCommand(obj as Object, _
    e as DataGridSortCommandEventArgs)
    'sort the data using the SortField property
    ' of the eventargs
    DataGrid1.DataBind()
end sub
```

参数 DataGridCommandEventArgs 的 SortField 属性指出被单击的列，您可以使用这种信息来相应地对数据进行排序，方法是对 DataView 进行排序（将在下一章介绍）或修改 SQL 命令。下一章将更详细地介绍数据排序方面的知识。

**注意：**如果想使用默认的排序机制，将 AutoGenerateColumn 设置为 true，允许 DataGrid 自动创建列。否则，排序功能将不会像预期的那样运行。图 9.13 显示了一个使用 AutoGenerateColumn=true 启用排序功能的例子。

要使用自定义排序机制，请启用排序功能，并为可用于排序的每个列指定一个 SortField。例如，如果想允许用户根据 address 对列表进行排序，可使用以下代码：

Firstname	Lastname	Address	City	State	Zip	Phone	UserID	Password	Username
Christopher	Payne	1335 ASP Street	ASPTown	ID	83657	800-555-4594	1	chrisman	clpayne
Eve	Sarinio	1335 ASP Street	ASPTown	ID	83657	800-555-4595	2	yoyo	esarinio
John	Mitchell	245 Wilson Blvd	Dayton	OH	32567	408-595-9548	3	sname	jmtchell
Sara	Connor	34 Evergreen Terrace	Columb	OH	38845	594-945-9879	4	rlfara	sconnor
Mark	Showat	343 North 8th Street	New York	NY	80236	206-505-9956	5	nickymouse	mshowat
Susan	Broward	93 Berenstate Road	Boston	MA	02215	8178459845	6	yellowdog	susarb
Billy	Brackon	875 North Street Apt H	Lalatown	NJ	01264	8504065190	8	happymeal	billyb
Jackson	Howard	232 Plains Lane Road	Viewlucky	VA	10656	8093076541	9	sevenup	jacksj
Chico	Rico	34 Terrace Lane	New Haven	CT	65986	1201326540	11	borderman	chicorico
Jose	Santiago	54 Lake Park	Yalltown	MA	02515	8006579876	12	papamatz23	jssantiago

图 9.13 当 AllowSorting=True 时, 所有列标题都变成可用于进行排序的链接

```

<asp:BoundColumn HeaderText="Address" DataField="Address"
    sortField="Address" />

```

### 3. 分页

DataGrid 具有分页的功能, 如果一个页面上显示的记录过多, DataGrid 可将数据分成多个页面。分页一度是一项非常复杂的功能, 开发人员必须自己实现, 但现在 ASP.NET 可代为处理。

设置分页功能后, ASP.NET 将根据您指定的页数来划分返回的数据, 并提供按钮, 供用户在列表中导航。然后, 它使用 DataGrid 的 CurrentPageIndex 属性来确定当前应显示的页面。当用户单击进入按钮, 进入下一页时, 将重新创建整个数据集, 并重新进行处理。如果数据量很大, 则数据装载的时间将很长。因此, DataGrid 控件允许您指定自定义的规程, 以避免数据装载时间过长的问題。

要启用分页功能, 请将 AllowPaging 设置为 True (默认情况下为 True), 将 PageSize 属性设置为页面一次显示的记录数。通过使用 PageStyle 属性, 还可以设置分页按钮的样式。两个内置的样式是 Next (上一页) 和 Previous (下一页) 按钮以及页码。我们来修改前面的清单, 以包含分页功能, 如清单 9.15 所示。

#### 清单 9.15 在 DataGrid 中加入分页功能

```

1 sub DataGrid1_PageIndexChanged(obj as Object, e as _
  DataGridPageChangedEventArgs)
2     DataGrid1.CurrentPageIndex = e.NewPageIndex
3     DataGrid1.DataBind()
4 end sub
5 ...
6 ...
7 <asp:DataGrid id="DataGrid1" runat="server"
8     ...

```

```

9   AllowPaging="True"
10  PageSize=2
11  PagerStyle-Mode=NumericPages
12  PagerStyle-PageButtonCount = 2
13  OnPageIndexChanged='DataGrid1_PageIndexChanged' >

```

您首先定义了一个新方法 `DataGrid1_PageIndexChanged` 来处理分页事件。因为 `DataGrid` 不会自动分页，所以需要您自己完成这项功能，如第 2 行所示。`DataGridPageChangedEventArgs` 包含 `NewPageIndex` 属性，用于指出用户单击的页码。您只需将 `DataGrid` 的 `CurrentPageIndex` 设置为被单击的页码，重新将数据绑定到控件，其余的工作将由 ASP.NET 来完成。

在 `DataGrid` 控件中，您添加了 5 个新属性。第 9 行启用分页功能（这是默认情况）。第 10 行指定每页显示两条记录。第 11 行将分页模式设置为使用数字页码。分页模式可以是 `NumericPages` 或 `NextPrev`，后者显示按钮 `Next` 和 `Previous`。第 12 行命令网格每次只显示两个分页导航的页码——仅当模式为 `NumericPages` 时，该属性才有效。如果页面数多于两个，用户将看到一个可单击的省略号。最后为 `PageIndexChanged` 事件声明了处理程序。图 9.14 显示了分页功能的运行结果。

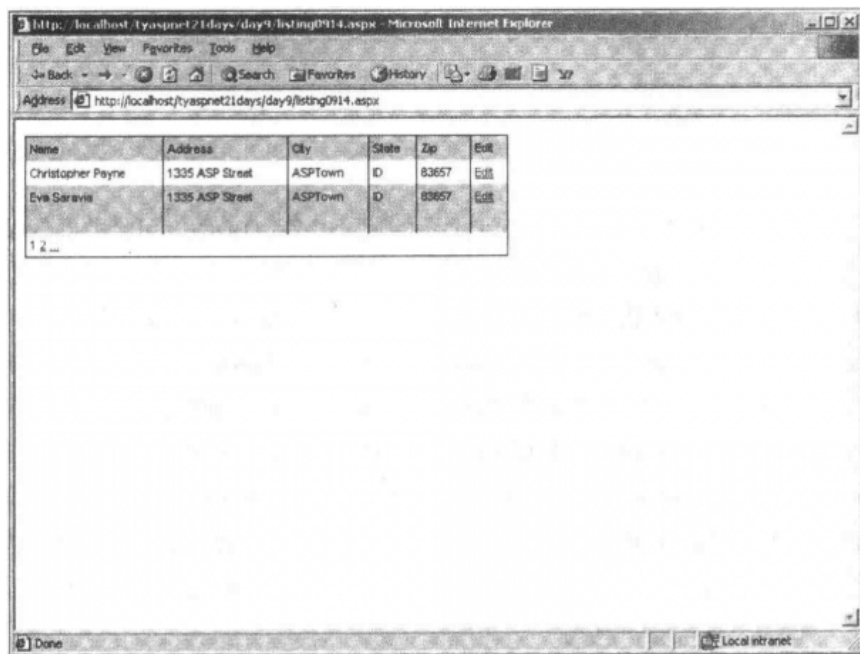


图 9.14 设置了分页功能的 `DataGrid` 控件

如果不想使用内置的分页按钮，可以创建自定义按钮。只需将 `PagerStyle-Visible` 属性设置为 `false`，并将自己的按钮放在所需的位置（如页眉或页脚）即可。您还需要创建自己的导航方法来处理这些事件——在代码中修改 `CurrentPageIndex` 属性，以便在页面间切换。

`DataGrid` 还让您能够使用手工分页功能来更好地控制分页过程，这涉及到 ADO.NET 机制，所以将在下一章讨论这一问题。

#### 9.4.4 数据绑定控件小结

现在您明白了为什么第 5 章没有介绍这三个控件——它们相当复杂！这几个控件使得显示动态数据非常简单，而无论使用的是哪种数据源，而这一度是 ASP 开发人员所面临的最大困难之一。

鉴于选择的余地很大,而这些控件又有很多相同的特性,因此在确定使用哪一个控件以及在何处使用方面,开发人员常常感到困惑。表 9.2 对这些控件做了总结,并就在什么情况下应该使用哪个控件提出了建议。

表 9.2 ASP.NET 数据绑定服务器控件总结

控 件	特性/何时使用
Repeater	<p>用于简单、只读的输出,本身不支持选择和编辑功能</p> <p>默认情况下不显示;必须使用模板手工指定</p> <p>没有分页机制</p> <p>仅当所显示数据很简单时才应该使用该控件。对于一维数组和集合,该控件很有用,并且不论在哪里使用,其效率都很高,同时是轻量级的</p>
DataList	<p>提供默认、可自定义的表格显示功能</p> <p>内容可编辑</p> <p>单个或多个选区</p> <p>可选中多列</p> <p>本身不具备分页功能</p> <p>DataList 比 Repeater 更高级,提供了更多的功能,允许用户选择和编辑条目。如果需要显示与用户进行交互的简单数据或二维数据,则 DataList 是理想选择。DataList 典型的用途是显示 Windows 资源管理器风格的文件层次结构。</p>
DataGrid	<p>提供一种默认的、可高度定制的网格显示</p> <p>内容可编辑,包括删除</p> <p>单个或多个选区</p> <p>本身支持分页功能</p> <p>支持自定义功能</p> <p>三个控件中该控件功能最强大。它提供了全套的显示功能、全套的自定义功能、强大的分页、排序和编辑功能。需要允许用户与数据交互或需要其数据组织功能时,请使用该控件。建议该控件用于复杂数据库中的数据。典型的用途是用于电子商务应用程序的购物车中,用户可直接更新、添加及删除商品。</p> <p>看起来任何时候使用该控件都是个好主意,但该控件的开销很大,对那些数据量少,并要求快速显示的场合不太适合。</p>

我们来创建一个例子,进一步巩固对数据绑定和这些控件的理解。您将创建一个简单的 DataGrid 程序应用(一个行着色器),如果您乐意的话可将其背景色设置为存储在 DataSet 中的值。这看起来好像没什么大不了的,但它涵盖了本章介绍的很多知识。图 9.15 是这个例子的运行结果。

首先要确定如何取得数据。我们使用一个自定义的 DataSet,避免涉及到数据库。可以使用该 DataSet 来填充 DataGrid。下面来看看创建 DataSet 的代码,清单 9.16 显示了一个方法,稍后将在其中添加代码。

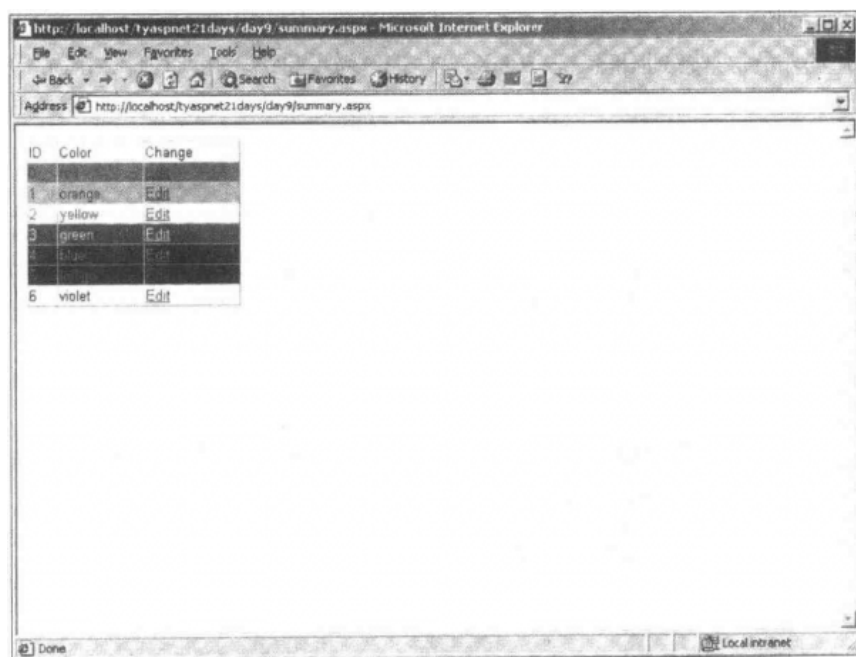


图 9.15 完成的行着色应用程序

## 清单 9.16 创建包含彩虹颜色的 DataSet

```

1  function CreateDataSet as DataSet
2      'create an array of colors
3      dim i as integer
4      dim arrColors() as String
5      if ViewState("Colors") is nothing then
6          arrColors = new String(6) {"red", "orange", _
7              "yellow", "green", "blue", "indigo", "violet"}
8          ViewState("Colors") = arrColors
9      else
10         arrColors = ViewState("Colors")
11     end if
12
13     'create an empty dataset
14     ds = new DataSet("MyDataSet")
15
16     'create a new table and columns
17     dim dTable as New DataTable("Colors")
18     dTable.Columns.Add("Color", GetType(String))
19     dTable.Columns.Add("ID", GetType(Int32))
20     'add table
21     ds.Tables.Add(dTable)
22

```

```

23      'add rows
24      for i = 0 to 6
25          dim dr as DataRow = dTable.NewRow()
26          dr(0) = arrColors(i).ToString
27          dr(1) = i
28          dTable.Rows.Add(dr)
29      next
30      .
31      return ds
32  end function

```

将该清单保存到文件 Summary.aspx 的代码声明块中（稍后将在该文件中加入其他一些方法和 UI 元素）。该清单类似于清单 9.1——本章创建的第一个 DataSet。您使用彩虹的颜色（red、yellow、green、blue、indigo 和 violet）作为数据。

我们来看看第 5-11 行，看起来有些可怕，但不用担心，其实并不可怕！想想典型情况下的行着色器。用户访问页面时，看到 DataGrid 中有颜色各不相同的几行，这些颜色存储在您创建的数组中。每当页面载入的时候，该数组都将被重新创建，而 DataGrid 则使用它来设置颜色。然后，用户编辑网格的颜色，很可能将第 1 行从红色更改为蓝色，但结果并非如此。当用户再次提交表单后，该数组将重新被创建，因此 DataGrid 将使用新创建的数组，这是因为负责创建数组的 Page\_Load 事件在处理 DataGrid 的更新事件的方法之前执行。因此，更新方法加入的不是新数据，而是以前的数据。图 9.16 说明了该过程。

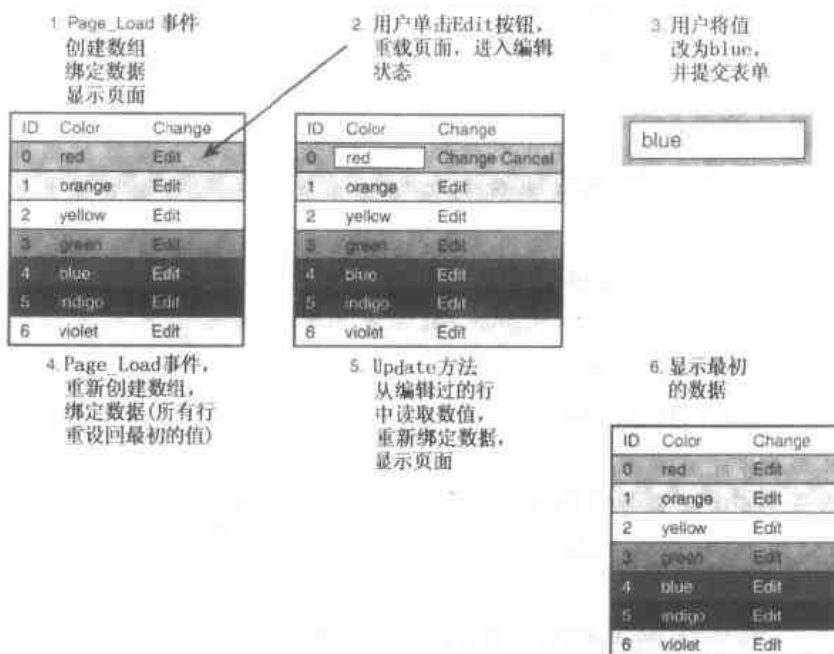


图 9.16 更新事件接收不到更改后的数据

您可能会问，第 4 步是否可以不绑定数据，这样做能够解决问题吗？不幸的是，不能。如果不绑定数据，DataGrid 将不包含任何数据，接下来的方法也不能访问这些数据。因此，第 4 步不绑定

数据将导致错误。

因此,需要将颜色数组存储在一个特殊的地方,以便页面提交时都不会被重新初始化为原来的值。可以使用状态包(更详细的信息,请参见第5章)。首次创建数组时,您将它存储在状态包中。随后载入页面和调用方法时,将引用该数组(该数组只被初始化一次)。更新方法可以修改状态包中数组,这样便能正常运行。我们来看看代码,这样将更有意义。

第5行检查状态包中是否存在数组。如果没有(首次查看该页面前,该数组不存在),则第6行、7行实例化一个数组;否则(即数组已被创建,或许被修改过)读取它。第13~29行创建了一个 DataSet,并将 DataTable 及数组中的信息加入到其中,现在您应该知道如何完成这项任务了。您使用第2行的值实例化一个数组,并使用一个 for 循环将它们添加到 DataSet 中,而不是手工创建7条记录,如24~29行所示。最后,给调用该函数的方法返回 DataSet,该方法可能将 DataSet 绑定到 DataGrid。

我们来看看 DataGrid 的声明,如清单9.17所示。

**清单 9.17 显示彩虹颜色的 DataGrid**

```

1 <html><body>
2   <form runat="server">
3     <ASP:DataGrid id="dgColors" runat="server"
4       AutogeneratedColumns="false"
5       width="200"
6       OnEditCommand="dgColors_Edit"
7       OnCancelCommand="dgColors_Cancel"
8       OnUpdateCommand="dgColors_Update"
9       OnItemCreated="ChangeColor" >
10
11     <Columns>
12       <asp:templateColumn headertext="ID">
13         <ItemTemplate>
14           <asp:Label id="lblID" runat="server"
15             text="<%= Container.DataItem("ID") %>" />
16         </ItemTemplate>
17       </asp:templatecolumn>
18
19       <asp:BoundColumn datafield="Color"
20         headertext="Color" />
21
22       <asp:EditCommandColumn headertext="Change"
23         EditText="Edit"
24         UpdateText="Change"
25         CancelText="Cancel" />
26     </Columns>

```

```

24         </ASP:DataGrid>
25     </form>
26 </body></html>

```

将该清单加入到文件 `summary.aspx` 的最后。这是一个标准的 `DataGrid`，包含处理 `Edit`、`Cancel`、`Update` 和 `ItemCreated` 事件的处理程序。注意，第 2 列（`BoundColumn`）包含 `DataSet` 的颜色，它是可编辑的。`EditCommandColumn` 将自动地显示所有的链接，以使用户能够进行编辑、更新和撤销。清单 9.18 列出了该页面中代码声明块的其他代码。

**清单 9.18 完整的 Summary.aspx**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     dim ds as DataSet
7     dim blnSet as Boolean = false
8
9     sub Page_Load(obj as Object, e as EventArgs)
10         ds = CreateDataSet
11         blnSet = true
12         if not Page.IsPostBack then
13             BindGrid
14         end if
15     end sub
16
17     sub BindGrid()
18         dgColors.DataSource = ds
19         dgColors.DataMember = "Colors"
20         DataBind()
21     end sub
22
23     sub ChangeColor(obj as object, e as DataGridItemEventArgs)
24         dim intIndex as Integer = e.Item.ItemIndex
25
26         if blnSet then
27             if intIndex > 0 then
28                 dgColors.Items(intIndex - 1).BackColor = _
29                     Drawing.Color.FromName(ds.Tables("Colors")._
30                         Rows(intIndex-1)("Color"))
31                 dgColors.Items(intIndex - 1).ForeColor = _

```



```

12         Drawing.Color.FromName(ds.Tables("Colors"). _
13             Rows(16-intIndex)("Color"))
14     end if
15 end if
16 end sub
17
18 sub dgColor_Edit(obj as object, e as DataGridCommandEventArgs)
19     dgColors.EditItemIndex = e.Item.ItemIndex
20     BindGrid
21 end sub
22
23 sub dgColors_Cancel(obj as object, e as DataGridCommandEventArgs)
24     dgColors.EditItemIndex = -1
25     BindGrid()
26 end sub
27
28 sub dgColors_Update(obj as object, e as DataGridCommandEventArgs)
29     dim strColor as string = CType(e.Item.Cells(1), _
30         Controls(0), TextBox).Text
31
32     ds.Tables("Colors").Rows(e.Item.ItemIndex) _
33         ("Color") = strColor
34     ViewState("Colors")(e.Item.ItemIndex) = strColor
35
36     dgColors.EditItemIndex = -1
37     BindGrid
38 end sub
39 </script>

```

该清单包含了除 HTML 和 `CreateDataSet` 函数外的所有代码。

第 6-7 行声明了两个在整个页面中都将使用的变量：一个是 `DataSet` 对象，另一个是布尔型的 `BinSet`，稍后将介绍这两个变量。从第 9 行开始的 `Page_Load` 方法使用 `CreateDataSet` 函数返回的数据填充（从第 6 行开始的）`DataGrid`。如果页面不能自动实现提交，页面上的控件将绑定到一个自定义函数 `BindGrid`，该函数将在稍后介绍。

该方法还有另一项功能：将 `binSet` 变量设置为 `true`。该变量指出数据的创建时间——`binSet` 最初被设置为 `false`（第 7 行），因为 `CreateDataSet` 方法还没有执行。该方法执行后，`binSet` 被置为 `true`。稍后我们还会讨论这一点。

接下来，第 17 行是 `BindGrid` 方法，该方法设置 `DataGrid` 的 `DataSource` 和 `DataMember` 属性，并调用 `DataBind`。

回到第 23 行的 `ChangeColor` 方法之前，我们来看看第 38、43 和 48 行的方法。前两个方法 `dgColors_Edit` 和 `dgColor_Cancel` 只是“打开”和“关闭”编辑模式，并重新绑定数据。第一个方法将

EditItemIndex 属性设置为用户选中的条目，而第二个方法将其值设置为-1，以关闭编辑模式。然后数据被重新绑定，以使修改生效。

第 48 行的 dgColor\_Update 方法稍微复杂些。首先，第 49 行取得 DataGrid 事件参数中被修改后的值。这个值存储在第一个控件中，即每行的第二个单元格，记作 Item.Cell(1).Controls(0)。由于返回的是一个通用控件，因此必须将其强制转换为文本框，然后使用 Text 属性来取得值。第 52 行更新了 DataSet 中相应行的值。准确地说，ds.Tables("Colors").Rows(e.ItemIndex)("Color")指的是 ds（您创建的 DataSet）的 Color 表中，与用户在 DataGrid 选中的行对应的记录的 Color 字段（该语句很长，但很合理）。

您将这个值设置为 DataGrid 中修改后的值，并在第 54 行中对存储在状态包中的颜色数组执行相同的操作，以便下次创建 DataGrid 时，使用更新后的数据而不是最初的数据。然后退出编辑模式。最后，重新绑定 DataGrid。

现在，用户对数据所做的修改将被保留，但是工作还没有结束！您必须根据 DataSet 中的相应值给各行上色。为此，将使用 DataGrid 的 Item\_Created 事件，该事件在网格中每一条目被创建前激发。该事件的处理程序将在后面讨论。

每当在 DataGrid 中创建新行时，第 23 行中的 ChangeColor 方法便被执行。由于颜色数组中有 7 个元素，所以每当页面被请求时，将至少执行该方法 7 次。第 24 行检索 DataGrid 中的行索引号，以便以后使用。第 26-35 行负责将各行的颜色更改为 DataSet 中的相应值。这里有几个需要注意的地方，下面进行更详细的介绍。

第 26 行再次检查 binSet，原因很简单：该变量指出是否创建了 DataSet。第 27-34 行访问 DataSet 中的记录，如果 DataSet 还没有被创建，将出现错误。因此，必须检查数据是否可以存取。

等一等，Page\_Load 方法不是总在该事件之前执行吗？Page\_Load 方法不会创建数据吗？从理论上说，在 ChangeColor 方法执行之前，数据已经创建了，那为什么还要进行检查呢？正常情况下，这没有问题，但 DataGrid 不是一般的服务器控件——其处理方式有些不同。

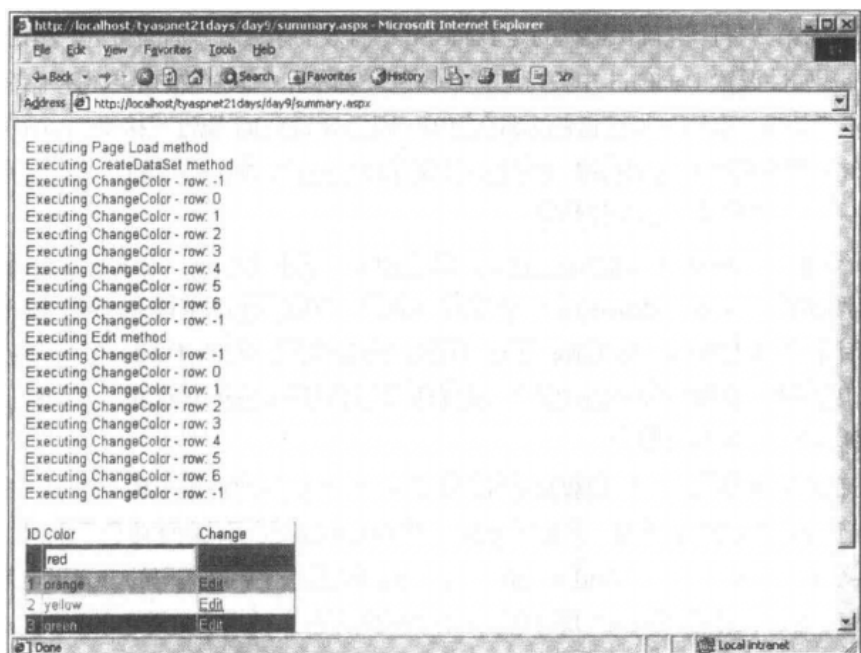


图 9.17 单击 Edit 按钮时，首先执行的是 Page\_Load

首次查看 DataGrid 或单击 Edit 按钮时, DataGrid 控件的工作原理同其他服务器控件完全相同: 首先执行 Page\_Load 方法, 然后执行事件处理程序 (不以特定的顺序), 包括 ItemCreated 事件的处理程序。在这个阶段, 不会出现问题。但用户单击 Update、Cancel 和 Delete 按钮后, 在 Page\_Load 方法执行之前, DataGrid 中的数据将被再次绑定。这就是说, 在 Page\_Load 方法执行并创建 DataSet 之前, ChangeColor 方法将执行 7 次。在这 7 次执行期间, 无法访问数据, 因为数据还没有创建。因此, 第 26 行对 binSet 进行检查就是为了预防这种情况。通过在合适的位置使用一些 Response.Write 方法, 可以显示单击 Edit 和 Cancel 按钮后方法的执行顺序, 如图 9.17 和图 9.18 所示

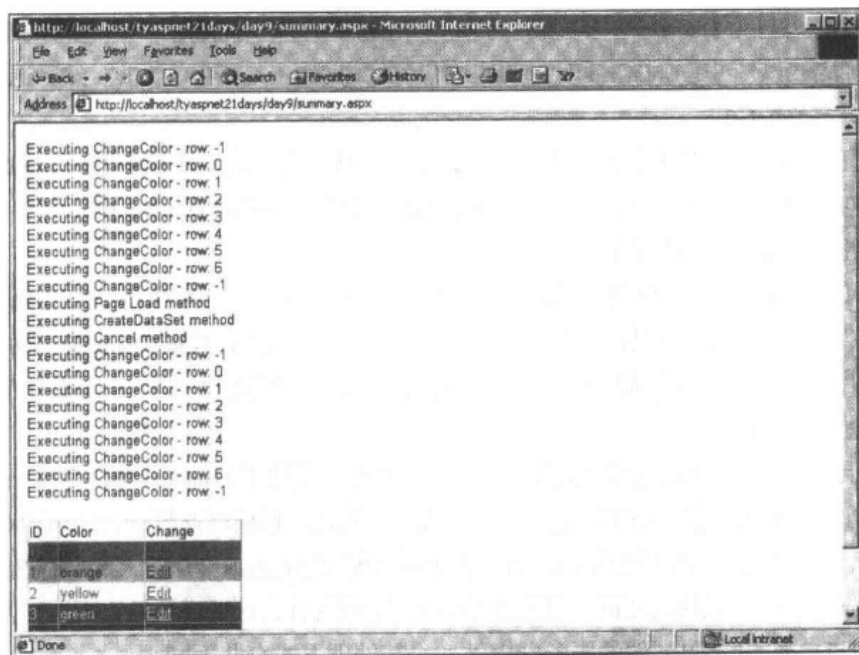


图 9.18 单击 Cancel、Update 和 Delete 按钮时, 数据在 Page\_Load 执行前被重新绑定

接着来看看第 27 行, 它确保操纵的是 DataGrid 中有效的条目——后面将进一步讨论。第 28 行使用 BackColor 属性设置 DataGrid 中各行的背景颜色。实际上, 您想做的是取得存储在用户所进行中的颜色。不幸的是, 事情并非看起来的那么简单, 因为 BackColor 属性只接受 System.Drawing.Color 对象, 而存储在数据源中的是字符串。因此, 必须进行合适的转换。不幸的是, 不能直接将字符串转换为颜色——必须用其他方式来实现。

**注意:** 对于页眉和页脚, e.Item.ItemIndex 将返回 -1; 对于其他行, 则返回相应的索引号。

Color 对象包含一个 FromName 方法, 它接受一个表示有效颜色的字符串, 例如“Red”或“Green”。FromName 将该字符串转换为一个 Color 对象。因此, 第 29 行将存储在 DataSet 中的颜色名称传递给 FromName, 以得到一个有效的 Color 对象, 这样便知道了用户所选行的颜色。第 28 行将当前行的 BackColor 属性设置为相应的颜色。

您在查看结果时发现, 文本好像融入到了背景中, 因此决定修改前景色。第 31-33 行返回前景色。您知道反色的效果将更好, 因此使用公式  $6 - \text{ItemIndex}$  来返回补色 (注意也可以使用  $e.\text{Item.Count} - e.\text{Item.ItemIndex}$ , 但知道 count 不会变化, 因此您使用硬编码的方式指定其值)。第 31 行设置前景色, 至此工作便完成了! 现在您拥有用户可操纵的、使用彩虹着色的 DataGrid (如图 9.12 所示)。用户可以修改数据源中的值, 从而修改相应行的颜色。

**注意：**如果指定的颜色不包含在Color对象中，例如“bluegree”、“violet”或“redde”，应用程序将出错。以后，您可能添加一个例程，核实用户输入的颜色是否包含在Color对象中，这可以使用Color对象的IsKnownColor方法来实现。

您可能注意 DataGrid 一些特别的地方，DataGrid 的最后一行不是本应该的紫色。

让我们来看看这个问题。ItemCreated 事件在 DataGrid 创建之前被激发。通过在 ChangeColor 方法执行过程中检查 DataGrid1.Items.Count 可验证这一点。DataGrid1.Items.Count 返回的第一个值为 0，这是 ItemCreated 事件第一次被激发时的值。因此，不能在 ItemCreated 事件执行期间设置该行的颜色，因为该行还没有被创建，否则将产生错误。

根据 e.Item.ItemIndex 来确定哪些行已被创建并不准确，特别是该例中<sup>1</sup>，e.Item.ItemIndex 两次都返回了-1。这也是清单 9.18 的第 27 行使用 if 语句的原因：您想避免给不存在的行设置颜色。由于该事件在条目被创建之前激发，所以必须在事件最后一次被激发前设置被创建的行的颜色，这就是在引用数值时使用 e.Item.ItemIndex - 1 的原因所在。

因此，无法为最后一行着色，因为当最后的 ItemCreated 事件激发时，是给倒数第二行着色。然而，没有任何损失，最后一行能被着色。可以采取一种机智的方式来操纵被创建的行并检查其属性，这将作为一个练习，留给您完成。

## 9.5 这不是 ASP

本章介绍了 ASP.NET 相对于传统 ASP 的两项重大改进：DataSet 和数据绑定。前者类似于以前的 RecordSet 对象，但功能更多。DataSet 可以表示完整的数据集，包括表之间的关系及层次式数据集。

数据绑定让您能够轻松地在页面中显示内容。在传统 ASP 中，必须手工遍历数据源，如下而代码段中的 RecordSet：

```
<%
Do While Not objRS.EOF
    Response.write "<B>" & objRS("FirstName") & " " & _
        objRS("LastName") & "</B><BR>" & _
        objRS("Address") & "<BR>" & objRS("City") & _
        "<BR>" & objRS("State") & "<BR>" & objRS("Zip") & _
        objRS("Phone") & "<P><HR><P>"
    objRS.MoveNext
Loop
%>
```

开发人员必须编写代码来处理其显示和 HTML。因此，代码和 UI 元素分散在页面中，这样很容易出错，并常常引起混乱（人们将这种问题称为“意大利面条式代码”）。

使用数据绑定和服务控件，再也不会存在上述问题。这些控件将自动遍历任何可遍历的对象（如 DataSet、数组或任何其他列表服务器控件）！另外，它们为显示数据提供了大量的功能。

使用数据绑定和列表服务器控件后，您一定会想，没有它们日子该怎样过！使用这些控件，可以轻松地将开发时间减半，同时它们提供的功能比传统 ASP 更多。

## 9.6 总 结

前一章介绍了数据库的基础知识以及如何在 ASP.NET Web 页面中使用数据库。本章大大地扩充了这一主题，介绍了如何在页面中添加诸如添加、编辑、排序及分页等功能。具体地说，介绍了数据绑定和三个新的服务器控件，这些控件提供了大量的功能，大大缩短了开发人员的开发时间。

首先，您学习了 ADO.NET 的新框架的一个有机组成部分：DataSet。您学习了如何使用 DataSet 来表示不同类型的数据，同时保持一致的界面。下一章将介绍整个 ADO.NET 工具集及其在新的分布式数据模式中的工作方式。

本章花了大量的篇幅介绍下面三个主要的数据绑定控件：Repeater、DataList 和 DataGrid 以及它们如何简化了数据的表示和操纵。这三个控件提供大量很复杂的功能，而开发人员只需做很少的工作。这三个控件提供了一些相似的功能，并具有一些相同的属性、方法及事件。学习一个控件的使用方法后，便可以将所学的知识用到其他两个控件中。学会使用这些控件将是朝着成为一名 ASP.NET 数据库专家迈出了一大步——您离这样的目标不远了！

至此，您对 ADO.NET 及其存取和修改数据的机制有了蜻蜓点水式的了解。下一章将深入介绍 ADO.NET 是如何在 ASP.NET 中运行的——这是一个很重要的话题。同时，将开始使用这些数据绑定服务器控件！

## 9.7 问与答

问：DataSet 可否保存 XML 数据？

答：完全可以。事实上，当 DataSet 在服务器和客户端之间传输时，它首先被转换为 XML。这使得它可以将数据传输到一些通常无法达到的地方，如穿过具有严格防范措施的防火墙，防火墙是设计用来阻止未经授权的数据的。因为 XML 是纯文本，因此防火墙将忽略它，并允许其通过。

问：清单 9.1 中的 MiniumCapacity 属性有何功能？

答：该属性设置表中的最小行数。设置该属性无法限制您向表中添加记录。该属性能够预留内存资源，这样在运行阶段访问这些记录时，速度将更快。下一章将介绍该属性以及另一个 ADO.NET 特有的属性。

## 9.8 作 业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 9.8.1 小测验

1. 要在 ASP.NET 页面中使用数据机制必须导入哪个名称空间？
2. 判断正误：DataSet 可以包含多个表。
3. DataSet 中的三个集合是什么，各有什么作用？
4. 数据绑定表达式的语法是什么样的？
5. DataList 控件是否支持分页功能？

6. 要启用排序和分页功能, 必须分别设置 DataGrid 的哪两个属性?

#### 9.8.2 练习

使用数据绑定对第 6 章的练习 2 中的会议安排程序进行改进。将 DayLabel 绑定到日历的 selectedDate, 并相应地格式化它。

# 第 10 章

## 与 ASP.NET 通信

仅两章，您便开始向数据库专家迈进了！在第 8 章中，您接触了数据库，并了解了如何在 ASP.NET 中应用数据库。在前一章，您深入学习了 DataSet 和 WEB 表单中的数据绑定控件。但到目前为止，您还没有深入学习 ADO.NET 固有的对象。

本章将介绍 ADO.NET、其框架以及它是如何与 ASP.NET 协同工作的。本章将涉及到大量的理论，但将列举大量的例子。阅读完本章后，您将能够在 ASP.NET 页面中访问任何类型的数据源，并使用它们来创建动态的 data-enabled 页面。

本章介绍以下内容：

- 如何修改 DataSet 中的数据；
- 如何操纵 DataRow 和 DataTable；
- 如何使用 OleDbCommandBuilder，将 DataSet 中数据的修改自动反映到数据源中；
- 如何使用 OleDbConnection 对象；
- 如何使用 OleDbCommand 对象；
- 如何使用 OleDbDataReader 对象；
- 如何使用 OleDbDataAdapter 对象。

### 10.1 ADO.NET 简介

ADO.NET 是下一代 ActiveX Data Objects (ADO)。它是一种数据存取模型，具有可扩展性、Web 无状态性，其内核使用的是 XML。ADO.NET 提供了到所有 OLE-DB 顺应数据源的接口，让您能够连接到、检索、操纵和更新这些数据源。无论是在远程环境、使用分布式应用程序，还是离线数据 (disconnected data) 时，都可以使用 ADO.NET。

就 ASP.NET 开发而言，ADO.NET 为在 ASP.NET 页面中存取任何类型的数据提供了框架。这让用户可以查看或修改存储在任何类型数据源中的信息，包括数据库、文本文件及 XML 数据源。您应该熟练掌握 ADO.NET，因为它对于动态应用程序的开发至关重要。掌握其复杂性可以避免开发过程中的许多麻烦。

#### 10.1.1 ADO.NET 和 ADO 的比较

尽管 Microsoft 宣称 ADO.NET 为下一代 ADO，同时它们确实有一些相同的对象，但是 ADO.NET

同 ADO 差别很大。ADO 是基于连接的，而 ADO.NET 依赖于简短的、基于 XML 的消息与数据源交互。这使得对于那些基于 Internet 的应用程序而言，ADO.NET 的效率要高得多。

从 ADO 到 ADO.NET 的一个本质变化是，后者采用 XML 来交换数据。XML 是一种基于文本的标记语言，类似于 HTML，在表示数据方面，提供了一种高效方式（将在本章后面和第 11 章介绍）。ADO.NET 同 XML 关系密切，并将其用于所有事务中，这使得 ADO.NET 在对数据源的访问、交互和永久化方面比 ADO 容易得多。同时，ADO.NET 的性能也高得多，这是因为 XML 数据和其他类型数据之间的转换很容易，不需要像传统 ADO 那样进行复杂的转换，这种转换将浪费大量的处理时间。

另一个重大变化是 ADO.NET 与数据库交互的方式。ADO 要求锁定数据库，并长时间地连接到数据库，而 ADO.NET 不需要。ADO.NET 使用离线数据集（使用 DataSet 对象），因此无需长时间地连接并锁定数据库。这使得 ADO.NET 具有更好的可扩展性，因为用户无需争夺数据库资源。

表 10.1 总结了从 ADO 到 ADO.NET 的主要变化。

表 10.1 从 ADO 到 ADO.NET 的主要变化

ADO	ADO.NET
表示数据的方式： RecordSet，表示单个表或查询结果	DataSet，可以包含任何数据源中的多个表
数据存取： 按 RecordSet 中的顺序存取记录	通过基于集合的层次，可以以任何顺序存取 DataSet 中的数据
多个表之间的关系： 需要使用 SQL JOIN 和 UNION 将多个表中的数据组合成一个 RecordSet	使用 DataRelation 对象，该对象可用在多相关的表之间导航
数据共享： 需要将数据转换为接受系统支持的数据类型，这降低了性能	使用 XML，所以不需要转换
编程方式： 使用 Connection 对象来将命令传输给数据源的底层结构	使用 XML 的强类型特性，无需使用数据结构，可通过名称引用任何东西
可扩展性： 数据库锁定和连接导致对数据资源的争夺	不需要锁定和长时间的连接，所以不存在数据资源的争夺问题
防火墙： 有问题，因为防火墙阻止很多类型的请求	没有问题，因为 XML 完全不受防火墙限制

### 10.1.2 ADO.NET 和 XML

XML 是一种非常有用的数据分发工具，它完全是基于文本的，这意味着人们很容易对其进行读写，并可避开 Internet 中的安全措施进行传输。

XML 通过层次方式表示字段及其数据来存储数据。例如，如果有一个名为 Users 的数据库，它包含 Name、UserID 和 BirthDate 三个字段，则其文本格式表示如下：



```
<Users>
<User>
  <Name />
  <UserID />
  <Birthdate />
</User>
</Users>
```

这种基本结构被称为 XML 模式 (schema), 实际情况要复杂些, 但这不在本书的讨论范围之内。然后, 您便可以使用这种模式来表示表中的所有数据:

```
<Users>
<User>
  <name>Chris Payne</name>
  <UserID>1</UserID>
  <birthdate>June 2</birthdate>
</User>
<User>
  <name>Eva Saravia</name>
  <UserID>2</UserID>
  <birthdate>July 15</birthdate>
</User>
<Users>
...
...
```

任何人都可以使用文本编辑器 (如记事本) 来阅读上述 XML, 而对应的数据库表只能通过特定的数据库应用程序或将其转化为另一种数据库应用程序进行阅读。在数据存储和传送方面, XML 是一种高效、独立于实现的方式, 这就是 XML 在 Web 中流行的原因。

因此, 数据库只有采用这种通信方式才合情合理。XML 使得每个人的工作都更为轻松。ADO.NET 在交换数据以及内部表示数据时, 都使用 XML。数据从数据库中被取出后, 立刻被表示为 XML, 然后被发往任何需要的地方。因为任何程序都能理解 XML (您在下一章将知道这一点), 因此这种方法确保了数据广泛的兼容性; 数据可以被发送到任何地方、任何系统, 并确保接收方能够理解它们。

ADO.NET 采用 XML, 这朝着将应用程序作为服务在 Internet 递送 (这是 .NET 框架的基本思想) 迈出了 一大步。这里只介绍了一些基本知识, 随着在接下来的几章中开发更多的分布式应用程序, 您将了解其优势。

### 10.1.3 ADO.NET 对象模型

ADO.NET 主要包括两部分: DataSet 和管理提供程序 (managed providers), 前者已经在上一章介绍过了。DataSet 用于表示在 ADO.NET 对象间传输的数据, 例如从数据源到 ASP.NET 页面的数据, 它是一种在数据源表示数据的机制。

**新术语:** 管理提供程序充当了 DataSet 和数据源之间的通信层, 为连接、存取、操纵、检索任

何 OLE-DB 顺应的数据源（如 Microsoft Access）中的信息提供了所有的机制。

Microsoft 在 ADO.NET 中提供了两种管理提供程序：SQL 管理提供程序和 OLE DB 管理提供程序。前者仅用于与 Microsoft SQL Server 交互，它为 SQL Server 和 DataSet 之间的通信提供了所有的方法；后者用于协调 DataSet 和 OLE DB 顺应数据源之间的通信。这两个管理提供程序在与数据源交互方面提供的基本功能相同，它们有什么区别呢？

SQL 管理提供程序使用一种叫做表格式数据流（tabular data stream）的协议与 SQL Server 通信。与 SQL Server 通信时，这种方式的效率非常高，同时它不依赖于 OLE DB、ADO 或 ODBC。表格式数据流协议完全由 CLR 管理，所以它能在第一部分介绍的所有特性中受益。这也是 Microsoft 推荐在 ADO.NET 和 ASP.NET 中使用 SQL Server 数据库的原因所在。

注意：ADO.NET 中的 SQL 管理提供程序只适用于 SQL Server 7.0 或更高的版本。如果使用的是以前的版本，请使用 OLE DB 管理提供程序。

而 OLE DB 管理提供程序可以同其他任何的数据源进行高效的通信——需要时，还可将其用于 SQL Server。

每种管理提供程序都包含下面三个组件：

- 连接、管理数据源以及与 DataSet 进行交互的接口。
- 用于快速、高效地存取数据的数据流（类似于 DataSet，但速度更快，功能更少）
- 用于连接数据库及执行数据库专用的低层命令的对象。

在本书余下的内容中，都将使用 OLE DB 管理提供程序，因为通过它可以存取您将使用的数据类型。两个管理提供程序的大部分语法都相似，所以转换到 SQL 管理提供程序不会太难。

提示：事实上，ADO 管理提供程序的所有对象都带前缀 OleDb。在大部分情况下，只需将该前缀替换为 SQL，并导入名称空间 System.Data.SQL，便可以从使用 OLE DB 管理提供程序转到使用 SQL 管理提供程序。

## 10.2 再谈 DataSet

虽然前一章介绍过 DataSet，但忽略了 DataSet 的一些概念和属性。

**新术语：**知道了什么是 DataSet 以及 DataSet 的作用后，重要的是牢记 DataSet 是完全独立于数据库的实体。二者之间没有任何联系，正因如此，DataSet 被称为是断开的。在断开方式下，每一个用户都将得到数据的一个备份，并可以对此备份进行任何操作。对数据的任何更改必须使用本章后面学习的方法准确地返回送到数据源中。

表 10.2 列出了 DataSet 的属性。

表 10.2 DataSet 属性

属 性	描 述
CaseSensitive	指示比较 DataTable 中的字符串时是否区分大小写
DataSetName	获得或设置当前 DataSet 的名称
DefaultView	获得 DataSet 数据的自定义视图，以便搜索、过滤、导航数据等
EnforceConstraints	指示数据更新时，是否考虑数据库中已有的约束
ExtendedProperties	取得由用户自定义信息组成的集合

续表

属 性	描 述
HasErrors	指示 DataSet 中的表或记录是否有错
Relations	取得 DataSet 中表关系的集合
Tables	取得 DataSet 的表集合
XML	取得或设置 DataSet 的 XML 数据或模式
XMLData	取得或设置 DataSet 的 XML 数据
XMLSchema	取得或设置 DataSet 的 XML 模式

表 10.3

DataSet 方法

方 法	描 述
AcceptChanges	提交 DataSet 被装载或上一次调用 Acceptchanges 后, 对 DataSet 做的所有修改
Clear	删除 DataSet 的所有表中的记录, 但不删除实际的数据库内容
Clone	复制 DataSet 的结构, 包括 DataTable、关系和约束
Copy	复制 DataSet 的结构及数据
GetChanges	返回 DataSet 的副本, 其中包含上次装载后对底层数据所做的所有修改
GetChildRelations	取得指定表的子关系集合
GetParentRelations	取得指定表的父关系集合
HasChanges	指示 DataSet 是否被更改
Merge	将 DataSet 与另一个 DataSet 合并
ReadXML	将 XML 模式和数据读取到 DataSet 中
ReadXMLData	将 XML 数据读取到 DataSet 中
ReadXMLSchema	将 XML 模式读取到 DataSet 中
RejectChanges	撤销对 DataSet 所做的任何修改
ResetRelations	将 Relations 属性设为默认值
ResetTables	将 Tables 属性设为默认值
ShouldPersistRelations	指示 Relation 属性是否为永久性的
ShouldPersistTable	指示 Table 属性是否为永久性的
WriteXML	将表示 DataSet 的 XML 写入到 XML 文件中, 包括数据和模式
WriteXMLData	将表示 DataSet 的 XML 写入到 XML 文件中, 只包括数据
WriteXMLSchema	将表示 DataSet 的 XML 写入到 XML 文件中, 只包括模式

注意：表10.3并没有列出DataSet对象的全部属性和方法，其中不包含一些通用项。更详细的信息，请参阅.Net框架SDK文档或附录D“ADO.NET控件：属性和方法”

正如您看到的，DataSet 提供了很多以前没有介绍过的特性，如读写 XML 数据，不要害怕去使用这些属性。DataTable 和 DataRow 也提供了很多与 DataSet 对象相同的属性和方法，所以这里没有列出来。图 10.1 说明了 DataSet 对象模型。

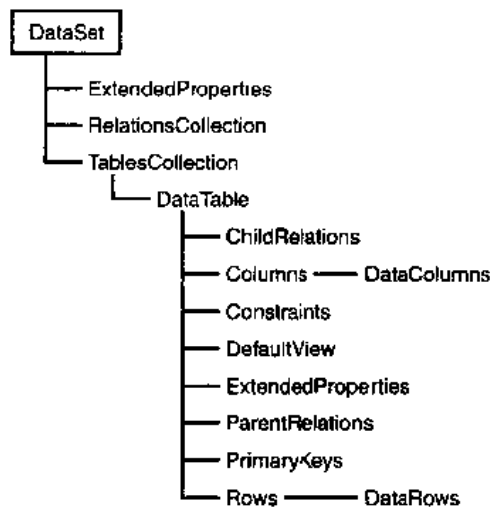


图 10.1 DataSet 对象模型

### 10.2.1 修改 DataRow 中的数据

熟悉 DataSet 各部分后，我们来讨论如何修改其中的数据。DataSet 存储数据的方式与数据库类似，它包含表、字段和记录。您通常使用一次修改多条记录的 SQL 语句来操纵 DataSet 中的数据，但有时需要直接控制每条记录。DataSet 对象表示 DataTable 中的一条记录，上一章介绍过，可以直接编辑每个 DataRow 中的内容。

您还需要了解 DataRow 和 DataTable 的其他一些特性。第一个是 RowState 属性，该属性指出了当前记录的状态。可能的值包括 Detached、Unchanged、New、Deleted 和 Modified。Detached 指记录已被创建，但还不属于 DataSet 中的任何 RowsCollection。后四个属性的含义是不言而喻的。

作为 RowState 的一部分，DataTable 保留每条记录的三个版本：原始的、当前的和提议的。原始版本指的是记录最初被添加到 DataTable 中的情形，通常与数据源中相同。当前版本指记录被修改后的情况。提议版本在一种特殊情况下存在——对记录调用了 BeginEdit 方法时。

BeginEdit 方法在不使用有效性验证规则的情况下，对记录做多次修改。例如，如果需要将多条记录的值增加到某一指定值，可以将这些记录切换到编辑模式，然后对值进行操纵。当调用 EndEdit 或 AcceptChanges 方法时，记录将退出编辑模式，有效性验证规则被应用。您还可以使用编辑模式来撤销任何提议的修改。图 10.2 说明了修改记录的过程。

当调用 Fill 时，数据源的原始值将被填充到 DataSet 中。对该值做出任何修改后，修改后的值将成为当前值。现在可以将其恢复到原始值、使用当前值更新数据源或进入编辑模式。在编辑模式下，可以接受修改并更新数据源，或撤销所做的修改并返回到原始值或当前值。事实上，在编辑模式下可以返回到任何一种值并更新数据库。这三种值分别可以使用以下三个属性进行访问：

DataRowVersion.Original、DataRowVersion.Current 和 DataRowVersion.Proposed。

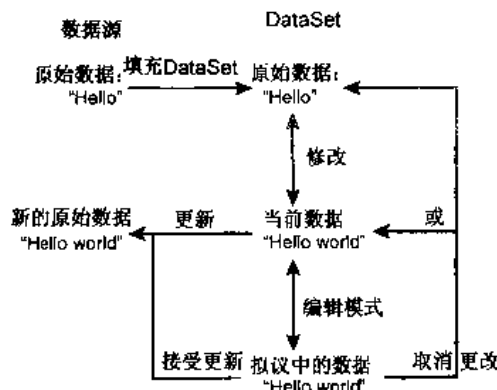


图 10.2 修改记录

DataRow 中的数据被修改后，由于某种原因可能出错。对于每种错误，DataRow 的 RowError 属性都会存储一个字符串，您也可以以手工方式在该属性中插入错误。通过调用 GetErrors 方法，可以一次性检索所有的错误，该方法返回一个 DataRow 数组。发生错误后，将不会对数据源进行合并和更新，所以首先要排除错误。如果觉得不好理解，也不用担心，开发了几个例子后，便会更明白这一点。

DataRow 提供了两种很相似的方法：Delete 和 Remove，但这两个方法之间有一个很重要的区别。Delete 方法完全删除记录及其数据。执行 Delete 方法后，其中数据再也不能访问。而 Remove 方法只是删除 DataTable 中的记录，这样代码将无法访问该记录，但数据源并没有更改，数据仍然存在，您只是看不到而已。如果不想使用 DataTable 中的所有记录，则 Remove 将很有帮助。

最后，RejectChanges 撤销记录被装载后或上一次调用 AcceptChanges 方法后对数据所做的任何修改。例如，下面的代码片段将数据装载到 DataSet 中，修改了第一条记录中的值，然后又撤销所做的修改。

```

dim objConn as new OleDbConnection
    ('Provider=Microsoft.Jet.OLEDB.4.0;' & _
    "Data Source=C:\ASPNET\data\banking.mdb')

dim objCmd as new OleDbDataAdapter _
    ('select * from tblUsers', objConn)

dim ds as DataSet = new DataSet()
objCmd.Fill(ds, "tblUsers")

ds.Tables("tblUsers").Rows(0)("FirstName") = "Chris"

'do some other stuff

ds.Tables("tblUsers").Rows(0).RejectChanges
  
```

### 10.2.2 查看 DataTable 中的数据

DataTable 有一个 Select 方法，让您能够对表中的数据进行过滤和排序。该方法返回一个 DataRows 数组，其调用语法如下：

```

Tablename.Select(filter expression, sort order, _
  
```

```
DataRow.NewState()
```

例如:

```
dim ds as new DataSet("MyDataSet")
dim dTable as new DataTable("MyTable")
' fill data set and datatable here
Dim MyRows() as DataRow = (ds.Tables("MyTable").Select( _
    'Nothing, 'UserName', DataViewRowState. _
    CurrentRows)
```

上述代码片段返回 `DataRow` 数组, 其中包含所有被修改过的记录, 这些记录按字段 `UserName` 进行排序。对于不需要的参数, 可以指定 `Nothing`; 因此, 可以返回记录的任何版本或所有版本, 并对其进行过滤或排序。我们来看另一个例子, 如清单 10.1 所示。

#### 清单 10.1 使用 `Select` 方法检索记录

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     sub Page_Load(obj as object, e as EventArgs)
7         dim objConn as new OleDbConnection _
8             ("Provider=Microsoft.Jet.OLEDB.4.0; & _
9             'Data Source=H:\ASPNET\data\banking.mdb")
10
11         dim objConn as new OleDbConnection _
12             ("Provider=Microsoft.Jet.OLEDB.4.0;DSN=21 JavsBanking")
13
14         dim objCmd as new OleDbDataAdapter _
15             ("select * from tblUsers", objConn)
16
17         dim dTable as DataTable = ds.Tables("tblUsers")
18         Dim CurrRows() as DataRow = dTable.Select(Nothing, _
19             Nothing, DataViewRowState.CurrentRows)
20         Dim I, J as integer
21         Dim strOutput as string
22
23         For I = 0 to CurrRows.Length - 1
24             For J = 0 to dTable.Columns.Count - 1
25                 strOutput = strOutput & dTable.Columns(J). _
26                     ColumnName & " " & CurrRows(I(J)).ToString & _
27                 '<br>'
```

```

28     next
29     next
30
31     Response.write(strOutput)
32 end sub
33 </script>
34
35 <html><body>
36
37 </body></html>

```

**分析：**该清单检索 DataSet 中所有的记录，并在浏览器中显示其字段及数据。在 Page\_Load 方法中，您创建了一个 OleDbConnection 和一个 OleDbDataAdapter 对象（第 7-12 行）。这些内容在上一章介绍过了，您应该熟悉。然后，您创建了一个 DataSet 对象，并使用 Fill 方法在其中填充了数据（第 14-15 行）。然后，在第 17 行，您取回了 DataSet 中唯一一个表，并将其保存在变量 dTable 中，以便以后进行访问。

第 18 行使用 Select 方法取得 DataTable 中所有被修改过的记录——当前记录，并将其存在一个数组中。第 23 行中的 for 循环用于遍历数组中的记录，第 24 行的嵌套 For 循环遍历记录的每个字段，各字段的名称及其值都存储在一个字符串中，然后在第 31 行输出该字符串。该清单的输出结果如图 10.3 所示。

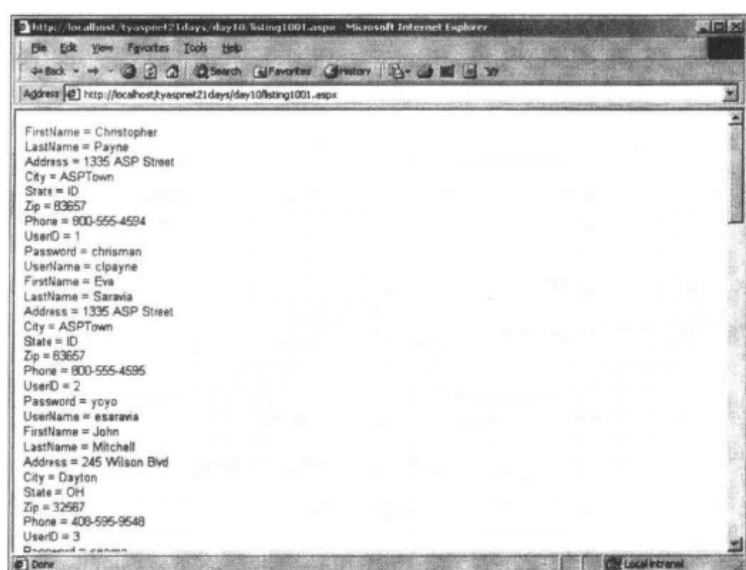


图 10.3 For 循环和 Select 方法可以遍历 DataSet 中的所有记录

另一种对数据进行排序和过滤的方法是使用 DataViews。DataViews 是一个对象，用于表示 DataTable，但与 DataTable 不同，它可以绑定到 Web 控件。

可以为一个 DataTable 创建多个 DataViews，这样，ASP.NET 页面可以包含两个被绑定到同一个 DataTable 的不同控件，它们显示不同的数据。例如，下面的代码片段演示了如何创建 DataView 并设置其属性：

```
dim MyView as new DataView(dTable)
```

```
MyView.RowStateFilter = DataViewRowState.ModifiedOriginal
MyView.Sort = "UserID ASC"
MyView.RowFilter = "City = ASPTown"
```

第 1 行使用 `dTable` 变量创建了一个新的 `DataView` (`dTable` 是以前创建的一个 `DataTable`, 其中填充了数据)。第 2 行设置 `DataView`, 以过滤掉除原始版本之外的所有版本。第 3 行指定了排序方式, 最后一行为返回记录指定了条件。`DataView` 包含的很多属性与 `Select` 方法用于取回记录的属性相同。要修改数据源时, 掌握这些属性将给您很大的帮助。

### 10.2.3 并发

由于每个用户都可以拥有自己的 `DataSet` 视图, 所以用户可以随心所欲地操纵 `DataSet`, 并在准备好后更新数据源。那么, 当两个或更多用户试图同时更新同一个表中的相同数据时, 将发生什么事情? 并发 (currency) 系统可以监视这种情况, 以保证不会出现任何问题。并发有两种: 保守式 (Pessimistic) 和开放式 (optimistic)。

每当用户以保守式并发方式访问或修改数据时, 数据将被锁定, 以防止其他用户修改该数据而影响第一个用户。当第一个用户完成操作后, 第二个用户可以重新尝试。

而开放式并发不锁定数据, 而是对记录进行监视, 以确定原始数据是否被修改过, 然后将修改应用到记录。假设有两个用户都取得了名为 `U.S.S.Enterprise` 的船舶的数据, 并开始操纵这些数据。第一个用户将船名改为 `U.S.S.Enterprise A`。过后, 如果第二个用户也试图修改船名, 则其修改不会生效, 因为得到的数据副本不再有效。他必须取得数据的当前版本, 才能再次进行修改。用数据库的术语说, 如果两个用户取得相同的数据, 则只有第一个用户的修改有效。第二个用户的修改无效, 因为数据源发生了变化, 其修改企图将以失败告终。

ADO.NET 可以使用任何一种并发方式。幸运的是, ADO.NET 提供了内置的机制来透明地处理这两种机制。尽管如此, 也有必要了解其工作原理, 以防遇到这种问题。

## 10.3 数据库和 ADO.NET 的交互

第 8 章的“数据库简介”一节中介绍过, 在 ASP.NET 页面中与数据交互分 5 步:

1. 创建一个数据库连接对象;
2. 打开数据库连接;
3. 使用所需的数据填充 `DataSet`;
4. 创建 `DataView` 来显示数据;
5. 将服务器控件绑定到 `DataView`。

前一章已详细介绍过第 5 步, 所以本章重点介绍前 4 步。其中的几个步骤实现的方法有多种 (使用不同的对象), 所以将依次介绍这些方法。但首先, 需要介绍连接到数据库所需的信息。

### 10.3.1 连接信息

要在 ASP.NET 页面中使用 ADO.NET 与数据库通信, 必须提供关于要访问的数据库的特定信息。这些重要的信息包括数据库的位置、数据库的类型 (如 MS Access、SQL Server 或 Oracle)、数据库的版本, 等等。这些信息是通过手工创建的连接字符串提供给 ADO.NET 的 (不要担心, 实际情况没有看起来的那么可怕)。

对于提供数据库连接信息, 最简单的方法是创建一个系统数据源名称 (System Data Source Name,



DSN) 文件。您应该有这样一个文件, 它提供了关于操作系统安装的一些数据源的默认信息。您只需在其中添加一些信息即可。幸运的是, 该过程很简单。现在我们来添加第 8 章创建的用户数据库的信息。

1. 如果用户数据库已经打开, 则关闭它。
2. 在 Windows 2000 中, 选择“开始”/“设置”/“控制面板”/“管理工具”/“数据源 (ODBC)”。
3. 单击标签“System DSN”, 对话框将与图 10.4 类似。该选项卡让您能够创建、编辑和删除已有的 ODBC 数据源。

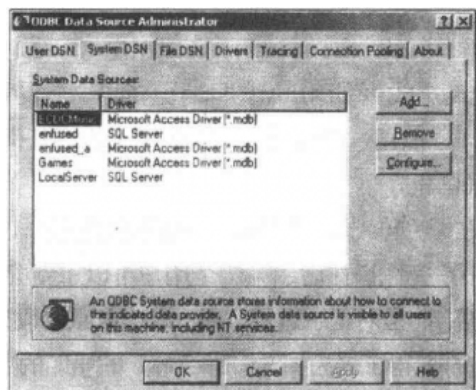


图 10.4 系统 DSN 信息

4. 单击“Add”按钮, 选择 Microsoft Access Driver (\*.mdb), 并单击“Finish”按钮。
5. 将 DSN 命名为 21DayBanking, 并输入描述信息。
6. 单击 Select 按钮, 找到第 8 章创建的 Access 数据库。单击 OK, 以选择该数据库。现在得到的结果类似于图 10.5。单击两次 OK 按钮, 任务便完成了。

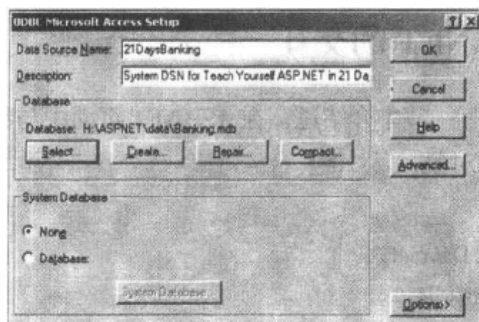


图 10.5 完成后的系统 DSN

数据库的系统 DSN 为 ADO.NET 提供了找到给数据库所需的所有信息。现在, 在 ASP.NET 页面中与 ADO.NET 交互时, 可以使用下面的连接字符串:

```
"DSN=21DaysBanking"
```

如果不想创建系统 DSN, 可以使用 DSN-less 连接, 这需要在连接字符串中指定所有必须的信息。以前面的数据库为例, 连接字符串将与下面类似:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
C:\ASPNET\data\banking.mdb"
```

对 SQL 数据库而言，连接字符串与下面类似：

```
"Provider=SQLNCLI;Initial Catalog=Northwind;  
Data Source=MyServer;User ID=sa;"
```

该字符串告诉 ADO.NET：应使用哪个提供程序以及数据库的位置。另外，还可能指定大量其他的参数，如 UID 和 PWD，它们分别是连接数据库时使用的用户名和密码。这里指定的连接字符串最普通，还没有涉及到其他内容。

除美观方面的考虑（连接字符串更短，代码更美观）外，使用系统 DSN 还有很多原因。其中最主要的原因是，如果使用 DSN-less 连接，则每次连接到数据库时，都必须验证连接信息。而使用 DSN，只在创建 DSN 创建时验证一次连接信息，从而可以大大提高性能。

### 10.3.2 OleDbConnection 对象

知道如何使用连接字符串来建立连接后，我们来打开数据库。这正是 System.Data.OleDb.OleDbConnection 对象应用之处。清单 10.2 是一个例子。

**清单 10.2 使用 OleDbConnection 对象来打开数据库**

```
1: Dim strConnectionString as string = _  
2:     "Provider=Microsoft.Jet.OLEDB.4.0;" & _  
3:     "Data Source=C:\ASPNET\data\banking.mdb"  
4: Dim Conn as New OleDbConnection( _  
5:     strConnectionString)  
6: Conn.Open()  
7: ...  
8: Conn.Close()
```

您现在打开了到数据库的连接。第 1 行声明了连接字符串，第 4 行的 OleDbConnection 对象使用该字符串来连接到数据库。第 6 行使用 Open 方法打开到数据库的连接，第 7 行的 Close 方法关闭该连接。当不再使用连接时一定要关闭它。

大多数情况下，使用 OleDbConnection 对象完成的所有工作便是打开和关闭数据库连接。第 12 章将介绍更多的技巧。

### 10.3.3 OleDbCommand 对象

连接到数据库后，便可以使用命令来操纵数据库，如填充 DataSet 或更新记录。图 10.6 是 OleDbCommand 对象模型的一部分。

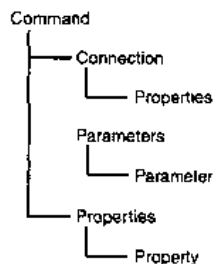


图 10.6 OleDbCommand 对象模型

数据库命令采用的是 SQL 语句, 因此您应该熟悉。所有要做的就是将给 OleDbcommand 对象指定一条 SQL 语句, 如清单 10.3 所示。

**清单 10.3 创建并初始化 OleDbcommand 对象**

```
1: 'set up SQL statement
2: Dim strSQL as string = "SELECT * FROM tblUsers"
3:
4: 'create object and set properties
5: Dim objCmd as New OleDbCommand()
6: objCmd.Connection = Conn
7: objCmd.CommandText = strSQL
8:
9: 'or
10: Dim objCmd as New OleDbCommand(strSQL, Conn)
11:
12: 'or
13: Dim objCmd as New OleDbCommand(strSQL, _
14:   strConnectionString)
```

**分析:** OleDbcommand 对象提供了很多通过指定不同的参数进行初始化的方法。OleDbcommand 接受的参数可以是 SQL 语句和 OleDbConnection 对象 (第 10 行) 或可用于创建 OleDbConnection 对象的连接字符串 (第 13 行)。然而, 只指定命令并不能完成很多的工作, 必须使用 Execute 方法之一来执行命令。至于使用哪个方法取决于您打算如何处理返回的数据。例如, 如果用来填充 OleDbDataReader (关于该对象的信息, 请参见下一节), 则使用:

```
'create a DataReader
dim objReader as OleDbDataReader
objReader = objCmd.ExecuteReader
```

如果执行不返回任何数据的查询, 则使用:

```
objCmd.ExecuteNonQuery
```

在本书后面, 您将接触到更多的 Execute 方法。

#### 10.3.4 OleDbDataReader 对象

OleDbDataReader 是一个轻量级对象, 只能用来访问数据源, 它实际上是一个流式 DataSet。既然有 DataSet, 为什么还要使用 OleDbDataReader 呢?

检索数据库的数据时, DataSet 取得所有的信息, 并将其保存在内存中, 直到您指出不要这样做。使用这种离线数据源, 可以完成一些简洁的工作。例如, 可以修改数据, 而无需担心用户将数据搞糟, 也可以将数据转换为其他格式。然而, 如果需要从数据库返回大量的数据, 将受到内存的限制, 因为整个 DataSet 都被保存在内存中。如果同时有成千上万的用户访问数据库 (每个用户都有自己的 DataSet), 将出现大问题 (注意, 这是一种极端情况, 但是有助于理解使用小型对象的必要性)。

OleDbDataReader 每次只将一条记录保存到内存中, 该对象按要求依次从数据源取得数据, 这样避免了使用大量内存, 从而提高了性能。不幸的是, 由于数据被依次取回, 所以 OleDbDataReader

提供的功能没有 DataSet 多。OleDbDataReader 是只读的，并且不能查看前面的记录。

填充 OleDbDataReader 对象后，很容易遍历记录，只要调用 Read 方法即可。清单 10.4 是一个例子。

**清单 10.4 遍历 OleDbDataReader 对象中的记录**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6   sub Page_Load(obj as object, e as eventargs)
7     dim objConn as new OleDbConnection _
7       'Provider=Microsoft.Jet.OLEDB.4.0;' & _
9       'Data Source=H:\ASPNET\data\banking.mdb')
10
11    dim objCmd as new OleDbCommand _
12      ('select * from tblUsers', objConn)
13
14    dim objReader as OleDbDataReader
15
16    objConn.Open
17    objReader = objCmd.ExecuteReader
18
19    while objReader.Read
20      Response.write(objReader.GetString(0) & "<br>")
21    end while
22    objConn.Close
23  end sub
24 </script>
25
26 <html><body>
27
28 </body></html>

```

**分析：**第 7-12 行很眼熟，这些代码创建一个 OleDbConnection 和一个 OleDbCommand 对象，并执行一条 SQL 语句。您将很快熟练地掌握这几个步骤。第 14 行创建了一个新的 OleDbDataReader 对象。OleDbDataAdapter 对象的 Fill 方法自动为您打开和关闭数据库连接，但使用 OleDbDataReader 时，必须手工完成这些工作，如第 16 和 22 行所示。第 17 行执行 SQL 语句，并将返回的记录依次传给 OleDbDataReader 对象。第 19-21 行遍历 OleDbDataReader 中的所有记录。

Read 方法自动转向下一条记录，到最后一条记录。第 20 行使用 GetString 方法取得每条记录的第一个字段，该方法返回一个字符串（后面将讨论该方法）。然后使用 Response.Write 显示该字符串。

输出结果如图 10.7 所示。

通过检查 `HasMoreRows` 属性，可以确定是否还有其他的记录。如果还有其他记录，则该属性为 `true`；否则为 `false`。

`OleDbDataReader` 还有一些 `get` 方法（`GetByte`、`GetInt32`、`GetString` 等），这些方法将字段中的数据作为本机类型返回。使用这些方法可以直接使用来自 `OleDbDataReader` 中的数据，而不用转换数据类型。

**警告：**不再使用时，一定要关闭 `OleDbDataReader`，这很重要。关闭 `OleDbDbConnection` 可以关闭 `OleDbDataReader`。

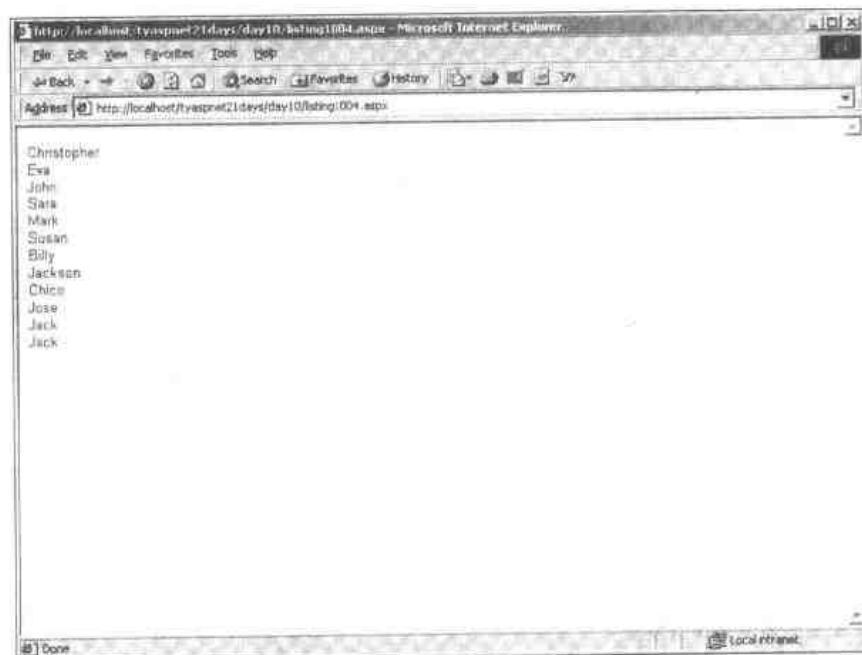


图 10.7 Read 方法可用来遍历 `OleDbDataReader` 中的记录

应 该	不 应 该
当性能至关重要，同时只需要显示数据库信息时，应该使用 <code>OleDbDataReader</code>	在显示数据之前或之后需要操纵或修改数据时，不要使用 <code>OleDbDataReader</code>

### 10.3.5 Update、Insert 和 Delete

因为 `OleDbDataReader` 是只读的，因此不能用于修改数据。所以，要修改数据，必须在 `OleDbCommand` 对象中使用合适的 SQL 语句：Update、Insert 和 Delete。

然而，这些语句返回的记录与 Select 语句不同。因此，应该使用 `ExecuteNonQuery` 方法，该方法返回一个整数，指出语句影响的记录数。清单 10.5 是一个例子。

#### 清单 10.5 使用 `OleDbCommand` 对象执行非 Select 语句

```
1:  dime i as integer
2:
3:  'set up SQL statement
```

```

4: Dim strSQL as string = "DELETE FROM tblUsers' & _
5:   "WHERE UserID = 5"
6: 'create object and set properties
7: Dim objCmd as New OleDbCommand(strSQL, Conn)
8:
9: I = objCmd.ExecuteNonQuery()

```

由于 UserID 是标识字段，所以 SQL 语句只影响 0 或 1 条记录，这取决于是否有一条记录的 ID 与此相同。您可以检查 I，来确定 SQL 语句是否起作用。Update 和 Delete 语句的作用类似（关于 SQL 语句的详细讨论见第 8 章）。

### 10.3.6 OleDbDataAdapter 对象

OleDbCommand 处理的是 OleDbDataReader，而 OleDbDataAdapter 处理的是 DataSet。OleDbDataAdapter 的主要功能是将数据源的数据写入到 DataSet 中以及将 DataSet 中的数据放回数据源中。图 10.8 简要描述了该对象的模型。其中的四个命令方法用于删除、插入、选择和更新 DataSet 中的数据。TableMapping 集合定义了数据库中的表和字段如何被映射到 DataSet 中。

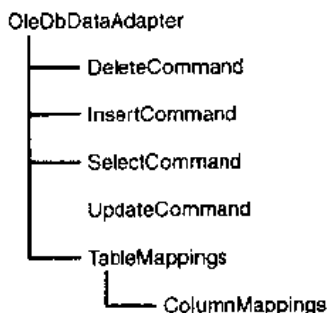


图 10.8 OleDbDataAdapter 对象模型

OleDbDataAdapter 对象的功能十分强大。尽管其主要用途是读取数据，但它还可以使用已有的数据创建一个全新的表或生成 XML。后面将介绍一些更高级的功能，但首先介绍一些基本知识。

OleDbDataAdapter 对象的创建过程类似于 ADOCommand 对象，如清单 10.6 所示。

#### 清单 10.6 创建 OleDbDataAdapter 对象

```

1: 'set up SQL statement
2: Dim strSQL as string = "SELECT * FROM tblUsers"
3:
4: 'create object and set properties
5: Dim objCmd as New OleDbDataAdapter()
6: objCmd.SelectCommand.Connection = Conn
7: objCmd.SelectCommand.CommandText = strSQL
8:
9: 'or
10: Dim objCmd as New OleDbDataAdapter _
11:   (strSQL, Conn)

```

```

12:
13: 'or
14: 'Dim objCmd as New OleDbDataAdapter( _
15:     strSQL, strConnectionString)

```

分析: 如图 10.8 所示, OleDbDataAdapter 有四个命令方法, 其中每个方法实际上是一个 OleDbCommand 对象, 有自己的 Connection 和 CommandText 属性, 如第 6-7 行所示。在初始化 OleDbDataAdapter 对象 (第 10 和第 14 行) 期间, 提供 SQL 语句时, 设置的命令对象方法是 SelectCommand 方法。如果想指定其他命令方法 (如 Insert), 必须手工进行。

注意: 使用命令方法 Update、Insert 和 Delete 时, 实际上修改的是底层的数据源, 而不是 DataSet, 后者已同数据源断开。只有在调用 OleDbDataAdapter.Update 方法后, 对数据的更改才会反映到数据源中。

#### 1. 填充 DataSet

我们来看一个填充 DataSets 的例子, 然后再使用各种方法。如清单 10.7 所示。

#### 清单 10.7 填充 DataSet

```

1:  dim strConectionString as String = _
2:  "Provider=Microsoft.Jet.OLEDB.4.0;" & _
3:  "Data Source=C:\ASPNET\data\banking.mdb"
4:  Dim ds as DataSet = New DataSet("myDataSet")
5:  Dim strSQL as String = "SELECT * FROM tblUsers"
6:
7:  Dim objCmd as new OleDbDataAdapter(strSQL, _
8:  strConnectionString)
9:
10:  objCmd.Fill(ds, "Users")

```

分析: 第 4 行创建了一个名为 MyDataSet 的空 DataSet, 第 7 行创建了一个新的 OleDbDataAdapter 对象, 该对象的 Select 命令被初始化为第 5 行创建的字符串。清单 10.7 得到的结果是一个 DataSet, 该 DataSet 包含一个名叫 Users 的 DataTable, 而后者包含 tblUsers 表中的所有记录。

这怎么可能呢? 因为您只是创建了一个空的 DataSet。OleDbDataAdapter 对象的 Fill 方法取得数据源的所有模式信息 (表、字段、主关键字等, 此时 DataSet 中还没有这些信息) 并自动在 DataSet 中创建这些信息。因此, OleDbDataAdapter 使用数据源的所有字段填充 DataTable Users。与此类似, 如果创建的 DataTable 包含一些字段, 则 FillDataSet 方法将创建其他的字段。如果所有的字段都已创建, 则 FillDataSet 方法将只给该表填充数据。OleDbDataAdapter 的这项功能十分强大, 下一节您将看到, 该功能有益无害。

#### 2. 更新数据源

上一章介绍了如何操纵 DataSet 中的数据, 方法是通过 DataSet 的集合来访问其字段和值。然而, 以这种方式修改数据后, 情况将如何呢?

修改数据后, 可以使用 OleDbDataAdapter 的辅助命令将修改返回到数据源。这些命令只关心被修改过的数据。例如, 无法只指定 Insert 语句, 而期望数据源中出现一条新记录。Insert 语句必须引用您在 DataSet 中创建的一条新记录。清单 10.8 是一个设置 OleDbDataAdapter 的 UpdateCommand 属

性的例子。

**清单 10.8 使用 OleDbDataAdapter 对象操纵 DataSet**

```

1:  sub Page_Load(obj as Object,e as EventArgs)
2:      create connection
3:      dim Conn as new OleDbConnection( _
4:          "Provider=Microsoft.Jet.OLEDB.4.0;" & _
5:          "Data Source=C:\ASPNET\data\banking.mdb")
6:
7:      'create DataSet and OleDbDataAdapter
8:      Dim ds as new DataSet("MyDataSet")
9:      Dim objCmd as new OleDbDataAdapter("SELECT * FROM " & _
10:         'tblUsers WHERE UserID < 10', Conn)
11:
12:      'fill DataSet
13:      objCmd.Fill(ds, "tblUsers")
14:
15:      'change some data
16:      ds.Tables("tblUsers").Rows(2)(3) = "ASPville"
17:
18:      dim dr as DataRow = ds.Tables("tblUsers").NewRow()
19:      dr(0) = "Greg"
20:      dr(1) = "Smith"
21:      dr(2) = "434 Maple Apt B"
22:      dr(3) = "Minneapolis"
23:      dr(4) = "MN"
24:      dr(5) = "12588"
25:      dr(6) = "5189876259"
26:      ds.Tables("tblUsers").Rows.Add(dr)
27:
28:      'provide SQL command and active connection
29:      objCmd.UpdateCommand = new OleDbCommand
30:      objCmd.UpdateCommand.CommandText = "Update tblUsers " & _
31:         'SET City='ASPville' WHERE UserID=3"
32:      objCmd.UpdateCommand.Connection = Conn
33:
34:      'provide another SQL command and active connection
35:      objCmd.InsertCommand = new OleDbCommand
36:      objCmd.InsertCommand.CommandText = "Insert INTO " & _
37:         'tblUsers (FirstName, LastName, Address, City, ' & _

```



```

38:         "State, ZIP, Phone)" VALUES ('Greg', 'Smith', ' & _
39:         ' 434 Maple Apt. B', 'Minneapolis', 'MN', 12588', ' & _
40:         ' 5189876259' "
41:     objCmd.InsertCommand.Connection = Conn
42:
43: end sub

```

在该清单中，您首先创建了一个 `OleDbConnection` 和一个 `OleDbDataAdapter` 对象（第 3–10 行），并在第 13 行使用 SQL `Select` 语句填充了 `DataSet`，这类似于使用 `OleDbCommand` 的情况。在第 16–26 行，您对 `DataSet` 做了一些修改：第 16 行编辑了一个值，第 18–26 行添加一条全新的记录。在第 29 行，您创建了一个新的 `OleDbCommand` 对象，该对象被用于 `OleDbDataAdapter` 的 `Update` 命令。第 30 行指定一条在更新数据源时使用的 `Update` 语句。与此类似，第 36 行指定了一条 `Insert` 语句。请注意 `Insert` 和 `Update` 语句是如何只引用修改过的记录中的数据的。实际上这些语句现在还不起作用，调用 `Update` 方法时，才对数据源执行这些方法。

**注意：**这些命令实际上并不会修改任何数据，它们只是提供命令，以告知 ADO.NET 如何处理修改过的数据。例如，如果第 16–26 行没有做任何修改，则不管指定何种 SQL 语句，命令方法都不会执行任何操作。

`OleDbDataAdapter` 的 `Update` 方法用来将 `DataSet` 中的修改返回到数据源。它使用您指定的 `Insert`、`Update` 和 `Delete` 命令来修改数据源。现在我们来修改清单 10.8，以包含 `Update` 命令。将下面的代码添加到第 41 行的后面：

```
objCmd.Update(ds, "tblUsers")
```

该方法的参数是一个 `DataSet` 对象变量和一个提供数据的表，前者包含修改后的数据。可以省略表参数，但必须提供表映射。

这看起来实在是一种痛苦：必须手工修改数据，然后创建一个 SQL 语句，指出您修改过数据。幸运的是，ADO.NET 可以帮忙。如果没有指定这些命令，并且 `DataSet` 中的记录已被修改，则 `OleDbDataAdapter` 对象可以使用 `OleDbCommandBuilder` 对象自动为您创建这些命令。例如，我们来看一下下面的代码：

```

dim ds as new DataSet("MyDataSet")
dim objCmd as new OleDbDataAdapter _
    ("select * from tblUsers", Conn)
dim objAutoGen as New OleDbCommandBuilder(objCmd)
...
'retrieve and
'modify some data
...
objCmd.Update(ds, "tblUsers")

```

前 3 行代码同以前的一样。但第 4 行创建了一个新的 `OleDbCommandBuilder` 对象，其参数为一个 `OleDbDataAdapter` 对象。`Update` 方法被调用后，该 `OleDbDataAdapter` 对象将检查 `DataSet` 与数据源之间的差别，并生成 SQL 语句，使前者与后者相匹配。请看下面的代码：

```
ds.Tables("tblUsers").Rows(3).Delete
```

同 `OleDbCommandBuilder` 对象和 `Update` 方法一起使用时，上述代码将生成如下所示的 SQL 语

句:

```
DELETE FROM tblUsers WHERE UserID = 4
```

OleDbDataAdapter 使用主关键字的值来确定要删除的记录。这里第 4 条记录的主关键字 UserID (由 Rows(3)可知) 为 4。仅当存在主关键字或唯一性的字段时, 自动生成特性才起作用。

### 3. 映射

表和字段映射让您能够将数据源的一个表或字段映射到 DataSet 中的表或字段。当数据从一个数据源传输到另一个数据源时, ADO.NET 将使用映射。这让您能够在不同地方使用不同的名称来引用字段或表, 甚至可以在两个完全不同的字段或表之间建立映射。图 10.9 说明了这种概念。

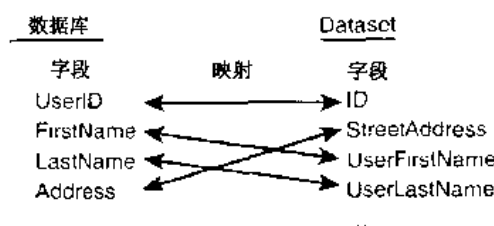


图 10.9 映射让您能够将两个看似无关的字段关联起来

这些映射存储在 OleDbDataAdapter 对象的 TableMapping 集合中。我们来看一个例子, 如清单 10.9 所示

#### 清单 10.9 为 banking 数据库创建表映射

```
1: Dim ds As DataSet = New DataSet()
2: Dim Conn as New OleDbConnection( _
3:     "Provider=Microsoft.Jet.OLEDB.4.0;" & _
4:     "Data Source=C:\ASPNET\data\banking.mdb")
5: Dim objCmd As New OleDbDataAdapter( _
6:     ("SELECT * FROM tblUsers", Conn)
7: )
8: 'add mappings
9: objCmd.TableMappings.Add('Table', 'Users')
10: With objCmd.TableMappings(0).ColumnMappings
11:     .Add('UserID', "ID")
12:     .Add("LastName", 'Lastname')
13:     .Add("FirstName", "Firstname")
14:     .Add("Phone", "Phone")
15:     .Add("Address", 'StreetAddress')
16:     .Add("City", "City")
17:     .Add('State', 'State')
18:     .Add('Zip', 'Zip')
19: End With
20: objCmd.Fill(ds)
```

分析：这个例子使用了第 8 章创建的 banking 数据库，并将 tblUsers 表中的字段映射到 DataSet 中的字段。

您对第 1-6 行应该很熟悉：创建并初始化 DataSet、连接和 OleDbDataAdapter 对象。第 9 行在数据源的表和 DataSet 的 Users 表之间建立了一个映射。其中第一个参数是源表（提供数据的表），第二个参数是目标表（接受数据的表）。由于在 OleDbDataAdapter 声明中提供了一个 Select 语句，所以 ADO.NET 知道您想使用 Select 语句返回的结果作为源表。

第 9 行指定的 table 是一个特殊的名称，供 ADO.NET 使用。如果执行 Fill 或 Update 命令时没有指定 DataSet 参数，ADO.NET 查找名为 Table 的映射，以确定数据的来源。例如，假设使用清单 10.9 中的映射，并执行下面的调用：

```
objCmd.Update(ds)
```

ADO.NET 将从 DataSet 的 Users 表取得数据，因为 Table 映射指示这样做。通常需要使用下面的语句来进行更新：

```
objCmd.Update(ds, "Users")
```

第 10-20 行将 ColumnMapping 添加到刚定义的 TableMapping 中。这些映射不仅为其中的几个字段提供了易于使用的字段名称，并将其映射到其他字段。

映射不只是提供了更友好的字段名。进一步熟悉 SQL 语句后，您将发现一些 SQL 语句不返回字段或指定的字段，而只返回一个值。然而，这种情况在 ASP.NET 开发过程中不常发生。我们将在第 12 章中做进一步的讨论。

## 10.4 在 ASP.NET 中使用 ADO.NET

理论准备充分后，我们开始讨论 ASP.NET 开发。您将创建第一个功能齐全（但是很简单）的数据库应用程序，让用户查看和修改数据。您需要将新学到的技术用于实践。

该应用程序只需创建一个页面，该页面使用 DataGrid 控件显示用户表中的数据。您将使用 DataGrid 的编辑特性来修改数据，然后使用 OleDbCommand 对象将所做的修改保存到数据源中。

首先创建用户界面。清单 10.10 列出了一个标准的 DataGrid 控件，对于数据库中的每一个字段，该控件都有一个对应的绑定列，该控件还包含一个 EditCommandColumn 和一个 ButtonCommandColumn 对象，用来删除数据。另外，还使用一个 panel 来显示其他的输入文本框，用于将新记录插入到数据库中。您还创建了一个标签控件，用于将消息显示给用户。

**清单 10.10 数据库应用程序的 UI 部分**

```
1 <html><body>
2   <asp:Label id="lblMessage" runat="server"/>
3
4   <form runat="server">
5     <asp:DataGrid id="dgData" runat="server"
6       BorderColor="black" GridLines="Vertical"
7       cellpadding="4" cellspacing="0" width="100%"
8       AutoGenerateColumns="False"
9       OnDeleteCommand="dgData_Delete">
```

```

10      OnEditCommand="dgData_Edit"
11      OnCancelCommand="dgData_Cancel"
12      OnUpdateCommand="dgData_Update"
13      OnPageIndexChanged="dgData_PageIndexChanged" >
14
15      <Columns>
16          <asp:TemplateColumn HeaderText="ID">
17              <ItemTemplate>
18                  <asp:Label id="Name" runat="server"
19                      Text="<%# Container.DataItem("UserID") %>" />
20              </ItemTemplate>
21          </asp:TemplateColumn>
22
23          <asp:BoundColumn HeaderText="FirstName"
24              DataField="FirstName" />
25          <asp:BoundColumn HeaderText="LastName"
26              DataField="LastName" />
27          <asp:BoundColumn HeaderText="Address"
28              DataField="Address" />
29          <asp:BoundColumn HeaderText="City"
30              DataField="City" />
31          <asp:BoundColumn HeaderText="State"
32              DataField="State" />
33          <asp:BoundColumn HeaderText="Zip"
34              DataField="Zip" />
35          <asp:BoundColumn HeaderText="Phone"
36              DataField="Phone" />
37
38          <asp:EditCommandColumn
39              EditText="Edit"
40              CancelText="Cancel"
41              UpdateText="Update"
42              HeaderText="Edit" />
43
44          <asp:ButtonColumn HeaderText="" text="Delete"
45              CommandName="delete" />
46      </Columns>
47  </asp:DataGrid><p>
48
49  <asp:Panel id="AddPanel" runat="server">

```

```

50     <table>
51     <tr>
52         <td width="100" valign="top">
53             First and last name:
54         </td>
55         <td width="300" valign="top">
56             <asp:TextBox id="tblName" runat="server" />
57             <asp:TextBox id="tblName" runat="server" />
58         </td>
59     </tr>
60     <tr>
61         <td valign="top">
62             Address:
63         </td>
64         <td valign="top">
65             <asp:TextBox id="tblAddress"
66                 runat="server" />
67         </td>
68     </tr>
69     <tr>
70         <td valign="top">
71             City, State, ZIP:
72         </td>
73         <td valign="top">
74             <asp:TextBox id="tblCity"
75                 runat="server" />,
76             <asp:TextBox id="tblState" runat="server"
77                 size="2" />&nbsp;
78             <asp:TextBox id="tblZIP" runat="server"
79                 size="5" />
80         </td>
81     </tr>
82     <tr>
83         <td valign="top">
84             Phone:
85         </td>
86         <td valign="top">
87             <asp:TextBox id="tblPhone" runat="server"
88                 size="1" /><p>
89         </td>

```

```

90         </tr>
91     </tr>
92     <td colspan="2" valign="top" align="right">
93         <asp:Button id="btSubmit" runat="server"
94             text="Add"
95             OnClick="Submit" />
96     </td>
97 </tr>
98 </table>
99 </asp:Panel>
100 </form>
101 </body></html>

```

该页面不需要完成太多的任务，只是显示几个具有不同属性的服务器控件。真正的工作是在代码声明块中完成的，稍后将讨论。该清单最重要的部分是 DataGrid 控件及其事件处理程序（第 9—13 行）。Panel 中的服务器控件让用户能够通过单击第 94 行的 Add 按钮，将新数据加入到数据库中。

下面来看看代码声明块。这部分要实现几项功能，包括将数据装载到 DataGrid 中、处理 DataGrid 的事件以及当用户填写好表单并单击 Submit 按钮后将数据加入到数据库中。清单 10.11 列出了这些代码。

#### 清单 10.11 清单 10.10 的 ASP.NET 代码

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     'declare connection
7     dim Conn as new OleDbConnection( _
8         "Provider=Microsoft.Jet.OLEDB.4.0;" & _
9         "Data Source=H:\ASPNET\data\banking.mdb")
10
11     sub Page_Load(obj as Object, e as EventArgs)
12         if Not Page.IsPostBack then
13             FillDataGrid()
14         end if
15     end sub
16
17     sub Submit(obj as object, e as EventArgs)
18         'insert new data
19         dim i, i as integer
20         dim params(7) as string

```

```

21     dim strText as string
22     dim blnGo as boolean = true
23
24     j = 0
25
26     for i = 0 to AddPanel.Controls.Count - 1
27         if AddPanel.Controls(i).GetType Is _
28             GetType(TextBox) then
29             strText = CType(AddPanel.Controls(i), _
30                 TextBox).Text
31             if strText <> "" then
32                 params(j) = strText
33             else
34                 blnGo = false
35                 lblMessage.Text = lblMessage.Text & _
36                     "You forgot to enter " & _
37                     "a value for " & AddPanel.Controls(i).ID & "<br>"
38                 lblMessage.Style("ForeColor") = "Red"
39             end if
40             j = j + 1
41         end if
42     next i
43
44     if not blnGo then
45         exit sub
46     end if
47
48     dim strSQL as string = "INSERT INTO tblUsers " & _
49         " (FirstName, LastName, Address, City, State, " & _
50         " Zip, Phone) VALUES (" & _
51         "'" & params(0) & "'," & _
52         "'" & params(1) & "'," & _
53         "'" & params(2) & "'," & _
54         "'" & params(3) & "'," & _
55         "'" & params(4) & "'," & _
56         "'" & params(5) & "'," & _
57         "'" & params(6) & "')"
58
59     ExecuteStatement(strSQL)
60

```

```

61     FillDataGrid()
62 end sub
63
64 sub dgData_Delete(obj as object, e as DataGridCommandEventArgs)
65     FillDataGrid(e.Item.ItemIndex)
66 end sub
67
68 sub dgData_Delete(obj as object, e as DataGridCommandEventArgs)
69     dim strSQL as string = "DELETE FROM tblUsers    & _
70         'WHERE UserID = " & e.Item.ItemIndex + 1
71
72     ExecuteStatement(strSQL)
73
74     FillDataGrid()
75 end sub
76
77 sub dgData_Update(obj as object, e as DataGridCommandEventArgs)
78     if UpdateDataStore(e) then
79         FillDataGrid(-1)
80     end if
81 end sub
82
83 sub dgData_Cancel(obj as object, e as DataGridCommandEventArgs)
84     FillDataGrid(-1)
85 end sub
86
87 sub dgData_PageIndexChanged(obj as Object, e as DataGridPageChangedEventArgs)
88     dgData.DataBind()
89 end sub
90
91 function UpdateDataStore(e as DataGridCommandEventArgs) _
92     as boolean
93
94     dim i,j as integer
95     dim params(7) as string
96     dim strText as string
97     dim blnGo as boolean = true
98
99     i = 0

```



```

100     for i = 1 to e.Item.Cells.Count - 3
101         strText = CType(e.Item.Cells(i).Controls(0), _
102             TextBox).Text
103         if strText <> "" then
104             params(j) = strText
105             j = j + 1
106         else
107             binGo = false
108             lblMessage.Text = lblMessage.Text & _
109             'You forgot to enter   & _
110             "a value<p>"
111         end if
112     next
113
114     if not binGo then
115         return false
116         exit function
117     end if
118
119     dim strSQL as string = 'UPDATE tblUsers SET " & _
120         'FirstName = "' & params(0) & '" & _
121         'LastName = "' & params(1) & '" & _
122         'Address = "' & params(2) & '" & _
123         'City = "' & params(3) & '" & _
124         'State = "' & params(4) & '" & _
125         'Zip = "' & params(5) & '" & _
126         'Phone = "' & params(6) & '" & _
127         " WHERE UserID = " & CType(e.Item.Cells(0), _
128             Controls(1), Label).Text
129
130     ExecuteStatement(strSQL)
131     return binGo
132 end function
133
134 sub FillDataGrid(Optional EditIndex as integer=-1)
135     'open connection
136     dim objCmd as new OleDbCommand _
137         ("select * from tblUsers", Conn)
138     dim objReader as OleDbDataReader

```

```

139     try
140         objCmd.Connection.Open()
141         objReader = objCmd.ExecuteReader()
142     catch ex as Exception
143         lblMessage.Text = "Error retrieving from the database. Please" & _
144             " make sure all values are correctly input"
145     end try
146
147     dgData.DataSource = objReader
148     if not EditIndex.Equals(Nothing) then
149         dgData.EditItemIndex = EditIndex
150     end if
151
152     dgData.DataBind()
153
154     objReader.Close()
155     objCmd.Connection.Close()
156
157 end sub
158
159 function ExecuteStatement(strSQL)
160     dim objCmd as new OleDbCommand(strSQL, Conn)
161
162     try
163         objCmd.Connection.Open()
164         objCmd.ExecuteNonQuery()
165     catch ex as Exception
166         lblMessage.Text = "Error updating the database. Please" & _
167             " make sure all values are correctly input"
168     end try
169
170     objCmd.Connection.Close()
171 end function
171 </script>

```

**分析:** 代码很长,但并不像看起来的那么可怕。首先让我们来看看用来读取数据并填充 DataGrid 的方法,然后再讨论更新数据库的方法。

第 7 行(不位于任何方法内)声明了一个 OleDbConnection 对象。由于该对象不属于任何特定的方法(而属于整个页面),因此任何方法都可以使用它。首先执行(总是这样)的方法是 Page\_Load,如第 11~15 行所示。该方法检查表单是否被提交,如果没有则调用另一个方法: FillDataGrid(如第 135~157 行所示,您应该很熟悉),该方法负责从数据库取得数据,并将其绑定到 DataGrid。请注意

该方法声明中的可选参数 `EditIndex`，稍后将解释它。在第 135–137 行，您使用第 7 行声明的 `OleDbConnection` 对象创建了 `OleDbCommand` 对象和 `OleDbDataReader` 对象。

接下来，将对数据库的调用放在一条 `try` 语句中。如果没有错误发生，该语句将尝试执行一个代码块；如果发生错误，则使用 `catch` 语句（如第 142 行所示）来处理错误，然后继续执行。如果不这样做，应用程序将瘫痪。以前之所以没有这样做，是因为都只是一些简单的范例。然而，当需要连接到 ASP.NET 环境之外的系统时（这里是数据库），常常会发生错误。开发专业应用程序时，必须防止这些错误导致应用程序崩溃（甚至更糟糕的是，让用户看到有错误出现）。

应 该	不 应 该
连接到 ASP.NET 环境之外的任何对象（包括数据库、管理组件和 COM 对象等）时，一定要使用 <code>try..catch..finally</code> 语句	不要期望一切都正常运行，而应该预防一切可能发生的问 题

如果不能理解这一点，也不用担心。在下 一部分讨论调试（参见第 20 章）时，将详细介绍 `try` 语句。

在 `try` 块内，您连接到数据库并试图填充 `OleDbDataReader`。如果连接失败，则使用标签控件向用户发送一条消息。第 147 行将 `DataGrid` 的 `DataSource` 设置为 `DataReader`。在绑定数据之前，让我们再来看看可选参数 `EditIndex`。每当用户试图编辑条目时，都必须正确设置 `DataGrid` 的 `EditItemIndex` 属性，然后再绑定数据。这种方法让您能够正确地设置 `EditItemIndex` 属性，从而省去以后再设置编辑模式和重新绑定数据的麻烦。因此，第 148 行使用 `if` 语句来检查 `EditIndex` 参数数据是否被指定。如果已指定，则设置 `DataGrid` 的 `EditItemIndex` 属性。

最后，在第 154 和 155 行关闭了 `DataReader` 对象和连接对象。在浏览器中请求该页面，将得到如图 10.10 所示的结果，其中的数据随数据库内容而定。

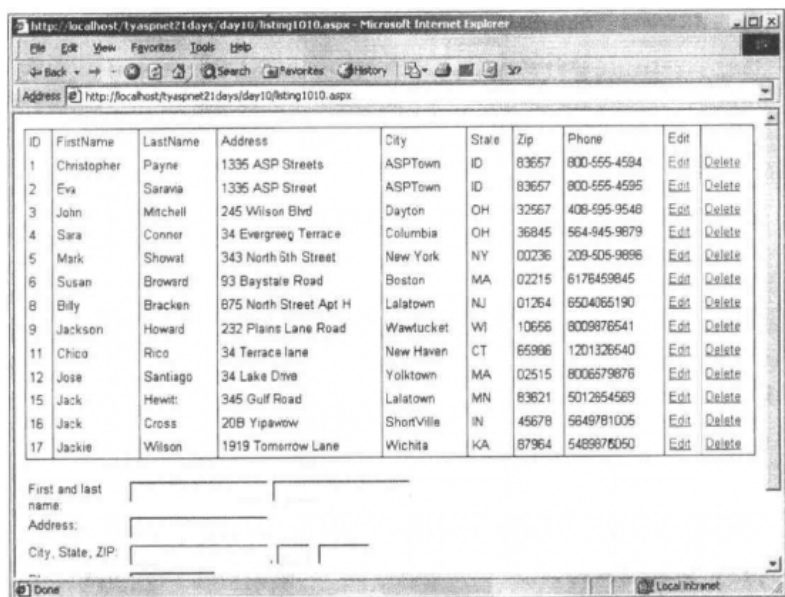


图 10.10 数据库程序的 UI

现在，可以查看数据了，但还需要添加方法，以使用户可以编辑 `DataGrid` 中的条目。第 64、68、77、83、91 行定义了 5 种方法，它们分别在 `DataGrid` 的 `Edit`、`Delete`、`Update`、`Cancel` 和 `Update`

事件发生时执行。

对于 DataGrid 的 Edit 和 Cancel 命令, 只需使用可选的编辑索引参数调用 FillDataGrid 方法即可, 如下所示:

```
sub dgData_Edit(obj as object, e as DataGridCommandEventArgs)
    FillDataGrid(e.Item.ItemIndex)
end sub

sub dgData_Cancel(obj as object, e as DataGridCommandEventArgs)
    FillDataGrid(-1)
end sub
```

Edit 方法将 DataGrid 的 EditItemIndex 属性设置为用户选择的条目, 而 Cancel 将给属性设置为 -1, 以退出编辑模式。

对于 Update 命令, 则需要一定的创造性。您需要收集文本框中的所有数据, 当用户单击 Edit 链接时, 这些数据将动态地生成。因此需要一个 for 循环, 如第 100 行所示。该循环遍历所选项中除最后三个字段外的所有字段 (记住, 最后三个字段中不含任何数据, 它们是按钮 Update、Cancel 和 Delete), 并将值存储在变量 strText 中, 如第 101 行所示。如果变量 strText 不是空字符串, 则将其值存储在一个数组中, 供以后更新数据库时使用。如果为空, 则终止这一过程, 并提醒用户, 如第 106~108 行所示。

注意循环变量 i 和 j。i 用于遍历记录中的各个字段, 而 j 用作参数数组的索引。同时使用 i 和 j 的原因在于, 字段的索引号与数组的索引不相同。例如, FirstName 单元格索引号为 1, 但在数组中, 其索引号为 0。如果不这样做, 将发生 “index out of bound” 错误。您使用 j 来跟踪数组的索引号。图 10.11 说明了这一概念。

单元格	0	1	2	3	4	5	6	7	8	9
	ID	FirstName	LastName	Address	City	State	Zip	Phone	Edit	Delete
	1	Christopher	Payne	335 ASP Street	ASPTown	IL	63657	800-555-4594	Edit	Delete
数组索引号		0	1	2	3	4	5	6	-	-

图 10.11 单元格索引号和数组索引号不同

如果任何一个文本框为空, 则第 106 行的布尔参数 blnGo 会指出这一点。如果其值为 false, 即至少一个文本框为空, 则必须停止更新, 否则数据将是无效的。您当然不希望用户数据库包含空的字段。上述工作是在第 113~116 行完成的:

```
if not blnGo then
    return false
exit function
end if
```

接下来需要创建 SQL Update 语句。由于所有的值存储在数组 param 中, 所以创建工作很容易, 如第 118~127 行所示。

第 126~127 行是插入语句的 where 子句。因为您只想更新那些编辑过的记录, 所以需要确定这些记录的标识字段。幸运的是, 标识字段是 UserID, 它已存在于 DataGrid 每行的第一个单元格中。

您从该单元格获得该控件，将其转换为一个标签，并取得其中的文本。根据图 10.10，如果编辑的是第一行，则使用如下的 SQL 语句：

```
UPDATE tblUsers SET FirstName = 'Christopher', LastName =
➤ 'Payne', Address = '1335 ASP Street', City = 'ASPTown',
➤ State = 'TX', Zip = '80657', Phone = '800-500-4594'
➤ WHERE UserID = 1
```

最后只需执行更新语句，这是通过调用另外一个方法 `ExecuteStatement`（如第 159–170 行所示）实现的。该方法只是封装了创建 `OleDbCommand` 对象和执行 SQL 语句的步骤。

同样，这里使用了一个 `try...catch` 语句来确保没有未处理的错误，并使用 `ExecuteNonQuery()` 方法来完成更新。

下面来看看 `Delete` 命令方法，如第 68–75 行所示：

```
sub dgData_Delete(obj as object, e as DataGridCommandEventArgs)
    dim strSQL as string = "DELETE FROM tblUsers " & _
        "WHERE UserID = " & e.Item.ItemIndex + 1

    ExecuteStatement(strSQL)

    FillDataGrid()
end sub
```

相对而言，该方法很简单。创建 SQL `Delete` 语句时，您在 `WHERE` 子句中使用了记录的 `UserID` 字段，然后执行该语句，并重新填充 `DataGrid`。

需要完成的最后一项工作用户填写好表单，并单击 `Add` 按钮后，将新记录加入到数据库中。第 17–62 行的 `Submit` 方法类似于刚才介绍的 `Update` 方法，只是执行的是 `Insert` 命令而不是 `Update`。

我们来简要地讨论一下 `Submit`。别忘了将所有的用户输入控件放在一个 `Panel` 中，因为这样才能遍历这些控件，就像遍历 `DataGrid` 中的单元格一样。现在，需要遍历的是 `Panel` 的 `Controls` 集合，而不是 `DataGridCommandEventArgs` 的 `Cells` 集合。现在知道了使用 `Panel` 的原因了吧！另外，SQL 语句的语法也稍有不同。除此之外，`Delete` 方法与 `Update` 方法完全相同。

祝贺您，一切都完成了！您现在拥有了一个完整的应用程序，允许用户更新、添加、删除数据库中的记录。该应用程序的代码很长，但大部分代码都很简单。

另外，您也通过将功能尽可能划分为独立的函数，实践了一些良好的代码设计方法。将应用程序模块化可以极大地提高其可读性和代码的重用性。将其中的大部分代码复制到其他应用程序中，无需做太多的修改，它们便能运行。

您也许已注意到，在 `DataGrid` 中使用 `EditCommand` 时，对字段显示的控制能力可能有所减弱。例如，文本框比实际需要的要长。以后使用 `DataGrid` 控件时，最好使用 `TemplateColumns` 和文本框，而不是 `BoundColumns` 和 `EditCommand`，因为前者比后者更易使用。不仅如此，`TemplateColumns` 和文本框的事件更多，能更好地控制显示方式。但这只是形式问题，您可以根据自己的喜好进行处理。

## 10.5 这不是 ASP

如果您熟悉传统的 ASP 或 ADO，您或许想知道一向受人称道的 `RecordSet` 的情况。

尽管真正的 ADO DB RecordSet 已成为历史,但其概念仍在。本质上,DataReader 是一个使用单向游标(只向前)的 RecordSet,而 DataSet 是一个使用可滚动(或动态)的 RecordSet。当然新的 ADO.NET 具有更多的功能,但基本功能与 ADO 相似。

使用 RecordSet 时,不管游标类型如何,您始终是与数据库相连的,因此需要锁定数据。这样做的效率可能相当低,尤其并行用户很多时。DataSet 完全是一个离线的数据存储器,这意味着将数据取出后,您就可以断开与数据库的连接,而不需要一直与数据库相连,也无需锁定数据。另外,DataSet 对数据源的表达仅仅局限于其数据。正如本章介绍的,DataSet 可以含有多个表(而对 RecordSet 来说,这是不可能的)、关系和映射。DataSet 是一个强大的数据处理模型,可以解决传统 ADO 中存在的一些难题。当然,得到这些功能是需要付出代价的。因此,ADO.NET 提供了一个与 RecordSet 更类似的控件:OleDbDataReader,不同之处在于前者是完全面向对象的,用于处理类型安全的数据。

OleDbConnection 对象和 OleDbCommand 对象非常类似于传统 ADO 的 ADO DB Connection 对象和 Command 对象,所以向前者转移应该相当容易。传统 ASP 应用的一些概念,如事务和参数对象,在 ASP.NET 中仍然适用。事实上,我们将在第 12 章更详细地介绍这些内容。

## 10.6 总 结

本章也很长,但您学到的知识也很多。现在您熟悉了数据库驱动的 ASP.NET 应用程序,使用数据绑定和列表控件也将得心应手。您已迅速地从一名新手成为一名中高级水平的开发人员了。

本章介绍了 ADO.NET 及其基础知识。ADO.NET 包括两个主要的部分:DataSet 和管理提供程序。DataSet 已在上一章介绍过了,本章重点介绍的是其数据模型的技术方面,包括 RowState 和并发。另外,还介绍了 OLE DB 管理提供程序及其对象,包括 OleDbCommand 和 OleDbDataAdapter 对象,这些对象为同数据库交互提供了大量的功能。

OleDbDataReader 相当于一个小型的 DataSet,对提高性能很有帮助。但 DataSet 的功能更为强大。一般说来,如果只想显示数据,请使用 OleDbDataReader。

最后,全面应用 ADO.NET 和 ASP.NET 功能创建了一个功能齐全的数据库驱动应用程序。将各种知识组合起来很有帮助,并指出了开发完整的应用程序时需要考虑的其他问题。

下一章将介绍 XML,指出如何在 ASP.NET 中使用 XML。XML 让您能够开发出真正的分布式应用程序,对于开发 Web 服务来说,这是必不可少的。

## 10.7 问与答

问: OleDbDataAdapter 与 OleDbCommand 对象完全不同吗?为什么存在两个不同的对象?

答:事实上,它们不完全是独立的。OleDbDataAdapter 是更抽象的 OleDbCommand 对象。当您设置 SelectCommand、UpdateCommand、InsertCommand 和 DeleteCommand 对象时,实际上是创建了新的 OleDbCommand 对象。这些对象用于同数据库交互,以执行 SQL 语句。使用 OleDbDataAdapter 只是可以将数据放入 DataSet 中(TableMappings 也是如此)。

可以将 OleDbDataAdapter 看作一个高级对象,而将 OleDbCommand 看作一个直接与数据库直交互的低级对象。

## 10.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 10.8.1 小测验

1. ADO.NET 的两个主要部分是什么？
2. 管理提供程序的功能是什么？
3. 什么是离线数据？
4. 对 `OleDbCommand` 对象而言，下面哪个构造程序是无效的（第 4、5、8 还是 9 行的）？  

```

dim strSQL as string = 'SELECT * FROM tblUsers'
dim Conn as New OleDbConnection( DSN=21DaysBanking')
dim objCmd as new OleDbCommand()
dim objCmd as new OleDbCommand(strSQL)
dim objCmd as new OleDbCommand(strSQL, _
    Conn)
dim objCmd as new OleDbCommand(Conn)
dim objCmd as new OleDbCommand(strSQL, _
    'DSN=21DaysBanking')

```
5. `OleDbCommand` 填充哪种数据模型？
6. 需要不需要关闭 `OleDbCommand` 对象？
7. `OleDbDataAdapter` 对象的五个成员是什么？
8. 下列 `OleDbDataAdapter` 对象的构造函数中，哪个是无效的（第 5、6、7、8 还是 9 行的）？

```

dim strSQL as string = "SELECT * FROM tblUsers"
dim Conn as new OleDbConnection("DSN=21DaysBanking")
dim objADOCmd as new OleDbCommand(strSQL, Conn)
dim objCmd as new OleDbDataAdapter()
dim objCmd as new OleDbDataAdapter(objADOCmd)
dim objCmd as new OleDbDataAdapter(strSQL, _
    Conn)
dim objCmd as new OleDbDataAdapter(Conn)
dim objCmd as new OleDbDataAdapter(strSQL, _
    DSN=21DaysBanking')

```

### 10.8.2 练习

修改本章创建的范例应用程序，请使用 `OleDbDataAdapter` 和 `DataSet` 对象来代替 `OleDbCommand` 和 `OleDbDataReader` 对象。

## 第 11 章

# 在 ASP.NET 中使用 XML

至此，您应牢固地掌握 ASP.NET 中的数据存取知识。您知道如何使用 ADO.NET 来存取、显示和修改数据——实际上是几种不同的方法。然而，如果不涉及 XML（可扩展标记语言），则对数据库和 Internet 的讨论将是不完整的。

XML 是一种新的、在 Web 上表示数据的通用语言。XML 已引起广泛的关注，因为它解决了与数据存储和传输相关的问题，如安全、易懂性、可读性和数据转换等。

ASP.NET 和 .NET 框架在设计上十分注重 XML，因为 XML 不仅增强了其功能，并使其更易于使用。本章将介绍有关 XML 的知识以及 XML 在 ASP.NET 框架中的地位，并花大量的篇幅讨论 XML 的读、写和转换。

本章包含以下内容：

- 什么是 XML；
- 如何使用 XmlTextReader 和 XmlTextWriter 对象来读、写和验证 XML；
- 如何使用 XML 文档对象模型；
- XML.NET 框架如何使用关系型数据。

### 11.1 XML 简介

XML 是一种基于文本的格式，用来描述数据。XML 在给 Web-enabled 应用程序递送数据方面提供了新的方式，使得几乎任何数据都可以被发送并在任何地方使用。这听起来有些神奇，但 XML 的功能来自两个简单的特性。首先，XML 是可扩展的（名称可扩展标记语言由此而来），这意味着通过添加自定义的标记符和结构，可以很容易地对其进行扩展。另外，XML 是基于文本的，这意味着可以使用任何类型的文本编辑器（如记事本）来创建或阅读 XML。

同 HTML 一样，XML 使用简单的置标标记来描述其内容。但与 HTML 不同的是，XML 没有标准的标记——您自己创建标记。这正是 XML 的强大之处——您可以根据需要创建任何类型的标记，以便表示任何类型的数据。这还意味着 XML 易于读取和修改。

例如，您可以创建下面的标记，而 XML 可以理解它们，虽然 HTML 浏览器不能理解：

```
<Name>...</Name>
<Occupation>...</Occupation>
< FavRestaurant>...</FavRestaurant>
```



XML 以纯文本表示, 所以如果您愿意, 可以使用记事本来创建 XML 数据。因此, 可以很容易地将数据传输到不同的地方。对于大多数的数据库应用程序来说, 内部数据都是以数据库特定的格式存储的。当在同一个工程中使用不同的数据源 (或不同的计算机平台) 时, 通常需要进行从一种格式到另一种格式的复杂转换操作。然而, 使用 XML 时, 可将数据表示成一种结构化的文本格式, 从而消除了复杂的转换工作。由于其文本性质, XML 对用户而言易于阅读和理解。另外, XML 可以避免复杂的安全措施, 后者常常会阻碍其他类型的通信。但安全措施通常不限制文本的出入, 所以不论在任何地方传输数据, XML 都是非常理想的。

**注意:** Internet 应用程序常常需要使用多个位于不同平台上的数据源。对于这种应用程序, XML 是一种理想的数据表示方式。

### 11.1.1 XML 数据模型

清单 11.1 包含 XML 文件, 该文件表示的是书店的库存目录。

**清单 11.1 XML 格式的书店库存目录**

```

1 <bookstore>
2   <book genre="novel" style="hardcover">
3     <title>The Handmaid's Tale</title>
4     <price>19.95</price>
5     <author>
6       <first-name>Margaret</first-name>
7       <last-name>Atwood</last-name>
8     </author>
9   </book>
10  <book genre="novel" style="paperback">
11    <title>The Poisonwood Bible</title>
12    <price>11.99</price>
13    <author>
14      <first-name>Barbara</first-name>
15      <last-name>Kingsolver</last-name>
16    </author>
17  </book>
18  <book genre="novel" style="hardback">
19    <title>Hannibal</title>
20    <price>27.95</price>
21    <author>
22      <first-name>Richard</first-name>
23      <last-name>Harris</last-name>
24    </author>
25  </book>
26  <book genre="novel" style="hardback">
27    <title>Poe's Pendulum</title>

```

```

28 <price>22.95</price>
29 <author>
30   <first-name>Umberto</first-name>
31   <last-name>Eco</last-name>
32 </author>
33 </book>
34 </bookstore>

```

**新术语:** 将该文件保存为 books.xml。本章后面关于 XML 模式一节将介绍具体的细节。现在, 请注意 XML 是由结构化、层次式标记符组成的。这里有两个<book>标记, 每个都有自己的属性 (genre 和 style) 和几个子元素 (title、author 和 price)。实际数据包含在子元素标记中, 整个数据集包含在<bookstore>标记中, 后者描述了该数据集。这种数据表示方法通常被称为文档树 (Document tree) 或数据树 (Data tree)。

在传统的数据库 (如 Access) 中, 数据的表示类似于图 11.1。

Bookstore					
Genre	Style	Title	AuFirstName	AuLastName	Price
novel	hardcover	The Handmaid's Tale	Margaret	Atwood	19.95
novel	paperback	The Poisonwood Bible	Barbara	Kingsolver	11.99

图 11.1 在 Microsoft Access 查看清单 11.1 中的 XML 数据

XML 数据的移植性更好, 其他人更容易读取和使用, 而且建立时不需要复杂的机制。让我们将该清单保存为文本文件 books.xml, 供以后使用。将该清单保存在文件夹 C:\inetput\wwwroot\tyaspnet21days\day11 中 (或其他容易记住的地方)。现在在浏览器中查看该文件, 如果使用的浏览器较新 (IE5.0 或更高版本), 将看到如图 11.2 所示的结果。



图 11.2 在浏览器中查看 XML 文件

IE 能够自动分析 XML，并以层次结构进行显示，可以单击“-”来关闭分支，也可以单击“+”来展开分。XML 提供了一种神奇的数据表达机制。

### 11.1.2 XML 模式

如果定义了自己的标记，其他人如何知道这些标记的含义呢？XML 模式（Schema）定义了这种格式。我们来看看清单 11.1 定义的模式。

清单 11.2 清单 11.1 的 XML 模式

```

1 <?xml version="1.0"?>
2 <Schema xmlns="urn:schemas-microsoft-com:xml-data"
3         xmlns:dt="urn:schemas-microsoft-com:datatypes">
4   <ElementType name="first-name" content="textOnly"/>
5   <ElementType name="last-name" content="textOnly"/>
6   <ElementType name="name" content="textOnly"/>
7   <ElementType name="price" content="textOnly"
8         dt:type="fixed14.4"/>
9   <ElementType name="author" content="eltOnly" order="one">
10     <group order="seq">
11       <element type="name"/>
12     </group>
13     <group order="seq">
14       <element type="first-name"/>
15       <element type="last-name"/>
16     </group>
17   </ElementType>
18   <ElementType name="title" content="textOnly"/>
19   <attributeType name="genre" dt:type="string"/>
20   <attributeType name="style" dt:type="enumeration"
21         dt:values="paperback hardcover"/>
22   <ElementType name="book" content="eltOnly">
23     <attribute type="genre" required="yes"/>
24     <attribute type="style" required="yes"/>
25     <element type="title"/>
26     <element type="author"/>
27     <element type="price"/>
28   </ElementType>
29   <ElementType name="bookstore" content="eltOnly">
30     <element type="book"/>
31   </ElementType>
32 </Schema>

```

将该清单保存在 books.xml 所在的文件夹中，并将其命名为 books-schema.xdr，该清单看起来也

很像 HTML，但远非如此，让我们仔细地看看。第 1 行指定了要创建的 XML 类型，这里为 1.0 版本。为使本章的范例正常运行，这一行是必不可少的。第 2 行打开 <schema> 标记。Xmldb 表示 XML 名称空间，它是一组标准的 XML 标记，用于实现进一步的标准化。您现在还不需要对此有更多的了解。指定名称空间时通常使用标准的名称空间，如这里的 schema-microsoft-com:xml-data。

<ElementType> 标记用来定义数据格式。第 4-7 行定义了 first-name、last-name、name 和 price 四个元素，将在该模式的后面使用它们。将这些元素放在这里类似于在页面的开始位置声明变量。Content 属性定义了该标记内可能放置的数据类型，datatype 定义了其几个属性，如数据类型以及如何格式化。

第 9-17 行定义了另外一个元素：author。该元素内部也有几个其他的标记，这些标记是由第 4-7 行描述的元素定义的。第 18-21 行定义了将在模式中使用其他几个元素和属性。

最后，第 22 行定义了 book 元素，其中包含前面定义的所有元素。该部分必须与 XML 文件的格式匹配。第 29 行定义了一个包含 <book> 元素的元素。

注意：不需要再使用元素定义它们——可以在 book 元素中定义所有的元素。然而，事先定义元素对于建立模式来说是一种更好的方法，就像在 ASP.NET 页面中使用变量之前定义它们是个好主意一样。

Books-schema.xdr 模式定义了数据库表中的字段，而 books.xml 定义了数据库的记录。这样这两个文件几乎可以表示任何类型的数据。

**新术语：**遵循某种模式，并正确地表示其标记（即必须遵守 3W 联盟制定的标准）的 XML 文件被称为成型（well-formed）的 XML 文档。创建成型的 XML 文档确保任何 XML 顺应应用程序都能够读取其中的数据。一般说来，成型的 XML 文档必须遵循以下原则：

- 必须至少包含一个元素；
- 必须只包含一个开始和结束标记，开始和结束标记组成根元素，包含整个文档；
- 其他所有标记必须包含在开始和结束标记之间，同时不能交叉。

有关 W3C 标准文档的更详细的信息，请访问下面的站点：

<http://www.breference.com/xml/reference/standards.html>

## 11.2 在 ASP.NET 中存取 XML

访问 XML 类似于使用 ADO.NET 访问数据库。NET 框架提供了很多对象，让您能够对数据有不同程度的控制能力，每种对象都有其优缺点。本节将介绍两个最简单的对象：System.XML 名称空间中的 XMLTextReader 对象和 XMLTextWriter 对象。

### 11.2.1 读取 XML

XMLTextReader 为存取 XML 文件中的原始数据提供了一种简单、快速的机制。该对象类和 OleDbDataReader 对象一样，只能向前存取，不像 DataSet 那样需要巨大的系统开销。

要打开 XML 文件，只需创建一个新的 XMLTextReader，并将相应 XML 文件的文件名传递给它即可。如果创建的 ASP.NET 页面文件名为 XMLReader.aspx，并且放在 XML 文件所在的目录中，则您只需使用下面的语句：

```
dim reader As new XmlTextReader(file name)
```

这里不用担心连接字符串或 DSN，但必须提供完整的路径。该路径可通过 Server.MapPath 方法

得到（见第4章）。要存取数据，可使用 Read 方法，该方法类似 OleDbDataReader，例如：

```
Do While (reader.Read())
    ' Do some work on the data
Loop
```

调用 Read 方法将自动遍历 XML 文件。清单 11.3 列出了一个 ASP.NET Web 页面，该页面使用 XMLTextReader 类来阅读和显示 books.xml 文件中的内容。

**清单 11.3 使用 XMLTextReader 存取 XML 数据**

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>
3
4 <script runat="server">
5     sub Page_Load(ByVal obj as Object, e as EventArgs)
6         dim reader as XMLTextReader
7         '
8         try
9             reader = new XMLTextReader(Server.MapPath _
10                ("books.xml"))
11             While reader.Read()
12                 Response.Write("<b>" & reader.Name & "</b> " & _
13                    reader.Value & "<br>")
14             End While
15         catch ex as Exception
16             Response.Write("Error accessing XML file")
17         finally
18             reader.close
19         end try
20     end sub
21 </script>
22
23 <.html><body>
24
25 </body></html>
```

**分析：**将该清单保存为 XMLReader.aspx。首先要注意的新东西是第 2 行的 Import 语句，它导入名称空间 System.Xml。该 Import 语句让您能够使用 XML 相关类，如 XMLTextReader。

接下来，第 6 行声明了 XMLTextReader。然后将存取 XML 的代码放在 try 语句块中（记住，访问 ASP.NET 外面的资源时，应该使用 try 语句）。第 9 行使用 XML 文件打开了一个阅读器，该语法要求指定 XML 文件的完整路径，所以使用 Server.MapPath 来返回这种信息。具体地说，Server.MapPath 将返回 c:\inetpub\wwwroot\Myaspnet21days\day11\books.xml。

最后，使用 Read 方法遍历 XML 文件的内容，并在浏览器中显示每个条目的名称和值。别忘了

关闭阅读器，如第 16 和 17 行中的 finally 语句块所示。图 11.3 是在浏览器中查看清单得到的结果。

图 11.3 包含 XML 文件中的所有标记，包括开始和结束标记——这不是您希望的。不幸的是，XMLTextReader 提供的功能很有限，它不关心返回的是什么标记，而是将所有的标记全部返回！

为解决这种问题，XMLTextReader 提供了一个 NoteType 属性，指出查看的数据是何种类型的。表 11.1 列出了一些最常用的节点类型。

表 11.1 XMLTextReader 的 NoteType 的取值

类 型	描 述
All	所有节点
Attribute	一个属性
CDATA	转义那些会被看作标记语言（如 HTML）的文本
Comment	使用<!--和-->分隔的注释
Document	XML 数据树的根节点
Element	一个元素，通常是 XML 文件中的实际数据
EndTag	元素的结束位置
None	不是节点
Text	返回元素的文本内容
XMLDeclaration	XML 声明节点，例如<?XML version='1.0'?>

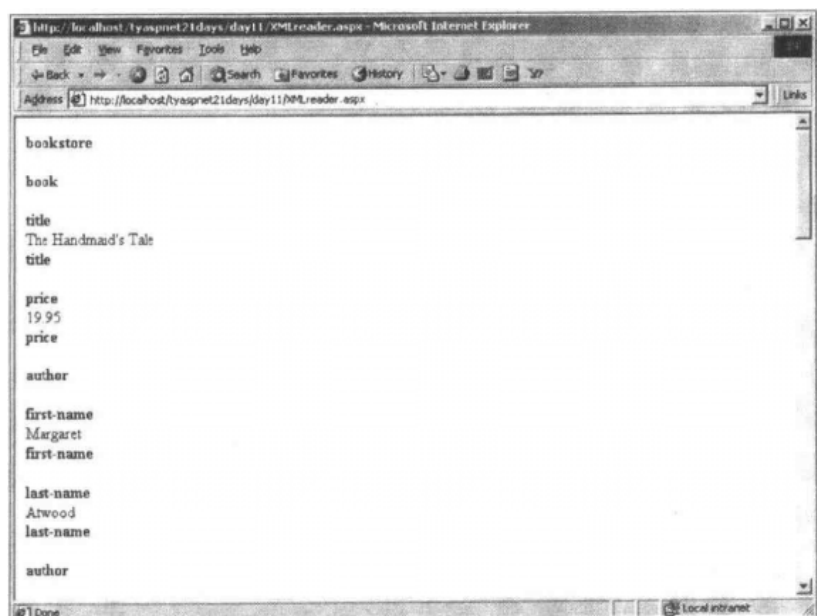


图 11.3 显示 books.xml 的内容

下面使用 NoteType 属性来修改清单 11.3，以便只返回需要的数据。

## 清单 11.4 使用 NoteType 来返回数据

```

1 <% @ Page language="VB" %>
2 <%@ Import Namespace='System.Xml' %>
3
4 <script runat=server>
5     sub Page_Load(obj as object, e as EventArgs)
6         dim reader as XMLTextReader
7         dim i as integer
8
9         try
10            reader = new XMLTextReader(Server.MapPath( ..
11                ("books.xml"))
12            while reader.Read()
13                Select Case reader.NodeType
14                    Case XMLNodeType.Element
15                        if reader.HasAttributes then
16                            For i = 0 to reader.AttributeCount - 1
17                                Response.Write(reader.GetAttribute(i) & " ")
18                            next
19                            Response.Write("<br>")
20                        end if
21                    Case XMLNodeType.Text
22                        Response.Write(reader.Value & "<br>")
23                End Select
24            End While
25        catch ex as Exception
26            Response.Write("Error accessing XML file")
27        finally
28            reader.close
29        end try
30    end sub
31 end script>
32
33
34 <html><body>
35
36 </body></html>

```

**分析：**第 13 行插入了一个 case 语句，以便在输出数据前确定节点的类型。如果 reader 对象的 NodeType 值为 Element，则执行第 14 行的第一个 case 语句。如果节点包含属性，则遍历这些属性，并使用 GetAttribute 方法返回相应的值。如果节点只是文本，则第 23 行直接输出节点的值 图 11.4

是清单 11.4 的结果。

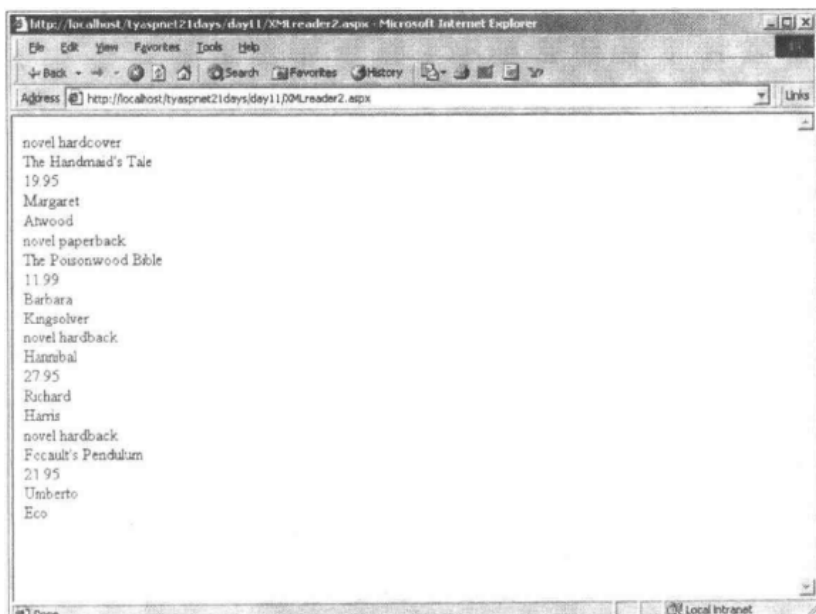


图 11.4 使用 NoteType 显示 books.xml 的内容

如果知道 XML 文件的具体结构，例如元素名及其属性，就可以进一步定制代码以便只显示感兴趣的条目。例如，如果只想看到 genre 属性，则可将下面的语句放在第 13-24 行的属性循环中：

```
Response.Write(reader.Item("genre") & "<br>")
```

### 11.2.2 写 XML

正如 XMLTextReader 使得读取 XML 文件很容易一样，XmlTextWriter 使得写 XML 文件很容易。XmlTextWriter 提供的 write 方法能够提供合适的输出。

例如，清单 11.5 在关于书店的 XML 文件中添加一个条目。

#### 清单 11.5 使用 XmlTextWriter 写 XML 文件

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object, e as EventArgs)
6         dim writer as XmlTextWriter
7
8         try
9             writer = new XmlTextWriter(Server.MapPath _
10                ('books2.xml'), nothing)
11
12             writer.WriteStartDocument
13             writer.Formatting = Formatting.Indented
```



```

11     writer.Indentation = 3
12
13     writer.WriteStartElement("bookstore")
14
15     writer.WriteStartElement("book")
16
17     writer.WriteAttribute("genre", "history")
18     writer.WriteAttribute("style", "hardcover")
19
20     writer.WriteString("title", "Vietnam");
21
22     writer.WriteStartElement("author");
23
24     writer.WriteString("first-name", _
25         "Michael")
26     writer.WriteString("last-name", _
27         "Avery")
28     writer.WriteEndElement()
29
30     writer.WriteString("price", _
31         "6.99")
32     writer.WriteEndElement()
33
34     writer.Flush
35
36     catch ex as Exception
37         Response.Write("Error accessing XML file")
38
39     finally
40         writer.Close
41         Response.Write("Finished processing")
42     end try
43 end sub
44 </script>
45
46 <html><body>
47
48 </body></html>

```

该清单的结构类似于 `XmlTextReader`。第 6 行声明了一个 `writer`，并将其他所有代码放在一个 `try` 语句块中。第 9 行实例化了 `writer`，第一个参数是目标 XML 文件的名称，第二个参数是编码方式（默认为 UTF-8）。如果目标 XML 文件不存在，ASP.NET 将自动创建该文件；否则使用已有的文件。

**警告：**以这种方式创建 XML 文件将覆盖已有的同名文件。这里，文件 `books2.xml` 的内容将丢失。所以一定要避免覆盖重要的数据！

第 12 行将 XML 声明标记（即 `<?XML version='1.0'?>`）写入到文件中。这是成型 XML 文档的一部分。第 13 行和第 14 行指定如何输出 XML。第 13 行告诉 ASP.NET 缩排文件，而第 14 行指定

缩进 3 个空格, 这样用户阅读文件时将更方便。Formatting 属性可设置为 Indented 或 None, 可以使用 IndentChar 属性设置缩排时使用的字符。

从第 15 行开始生成数据。您使用 WriteStartElement 和 WriteEndElement 方法来开始和结束元素标记。第 15 行生成<bookstore>标记, 第 32 行生成相应的结束标记</bookstore>。第 16 行生成开始标记<book>, 第 31 行生成相应的结束标记。

第 17 行和第 18 行创建<book>标记的属性: genre 和 style, 并设置相应的值。WriteElementString 给简单元素加上开始和结束标记。下面是第 20 行:

```
Writer.WriteElementString("title", "Vietnam")
```

其输出结果如下:

```
<title>Vietnam</title>
```

实际上, WriteElementString 生成第一个参数指定的开始和结束标记, 并插入第二个参数指定的字符串值。

关闭所有被打开的标记后, 第 34 行刷新了内容。这类似于刷新响应缓存区, 只是内容被写入到文件而不是浏览器中。最后, 第 38 行关闭了 writer。

注意: 通常, 调用 Close 方法将输出 XML。仅当需要再次将 writer 用于其他输出时, 才需要调用 flush 方法。

调用 Close 方法也将关闭所有打开的元素和属性标记, 但强烈推荐不要忘记手工关闭它们。

图 11.5 显示了输出的 XML 文件。



图 11.5 清单 11.5 输出的 XML 文件

### 11.2.3 验证 XML

XML 必须是成型的, 毕竟它被看作是一种通用的数据描述语言。如果 XML 不成型, 则实现通用性这个目标将非常困难。

假设两个公司要共享一个 XML 数据文件。不成型的文件提供了可解释的空间。如果两个公司

的解释方式不同，则后果将是灾难性的。

ASP.NET 让您能够根据模式或 DTD (Document Type Definition, 文档类型定义, 另一种描述 XML 数据类型的方法, 本书介绍它) 来验证 XML。为此, 需要创建一个新对象 `XmlValidatingReader`, 并指定要使用的验证类型:

```
create reader here

dim validator as new XmlValidatingReader(reader)
validator.ValidationType = ValidationType.DTD
    (if using DTD Validation)
或
validator.ValidationType = ValidationType.XDR
    (if using XDR Validation)
```

然后添加用于验证的事件处理程序, 该程序将使用 XML 文档名称空间指定的模式或 DTD。这里将使用模式验证。我们将这种声明加入到一个新的 XML 文件 `book2.xml` 中, 只需将 `bookstore` 改成如下所示即可:

```
<bookstore xmlns="x-schema:books_schema.xdr">
```

现在不要担心语法。该语句告知 ASP.NET, 您将使用 `books-schema.xml` 中描述的模式来验证文件。如果 XML 文件有效并遵循该模式, 则没有任何差别, 其输出结果同以前相同。

清单 11.6 列出了范例的开始部分 (其他代码将在清单 11.7 中列出)。

#### 清单 11.6 验证 XML

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>
3 <%@ Import Namespace="System.Xml.Schema" %>
4
5 <script runat="server">
6     private reader as XmlTextReader
7     private validator as XmlValidatingReader
8
9     sub Page_Load(obj as object, e as EventArgs)
10         try
11             reader = new XmlTextReader(Server.MapPath _
12                 ("books2.xml"))
13             validator = new XmlValidatingReader(reader)
14             validator.ValidationType = ValidationType.Schema
15
16             AddHandler validator.ValidationEventHandler, new _
17                 ValidationEventHandler(AddressOf ShowError)
18
19             While validator.Read()
20             End While
```

```

21     catch ex as Exception
22         Response.Write("Error accessing XML file")
23     finally
24         reader.close()
25     end try
26 end sub

```

该清单与其他清单有些类似。注意第 3 行导入的另一个名称空间 `System.XML.Schema`。您需要该名称空间来根据模式信息进行验证。验证时，首先创建一个 `XmlTextReader` 对象（第 6 行），并打开 XML 文件 `books.xml`（第 11 行）。接下来在第 13 行通过提供一个 `xmlTextReader` 对象来实例化 `xmlValidatingReader` 对象，以便验证 `Reader` 是否知道要验证哪个 XML 数据文件。第 14 行指出要使用 XML 文件中定义的 XML 模式来验证数据。

ASP.NET 中大多数内置的事件都有预定义的事件处理程序（例如，`Page` 对象的 `Load` 事件处理程序为 `Page_Load` 方法），但验证事件没有，因此必须使用 `AddHandler` 方法定义一个（如第 16–17 行所示）。第一个参数是要处理的事件，第二个参数必须指向事件处理程序。第 16 行的 `AddressOf` 运算符指向 `ShowError` 方法，当验证失败时，`ShowError` 方法将采取某种措施，可以给该方法取任何名称。您无需过多关心验证机制背后的细节，但掌握其语法对于以后使用将很有帮助。

另外，请注意第 19–20 行的 `While` 循环。通常，在这里可以读取和输出数据，但对于验证来说，并不需要输出数据，因为您只想读取数据，并对其进行验证，而不需要显示数据。

该 ASP.NET 页面余下的内容只有 `ShowError` 函数，如清单 11.7 所示。

#### 清单 11.7 ShowError 函数

```

1 ' sub ShowError(obj as object, e as ValidationEventArgs)
28     Response.write('<font color=" red">' & e.Message & _
29         '<br>')
30
31     if (reader.LineNumber > 0)
32         Response.Write("Line: " & reader.LineNumber & _
33             " Position: " & reader.LinePosition & _
34             "</font><p>")
35     end if
36 end sub
37 </script>
38
39 <html><body>
40
41 </body></html>

```

这里定义了 `ShowError` 函数，每当 XML 文件验证失败时，该函数便被调用。注意方法声明中的 `ValidationEventArgs` 参数，该参数含有一个 `Message` 属性，它包含对发生的错误的描述。因此，第 32 行以红色字体将该错误描述消息输出到浏览器中。第 32 行和 33 行使用 `XmlValidatingReader` 的属性 `LineNumber` 和 `LinePosition` 来显示 XML 文件中发生错误的位置。该方法对于跟踪问题很有用。

**注意：**您或许已注意到，在清单11.6中，XmlTextReader的声明放在Load事件的外面。这样可以在页面的任何方法中，而不仅仅是在Page\_Load事件中使用该对象，这也是可以在ShowError方法中使用该对象的原因所在。

如果在Page\_Load事件处理程序中声明该对象，则用ShowError方法访问该阅读器将出错。

如果books2.xml文件与books1.xml文件完全相同，则不会出现什么错误。然而，让我们做一些修改（如清单11.8），以便您可以看到验证过程。

#### 清单 11.8 Books.xml: 一个无效的XML文件

```
1 <?xml version="1.0"?>
2 <bookstore xmlns="x-schema:books-schema.xml">
3   <book genre="history">
4     <title>Vietnam</title>
5     <author>
6       <first-name>Michael</first-name>
7       <last-name>Avery</last-name>
8     </author>
9     <price>hello!</price>
10  </book>
11 </bookstore>
```

根据模式文件books-schema.xml，清单11.8中有两个错误，一是book元素中没有style属性，二是在price属性中使用了一个字符串。当在浏览器中查看验证代码（清单11.6和清单11.7）的输出时，将出现如图11.6所示的错误消息。

如果在清单11.6中使用while循环来输出数据，数据将和错误信息一起输出到浏览器中（如图11.6）。

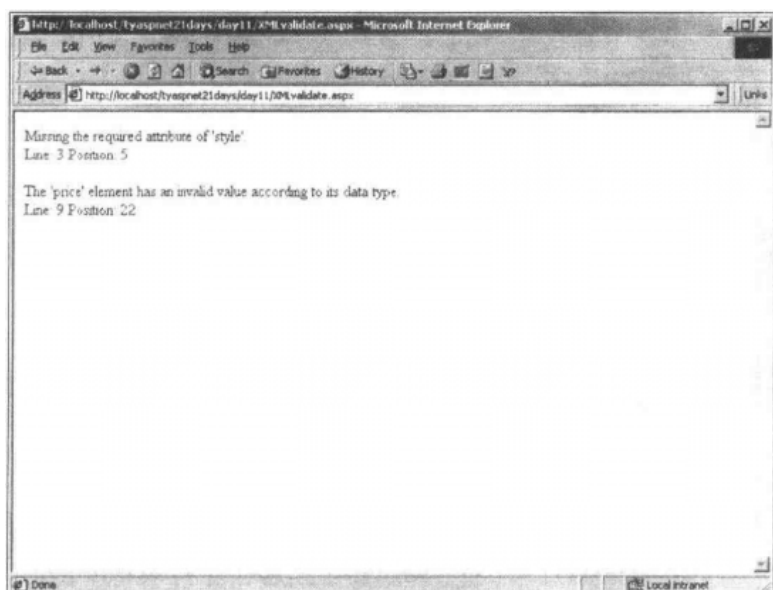


图11.6 验证中发现的错误

### 11.3 XML 文档对象模型

XML 文档对象模型 (XML Document Object Model, DOM) 是由 W3C 制定的一种规范, 它详细规定了访问 XML 的应用程序的行为, 包括这些应用程序应创建的类、如何读写 XML 以及类必须具备的特性。要查看该规范, 请访问 W3C 网站的下述网页:

- <http://www.w3.org/TR/REC-DOM-Level-1/>;
- <http://www.w3.org/TR/DOM-Level-2-Core/>;

至此, 还没有涉及到 XML DOM、XmlTextReader 和 XmlTextWriter 并不实现 DOM, 因为 DOM 带来的开销过大。上述两个对象用于实现快速、轻量级的 XML 存取。然而, 有时需要 DOM 的全部功能来编辑、导航和修改 XML 文件。

XmlNode 类提供了 XML DOM 描述的基本功能。它表示 XML 文档树的一个元素, 可用于浏览子节点和父节点以及编辑和删除数据。XmlDocument 类扩展了 XmlNode 类, 让您能够将 XML 文件作为一个整体进行操作, 如装载或保存文件。DOM 中还有很多从 XmlNode 类派生而来的其他类, 如 XElement 和 XMLAttribute, 但这里不介绍它们。图 11.7 详细描述了 XML DOM 类之间的关系。

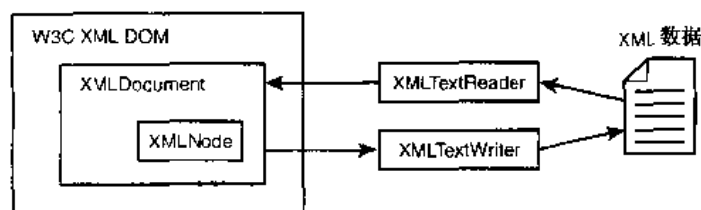


图 11.7 Microsoft XML 类之间的关系

XmlNode 表示 XML 文件的一个分支 (branch), 包括所有的属性、子元素以及开始和结束标记。例如, books.xml 中的 <title>...</title> 标记表示两个元素, 但只是一个节点。每个节点都可以有多个子节点, 其中每个子节点又表示一个分支。因此, 使用 DOM 与 XML 交互时, 查看数据的方式与传统数据存储器方式相同。

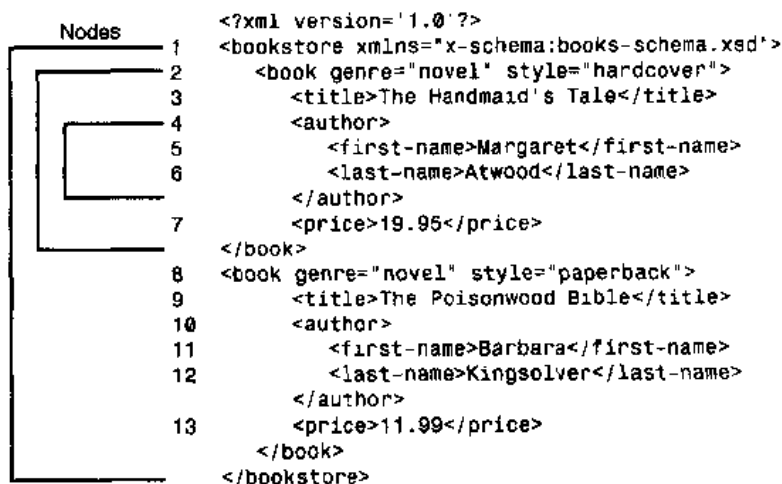


图 11.8 XML 文件中的节点

例如, 查看 Microsoft Access 中的数据时, 您不会将字段的开始和结束部分看作是两个不同的对象——它们属于同一个字段。XmlTextReader 和 XmlTextWriter 将每个对象看作是一个独立的实体, 但是 DOM 却将其看作一个字段。图 11.8 说明了 XML 文件中的节点概念。

### 11.3.1 装载 XML 数据

将数据装载到 XmlDocument 中的方法很多。最常用的两种方法是从 XmlTextReader 或直接从 XML 文件装载, 下面的代码片段说明了这两种方法:

```
'loading from an XmlTextReader
dim reader as new XmlTextReader(server.MapPath("books.xml"))
xmldocument.Load(reader)

'loading directly from a file
xmldocument.Load(server.MapPath("books.xml"))
```

既然可以直接装载数据, 为什么还要从 XmlTextReader 装载呢? 通常不需要使用 XmlDocument, 使用 XmlTextReader 就足够了。然而, 如果需要使用 XmlDocument 的功能, 则不必再次检索 XML 的数据, 而只需从 Reader 对象装载这些数据即可来装载数据。

数据被装载后, 便可以使用 XmlNode 对象来查看文件中的数据。例如, 可以遍历节点以显示数据, 就像使用 XmlTextReader 的 Read 方法遍历元素一样。清单 11.9 和清单 11.10 列出了一个例子:

**清单 11.9 使用 XmlDocument 打开文件**

```
1 <@ Page Language="VB" %>
2
3 / <@ Import Namespace="System.Xml" %>
4
5
6
7
8 <script runat="server">
9     private i as integer
10    private strOutput as string = ""
11
12    sub Page_Load(obj as object, e as EventArgs)
13        dim xmldoc as new XmlDocument()
14
15        try
16            xmldoc.Load(Server.MapPath("books.xml"))
17            ShowTree(xmldoc.DocumentElement)
18
19        catch ex as Exception
20            strOutput = "Error accessing XML file"
21        end try
22
23        output.Text = strOutput
24    end sub
```

**分析:** 该文件的第一部分很简单。第 9 行新建一个名为 xmldoc 的 XmlDocument, 第 12 行装载了 XML 文件中的数据。清单 11.10 中的 ShowTree 方法遍历所有的节点, 并显示其中的数据

XmlDocument 的 DocumentElement 属性返回文件的第一个元素（这里为 XML 版本声明）。您将该起始点提供给 ShowTree 方法，以便它能够遍历整个文件。

清单 11.10 使用 XmlNode 遍历 XML 文件

```

21 sub ShowTree(node as XmlNode)
22     Dim attrNode As XmlNode
23     Dim map As XmlNamedNodeMap
24
25     If Not node.HasChildNodes()
26         strOutput += "&nbsp;&nbsp;&nbsp;<b> " & node.Name & _
27             <b> </b>" & _
28             node.Value & "<br><br>" & vbCrLf
29     Else
30         strOutput += "<b> " & node.Name & "<br>"
31         If node.NodeType = XmlNodeType.Element Then
32             map = node.Attributes
33             For Each attrNode In map
34                 strOutput += " <b>" & attrNode.Name &
35                     "<b> </b>" & _
36                     attrNode.Value & "<br>" & vbCrLf
37             Next
38         End If
39         strOutput += "<br>"
40     End If
41
42     If node.HasChildNodes Then
43         node = node.FirstChild
44         While Not IsNothing(node)
45             ShowTree(node)
46             node = node.NextSibling
47         End While
48     End If
49 end sub
50 </script>
51
52 <html><body>
53     <asp:Label id="output" runat="server" />
54 </body></html>

```

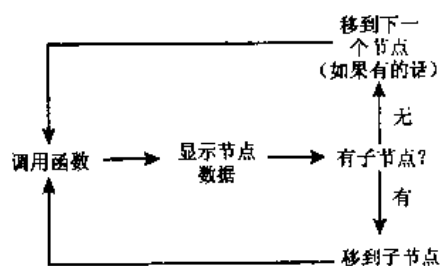
**新术语：**ShowTree 方法使用了一个叫作递归的编程概念。递归函数将不停地调用自己，直到满足某种条件为止，就像循环一样。这里，ShowTree 方法显示某一节点的信息，如果该节点拥有子



节点，对于每个子节点，该方法都调用自己。如果没有子节点，则转向下一个节点。这样，ShowTree 方法将遍历 XML 层次结构，显示所有子节点中的信息。图 11.9 说明了这种概念。

在第 25 行，如果节点没有子节点，则显示该节点名称和值。StrOutput 是一个字符串，用于收集输出。遍历结束后，将在第 53 行的标签（output）上显示该字符串。如果节点拥有子节点，并且属于元素类型，则显示该节点的所有属性。（在第 23 行声明，在第 32 行使用的）XMLNameNodeMap 对象表示每个节点的属性集合。然后，可以遍历该集合来读取数据（第 33–37 行）。

如果节点拥有子节点，则需要开始递归调用。这是由第 42–48 行的代码实现的。首先将当前节点的第一个子节点赋给节点变量，然后从该子节点开始调用 ShowTree，以显示与该节点相关的所有数据。如果该节点还有子节点，则取得其第一个子节点，然后重复上述过程。第 44 行的 while 循环和第 46 行中的 NextSibling 属性遍历每个节点的所有子节点。



调用次数	当前节点	子结点	移动
1	bookstore	2	bookstore child 1
2	book	3	book child 1
3	title	0	book child 2
4	author	2	author child 1
5	first-name	0	author child 2
6	last-name	0	book child 3
7	price	0	bookstore child 2
8	book	3	book child 1
9	title	0	book child 2

图 11.9 使用递归来遍历节点

让我们来想象打开圣诞节礼物的情景。您打开一个盒子，看看里面的东西，然后转向下一个盒子。然而，您的一个亲戚耍了一个把戏，送给您一个大盒子。打开盒子时，您看到里面有一个稍小的盒子，这个盒子里面还有一个更小的盒子，等等，直到最后找到礼物。整个过程是这样的：您打开一个礼物，如果里面没有更小的盒子，则转向下一个礼物。如果里面有小盒子，您需要打开所有的盒子，然后再去打开下一份礼物。递归过程与此类似——您访问所有的 XML 分支，如果有了分支，则转向下一分支前需要访问该分支的所有分支。

ShowTree 方法遍历整个 XML 文档，输出的结果如图 11.10 所示

应 该	不 应 该
当需要修改或创建 XML 数据时，应使用 XmlDocument 和 XmlNode	只是查看数据时，不要使用 XmlDocument 和 XmlNode 这两个对象的开销过大，可能会降低应用程序的性能

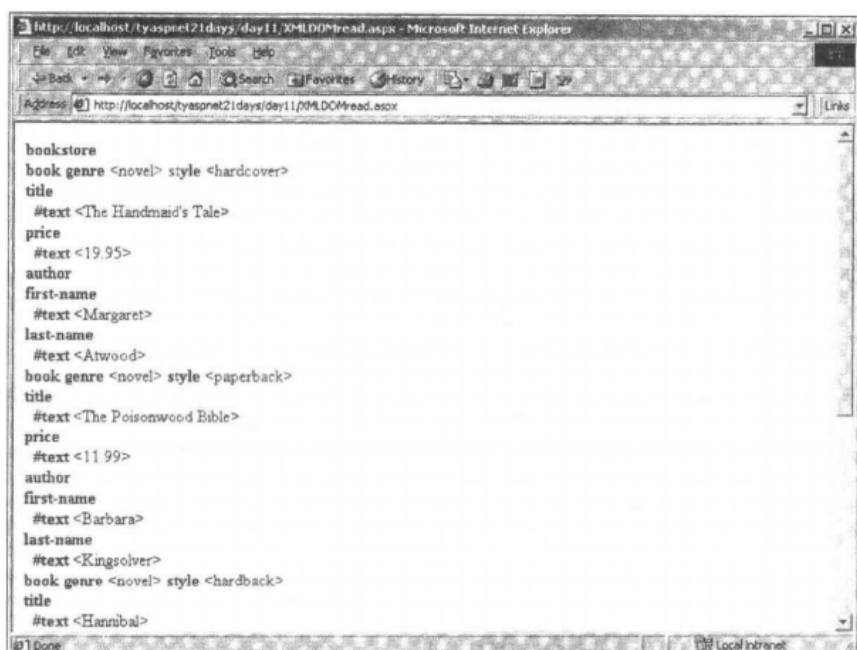


图 11.10 在浏览器中查看清单 11.9 和清单 11.10 得到的结果

### 11.3.2 修改 XML 数据

XmlDocument 对象还为创建和修改 XML 文档提供了很多方法。要添加新元素，可以使用 Create 方法系列——CreateComment、CreateAttribute、CreateNode 等。使用这些方法几乎可以创建任何类型的数据。例如：

```
'load data into xmldoc here
dim eleBook as XmlElement = xmldoc.CreateElement _
('Book')
dim attStyle as XmlAttribute = xmldoc.CreateAttribute _
('style')
eleBook.SetAttributeNode(attStyle)
eleBook.SetAttribute("style", "hardback")
dim root as XmlElement = xmldoc.Item('bookstore')
root.AppendChild(eleBook)
```

第 2 行和第 4 行分别创建了新的 XmlElement 和 XmlAttribute 对象。然后，使用 SetAttributeNode 方法给元素添加属性，并使用 SetAttribute 方法设置了添加的属性。最后，将新元素添加到 XML 文档的 book 元素中。

修改数据更简单，只需设置需要的值即可。例如：

```
if node.Name = "price" then
    node.Value = "8.99"
end if
```

仅此而已！如果您需要修改文件中的所有数据，只需使用递归来遍历每个元素即可。表 11.2 列出了一些用于创建或修改 XmlDocument 中的值的方法。

表 11.2 修改 XmlDocument 的方法

方 法	描 述
AppendChild	将指定的节点添加到当前节点的子节点列表的末尾
CreateAttribute	使用指定的名称创建一个 XmlAttribute
CreateCDataSection	使用指定的数据创建一个 Cdata 段
CreateComment	使用指定的数据创建一个 XmlComment
CreateElement	使用指定的名称创建一个 XmlElement
CreateNode	使用指定的类型、名称和名称空间 URI（确保命名方案的唯一性）的 XmlNode
CreateTextNode	使用指定的数据创建一个 XmlText
CreateXmlDeclaration	使用指定的版本、编码方式和字符串（指定属性是否是独立的）创建一个 XmlDeclaration 区；其中版本必须是 1.0
GetElementById	返回指定 id 对应的 XmlElement
GetElementsByTagName	返回一个 XmlNodeList 集合，其中包含的元素与指定的名称匹配
ImportNode	导入另一个文档中指定的节点；布尔型值指定是否导入所有子节点
InsertAfter	将第一个 XmlNode 插入到第二个 XmlNode 的后面
InsertBefore	将第一个 XmlNode 插入到第二个 XmlNode 的前面
PrependChild	将指定节点插入到当前节点的子节点列表的最前面
RemoveAll	删除当前节点的所有子节点和属性
RemoveChild	删除指定的子节点
ReplaceChild	用第一个 XmlNode 替换第二个 XmlNode
WriteContentTo	将当前节点的所有子节点保存到指定的 XmlWriter（如 XmlTextWriter）中
WriteTo	将当前节点保存到指定的 XmlWriter（如 XmlTextWriter）中

注意：其中的一些方法拥有重载版本，可使用不同的参数。更详细的信息，请参阅 .NET 框架 SDK 文档。

XmlElement 和 XmlNode 对象拥有很多与 XmlDocument 相同的方法。

使用 Save 方法将修改写入到文件中：

```
xmlDoc.Save("books2.xml");
```

提示：如果在已有 XML 文件中追加数据，只要将所有数据装载到 XmlDocument 中，然后使用该文件名保存修改后的数据。以前的文件被覆盖，但由于数据已装载到 XmlDocument 中，所以不会丢失任何数据。

清单 11.11 装载 XML 文件，并将其作为一个新的 book 节点追加到文件的最后面。

清单 11.11 使用 DOM 追加数据

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object, e as EventArgs)
6         dim xmlDoc as new XmlDocument()
7         dim strOutput as string=""
8         try
9             xmlDoc.Load(Server.MapPath("books.xml"))
10            dim eleBook as XmlElement = xmlDoc.CreateElement _
11                ("book")
12            dim attStyle as XmlAttribute = xmlDoc.CreateAttribute _
13                ("style")
14
15            eleBook.SetAttributeNode(attStyle)
16            eleBook.SetAttribute("style", "hardback")
17
18            dim root as XmlElement = xmlDoc.Item("bookstore")
19            root.AppendChild(eleBook)
20
21            xmlDoc.Save(Server.MapPath("books.xml"))
22
23        catch ex as Exception
24            strOutput = "Error accessing XML file"
25        end try
26
27        output.Text = "Append operation successful"
28    end sub
29 </script>
30
31 <html><body>
32     <asp:Label id="output" runat="server" />
33 </body></html>

```

第 1-9 行没什么特别的，只是创建了一个 `XmlDocument`，并使用 `books.xml` 的内容填充它。第 10 行创建了一个名为 `book` 的 `XmlElement`，第 12 创建了一个新的 `XmlAttribute` 属性，用于保存关于书籍的信息。注意，您使用已有 `XmlDocument` 的 `xmlDoc` 来创建这些条目，这确保新条目的格式与已有数据完全相同。

第 15-16 行声明了一个属性（该属性是 `book` 元素的一部分）并给它赋值。第 18 行检索根元素，以便将新的 `book` 元素追加到根元素中，如第 19 行所示。最后，使用 `Save` 方法将修改写入到 `books.xml`

文件中，所以不会丢失任何数据。

## 11.4 XML 和 DataSet

在 .NET 框架中，XML 与 ADO.NET 的关系非常密切。在 DataSet 内部，数据被表示为 XML，这意味着在计算机内存中，DataSet 是以 XML 格式（而不是一些抽象数据模型）存储的。所以查看 XML 数据的方式有两种：使用 XML 类直接查看或间接地通过 ADO.NET 进行查看。DataSet 只是提供的视图不同于 XML。

让我们来看看当前的情况。一方面，您拥有 ADO.NET 及其对象。一些简单对象（如 OleDbDataReader）提供了快速、简单的数据存取方式。一些复杂的对象（如 DataSet）包含关系信息，提供的功能比 OleDbDataReader 多。

另一方面，您还拥有 XML .NET 框架，它也包含一些简单对象和复杂对象。XmlTextReader 为读取 XML 数据提供了简单、轻量级的途径，而 XmlDocument 提供了更多的功能。然而，后者并不能很好地表示关系型数据，所以引入了 XmlDataDocument 对象。

XmlDataDocument 之于 XML 就像 DataSet 之于 ADO.NET。这两个对象非常相似，可以很容易地相互转换。从某种意义上说，这两个对象是 ADO.NET 和 XML 之间的桥梁。

XmlDataDocument 类似于 XmlDocument，但能够表示关系型数据，与 DataSet 类似。可在使用 XmlDocument 的任何地方使用 XmlDataDocument，实际上，参数和属性也相同。

数据被装载到 XmlDataDocument 中时，.NET 自动为您创建一个 DataSet 对象，可通过 DataSet 属性来访问该对象。XML 模式被用于创建 DataSet 中的字段和数据类型。如果未提供 XML 模式，则 ASP.NET 将自动推断其结构。

这让您能够按自己的意愿修改数据。例如，可以使用 XML 对象打开 XML 文件，然后将 XML 对象移到 DataSet 中，以绑定服务器控件。或者使用 DataSet 检索数据库的数据，然后将其保存到 XML 文件中。对 DataSet 所做的任何修改都将在 XmlDataDocument 中体现出来，但 XmlDataDocument 中的修改则不一定会在 DataSet 中体现出来。如果新添加的数据符合 DataSet 中的字段，则将添加新的记录。

图 11.11 说明了这两个对象之间的关系。

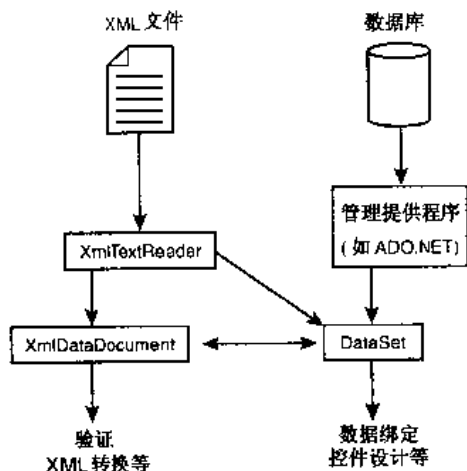


图 11.11 DataSet 和 XmlDataDocument 之间的关系

```

1 <%@ Page Language="VB" %>
2 <% Import Namespace='System.Xml' %>
3 <% Import Namespace='System.Data' %>
4 <% Import Namespace='System.Data.OleDb' %>
5
6 <script runat=server>
7     private i, j as integer
8     private strOutput as string = ""
9
10    sub Page_Load(obj as object, e as eventargs)
11        dim xmldoc as new XmlDocument()
12
13    try
14        xmldoc.DataSet.ReadXml(Server.MapPath("books.xml"))
15
16        select data view and bind to server control
17        DataGrid1.DataSource = xmldoc.DataSet
18        DataGrid1.DataMember = xmldoc.DataSet.Tables(0). _
19            TableName
20        DataGrid2.DataSource = xmldoc.DataSet
21        DataGrid2.DataMember = xmldoc.DataSet.Tables(1). _
22            TableName
23
24        DataGrid1.DataBind()
25        DataGrid2.DataBind()
26
27        For i = 0 To xmldoc.DataSet.Tables.Count - 1
28            strOutput += "TableName = '" & _
29                xmldoc.DataSet.Tables(i).TableName & "'<br>"
30            strOutput += "    " & "Columns count ' & _
31                xmldoc.DataSet.Tables(i).Columns.Count. _
32                ToString() & '<br>'
33
34            For j = 0 To xmldoc.DataSet.Tables(i).
35                Columns.Count-1
```

```

36         strOutput += " &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;" & _
37             'ColumnName = "' & xmlDoc.DataSet.Tables(i).Columns(j).ColumnName & '"', _
38             type = " & xmlDoc.DataSet.Tables(i).Columns(j).DataType.ToString() & "<br>"
39         Next
40     Next
41     strOutput += "<p>"
42 
43     catch ex as Exception
44         strOutput = "Error accessing XML file"
45     end try
46 
47     output.Text = strOutput
48 end sub
49 
50 </script>
51 <html><body>
52 <asp:Label id="output" runat="server" />
53 
54 <asp:DataGrid id="DataGrid1" runat="server"
55     BorderColor='black'
56     GridLines='Vertical'
57     cellpadding="4"
58     cellspacing="0"
59     width='450'
60     Font-Name="Arial"
61     Font-Size="8pt"
62     HeaderStyle-BackColor="#cccc99"
63     FooterStyle-BackColor="#cccc99"
64     ItemStyle-BackColor="#ffffff"
65     AlternatingItemStyle-BackColor="#cccccc" />
66 
67 <p>
68 <asp:DataGrid id="DataGrid2" runat="server"
69     BorderColor="black"
70     GridLines='Vertical'
71     cellpadding="4"
72     cellspacing="0"
73     width="450"
74     Font-Name="Arial"

```

```

76      Font-Size="8pt"
77      HeaderStyle-BackColor="#cccc99"
78      FooterStyle-BackColor="#cccc99"
79      ItemStyle-BackColor="#ffffff"
80      AlternatingItemStyle-BackColor="#cccccc" />
81 </body></html>

```

分析：该清单首先创建了一个 `XmlDataDocument` 对象，这里没有直接装载数据，而是使用 `DataSet` 属性的 `ReadXml` 方法来读入数据，如第 14 行所示。该方法自动创建了一个关系型数据视图，稍后您将看到。

我们暂时略过第 27 行。接下来遍历 `DataSet` 中的表、输出每个表的名称和字段数（第 27-32 行）第 34-41 行的第二个 `for` 循环输出在 `DataSet` 中各字段的名称及其数据类型。`DataSet` 知道每个字段表示的数据类型，因为它能够从数据结构推断出数据模式。这两个循环将以关系型方式显示 XML 数据。

但由于您使用的是 `DataSet`，因此有一种更简单的方式来完成这项工作。第 17-22 行将数据绑定到页面中定义的两个不同的 `DataGrid` 控件（稍后将讲说明为何使用两个 `DataGrid` 控件）。

等等！图中有两个表，但是只有一个 XML 文件，这是怎么回事？

.NET 框架读取 XML 模式，知道数据可以表示成关系型。具体地说，将 `author`（作者）信息分离出来，并将其放到另一个表中，并且自动创建一个外关键字来连接这两个表！在两个 `DataGrid` 中，数据以更为传统的方式表示。

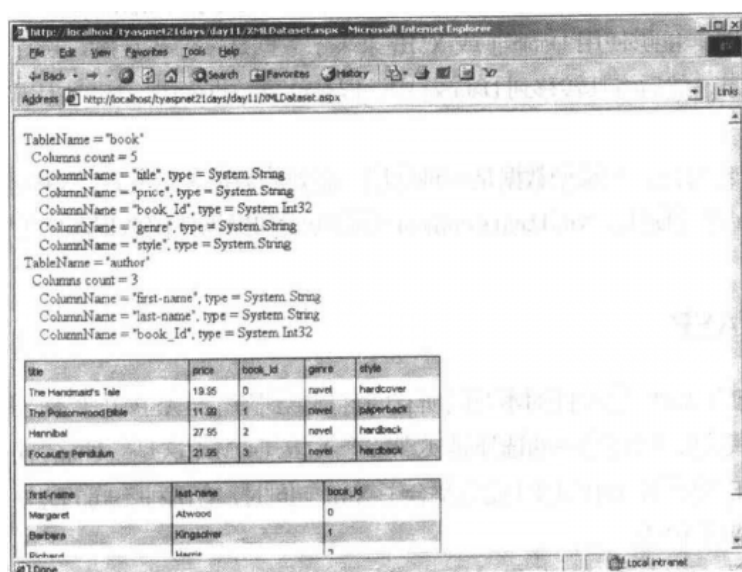


图 11.12 使用 `XmlDataDocument` 和 `DataSet` 来查看关系型 XML 数据

没有模式，`DataSet` 如何确定数据结构呢？方法很简单：

- 将所有包含属性的元素转换为表；
- 将所有包含其他元素的元素转换为表；
- 如果两个或多个元素的名字相同，则将这两个元素转换为一个表；
- 将根节点的所有直系子节点转换为表。



其他元素则转换为字段。然后，XML 文件中与字段匹配的数据将作为一条记录添加到 DataSet 中。

这是真正的关系型吗？当您在 XML 文件中添加一个与现在节点的作者名相同的 book 节点时，将发生什么情况？图 11.13 使用 DataGrid 说明了这种情况。

genre	style	title	book_id	price
novel	hardcover	The Handmaid's Tale	0	19.95
novel	paperback	The Poisonwood Bible	1	11.99
novel	paperback	The Poisonwood Book	2	13.99

first-name	last-name	book_id
Margaret	Atwood	0
Barbara	Kingsolver	1
Barbara	Kingsolver	2

图 11.13 添加关系型数据

糟了，情况看来不妙。ASP.NET 没有发现已经有了一个名为 Barbara Kingsolver 的作者，而将其作为一个新条目添加到作者表中。试图将字段 first name 和 last name 作为主关键字将导致错误。不幸的是，没有简单的方法来解决这个问题，您只有手工更新外关键字并删除 author 表中的多余记录。

尽管有些局限性，能够使用 DataSet 或 XML 来表示数据是一种非常强大的特性。可以用 XML 格式创建关系型信息，然后将其转移到 DataSet 中，以存储到数据库中，还可以以 XML 文件来传输数据。

正如前几章介绍过的，关系型数据是一种常用、高效的数据表示方式，XML 在关系型领域也不落后。同 DataSet 搭配使用，XmlDataDocument 可以访问和操纵任何数据源中的关系型数据。

## 11.5 这不是 ASP

如果熟悉传统的 ASP，您可能对本章的所有内容都感到陌生。ASP 几乎不支持 XML，没有内置对象来处理这种数据，因此唯一的选择是实现整个 XML DOM（这是一种痛苦的选择）。

幸运的是，.NET 框架（ASP.NET）完全整合了 XML。同时，有两种不同的方法来同 XML 交互：ADO.NET 或 XML.NET 模式。

不幸的是，这种新的体系结构与任何以前访问 XML 的方法的共同之处很少，也就是说其中的很多方法都是新的，需要花费很多时间去学习。幸运的是，通过提供定义良好的类和对象（给出了统一的编程模型），ASP.NET 使这一切变得很容易。

## 11.6 总 结

本章介绍了在 ASP.NET 中访问 XML 的知识。由于 XML 已同 .NET 框架密切地集成在一起，所

以在 ASP.NET 页面中访问大部分 XML 数据很容易。

XmlTextReader 用于便捷、快速地访问 XML 数据源。使用该对象可以访问原始数据，也就是说每一个对象（例如开始和结束标记）都将被单独处理。Read 方法可用于遍历元素。

XmlTextWriter 对象类似于 XmlTextReader，只是它被用于写 XML 文档。它使用一系列 Write 方法来创建 XML 元素。

XmlValidatingReader 提供了验证功能，可以使用该对象根据任何模式来验证 XML 源，并添加验证失败时执行的方法。

XmlDocument 和 XmlNode 对象可以访问 XML，并完全支持 DOM。这意味着 XML 文件更被看作是传统的数据模型，而不是简单的文本文件。可使用 XmlDocument.Load 方法来装载数据，并使用 XmlNode 对象和表 11.2 列出的函数来操纵数据。

XML.NET 框架还通过 XmlDataDocument 对象来支持关系型 XML 数据。该对象类似于 ADO.NET 中的 DataSet 对象。这让您能够使用 ASP.NET 的数据绑定功能，为 XML 数据提供 UI 支持。下一章将介绍 ASP.NET 中的文件输入/输出。读写服务器文件的功能对 ASP.NET 有极大的帮助，这为配置和存储数据提供了额外的信息。

## 11.7 问与答

问：哪种访问 XML 的方法更好？通过 ADO.NET 还是通过 .NET 框架的 XML 类？

答：该问题的答案有很大的主观性。很多人认为 ADO.NET 的对象易于操作，其他人则喜欢直接操纵 XML。幸运的是，您可以使用任何一种方式。如果开始时使用的是某种方式，可以很容易地转向另一种方式。两种方式的性能没什么差别。

## 11.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 11.8.1 小测验

1. 什么时候应该使用 XmlTextReader，而不是 XmlDocument？
2. XML DOM 代表的是什么？
3. 节点和元素间有什么差别？
4. 下面的代码片段能正常运行吗？

```
Dim xmldoc as new XmlDocument()
xmldoc.DataSet.ReadXml("book.xml")
```

### 11.8.2 练习

1. 创建一个使用 books-chema.xml 模式的 XML 文件。使用 ASP.NET 来验证它，以确保格式是正确的。
2. 创建一个界面，让用户通过 DataGrid 来编辑 XML 数据。请务必将修改后的数据写入到文件中。

## 第 12 章

# 应用高级数据技术

在很短的时间内，您便学习了各种数据访问方法，您已打下了坚实的基础，可以学习更高级的数据技术了。本章重点介绍比以前更复杂的技术，包括新的数据库和 XML I/O 技术，这些技术将帮助您开发出大师级的应用程序。

首先介绍几种使用参数和存储过程（Stored procedure）来检索数据的新方法。参数让您能够以更有效的方式创建查询。存储过程是预先创建的 SQL 语句，它为应用程序提供了很多好处，包括速度和可读性。您还将学习如何使用事务来确保 SQL 命令被正确地执行。

接下来，您将进一步学习 XML。本章将介绍 XmlNavigator 对象，该对象类似于 XmlDocument 对象，用来读取 XML 文件。使用该对象，便可以使用 Xpath 查询和 XSL 转换（transform），前者是 XML 中使用的查询语言，而后者可用于将 XML 文件转换为其他类型的结构化文档，如 HTML 文件。要成为 ASP.NET 专家，必须学会使用多种方法来完成同一项工作。阅读本章后，您将知道几种访问数据库和 XML 文件中的数据的方法。

本章包括以下内容：

- 什么是参数化查询（parameterized query），如何使用这种查询；
- 什么是存储过程，如何使用它；
- 如何使用事务；
- 另一种读取 XML 文档的方法；
- 如何查询 XML 文件；
- 如何转换 XML 文件。

### 12.1 高级数据库技术

到目前为止，您已学会使用 OleDbDataAdapter、OleDbCommand 和 DataSet 对象（还有其他几个辅助对象）来处理数据库需求。这些对象提供应用程序需要的所有功能，但它们还提供了一些前面没有介绍过的东西，包括参数、存储过程和事务。

参数是一种新的、创建动态 SQL 语句的方法。可以使用参数告知数据库您需要哪些数据，而不用使用各种信息源来组装 SQL 语句，这种方法更简单、更精致。存储过程是预先创建好的 SQL 语句，可提高应用程序的性能。同参数搭配使用时，存储过程将是一种非常优秀的数据库查询方法。通过采用一种要么全部要么都不的数据库操纵模式——要么所有修改都生效，要么撤销所有的修改

——事务让您能够防止数据遭到破坏。您将学习几种常见的、需要使用事务的情形。

这些高级的数据库技术将帮助您创建更强大的应用程序，并提高这些程序的性能。

### 12.1.1 参数化查询

假设您要盖房。您需要清楚有关材料规格及其被用于什么地方。一种方法是将这些信息写在所有材料上。每块材料上都标上其大小和使用位置，窗户、门框和水管等也如此。该方法行得通，但不准确，容易出问题。相反，您使用图纸指明材料的位置以及如何将其组装起来。

数据访问与此类似。创建 SQL 语句来访问数据时，可以取出查询语句的各个部分，并将它们连接起来，同时标出各语句的功能。或者应用一种组织性更强的方法来确定各种信息的去向。第二种方法使用参数（独立于 SQL 查询的信息）来提供数据。数据库应用参数的方式同建筑师使用图纸相同。

我们在 OleDbCommand 对象中使用参数来指定其他的信息，如要返回的数据或如何将数据插入到 DataSet 中。第 10 章介绍过，要创建动态查询，通常需要使用 ASP.NET 页面中各种 Web 控件中的数据来创建语句。例如，对于下面的 SQL 语句：

```
strSQL = "select * from tblUsers where UserID = 1"
```

UserID 的值可能来自页面的文本框。您可以使用下面的语句，其中 tid 是文本框的名称：

```
strSQL = "select * from tblUsers where UserID = " & tid.Text"
```

这样便创建了一个动态查询，但组织性不好。如果有三个不同的文本框，情况将如何呢？在这种情况下，很难知道各文本框包含的信息，尤其当其他开发人员阅读您的代码。

**新术语：**一个更有效的方法是使用参数。参数是传递给查询或由查询返回的值。使用参数有助于使信息更为直观，提供查询的可读性。我们用一个带参数的查询替换前面的查询：

```
strSQL = "select * from tblUsers where UserID = @ID"
```

您使用一个查询参数替换了字符串中的动态部分，参数以 @ 表示，它是 SQL 语句的一部分。现在需要在页面的某个位置为该参数提供一个值。您可以使用 OleDbCommand 对象的 Parameters 集合来指定该参数（关于 OleDbCommand 的更详细的信息，请参阅第 10 章）。清单 12.1 列出了一个例子。

**清单 12.1 为 SQL 语句指定参数——部分代码**

```
1: dim objCmd as OleDbCommand = new OleDbCommand _
2:    ('select * from tblUsers where UserID = @ID', Conn)
3:
4: dim objParam as OleDbParameter
5: objParam = objCmd.Parameters.Add('@ID', OleDbType.Integer)
6: objParam.Direction = ParameterDirection.Input
7: objParam.Value = tid.Text
```

第 1-2 行像通常那样创建了一个 OleDbCommand 对象（假设您已创建了一个名为 Conn 的 OleDbConnection 对象）。注意第 2 行的参数化查询语句。第 4 行中创建了一个 OleDbParameter 对象，用于传入参数值。第 5 行将该参数添加到 OleDbCommand 对象中，并指定名称和类型。名称必须同查询中的参数（这里为 @ID）匹配。类型是一个 OleDbType 值，表示传入值的数据类型。表 12.1 列出了最常用的值。

表 12.1

常用的 OleType 值

类 型	描 述
Binary	字节流（对应于一个字节数组）
Boolean	布尔值
BSTR	字符串（对应于 String）
Char	字符串（对应于 String）
Currency	货币值（对应于 Decimal）
Date	日期（对应于 DateTime）
Decimal	Decimal 值
Double	Double 值
Empty	空值
Error	32 位错误代码（对应于 Exception）
Integer	32 位整数（对应于 Integer）
LongVarChar	长字符串（对应于 String）
VarChar	字符串值（对应于 String）
Variant	一种特殊的数据类型，在没有指定数据类型时，可用于表示任何类型的数据（对应于 Object）

第 6 行指定了参数的传输方向。由于该参数将用于 select 查询，而且是在传入值，所以方向为 Input。如果您希望返回一个值，并将其存储在参数中，则方向为 Output。本章后面关于存储过程的一节将更详细地介绍参数的传递方向。

最后，第 7 行将参数的值设置为文本框中的文本（txtId）。文本框中的内容将作为参数传递给 Select 语句。让我们来看一个完整的例子，如清单 12.2 所示。

#### 清单 12.2 使用参数返回值

```

1:  <%@ Page Language="VB" %>
2:  <%@ Import Namespace="System.Data" %>
3:  <%@ Import Namespace="System.Data.OleDb" %>
4:
5:  <script runat="server">
6:      dim Conn as new OleDbConnection("Provider=' & _
7:          "Microsoft.Jet.OLEDB.4.0;" & _
8:          "Data Source=C:\ASPNET\data\banking.mdb")
9:
10:      sub GetData(obj as Object, e as EventArgs)
11:          dim objCmd as OleDbCommand = new OleDbCommand _
12:              ("select * from tblUsers where UserID = @ID", Conn)

```

```

13:     dim objReader as OleDbDataReader
14:     dim objParam as OleDbParameter
15:
16:     objParam = objCmd.Parameters.Add("@ID", _
17:         OleDbType.Integer)
18:     objParam.Direction = ParameterDirection.Input
19:     objParam.Value = tbID.Text
20:
21:     try
22:         objCmd.Connection.Open()
23:         objReader = objCmd.ExecuteReader
24:     catch ex as OleDbException
25:         Label1.Text = "Error retrieving from the database."
26:     end try
27:
28:     DataGrid1.DataSource = objReader
29:     DataGrid1.DataBind()
30:
31:     objReader.Close
32:     objCmd.Connection.Close()
33: end sub
34: </script>
35:
36: <html><body>
37:     <form runat='server'>
38:         <asp:Label id="Label1" runat='server' /><br>
39:         Enter an ID: <asp:TextBox id="tbID" runat="server"
40:             AutoPostBack=True
41:             OnTextChanged=GetData /><p>
42:         <asp:DataGrid id="DataGrid1" runat="server"
43:             BorderColor="black" GridLines="Vertical"
44:             cellpadding=4" cellspacing="0" width="100%"
45:             Font-Name="Arial" Font-Size="8pt"
46:             HeaderStyle-BackColor="#cccc99"
47:             ItemStyle-BackColor="#ffffff"
48:             AlternatingItemStyle-BackColor="#cccccc"
49:             AutoGenerateColumns="true" />
50:     </form>
51: </body></html>

```

**分析:** 您应该熟悉该清单, 它与第10章开发的数据库范例很相似。HTML 部分创建了三个服

服务器控件：一个 DataGrid（第 42-49 行）、一个标签（第 38 行）和一个 TextBox（第 38-41 行）。用户在文本框中输入值后，TextEvent 事件将被激发，将过滤出的数据显示在 DataGrid 中（注意第 40 行的 AutoPostBack=True）。第 6 行声明了一个 OleDbConnection 对象，供页面方法使用。

文本框的 TextChange 事件被激发时，将执行 GetData 方法，以显示合适的数据。参数化查询位于第 11-12 行。第 16-18 行创建了参数，并设置了其方向，并将其值设置为 txtID 文本框中的文本。该方法的其他代码检索数据，并将其显示在 DataGrid 中。在文本框中输入值后，得到的结果如图 12.1 所示。

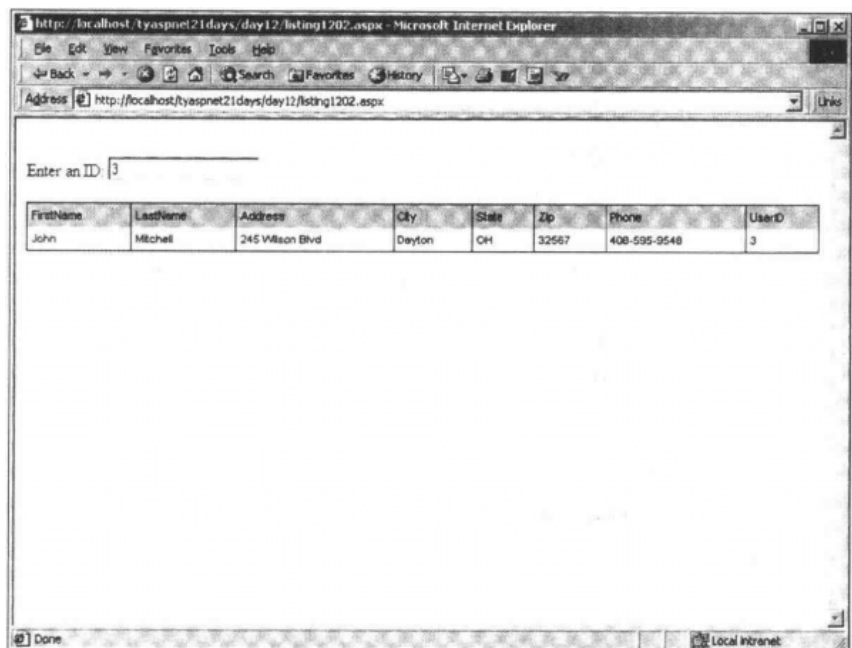


图 12.1 参数化查询使用文本框中的值来返回数据

一个查询可使用多个参数。例如：

```
strSQL = "SELECT * FROM tblUsers WHERE UserID=@ID AND  
        FirstName=@Name"
```

甚至可以将返回的值放到参数中：

```
strSQL = "SELECT @Phone=Phone FROM tblUsers WHERE UserID=@ID  
        AND FirstName=@Name"
```

返回信息的参数称为输出参数（output parameters）。执行上述查询时，语句 `SELECT Phone FROM tblUsers WHERE UserID=@ID AND FirstName=@Name` 返回的值（应该是一个电话号码）将被存储到参数 `@phone` 中。查询执行后，可使用参数集合来获取 `@phone` 的值。例如，下面的代码片段为前面的 SQL 语句创建一个输出参数：

```
dim objParam as OleDbParameter  
objParam = objCmd.Parameters.Add("@Phone", OleDbType.BSTR)  
objParam.Direction = ParameterDirection.Output
```

查询执行后，可访问参数的 Value 属性来获得这个值：

```
dim strPhone as string = objParam.Value
```

参数在创建动态查询方面很有用，但只有与存储过程搭配使用时，参数的威力才能显示出来。

### 12.1.2 存储过程

存储过程是数据库能够执行的命令集（通常为结合了其他 DB 专用语言的 SQL 语句）。存储过程和普通的 SQL 语句间有什么差别呢？

首先，存储过程是编译后的。您知道编译后的 ASP.NET 代码给页面带来的好处。编译后的存储过程具有类似的优点，包括运行速度更高。

**新术语：**在数据库中执行存储过程时，将制定一个执行方案（execution plan），使得随后执行时，数据库检索信息的速度更快。数据库查看数据和查询，并确定最有效的、返回数据的方法，然后将该方式保存为执行方案。因此，存储过程的好处不仅来自其被编译过，还来自执行方案的协助。

创建存储过程还在 ASP.NET 页面和数据之间建立了另一个抽象层。牢记，模块化是面向对象编程的目标之一。将 SQL 语句与 ASP.NET 页面分开，实现了以下三个目标：

- 可以重用查询；
- 代码的可读性更强；
- 可节省时间。

如果 SQL 语句很长，将其放在 ASP.NET 页面中将导致其中包含一些不用放在这里的代码，并使真正起作用的代码很零乱。另外，由于这些语句必须从 ASP.NET 页面发送到数据库，所以会占用宝贵的带宽。最后假设在几个不同的 ASP.NET 页面中使用了同一条 SQL 语句。如果由于某种原因，数据库发生了变化，导致需要修改该 SQL 语句，在这种情况下，必须修改每一个页面的 SQL 语句。而使用存储过程，只需要修改一次，这种修改将反映到整个应用程序中。

将 SQL 语句转换为存储过程，不仅解决了以上所有这些问题，还提供了其他好处。即使是一行 SQL 语句也将通过转换到存储过程而受益。

接下来我们创建一些存储过程！

#### 1. 在 SQL Server 2000 中创建存储过程

存储过程被用于许多不同的数据库系统中。例如，SQL Server 2000 和 Access 都使用存储过程，尽管两者创建存储过程的方法不同。本节将介绍如何在 SQL Server 2000 中创建存储过程，接下来的一节介绍如何在 Access 2000 中创建存储过程。

为此，我们将使用 SQL Server，尽管本书一直使用 Access。让我们试着创建一个简单的存储过程。

按第 8 章介绍的那样打开 Enterprise Manager。展开 Microsoft SQL Server 和 SQL Server Group 节点，同时展开指示服务器名称和数据库名称的节点。单击“+”，打开第 8 章创建的 Banking 数据库，如图 12.2 所示。

SQL Server 包含大量内置的存储过程。单击 Stored Procedure 节点，双击右边窗口中的某个名称，快速查看一下这些内置的存储过程。这里大部分的存储过程都比您将创建的存储过程要复杂得多，但它有助于您对存储过程有个整体的印象。在存储过程的图标上单击鼠标右键，然后选择 New Stored Procedure，打开如图 12.3 所示的对话框。

用存储过程的名称（如 SelectIDFromName）替换[OWNER].[PROCEDURE NAME]。（现在不要管 OWNER 属性。更详细的信息，请参阅 SQL Server 2000 在线文档）。由于要使用参数化查询，所以还必须在名称和 AS 关键字之间指定参数：



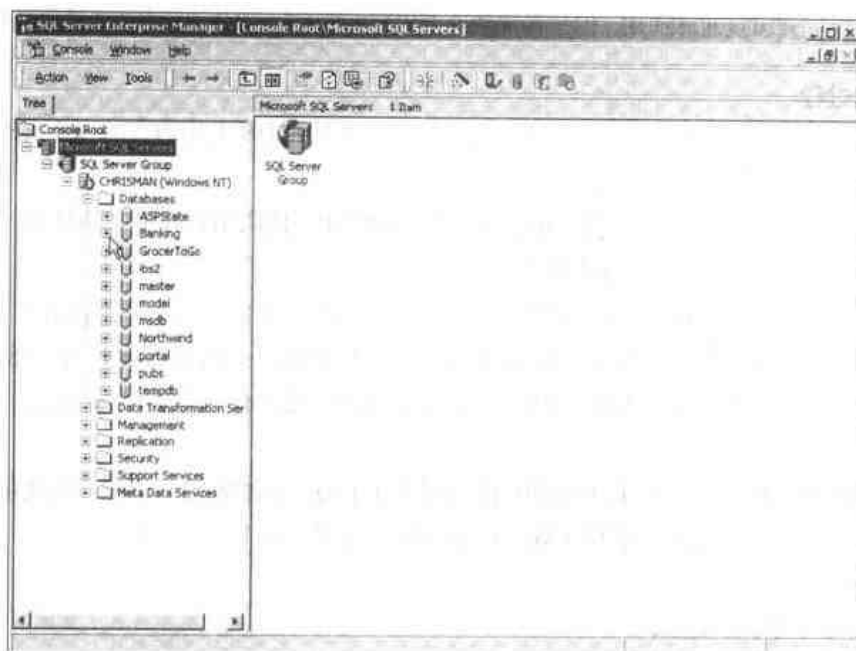


图 12.2 展开第 8 章创建的 Banking 数据库



图 12.3 输入存储过程使用的 SQL 命令

```
CREATE PROCEDURE selectIDFromName  
    @FirstName varchar ,  
    @LastName varchar ,  
    @ID int OUTPUT  
AS
```

关键字 OUTPUT 告知 SQL Server，您想在该参数中存储一个值，并将该值返回给调用存储过程的程序。在 AS 关键字后，输入 SQL 语句：

```
SELECT @ID = UserID  
FROM tblUsers  
WHERE FirstName = @FirstName  
    AND LastName = @LastName
```

SQL 语句检索 First Name 和 Last Name 字段与您输入的值相同的记录，并将该记录的 UserID 值赋给 @ID。后面可以将其作为一个输出参数。您也可以单击 Check Syntax 按钮来确保创建的存储过程的语法是正确的。SQL Server 将检查代码，如果有错误，则指出。单击 OK 按钮，保存该存储过程。现在可以在列表中看到新创建的存储过程，表中还有其他的存储过程。

## 2. 在 Access 2000 中创建存储过程

由于本书一直使用 Access，所以您还应该学习如何在 Access 中创建存储过程。Access 支持存储过程，但将其称为“查询”。我们打开第 8 章创建的 Banking 数据库，单击左边的“Queries”图标，如图 12.4 所示。

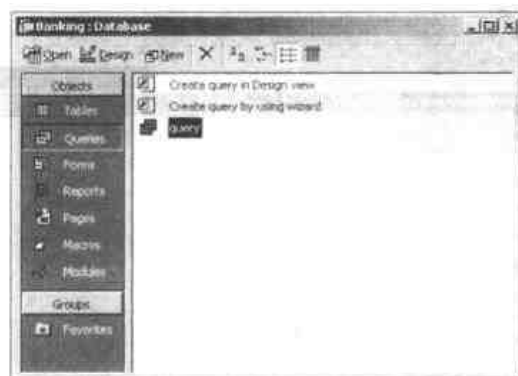


图 12.4 在 Access 中，存储过程被称为查询

由于您已熟悉 SQL 语句，因此没有必要使用向导。单击 Create query in Design View，出现一个标题为“Show Table”的对话框，询问要包含哪些表。单击 Close，关闭该对话框，然后将目光转向 Access 菜单的左上方；其中有一个 SQL 菜单。单击该菜单，然后选择“SQL View”，如图 12.5 所示，这样便可以将 SQL 语句直接输入到查询中。

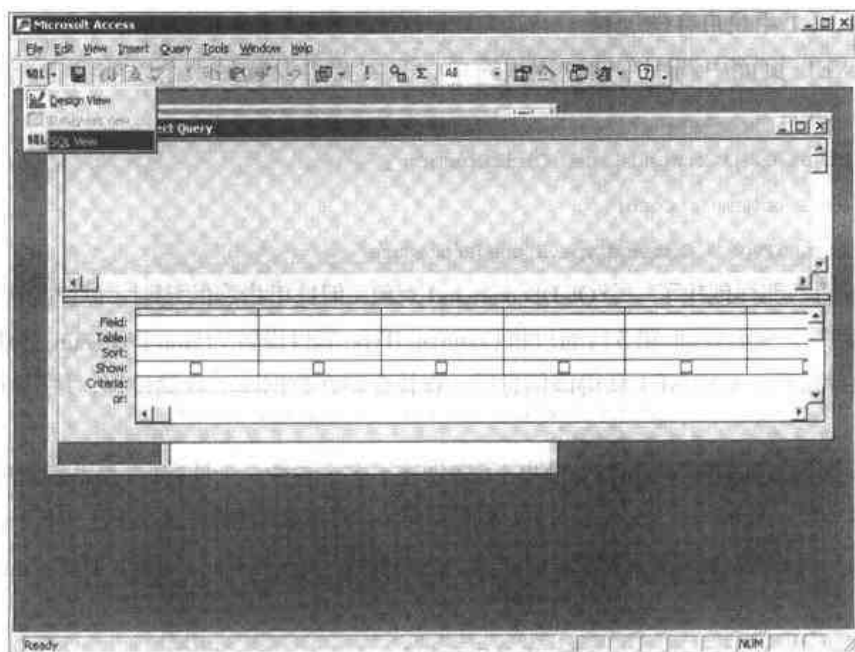


图 12.5 单击 Access 左上方的下拉菜单，进入 SQL 模式

```
SELECT UserID FROM tblUsers
WHERE FirstName = @FirstName
AND LastName = @LastName
```

在窗口中输入 SQL 语句，如图 12.6 所示。单击右上方的“x”按钮，关闭该窗口，当系统提示是否保存时，单击 Yes。将该查询保存为 SelectIDFromName，现在它应该出现在查询列表中。

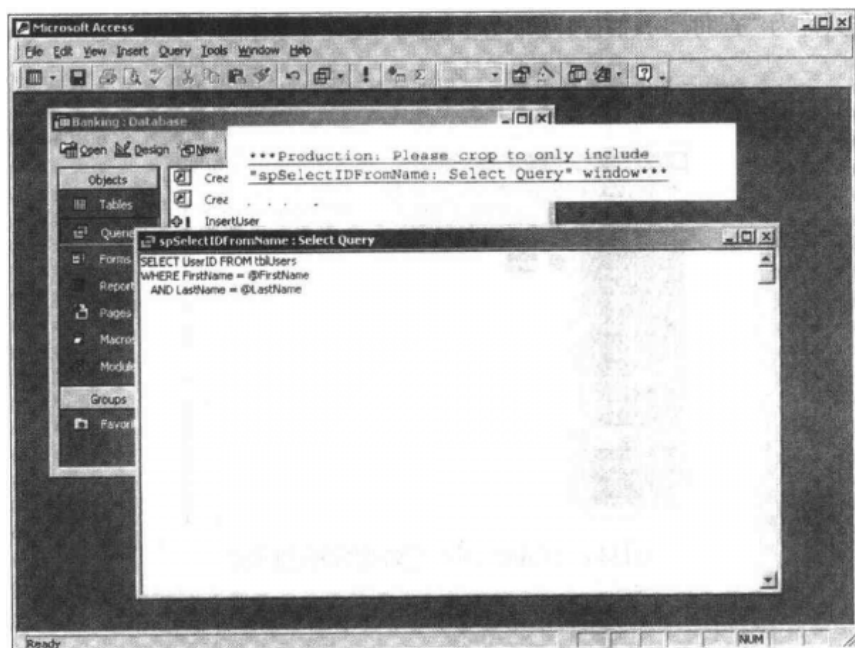


图 12.6 在 Access 的 SQL 视图输入查询

**注意：**Access 数据库引擎不支持在查询中使用输出参数，这就是查询中没有 @ID 参数的原因。后面将介绍如何解决这种问题。

### 3. 在 ASP.NET 中使用存储过程

在 ASP.NET 页面中调用存储过程很简单，只需要再设置另一个以前没有介绍过的属性 `CommandType` 即可：

```
dim objCmd as OleDbCommand = new OleDbCommand _
    ('SelectIDFromName', Conn)

objCmd.CommandType = CommandType.StoredProcedure
```

第 1 行像通常那样创建了一个 `OleDbCommand` 对象。但这里指定的是刚才创建的存储过程的名称，而不是指定一个 SQL 查询。第 3 行通过将 `CommandType` 属性设置为 `StoreProcedure` 告诉 ASP.NET，要使用一个存储过程。ADO.NET 获得该信息后，查找存储在数据库的存储过程，并执行它，以返回相应数据。

然而，上面的方法并不十分有效。别忘了您创建了多个参数。为使存储过程返回任何必需的数据，需要指定参数 `@FirstName` 和 `@LastName` 的值。这可以通过两种方法来实现：内联（in-line）或通过 `OleDbParameters` 集合。第一种方法很简单，按如下所示修改第 1 行即可，其中 `values` 是要传入的参数：

```
dim objCmd as OleDbCommand = new OleDbCommand _
    ("SelectIDFromName value, value", Conn)
```

例如:

```
dim objCmd as OleDbCommand = new OleDbCommand _
    ("SelectIDFromName `Chris`, `Payne`", Conn)
```

该方法简单,但效率低,尤其需要从诸如表单对象中获取数据时。应用参数集合,可以用下面的代码来定义前面的参数:

```
dim objParam as OleDbParameter
objParam = objCmd.Parameters.Add("@FirstName", _
    OleDbType.Char)
objParam.Direction = ParameterDirection.Input
objParam.Value = tbFirst.Text
objParam = objCmd.Parameters.Add("@LastName", _
    OleDbType.Char)
objParam.Direction = ParameterDirection.Input
objParam.Value = tbLast.Text
```

您对该代码段应该很熟悉,其功能类似于本章前面的“参数化查询”一节中介绍的例子。现在可以像通常那样使用 Command 对象的 ExecuteReader 方法来填充 DataReader,并将其绑定到控件:

```
try
    objCmd.Connection.Open()
    objReader = objCmd.ExecuteReader
catch ex as OleDbException
    Label1.Text = "Error retrieving from the database."
end try
DataGrid1.DataSource = objReader
DataGrid1.DataBind()
objReader.Close
objCmd.Connection.Close()
```

那么,输出参数的情况如何呢?在 Access 中,不能将 @ID 作为输出参数来检索其值。但 Command 对象将 @ID 的值作为 SQL SELECT 语句的结果返回。在前面的清单中,可以通过 DataReader 访问这个值。清单 12.3 是一个使用参数的完整范例,它将前面的代码片段组合在一起。图 12.7 是将数据显示在 DataGrid 的情况。

### 清单 12.3 参数协助检索数据

```
1:  <%@ Page Language="VB" %>
2:  <%@ Import Namespace="System.Data" %>
3:  <%@ Import Namespace="System.Data.OleDb" %>
4:
5:  <script runat="server">
6:      dim Conn as new OleDbConnection("Provider=" & _
7:          "Microsoft.Jet.OLEDB.4.0;" & _
8:          "Data Source=H:\ASPNET\data\banking.mdb")
```

```
9:
10: sub Submit(obj as object, e as eventargs)
11:     dim objCmd as OleDbCommand = new OleDbCommand _
12:         ("SelectIDFromName", Conn)
13:     dim objReader as OleDbDataReader
14:     objCmd.CommandType = CommandType.StoredProcedure
15:
16:     dim objParam as OleDbParameter
17:     objParam = objCmd.Parameters.Add("@FirstName", _
18:         OleDbType.Char)
19:     objParam.Direction = ParameterDirection.Input
20:     objParam.Value = tbFirst.Text
21:
22:     objParam = objCmd.Parameters.Add("@LastName", _
23:         OleDbType.Char)
24:     objParam.Direction = ParameterDirection.Input
25:     objParam.Value = tbLast.Text
26:
27:     try
28:         objCmd.Connection.Open()
29:         objReader = objCmd.ExecuteReader
30:     catch ex as OleDbException
31:         Response.Write("Error retrieving data.")
32:     end try
33:
34:     DataGrid1.DataSource = objReader
35:     DataGrid1.DataBind()
36:
37:     objCmd.Connection.Close()
38: end sub
39: </script>
40:
41: <html><body>
42:     <form runat="server">
43:         Enter a first name:
44:         <asp:TextBox id="tbFirst" runat="server"/><br>
45:         Enter a last name:
46:         <asp:TextBox id="tbLast" runat="server"/><p>
47:
48:         <asp:Button id="btSubmit" runat="server"
```

```

49:         Text="Submit"
50:         OnClick="Submit" /><p>
51:
52:         <asp:DataGrid id="DataGrid1" runat="Server"
53:             BorderColor="black"
54:             GridLines="Vertical"
55:             cellpadding="4"
56:             cellspacing="0"
57:             width="100%"
58:             Font-Name="Arial"
59:             Font-Size="8pt"
60:             headerStyle-BackColor="#cccc99"
61:             ItemStyle-BackColor="#ffffff"
62:             AlternatingItemStyle-BackColor="#cccccc"
63:             AutoGenerateColumns="true" />
64:     </form>
65: </body></html>

```

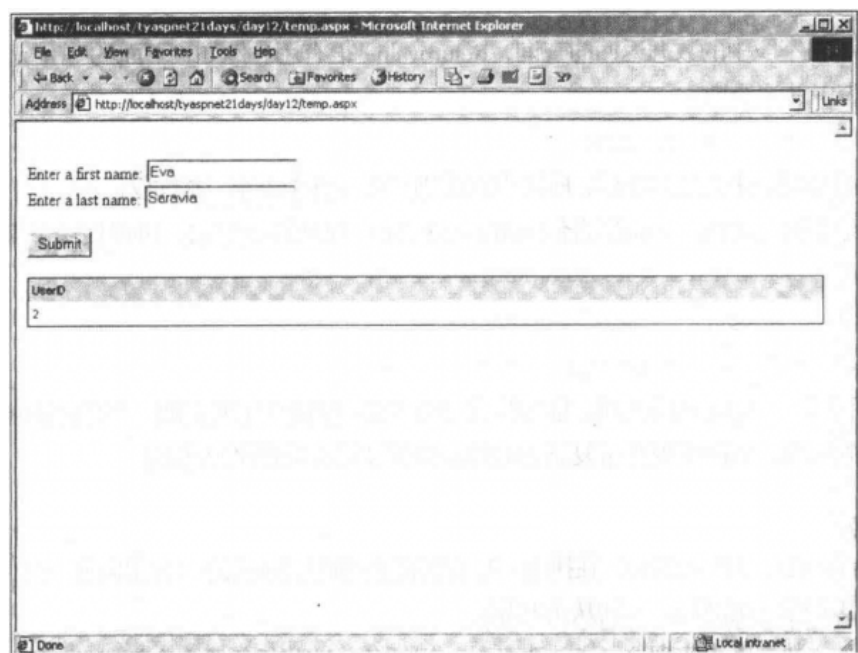


图 12.7 参数化存储过程返回的值

DataReader 包含一行，一列，其中存储了存储过程返回的值。

**注意：**SQL Server 喜欢使用输出参数，所以可充分利用这一特点，免去了使用 DataReader 的麻烦。关于如何避免使用 DataReader 的信息，请参阅本章前面的“参数化查询”一节。

输入参数和输出参数在为存储过程提供数据方面很有用。还有其他几种参数我们没有介绍，如 InputOutput 和 ReturnValue 参数。表 12.2 描述了所有可能的参数方向。

表 12.2

参数方向

方 向	描 述
Input	表示传入给查询的值
InputOutput	既可以是传入给查询也可以是查询返回的值
Output	查询返回的值
ReturnValue	表示返回值, 而不是参数

Input 和 Output 参数已在前面介绍过。InputOutput 参数对于传递给查询的数据将被修改时（例如更新数据库时）很有用。下面的查询就是一个这样的例子，这里需要根据传递给 FirstName 字段的值来更新 FirstName 字段：

```
UPDATE tblUsers SET FirstName = 'Christopher'
WHERE FirstName = 'Chris'
```

可以将其修改成如下所示来参数化上面的例子：

```
UPDATE tblUsers SET @FirstName = 'Christopher'
WHERE @FirstName = 'Chrie'
```

现在@FirstName 既是输入参数，又是输出参数，其值在查询执行后被修改，这种情况下 InputOutput 很合适。

对那些不将字段中的数据返回的查询，ReturnValue 很有用。例如，下面的 SQL 语句返回一个整数，指出表中的记录数：

```
Select Count(*) from tblUsers
```

该信息并不只是来自某个字段，所以没有使用字段名称来表示。查询仅返回一个值，而不带名称和参数。对于这类数据，非常适合使用 ReturnValue。要访问该数据，可以使用下面的代码：

```
objParam = objCmd.Parameters.Add('RETURN VALUE', _
    OleDbType.Integer)
objParam.Direction = ParameterDirection.ReturnValue
```

存储过程是一种很有用的工具，能够提高 ASP.NET 应用程序的性能。存储过程还开辟了与数据库交互的新领域，允许实现非常复杂的查询和使用更高级的数据库控件。

### 12.1.3 事务

您是否经常遇到这样的情况，在执行一项艰巨的任务时，当完成一半时发现所有的一切都是错误的。您真恨不得时光倒流，一切从头开始。

使用事务，数据库可实现这一点。事务是一组任务，这些任务要么全部成功，要么全部失败。例如，假设您创建了一个复杂的存储过程，它执行 50 条 SQL 语句。如果不使用事务，则任何一条 SQL 语句失败（如第 50 条）时，存储过程都将停止执行，导致最后一条 SQL 语句没有被执行。要执行这条 SQL 语句，必须再次执行前面的所有 49 条语句。

很多情况下会出现这种令人讨厌的情况。假设有这样一个银行应用程序，用户想将其支票账户（储在数据库中）里的钱转到储蓄账户（也存储在数据库中）中。假设第一步顺利完成，但当您试图将钱加到储蓄账户时，系统提示该账户出现故障，不能将钱存入。糟了，问题出现了，支票账户结余不对了，因为您已从其中取出一些钱。

事务正是为解决上述问题而设计的。如果使用事务，就无需担心操作过程中出现问题。发生意外时，可以撤销前面的操作。

还记得第 10 章介绍的 DataSet 的方法 AcceptChanges 和 RejectChanges 吗？这两个方法可以实现类似于事务的功能，但它们只能用于离线数据。事务则可通过活动连接用于整个数据库，只要使用的数据库应用程序支持事务（大多数商务应用程序都支持）。

事务中有三种基本操作：begin（开始）、rollback（撤销）和 commit（提交）。begin 开始一个事务。随后执行的操作将被存储在一个特殊的日志文件中，供数据库以后检查。Rollback 让您能够撤销执行的任何修改。数据库根据日志可以确定操作执行前的数据状态。Commit 提交所做的修改，修改被提交后将不能被撤销，实际上是将操作从日志中删除。

我们来看一个典型的例子，如清单 12.4 所示。

#### 清单 12.4 应用数据库事务

```

1:  <%@ Page Language="VB"%>
2:  <%@ Import Namespace="System.Data" %>
3:  <%@ Import Namespace="System.Data.OleDb" %>
4:
5:  <script runat="server">
6:      'declare connection
7:      dim Conn as new OleDbConnection("Provider=" & _
8:          "Microsoft.Jet.OLEDB.4.0;" & _
9:          "Data Source=C:\ASPNET\data\banking.mdb")
10:
11:      sub Page_Load(obj as object, e as eventargs)
12:          dim objTrans as OleDbTransaction
13:          dim objCmd as OleDbCommand = new OleDbCommand _
14:              ("DELETE from tblUsers where UserID = 32",Conn)
15:
16:          Conn.Open()
17:          objTrans = Conn.BeginTransaction()
18:          objCmd.Transaction = objTrans
19:
20:          try
21:              objCmd.ExecuteNonQuery()
22:
23:              objCmd.CommandText = "INSERT INTO tblUsers " & _
24:                  "(FirstName, LastName, Address, City, State, " & _
25:                  "Zip, Phone) VALUES " & _
26:                  "(`Jose`, `Santiago`, `34 Lake Drive`, ' & _
27:                  "`Yolktown`, `MA`, `02515`, `8006579876`)"
28:              objCmd.ExecuteNonQuery()

```



```

29:         objTrans.Commit()
30:         Label1.Text = "Both operations performed successfully"
31:     catch ex as CleDbException
32:         objTrans.Rollback()
33:         Label1.Text = ex.Message & "<p>"
34:         Label1.Text = "Both operations failed"
35:     finally
36:         objCmd.Connection.Close()
37:     end try
38: end sub
39: </script>
40:
41: <html><body>
42:     <form runat="server">
43:         <asp:Label id="Label1" runat="server"
44:             maintainstate=false /><br>
45:     </form>
46: </body></html>

```

**分析:** 该页面执行两条 SQL 语句。然而,如果在执行过程中发生了意外的错误,所做的修改不会应用到数据库中。例如,如果第二条 SQL 语句有语法错误,中途将停止执行。由于事务是在任何一条语句被执行前开始的(第 17 行),所以第一条语句所做的修改也将被撤销。

第 7 行像通常那样创建了一个 OleDbConnection 对象。第 12 行创建了一个 OleDbTransaction 对象,并在第 14 行为 Command 对象指定了一条 SQL 语句。事务需要一个有效、打开的连接,所以第 16 行打开了 connection 对象,然后,通过调用 BeginTransaction 来开始事务。注意,该方法是 OleDbConnection 对象的一个成员,Connection 对象必须初始化事务,这样如果数据库不支持,您将不会使用事务。connection 对象检测到初始化的执行,如有必要将中止。然后,所有其他命令(rollback、commit)将在 OleDbTransaction 上执行。

第 18 行通过 Transaction 属性告知 OleDbCommand 对象,您将使用哪一个 OleDbTransaction 对象。在 try 语句块中,您执行一条 DELETE SQL 语句,创建一个 SQL INSERT 语句,并执行它。如果没有出现错误,则调用 Commit 方法使修改生效。

如果出现错误,则撤销所有的修改。Try 语句块捕获异常,并转到 catch 语句处执行,该语句调用 RollBack 方法,并向用户发出一条消息。不管 try 语句块中是否会发生错误,程序最后都会调用 finally 语句块,来关闭连接。

这样,要么所有的语句都被执行,要么都不执行。

## 12.2 高级 XML 技术

第 11 章介绍了 XML 如何用于表示几乎任何数据类型以及如何通过 ASP.NET 打开、读取 XML 文件和将数据写入到 XML 文件中。当然,XML 的功能不止这些。

接下来的几节将介绍一些操纵 XML 的高级技术。您将学到如何使用 XmlNavigator 对象在 XML

文档中导航, XmlNavigator 对象让您能够使用接下来的两节介绍的概念: Xpath 查询和 XSL 转换。Xpath 是一种查询语言, 用于检索 XML 文件中的信息, 就像 SQL 用于返回数据库中的数据一样。XSL 转换使得将 XML 文件转换为其他任何类型的结构化文档 (如 HTML 页面) 成为可能。使用这些方法, 您可以随心所欲地控制 XML。

### 12.2.1 XpathDocument

**新术语:** 在本部分的前面, 您学习了如何使用 XmlNode 和 XmlDocument 对象在 XML 文本文件中导航。访问 XML 数据时, 这些对象创建节点树。也就是说, 这些对象能够查看整个 XML 文件, 并建立一种面向对象的、层次式数据表达。基本上, XmlDocument 收集整个 XML 文件的内容, 然后让您查看数据。然而, XpathDocument 并不创建节点树, 而只是每次查看一个节点。当您移到这些节点时, XpathDocument 创建对应的节点——动态节点。

XpathDocument 类似于 XmlDocument, 但是为提高性能而设计的, 因此提供的特性没有后者多。XpathDocument 最适合用于 Xpath 查询 (因此名叫 XpathDocument) 和 XSLT 转换, 这两个主题将在接下来的几节中介绍。首先, 让我们看看如何使用 XpathDocument, 如清单 12.5 所示:

#### 清单 12.5 创建一个 XpathDocument

```
1:  <%@Page Language="VB" %>
2:  <%@Import Namespace="System.Xml" %>
3:  <%@import Nmaespace="System.Xml.XPath" %>
4:
5:  <script runat="server">
6:      sub Page_Load(obj as object,e as eventargs)
7:          ' Create an XPathDocument
8:          Dim objDocument as New XPathDocument _
9:              (Server.MapPath("../day11/books.xml"))
10:      end sub
11:  </script>
12:
13:  <html><body>
14:  </body></html>
```

**分析:** 清单 12.5 并没有什么新的内容, 只是在第 8 行创建了一个新的 XPathDocument 对象, 仅此而已。事实上, XPathDocument 的功能仅限于此。例如, 不能将其用于在 XML 文件中导航或编辑它。创建该对象的目的在于快速、便捷地访问 XML 文件以及直接将数据发送给其他对象进行处理。

具体地说, 最常与 XPathDocument 搭配使用的对象是 XPathNavigator, 后者提供了大量在 XML 文件中导航的方法, 它也只包含这些方法, 这使得 XPathNavigator 是一种高效的导航工具。清单 12.6 是一个例子。

#### 清单 12.6 使用 XPathNavigator 在 XML 文档中导航

```
1  <%@Page Language="VB" %>
2  <%@Import Namespace="System.Xml" %>
```

```

3 <%@Import Namespace="System.Xml.XPath" %>
4
5 <script runat="server">
6     sub Page_Load(obj as object,e as eventargs)
7         Dim objDocument as New XPathDocument _
8             (Server.MapPath("../day11/books.xml"))
9
10        Dim objNav as XPathNavigator = objDocument.
11            CreateNavigator
12        objNav.MoveToRoot()
13        DisplayTree(objNav)
14    end sub
15
16    public sub DisplayTree (objNav as XPathNavigator )
17        if (objNav.HasChildren)
18            objNav.MoveToFirstChild()
19
20            Format(objNav)
21            DisplayTree(objNav)
22
23            objNav.MoveToParent()
24        end if
25
26        while (objNav.MoveToNext())
27            Format (objNav)
28            DisplayTree (objNav)
29        end while
30    end sub
31
32    private sub Format (objNav as XPathNavigator)
33        if Not objNav.HasChildren
34            if (objNav.NodeType = XPathNodeType.Text)
35                lblMessage.Text += "" & objNav.Value & "<br>"
36            end if
37        else
38            lblMessage.Text += "<lt;" & objNav.Name & _
39                "<gt;<br>"
40
41            if objNav.HasAttributes
42                while (objNav.MoveToNextAttribute())

```

```

43         lblMessage.Text += '&nbsp;&nbsp;&nbsp;&lt;' & _
44         objNav.Name & '&gt;' & objNav.Value & _
45         "<br>"
46     end while
47
48     objNav.MoveToParent()
49 end if
50 end if
51 end sub
52 </script>
53
54 <html><body>
55   <ASP:Label id='lblMessage' runat='server' />
56 </body></html>

```

该清单的大部分内容类似于上一章使用 XmlDocument 和 XmlNode 对象的情况。第 7-8 行和清单 12.5 那样创建 XPathDocument 对象。在第 10-11 行, XPathDocument 的 CreateNavigator 方法创建了一个 XPathNavigator 对象, 用于在 XML 文件中导航。第 12 行的 MoveToRoot 将 XPathNavigator 移到文件的开头, 第 13 行调用了自定义的 DisplayTree 函数。

第 16-30 行中的 DisplayTree 是一个递归函数。首先, 该函数确定本结点是否有子结点。如果有子结点, 则显示每个子节点的数据。MoveToFirstChild 将游标移到当前节点的第一个子节点。Format 是一个自定义的函数, 负责将数据写入到页面中——稍后将进行讨论。第 21 行再次调用 DisplayTree, 对该子节点重复执行这一过程。该过程重复执行, 直到没有其他的子节点为止。遍历完所有的子节点后, MoveToParent 方法被调用, 将游标移到 XML 层次结构的上一层。

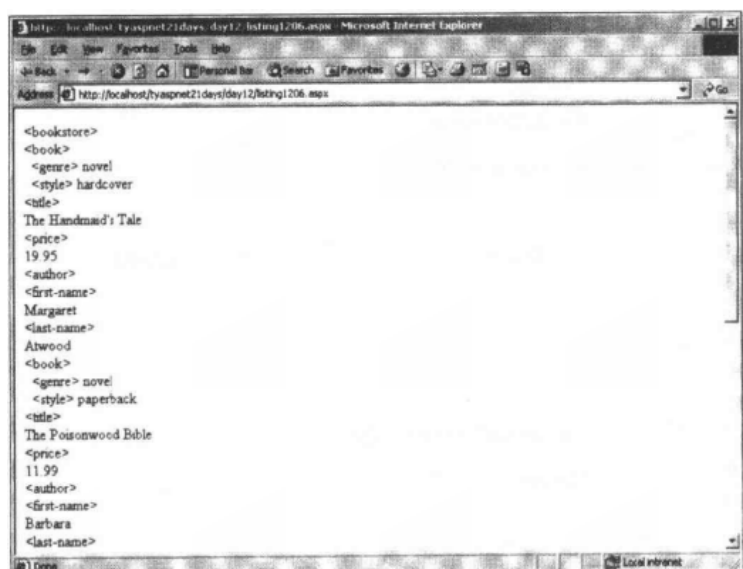


图 12.8 使用 XPathNavigator 显示 XML 数据

DisplayTree 的第二部分也是一个递归调用, 用于在同一层次的节点间移动。MoveNext 方法移到

下一节点,直到本层次末尾。此时,该方法将返回 `false`,导致 `while` 循环结束。第 27 行再次调用 `Format` 方法来显示数据并调用 `DisplayTree`。这一过程将不断重复下去,直到 XML 文件中没有更多的节点为止。

下面,让我们看看第 32-51 行的 `Format` 方法。不要被该方法吓倒,其大部分代码只是将 HTML 格式标签写入到浏览器中。第 33 行的 `if` 语句确定本节点是否有子节点:如果没有,则显示该节点的值;如果有子节点,则将值显示在括号内。第 41 行的 `if` 语句确定是否有属性,如果有,则打印一条消息。最后,第 42 行的 `while` 循环遍历并显示所有的属性。图 12.8 是该清单的运行结果。

### 12.2.2 Xpath

Xpath 是 W3C 为访问 XML 文件而制定的一种语言规范,它让您能够像使用 SQL 语句查询传统数据库那样查询 XML 数据。该语言非常复杂,所以本章不介绍其语法,而是介绍如何对 XML 文档使用这种查询。

Xpath 查询是由关键字组成的字符串,其中的关键字表示 XML 文件的组成部分。这些查询是由 `XpathNavigator` 的 `Select` 方法执行的。以 `books.xml` 文件为例,可以使用下面的语句:

```
objNav.Select("descendant::book/author/last-name")
```

该查询返回所有书籍的作者名。下面的查询语句只返回最后一本书的价格:

```
objNav.Select("//book[last()]/price/text()")
```

清单 12.7 说明了如何创建一个这样的简单页面,即接受用户提供的 Xpath 查询并显示返回的数据。

#### 清单 12.7 使用 Xpath 查询来返回 XML 数据

```
1 <%@Page Language="VB" %>
2 <%@Import Namespace="System.Xml" %>
3 <%@Import Namespace="System.Xml.XPath" %>
4
5 <script runat="server">
6     sub SelectData(obj as object,e as EventArgs)
7         Dim objDocument as New XPathDocument _
8             (Server.MapPath("../day11/books.xml"))
9
10        Dim objNav as XPathNavigator = objDocument.CreateNavigator()
11
12        lblMessage.Text = ""
13        try
14            dim objIterator as XPathNodeIterator = _
15                objNav.Select(tbQuery.Text)
16
17            While objIterator.MoveNext()
18                lblMessage.Text += "<";
19                objIterator.Current.Name & ">";
20                objIterator.Current.Value & "<br>"
```

```

21     end while
22     catch ex As Exception
23         lblMessage.Text = ex.Message
24     end try
25 end sub
26 </script>
27
28 <html><body>
29     <form runat="server">
30         Enter an XPath query. For instance:<p>
31
32         <b>//book[last()]/@ISBN/text()</b> or
33         <b>descendant::book/author/last-name</b><p>
34
35         <asp:Textbox id="tbQuery" runat="server"/>
36         <asp:Button id="btnSubmit" text="Submit"
37             runat="server" OnClick="SelectData" /><p>
38         <asp:Label id="lblMessage" runat="server"/>
39     </form>
40 </body></html>

```

分析：大部分工作都是在 SelectData 方法中完成的，该方法在第 36 行的按钮控件的事件处理程序。第 7-10 行看起来很熟悉，这里创建了 XpathDocument 和 XpathNavigator 对象。第 13 行的 try 语句块是函数真正开始的地方。您使用 Select 方法，根据文本框中输入的查询语句来返回 XML 数据。该方法返回一个 XmlPathNodeIterator，让您能够轻松地遍历返回的结果。While 循环调用 MoveNext 移到查询返回的下一个节点，并使用 Current.Name 和 Current.Value 属性将当前条目显示到浏览器中。清单 12.7 的运行结果如图 12.9 所示。



图 12.9 使用 Xpath 查询 XML 数据

Xpath 语言为检索 XML 数据提供了一种非常健壮、神奇的机制。现在您不需使用 DataSet 来执行这些查询。有关 Xpath 的更详细的信息，请参阅 W3C 在线资源，其网址为 <http://www.w3c.org/TR/xpath>。

### 12.2.3 XslTransforms

所有的 XSL 指令都是由 XSL 转换处理器 (XsIT) 处理的。您使用 XSL 创建样式表单，告诉 XsIT 如何转换数据。正如使用样式表单告知 HTML 如何格式化页面部件一样，可以使用 XML 来告知 XsIT 如何格式化转换后的 XML 数据。图 12.10 说明了将 XML 文档转换为其他文档的过程。

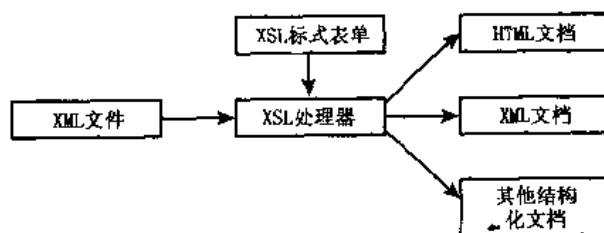


图 12.10 XSL 处理器使用 XSL 样式表单将 XML 文档转换为另一种结构化文档

XSL 处理器依赖于 Xpath 查询来返回 XML 文件，后者再根据样式表单来确定其格式。在 ASP.NET 实现转换的过程很简单——您只需要提供一个 XSL 样式表单。清单 12.8 列出了样式表单 books.xsl。

#### 清单 12.8 一个 XSL 样式表单

```

1 <xsl:stylesheet
2   xmlns:xsl="http://www.w3.org/1999/XSL-Transform"
3   version="1.0">
4   <xsl:template match="/">
5     <root>
6       <xsl:apply-templates/>
7     </root>
8   </xsl:template>
9   <xsl:template match="bookstore">
10    <HTML><BODY>
11    <TABLE width="450">
12    <TR>
13      <TD><b>Title</b></TD>
14      <TD><b>Price</b></TD>
15    </TR>
16    <xsl:apply-templates select="book">
17    </TABLE>
18    </BODY></HTML>
19  </xsl:template>
20  <xsl:template match="book">
  
```

```

21      <TR>
22          <TD><xsl:value-of select="title"/></TD>
23          <TD><xsl:value-of select="price"/></TD>
24      </TR>
25  </xsl:template>
26 </xsl:stylesheet>

```

**分析：**该样式表单将 XML 文件转换为 HTML 文档。Xsltemplate 标记指定如何格式化 XML 文档的特定部分。例如，第 20 行格式化所有名为 book 的节点。

您这里选择的是将 XML 文件转换为 HTML 文件，所以使用了 HTML 和表标记。您可以很容易地将 XML 转换为其他的 XML 文件——只是提供的标记不同而已。

清单 12.9 使用该样式表单将第 11 章创建的 books.xml 文件转换为 HTML 文档。

#### 清单 12.9 使用 XSL 样式表单和 XsIT 将 XSL 文件转换为 HTML 文档

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>
3 <%@ Import Namespace="System.Xml.XPath" %>
4 <%@ Import Namespace="System.Xml.Xsl" %>
5
6 <script runat="server">
7     sub Page_Load(obj as object, e as eventargs)
8         Dim objDocument as New XPathDocument _
9             (Server.MapPath("../day11/books.xml"))
10
11         Dim objNav as XPathNavigator = _
12             objDocument.CreateNavigator
13
14         Dim objXslT As XslTransform = New XslTransform()
15         dim objWriter as XmlTextWriter = new XmlTextWriter
16             (Server.MapPath("output.html"), nothing)
17
18         try
19             objXslT.Load(Server.MapPath("books.xsl"))
20             objXslT.Transform(objNav, nothing, objWriter)
21             objWriter.Close
22
23             lblMessage.Text = "File written successfully"
24         catch ex As Exception
25             lblMessage.Text = ex.Message
26         end try
27     end sub

```



```

28     </script>
29
30     <html><body>
31         <asp:Label id='lblMessage' runat='server'
32             maintainstate=false/>
33     </body></html>

```

分析: 首先要注意的是新添加的名称空间 `System.Xml.Xsl`。第 6-11 执行的是标准流程: 创建 `XpathDocument` 和 `XpathNavigator`。第 13 行创建了 `XslTransform` 对象, 第 14 行创建了一个 `XmlTextWriter` 对象, 用来将转换后的文档写入名为 `output.html` 的 HTML 文件中

在 `try` 语句块内, XSL 文件被载入到 `XslTransform` 对象中, 后者将该样式表单作为模式来格式化一个新的 HTML 文档, 然后, `XslTransform` 对象调用 `Transform` 方法, 根据 XSL 样式表单转换文档。其中第一个参数是要转换的 XML 文档的内容 (由 `DocumentNavigator` 表示), 第二个参数给出了要提供给 XSL 文件的所有其他参数 (这里没有), 最后一个参数是用于放置转换后的内容的 `XmlTextWriter`。

最后, 您关闭 `writer`, 并向用户发送一条简单的消息。现在清单所在的目录中将包含一个名为 `output.html` 的文件。该文件包含从 XML 文件转换得到的内容, 如图 12.11 所示。

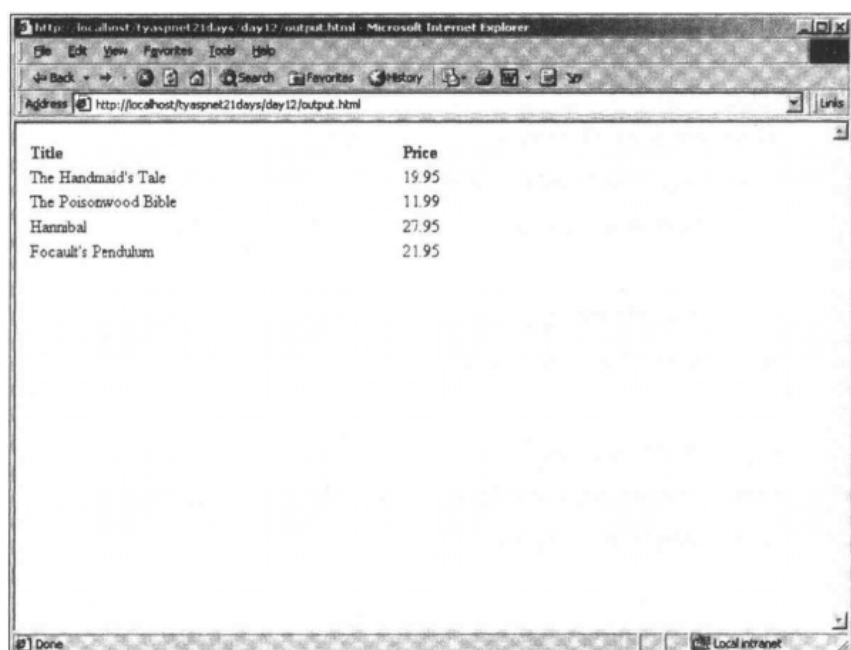


图 12.11 XSL 转换输出的 HTML 结果

也可以使用下面的代码替换第 19 行, 以便将转换后的内容载入到 `XmlReader` 中, 并立即显示出来。

```
objReader = objXslT.Transform(objNav, nothing)
```

其中, `objReader` 是 `XmlReader`。清单 12.10 是一个使用 `XmlReader` 代替 `XmlText` 的完整范例。

#### 清单 12.10 使用 `XmlReader` 显示转换后的 XML 数据

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>

```

```

3 <%@ Import Namespace="System.Xml.XPath" %>
4 <%@ Import Namespace="System.Xml.Xsl" %>
5
6 <script runat="server">
7     sub Page_Load(obj as object, e as eventargs)
8         Dim objDocument as New XPathDocument _
9             (Server.MapPath("../day11/books.xml"))
10
11         Dim objNav as XPathNavigator = _
12             objDocument.CreateNavigator
13         Dim objXSLT As XslTransform = New XslTransform()
14         dim objReader as XmlReader
15
16         try
17             objXSLT.Load(Server.MapPath("books.xsl"))
18             objReader = objXslT.Transform(objNav, nothing)
19             While objReader.Read()
20                 Response.Write("<b>" & objReader.Name & "</b> " & _
21                     objReader.Value & "<br>")
22             End While
23
24             lblMessage.Text = "File written successfully"
25         catch ex As Exception
26             lblMessage.Text = ex.Message
27         end try
28     end sub
29 </script>
30
31 <html><body>
32     <asp:Label id="lblMessage" runat="server"
33         maintainstate=false/>
34 </body></html>

```

该清单同清单 12.9 相同，只是使用了 `XmlReader` 对象，而不是 `XmlTextWriter`。因此，内容没有发送到输出文件中。第 18-21 行使用 `while` 循环和 `Read` 方法来遍历并显示数据。

`XmlTransform` 对象只有两个方法：`load` 和 `Transform`，所以很容易使用。

您也许还记得，`XslTs` 依赖于 `Xpath` 查询。`XSL` 文件指定了要转换的节点的名称。而 `XmlTransform` 对象使用 `Xpath` 查询来检索指定的节点。您可以使用这些查询来检索数据，并手工格式化它们，但 `XmlTransform` 可以自动为您完成这项任务，何苦自己动手呢？

更多有关 `XSL` 的信息，请参阅 W3C 在线资源，其网址为 <http://www.w3c.org/TR/xsl> 和 <http://www.w3c.org/TR/xslt>。

## 12.3 这不是 ASP

本章介绍的很多技术在传统 ASP 中也可用。例如，参数化存储过程曾是一种常用的数据库查询方式。ASP 插件中也具有 XML 操纵能力。ASP.NET 为实现这些技术提供了更简单的方式，因为所有必须的功能都是内置的，并且完全是面向对象的。

对很多 ASP 开发人员来说，这些概念并不陌生，所以转向 ASP.NET 不会太难。只是实现方式不同而已。已有的存储过程、XSL 样式表单及 Xpath 查询仍然可用，只需使用不同的对象来访问它们。

## 12.4 总 结

在 ASP.NET 中，实现某一任务的方法很多。例如，第 10 章介绍了如何在 SQL 语句中使用 `OleDbCommand` 和 `OleDbDataAdapter` 对象来检索数据库中的数据。而本章介绍了如何参数化 SQL 语句，并检索相同的数据。第 11 章介绍了如何使用 `XmlDocument` 对象操纵 XML 文件，而本章介绍的 `XmlNavigator` 对象可以完成相同的任务，还能完成其他一些任务。虽然掌握的方法越多，作为程序员的能力越强，但即使您知道所有可用的方法，也应该选择某种特定的方法，并一直使用这种方法，这使得您和其他开发人员以后阅读您编写的代码时，更容易理解它们。

参数化查询让您能够将不同的数据传递给 SQL 语句，包括输入和输出。要使用参数，需要使用带前缀@的变量替换查询中可变动的部分。例如：

```
SELECT FirstName FROM tblUsers WHERE UserID = @ID
```

在 ASP.NET 中，可使用 `OleDbParameter` 对象来创建参数。您可以指定参数的名称、值、类型和方向（如输入和输出）。例如：

```
SELECT FirstName FROM tblUsers WHERE UserID = @ID
```

存储过程是编译过的 SQL 语句，使得数据库程序执行查询的效率比常规语句高。使用存储过程有很多优点，包括性能和模块化程度更高。

存储过程是在数据库应用程序（例如 MS SQL Server 2000 或 Access）中使用常规 SQL 语句来创建的。通过将 `CommandType` 属性设为 `StoreProcedure`，可以在 ASP.NET 页面中使用 `OleDbCommand` 对象执行存储过程。您还像常规参数化查询那样为存储过程创建参数。查询的返回值可以放置在输出参数中，参数化查询还可以返回数值参数或数据控件（如 `DataReader`）。

事务让数据库能够撤销所做的修改。在事务中执行的所有命令都被记录在日志中，这样数据库根据日志恢复到命令执行前的状态，以撤销所做的修改。实际上，事务让您能够实现要么全部执行要么全不执行的目标。使用 `connection` 对象调用 `BeginTransaction` 方法可以开始一个事务，随后执行的所有语句将得到保护——这些语句可以被撤销。`Commit` 方法使修改生效，而 `RollBack` 方法用于撤销所有的修改。

`DocumentNavigator` 使得可以实现对 XML 文档的游标式访问，根据需要动态地载入节点。该对象类似于 `XmlDocument` 对象，并支持很多相同的方法。另外，`DocumentNavigator` 支持 Xpath 查询和 XSL 转换。

Xpath 是收集 XML 文件特定部分的语句，就像 SQL 语句用于收集数据源中的特定部分一样。

您使用 `DocumentNavigator.Select` 方法来执行 Xpath 查询。

XSL 转换让您能够将 XML 文件转换为另一种结构化文档，如 HTML 页面。`XslTransform` 类使用 XSL 样式表单来确定转换后的文档的格式，它支持两个方法：`Load` 和 `Transform`，前者载入 XSL 样式表单，后者用于转换 XML 文档，并将转换结果放到 `XmlTextReader` 或 `XmlTextWriter` 中。

下一章将介绍数据访问的另一个重要部分：文件 I/O。ASP.NET 使用许多不同的文件来执行应用程序，下一章将介绍这些文件。下一章还将介绍如何读写文件以及如何查看和修改其属性。

## 12.5 问与答

问：事务是如何恢复数据的？

答：数据库存储了一个事务日志。该日志记录了所有执行过的语句以及被修改的数据。数据库根据该日志来确定语句执行前的数据状态，必要时恢复到修改前的状态。

问：存储过程到底能在多大程度上提高性能？

答：这很难说。主要取决于使用的数据库应用程序和执行的命令以及其他一些因素。但研究表明，使用存储过程可减少的系统处理时间可高达 25%。更详细的信息，请参阅 <http://www.4guysfromrolla.com/webtech/sqlguru/q120899-2.shtml> 中的文章。

问：有没有一些高级的文件 I/O 技术？

答：有！例如，`FileSystemWatch` 组件让您能够监视文件系统，以发现并处理发生的变化，例如重命名或创建和删除目录。

本章之所以没有介绍这些技术，是因为这些技术不能直接用于 ASP.NET 中。例如，只要应用程序在运行，`FileSystemWatcher` 便会监视所做的修改。但在 ASP.NET 中，这没什么意义——在 Web Server 停止之前，应用程序将一直运行，但在页面被请求之前，不会发生事情。页面极少有执行的时候，所以实现该对象没什么意义。即便如此，还是应通过 .NET 框架 SDK 文档了解其他对象。

## 12.6 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 12.6.1 小测验

1. 指定一条 SQL 语句，以创建一个可接受一个输入参数和一个输出参数的空存储过程（不执行任何 SQL 语句）。
2. 有哪四种参数方向？
3. 判断正误：要返回多条记录，可使用输出参数。
4. 下面的代码是否能够运行？

```
Response.Write("ID = " & objCmd.Parameters("@ID").Value.ToString());
objReader.Close();
```

5. `DocumentNavigator` 和 `XmlDocument` 之间有什么差别？
6. XSL 样式表单有什么作用？它与 Xpath 查询有何关系？
7. 如何使用 `XmlTextReader` 来返回 XSL 转换的结果？使用 `XmlTextWriter` 呢？

### 12.6.2 练习

创建一个存储过程，使用参数将新记录插入到 `tblUsers` 表中。创建一个 ASP.NET 页面，让用户输入记录的数据，并使用参数执行存储过程。

## 第 13 章

# Web 服务器上的文件读写

对 ASP.NET 来说, 文件输入/输出 (I/O) 是一个很重要的课题。您已经在 ASP.NET 页面使用过外部文件 (如 `config.web` 文件, `code-behind` 表单、XML 数据文件等), 但本章将介绍如何使用其他类型的文件来扩展 ASP.NET 应用程序。

除可以存储数据外, 外部文件还有很多其他的功能。本章将介绍以下内容:

- ASP.NET 如何使用文件, 在何处使用;
- 如何在 ASP.NET 页面中包含文件;
- 如何访问文件和目录的属性;
- 如何打开和读写文件;
- 如何为每个用户提供自定义的文件存储空间 (独立的存储控件)。

### 13.1 在 ASP.NET 中使用文件

通过 .NET 框架, ASP.NET 使得访问外部文件 (除 `.aspx` 文件之外的其他文件) 非常容易。实际上, ASP.NET 要正确运行, 必须使用外部文件, 这种文件包括 `web.config` 和 `global.asax`, 这些文件控制了应用程序的运行方式。

外部文件还可用于扩展 ASP.NET。您已接触过几种这样的文件, 如被编译到 DLL 中的用户控件、自定义控件和名称空间。这些文件为应用程序提供了更多的功能, 同时保持页面相对简单。

最后, ASP.NET 还允许您访问那些不属于应用程序的文件, 这很有用。这包括文本数据源、导入的新闻摘要文件、应用程序的 `readme` 文件简历或其他任何您认为有用的文件。ASP.NET 还允许您访问服务器的整个文件系统, 这为您提供了很大的灵活性。

### 13.2 包含外部文件

ASP.NET 访问文件的方法之一是文件包含 (file inclusion)。该方法类似于将一个文件的内容插入到另一个文件中。幸运的是, 事实上您不需要这样做。接下来几节的讨论将说明各种包含文件的方法。

### 13.2.1 服务器端包含

服务器端包含是一种将代码从页面中分离出来的方法。如果需要重用代码或简化页面的编辑,则服务器端包含将很有帮助。

服务器端包含的语法如下:

```
<!--#include file="FileName"-->
```

或

```
<!--#include Virtual="FileName"-->
```

关键字 `virtual` 用于通过虚拟目录指定文件,而 `file` 指定文件在本地目录结构的路径。例如,如果想将位于 `http://www.yourserver.com/includes/file1.aspx` 中的文件内容包含到通过 `http://www.yourserver.com/someDir1/someDir2/someFile.aspx` 运行的 ASP.NET Web 页中,可以使用下面的两条 `#include` 语句之一:

```
<!--#include file="../../../includes/file1.aspx"-->
```

```
<!--#include virtual="/includes/file1.aspx"-->
```

注意,必须根据使用服务器端包含的 ASP.NET 页面的位置相对于被包含文件的位置对 `#include file` 语句进行修改。这种包含称为相对文件路径包含,当文件的位置发生变化时,将很麻烦,因此 ASP.NET 还允许您使用语法 `#include virtual` 来实现绝对路径包含,如上述代码片段的第二条语句所示。这让您指定服务器端包含的路径时不用考虑调用包含文件的 ASP.NET 页面所在的位置。

让我们将一个常用的页眉封装到一个服务器端包含中,以便在 ASP.NET 页面中引用它。首先创建一个名为 `header.aspx` 的文件,然后在该文件中加入以下代码:

```
<a href="index.aspx"><img src='header.gif' alt="Home"></a><hr>
```

```
<a href='index.aspx">Home</a> <a href="logout.aspx">Log out</a>
```

请不要加入任何其他的标记,如 `<form>`,这些标记应加入到调用页面中。现在让我们在同一目录下创建下面的 ASP.NET 页面,并包含上述页眉:

```
<script language="VB" runat="server">
    Sub Page_Load(obj as Object, e as EventArgs)
        'do something
    End Sub
</script>
<html><body>
    <!--#include file="header.aspx"-->
    Hi there!
</body></html>
```

在处理任何 ASP.NET 命令之前,ASP.NET 先将 `header.asp` 文件的内容插入文件中。所以,当页面准备好执行时,ASP.NET 认为文件的实际内容如下:

```
<script language="VB" runat="server">
    Sub Page_Load(obj as Object, e as EventArgs)
        'do something
    End Sub
</script>
<html><body>
```

```

<a href="index.aspx"></a><br>
<a href="index.aspx">Home</a> <a href="logout.aspx">log out</a>
Hi there:
</body></html>

```

上述代码的运行结果如图 13.1 所示。

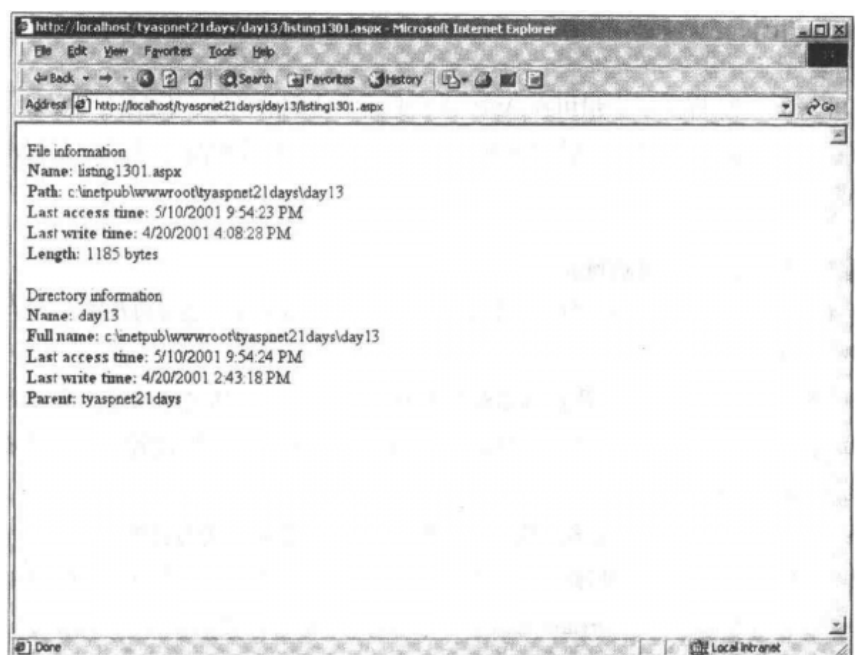


图 13.1 服务器端包含文件被看作是调用页面的一部分

**注意：**这一切都是在服务器上进行的，然后结果才被发送给浏览器。在执行代码并将结果发送给客户端之前，将在服务器上合并文件，所以 `Response.Redirect` 仍然适用。

可以在页面任何位置使用服务器端包含，所以可将该页眉用作页脚，其中甚至可以包含诸如 `<html>`、`<body>` 等标记或 ASP.NET 服务器控件。

**警告：**默认情况下，只能在扩展名为 `.shtml`、`.shtm`、`.asp`、`.asa`、`.asax` 和 `.aspx` 的文件中使用服务器端包含。如果在 `.html` 文件中使用服务器端包含，则包含将不起作用。可以在 IIS 5.0 中使用 Internet Services Manager 修改可使用服务器端包含的文件扩展名。

#### 服务器端包含与其他包含的比较

您也许想知道服务器端包含和用户控件的区别。两者的使用原因相同：封装 UI 部分。差别在于：用户控件用于显示 UI，而服务器端包含可用于任何目的，包括封装常用的函数和常量。

通常，如果想封装一个 UI 元素，则最好使用用户控件，因为这样可得到一个可通过编程进行操纵的元素，而服务器端包含无法实现这一目的。

应 该	不 应 该
当需要在页面中包含常用的元素，如类或函数时，应使用服务器端包含	只是想封装 UI 元素时，不要使用服务器端包含，而应使用用户控件



### 13.2.2 其他包含

还有其它方法可在 ASP.NET 页面中包含文件——其中的大部分方法已经介绍过。更多关于 code-behind 表单的信息，请参见第 6 章；更多关于关键字 `import` 的信息，请参见第 2 章；关于用户控件的信息，请参见第 5 章。

虽然这些方法实现方式不同，但每种方法都让 ASP.NET 页面访问其他文件的内容。

## 13.3 文件访问

名称空间 `System.IO` 提供了大量可在 ASP.NET 页面中使用的功能，包括读写文件、创建和删除目录以及查看文件和目录的属性。该名称空间还让您能够对文件系统中发生的事件做出反应，如另一用户删除或创建目录。

### 13.3.1 文件、流、Reader 和 Writer

介绍 `System.IO` 类并访问文件之前，先介绍文件和流（stream）之间的差别，因为在 ASP.NET 中，两者绝然不同。

**新术语：**从技术角度讲，文件是具有名称的数据集合，它位于确定的位置，是永久性的。例如，计算机硬盘上的文件具有名称和路径，需要时可以访问它。可以将文件看作是一个里面装有东西的盒子——一个有形的东西。

**新术语：**而流并不涉及到文件名、路径或存储位置。流可用于读写任何位置（如硬盘、网络或内存）的数据。因此，流是一种可访问任何数据（包括文件）的方式。打开或读取文件时，您实际上是使用流来完成上述工作的。`Stream` 类是一个基类，很多类都是从它派生而来的。

流可用于二进制存取：以 1 或 0 的形式来读取文件中的原始数据。这种存取方法有其用武之地，但在 ASP.NET 中，主要使用 Unicode 存取方式，读取的是字符和字符串，而不是 0 和 1。

为此，.NET 框架提供了另外两个对象：`TextReader` 和 `TextWriter`。这里不打算过多讨论这些基类，但您需要知道的是，很多类都是从这两个类派生而来的，第 11 章介绍的 `XmlTextReader` 和 `XmlTextWriter` 对象是其中的两个，本章还将介绍其他几个。

因此存在两个独立的分支：用于二进制存取的 `Stream` 及其子类；用于 Unicode 存取的 `TextReader`、`TextWriter` 及其子类。在 .NET 框架中，很容易区分它们：对于从 `Stream` 派生而来的类，其名称中通常包含 stream（`FileStream`、`MemoryStream` 等）；对于从 `TextReader` 和 `TextWriter` 派生而来的类，其名称中通常包含 reader 或 writer（`XmlTextReader`、`BinaryWriter` 等）。

幸运的是，有些类可以实现二进制数据和 Unicode 数据的转换，本章后面将介绍其中的两个：`StreamReader` 和 `StreamWriter`，它们对 ASP.NET 文件 I/O 非常重要。图 13.2 描述了所有这些对象之间的关系。

**注意：**不要将 stream 和 streaming（流式访问）搞混了。stream 可用于访问数据源，而 streaming 提供对数据的动态存取，需要时可逐条存取数据。Stream 是名词，而 streaming 是动词。Stream 可以是流式的（streamed），但不一定非得如此。

**新术语：**`Stream` 对象可用于异步（asynchronous）访问。也就是说，在您对文件执行某种操作的同时，还可执行其他任务。例如，将数据写入到大型文件时，您预计需要花 1 分钟。这时可以以异步方式打开文件，在后台执行写操作，同时运行代码，将用户重定向到网站的其他页面。

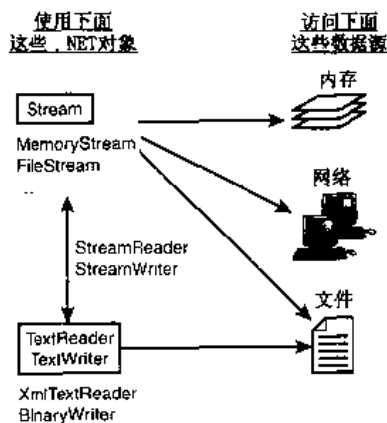


图 13.2 Stream、TextReader 和 TextWriter 之间的关系

与异步相对的是同步 (synchronous) 执行。在同步模式下, 必须等到当前任务完成后才能继续执行下一个任务, 这可能会降低性能。图 13.3 说明了两种模式下操作顺序的差别。这里重点介绍同步访问, 更详细的信息, 可参见 .NET 框架 SDK 文档。

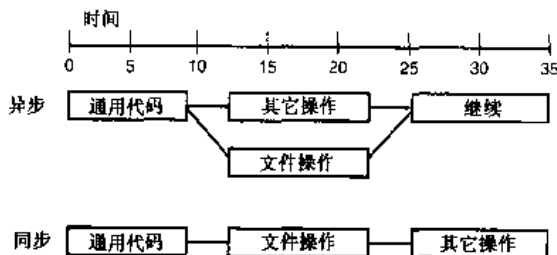


图 13.3 异步操作的性能更好

### 13.3.2 查看文件和目录

这里将重点介绍 4 种主要对象: File 对象、Directory 对象、FileInfo 对象和 DirectoryInfo 对象。前两个对象提供了用于创建、修改和删除文件或目录的方法, 而后两个对象提供了用来查看文件或目录信息的属性。清单 13.1 是一个范例。

清单 13.1 查看文件和目录信息

```

1 <%@ Import Namespace="System.IO" %>
2 <script language="VB" runat="server">
3
4 sub Page_Load(obj as object, e as eventargs)
5     dim f as new FileInfo(Server.MapPath("listing1301.aspx"))
6
7     lblMessage.Text = "File information<br>" & _
8         "<b>Name: </b>" & f.Name & "<br>" & _
9         "<b>Path: </b>" & f.DirectoryName & "<br>" & _
10        "<b>Last access time: </b>" & f.LastAccessTime & _

```

```

11         "<br>" & _
12         "<b>Last write time: </b>" & f.LastWriteTime & _
13         "<br>" & _
14         "<b>Length: </b>" & f.Length & " bytes<p>"
15
16     'return parent directory
17     dim dir as DirectoryInfo = f.Directory
18
19     lblMessage.Text += "Directory information<br>" & _
20         "<b>Name: </b>" & dir.Name & "<br>" & _
21         "<b>Full name: </b>" & dir.FullName & "<br>" & _
22         "<b>Last access time: </b>" & dir.LastAccessTime & _
23         "<br>" & _
24         "<b>Last write time: </b>" & dir.LastWriteTime & _
25         "<br>" & _
26         "<b>Parent: </b>" & dir.Parent.Name & "<br>"
27
28     end sub
29
30 </script>
31
32 <html><body>
33     <asp:label id="lblMessage" runat=server/>
34 </body></html>

```

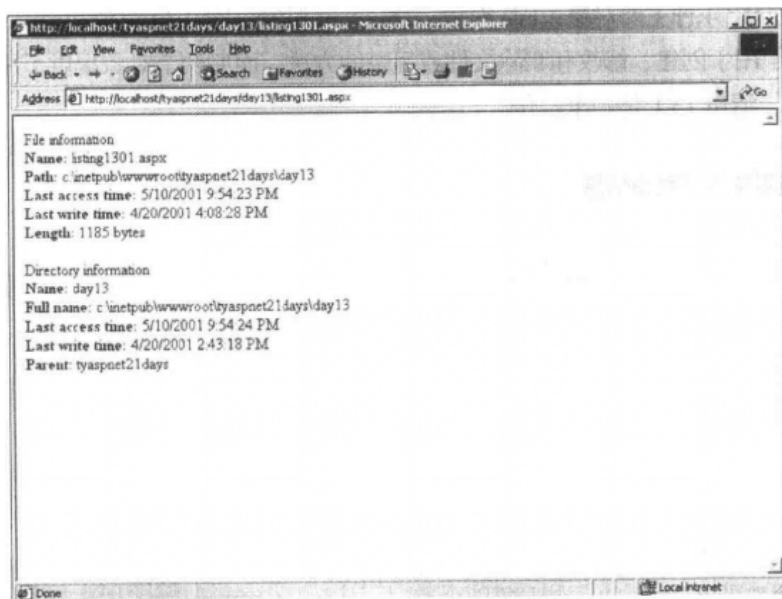


图 13.4 查看文件和目录的属性

分析：将该清单保存为 Listing1301.aspx。第 1 行导入名称空间 System.IO。第 5 行使用包含清单的文件实例化一个新的 FileInfo 对象（这确保文件是存在的）。第 4 章介绍过，Server.MapPath 将虚拟目录映射到物理路径。第 8–14 行列出该文件的所有属性。图 13.4 显示了各属性返回的值。

第 17 行使用 Directory 属性返回一个基于文件的父目录的 DirectoryInfo 对象。然后，在第 20–26 行显示该目录的属性。FileInfo 和 DirectoryInfo 对象的很多属性是类似的。

**警告：**如果指定的文件不存在，系统会返回一个错误。可使用 File 对象的 FileExists 方法来确定文件是否存在。与此类似，Directory 对象的 DirectoryExists 方法也可用于确定目录是否有效。

File 和 Directory 对象有一个 Attribute 集合，它提供了额外的信息，如文件是否是只读的或隐藏的。表 13.1 列出了这些属性。

表 13.1 文件和目录的属性及其值

属 性	值
ReadOnly (只读)	1
Hidden (隐藏)	2
System (系统)	4
Directory (目录)	16
Archive (存档)	32
Encrypted (加密)	64
Normal (普通)	128
Temporary (临时)	256
SparseFile	512
ReparsePoint	1024
Compressed (压缩)	2048
Offline (脱机)	4096
NotContentIndexed (非内容索引)	8192

每个文件中的这些值将累积。例如，对于文件 listing1301.aspx，语句 Response.Write (f.Attribute) 返回的结果将为字符串“32”，这表明该文件是存档文件。要改变文件属性，只需将值累加，并将 Attributes 属性设置为相应的值。例如，要将文件设置为隐藏、压缩、加密和不被索引，可以使用下面的语句：

```
f.Attribute = 10306
```

**警告：**不能直接设置属性值，例如语句 f.Attribute.Hidden = 2 将出错。Hidden 属性是一个常量，不能被修改。您必须直接修改 Attributes，如 f.Attribute = 2。

这些属性值难于记忆，所以可以直接使用其名称，并使用 BitOr 操作符分隔各个属性：

```
f.Attributes = FileAttributes.Hidden BitOr _
FileAttributes.Compressed BitOr _
```

```
FileAttributes.NotContentIndexed
```

上述代码相当于将 `Attributes` 的值设置为 10306。`BitOr` 是一种 VB.NET 按位操作符, 让您能够比较各位的值。其他 VB.NET 按位操作符包括 `BitAnd`、`BitNot` 和 `BitXor`。要确定文件的某个属性是否被设置, 可使用 `BitAnd` 操作符:

```
if f.Attributes BitAnd FileAttributes.Hidden > 0 then
    Response.write("Hidden")
end if
```

上述代码对于显示关于一个文件或目录的信息很有用。但如果要查看更多信息, 如某目录下的所有文件, 该如何办? `DirectoryInfo` 对象提供了两个方法, 它们返回文件或目录集合: `GetFiles` 和 `GetDirectories`。清单 13.2 是一个遍历这些集合的例子。

### 清单 13.2 遍历目录下的所有文件和目录

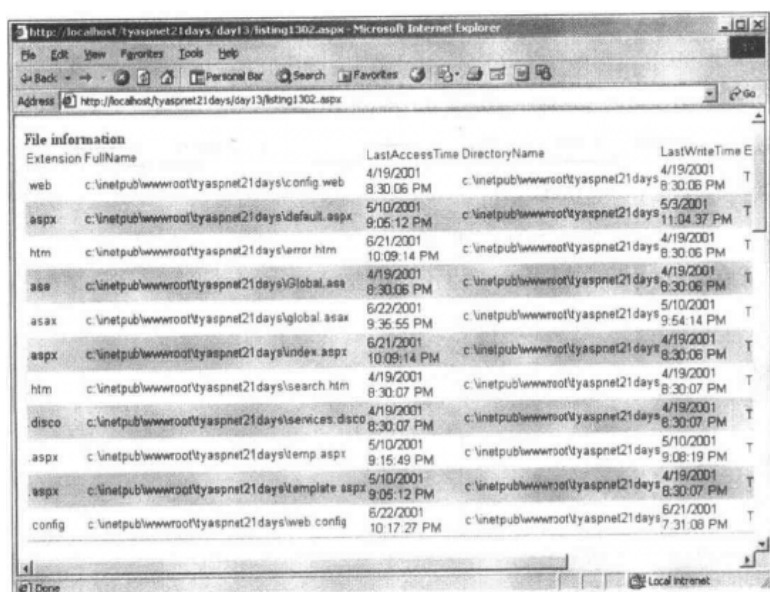
```
1 <%@ Import Namespace="System.IO" %>
2 <script language="VB" runat="server">
3     sub Page_Load(obj as object, e as eventargs)
4         dim dir as new DirectoryInfo(Server.MapPath _
5             ("/tyaspnet21days"))
6
7         DataGrid1.DataSource = dir.GetFiles("*.txt")
8         DataGrid1.DataBind()
9
10        DataGrid2.DataSource = dir.GetDirectories
11        DataGrid2.DataBind()
12
13    end sub
14
15 </script>
16
17 <html><body>
18     <b>File information</b><br>
19     <asp:DataGrid id="DataGrid1" runat="server"
20         width="100%"
21         Font-Name="Arial"
22         Font-Size="10pt"
23         AlternatingItemStyle-BackColor="#cccccc"
24         AutogenerateColumns="true" />
25
26     <p>
27         <b>Directory information</b><br>
28         <asp:DataGrid id="DataGrid2" runat="server">
```

```

29     width='100%'
30     Font-Name="Arial"
31     Font-Size="10pt"
32     AlternatingItemStyle-BackColor=" #cccccc"
33     AutogenerateColumns="true" />
34 </body></html>

```

分析：第 7 行使用 `GetFile ("*.*)` 返回当前目录下的所有文件，“\*.\*)”指的是任何两个由句点分隔的字符串（例如 `Listing1301.aspx`）。由于该方法返回一个集合，所以可以使用 `DataGrid` 来显示其中的条目。第 10 行使用 `GetDirectories` 返回当前路径下的目录集合，并将这些信息显示在 `DataGrid` 中。图 13.5 为清单的运行结果。



The screenshot shows a web browser window displaying a table of file and directory information. The table has four columns: Extension, FullName, LastAccessTime, and LastWriteTime. The data is as follows:

Extension	FullName	LastAccessTime	LastWriteTime
web	c:\inetpub\wwwroot\tyaspnet21days\config web	4/19/2001 8:30:06 PM	4/19/2001 8:30:06 PM
aspx	c:\inetpub\wwwroot\tyaspnet21days\default.aspx	5/10/2001 9:05:12 PM	5/3/2001 11:04:37 PM
htm	c:\inetpub\wwwroot\tyaspnet21days\error.htm	6/21/2001 10:09:14 PM	4/19/2001 8:30:06 PM
asa	c:\inetpub\wwwroot\tyaspnet21days\Global.asa	4/19/2001 8:30:06 PM	4/19/2001 8:30:06 PM
asax	c:\inetpub\wwwroot\tyaspnet21days\global.asax	6/22/2001 9:36:55 PM	5/10/2001 9:54:14 PM
aspx	c:\inetpub\wwwroot\tyaspnet21days\index.aspx	6/21/2001 10:09:14 PM	4/19/2001 8:30:06 PM
htm	c:\inetpub\wwwroot\tyaspnet21days\search.htm	4/19/2001 8:30:07 PM	4/19/2001 8:30:07 PM
disco	c:\inetpub\wwwroot\tyaspnet21days\services.disco	4/19/2001 8:30:07 PM	4/19/2001 8:30:07 PM
aspx	c:\inetpub\wwwroot\tyaspnet21days\temp.aspx	5/10/2001 9:15:49 PM	5/10/2001 9:08:19 PM
aspx	c:\inetpub\wwwroot\tyaspnet21days\template.aspx	5/10/2001 9:05:12 PM	4/19/2001 8:30:07 PM
config	c:\inetpub\wwwroot\tyaspnet21days\web.config	6/22/2001 10:17:27 PM	6/21/2001 7:31:08 PM

图 13.5 遍历文件和目录集合

也可以使用一个 `for` 循环来遍历集合，但 `DataGrid` 提供了一种更简单的机制来显示其中的条目。要遍历并显示所有的子目录及其包含的文件，可以使用一个递归函数。

让我们来开发一个更有用的例子。您将创建一个文件浏览器，用于显示硬盘的内容，就像 Windows 资源管理器一样。该页面允许用户在目录中导航（就像 Windows 中一样），用户还可以通过输入目录名来快速切换到该目录。清单 13.3 列出了该页面的 ASP.NET 代码。

#### 清单 13.3 创建一个文件导航系统——Listing1303.aspx

```

1 <% Import Namespace="System.IO" %>
2 <script language="VB" runat="server">
3     private dir as DirectoryInfo
4     private f as FileInfo
5     private strDir as string
6
7     sub Page_Load(obj as object, e as EventArgs)

```

```
8     if not Page.IsPostBack then
9         strDir = Request.Params("dir")
10
11         if strDir = "" then
12             strDir = "c:\\"
13         end if
14
15         tbDir.Text = strDir
16         dir = new DirectoryInfo(strDir)
17         ListFiles()
18     end if
19 end sub
20
21 sub tbDir_Handle(obj as object, e as EventArgs)
22     strDir = obj.Text
23     if Directory.Exists(strDir) then
24         dir = new DirectoryInfo(strDir)
25         ListFiles()
26     else
27         lblMessage.Text = "Invalid directory"
28     end if
29 end sub
30
31 sub ListFiles()
32     dim hl as HyperLink
33     dim d as DirectoryInfo
34     if not dir.Root.FullName = dir.FullName then
35         hl = new HyperLink
36         hl.Text = ".."
37         hl.NavigateURL = "listing1203.aspx?dir=" & _
38             Server.URLEncode(dir.Parent.FullName)
39         Panel1.Controls.Add(hl)
40
41         Panel1.Controls.Add(new LiteralControl("<br>"))
42     end if
43
44     for each d in dir.GetDirectories
45         hl = new Hyperlink
46         hl.Text = d.Name
47         hl.NavigateURL = "listing1303.aspx?dir=" & _
```

```

48         Server.URLEncode(d.FullName)
49         Panel1.Controls.Add(h1)
50
51         Panel1.Controls.Add(new LiteralControl('<br>'))
52     next
53
54     for each f in dir.GetFiles("*.")
55         lblMessage.Text += f.Name & "<br>"
56     next
57 end sub
58 </script>

```

分析：将该清单保存为 listing1303.aspx。用户首次访问该页面时，将看到根目录（这里为 C:\）中的目录列表。第 9 行检查 Request 的参数 dir，以确定当前目录。

如果参数 dir 为空（首次访问页面时就是如此），则使用默认目录 C:\。然后第 16 行为该目录创建了一个 DirectoryInfo 对象，第 15 行将文本框的 Text 属性设置为该目录（用于提供 UI），然后第 17 行调用 ListFiles 方法。

从第 31 行开始的 ListFiles 函数像以前所做的那样遍历当前目录的目录集合。但首先在第 34 行进行检查，以确定当前目录是否是根目录，如果不是，则显示“..”，让用户可以移到上一层目录。然后，创建一个 Hyperlink 控件来显示“..”，并将当前目录的父目录添加到查询字符串的最后面。当用户单击该链接时，出现相同的页面，并在查询字符串的最后加上参数 dir，该参数指定了当前的路径，这里为次最高目录。Page\_Load 事件使用参数 dir 创建一个新的目录对象。第 41 行创建了一个新的 LiteralControl 对象，用于在“..”后面换行。

对于每个目录，执行类似的操作：创建一个 Hyperlink 控件，让用户返回到该页面，Hyperlink 显示目录的缩写（如第 46 行所示），而您将完整的目录名加在查询字符串的最后面（如第 47 行所示）页面将使用该参数来显示新目录。第 51 行又添加了一个 LiteralControl 对象，以便在目录链接之间换行。

第 54–56 行遍历了目录中的所有文件，并将它们的名称添加到标签控件中。讨论最后一个方法 tbDir\_Handle 之前，先看看该页面的 UI 部分，如清单 13.4 所示。

**清单 13.4 目录列表页面的 UI 部分——Listing1304.aspx**

```

1 <html><body>
2   <form runat="server">
3     <b>Directory:</b>
4     <asp:Textbox id="tbDir" runat="server"
5       OnTextChanged="tbDir_Handle"
6       AutoPostBack=true /><p>
7     <asp:Panel id="Panel1" runat="server"
8       MaintainState="true" />
9     <asp:Label id="lblMessage" runat="server"
10      maintainstate=false />

```



```

11     </form>
12 </body> </html>

```

分析：该页面包含三个 UI 元素：一个用于显示当前目录的 TextBox 控件、一个用于容纳 HyperLink 控件的 Panel 和一个包含文件名及其他信息的标签控件。当文本框中的文本发生变化时（假设用户手工更改了目录），tbDir\_Handle 事件处理程序将被执行。

该处理程序（如清单 13.3 中第 21-29 行所示）检索文本框中的文本，并用它来设置新的目录。注意第 21 行的 if 语句，该语句用于确保用户输入的有效性。DirectoryExists 方法用于核实指定的路径是否存在。如果路径无效，将向用户发送一条错误消息。这种检查非常重要，对于这类应用程序更是如此。

最后，如果目录有效，则再次调用 ListFiles 方法来列出目录。图 13.6 为输出结果。

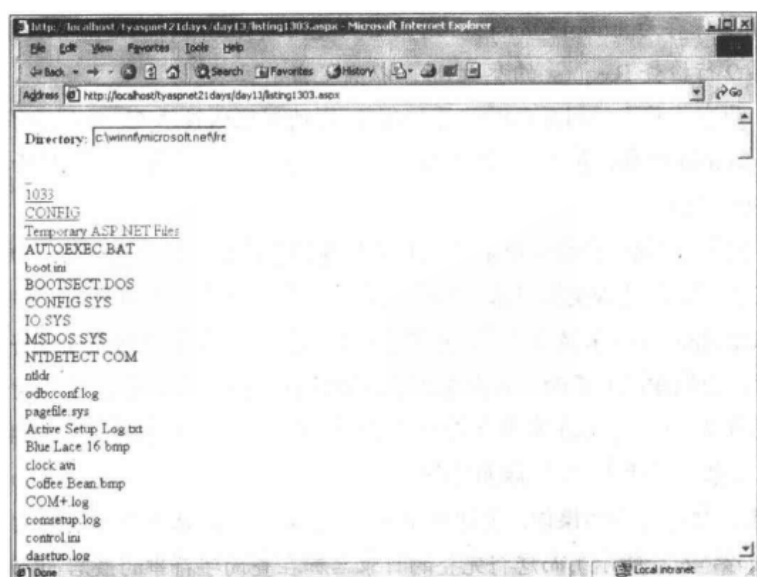


图 13.6 目录浏览程序列出的目录列表

该应用程序中有两项内容至今尚未讨论过。第一个是 Server.URLEncode，每当创建 Hyperlink 并指定查询字符串时，都使用 Server.URLEncode 来确保将所有特殊的字符转换为不会引起 URL 混乱的字符串。例如，如果路径名为 c:\Program Files\temp.txt，则遇到 Program 和 Files 之间的空格时将出错。可使用 Server.URLEncode 来将该字符串替换为：

```
C%3a%5cprogram+files%5ctemp.txt
```

所有会引起混乱的字符都被替换为 HTML 认为是替代字符的其他字符串。

第二个未讨论的问题是不同的页面调用方法。在 Page\_Load 事件中，您进行检查以确保页面没有提交过。如果没有，则意味着用户单击了 Hyperlink 控件（只是重定向，而不是提交表单），从而改变了当前目录，因此使用 Page\_Load 事件处理程序列出目录和文件。如果页面被提交过，则使用 thDir\_Handle 方法列出目录和文件。这种检查十分重要，如果不检查，目录可能会在页面中列出两次。

### 13.3.3 打开文件

根据情况的不同，在 ASP.NET 中打开文件的方法有多种。正如前面讨论过的，ASP.NET 允许您以二进制和 Unicode 存取模式打开文件。二进制存取模式可以直接读取计算机文件中的位（0 和 1）。

Unicode 模式将这些位转换为人类能够识别的东西，如字母和数字。对我们来说，二进制存取并不那么重要，所以这里重点介绍 Unicode 方法。

#### 1. 使用 File 对象

一种打开文件的方法是使用 File 对象。实例化该对象后，便可以根据如何处理数据来使用几种不同的方法来打开文件。表 13.2 对这些方法做了总结。

表 13.2 File 对象的 Open 方法

方 法	描 述
Open	按指定权限打开文件，返回一个 Stream 对象
OpenRead()	返回文件的只读流
OpenText()	从已有文件返回一个 StreamReader 对象
OpenWrite()	返回一个读写 Stream

关于这几个方法，有几点需要注意。首先，让我们来看看可用的权限类型。

Open 方法接受三个参数，依次为 FileMode、FileAccess 和 FileShare。FileMode 告知操作系统如何打开文件，例如，是覆盖原来的文件，还是追加数据。表 13.3 对可用的模式做了总结。

表 13.3 FileMode 的取值

模 式	描 述
Append	如果文件存在，则打开它，并移到文件的末尾；如果文件不存在，则创建一个新的文件。本模式仅适用于 Write 文件访问权限（Write 权限将稍后介绍）
Create	创建新文件或覆盖已有的文件
CreateNew	创建新文件
Open	打开已有的文件
OpenOrCreate	如果文件存在，则打开它，否则创建一个新文件
Truncate	打开已有的文件，将其长度截为 0 字节，相当于清除原有的内容

FileAccess 指定对文件的访问权限，可能的取值是 Read、ReadWrite 和 Write，分别对应于只读、读写和只写权限。

最后，FileShare 指定如何处理两个进程同时访问同一个文件的情况。例如，网站的两个访问者同时试图修改同一个文件。FileShare 的值将影响第二个进程的行为。可取的值与 FileAccess 相同，只是多了一个值：None，它规定第二个进程不能访问该文件。

您也许已注意到，除 OpenFile 外，File 类的所有方法都返回一个 Stream 对象，这使得只能进行二进制访问，这并不是我们希望的。我们将在讨论读取文件时介绍如何避免这种问题。

以下代码是一个使用 File 对象打开文件的例子。

```
`create a File object and StreamReader
```

```

dim objFile as new File(Server.MapPath('log.txt'))
dim objReader as StreamReader
`open the file
objReader = objFile.OpenText
`do some stuff
`close the StreamReader
objReader.close

```

或者使用 Open 方法:

```

dim objFile as new File(Server.MapPath("log.txt"))
dim objStream as Stream
objStream = objFile.Open(FileMode.OpenOrCreate, FileAccess.Read)

```

## 2. 使用 FileStream 对象

FileStream 对象让您能够创建 Stream 来访问文件。使用 FileStream 对象的好处之一在于,不必首先使用 File 对象,但必须确保要访问的文件存在。但使用 File 对象打开文件时,不需要这样做,因为如果创建 File 对象成功,则说明该文件确实存在。下面的代码说明了如何打开文件:

```

dim fs as new FileStream(Server.MapPath('log.txt'), _
    FileMode.OpenOrCreate)

```

上述代码使用 OpenOrCreate 指令打开一个文件。FileStream 对象使用的参数与 File.Open 相同,只是添加了文件路径。

FileStream 对象还提供了一个 Seek 方法,让您能够移到流的任何位置(如开头或结尾)。该方法的语法如下:

```
Seek (offset, origin)
```

Offset 参数指定移动距离, Origin 可以是下列 SeekOrigin 枚举值之一: SeekOrigin.Begin (流的开头)、SeekCurrent (当前位置)、SeekEnd (流的结尾)。例如,下面的代码片段移到文件的开始和结束位置:

```

`Seek to the beginning of a file
objFile.Seek(0, SeekOrigin.Begin)
`Seek to the end of a file
objFile.Seek(0, SeekOrigin.End)

```

### 13.3.4 读文件

打开文件后,便可以读取其内容了。对于 ASP.NET 来说,这项工作主要由 StreamReader 对象来完成。前面讨论过,该对象将二进制转化为 Unicode,从而提供页面中需要使用的字符和字符串。实例化 StreamReader 对象很简单:

```
dim objReader as new StreamReader(object)
```

object 参数可以是一个 Stream 对象(例如前面的代码中创建的 FileStream 对象),也可以是特定文件的路径。我们来创建一个简单文件,供范例使用。将下列代码放入 TYASPNET21Days/Day13 目录的 log.txt 文件中。

```

The Quick Foxes
Jumped Over

```

```
The Lazy
Brown Dog
```

`StreamReader` 提供了好几种读取文件中字符的方法。第一个方法是 `Read`，它以整数方式返回流中的一个字符，然后移到下一个字符。例如：

```
dim f as new File(Server.MapPath("log.txt"))
objReader = f.OpenText
Response.Write(objReader.Read & "<br>")
objReader.Close
```

上述代码返回 84，这是大写字母 T 的 ASCII 码。再次调用 `Read` 将返回 104，这是小写字母 h 的 ASCII 码。这些数字不是很有用，因此应使用 `String` 类的 `Chr` 方法将其转换为实际的 Unicode 字符：

```
Response.Write(string.Chr(objReader.Read) & "<br>")
```

第一次调用时，该语句返回字符 T，第二次调用时返回 h。

一次只读取一个字符有些麻烦，为此 `StreamReader` 提供了另外两个方法：`ReadLine` 和 `ReadEnd`，用于每次返回多个字符。前者读取字符直到行尾，后者直到文件流的结尾。例如：

```
objReader = new StreamReader(Server.MapPath("log.txt"))
Response.Write(objReader.ReadToEnd)
objReader.Close
```

将返回：

```
The Quick Foxes Jumped Over The Lazy Brown Dog
```

对于换行符，如何处理呢？如果查看输出的 HTML 源代码，您会看到换行符仍在。不幸的是，HTML 不将换行符解释为 HTML 的换行符（即 `<br>`）。您可以使用 `ReadLine` 方法插入换行符：

```
Response.Write(objReader.ReadLine & "<br>")
```

然而，使用该方法每次只能读取一行，所以必须使用循环来遍历整个文件。使用 `ReadLine` 或 `Read` 时，到达文件末尾时，如果再试图读取下一个字符，将出错。幸运的是，`StreamReader` 对象提供了一个 `Peek` 方法，可帮助您避免这种错误。

`Peek` 方法读取下一个字符，但并不将其返回。这样让您能够查看流中的下一个字符。如果该字符为流尾，则 `Peek` 返回 -1，这样，便可以遍历文件流而无需任何担心。

```
dim objReader as new StreamReader(Server.MapPath("log.txt"))
while objReader.Peek() > -1
    Response.Write(objReader.ReadLine & "<br>")
end while
objReader.Close
```

图 13.7 是上述代码的运行结果。

**警告：** 由于 `Peek` 方法实际上并不读取流中的数据，所以 `StreamReader` 并不会前进。一定要使用 `StreamReader` 的方法来推进 `StreamReader` 在流中的位置，否则将导致死循环。例如：

```
while objReader.Peek() > -1
    Response.Write("<br>")
end while
```

将导致死循环，因为 `StreamReader` 在流中的位置没变。

`Read` 方法还可用于从流中返回指定数目的字符，其语法如下：

```
ObjReader.Read(char(), start, count)
```

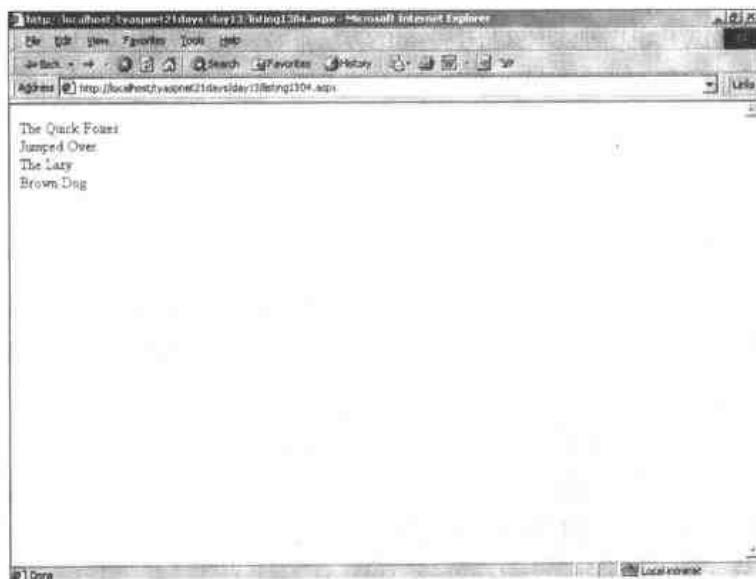


图 13.7 使用 ReadLine 方法读取流

该方法从 **start** 指示的位置开始，从流中读取 **count** 个字符，并将结果放入字符数组 **char()** 中。

例如：

```
objReader = new StreamReader(Server.MapPath("log.txt"))
dim arrString() as Char = new Char(10)
objReader.Read(arrString, 0, 10)
for i = 0 to (bound(arrString) - 1)
    Response.Write(arrString(i))
next
objReader.Close
```

第 2 行创建了一个字符数组 **arrString**。第 4 行从位置 0（流的开头）开始读取 10 个字符，并将其放入 **arrString** 数组中。返回的结果为：

```
The Quick
```

返回的结果为 8 个字符和一个空格（k 为结尾）。

假设出于教育的目的（例如教授一种特殊的方法），您想将一个 ASP.NET 页面的源代码显示给访问者。当访问者在浏览器中查看 ASP.NET 页面的源代码时，看到的是 HTML 代码——所有的 ASP.NET 命令都被处理并删除。因此，您得在 ASP.NET 页面被处理前将其显示给访问者。可使用 **System.IO** 类来显示 ASP.NET 源代码，清单 13.5 说明了如何做。

#### 清单 13.5 在浏览器中查看 ASP.NET 源代码

```
1 <%@ Import Namespace="System.IO" %>
2 <script language="VB" runat="server">
3     sub Page_Load(obj as object, e as EventArgs)
4         dim fs as new FileStream(Server.MapPath _
5             ("listing1305.aspx"), FileMode.Open,
```

```

6 FileAccess.Read)
7     dim objReader as new StreamReader(fs)
8
9     lblMessage.Text = "<pre>"
10    while objReader.Peek() > -1
11        lblMessage.Text += Server.HtmlEncode _
12            (objReader.ReadLine) & "<br>"
13    end while
14    objFeader.Close
15    fs.close
16    lblMessage.Text += "</pre>"
17 end sub
18 </script>
19
20 <html><body>
21     <form runat="server">
22         <asp:label id="lblMessage" runat=server />
23     </form>
24 </body></html>

```

**分析：**在 Page\_Load 事件处理程序中，您根据要查看的文件创建了一个 FileStream 对象。第 7 行创建了一个 StreamReader 来读取流的内容。<pre>标记让您能够保留源代码的格式。然后，您使用 Peek 和 ReadLine 方法来遍历文件中所有的行，并确保调用 Server.HtmlEncode，避免 HTML 标记被解释。最后，您关闭 StreamReader 和 FileStream 对象，并将结束标记</pre>写到标签中，该清单的运行结果如图 13.8 所示。

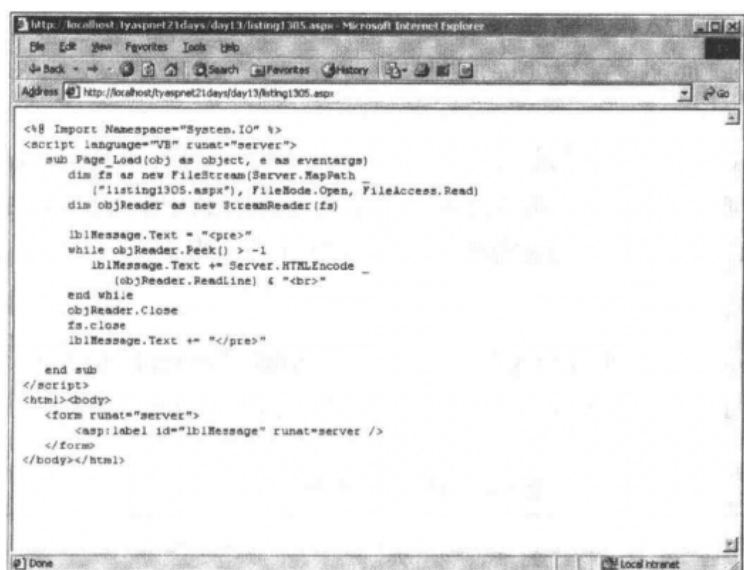


图 13.8 在浏览器中查看 ASP.NET 源代码

13.3.5 写文件

StreamWriter 对象让您能够将数据写入流中。StreamWriter 的语法很简单。同 StreamReader 对象一样，可以使用 Stream 对象或指定一个文件的路径来创建 StreamWriter。

```
dim objWriter as new StreamWriter(FileStream, Append)
```

Append 参数是一个布尔值，指定是否将数据追加到文件的最后面。如果为 false，则原来的文件将被覆盖；如果文件不存在，则创建该文件；如果 Append 为 true，则写入的数据将被添加到原来内容的后面。

StreamWriter 使用两个方法来写数据：Write 和 WriteLine。例如：

```
dim objWriter as new StreamWriter(Server.MapPath _  
    ("log.txt"), true)  
objWriter.WriteLine("And I care because? ")  
objWriter.Write("because I say so.")  
objWriter.Close
```

第 1 行使用文件 log.txt 创建了一个 StreamWriter 对象，以便在 log.txt 文件中追加数据。第 4-5 行使用两种不同的方法将数据写入到文件中，第 7 行关闭 StreamWriter 对象。上述代码执行后，log.txt 文件的内容将如下所示：

```
The Quick Foxes  
Jumped Over  
The Lazy  
Brown DogAnd I care because?  
because I say so.
```

调用 WriteLine 将添加一个换行符。然而，新添加的文本与原有文本之间并没有换行。要添加换行符，只要不带参数调用 WriteLine 即可：

```
objWriter.WriteLine()
```

StreamWriter 对象支持缓冲输出。还记得第 4 章关于缓冲页面的讨论吗？默认情况下，使用 Write 和 WriteLine 方法写入数据时，StreamWriter 对象在将数据发送给流之前，先将输出保存在缓冲区中。您可以将 AutoFlush 属性设为 True，强制 StreamWriter 在每次调用 Write 或 WriteLine 后刷新其缓冲区。

还可以调用 Flush 方法来刷新缓冲区。将 AutoFlush 设为 false 后，性能会稍有提高。但如果用户期望尽快得到反馈，可将 AutoFlush 设为 true。例如，当您向文件中添加了大量内容，并希望用户在内容处理结束前能够看到文件的变化时，可将 AutoFlush 设置为 true。

13.3.6 其他文件和目录操作

File 和 Directory 对象分别提供了复制、创建、移动和删除文件和目录的方法。这些方法都很简单。表 13.4 总结了这些方法。如果没有特别声明，则方法适用于这两个对象。

表 13.4 文件和目录的其他方法	
方 法	描 述
Directory.CreateDirectory	在指定路径中创建所有的目录

续表

方 法	描 述
Directory.CreateDirectory	在指定的路径下创建一个目录。如果指定的路径不存在, ASP.NET 将引发错误。
Directory.Delete	删除一个目录。Recursive 是一个布尔值, 指定是否删除其中的子目录和文件。
File.Copy	将文件复制到一个新文件中。Overwrite 指定是否覆盖已有的文件。
File.Create	在指定路径下创建一个新文件。
File.CreateText	创建一个 StreamWriter 对象, 用于将数据写入到一个新的文本文件中。
File.Delete	删除一个文件。
File.ChangeExtension	修改指定路径下文件的扩展名。将 extension 设为 nothing 将删除扩展名。扩展名中必须包含“.”。该静态函数返回具有新扩展名的文件路径。
File.GetExtension	该静态函数返回一个文件扩展名。
File.HasExtension	一个静态函数, 如果指定的文件有扩展名, 则返回 true。
Move	将指定文件或目录从 oldpath 移到 newpath。

要修改文件的名称或路径, 可使用 File.Move 方法。也可以使用 Copy 方法来修改文件的名称, 但别忘了在复制后, 删除原来的文件。

### 13.3.7 文件对象小结

在 ASP.NET 中, 访问文件的方法很多, 因此在何时使用何种对象方面, 您可能感到困惑。表 13.5 总结了各对象及其用途。

表 13.5 System.IO 对象和枚举小结总结

对 象	何 时 使 用
BinaryReader 和 BinaryWriter	从流中读取或向流中写入二进制数据 (例如, 数据类型和对象)
Directory	创建、删除和操纵目录
File	创建、删除和操纵磁盘上的文件
FileInfo 和 DirectoryInfo	查看或操纵文件和目录的属性
FileAttributes 和 DirectoryAttributes	确定文件或目录的属性
FileMode、FileAccess 和 FileShare	指定打开文件的权限和行为
FileStream	访问文件, 能够定位到文件中的任何位置
MemoryStream	访问内存中的流
SeekOrigin	为 FileStream.Seek 方法指定起点
StreamReader 和 StreamWriter	从流中读取或向流中写入字节或 Unicode 字符
StringReader 和 StringWriter	读写字符串, 功能与 StreamReader 和 StreamWriter 相同



## 13.4 隔离存储区

使用本章介绍的方法，可以将数据存储在特定路径下的文件中，如 `c:\inetpub\wwwroot\tyaspnet21days\day13\log.txt`。该数据存储方法既强大、又有用，但存在一些缺点。

首先，您必须知道每个文件的路径，或者为每个文件创建一个唯一性的路径。在本章介绍的情况下，这不是什么大问题，但想象一下，如果创建的应用程序要分发到不同的服务器中，而每个服务器的文件系统结构各不相同，情况将会如何？要所有这些不同的文件位置，无疑是件苦差事。其次，创建的文件易为它人（包括 Web 访问者和能够接近计算机的人）或其他能够访问该文件的应用程序所破坏。对于记录页面计数器的文件来说，这至关重要，如果每个人都能访问其内容，如何保存个人信息？要记录信息的归属将十分困难。

ASP.NET 的隔离存储机制能够解决这些问题。就根据用户来隔离数据而言，隔离存储与 cookie 类似。每一个用户都有自己的隔离存储区，使用隔离存储，您无需考虑存储位置的问题，也无需担心唯一性的路径名问题。隔离存储可将数据存储在任意地方，甚至可以同 cookie 一样，存储在客户计算机上。

### 13.4.1 创建隔离存储区域

对于隔离存储，有两个对象值得注意：`IsolatedStorageFile` 和 `IsolatedStorageFileStream` 对象，它们类似于本章前面介绍的 `File` 和 `FileStream` 对象。实际上，`IsolatedStorageFileStream` 与 `FileStream` 对象包含相同的方法和属性。

然而，不同于 `File` 对象，`IsolatedStorageFile` 对象并不表示文件，而表示存储位置。存储位置中的文件由 `IsolatedStorageFileStream` 对象表示。开始的时候这有些令人混淆，所以弄明白这些对象很重要。

清单 13.6 说明了如何在隔离存储区中创建文件。

**清单 13.6 创建隔离存储文件**

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.IO" %>
3 <%@ Import Namespace="System.IO.IsolatedStorage" %>
4
5 <script runat="server">
6     sub Page_Load(obj as object, e as eventargs)
7         dim stream as IsolatedStorageFileStream
8         dim writer as StreamWriter
9         dim data as string = "blue"
10
11         stream = new IsolatedStorageFileStream("colors.txt", _
12             FileMode.OpenOrCreate)
13         writer = new StreamWriter(stream)
14         writer.WriteLine(data)
```

```

15     writer.Close()
16 end sub
17 </script>
18
19 <html><body>
20 </body></html>

```

分析：注意第 3 行的新名称空间 `System.IO.IsolatedStorage`，别忘了将该名称空间包含到页面中。第 7-9 行声明了三个变量，其中包括新的 `IsolatedStorageFileStream` 对象。您应该早已熟悉第 8 行的 `StreamWriter` 对象。

接下来的代码也不陌生——唯一的新内容是 `IsolatedStorageFileStream` 对象，创建该对象的语法与创建 `FileStream` 对象相同。然后，第 13-14 行使用 `StreamWriter` 执行了我们业已熟悉的操作，第 15 行关闭了 `Writer`，同时关闭了 `IsolatedStorageFileStream`。

上述代码在一个隔离存储区中创建了一个名为 `Colors.txt`，供当前用户专用。其他用户不能访问该文件，就像不能访问其他人的 `Cookie` 一样。该文件的存储位置随使用的操作系统而异，如表 13.6 所示。

表 13.6 默认的隔离文件存储位置

操作系统	位 置
Windows 95/98 或 Me	C:\Windows\Local Settings\Application Data
Windows NT 4.0	C:\WinNT\Profiles\user\A\Application Data
Windows NT 4.0 (Service Pack 4) 和 Windows 2000 (从 NT 4.0 升级)	C:\WinNT\Profiles\user>\Local Settings\Application Data
Windows 2000 (新安装或从 Windows 95、98、ME 及 WinNT 3.51 升级)	C:\Documents and Settings\user>\Local Settings\Application Data\Microsoft

例如，在 Windows 2000 (新安装)上执行清单 13.6 后，切换到目录 `C:\Documents and Settings\Default user\Local Settings\Application Data\Microsoft`，您将看到一个名为 `IsolatedStorage` 的新文件夹。该文件夹包含其他几个文件夹（ASP.NET 使用这些文件夹来记录发生的情况）和第 11 行创建的文件。通常，您不会关心这些目录中发生的情况，但清楚文件的物理位置将很有帮助。

#### 13.4.2 访问隔离存储区

对隔离存储数据进行隔离的方法有两种据：一是按用户（`user`）、组合体（`assembly`）；二是按用户组合体和域（`domain`）。第一种方法意味着为用户使用的每一个程序创建一个存储区。例如，用户 A 不能在应用程序 B 中访问他在应用程序 A 中创建的存储文件。第二种方法还加入了应用程序域。对 ASP.NET 而言，应用程序域是该应用程序的 URL 地址。

第 2 章介绍过，每个 ASP.NET 页面都将被编译为一个动态生成的组合体。也就是说，通常情况下，不同的 ASP.NET 页面将使用不同的组合体，这意味着不同的 ASP.NET 页面不能访问同一个隔离存储区。图 13.9 说明了这种概念。

可以使用 `IsolatedStorageFileStream` 来读取隔离存储区，清单 13.7 说明了这一点。

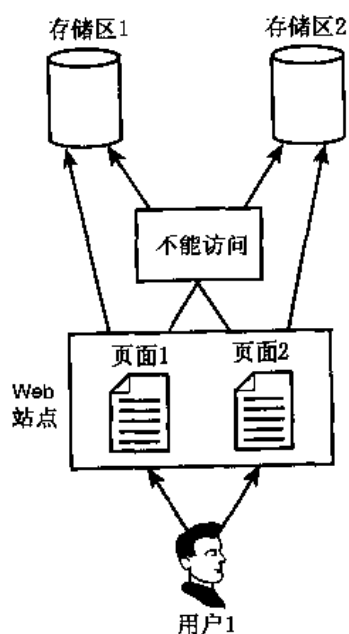


图 13.9 访问应用程序中不同 ASP.NET 页面的同一个用户不同访问相同的存储区

## 清单 13.7 使用 IsolatedStorageFileStream 读取隔离存储区

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace='System.IO' %>
3 <%@ Import Namespace='System.IO.IsolatedStorage' %>
4
5 <script runat='server'>
6   sub Page_Load(obj as object, e as eventargs)
7       dim stream as new IsolatedStorageFileStream _
8           ("colors.txt", FileMode.OpenOrCreate)
9       dim objReader as new StreamReader(stream)
10
11       while objReader.Peek() > -1
12           Response.Write(Server.HtmlEncode _
13               (objReader.ReadLine) & "<br>")
14       end while
15       objReader.Close
16   end sub
17 </script>
18
19 <html><body>
20   <asp:Label id="lblMessage" runat="server" />
21 </body></html>

```

该清单看起来很熟悉，因为其中没有新的方法。这个清单将输出 blue，该单词是在上个清单中

存储到存储区的。

**注意：**您或许看不到该清单的输出结果。还记得吗，不同的ASP.NET页面不能访问相同的隔离存储区。因此，清单13.7应该不能访问清单13.6的存储区。将清单13.6和清单13.7合并到一起后，这个问题就解决了（清单13.6和清单13.8也是如此）。

使用 `IsolatedStorageFile` 对象读取隔离存储区稍微有些不同。首先您必须获得到隔离存储区中文件的引用，然后便可以使用 `IsolatedStorageFileStream` 对象来处理其中的文件，如清单 13.8 所示。

**清单 13.8 使用 `IsolatedStorageFile` 对象读取隔离存储区**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.IO" %>
3 <%@ Import Namespace="System.IO.IsolatedStorage" %>
4
5 <script runat="server">
6   sub Page_Load(obj as object, e as EventArgs)
7     dim objISOFile as IsolatedStorageFile = _
8       IsolatedStorageFile.GetUserStoreForDomain()
9
10    lblMessage.Text = '<b>Files:</b>'
11    dim intCount = Ubound(objISOFile.GetFilesNames('*.*'))
12    for i = 0 to intCount
13      lblMessage.Text += objISOFile.GetFilesNames _
14        {"*.*")(i) & "<br>"
15    next
16
17    lblMessage.Text += "<br>Assembly: </b>" & _
18      objISOFile.AssemblyIdentity.ToString & "<br>"
19    lblMessage.Text += "<b>Domain: </b>" & _
20      objISOFile.DomainIdentity.ToString & "<br>"
21    lblMessage.Text += "<b>Current Size: </b>" & _
22      objISOFile.CurrentSize.ToString & "<br>"
23    lblMessage.Text += "<b>Max Size: </b>" & _
24      objISOFile.MaximumSize.ToString & "<br>"
25  end sub
26 </script>
27
28 <html><body>
29   <asp:Label id="lblMessage" runat="server" />
30 </body></html>

```

**分析：**第 7 行实例化了一个 `IsolatedStorageFile` 对象。`IsolatedStorageFile.GetUserStoreForDomain` 方法返回当前用户、组合体和域专用的 `IsolatedStorageFile`。`IsolatedStorageFile.GetUserStoreForAssembly` 方法

返回当前用户、组合体专用的存储区。

`GetFileNames` 方法返回由与指定搜索字符串（这里为 “\*.\*”）匹配的文件名称组成的数组。第 12-15 行遍历该数组并将名称显示到标签控件上。`GetDirectoryNames` 方法的功能类似，也使用一个搜索字符串来过滤结果。

最后，第 17-24 行输出隔离存储区的属性。`AssemblyIdentity` 属性返回一个 URL，该 URL 指定了该存储区对应的组合体（即 .NET 为 Web 应用程序生成的动态组合体）。`DomainIdentity` 指定了与该存储区相关的域——类似于 cookies 的域属性。`CurrentSize` 和 `MaxSize` 属性指定了用户存储区的大小（单位为字节）。图 13.10 是该清单的运行结果。

图 13.10 显示了 ASP.NET 为应用程序生成的动态组合体、域（Web 站点域）和存储区的当前和最大长度。

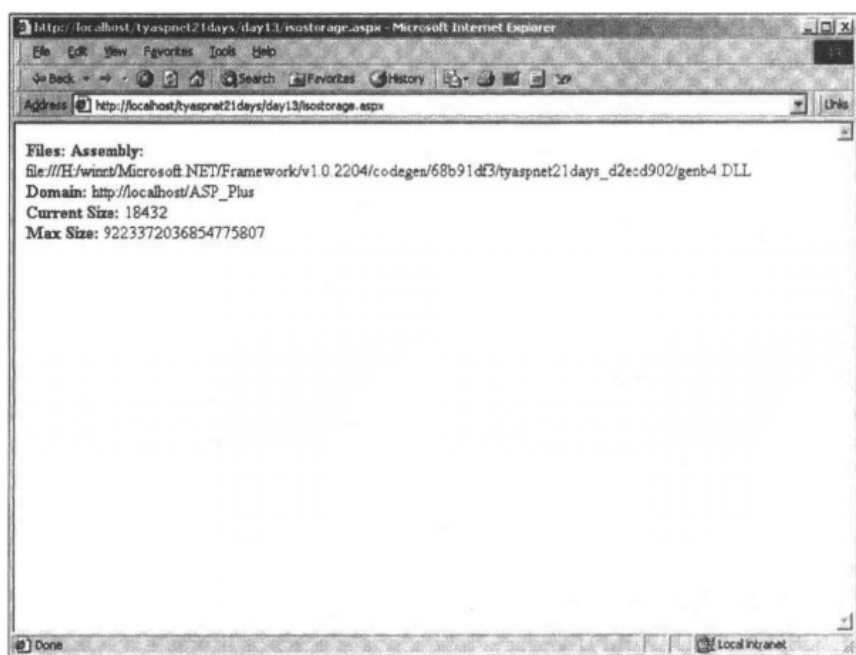


图 13.10 使用 `IsolatedStorageFile` 对象查看隔离存储区的信息。

可以为 `IsolatedStorageFile` 创建一个 `IsolatedStorageFileStream`，并使用 `StreamReader` 来存取其中的内容，就像前面介绍的那样：

```
dim objISOFile as IsolatedStorageFile = _  
    IsolatedStorageFile.GetUserStoreForDomain()  
  
dim stream as new IsolatedStorageFileStream _  
    ("options.txt", FileMode.OpenOrCreate, objISOFile)
```

## 13.5 这不是 ASP

传统 ASP 中文件 I/O 操作是通过 `FileSystem` 对象完成的。该对象提供的许多功能与本章介绍的相同。ASP.NET 能够访问所有的 .NET 框架类，也就是说，与传统 ASP 不同，ASP.NET 可以像独立

应用程序一样来完成 I/O 操作。

幸运的是, ASP.NET 中的许多概念甚至语法都与传统 ASP 相同, 虽然有几个对象的名称不同, 一些底层功能有所改变, 但熟悉 `FileSystemObject` 对象的开发人员在转向新的模式时不会有太大的困难。

ASP 与 ASP.NET 间最大的变化之一在于, ASP.NET 采用了完全面向对象的设计。每个文件和目录可使用 .NET 框架的对象来表示, 同时每个对象又有自己的方法、属性和对象集合。同 `FileSystemObject` 松散的结构相比, 要掌握这些关系密切的对象的确需要一些时间。

在 ASP.NET 中, 包含文件也有些不同。由于 ASP.NET 同 .NET 框架交互的方式, 因此 ASP.NET 应用程序必须包含其他文件中的信息。传统的服务器端包含方法仍然适用, 但现在还拥有很多其他的文件包含方法, 以扩展 ASP.NET 页面。

最后, 隔离存储完全是一种全新的数据存储和文件访问机制, 它解决了开发人员开发定制的用户存储解决方案时面临的问题。除 cookie 或数据库外, 传统 ASP 没有提供其他的内置方法来为各个用户保存永久性数据。为什么还要使用 cookie 呢? 与 cookie 相比, 使用隔离存储区能够存储更多的信息, 这让您的 Web 站点个性化更强。另外, 隔离存储区易于部署, 它可以位于服务器上, 也可以位于客户端, 并提供了更强大的安全措施。

隔离存储区与数据库孰优孰劣, 众说纷纭。具体采用哪种机制取决于具体情况和服务器的配置。然而, 两种方法都为永久性存储数据提供了可行的机制。

## 13.6 总 结

本章介绍了有关 ASP.NET 如何访问文件的所有内容。ASP.NET 允许您将其他文件包含到页面中, 以便在页面上使用其他文件来扩充页面的功能以及读写服务器上的文件。

可以使用服务器端包含语法来将其他文件包含到 ASP.NET 页面中:

```
<!--#include file = "filename"-->
```

或

```
<!--#include Virtual = "filename"-->
```

该方法适用于那些不需要使用用户控件的情况。

文件是存储在硬盘上的物理对象, 由二进制数据组成。流是让您能够访问这些文件以及其他数据存储区域 (如内存单元或网络计算机) 的对象。System.IO 名称空间提供了这里描述的所有对象。

`File` 和 `Directory` 对象可以访问文件和目录的属性, 并且可以操纵文件和目录, 即复制、删除、移动等。

每个对象的 `Attribute` 集合都列举了诸如存档、隐藏或索引等属性。

`FileShare`、`FileAccess` 和 `FileMode` 指定打开文件时使用的权限, 如读写权限、是新建文件还是打开已有的文件, 以及文件被打开后, 其他应用程序对其访问权限。

`StreamReader` 和 `StreamWriter` 对象通过将 Unicode 字符转换为二进制数据, 让您能够将 Unicode 字符写入到流中。`Read`、`ReadLine`、`ReadToEnd` 和 `Peek` 方法让您能够读取流的内容, 而 `Write` 和 `WriteLine` 方法让您能够将数据写入到流中。

隔离存储为将数据存储到用户、组合体或域专用的文件中提供了一种机制, 从而无需确定唯一性的路径名。这与 Cookie 类似, 得以实现高度个性化的用户数据存储。`IsolatedStorageFile` 对象表示特定用户专用的虚拟存储区, `IsolatedStorageFileStream` 对象表示存储区中的文件, 后者同 `StreamReader`

和 StreamWriter 对象搭配使用可以读写文件

下一章将介绍数据库文件 I/O 设计中常用的概念：缓存技术（Caching）。ASP.NET 提供了一种健壮的缓存机制，该机制很有趣。下一章见！

## 13.7 问与答

问：包含文件是否会增大文件？

答：文件增加的大小只是 include 命令。例如，语句 `<!--#include file = "log.txt" -->` 将使文件增大几个字节，因为您得输入这几个字符。由于在 ASP.NET 页面被处理之前，文件并没有真正包含进来，所以文件不会增大。

然而，当用户请求该页面时，所有的包含文件都将被插入到调用页面中，因此文件将增大，这可能会影响客户的下载时间，这取决于包含文件的大小。

问：文件访问权限如何影响文件 I/O？

答：要在 ASP.NET 中访问文件，首先必须拥有合适的访问权限。例如，假设您使用的计算机归自己所有，并且是直接访问这些文件（例如，不是通过 Internet），您应该有合适的权限，因为文件是您自己的。

如果是通过 Internet 访问其他服务器上的文件，则有很多文件您可能无法访问。在第三部分介绍安全方面的知识时将进一步讨论这个问题。

## 13.8 作业

下面的作业帮助巩固本章介绍的概念，继续阅读下一章前，完全理解其答案将很有帮助。答案见附录 A。

### 13.8.1 小测验

1. 文件和流有什么差别？
2. FileMode 的 6 种取值分别是什么？
3. Peek 方法有什么功能，为什么它很有用？
4. 如何验证服务器上的文件或目录是否存在？
5. 使用 File 对象打开根目录 CA 下的文件 userdata.txt，以追加数据。如果该文件不存在，则使用 File 对象创建它，并确保在您使用该文件时，其他应用程序不能访问它。

### 13.8.2 练习

1. 创建一个 ASP.NET 页面，该页面打开用户在文本框中指定的文件，并将其内容显示在一个标签上。一定要检查该文件是否存在，同时必须要求用户输入文件的全部路径。
2. 修改练习 1，将文件内容显示在一个可编辑的文本框中，让用户能够修改其中的内容，当用户单击提交按钮时，将所做的修改写入文件中。

## 第 14 章

# 使用 ASP.NET 改良后的缓存功能

本部分介绍了多种不同的数据访问方法，第 8、9、10、12 章介绍了如何使用数据库，第 11 章介绍了 XML，第 13 章介绍了文件的存取。本章将继续讨论数据存取方面的内容，介绍取得数据后，如何处理它们。

缓存技术（Caching）是计算中常用的技术，它通过将经常存取的数据存储在内存中来提高性能。ASP.NET 为缓存数据提供大量机制。本章将介绍以下内容：

- 什么是缓存技术，为什么要使用这种技术；
- ASP.NET 如何使用缓存技术；
- 如何缓存页面和对象；
- 如何使用缓存依存性（dependencies）；
- 操纵缓存的方式；
- 如何有效地使用缓存技术。

### 14.1 什么是缓存技术

缓存是计算机快速地再次取得数据的方式。其原理是将经常被访问的数据（甚至那些不那么经常被访问的数据）存储到计算机可以更快、更容易地读取的位置。缓存是一个信息容器，使得信息更容易被存取。图 14.1 说明了缓存带来的好处。

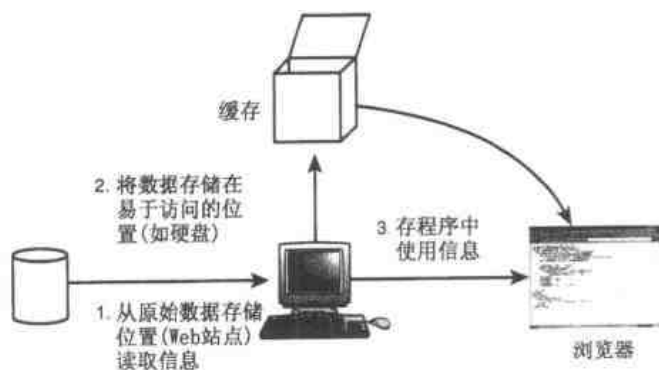


图 14.1 缓存是一种很容易访问的数据存储区



如果您使用过 Web 浏览器,则应该对缓存不会感到陌生。当访问页面时,浏览器将把该页面(和其中的图像)缓存到硬盘中的一个特殊位置。这样做有两个好处,首先,当您下次访问该页面时,浏览器仅从硬盘上载入它,而无需再次下载该页面(和其中的图像)。其次,您可以在脱机情况下浏览页面(和其中的图像),而无需重新访问该页面。这说明了缓存的两个最重要方面:快速存取、存取通常情况下无法存取的数据。

**新术语:** 这引出一个有趣的问题:如果缓存的数据太旧,情况将如何? Web 浏览器是否使用最新的内容更新页面或图像?由于缓存是计算机首先考虑的地方,所以显示的将是旧的内容。在这种情况下,应将缓存的内容作废——其可用期已过。

## 14.2 ASP.NET 如何使用缓存技术

ASP.NET 内置了多种缓存机制。对 Web 应用程序来说,缓存技术极其重要,因为通过 Internet 传输数据的速度可能非常缓慢。通过缓存数据,ASP.NET 极大地提高了其应用程序的性能。ASP.NET 还让开发人员能够根据需要修改缓存设置。

对于 ASP.NET 中的数据,缓存位置有两种:客户端和服务端。浏览器缓存是在客户端进行的——所有数据都被存储在用户的计算机中,并由浏览器发起和管理。服务器端缓存由服务器使用服务器资源进行管理,浏览器无法控制服务器端缓存。

本章重点介绍服务器端缓存。您将发现,服务器端缓存比客户端缓存提供了更大的灵活性和好处。图 14.2 说明了这两种缓存技术之间的差别。

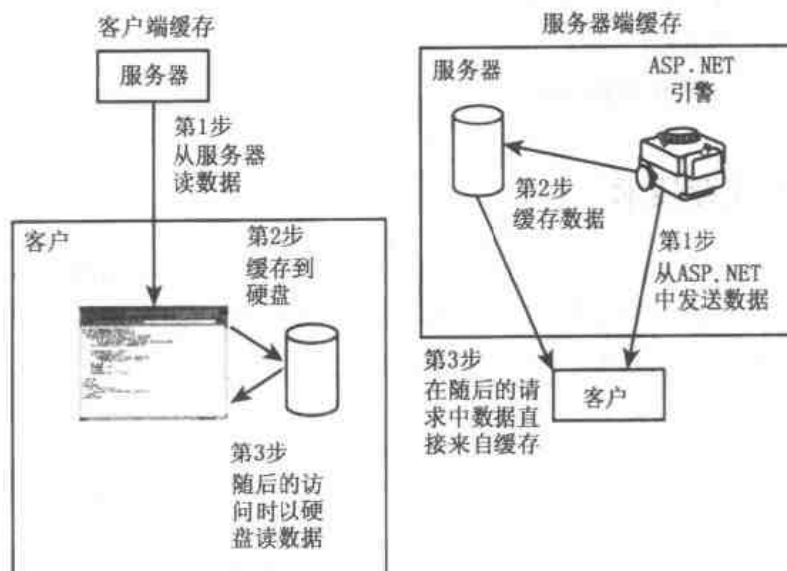


图 14.2 服务器端缓存和客户端缓存之间的差别

您还将发现,可缓存的不仅仅是图像和 Web 页面,还包括众多的数据类型。

### 14.2.1 页面缓存

所有的 ASP.NET 页面在被请求时都被编译。如果您在编译型语言环境下开发过应用程序,则

肯定知道,编译程序需要时间和计算能力。如果 ASP.NET 页面在每次被请求时都需要重新进行被编译,则 ASP.NET 应用程序的性能将极大地降低。

幸运的是,ASP.NET 页面被编译后,将被缓存到服务器中。服务器给客户提供的正是编译后的页面。由于需要编译,用户首次请求页面时可能需要等待较长的时间,但随后对该页面的请求将快得多。

运行阶段通用语言监视源文件的修改情况。如果您修改了原来的.aspx 文件,ASP.NET 将知道这一点,将当前缓存的页面作废并编译和缓存新的页面。

因此,通过使用编译后的页面,而不是每次重新编译页面,ASP.NET 提高了页面的性能。

#### 14.2.2 配置缓存

如果按本书的介绍开发 ASP.NET 应用程序,您可能还将发现另一个影响性能的因素。应用程序初次启动时—即首次请求某一个虚拟目录中的页面时—ASP.NET 必须装载 web.config 中的所有配置设置(参见第 18 章)。这需要时间,其长短取决于应用程序的复杂程度。

我们来看一下系统的 web.config 文件(通常位于目录 C:\WINNT\Microsoft.NET\Framework\version\config.web 中)。该文件相当复杂,通常超过 450 行!应用程序首次启动时,将缓存这些配置设置,这使得 ASP.NET 能够比不使用缓存技术时更快地将这些配置信息应用到文件中。

#### 14.2.3 输出和数据缓存

页面输出缓存(Page output caching)指的是缓存所有由动态页面生成的内容,如计算结果,甚至 DataGrid。输出缓存不会自动进行,但易于使用。本章后面的“如何使用缓存”一节将做详细的介绍。

**注意:** 不要将缓存与缓冲搞混了。您使用缓冲区来暂时存储执行过程中的输出,执行结束时缓冲区将被清空。而缓存可以保存数据很长一段时间,供以后使用。对初学者来说,这两种数据存储方法很容易混淆。

数据缓存让您能够将任何类型的数据存储在高速缓存中,从数据库记录到用户控件。数据缓存类似于输出缓存,但可缓存的内容不仅仅是页面生成的内容。该方法不会自动发生,所以需要缓存哪些内容由您决定。

### 14.3 如何使用缓存

ASP.NET 让您能够根据应用程序的需要控制输出缓存和数据缓存。每种缓存方法提供的机制稍有不同,所以理解所有这些方法很重要。

接下来的几节将讨论 ASP.NET 的三种缓存数据的方法。每种方法都有优缺点。幸运的是,这些方法实现起来很简单,所以您很快便能缓存数据!

#### 14.3.1 缓存页面输出

启用输出缓存功能很简单,只需使用页面编译指令 OutputCache 即可:

```
<% @ OutputCache Duration="Time" Location="Where" %>
```

输出被缓存到 where 参数指定的位置,缓存时间由 time 参数指定。Location 参数可设置为表 14.1 列出的任何值。(注意 Location 是一个可选参数。)

表 14.1 有效的 Location 参数值

值	描 述
Any	输出可缓存到下面列出的任何位置, 这是 Location 的默认值
Client	输出将被缓存到客户机上
Downstream	输出将被缓存到处理请求服务器的下游服务器上, 常用于代理 (Proxy) 服务器中
None	对被请求的页面不启用输出缓存功能
Server	输出被缓存到处理请求的服务器中

使用输出缓存时, 所有信息都将被存储在 Location 参数指定的计算机的硬盘上, 而不是内存中, 这对性能的影响很大。这意味着输出缓存提供了一种几乎是免费的缓存方法, 它不占用任何宝贵的资源。

我们来看一个例子。清单 14.1 显示一条简单的欢迎消息和当前的时间。

#### 清单 14.1 使用输出缓存

```

1 <%@ Page Language="VB" %>
2 <%@ OutputCache Duration="30" VaryByParam="*" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object,e as eventargs)
6         lblMessage.Text = "Welcome!The time is now " & _
7             DateTime.Now.ToString("T")
8     end sub
9
10 </script>
11
12 <html><body>
13     <font size="5" >Using the Output Cache</font><br>
14
15 <asp:Label id="lblMessage" runat="server"/>
16 </body></html>

```

**分析:** 这看起来像一个常规的 ASP.NET 页面, 只是多了编译指令 OutputCache。该编译指令导致 ASP.NET 页面生成的内容被缓存 30s。30s 后, 缓存将被作废 (即过期), 下次请求时将再次缓存该页面。图 14.3 是该清单的运行结果。

请注意输出的时间。单击 refresh 按钮, 页面将显示相同的时间。这是因为页面是直接从服务器缓存中取得的, 页面中的代码并没有执行。如果 30s 后再访问该页面, 时间将更新, 因为缓存已过期。

缓存机制的功能远不仅如此。例如, 输出缓存能够对提供给页面的查询字符串进行比较, 以确定要缓存的内容。下面对清单 14.1 稍做修改, 来说明这种效果。

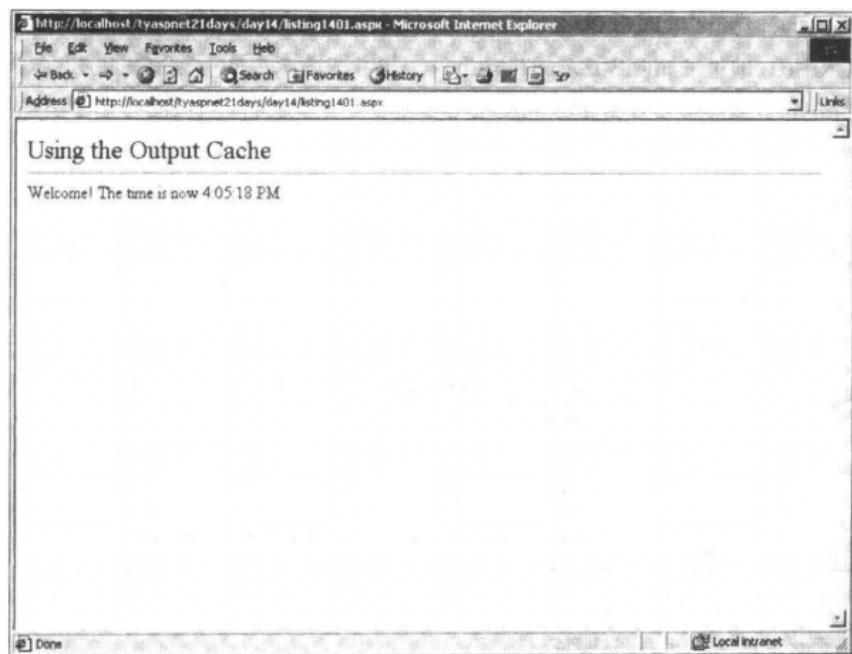


图 14.3 缓存当前时间的结果

**清单 14.2 支持查询字符缓存**

```

1 <%@ Page Language="VB" %>
2 <%@ OutputCache Duration="60" VaryByParam="*" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object,e as eventargs)
6         lblMessage.Text = "Welcome" & Request.Params("id" & _
7             "! The time is now " & DateTime.Now.ToString("T")
8     end sub
9
10 </script>
11
12 <html><body>
13     <font size="5">Using the Output Cache</font><hr>
14
15     <asp:Label id="lblMessage" runat="server"/>
16 </body></html>

```

**分析：**清单 14.2 在第 6 行添加了一个方法，来读取查询字符串参数 id。另外还延长了缓存的持续时间，以便在缓存过期前有更多的时间显示效果。将该页面保存为 listing1302，并在浏览器中请求它，在查询字符串中输入您的姓名，如 <http://localhost/TYASPNET21Days/Day13/listing1302.aspx?id=Chris>。您将看到一条个性化的欢迎消息。另外请注意时间——如果单击 Refresh 按钮，您仍将看到相同的时间。

现在对 URL 稍做修改：[listing1302.aspx?id=Sammy](http://localhost/TYASPNET21Days/Day13/listing1302.aspx?id=Sammy)。您将看到一条新的欢迎消息以及当前的时

间。将页面刷新几次，确保页面被缓存。

最后，输入 URL 地址 `listing1302.aspx?id=Chris`，并再次请求页面。您看到的将是原来的欢迎消息和时间！这并不是您能使得时光倒流，而只是请求了一个已被缓存的页面。图 14.4 显示了对该页面进行三次请求的过程

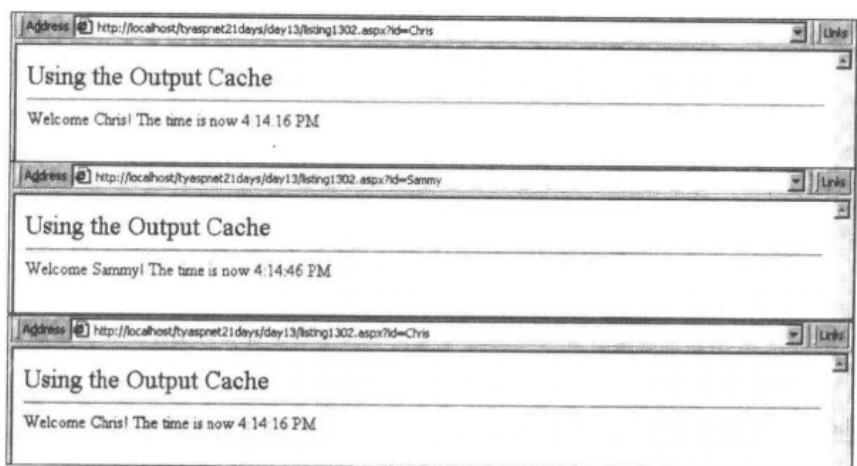


图 14.4 ASP.NET 通过读取查询字符串参数来确定缓存过程

**新术语：**ASP.NET 读取查询字符串来确定被请求页面是否在缓存中。如果查询字符串不同，则 ASP.NET 注册一个缓存遗漏（cache miss），即没有在缓存中找到页面，然后重新执行该页面，并将其存储到缓存中（注意，缓存过期后请求页面时，会生成缓存遗漏，这时页面将被再次执行）。

**注意：**这意味着使用相同的字符串参数（但排列次序不同）请求页面时，将生成两个不同的缓存页面。例如，字符串参数 `page.aspx?first=Chris&last=Payne` 和 `page.aspx?last=Payne&first=Chris` 将生成不同的缓存页面。

可以通过在 `<%@OutputCache%>` 编译指令中加入其他的参数（`VaryByParam`、`VaryByHeader`、`VaryByCustom`）来控制这种功能。这些参数接受用逗号分隔的变量列表，ASP.NET 将使用这些变量列表来区分被缓存的内容。例如，下面的编译指令告诉 ASP.NET，仅当 `firstname` 的值不同时，才将页面输出缓存为独立的条目，在其他情况下，不缓存新的条目：

```
<%@OutputCache Duration="600" VaryByParam="firstname"%>
```

将该编译指令添加到清单 14.2 中，并在浏览器中使用下面的 URL 请求该清单：  
`http://localhost/tyaspnet21days/day14/listing1302.aspx?firstname=Chris&lastname=Payne`。

注意显示的时间。现在将 URL 改为 `listing1302.aspx?firstname=Chris&lastname=Saravia`，并刷新页面。时间并没有改变，因为查询字符串中 `firstname` 为 `Chris` 的页面已被缓存，而查询字符串 `lastname` 被忽略。可以将 `lastname` 参数添加到上述编译指令中，以便根据两个查询字符串值来缓存页面：

```
<%@OutputCache Duration="600" VaryByParam="firstname,lastname"%>
```

注意，如果省略 `VaryByParam` 参数，则默认情况为根据提供的查询字符串缓存页面。本章后面的“各种缓存机制”一节中将对 `VaryByParam` 做进一步的讨论。`VaryByHeader` 参数的工作原理与 `VaryByParam` 类似，但它评估的是 HTTP 报头而不是查询字符串。由于还没有介绍过 HTTP 报头，所以这里不介绍这种方法。最后，`VaryByCustom` 参数让您能够根据 HTTP 报头和查询字符串之外的其他信息（如浏览器名称和版本）来使用不同的缓存机制。

例如，可以在销售音乐会票的网站中使用 VaryByCustom。首先，用户选择州名（音乐会举行的地点）和价格（她能接受的价格）。所选的州和价格将被存储在查询字符串中。下一个页面从数据库中取得音乐会列表，按州进行排序，并将缓存该页面输出。页面将根据选择的州进行缓存，而忽略价格，价格只是“附”在了查询字符串的最后面。通过设置 VaryByParam=“state”，而不是 Price，可以限制缓存的页面数以及耗用的资源。

#### 用户控件

缓存可扩展到用户控件。例如，如果只想缓存页面的部分内容——如一个 DataGrid，则可以将该部分封装到一个用户控件内，并使用 OutputCache 编译指令设置其缓存选项，这被称为部分缓存（fragment caching）。

当只需缓存页面的部分内容时，部分缓存很有用。例如，假设您创建了一个简单的控件，该控件显示文本框，供用户输入用户名和密码信息。没有必要在每次请求时都要重新生成该控件，因为它永远不会改变。因此，可以缓存该控件。然而，包含该用户控件的页面也许还含有其他敏感的信息，如时间和个人信息。在这种情况下，您不想缓存整个页面，否则用户将看到以前的信息。只应缓存不变的部分，如用户控件。

由于部分缓存只是输出缓存的一种高级形式，所以所有被缓存的数据也被存储在硬盘上，这只需极少的服务器资源。清单 14.3 列出了一个简单用户控件的定义（关于如何创建用户控件的更详细的信息，请参阅第 6 章）。

**清单 14.3 一个被缓存的用户控件**

```
1:  <%@ Control Language="VB" %>
2:  <%@ OutputCache Duration="30" VaryByParam="none" %>
3:
4:  <script runat="server">
5:      sub Page_Load(obj as object, e as eventargs)
6:          lblMessage.Text = 'Control last viewed at ' & _
7:              DateTime.Now.ToString("T")
8:      end sub
9:  </script>
10:
11:  <font size="5">Using the Output Cache</font><br>
12:  <asp:Label id="lblMessage" runat="server"/>
```

该控件仅显示了一个标签，后者显示了该控件最后一次被访问的时间。注意第 2 行将缓存持续时间设置为 30s。将该控件保存为 control.ascx。清单 14.4 是一个使用该控件的 ASP.NET 页面。

**清单 14.4 实现用户控件**

```
1:  <%@ Page Language="VB" %>
2:  <%@ Register TagPrefix="MyControl" TagName="View" %>
3:  Src="control.ascx" %>
4:
5:  <script runat="server">
6:      sub Page_Load(obj as object, e as eventargs)
```

```
7:      lblMessage.Text = "Page last viewed at ' & _  
8:      DateTime.Now.ToString("T")  
9:      end sub  
10: </script>  
11:  
12: <html><body>  
13:     <MyControl:View runat='server' /><p>  
14:     <asp:Label id='lblMessage' runat='server' />  
15: </body></html>
```

该页面也很简单。第2行注册了用户控件。Page\_load 事件显示页面被访问的时间。第13行实现了用户控件。注意,清单14.4中并没有使用 OutputCache 编译指令,该编译指令仅包含在控件中。

在浏览器中请求该页面,将发现页面的访问时间和控件的访问时间相同。但刷新页面后,这两个时间将不同。控件被缓存,所以您看到的是以前的访问时间。但页面没有被缓存,所以访问时间在不断地更新。图14.5显示了页面被刷新后的访问时间的差别。

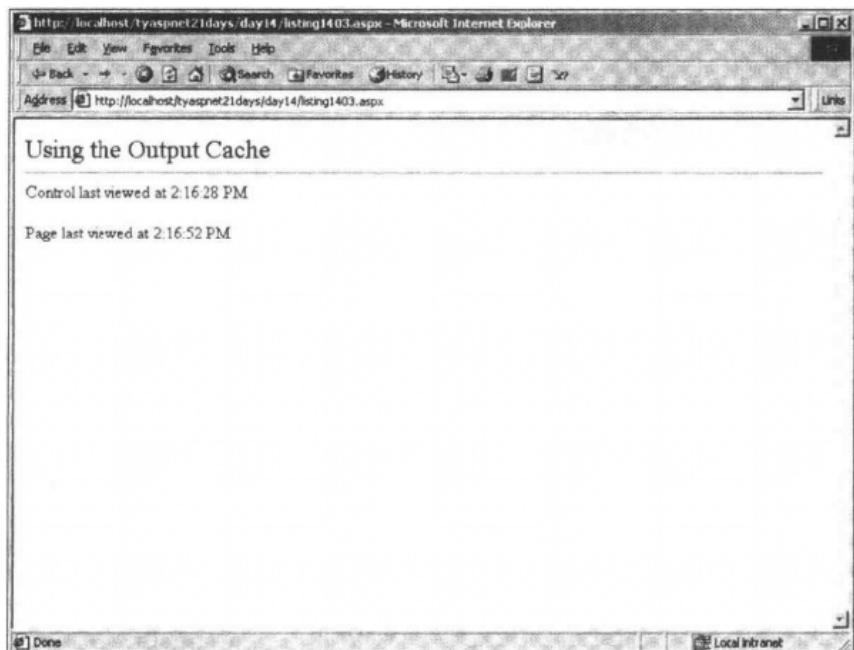


图 14.5 部分缓存

**警告:** 由于用户控件被缓存,所以如果试图在ASP.NET页面中对其进行修改将出错。例如试图给该用户控件指定id值或设置其他属性都将出错。由于控件被缓存,因此被请求时,其代码将不会运行,任何对控件修改的企图都将以失败告终。

看起来这好像是缓存的一个重大缺陷,但是只要正确使用,将不会遇到麻烦。只缓存那些不变的项。通常情况下,无需通过代码操纵这些项。如果发现确实需要操纵被缓存的控件,则也许本来就不应该缓存它。

在缓存页面的部分内容方面,这是一种非常好的方法。然而,在下一小节您将发现,Cache对象甚至提供了一种更有用的方法。

### 14.3.2 缓存对象

可以使用 Cache 类将对象加入到缓存中,这样可以提高性能。想象一下创建大型 DataGrid 对象,这需要花费一定的时间。您必须连接到数据库、取回数据,然后创建并填充 DataGrid。如果能够完成这些操作后,将结果缓存起来该有多好!事实上,缓存数据库返回的数据是 Cache 类最常见的用途之一。

应用程序启动时,将创建 Cache 类的一个实例。不同于输出缓存,放置在 Cache 对象中的对象驻留在服务器的内存中,这可能会降低性能,这取决于缓存中对象的大小。正因如此,使用 Cache 对象要谨慎,不能像输出缓存技术那样频繁地使用。

您可能会问,既然如此,为何还要使用 Cache 类呢?与输出缓存相比,Cache 对象具有三大优势。首先,可以缓存任何对象,您无需缓存整个页面或用户控件,而可以更细致地进行缓存,例如缓存数据库返回的数据或单个的服务器控件。其次,Cache 类允许您使用动态的失效期 (sliding expiration)。最后,可以为放置在 Cache 类中的对象设置依存关系 (dependencies)。这意味着您可以将被缓存的数据设置为依赖于另一项内容。缓存的 DataGrid 可以依赖于底层的数据。当源数据发生变化时,缓存的 DataGrid 将被作废,并重新创建一个新的 DataGrid。更详细的信息,请参阅本章后面的“缓存依存关系”一小节。

清单 14.5 是一个使用 Cache 类来缓存从数据库返回的数据的例子。当页面首次被装载时,您从数据库取回数据,并将其绑定到 DataGrid。同时将数据存储到缓存中,以便下次页面被请求时,能够直接从缓存中取得数据,这样执行速度将大大提高。

**清单 14.5 使用 Cache 类缓存从数据库返回的结果**

```

1 <%@Page Language="VB" %>
2 <%@Import Namespace="System.Data" %>
3 <%@Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     sub Page_Load(obj as Object,e as EventArgs)
7         if not Page.IsPostBack then
8             CreateData()
9         end if
10    end sub
11
12    sub CreateData
13        dim source as DataView
14
15        source = Cache("DataView")
16
17        if source is nothing then
18            dim strConnString as string = "Provider=" & _
19                "Microsoft.Jet.OLEDB.4.0;" & _
20                "Data Source=C:\ASPNET\data\tanking.mdb"
```



```
21
22     dim objCmd as OleDbDataAdapter =new _
23         OleDbDataAdapter('select * from tblUsers ', _
24             strConnString)
25
26     dim ds as DataSet =new DataSet()
27     objCmd.Fill(ds,"tblUsers")
28
29     source =new DataView(ds.Tables("tblUsers"))
30     Cache("DataView")=source
31
32     lblMessage.Text ="Data set created explicitly "
33 else
34     lblMessage.Text ="Data set retrieved from cache "
35 end if
36
37     dgData.DataSource =source
38     dgData.DataBind()
39 end sub
40
41 sub ExpireCache(obj as object,e as eventargs)
42     dim dv as dataview =Cache.Remove('DataView')
43     CreateData()
44 end sub
45 </script>
46
47 <html><body>
48     <form runat="server">
49         <asp:Label id='lblMessage' runat='server'
50             maintainState=false/><p>
51         <asp:Button id="btSubmit" runat="server"
52             text="Expire Cache"
53             OnClick="ExpireCache"/><p>
54         <asp:DataGrid id="dgData" runat="server"
55             BorderColor="black"
56             GridLines="Vertical"
57             cellpadding="4"
58             cellspacing="0"
59             width="450"
60             Font-Name="Arial"
```

```

61      Font-Size="8pt"
62      HeaderStyle-BackColor="#cccc99"
63      ItemStyle-BackColor="#ffffff"
64      AlternatingItemStyle-BackColor="#cccccc" />
65  </form>
66  </body></html>

```

**分析：**第 6~10 行的 Page\_Load 事件处理程序很规范。真正的操作是从第 12 行的 CreateData 方法开始的。

您希望为数据库返回的数据存储一个 DataView，以方便以后的访问，因此第 13 行创建了一个新的 DataView。第 15 行试图通过 Cache 类检索该 DataView。如果缓存中没有指定名称的 DataView（首次查看页面时的情况），变量 source 将被设置为 nothing。

第 17 行的 if 语句检查这种条件。如果 source 等于 nothing，则表明缓存是空的，因此从数据库中取回数据。第 18~25 行完成读取数据的工作，现在您应该十分熟悉该过程了。第 27 行将 source 变量设置为 DataSet 的一个 DataView，并在第 28 行将其存储到缓存中。这样返回数据的视图被存储在缓存中，这意味着返回的数据已保存在缓存中。第 30 和 32 行显示一条文本消息，指出数据来自何方。

为测试该应用程序，您还希望能够手工清除缓存，这是由第 39 行的 ExpireCache 方法完成的。每当用户单击第 49 行的按钮时，该事件处理程序便执行。Cache 对象的 Remove 方法删除缓存指定的内容，这里要删除的是 DataView。最后，再次调用 CreateData 方法，以再次显示 DataGrid。

**注意：**Remove 方法返回一个同要删除的对象类型相同的对象。这让您能够在必要时可以对该对象执行其他的操作。例如，可以使用下面的代码替换第 40 行的代码：

```
dim dv as DataView = Cache.Remove("DataView")
```

这让您能够在缓存中删除对象之前，执行一些操作，例如最后一次读取信息。

如果缓存中没有指定的对象，则 Remove 方法将返回 nothing。

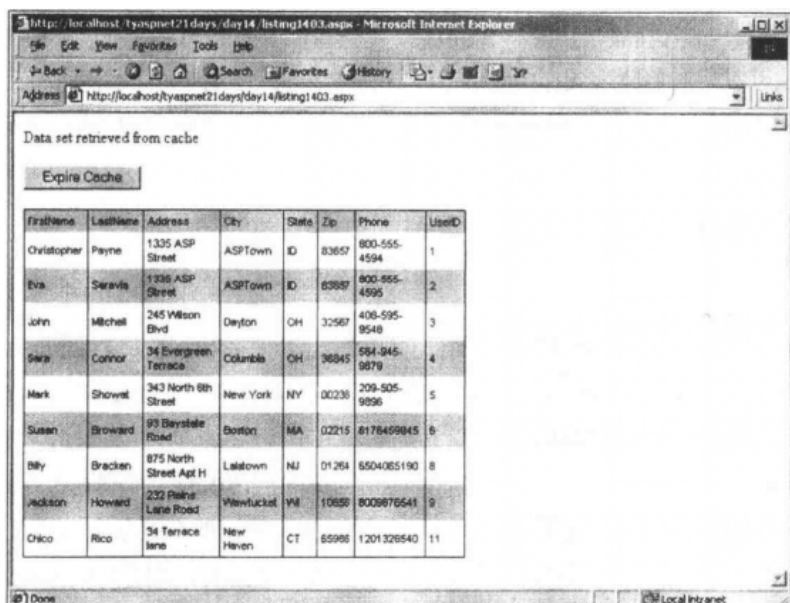


图 14.6 缓存从数据库返回的数据

对 DataView 调用 Remove 方法后, 必须再次从数据库中取回数据。这意味着第 17 行中的 if 语句要执行语句块中的代码。图 14.6 为首次查看该清单时的结果, 图 14.7 为单击“Refresh”后的结果。

首次查看该页面时, 您将看到消息“Data set Created explicitly”。单击“Refresh”按钮后, 将出现消息“Data set retrieved from cache”。最后, 单击“Expire Cache”按钮将再次显示最初的消息。从缓存中读取数据时, 性能将有明显的变化, 这取决于计算机的配置。尽管变化可能很小, 但如果成千上万的用户同时访问同一个页面时, 性能的变化将很大。随着负载的增大, 缓存的好处将呈几何级数增长。



图 14.7 以后查看时, 将直接从缓存中取回 DataSet

**注意:** 如果在单击“Expire Cache”按钮后再单击“Refresh”按钮, 浏览器将询问是否重新提交表单数据。回答 Yes 会重新提交, 导致缓存被再次清除。粗心的用户可能会有这样的疑问: 为什么消息是“Data set created explicitly”, 而不是“Data set retrieved from cache”?

如果想在单击“Expire Cache”按钮后刷新页面, 必须在浏览器中重新输入 URL 地址。或在页面中添加一个超链接 (hyperlink), 并将其 URL 设置为与本页面相同, 这样做的效果相当于刷新。

有三种方法可以将对象放入缓存中。第一种方法已在前面的清单中介绍过。后两种方法分别是使用 Add 方法和 Insert 方法。这两种方法类似, 但 Insert 方法提供的选项更多。下面先来看看 Add 方法的语法:

```
Cache.Add(key, value, dependencies, absoluteExpiration, _
    slidingExpiration, priority, priorityDecay, _
    onRemoveCallback)
```

Add 方法的参数很多, 下面逐个详细介绍。前两个参数 (key 和 value) 在前面的清单中使用过。Key 是要存储到缓存中的对象的名称, value 是实际要存储的对象。Dependencies 稍后再介绍。AbsoluteExpiration 缓存过期的绝对日期, 如“June 6, 2001”。

SlidingExpiration 与 AbsoluteExpiration 类似, 但每当页面被请求时, 都刷新过期日期。例如, 如果使用 AbsoluteExpiration 将过期时间设置为从现在开始 5h 后, 那么缓存肯定将在 5h 后过期。但对于 SlidingExpiration, 则缓存将在最后一次请求起的 5h 后过期。这意味着如果有人 4h 后请求页面, 则缓存将在请求后的 5h 后过期, 也就是从现在起的 9h 后。因此请求后, 过期日期将滑动 (slide)。

可以使用 `timespan` 对象设置滑动时间。

`Priority` 参数指定缓存如何在特定情况下丢弃对象,例如内存不足时。通常情况下,优先级高的对象比优先级低的对象驻留在缓存中的时间长。这大体与在缓存中保留对象的成本相称,如果在时间和系统资源方面,对象创建的成本较高,则它在缓存中保存的时间应该长些,反之亦然。

`PriorityDecay` 指定对象的优先级随时间的衰变速度,换句话说,内存资源不足时,该对象在 20min 后的重要性。最后, `onRemoveCallBack` 是将对象从缓存中删除后应调用的方法。

下面使用 `Add` 方法将 `DataView` 添加到缓存中,使用下面的代码替换清单 14.5 的第 28 行:

```
Cache.Add("DataSet", source, nothing, DateTime.Now, _
    AddMinutes(1), TimeSpan.Zero, _
    CacheItemPriority.High, CacheItemPriorityDecay.Slow, _
    addressof HandleRemove)
```

分析: 这里添加了 `source` 变量,该变量包含要存储的 `DataView`。当前,可将依存关系设置为 `nothing`,将过期时间设置为 1 分钟以后,滑动过期设置为 0,即关闭滑动功能。您使用 `CacheItemPriority` 和 `CacheItemPriorityDecay` 对象的 `High` 和 `Low` 值设置优先级和优先级的衰变参数(对这些值的描述,请参见表 14.2)。最后一个参数指定当从缓存中清除该对象时要执行的方法(如果不想处理该事件,可设置为 `nothing`)。如果设置为一个并不存在的方法将出错。

`HandleRemove` 方法如下:

```
sub HandleRemove(key as string, value as object, _
    reason as CacheItemRemovedReason)
    [code]
end sub
```

该方法接受被删除的对象的 `key` 和 `value`,以及删除的原因(`reason`)。`Reason` 参数可能的取值如下:

- `DependencyChanged`: 该对象所依存的对象(如数据库)已改变;
- `Expired`: 对象已超过有效期;
- `Removed`: 使用了 `Remove` 方法;
- `Underused`: 计算机内存不足。

表 14.2 列出了参数 `CacheItemPriorities` 和 `CacheItemPriorityDecay` 的取值

表 14.2 `CacheItemPriorities` 和 `CacheItemPriorityDecay` 的取值

Priority 值	描 述
<code>AboveNormal</code>	比 <code>Normal</code> 优先级稍高
<code>BelowNormal</code>	比 <code>Normal</code> 优先级稍低
<code>Default</code>	<code>Normal</code> 优先级
<code>High</code>	被删除的可能性较小
<code>Low</code>	被删除的可能性最大
<code>Normal</code>	基准优先级

续表

Priority 值	描 述
NotRemovable	不能从缓存中删除
Priority Decay 值	描 述
Default	等于 Medium
Fast	不经常被访问时, 被删除的可能性最大
Medium	基准值
Never	永远不会被删除
Slow	不经常被访问时, 被删除的可能性最小

注意: 表14.2中的描述仅适用于由于内存不足导致缓存自动删除对象的情况, 而不适用于手工删除的情况。

Insert 方法的语法与 Add 语法完全相同, 只是一些参数是可选项。您可以用下面任何一种方式调用 Insert 方法:

```
Cache.Insert(key, value)
Cache.Insert(key, Value, dependencies)
Cache.Insert(key, Value, dependencies, absoluteExpiration, _
    slidingExpiration)
Cache.Insert(key, Value, dependencies, absoluteExpiration, _
    slidingExpiration, priority, priorityDecay, _
    onRemoveCallback)
```

**警告:** 如果插入或添加的对象与缓存中的对象重名, 则原来的对象将被删除。

使用这种机制, 几乎可以缓存页面的任何一部分, 从 DataViews 到 DataGrids 再到用户控件。

### 14.3.3 缓存依存关系

ASP.NET 的缓存机制让您能够通过 Cache 类指定缓存中的对象依赖于其他的对象。例如, 假设您使用 XML 文件中的数据创建一个缓存的 DataView 对象。如果 XML 文件发生变化, 您希望 DataView 也随之变化。图 14.8 说明了缓存与外部数据源的联系。

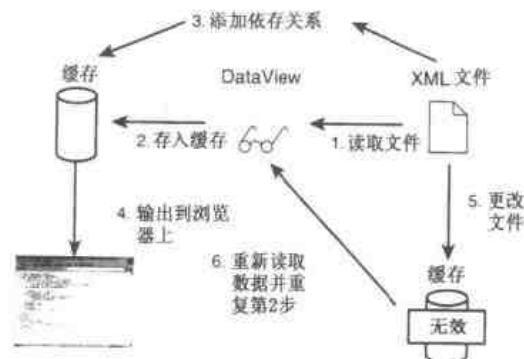


图 14.8 依存关系使得缓存依赖于另一个数据源

如果没有图 14.7 所示的依存关系，缓存将不知道数据已改变，因此用户得到的将是过期的数据。

可以为缓存中的文件、目录或其他对象创建缓存依存关系。让我们对清单 14.5 稍做修改，以读取第 11 章创建的 XML 文件中的数据，并给对象添加依存关系（见清单 14.7）。

#### 清单 14.7 使用缓存依存关系

```

1 <%@Page Language="VB" %>
2 <%@Import Namespace="System.Data" %>
3 <%@Import Namespace="System.Data.OleDb" %>
4 <%@Import Namespace="System.IO" %>
5
6 <script language="VB" runat="server">
7   sub Page_Load(obj as Object,e as EventArgs)
8     if not Page.IsPostBack then
9       CreateData()
10    end if
11  end sub
12
13  sub CreateData
14    dim source as DataView
15    dim objFs as FileStream
16    dim objReader as StreamReader
17    dim ds as new DataSet()
18
19    source =Cache("DataView")
20
21    if source is nothing then
22      objFs =New FileStream(Server.MapPath(_
23        (" ../day11/books.xml"), FileMode.Open, _
24        FileAccess.Read)
25      objReader =New StreamReader(objFs)
26      ds.ReadXml(objReader)
27      objFs.Close()
28
29      dim objDepend as new _
30        CacheDependency(Server.MapPath("../day11/books.xml"))
31
32      source =new DataView(ds.Tables(0))
33      Cache.Insert("DataView", source, objDepend,
    DateTime.Now.AddMinutes(1), TimeSpan.Zero)
  
```

```

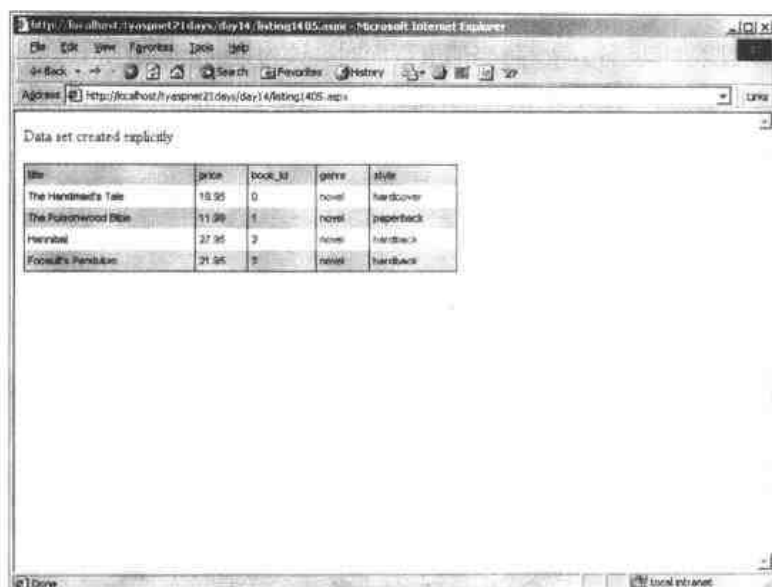
34
35     lblMessage.Text = "Data set created explicitly"
36 else
37     lblMessage.Text = "Data set retrieved from cache"
38 end if
39
40 dgData.DataSource = source
41 dgData.DataBind()
42 end sub
43
44 </script>
45
46 <html><body>
47     <form runat="server">
48         <asp:Label id="lblMessage" runat="server"
49             maintainState=false/><p>
50         <asp:DataGrid id="dgData" runat="server"
51             BorderColor="black"
52             GridLines="Vertical"
53             cellpadding="4"
54             cellspacing="0"
55             width="450"
56             Font-Name="Arial"
57             Font-Size="8pt"
58             HeaderStyle-BackColor="#cccc99"
59             ItemStyle-BackColor="#ffffff"
60             AlternatingItemStyle-BackColor="#cccccc" />
61     </form>
62 </body></html>

```

**分析：**清单中的 HTML 部分没变（只是删除了 Expire Cache 按钮），该清单的大部分内容与清单 14.5 相同，我们来看看不同的地方。

第 22~27 行读取 XML 文件（而不是数据库）中的数据，但与以前一样，将数据放入 DataSet（别忘了导入名称空间 System.IO）。第 29 行为第 11 章创建的 books.xml 文件创建一个 CacheDependency 对象。最后，显示一条消息（告诉用户数据的来源）并将数据绑定到 DataGrid。

该页面的运行过程同以前一样。DataView 将在首次查看该页面时被创建。以后再请求该页面时，将直接从缓存中取得数据。现在在记事本或其他文本编辑器中打开 XML 文件，并添加一条新记录。再次请求页面，您将发现 DataView 将重新创建。如果没有依存关系，ASP.NET 将不知道数据已变化，继续提供缓存中的旧数据。图 14.9 是在 XML 文件中添加记录后，该清单的运行结果。



title	price	book_id	genre	style
The Handmaid's Tale	18.95	0	novel	hardcover
The Poisonwood Bible	11.99	1	novel	paperback
Hemlock	27.95	2	novel	hardback
Frostbite's Penetration	21.95	3	novel	hardback

图 14.9 更新依赖的对象将导致失效并被重新创建

#### 14.3.4 使用 HttpCachePolicy 类

前面几节介绍了使用页面缓存技术的基本方法，这很可能已经能够满足您的需要，但 ASP.NET 提供的另外一个类让您能够更好地控制缓存机制——HttpCachePolicy 类。

可以通过 HttpResponse 对象的 Cache 成员（Response.Cache）来访问 HttpCachePolicy 类，该类包含能够控制缓存如何被处理的高级方法。HttpCachePolicy 对象使用 HTTP 报头来控制缓存，从而控制页面输出缓存。HttpCachePolicy 和页面输出缓存的不同之处在于，前者对缓存机制具有更大的控制能力。

##### 1. 操纵缓存

Response.Cache 对象提供了几个操纵过期时间的方法，可以使用这些方法代替编译指令 OutputCache。

第一个方法 SetExpires 设置缓存的绝对过期时间。例如，下面语句将缓存过期时间设置为 30 秒后（从现在开始）：

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(30))
```

下面的代码将缓存的过期时间设置为当天下午 3 点（当地时间）：

```
Response.Cache.SetExpires("3:00:00PM".ToDateTime())
```

另一种设置绝对过期时间的方法是使用 SetMaxAge 方法。该方法告知缓存在保持多长时间后将作废。例如，下面的语句指定缓存必须在 0 小时 30 分 0 秒后作废：

```
Response.Cache.SetMaxAge(new TimeSpan(0,30,0))
```

这两个方法很类似。如果知道缓存过期的准确时间（或相对于当前的时间），则使用 SetExpires 方法。如果不知道或不关心缓存过期的准确时间，但希望缓存在过一段时间后过期，则使用 SetMaxAge 方法。另外需要注意的是，SetExpires 方法接受一个 Datetime 参数，而 SetMaxAge 方法接受一个 TimeSpan 参数。

Response.Cache 对象还可用于设置滑动过期时间（参见本章前面的“缓存对象”小节）。要启用滑动过期功能，只需将该方法设置为 true：



```
Response.Cache.SetSlideExpiration(true)
```

缓存将在滑动过期时间后过期，每当请求页面时，该过期时间都将更新。

您还可以控制是否进行缓存以及缓存的存储位置。`SetCacheability` 方法可以指定谁可以缓存页面。表 14.3 列出 `cacheability` 的类型。

**注意：**如果滑动过期功能被启用，则必须以相对于当前时间的方式设置过期日期。例如，在下面的代码片段中，滑动过期日期将无效：

```
Response.Cache.SetExpires("11/21/01 3:00:00PM", ToDateTime)
Response.Cache.SetSlideExpiration(true)
```

ASP.NET 无法通过具体的时刻来滑动过期时间。然而，下面的代码片段能够正确运行：

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(30))
Response.Cache.SetSlideExpiration(true)
```

滑动过期时间将在最后一次请求的 30 秒后。

表 14.3 Cacheability 的类型

类 型	描 述
NoCache	客户和共享（代理）服务器都不能缓存对象，客户必须通过服务器重新验证的有效性
Private	客户可以缓存响应（response），但共享（代理）服务器不能，这是默认设置
Public	客户和共享服务器都能缓存响应

当您知道响应的去向以及将被如何处理时，设置该属性很有帮助。例如，如果响应将通过代理服务器进入网络，则可以禁止代理服务器缓存响应，避免客户使用从“中间人”那里得到的缓存：

```
Response.Cache.SetCacheability(HttpCacheability.Private)
```

同样的道理，`SetNoServerCaching` 方法防止服务器缓存整个文档。以后对该文档的所有请求都将重新进行处理——而不是从缓存中取得。例如：

```
Response.Cache.SetNoServerCaching()
```

**注意：**您或许会问，应在什么位置使用 `Response.Cache` 对象的这些方法。您的自由度很大，只要放在代码声明处的方法中就行，但通常将这些命令放在页面的 `Page_Load` 处理程序的开始位置，以确保所有的输出都能被正确处理。

## 2. 改变缓存机制

还记得吗，本章前面介绍了缓存技术以及使用 `VaryByParam` 属性来查看查询字符串。默认情况下，对于每个不同的查询字符串，ASP.NET 都将在缓存中保存一个独立的对象，但您可以控制这种行为。

`Response.Cache` 的 `VaryByParams` 属性指定如何根据页面提供的参数（查询字符串或表单变量）来缓存对象。其工作原理与页面输出缓存相同，清单 14.8 是一个例子。

### 清单 14.8 缓存机制随参数而异

```
1 <%@ Page Language="VB" %>
2 <%@ OutputCache Duration="60" VaryByParam="*" %>
3
```

```
4 <script runat="server">
5     sub Page_Load(obj as object,e as EventArgs)
6         Response.Cache.VaryByParams.Item("first")=true
7         Response.Cache.VaryByParams.Item("last")=false
8
9         lblMessage.Text="Welcome " & Request.Params("first") & _
10             "The time is now " & DateTime.Now.ToString("T")
11
12         lblMessage.Text += "<br>" & Response.Cache._
13             VaryByParams.Item("first ").ToString
14         lblMessage.Text += "<br>" & Response.Cache._
15             VaryByParams.Item("last ").ToString
16     end sub
17 </script>
18
19 <html><body>
20     <font size="5">Using the Output Cache</font><br>
21     <form runat="server">
22         <asp.Label id="lblMessage" runat="server"/><p>
23     </form>
24 </body></html>
```

分析：该清单与清单 14.2 类似，只是在第 10~13 行添加了 VaryByParams 属性。请求该页面时，用户提供两个参数：first 和 last，它们分别表示用户的名和姓。首次查看该页面时，运行结果如图 14.10 所示。

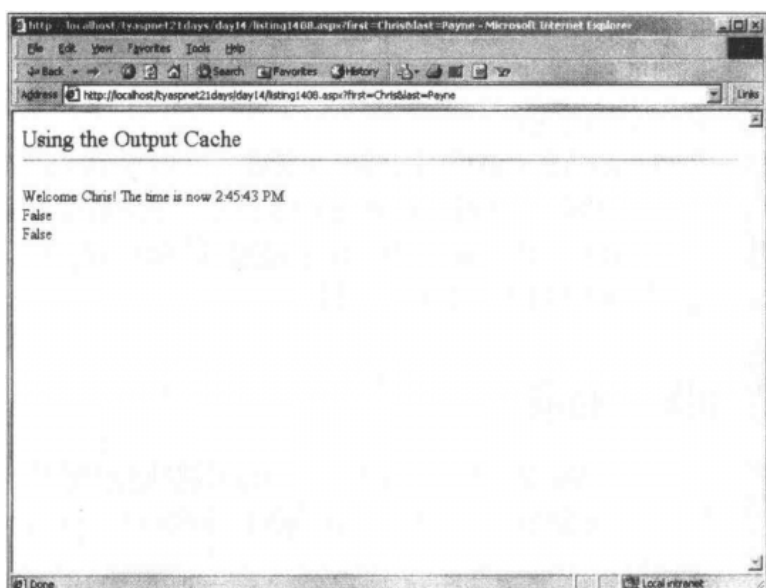


图 14.10 显示 VaryByParams 条目

VaryByParams 属性中的 first 和 last 值（即查询字符串的 first 和 last 参数）都为 false，如图 14.9 所示。这意味着其中每个参数不同时，都将在缓存创建该文档的一个独立版本。版本随参数值而异。

这是默认行为，您在本章前面的“缓存页面输出”一小节中见过。为 first 和 last 指定不同的值，将在缓存中存储另一个页面。若指定的参数相同，则取得缓存中存储的页面。

这是有趣的地方。让我们将以下代码加到清单 14.8 中 Page\_Load 事件的最前面：

```
Response.Cache.VaryByParams.Item("first")=true
```

现在试着修改查询字符串参数。修改 first 将导致缓存一个新的页面。修改 last，并保持 first 不变，将不会缓存新的页面，而是返回一个缓存过的页面，从时间戳戳上可以明显地看出这一点。图 14.11 说明了这种差别。

将 VaryByParams 值改为 true 后，相当于告诉 ASP.NET，仅当 first 参数不同时，才缓存新的页面。同样，下面的设置将导致这样的结果，即仅当 last 值不同时，ASP.NET 才缓存新的页面：

```
Response.Cache.VaryByParams.Item("last")= true
```

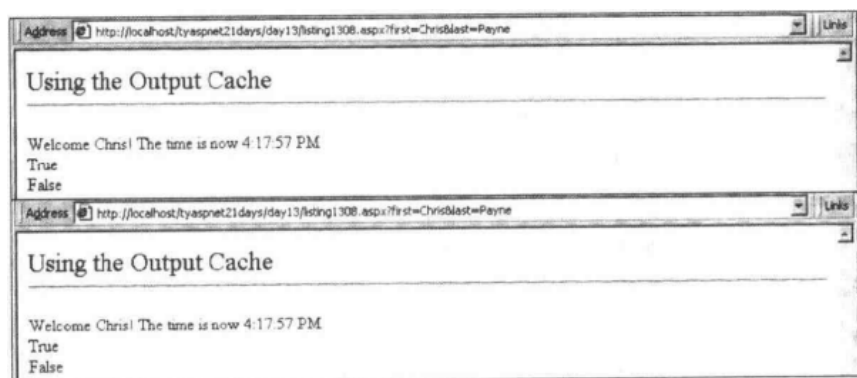


图 14.11 last 参数不同时不会缓存新的页面

这是一个非常有用的属性。例如，假设您要一个根据州来确定个人纳税情况的网站中缓存页面。您在查询字符串中传递州名缩写和纳税人姓名。但纳税情况只与纳税人所在的州相关，所以不必根据纳税人姓名来缓存页面。您可以进行如下设置，以便仅当州不同时才更新缓存：

```
Response.Cache.VaryByParams.Item("State") = true
```

正确的使用可以极大地提高性能。

也可以使用 VaryByHeaders 方法来根据 HTTP 报头更新缓存，其工作原理也与前面关于页面输出缓存一节中讨论过的属性类似。本书没有深入讨论各种不同的 HTTP 报头，所以将该话题留给读者去探索。虽然 Response.Cache 对象没有 VaryByCustom 属性，但 SetVaryByCustom 方法的工作原理类似（更详细的信息，请参考 .NET 框架的 SDK 文档）。

## 14.4 高效地使用缓存技术

缓存技术可以极大地提高 ASP.NET 应用程序的性能。在负载很大的情况下，即使缓存时间非常短，其效果也将十分明显。如果每缓存一个对象节省一秒钟，想象一下一分钟内收到一百个对该对象的请求时，其益处有多大。

通常，有很多信息可以进行缓存，包括从数据库或文件中取回的数据、复杂计算的结果以及应

用程序的设置。

但有时候使用缓存并不是一个好主意，如必须提供最新股票报价的在线代理程序。除非设置足够的依存关系，否则在这种情况下使用缓存将是非常糟糕的。即便设置足够的依存关系，这样做也是得不偿失的。

表 14.4 高效地使用缓存技术

应 该	不 应 该
应缓存那些经常被访问，同时变化频率不大的数据	不要缓存个人信息。如果缓存用户的个人信息，如密码，否则其他人将很容易取得这些信息
应缓存整个应用程序都要使用的设置或对象，但这些设置和对象必须在其生存期内不会变化	不要缓存包含基于时间值的页面，否则用户将无法理解为何时间总是滞后
	不要缓存用户随时都会修改的对象，如购物车

## 14.5 这不是 ASP.NET

在传统 ASP 中，只能使用 `Response.Expires` 和 `Response.ExpireAbsolute` 方法来控制客户端缓存。而在 ASP.NET 中，不仅可以控制客户端缓存，还可以控制服务器端缓存。这是 ASP.NET 对 ASP 最重大的改进之一：ASP.NET 拥有健壮的缓存机制，可以通过很多方式进行控制。这种健壮的缓存机制可极大地提高了应用程序的性能。

## 14.6 总 结

本章介绍了 ASP.NET 众多的缓存特性，其中的很多特性是自动实现的。例如，ASP.NET 自动缓存应用程序设置和编译后的 ASP.NET 页面。

本章还介绍了如何在 Web 服务器上缓存 ASP.NET 页面的输出。ASP.NET 的强大功能之一是实现缓存输出非常简单，只需包含 `<%@ OutputCache %>` 编译指令并设置缓存期即可。

本章介绍了 ASP.NET 众多的缓存特性。缓存数据有很多优点，ASP.NET 充分利用了缓存的这些优势，并且其中的很多功能是自动实现的。例如，ASP.NET 自动缓存许多我们意识不到的对象，如应用程序设置、编译后的 ASP.NET 页面。本章还介绍了使用输出缓存技术在 Web 服务器上缓存 ASP.NET 页面。使用输出缓存技术很简单，这也是 ASP.NET 强大之处，要启用输出缓存功能，只要使用编译指令 `<%@ OutputCache %>`，并指定缓存期即可。

除了缓存整个页面外，还可以使用部分缓存，即只缓存 ASP.NET 页面中的某些用户控件。

`Cache` 对象让您能够轻松地检索和存储缓存中的对象。`Add` 和 `Insert` 方法让您能够指定缓存的优先级、过期时间和依存关系。

依存关系使得缓存将其内容的有效性建立在其他对象（如文件、目录或缓存中的其他对象）的基础之上。

通过 `Response.Cache` 对象可以使用 `HttpCachePolicy` 类，后者让您能够进一步控制缓存机制。当有其他信息（如来自查询字符串或表单提交的信息）可用时，可以使用 `VaryByHeaders` 属性来指定如何缓存页面。

又一部分的内容结束了！下一部分将学习 ASP.NET 应用程序的高级程序概念，包括 Web 服务和配置。这些概念将丰富您的 ASP.NET 技能，使您可以应付任何 ASP.NET 问题。别忘了阅读附加项目 1 中的内容！

## 14.7 问与答

问：为什么不缓存所有内容呢，包括那些极少被访问的内容？值得通过这样做来提高性能吗？

答：没有必要。还记得吗，除输出缓存之外，缓存内容都要占用内存。缓存的内容越多，其他程序可用的内存就越少。这将严重影响 Web 服务器的性能。然而，您可以无限制地使用输出缓存！

缓存对象的数目和由此给性能带来的益处取决于具体的情况和开发人员，所以主观性相当强。幸运的是，缓存永远不会过多，因为当内存不足时，ASP.NET 将自动删除缓存。即便这样，也不应将所有内容装载到缓存中。

问：ASP.NET 如何将数据缓存到服务器之外的计算机上？

答：利用 HTTP 缓存报头。HTTP 缓存报头由 W3C 标准化，并被用来设置浏览必须遵循的指南，包括数据存储的方式和位置以及其他的信息（如过期时间和缓存改变参数）。本章介绍的各种缓存机制都以某种方式使用了这些 HTTP 报头。有关 HTTP 缓存报头的更详细信息，请参阅 W3C 网站：[www.w3c.org](http://www.w3c.org)。文档 RFC 2616: Hypertext Transfer Protocol HTTP/1.1 提供了关于缓存技术使用 HTTP 报头的详细技术信息。

## 14.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 14.8.1 小测验

1. 客户端缓存与服务器端缓存有什么区别？
2. 哪种缓存机制将数据存储到计算机硬盘上？
3. Cache 类相对于输出缓存的优缺点是什么？
4. Response.Cache 对象的 SetExpires 方法和 SetMaxAge 方法之间有什么区别？
5. Cache.Add 的完整语法结构是什么样的？
6. 假设有一个页面可以接受下列查询字符串参数：userid、searchstring、pagenumber 和 orderby。

当使用 VaryByParam 时，应根据哪个参数来缓存页面，需要添加什么代码？

### 14.8.2 练习

使用清单 14.5 测试从缓存中读取 DataView 相对于重新创建 DataView 的性能优势。（提示：可以使用 DateTime 实例的 Ticks 属性返回单位非常小（100ns）的时间）。讨论时间上的差异。

## 第二部分 复 习

一切进展顺利！在这一部分，您掌握了各种数据访问方法，从 ASP.NET 新手成为一名资深的 ASP.NET 开发人员！现在您已掌握了如何在页面中使用各种控件、收集用户的信息，创建自己的应用程序、与数据库交互、读写文件以及处理缓存。仅仅在几章内，您便取得这么大的成就。

现在学习新的 ASP.NET 技术，您将得心应手，并能读懂 ASP.NET 页面中的大部分代码。然而，有几个有助于丰富您的技能的重要主题还没有介绍过，包括高级编程技能，如调试和安全以及 Web 服务。下一部分将介绍这些内容。

### 附加项目 2

请稍等，您怎能这么轻松地前进！首先，为确保您已牢固地掌握了 ASP.NET 技术，请对在“附加项目 1”中开发的银行业在线应用程序进行改进。

这里将在该银行业应用程序中加入数据功能。具体地说，将添加记录并显示交易的功能。阅读完第二部分后，这项任务很简单。加入新功能后，该应用程序的功能将强大得多。

别忘了，您可以随便对该项目进行改进。放开胆量去探索其他的主题，或在应用程序中添加一些现有的特性。您在该项目上下的功夫越多，您得到的技能就会越强。

### 数据添加功能

为在应用程序中添加数据功能，您需要完成两项工作：更新数据库（添加新表、修改原来的表以及创建几个存储过程）、在 ASP.NET 页面中加入数据功能。

您已经在数据库中创建了一个表，用于存储用户信息，但该应用程序的要求不仅是这些。具体地说，您需要添加两个表，一个用于存储在线交易数据、另一个用来存储账户信息。接下来添加存储过程，来处理与数据库的交互，并获得比不使用存储过程要高的性能。最后，您将更新“附加工程 1”中创建的.aspx 文件，在三个主要的页面中加入数据功能（用户控件保持不变）。

#### 数据库

首先，您需要创建数据库表！您已在第 8 章创建了一个用户表。现在对该表稍做修改，然后创建几个新的表。

在 Microsoft Access 中打开 Banking 数据库，并在设计模式下打开 tblusers 表。您需要在表中添加一个密码字段，以使用户要登录应用程序时可以验证其身份。您还需要添加一个用户名字段，以保证用户登录时其用户名是唯一的——用户名必须是唯一的。将这两个新字段添加到设计视图网格中，并将其数据类型都设置为 Text。

注意：创建这两个字段后，原有的每条记录后都会加上两个字段（password 和 username），其值为 null。您必须手工为输入每条记录的值，否则这些用户将无法登录。

接下来，需要在数据库中添加两个更重要的表：一个用于记录交易，另一个用于记录账户信息（即结余）。在设计视图下创建第一个表，并添加以下字段。

- TransactionID：一个数据类型为 autonumber 的字段，用于唯一地标识每一次交易。在该字段上单击鼠标右键，并选择 Primary Key，将其设置为主关键字；

- DataPosted：data/time 类型，用于存储日期；
- Amount：Currency 类型，用来存储交易量（可能是负数）；
- Payee：一个 Text 字段，用于存储收款人；
- CheckNumber：一个可选的 number 字段，用来存储账号；
- UserID：一个 number 类型的字段，用来将交易表 and 用户表关联起来。

将该表保存为 tblTransactions。现在必须在该表的 UserID 字段和 tblUsers 表的 UserID 字段之间建立关系。选择 Tools/Relations，打开一个询问您需要关联哪些表的对话框。加入上述两个表后，得到如图 BP2.1 所示的结果，然后关闭对话框。

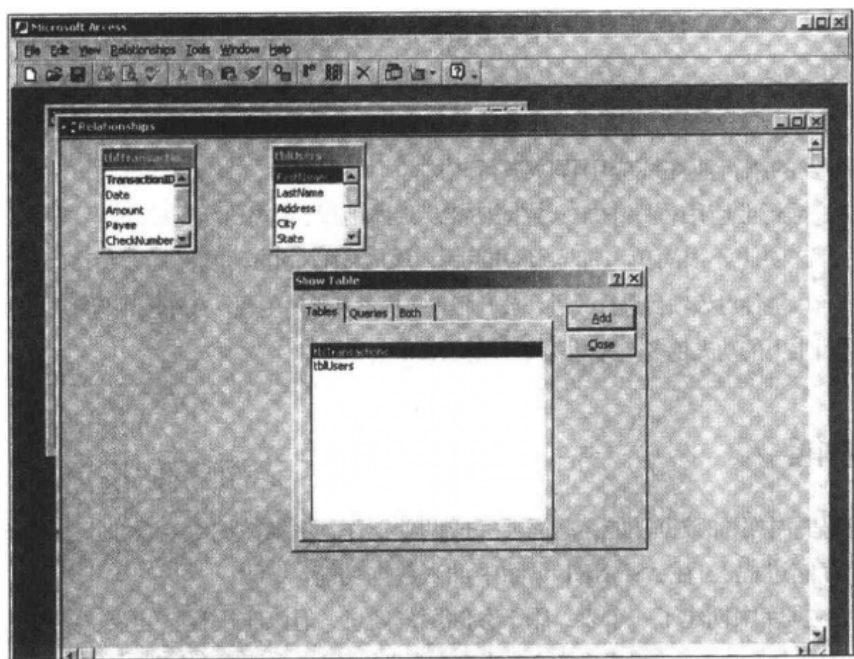


图 BP2.1 在两个表之间建立关系

滚动两表的对应窗格，直到看到两表的 UserID 字段。单击其中一个表的 UserID 字段，按住鼠标，并将其拖拽到另一个表的 UserID 字段上，然后松开鼠标。这时将出现一个对话框，确认要关联的字段名。如果某个字段中显示的不是 UserID 字段，则修改为 UserID 字段。单击 Create，两个表窗口之间将出现一条线，将两个 UserID 字段连接起来。关闭 Relations 窗口，并保存刚才创建的关系。

在设计视图下新建一个包含以下字段的新表。

- **AccountID**: 一个数据类型为 `autonumber` 的字段, 用于唯一地标识每个账户。将该字段设置为主关键字;

- **Balance**: 一个 `currency` 类型的字段, 用于保存账户的结余;

- **UserID**: 一个 `number` 类型的字段, 用于将账户表 and 用户表关联起来。

将该表保存为 `tblAccount`, 并像 `tblTransaction` 表那样, 使用 `UserID` 字段将该表和 `tblUsers` 表关联起来。

**警告:** 一定要将 `tblAccount` 表和 `tblUser` 表, 而不是 `tblTransaction` 表关联起来。无论与哪个表建立关系, 应用程序都能运行, 但您要做的是通过关联正确的字段来建立真正的关系型数据库。另外, 建立正确的关系也使数据库更容易理解。

接下来, 需要创建几个简单的存储过程。前面介绍过, 存储过程在 `Access` 中被称为查询。将切换到 `query` 选项卡, 按第 14 章介绍的方法新建一个存储过程, 然后在该查询中输入清单 BP2.1 中 SQL 语句。

#### 清单 BP2.1 返回账户结余

```
1: SELECT Balance FROM tblAccounts WHERE
2: UserID = @UserID
```

提供一个有效的 `UserID` 时, 该查询将返回账户的结余。将该查询保存为 `spRetrieveBalance`。接下来需要创建一个将交易插入到数据库中的查询, 如清单 BP2.2 所示。

#### 清单 BP2.2 将交易插入到数据库中

```
1: INSERT INTO tblTransactions (DatePosted, Amount, Payee,
2: UserID)
3: VALUES
4: (@Date, @Amount, @Payee, @UserID)
```

将该存储过程保存为 `spInsertTransaction`。注意该查询语句中没有账号字段, 因为该存储过程用于在线交易, 因此在这里使用账号不合适, 因此没有使用账号。`Access` 将自动将账号设置为默认值 (应该是一个空字符串)。您还需要添加一个根据用户提供的用户名和密码来验证用户身份的存储过程。清单 BP2.3 列出了该查询的代码。

#### 清单 BP2.3 验证用户

```
1: SELECT UserID FROM tblUsers
2: WHERE UserName = @UserName
3: AND Password = @Password
```

将该清单保存为 `spValidateUser`。您还需要创建另一个用来 (当在线进行交易时) 更新账户结余的存储过程。清单 BP2.4 为其代码, 将该清单保存为 `spUpdateBalance`。

#### 清单 BP2.4 更新结余

```
1: UPDATE tblAccounts SET Balance = @NewBalance
2: WHERE UserID = @UserID
```

最后, 需要再创建一个返回给定用户的所有交易的存储过程, 其代码如清单 BP2.5 所示



## 清单 BP2.5 收集交易

```
1: SELECT TransactionID, DatePosted, Amount, Payee,  
2:     CheckNumber, UserID  
3: FROM tblTransactions  
4: WHERE UserID = @UserID
```

将该清单保存为 `spGetTransactions`。所需要的存储过程都有了，下面进入 ASP.NET 页面部分。

## ASP.NET 页面

您仍需要在“附加项目 1”中创建的 `account.aspx`、`bills.aspx` 和 `login.aspx` 页面以及各种用户控件，您必须在这些页面中完成以下几项工作。

首先，需要修改 `login`（登录）页面，以便根据数据库中的记录，而不是根据硬编码值来验证用户的身份。然后，需要更新账户摘要页面，从数据库中取得账户结余和交易记录。这两种值都是只读的，所以不能允许用户修改它们。最后，更新账单支付页面，让用户输入交易。该页面以两种方式更新数据库：更新账户结余、向数据库添加新的交易。这里描述的所有数据库操作都将使用前面创建的存储过程来完成。图 BP2.2 说明了应用程序最后的情况。

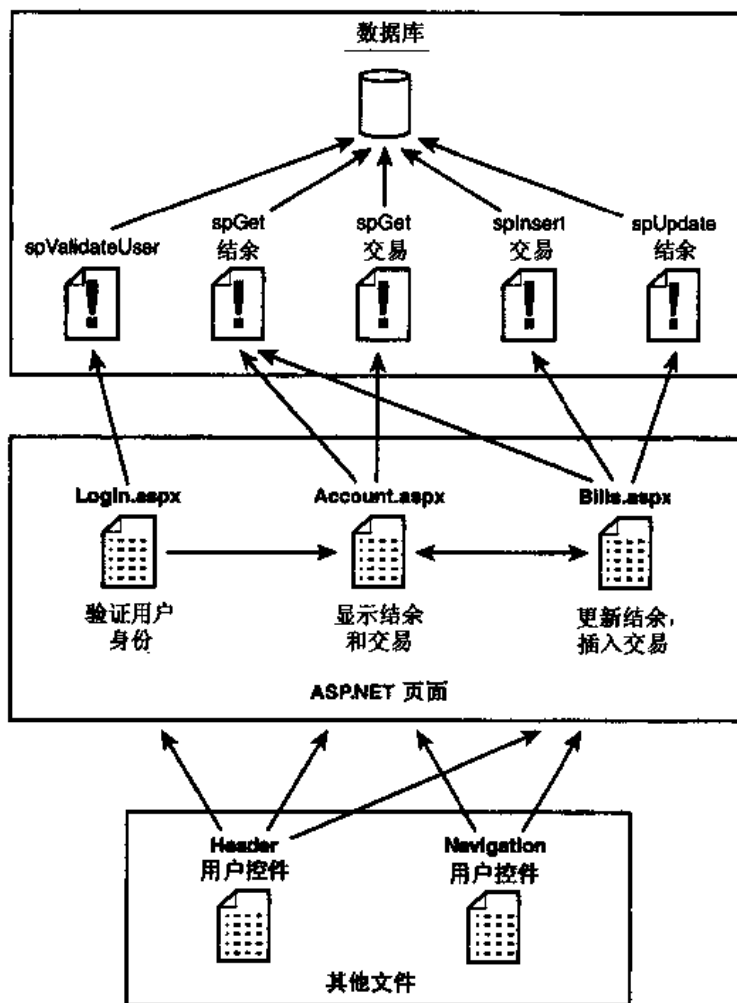


图 BP2.2 银行业应用程序程序站点图表

首先，修改 login.aspx 以验证用户的身份。该页面在一个方法（Submit 按钮的事件处理程序）中处理所有的工作。取得文本框中的用户名和密码后，该方法执行 spValidateUser 存储过程来确定用户是否合法。如果合法，则将用户重定向，否则给出提示。由于该页面的 HTML 部分与“附加项目 1”中相同，所以清单 BP2.6 仅列出了代码声明块。

#### 清单 BP2.6 Login 方法

```

1 <%@ Page Language="VB" %>
2 <%@ Register TagPrefix="ASPNETBank" TagName="Header" src="header.ascx" %>
3 <%@ Import Namespace="System.Data" %>
4 <%@ Import Namespace="System.Data.OleDb" %>
5
6 <script runat="server">
7     '*****
8     '
9     ' Login.aspx: Logs users in
10    '
11    '*****
12
13    '*****
14    ' When user clicks submit button, verify that they are a
15    ' valid user. If they are, log them in, and set a cookie
16    ' with their user name, and redirect to account.aspx.
17    ' Otherwise display error message
18    '*****
19    sub Login(obj as object, e as eventargs)
20        dim intId as integer = 0
21        dim Conn as new OleDbConnection("Provider=" & _
22            "Microsoft.Jet.OLEDB.4.0;" & _
23            "Data Source=c:\ASPNET\data\banking.mdb")
24
25        dim objCmd as OleDbCommand = new OleDbCommand _
26            ("spValidateUser", Conn)
27        objCmd.CommandType = CommandType.StoredProcedure
28
29        'set parameters for stored procedure
30        dim objParam as OleDbParameter
31        objParam = objCmd.Parameters.Add("@UserName", _
32            OleDbType.BSTR)
33        objParam.Value = tbUserName.Text
34    end sub

```

```

35     objParam = objCmd.Parameters.Add("@Password", _
36         OleDbType.BSTR)
37     objParam.Value = tbPassword.Text
38
39     try
40         objCmd.Connection.Open
41         intID = CType(objCmd.ExecuteScalar, Integer)
42         objCmd.Connection.Close
43     catch ex as OleDbException
44         lblMessage.Text = ex.Message
45     end try
46
47     if intID <> 0 then
48         Response.Cookies("Account").Value = intID
49         Response.redirect("account.aspx")
50     else
51         lblMessage.Text = "<font color=red>Sorry, " & _
52             "invalid username or password!</font><p>"
53     end if
54 end sub
55 </script>

```

**分析：**开始的几行很简单。第 2 行注册了页眉用户控件，该控件为所有页面显示通用的页眉。第 3~4 行导入两个名称空间。接下来的 login 方法实现了所有必需的功能。第 21~30 行实例化了 OleDbConnection、OleDbCommand 和 OleDbParameters 对象。然后，将参数值设置为文本框（这里没有列出）中的用户名和密码。

在 try 语句块中，您打开并执行查询。如果用户合法，则使用 ExecuteScalar 方法从 reader 中读出其 UserID；否则，intId 将保持其初始值，即 0。

第 47~53 行的代码真正验证用户的身份。如果变量 intID 值不为 0，则说明用户合法，因此使用 Request.Cookie 对象存储账户 ID，并重定向到 account 页面。如果 intID 值为 0，表明信息不正确，因此显示错误消息。

接下来，需要更新 account.aspx，以动态地显示数据库中的数据。首先，不像“附加项目 1”那样使用静态的账户结余，而是使用数据库中的值。然后，需要在一个 DataGrid 中显示当前用户的交易信息。这里没有什么新内容——您应该很熟悉这些技术。

**注意：**您必须首先在数据库中输入数据（如账户结余和交易信息），然后该页面才能显示这些数据。

由于 UI 部分变化不大，因此清单 BP2.7 也仅列出代码声明块。

#### 清单 BP2.7 Account.aspx——显示账户结余和交易信息

```

1 <%@ Page language="VB" %>
2 <%@ Register TagPrefix="ASPNETBank" TagName="Header" src='header.ascx' %>

```

```

3 <%@ Register TagPrefix='ASPNETBank' TagName='Menu' src='nav.ascx' %>
4 <%@ Import Namespace='System.Data' %>
5 <%@ Import Namespace='System.Data.OleDb' %>
6
7 <script runat="server">
8
9  '*****
10  '
11  ' Account.aspx: lists account summaries and current balances
12  ' for the current user
13  '
14  '*****
15
16  'declare connection object
17  dim Conn as new OleDbConnection("Provider=' & _
18      "Microsoft.Jet.OLEDB.4.0;" & _
19      "Data Source=C:\ASPNET\data\banking.mdb")
20
21  '*****
22  ' When the page loads, display a welcome message in the
23  ' and the current balance of the account, as well as
24  ' any transactions
25  '*****
26  sub Page_Load(obj as object, e as eventargs)
27      GetBalance()
28      GetTransactions()
29  end sub
30
31  '*****
32  ' This function returns the balance of the account with
33  ' the specified user ID (stored in the cookie created
34  ' by forms authentication) and displays the value
35  ' in a label
36  '*****
37  sub GetBalance
38      dim decBalance as decimal
39      dim objCmd as OleDbCommand = new OleDbCommand _
40          ("spkretrieveBalance", Conn)
41      objCmd.CommandType = CommandType.StoredProcedure
42

```

```

43     'set parameters for stored procedure
44     dim objParam as OleDbParameter
45     objParam = objCmd.Parameters.Add("@UserID", _
46         OleDbType.BSTR)
47     objParam.Direction = ParameterDirection.Input
48     objParam.Value = Request.Cookies("Account").Value
49
50     try
51         objCmd.Connection.Open()
52         decBalance = CType(objCmd.ExecuteScalar, Decimal)
53         objCmd.Connection.Close
54     catch ex as OleDbException
55         lblMessage.Text = ex.Message
56     end try
57
58     lblBalance.Text = "<b>$" & decBalance.ToString &
59 "</b>"
60 end sub
61
62 *****
63 ' Returns all transactions for a given user id (stored
64 ' in the cookie (created by forms authentication), and
65 ' displays them in a datagrid
66 *****
67 sub GetTransactions
68     dim objCmd as OleDbCommand = new OleDbCommand _
69         ('spGetTransactions', Conn)
70     objCmd.CommandType = CommandType.StoredProcedure
71     dim objReader as OleDbDataReader
72
73     'set parameters for stored procedure
74     dim objParam as OleDbParameter
75     objParam = objCmd.Parameters.Add("@UserID", _
76         OleDbType.BSTR)
77     objParam.Direction = ParameterDirection.Input
78     objParam.Value = Request.Cookies("Account").Value
79
80     try
81         objCmd.Connection.Open()
82         objReader = objCmd.ExecuteReader

```

```

83
84     dgTransactions.DataSource = objReader
85     dgTransactions.DataBind()
86     catch ex as OleDbException
87         lblMessage.Text = ex.Message
88     end try
89 end sub
90 </script>

```

**分析：**代码真长！幸运的是，大部分代码都很简单，并且很容易编写。第 26 行的 Page\_Load 事件只是调用了清单中的其他两个方法：GetBalance 和 GetTransactions。

从第 37 行开始的 Getbalance 执行存储过程 spRetrieveBalance，传入验证用户身份时存储在 Cookie 中的 UserID 参数。这是由第 48 行的 Request.Cookies("Account").value 指定的。第 52 行执行查询，并以 decimal 数据类型取回结余，第 58 行显示了结余。

GetTransactions 方法与 Getbalance 类似，只是执行的是存储过程 GetTransaction，并返回指定用户的所有交易信息。同样，再一次使用了存储在认证 Cookie 中的 ID，如第 78 行所示。在第 82 行执行存储过程后，数据被绑定到 DataGrid。

该清单没什么新的内容。您早已知道如何执行存储过程和传入参数，所以对这种工作应该能够得心应手。图 BP2.3 是该清单的运行结果。

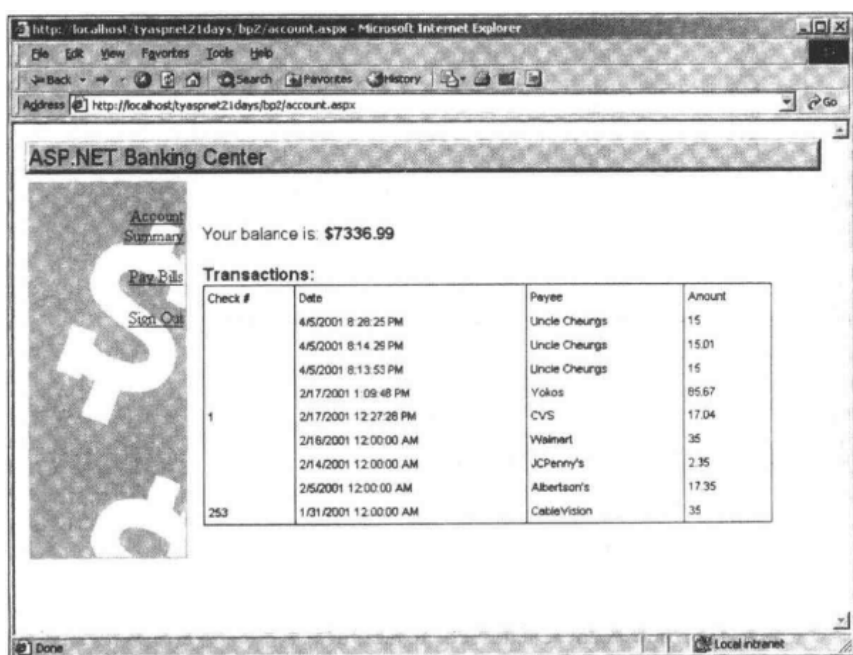


图 BP2.3 显示当前用户的账户结余和交易信息

最后，您必须更新 bills.aspx 页面，以便用户能够输入新的交易（换句话说就是在线支付账单）。当用户输入交易时，您必须检查该用户是否有足够的资金，如果资金足够，则从其结余中扣除交易额，然后更新交易表。注意，这里的在线交易只能付款，不能信贷。银行业还没有发展到任我们自己在线指定存款额的程度。

该页面应该十分简单——执行存储过程、提供参数，这些您都已很熟悉。您还必须执行其他几种检查，以确保用户输入的交易量合法。清单 BP2.8 列出了该页面的代码。

**清单 BP2.8 bills.aspx——输入交易信息**

```

1 <%@ Page language="Vb" %>
2 <%@ Register TagPrefix="ASPNETBank" TagName="Header" src="header.ascx" %>
3 <%@ Register TagPrefix="ASPNETBank" TagName="Menu" src="nav.ascx" %>
4 <%@ Import Namespace="System.Data" %>
5 <%@ Import Namespace="System.Data.OleDb" %>
6
7 <script runat="server">
8
9  '*****
10  '
11  ' Bills.aspx: Allows user to pay bills online
12  '
13  '*****
14  'declare connection object
15  dim Conn as new OleDbConnection("Provider=" & _
16    "Microsoft.Jet.OLEDB.4.0;" & _
17    "Data Source=C:\ASPNET\data\banking.mdb")
18
19  '*****
20  ' Event handler for submit button. Determines if amount
21  ' is valid, and if so, calls necessary functions to
22  ' update balance and add transaction
23  '*****
24  sub PayBill(obj as object, e as EventArgs)
25    dim decBalance as Decimal = GetBalance
26    dim decAmount as Decimal = lbAmount.Text
27
28    if decAmount < 0 then
29      lblMessage.Text = '<font color=red>The ' & _
30        "transaction amount cannot be negative!" & _
31        "</font>"
32      exit sub
33    end if
34
35    if decAmount <= decBalance then
36      UpdateBalance(decBalance - decAmount)

```

```

37         AddTransaction(tbPayee.Text, decAmount)
38
39         lblMessage.Text = '<font color=red>Transaction " & _
40             "added.</font>'
41         tbAmount.Text = ''
42         tbPayee.Text = ''
43     else
44         lblMessage.Text = "<font color=red>You do not " &
45             'have enough funds to complete this " & _
46             'transaction!</font>'
47     end if
48 end sub
49
50 *****
51 ' This function returns the balance of the account with
52 ' the specified user ID (stored in the cookie created
53 ' by forms authentication)
54 *****
55 function GetBalance as Decimal
56     dim decBalance as decimal
57     dim objCmd as OleDbCommand = new OleDbCommand _
58         ("spRetrieveBalance", Conn)
59     objCmd.CommandType = CommandType.StoredProcedure
60     dim objReader as OleDbDataReader
61
62     'set parameters for stored procedure
63     dim objParam as OleDbParameter
64     objParam = objCmd.Parameters.Add("@UserID", _
65         OleDbType.BSTR)
66     objParam.Direction = ParameterDirection.Input
67     objParam.Value = Request.Cookies("Account").Value
68
69     try
70         objCmd.Connection.Open()
71         decBalance = CType(objCmd.ExecuteScalar, Decimal)
72         objCmd.Connection.Close()
73     catch ex as OleDbException
74         lblMessage.Text = ex.Message
75     end try
76

```



```

77     return decBalance
78 end function
79
80 '*****
81 ' Adds a transaction into the database, calling
82 ' spInsertTransaction and specifying amount, payee, and
83 ' user id
84 '*****
85 sub AddTransaction(strPayee as string, decAmount _
86     as Decimal)
87     dim objCmd as OleDbCommand = new OleDbCommand _
88         ("spInsertTransaction", Conn)
89     objCmd.CommandType = CommandType.StoredProcedure
90
91     'set parameters for stored procedure
92     dim objParam as OleDbParameter
93     objParam = objCmd.Parameters.Add("@Date", _
94         OleDbType.Date)
95     objParam.Direction = ParameterDirection.Input
96     objParam.Value = Datetime.Now
97
98     objParam = objCmd.Parameters.Add("@Amount", _
99         OleDbType.Decimal)
100    objParam.Direction = ParameterDirection.Input
101    objParam.Value = decAmount
102
103    objParam = objCmd.Parameters.Add("@Payee", _
104        OleDbType.BSTR)
105    objParam.Direction = ParameterDirection.Input
106    objParam.Value = strPayee
107
108    objParam = objCmd.Parameters.Add("@UserID", _
109        OleDbType.BSTR)
110    objParam.Direction = ParameterDirection.Input
111    objParam.Value = Request.Cookies("Account").Value
112
113    try
114        objCmd.Connection.Open()
115        objCmd.ExecuteNonQuery()
116    catch ex as OleDbException

```

```

117         lblMessage.Text = ex.Message
118     finally
119         objCmd.Connection.Close()
120     end try
121 end sub
122
123 '*****
124 ' Updates the account balance
125 ' calls spUpdateBalance
126 '*****
127 sub UpdateBalance(decNewAmount as Decimal)
128     dim objCmd as OleDbCommand = new OleDbCommand _
129         ("spUpdateBalance", Conn)
130     objCmd.CommandType = CommandType.StoredProcedure
131
132     'set parameters for stored procedure
133     dim objParam as OleDbParameter
134     objParam = objCmd.Parameters.Add("@NewBalance", _
135         OleDbType.Decimal)
136     objParam.Direction = ParameterDirection.Input
137     objParam.Value = decNewAmount
138
139     objParam = objCmd.Parameters.Add("@UserID", _
140         OleDbType.BSTR)
141     objParam.Direction = ParameterDirection.Input
142     objParam.Value = Request.Cookies("Account").Value
143
144     try
145         objCmd.Connection.Open()
146         objCmd.ExecuteNonQuery()
147     catch ex as OleDbException
148         lblMessage.Text = ex.Message
149     finally
150         objCmd.Connection.Close()
151     end try
152 end sub
153 </script>

```

**分析：**第 24 行中的 PayBill 方法是 Submit 按钮的 Click 事件的处理程序。该方法获取文本框中的信息，然后核实交易量不为负，同时超过账户结余，然后调用适合的方法来更新结余，并将交易添加到数据库中。

接下来的是 GetBalance 函数（第 55 行），该函数与 Account.aspx 中的相应函数类似，只是结余作为一个 decimal 值返回，而不是输出结余。

第 85 行中的 AddTransaction 方法执行了存储过程 spInsertTransaction，并提供合适的参数值。最后，UpdateBalance 方法执行存储过程 spUpdateBalance。

在该页面中输入一笔交易，然后回到账户摘要页面，将可以在 DataGrid 中看到一笔新的交易以及新的结余。

## 总 结

上面是您需要做的所有修改，现在该应用程序已是一个完整的银行业程序，具有在线付款和在线交易查看功能。在设计开始就规划好应用程序的功能将很有帮助。当开始开发程序时，首先要创建所有必要的存储过程。尽管这项工作有些繁琐，但这对保证应用程序强大的设计结构是必不可少的。

尽管现在该应用程序已功能齐备，但仍有一些可改进的地方。您可以添加验证控件来检查收款人和用户输入的交易量是否有效。您还可以给 DataGrid 添加过滤器，以便用户只能查看特定时间段内的交易。

现在，所有的钱都存在一个支票账户中，您可以在应用程序中添加另一种账户，如储蓄账户。您可以将账户信息存储在同一个 tblAccounts 表中，并添加一个指示账户类型的字段。

您或许已注意到，清单中有很多重复的代码，并且这些代码分散在几个页面中。下一部分将介绍将 ASP.NET 页面组件化的知识，通过将常用代码封装在模块中来解决这种问题。

- 第 15 章 使用业务对象
- 第 16 章 创建 Web 服务
- 第 17 章 使用 Web 服务并确保其安全
- 第 18 章 配置和部署 ASP.NET 应用程序
- 第 19 章 将内容和代码分开
- 第 20 章 调试 ASP.NET 页面
- 第 21 章 确保 ASP.NET 应用程序的安全



## 第 15 章

# 使用业务对象

本章将进入新的 ASP.NET 开发领域。前两部分介绍了 ASP.NET 基础知识和数据访问机制。接下来将介绍高级主题，让您能够创建完整的 ASP.NET 应用程序。

本章介绍业务对象，一种可用于 ASP.NET 应用程序中的组件。这些组件与前面使用的组件（如 ADO.NET 或 XML 对象）之间的差别在于，您从空白开始创建这些组件，供自己使用。本章介绍业务对象的用途、如何创建和使用它们。

本章介绍以下内容：

- 什么是组件和业务对象？
- ASP.NET 如何使用组件？
- 为何要使用组件？
- 如何创建并使用业务对象？
- 如何使用非 .NET 环境中创建的组件？

### 15.1 组件简介

组件是可以在不同的应用程序中重复使用的对象，它们通常具备真实世界中对象的特征。回到第 3 章中关于钟表厂的类比，为组装一个钟表，需要使用大量的组件，如法条、铰链、玻璃、木材和钟摆等，然后将它们组装起来。ASP.NET 应用程序以及常规的应用程序，与此完全相同——它们由大量的组件构成，这些组件被组装起来，形成一个统一的整体。

如果只使用一个组件来制作钟表——整个钟表由一块木料构成，则钟表将无法很好地工作——指针不动、钟摆不摆等等。将钟表分为不同的部分——秒针、分针、时针、钟摆、钟面后，将更容易组装，并使之正常工作。同时，修理起来也更容易，例如如果分针断了，只需拆下它，换一个新的即可。或者，如果开发出了更高级的钟摆时，只需用它替换旧的钟摆即可，省时又省钱。

但这种方法也存在一些消极因素，组件过多会不必要地增加复杂性，抵消带来的好处。还有必要将钟摆分成多个部件吗？细分后，并不会更好。

在 ASP.NET 中，组件是可重用的代码块，它可增强应用程序的功能或提供新的功能。第 5 章开发的用户控件便属于组件，Code-behind 表单也可以看作是组件，甚至 Web 控件也是组件，您可以将其插入到页面中。组件用于描述真实的对象，如日历或书籍。在 ASP.NET 页面中，文本框或数据库结果集也被看作组件。

### 15.1.1 业务对象是什么

新术语：业务对象（business object）是这样的组件，即封装了可用于应用程序的代码。用于提供非 UI 功能的代码被称为业务逻辑（business logic）或业务规则（business rule），因此实现业务逻辑的组件被称为业务对象。

例如，在“第二部分复习”中的最后一个附加工程中，所有与数据库交互的代码都是业务逻辑，它们用于在 UI 和数据库之间进行通信。理想情况下，这种逻辑应该与 ASP.NET 分开，作为一个单独的业务对象，ASP.NET 页面只用于实现 UI 以及处理应用程序客户端的信息。

在电子商务站点中，用于从运输公司检索运输费的对象是一个典型的业务对象。开发人员可以将这种逻辑放在 ASP.NET 页面中，但如果以后运输费的计算方法发生变化时，将难以进行修改（更不用说其可重用性也很差）。明智的开发人员将创建一个运输组件，供任何 ASP.NET 应用程序使用。该组件从运输公司的数据库中检索信息，给电子商务应用程序提供其需要的所有信息。该组件是可重用的，同时修改只需在一个地方进行，而不会影响 ASP.NET 页面。

### 15.1.2 为何使用组件

您可能听说过三层应用程序模型，该模型将应用程序分成三层（有时不太明显）：UI 或表示层、业务逻辑/对象层和数据层。就开发 Web 应用程序而言，该模型非常适合，而且实现起来并不太难。图 15.1 说明了该模型。

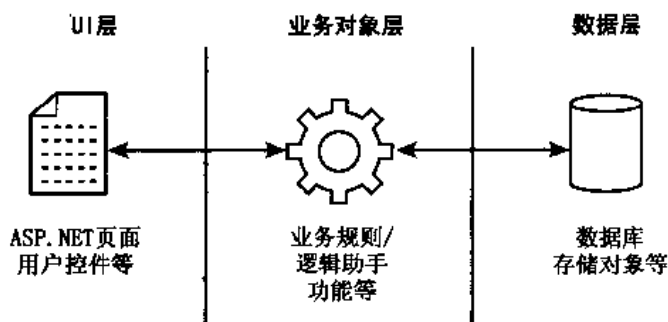


图 15.1 由 UI 层、业务逻辑层和数据层组成的三层模型

这就像一部戏剧。第一层是舞台上的演员，他们为观众提供“UI”，以吸引观众的注意力，给观众以体验等等。这是观众能见到的一层。

第二层由提供指示和提示的人组成，包括管弦乐团、舞台管理人员、导演等等。这些人与舞台上的演员打交道，但观众看不到。他们指挥演员，并给演员提供材料。

最后，第三层由负责布景和素材的人员组成，包括编剧、舞美、布景师等。这些人一道提供了戏剧素材。观众看不到这些人，而只能见到他们的作品。

这种维系表演的模型分工明确、井然有序。如果没有演员，表演将无法进行；如果没有编剧和舞美，将没有表演素材；如果没有舞台管理等组成的中间层，演员将难以表演，必须有另一层的人员来接管这些工作。

这种模型也适用于开发 Web 应用程序。少了其中的某一层，应用程序将难以开发。在电子商务站点中，第一层为 UI（表单、购物车、图形等）；中间层（业务层）由控制定价、运输费等逻辑组成；第三层（数据层）由数据库中的存货清单组成。缺少其中的任何一层时，都必须由其他层来完成其工作。

具体地说，在中间层使用业务对象让您能够更好地定义和分离应用程序。ASP.NET 不再包含大量难懂的、不是用于处理 UI 的代码。ASP.NET 页面用于为用户提供可视化界面，为何要将那些与 UI 不相关的代码放在页面中呢？

在前几章，您正是这样做的。诸如数据访问逻辑等功能可以放在中间层的业务对象中。虽然，在许多例子中，功能很简单，没必要引入第三层，而增加复杂性。业务对象非常适合用于封装非 UI 逻辑，但您必须搞清楚，应用程序是否复杂到需要使用第三层。就执行功能而言，组件还提供了效率更高的机制。例如，在第 5 章中的日历控件只需要编写几行 UI 代码，您还可以根据设计方案，显示一个完全交互性的日历。您不要担心日历的交付、星期的填充以及每月天数的计算等工作，这些工作都由组件为您完成了。

同样您在创建组件时，应该为使用它的用户完成所有的幕后工作，让他们不必为这些东西操心，他们只需使用它。即使开发人员和用户是同一个人，这样做也提供了易于实现功能的机制。

当然，还有显而易见的好处。代码的重用性提高了，应用程序将更小、更紧凑。独立于 ASP.NET 来编译代码可以提高性能，同时维护也更容易——只需一次修改业务逻辑，便可以在整个应用程序中生效。另外，作为 .NET 框架的一部分，如果需要，可以以您创建的对象为父类派生出其他类，对其进行扩展。

虽然有时候层之间的界线非常明显，但有时候，开发人员必须弄清楚 UI 功能和业务逻辑之间的界线。本章的范例将使这种界线尽可能清晰，但有时候这需要开发人员做出判断。

### 15.1.3 ASP.NET 如何使用组件

ASP.NET 使用被称为组合体仓库（assembly cache）的/bin 目录存储编译后的对象。在本章的后面，开发好业务对象后，将对其进行编译，并将其放在该目录中。ASP.NET 应用程序启动时，将自动装载/bin 目录中的所有对象，这让您能够在页面中使用开发好的组件。

也可以使用 web.config 文件手工将不在该目录中的对象装载到页面中，但这方面的内容不在本书的讨论范围之内。在大多数情况下，没有理由不将对象放到目录/bin 中。

对象被装载后，就可以像内置对象一样使用它们。例如，System 名称空间及其所有的类被编译到一个对象文件中，该文件被存储在全局组合体仓库中。在页面中使用这些对象时，可以导入名称空间 System，或使用完整的名称来引用对象，例如 System.Integer。您也可以以相同的方式使用对象，这将在本章后面介绍。

## 15.2 创建业务对象

创建业务对象和创建 code-behind 表单非常相似。对象只不过是使用 VB.NET（或您喜欢的任何语言）开发的类而已，它们被组织为逻辑分组。

让我们开始开发吧。清单 15.1 显示了一个简单对象的框架。

清单 15.1 一个业务对象的简单框架

```
1 Imports System
2 Imports System.Data
3 Imports System.Data.OleDb
4
5 Namespace TYASPNET
```



```

6
7 Public Class Database
8 End Class
9
10 End Namespace

```

分析：将该文件保存为 database.vb。该对象将用于为页面提供抽象数据库功能，如连接到数据库、查询数据库、返回数据等。换句话说，它代表一个真实的数据库，因此具备定义一个数据库的所有属性和方法。

其格式与 code-behind 表格类似。首先，导入对象中要使用的名称空间。在这个例子中，需要使用 System、System.Data 和 System.Data.OleDb（分别在第 1、2 和 3 行）。第 5 行声明了一个当前对象所属的名称空间。名称空间 TYASPNET 来自何方呢？不来自任何地方——这是您刚才创建的。扩展 .NET 框架就是这样简单——只需声明您将使用一个新的名称空间就可以创建该名称空间。当您要为应用程序开发其他的对象时，只需将这些对象插入到该名称空间中即可。因此，它将是一个包括所有业务逻辑的逻辑分组。

注意：例如，名称空间 System.Data.OleDb 包含很多类，有 OleDbCommand、OleDbConnection 和 OleDbDataAdapter 等。名称空间只不过是用于按逻辑将类进行分组而已。

如果愿意，也可以指定一个已有的名称空间，如 System.Web，这样创建的对象将被加入到该名称空间中。记住，名称空间用于将对象按逻辑进行分组，而您将创建的对象不属于任何已有的名称空间，因此您创建自己的名称空间。您创建的每个 ASP.NET 应用程序都可以有不同的名称空间。

应 该	不 应 该
应为应用程序中的对象创建唯一性的名称空间。	不要将您创建的对象放在已有的名称空间中，这将带来混乱，并引发问题。

最后，第 7 行声明了一个类，当前它为空白。后面将给它加上内容。别忘了加上类和名称的结束标记。

让我们编译该对象，以便能够在 ASP.NET 页面中使用它。为此，需要使用 .NET 框架 SDK 自带的 VB.NET 编译器。单击 Start 按钮，然后执行 Run 命令，并输入 cmd.exe，以打开一个命令提示符窗口。首先切换到应用程序的根目录（C:\inetpub\wwwroot\tyaspnet21days），并使用下面的命令创建一个新的 bin 目录：

```
mkdir bin
```

后面将把对象放在该目录中。接下来，切换到存储该文件的目录，如 C:\inetpub\wwwroot\tyaspnet21days\day15。

输入下面的命令，然后按 Enter 键：

```

vbc /t:library /out:..\bin\TYASPNET21days.dll /r:System.dll
/r:System.Data.dll Database.vb

```

VB.NET 编译器有很多选项，并且非常复杂，因此本节只介绍上述命令中使用的选项（很可能，在 ASP.NET 开发中只需要了解这些即可）。

vbc.exe 是 VB.NET 编译器的名称。您使用该程序将文件编译成 DLL，以便在应用程序中使用。/t 指定要将其编译成哪种文件，library 意味着对象将被其他应用程序使用——自己无法使用。/h 选项也可以是 exe，这将创建一个可执行文件。

`/out` 指定将编译结果存储在哪个目录下以及存储的文件名。您需要将编译结果保存在目录 `bin` 中，因此以相对于当前目录方式指定了路径。 `/r` 选项指的是引用。在清单 15.1 中，您引用了名称空间 `System`、`System.Data` 和 `System.Data.OleDb`，而这些名称空间存储在文件 `System.dll` 和 `System.Data.dll` 中，因此，编译时需要加上这两个文件的引用。

**警告：**不但在代码中需要导入名称空间，在编译命令中，也要使用引用。没有这些引用，`Import` 命令将不会做任何工作，编译对象时将出错。

最后，指定了要将对象编译成名为 `Database.vb`。该类文件将被编译并自动装载到组合体仓库中，以便在页面中使用它。您将看到与图 15.2 类似的输出。

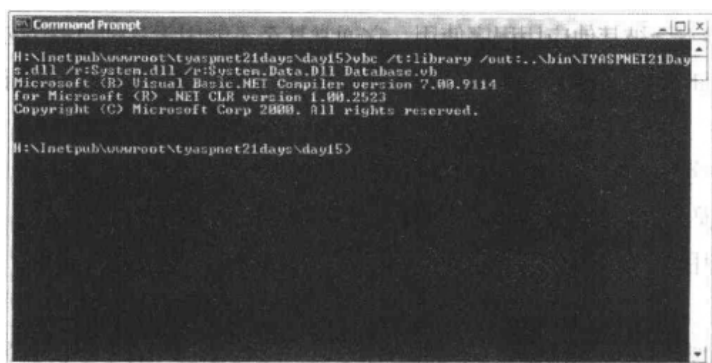


图 15.2 使用 VB.NET 编译器来编译对象

**注意：**如果还不习惯使用 VB.NET 编译器，请不用担心。您将经常使用该命令，当语法不同时，书中将指出。更详细的信息，请参考 .NET 框架 SDK 文档。

清单 15.2 列出了一个实现该对象的简单页面。

#### 清单 15.2 使用业务对象

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as EventArgs)
5         dim objDatabase as TYASPNET.Database
6         lblMessage.Text = "Object created"
7     end sub
8
9 </script>
10
11 <html><body>
12     <asp:Label id="lblMessage" runat="server" />
13 </body></html>
```

**分析：**第 5 行基于刚创建的业务对象声明了一个新对象。您还不能使用该对象完成任何工作，因为您还没有为它实现任何方法。在本章后面的“开发业务对象”一节中，您将添加一些方法。

**注意：**您使用完整的名称 `TYASPNET.Database` 来引用该对象。通过导入自定义的名称空间（就

像导入其他名称空间一样), 可以只使用对象名称 (不需要名称空间名称) 来引用对象。例如, 假设在第 2 行加入了如下代码:

```
<%@ Import Namespace = "TYASPNET" %>
```

则可以在第 5 行使用下面的代码:

```
dim objDatabase as new DataBase
```

### 15.2.1 为何需要编译 Database 对象

在编译之前, 创建 Database 对象和创建 code-behind 表单极其类似。当然, code-behind 表单不需要编译。为何需要编译业务对象呢?

code-behind 表单是一个 ASP.NET 类, 其中包含只适用于特定页面的代码, 因为它定义了特定 UI 的功能。这种类不会被其他应用程序使用, 它通常甚至不能用于其他 ASP.NET 页面。当用户请求该 ASP.NET 页面时, 页面和 code-behind 表单都将被编译。然后, 页面将使用编译后的 code-behind 表单的方法和属性。因此 code-behind 表单确实是被编译的。

根据定义, 业务对象将被用于许多不同的地方, 它不属于某个特定的页面。它也不必只包括与 ASP.NET 交互的代码。它将用于整个应用程序, 甚至其他的应用程序中。例如, 数据库对象将被用于许多地方。在使用它之前, 必须编译所有的代码, 因此需要首先编译该对象, 因为它不与任何页面相关联。

您也可以预先编译 code-behind 表单, 然后在 ASP.NET 中将其作为一个业务对象使用, 但为何这样劳神呢? code-behind 表单将之用于一个地方, 因此它肯定会被编译, 因此没有必须预先对其进行编译。

## 15.3 开发业务对象

让我们回过头来, 为清单 15.1 中创建的业务对象实现一些功能。因为该对象代表的是一个数据库, 因此首先需要连接字符串属性。您将在 Database 类中创建该属性, 如清单 15.3 所示。

**清单 15.3 连接字符串属性**

```
1 Imports System
2 Imports System.Data
3 Imports System.Data.OleDb
4
5 Namespace TYASPNET
6
7     Public Class Database
8         public ConnectionString as String
9         private objConn as OleDbConnection
10        private objCmd as OleDbCommand
11        ...
```

使用该业务对象的页面将能够在任何时候设置和检索当前数据库的连接字符串。另外还需要加入两个代表数据对象的私有变量。由于这两个变量是私有的, 因此该类中的方法可以访问它们, 但类之外的方法则不能访问。换句话说, 使用该业务对象的 ASP.NET 页面将不能访问这些变量, 但可

以访问共有变量 `ConnectionString`。

**提示：**可以将类看作一个汽车引擎，任何私有东西都在内面，在外面不可见。您可能知道引擎内有活塞，但您看不到，也接触不到它（除非您打开引擎，那是另一码事）。但公有成员暴露在外面，这包括油量表、火花塞和空气过滤器。

接下来，需要一个从数据库那里检索信息的方法。您将创建一个函数，它执行指定的 SQL 语句，将数据作为一个 `OleDbDataReader` 返回。您还需要一个执行数据库操作，但不返回任何数据的函数，例如 `Insert` 和 `Update` 语句。清单 15.4 列出了两个用于访问数据的方法：一个返回信息，另一个则不。

**清单 15.4 执行选择查询**

```

1      public function SelectSQL(strSelect as string) as _
2          OleDbDataReader
3      try
4          objConn = new OleDbConnection(ConnectionString)
5          objCmd = new OleDbCommand(strSelect, objConn)
6          objCmd.Connection.Open
7          return objCmd.ExecuteReader
8          objCmd.Connection.Close()
9      catch ex as OleDbException
10         return nothing
11     end try
12 end function
13
14 public function ExecuteNonQuery(strQuery as string) as _
15     Boolean
16     try
17         objConn = new OleDbConnection(ConnectionString)
18         objCmd = new OleDbCommand(strQuery, objConn)
19         objCmd.Connection.Open()
20         objCmd.ExecuteNonQuery
21         objCmd.Connection.Close()
22         return true
23     catch ex as OleDbException
24         return false
25     end try
26 end function
27
28 End Class
29
30 End Namespace

```

SelectSQL 是一个标准函数。它打开一条到 OleDbCommand 对象的连接，执行查询，并返回一个 DataReader，如第 18 行所示。第 19 行关闭连接。如果出现问题，try 语句块将捕获错误，并返回 nothing。

ExecuteNonQuery 与此类似，但不返回 DataReader，而是根据命令是否成功执行而返回 true 或 false。

使用与前面相同的命令重新编译该对象：

```
vbc /t:library /out:..\bin\TYASPNET21\days.dll /r:System.dll
/r:System.Data.dll Database.vb
```

接下来修改前面的 ASP.NET 页面，清单 15.5 是对清单 15.2 修改后的结果。它设置新数据库对象的属性，并在一个 DataGrid 中显示数据。

#### 清单 15.5 使用对象的方法

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     sub Page_Load(obj as object, e as EventArgs)
7         dim objDatabase as new TYASPNET.Database
8
9         objDatabase.ConnectionString = "Provider='& _
10             "Microsoft.Jet.OLEDB.4.0;" & _
11             "Data Source=C:\ASPNET\data\banking.mdb"
12
13         dim objReader as OleDbDataReader
14         objReader = objDatabase.SelectSQL _
15             ("Select * from tblusers")
16
17         if not objReader is nothing then
18             DataGrid1.DataSource = objReader
19             DataGrid1.DataBind()
20             objReader.Close
21         end if
22     end sub
23
24 </script>
25
26 <html><body>
27     <asp:Label id="lblMessage" runat="server" />
28
```

```

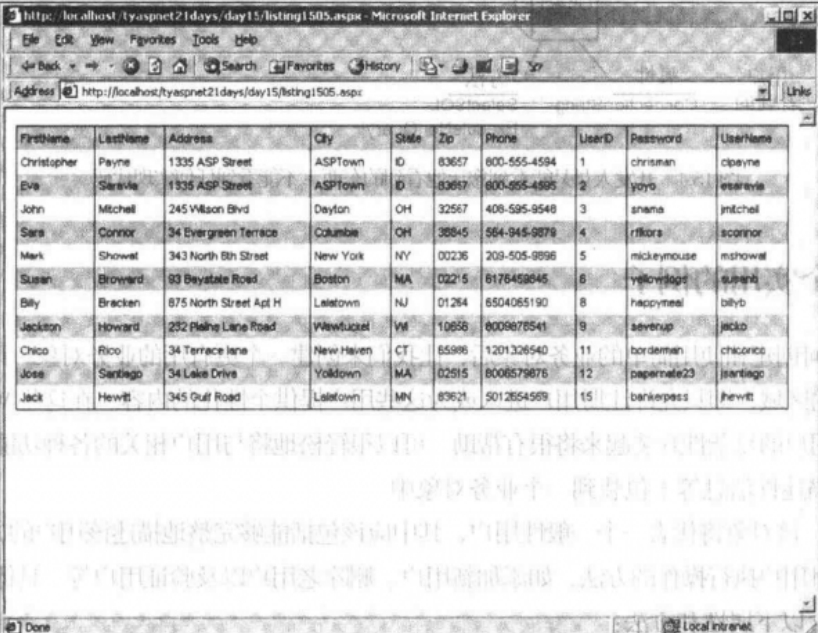
29 <asp:DataGrid id="DataGrid1"
30     runat="server" BorderColor="black"
31     GridLines="Vertical" cellpadding="4"
32     cellspacing="0" width="100%"
33     Font-Name="Arial" Font-Size="8pt"
34     HeaderStyle-BackColor="#cccc99"
35     ItemStyle-BackColor="#ffffff"
36     AlternatingItemStyle-BackColor="#cccccc" />
37 </body></html>

```

分析：第 7 行声明了一个数据库对象，然后在第 9 行设置其 `ConnectionString` 属性。第 14 行执行清单 15.4 中创建的 `SelectSQL` 语句，然后将得到的结果放在一个 `OleDbDataReader` 中。

记住，如果发生错误，`SelectSQL` 函数将返回 `nothing`。在第 17 行，使用数据阅读器执行任何工作之前，您检查其是否确实包含结果。如果包含，则在第 19 行将数据绑定到一个 `DataGrid`，并在第 20 行关闭该阅读器。该清单在浏览器中生成的结果如图 15.3 所示。

注意：字段 `Username` 和 `Password` 是在“第二部分复习”中加入的。有关如何加入以及为何要加入这些字段的信息，请参考其中的附加工程。



FirstName	LastName	Address	City	State	Zip	Phone	UserID	Password	Username
Christopher	Payne	1335 ASP Street	ASPTown	ID	83657	800-555-4584	1	chrismen	cipayne
Eve	Sierke	1335 ASP Street	ASPTown	ID	83657	800-555-4585	2	yoyo	esierke
John	Mitchell	245 Wilson Blvd	Dayton	OH	32567	408-595-9548	3	snana	jmtchell
Sara	Connor	34 Evergreen Terrace	Columbus	OH	38845	564-945-9878	4	rfkora	sconnor
Mark	Showal	343 North 8th Street	New York	NY	00236	208-505-8998	5	mickymouse	mshowal
Susan	Broward	83 Baystate Road	Boston	MA	02215	617-645-9845	6	yellowlegs	susana
Billy	Bracken	875 North Street Apt H	Laitown	NJ	01264	850-405-1190	8	hacpyntal	bilby
Jackson	Howard	252 Raing Line Road	Wewtucal	VM	10658	800-987-6541	9	adverup	jackson
Onico	Rico	34 Terrace lane	New Haven	CT	65986	1201-326-540	11	borderman	oniconic
Jose	Santiago	34 Lake Drive	Yoldtown	MA	02915	800-657-9876	12	papermate23	jsantiago
Jack	Hewitt	345 Gulf Road	Laitown	MN	83621	501-254-569	15	bankerpass	jhe Witt

图 15.3 业务对象返回数据，然后您可以在 ASP.NET 使用这些数据

使用自定义的 `Database` 对象与使用其他对象的方法相同。您创建该对象的一个实例，设置一些属性，并执行一些方法。

您可能会说，为何仅为了在 ASP.NET 页面节省几行代码而这样做呢？对于简单例子（例如前面的例子），没必要为数据库查询而创建一个业务对象。当应用程序非常人时，业务对象将变得非常

复杂。在这种情况下，创建业务对象将可以节省大量编写 ASP.NET 页面的时间。

另外，现在您可以在任何 ASP.NET 页面中使用该对象，以获得结果。您可以修改连接字符串，以连接到其他数据库，还可以指定不同的 SQL 语句，以返回不同的数据。虽然在这个 ASP.NET 页面中只节省 10~15 行代码，但当 10 个乃至 20 个 ASP.NET 页面都可以使用该对象时，将节省多少行代码？

您可能会问的另一个问题是，在可以在 ASP.NET 页面中直接使用 OleDb 数据类的情况下，为何要创建一个数据库业务对象呢？为何不创建一些更有用的东西？实际上，您创建的业务对象很有用。它不但让用户只需一行代码便可以检索数据，而且将所有使用数据对象的复杂性隐藏起来了。用户（在这里，是另一个程序员）不用担心 OleDbCommands 和连接，也不用担心将语句包装在一个 try 语句块中（或捕获错误）。您创建的类为用户处理了所有这些工作。图 15.4 说明了用户看到的东西和看不到的东西。

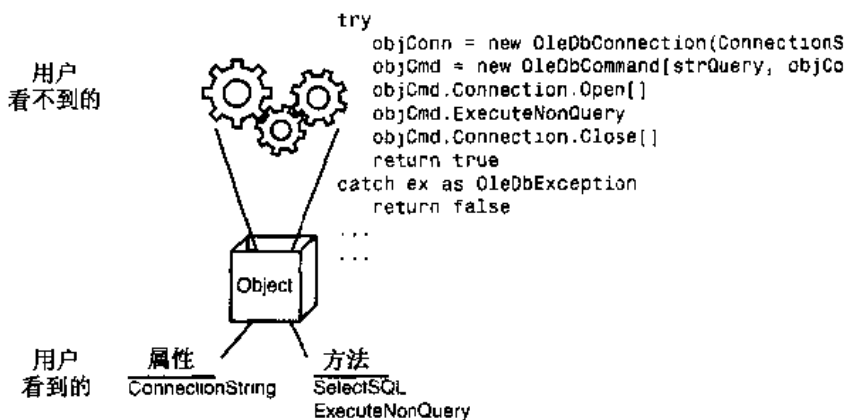


图 15.4 开发人员只能看到您让他看到的东西，不能看到任何实现代码

## 15.4 一个实用的例子

学习如何创建和使用简单的业务对象后，让我们来创建一个更有用的业务对象。许多 Web 站点都有这样的区域，即只允许注册用户进入或为这些用户提供个性化的内容。在这些 Web 站点中，将表示注册用户的复杂性封装起来将很有帮助。可以很轻松地将与用户相关的各种功能（如登录、注销、更新描述性信息等）包装到一个业务对象中。

实际上，该对象将代表一个一般性用户，其中应该包括能够完整地描述该用户的方法和属性，还应该包括对用户执行操作的方法，如添加新用户、删除老用户以及验证用户等。具体地说，您希望该对象包含以下属性和方法：

- 一个表示用户身份的对象，包括其全名、用户名、ID 等；
- 添加、删除和更新用户的方法；
- 验证用户的方法。

将使用两个独立的类来实现上述功能。其中一个类表示用户所有的详细信息，另一个实现前面描述的方法。这样让您能够将用户的详细信息看作一个可以操纵的独立的实体。图 15.5 描述了这两个对象之间的关系。

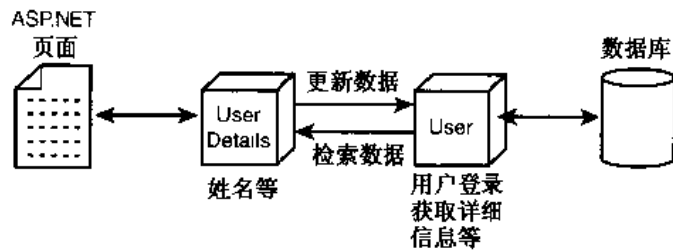


图 15.5 UserDetail 对象将存储用户的详细信息，而 User 对象将实现功能

您将使用第 8 章创建的用户数据库，文件的开始部分如清单 15.6 所示。

#### 清单 15.6 UserDetail 类

```

1 Imports System
2 Imports System.Data
3 Imports System.Data.OleDb
4
5 Namespace TYASPNET
6
7   Public Class UserDetails
8     public FirstName as string
9     public LastName as string
10    public UserName as string
11    public Password as string
12    public UserID as string
13  End Class

```

**分析：**将该文件保存为 user.vb。该文件的开始与前一个对象相同，它导入必要的名称空间（1-3 行），并声明了自己的名称空间（第 5 行）。Userdetail 类（从第 7 行开始）包含了表示用户所需的所有属性（为简化问题，这并不包括数据库中的所有属性，如果愿意，您可以添加其他的属性）。

接下来创建 User 类，其中包含所需的所有函数，如清单 15.7 所示。

#### 清单 15.7 User 类

```

14 Public Class User
15   public function Login(UserName as string, Password as _
16     string) as string
17     dim intId as string = "0"
18     dim Conn as new OleDbConnection('Provider=" & _
19       "Microsoft.Jet.OLEDB.4.0;" & _
20       "Data Source=C:\ASPNET\data\banking.mdb")
21
22     dim objCmd as OleDbCommand = new OleDbCommand _
23       ('SELECT UserID FROM tblUsers WHERE " & _
24       "UserName = '" & UserName & "'" AND ' & _

```



```
25         "Password = '" & Password & "'", Conn)
26     dim objReader as OleDbDataReader
27
28     try
29         objCmd.Connection.Open()
30         objReader = objCmd.ExecuteReader
31
32         do while objReader.Read
33             intID = objReader.GetInt32(0).ToString
34         loop
35     catch ex as OleDbException
36         throw ex
37     end try
38
39     return intID
40 end function
41
42 public function GetDetails(UserID as integer) as _
43     UserDetails
44     dim Conn as new OleDbConnection("Provider=" & _
45         "Microsoft.Jet.OLEDB.4.0;" & _
46         "Data Source=C:\ASPNET\data\banking.mdb")
47
48     dim objCmd as OleDbCommand = new OleDbCommand _
49         ("SELECT FirstName, LastName, UserName, * & _
50         'Password FROM tblUsers WHERE UserID = " & _
51         UserID, Conn)
52     dim objReader as OleDbDataReader
53
54     try
55         objCmd.Connection.Open()
56         objReader = objCmd.ExecuteReader
57     catch ex as OleDbException
58         throw ex
59     end try
60
61     dim objDetails as new UserDetails
62
63     while objReader.Read()
64         objDetails.FirstName = objReader.GetString(0)
```

```

65         objDetails.LastName = objReader.GetString(1)
66         objDetails.UserName = objReader.GetString(2)
67         objDetails.Password = objReader.GetString(3)
68         objDetails.UserID = UserID.ToString
69     end while
70     objReader.Close
71
72     return objDetails
73 end function
74
75 public function Update(objDetails as UserDetails, _
76     intUserID as integer) as boolean
77     dim objOldDetails as new UserDetails
78     objOldDetails = GetDetails(intUserID)
79
80     with objDetails
81         if .FirstName = "" then
82             .FirstName = objOldDetails.FirstName
83         end if
84         if .LastName = "" then
85             .LastName = objOldDetails.LastName
86         end if
87         if .Username = "" then
88             .Username = objOldDetails.Username
89         end if
90         if .Password = "" then
91             .Password = objOldDetails.Password
92         end if
93     end with
94
95     dim Conn as new OleDbConnection("Provider=" & _
96         "Microsoft.Jet.OLEDB.4.0;" & _
97         "Data Source=C:\ASPNET\data\banking.mdb")
98
99     dim strSQL as string = "UPDATE tblUsers SET " & _
100         "FirstName = '" & objDetails.FirstName & "', " & _
101         "' & _
102         "LastName = '" & objDetails.LastName & "', " & _
103         "Username = '" & objDetails.Username & "', " & _
104         "[Password] = '" & objDetails.Password & "' " & _

```

```

105         "WHERE UserID = " & intUserID
106         dim objCmd as OleDbCommand = new OleDbCommand _
107             (strSQL, Conn)
108
109         try
110             objCmd.Connection.Open()
111             objCmd.ExecuteNonQuery()
112         catch ex as OleDbException
113             throw ex
114         finally
115             objCmd.Connection.Close()
116         end try
117
118         return true
119     end function
120 End Class
121
122 End Namespace

```

**分析：**这个类非常大，而其中还没有包括所有的方法。所幸的是，对您而言，它应该很容易理解，您可以很快看懂它。

15 行的 `Login` 函数接受用户名和密码，将其与数据库中的用户名和密码进行核对。如果它们有效，则返回相应的 ID。

第 42 行的 `GetDetails` 函数接受一个 `UserID` 值，并从数据库中检索用户的详细信息。第 61 行创建了前面定义的 `UserDetails` 类的一个实例。然后将其属性设置为从数据库中返回的值，并将该 `UserDetails` 对象返回给用户。通过将这些值存储在该 `UserDetails` 对象中，为用户提供了易于存取这些值的机制。

`Update` 函数更新用户数据库中的一条记录，它接受一个 `UserDetails` 对象，并使用其属性的值来更新记录。如果用户只想更新其中的一个值，情况将如何呢？如果其中的大部分值不需要更新，则不用要求用户指定所有的值。因此，您允许用户只指定 `UserDetails` 对象中要修改的值，该函数将自动使用原来的值填充其他字段。这种功能的实现代码如 77-93 行所示。

77 行的 `objOldDetails` 对象存储了用户原来的属性值，这是使用函数 `GetDetails` 从数据库中检索得到的。接下来的一系列 `if` 语句检查用户未指定的用户详细信息，并将其设置为原来的值。这样，就可以创建一个使用 `UserDetails` 对象的 SQL 语句，而不用担心会丢失原来的值。

**提示：**80 行的关键字 `with object` 意味着接下来前面带句点的所有变量都属于 `object` 对象。例如，下述代码：

```

objDetails.FirstName = " "
objDetails.LastName = " "

与下面的代码等价：

With objDetails
    .FirstName = " "

```

```
.LastName = " "
```

当需要设置该对象的许多属性的值时，后一种方法可以省去许多输入。

第 99 行创建了更新所需的 SQL 语句，然后在 try 语句块中执行该查询。如果更新成功，将返回 True。

将该文件保存为 User.vb，然后切换到相应的目录，并使用下面的命令对其进行编译：

```
vb /t:library /out:..\bin\User.dll /r:System.dll
/r:System.Data.Dll User.vb
```

**提示：**如果愿意，可以将多个文件编译到一个 DLL 中，例如：

```
vb /t:library /out:..\bin\TYASPNET.dll /r:System.dll
/r:System.Data.Dll User.vb Database.vb
```

上述命令将前面创建的用户对象和数据库对象编译到一个文件中。与多个 DLL 相比，这更容易跟踪。事实上，要将多个文件编译到同一个 DLL 中，不用指定每个文件的名称，而可以使用通配符。例如，下面的将当前目录中所有扩展名为 .vb 的文件编译到一个 DLL 中：

```
vb /t:library /out:..\bin\TYASPNET.dll /r:System.dll
/r:System.Data.Dll *.vb
```

只是一定要确定您想将当前目录下所有这样的文件都编译到一起。

清单 15.8 列出了一个使用这些对象（UserDetails 和 User）的简单 ASP.NET 页面范例。

#### 清单 15.8 使用前面创建的用户对象

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         if Not Page.IsPostBack then
6             dim objUser as new TYASPNET.User
7             dim objDetails as new TYASPNET.UserDetails
8
9             objDetails = objUser.GetDetails(1)
10            lblMessage.Text = "Hello ' & objDetails.FirstName & "!"
11            ObjDetails.FirstName & "!'
12        end if
13
14    end sub
15
16    sub Update(obj as object, e as eventargs)
17        dim objUser as new TYASPNET.User
18        dim objDetails as new TYASPNET.UserDetails
19
20        objDetails.FirstName = tbName.Text
21        if objUser.Update(objDetails, 1) then
```

```

22     objDetails = objUser.GetDetails(1)
23     lblMessage.Text = "Hello " & _
24         objDetails.FirstName & "!"
25     else
26         lblMessage.Text = "Update failed!"
27     end if
28 end sub
29 </script>
30
31 <html><body>
32     <form runat="server">
33         <asp:Label id="lblMessage" runat="server" /><p>
34             Change your name?<br>
35             <asp:Textbox id="tbName" runat="server" /><br>
36             <asp:Button id="btSubmit" runat="server"
37                 OnClick="Update" Text="Submit" />
38     </form>
39 </body></html>

```

当用户首次打开该页面时，将看到一条欢迎消息，其中包含其 UserID。在 Page\_Load 事件处理程序中，您实例化两个新对象，并使用 GetDetails 方法返回一个 UserDetails 对象。然后显示一条简单的欢迎消息。

用户在文本框中输入新的姓名，并单击 Submit 按钮后，update 方法将接管控制权。它将 UserDetails 对象的 name 属性设置为用户输入的新值，然后调用 User.update 方法。如果更新成功，该方法将返回 true，并使用新的姓名显示一条欢迎消息。图 15.6 显示了输入新姓名后的结果。

现在，ASP.NET 页面非常简短，如果不使用业务对象，则相应的页面将长得更多。现在，页面中只包含直接与 UI 交互的代码。同样，其他需要显示这项功能的页面也不用直接处理数据库对象。

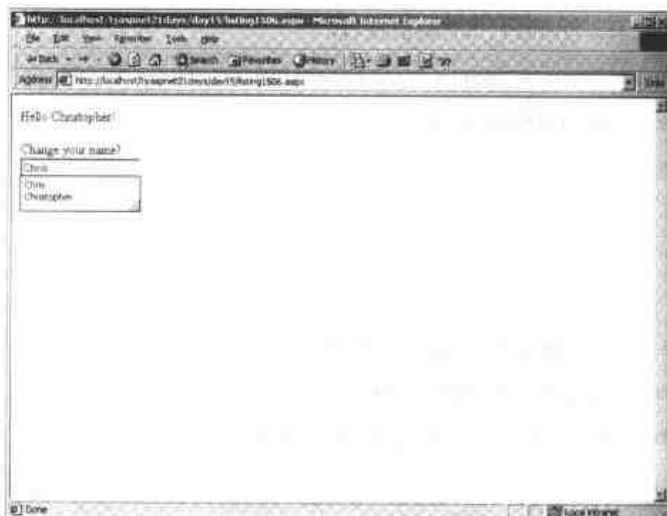


图 15.6 在 ASP.NET 页面中，使用操纵用户信息的用户对象只需要几行代码

如果需要在 User 对象中添加其他功能,或数据库的结构发生了变化。如果分别在每个 ASP.NET 页面中实现上述功能,则需要修改很多地方,已反映数据库的变化。使用 User 对象,只需修改.vb 文件,并重新编译它即可。您无需修改任何 ASP.NET 页面,便可以继续运行它们。

使用业务对象的另一个好处是,它们可用于另一个 Web 服务器。您可以将 User 对象卖给其站点也需要使用该对象的其他开发人员(当然,您可能首先需要添加一些功能)。

**提示:** 您可以使用本章前面的“开发业务对象”一节中创建的 Database 类,而不用在 User 类中重新创建所有的数据库命令和对象。编译 User 对象时,应该使用下面的命令:

```
vb /t:library /out:..\bin\User.dll /r:System.dll
/r:System.Data.dll /t:TYASPNET.dll User.vb
```

具体地说,您需要加入到 TYASPNET.DLL 中 Database 对象的引用。

#### 15.4.1 一些需要考虑的因素

注意到在 user.vb 文件中,对于一些值(如连接字符串),您使用的是硬代码。这类变量不属于该业务对象,因此应该将其放置在一个单独的配置文件中,如 web.config。业务对象可以使用 HttpContext 对象的 GetConfig 方法轻松地获得该字符串(参考第 18 章“配置和部署 ASP.NET 应用程序”)。这使得应用程序灵活得多——您无需每当数据库发生变化时都重新编译业务对象。

业务对象不能依赖于您使用的数据库,并不意味着对象应与数据库毫无关联。在业务对象中包括针对特定数据库的 SQL 语句便是一种不错的选择,而且人们经常这样做。业务对象应该依赖于数据库的格式,但不应该依赖于数据库的实际位置。

当然,所有的 SQL 语句都可以移到存储过程中,然后对象向存储过程传递参数。只要过程的名称保持不变,就可以添加更多的功能,而不用修改业务对象的源代码,并重新编译它。

开发业务对象时,应尽可能将逻辑与功能分开。前面的对象表示一个用户,因此其中应该包括将数据显示给用户的代码以及返回数据的 SQL 语句。它应该只包括用户特有的方法和属性,在您设计应用程序时,这将给您带来极大的帮助。

另外,不要在一个对象中实现过多的功能。例如,用户信息可以很容易地与用户方法分开,因此您不是在一个对象中实现所有的功能,而是使用两个相互依赖的对象。

## 15.5 使用非.NET 组件

还记得吗,在.NET 框架中编译对象时,将生成描述对象的元数据。CLR 使用这些数据来装载这些对象,而无需开发人员的任何干预。只需将对象放到 bin 目录下便万事大吉了。

然而,不在.NET 框架下开发的老式对象(通常被称为组件对象模型(COM)对象)并没有这项功能。CLR 不会自动管理并装载这些对象,也没有元数据告知 CLR 关于这些对象的信息。开发人员必须使用 REGSVR32.exe 手工注册这些组件,REGSVR32.exe 将信息加入到 Windows 注册表(包含关于计算机上安装的所有应用程序和组件的信息)中。

**注意:** 之所以说 COM 对象是“老式的”,是因为用来创建这些对象的技术比.NET 技术旧,而不是因为对象本身旧。

COM 对象是在非.NET 环境中创建的,因此不由 CLR 管理,所以被称为不可管理的代码(unmanaged code)。由于上述原因,ASP.NET 弄清如何使用这些对象时稍微困难些。例如,您不能在设计阶段设置对象的属性。

ASP.NET 仍然支持使用这些对象，这是通过 `HttpServerUtility` 类的 `CreateObject` 方法实现的。该方法接受一个描述对象在注册表中位置的字符串，并使用该字符串来设置对象的属性。该字符串被称为 `progID`。该方法的语法如下：

```
Server.CreateObject(progID)
```

例如，在传统的 ASP 页面中，I/O 操作是使用脚本库中的 `FileSystemObject` 实现的。该对象提供了一些与第 13 章介绍的类相同的功能。要在页面中使用该对象，必须使用下面的代码：

```
dim objFSO as object
objFSO = Server.CreateObject("Scripting.FileSystemObject")
```

例如，清单 15.9 演示了如何显示给定文件的路径。

#### 清单 15.9 使用 COM 对象

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4   sub Page_Load(obj as object, e as eventargs)
5       dim objFSO, objFile
6       objFSO = Server.CreateObject _
7           ('Scripting.FileSystemObject')
8       objFile = objFSO.GetFile _
9           (Server.MapPath("../day13/log.txt"))
10
11       lblMessage.Text = objFile.Path
12   end sub
13 </script>
14
15 <html><body>
16   <asp:Label id = "lblMessage" runat = "server"/>
17 </body></html>
```

分析：在第6行和第8行，分别创建了一个 `Scripting.FileSystemObject` 和一个 `Scripting.File` 对象。然后便可以使用这些对象来实现一些 I/O 功能，如显示文件的完整路径。图 15.7 显示了该清单的结果。

但使用 `FileSystemObject` 没有多大的意义，因为 .NET 框架中有更好的对象（如 `TextReaders` 和 `TextWriters`），它们是完全面向对象的。只有对那些已经创建了许多自定义 COM 对象的开发人员来说，才能从 .NET 使用 COM 对象的功能受益。许多拥有 Web 站点的公司常常使用多个 COM 对象来执行本章前面的业务对象的功能。重新编写所有这些 COM 对象涉及的工作量实在太大了。

许多其他应用程序也通过 COM 来提供其功能。例如，您可以使用 Word 提供的方法在 ASP.NET 页面中创建并操纵 Word 文档。Word 所有的 COM 对象都是不可管理的代码，但这并不能阻止您在 .NET 框架中使用它们。

新术语：但使用 COM 对象有一个缺点。因为没有与这些对象关联的元数据，ASP.NET 在判断这些对象接受和返回的数据类型方面有些困难，因此经常需要转换数据类型。这种处理被称为整理（marshalling）。由于增加了处理开销，因此这通常会降低性能，应尽可能避免使用 COM。

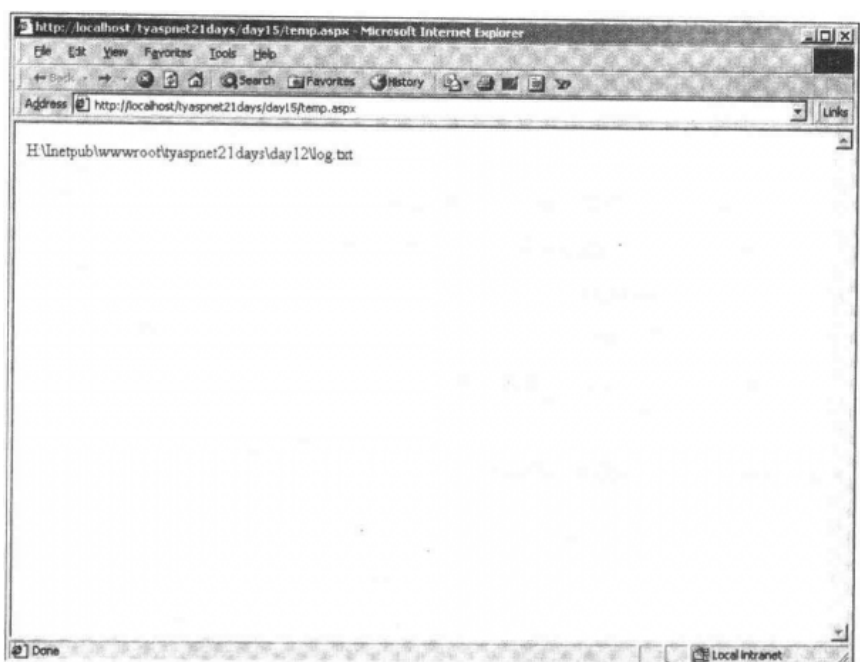


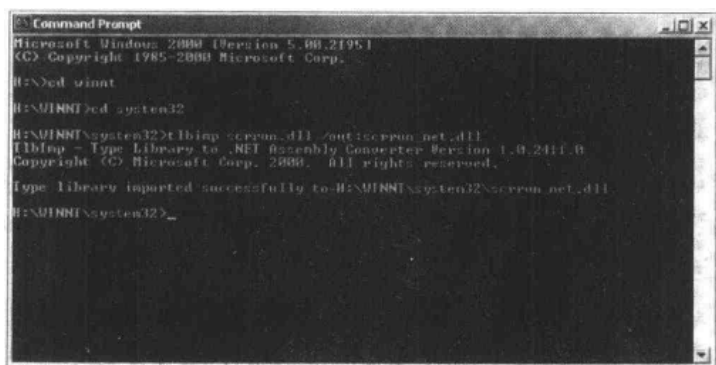
图 15.7 ASP.NET 支持使用老式的、不可管理的 COM 对象

所幸的是，.NET 框架自带了一个可以将不可管理的 COM 对象转换为可管理的 .NET 对象的程序，该程序被称为类型库导入器（Type Library Importer），它检查 COM 对象，并创建合适的元数据，以便您可以在 ASP.NET 应用程序中使用它。这种应用程序是使用文件 `tlbimp.exe` 执行的。

例如，假设您想在 ASP.NET 页面中使用 `FileSystemObject`，并避免整理带来的开销。该对象存储在文件 `scrnrun.dll` 中，而该文件通常位于目录 `c:\winnt\system32` 下。请打开一个命令提示符窗口，并切换到该目录，然后输入下面的命令：

```
tlbimp scrnrun.dll /out:scrnrun_net.dll
```

这将使用 COM 文件 `scrnrun.dll` 创建一个名叫 `scrnrun_net.dll` 的新文件，其中包含合适的元数据。输出与图 15.8 类似。

图 15.8 `tlbimp.exe` 将旧的 COM 对象导入到 .NET 框架中

将该文件复制到组合体仓库（`c:\inetpub\wwwroot\tyaspnet21days\bin`）中。现在，您便可以修改前面的清单，以使用新的可管理的 `FileSystemObject`。清单 15.10 列出了一个使用新的 `FileSystemObject` 的范例。



**清单 15.10 使用导入的 COM 对象**

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as eventargs)
5         dim objFSO as new Sscrun_net.FileSystemObject
6         dim objFile as Sscrun_net.File
7         objFile = objFSO.GetFile _
8             (Server.MapPath("../day13/log.txt"))
9
10        lblMessage.Text = objFile.Path
11    end sub
12 </script>
13
14 <html><body>
15     <asp:Label id = "lblMessage" runat = "server"/>
16 </body></html>
```

现在可以重新使用熟悉的实例化对象的方式。注意，现在 `FileSystemObject` 属于名称空间 `Sscrun_net`，它与您创建的新文件的名称相同。与前一个清单相比，性能将得到极大的提高，并允许您像 .NET 对象一样使用旧的 COM 对象。

**注意：**.NET 框架并未真的将 COM 对象转换为 .NET 对象，它只是提供了一个包装，以生成必要的元数据，并没有修改原有的对象。因此，CLR 仍然不完全支持 COM 对象。

## 15.6 这不是 ASP

传统 ASP 的开发人员可能震惊。ASP.NET 最大的好处包括创建 DLL 时，不用重新启动 Web 服务器、不用担心版本冲突、也不用使用 `REGSVR32.exe` 来注册 DLL。

以前，所有的 COM 对象都需要手工进行注册，同时要使所作的修改生效，必须重新启动服务器。事实上，这使大部分服务器远程管理无法实现。开发人员或管理员必须在服务器上进行修改，而在 ASP.NET 中，再也不需要这样做。

正如第 7 章介绍过的，ASP.NET 自带了一个新的配置系统，只要有合适的权限，任何人都可以很容易地修改它。运行阶段，`bin` 目录中的文件将自动被装载，而且很容易修改这些文件，因为它们没有被服务器锁定。您只需不断地覆盖它们，而无需有任何担心。

这也意味着部署 Web 应用程序非常简单，只需将必要的文件复制到合适的目录中，最新的版本便会运行。您无需重新启动服务器（或者当您不在现场时，指导其他人重新启动服务器）。

在 .NET 框架中开发 DLL 与以前类似，但也有一些区别，如新增的名称空间和新的 VB.NET 编译器。但大部分设计和编写代码的方法仍然相同。

如果您是 ASP 开发人员，您也将为您的 COM 对象库不会失效而庆幸。ASP.NET 可以通过您熟悉的 `Server.CreateObject` 方法使用这些对象。这创建的是晚期联编对象，就像 ASP 中一样。您也可以使用类型库导入器给这些 COM 对象提供一些 CLR 和早期联编方面的支持。但在大多数情况下，您

将想转换这些对象，以使用.NET 框架中的许多特性。

在 ASP.NET 中，开发和使用业务对象的许多概念仍然没变。变化最大的是对象所处的环境，这使得在实现对象方式上，出现了一些受欢迎的变化。

注意，您创建的.NET 业务对象并不向后兼容，因此不能在传统 ASP 页面中使用清单 15.1 中创建的 Database 对象。记住，.NET 对象被编译成 MSIL，然后由 CLR 编译成本机机器指令。旧的 ASP 应用程序没有 CLR 带来的益处，不能理解 MSIL，因此无法使用新的对象。

## 15.7 总 结

本章介绍许多高级 Web 开发方面的知识。使用和创建业务对象向成为 ASP.NET 开发高手迈出了一大步。正如您在本章中学到的，在 ASP.NET 中，使用业务对象非常简单！

组件是可重用的编程元素，可以给 ASP.NET 页面提供额外的功能。它们通常代表真实世界中的对象，虽然不要求是这样。业务对象是实现代码的组件，可作为三层应用程序的中间层。

创建业务组件与创建 code-behind 表单类似。首先使用 VB.NET 创建一个类，然后添加必要的功能，以描述对象。可以在命令行中使用 VB.NET 编译器对业务对象进行编译，然后在 ASP.NET 页面中使用它们，就像使用其他对象一样——声明一个新对象类型的变量，使用关键字 new 实例化它，然后使用其属性和方法。

COM 对象是在非.NET 环境中开发的旧对象，这些对象不支持 CLR，也不包含元数据。在 ASP.NET 页面中，可以通过 Server.CreateObject 方法，并提供有效的 progID 来使用它们。

通过使用类型库导入器（tlbimp.exe）来生成元数据，可以使 COM 对象响应 CLR。类型库导入器并不真正修改已有的 COM 对象，而是在 COM 对象和 CLR 之间创建一个解释层。这消除了整理对象的开销，并让您能够在 ASP.NET 页面中就像.NET 对象一样使用它们。

下一章将介绍 Web 服务，一种新的通过 ASP.NET 在 Internet 上递送应用程序的方式。

## 15.8 问与答

问：什么情况下，应该考虑将通用的 ASP.NET 功能移到业务对象中？

答：这是一个非常个性化的问题。许多开发人员认为，所有的非 UI 功能都应该放到业务对象中，而不管应用程序的情况如何。但这取决于应用程序的规模，如果应用程序只有几个页面，则没有必要使用业务对象，而增加复杂性。如果能从理论上看清层之间的界线，则使用业务对象可能是个不错的主意。

问：能够创建在实例化时能够接受参数的对象吗？

答：完全可以。VB.NET 提供了名叫构造函数的方法，让您能够指定如何实例化对象。在 VB.NET 中，该方法叫做 new。例如：

```
Class MyObject
    Public overloads Sub New()
        'do something
    End Sub
    Public overloads Sub New(strString as string)
        'do something
    End Sub
End Class
```

```
...
end sub
End Class
```

这个类有两个构造函数——一个默认的，不接受任何参数；另一个接受一个字符串参数。在第二个构造函数中，您可以使用字符串执行任何操作。在 ASP.NET 页面中，您可以使用下面的代码：

```
dim objMyObject as new MyObject("Hello")
```

只有几件事情您需要知道。如果类是从另一个类继承而来的，则一定要调用 `MyBase.new`，告诉 VB.NET 创建一个基于这个类的新对象。另外，只能使用您创建的构造函数实例化对象。如果没有提供不接受任何参数的构造函数，则不能使用下面的代码：

```
dim objMyObject as new MyObject
```

如果您根本不提供任何构造函数，则 VB.NET 将自动为您创建一个不接受任何参数的构造函数。另外，如果您提供了多个构造函数，则一定要使用关键字 `overload`，以避免由于有两个名称相同的方法而引发错误。

更详细的信息，请参考 .NET 框架 SDK 中的 VB.NET 文档。

问：如果找到 COM 对象的文件名？或者说如何找到 `progID` 值？

答：有时候，则很困难。最简单的方式是使用编程环境，如 VB.NET 或 Visual Studio。这些应用程序让您能够查看 COM 对象的类库，并找到文件名。

或者，您可以进入 Windows 注册表，并搜索需要的对象。在那里通常总能找到 `progID` 和文件名。

最后，您可以使用一个名叫 OLE/COM 对象查看器 (Object Viewer) 的应用程序 (MS Visual Studio 通常捆绑了该应用程序)，来确定必要的字符串。

对于 .NET 对象，方法要简单得多。通常，所有的 .NET 框架类都位于以名称空间命名的文件中，例如名称空间 `System` 中的对象都位于 `System.dll` 中，而 `DataGrid` 对象位于 `System.Data.dll` 中。

## 15.9 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 15.9.1 小测验

1. 三层应用程序模型由哪三层组成？
2. 判断正误：您手工编译对象是因为它是某个页面所特有的？
3. 编写命令行命令，将文件 `MyObject.vb` 编译到子目录 `bin` 中，并引用名称空间 `System` 和 `System.web`。
4. 下述那些代码应该放到一个业务对象中：一个实现 `DataGrid` 的类、一个从配置文件中检索连接字符串的方法、一个将信息写入到浏览器中的方法、一个描述对象实例数目的属性。
5. 判断正误：您使用 `Server.CreateObject` 来创建 .NET 框架中创建的对象。

### 15.9.2 练习

1. 修改本章创建的 `User` 对象，以使用本章前面创建的 `Database` 对象。

## 第 16 章

# 创建 Web 服务

前一章介绍了如何为 ASP.NET 页面创建组件,这些组件可用于通过 Internet 为 ASP.NET 页面提供功能。如果您希望不用通过 ASP.NET 页面,也能使用这些组件,该如何办呢?

Web 服务是一种新的部署应用程序的方式。使用 XML 和其他标准技术,Web 服务使得应用程序和组件能够与其他应用程序通信,而不管这些应用程序位于什么地方——无论是在同一台计算机中,还是位于地球的另一端。

本章将介绍如何创建 Web 服务,这是一种看待 Web 应用程序的新方式,是 ASP.NET 开发人员必须了解的一个主题。下一章将介绍如何在 ASP.NET 应用程序中使用这些服务。

本章将介绍以下内容:

- Web 服务是什么?
- Web 服务的工作原理。
- 什么情况下应使用 Web 服务?
- 如何创建 Web 服务?
- 如何使用已有的业务对象来创建 Web 服务?

### 16.1 Web 的工作方式——再访问

在本书的开始介绍了 Web 的基本操作方式,即请求/应答操作模型。客户(如浏览器)向 Web 服务器请求页面,后者通过应答将页面发回给客户。

接下来介绍了可编程 Web。ASP.NET 和其他技术让您能够在页面被请求时执行一些任务。通过在服务器上提供编程能力,您可以动态地为客户提供数据。虽然 ASP.NET 通过在顶层加入事件驱动机制来扩展了这种模型,但 Web 的基础机制并未改变:请求/应答。

ASP.NET 提供了一个前端,让用户能够与 Web 站点交互。在 UI 的后面(可能是业务对象中)存在一些功能强大的应用程序逻辑。如果某个站点包含的功能很有用,其他的程序员希望在自己的站点中使用这些功能,该如何办呢?或者说,有人想使用这些功能,但不能使用 UI(就想在命令提示符下工作),该如何办呢?在这种情况下,如果利用请求/应答模型。

这很容易。还记得 ASP.NET 页面与业务对象交互的方式吗?它们使用对象提供的方法和属性来执行某些功能,而对象可能返回一些信息,也可能不返回。实际上,您是发送执行某些功能的请求,并等待应答。

为何其他应用程序不能使用这种方式与对象交互呢？只需通过 Internet 发送命令，并等待应答即可。图 16.1 说明了这种概念。其思想非常简单，但能够实现这种想法的技术直到现在才出现。

一个更为真实的例子是，Web 站点应该能够向另一个站点发出请求，并等待应答。前一个站点可以与第二个站点中的方法和属性交互。图 16.2 通过一个股票报价服务系统说明了这种概念。用户向某个站点发出请求，后者再向证券交易站点发出请求。证券交易站点将数据返回给被用户请求的站点，后者再以任何其期望的方式将数据显示给用户。

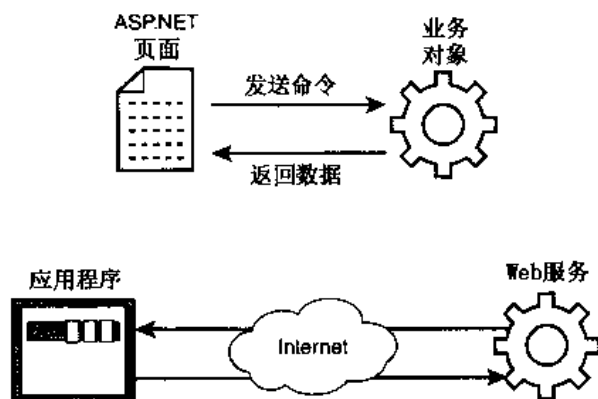


图 16.1 应用程序应该能够与 Web 服务交互，就像 ASP.NET 页面能够与业务对象交互一样

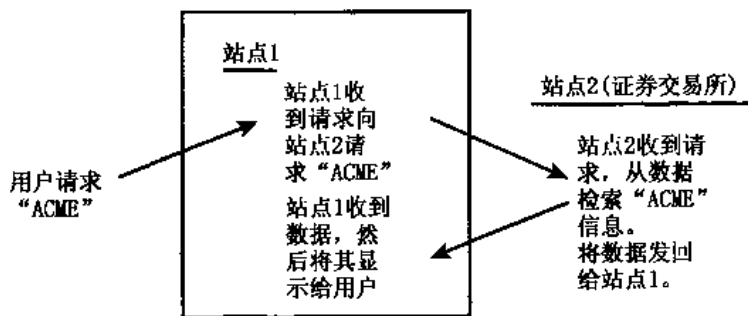


图 16.2 Web 服务让 Web 站点能够使用其他站点的功能

## 16.2 Web 服务简介

**新术语：**介绍 Web 服务之前，先介绍一些常规服务。当他人为您完成某项工作时，他便是在为您提供服务。例如，您到加油站将油箱加满，这种服务是由加油站为您提供的，因此您不必自己动手。想象一下，如果每个人都自己加油，情况将如何。这不是一种理想的办法。您到餐馆就餐，您可以自己做饭，但其中的一些工作您不愿意做。因此，服务是由公司或个人提供的增值工作，让您免于自己动手。

Web 服务与此相同。Web 站点能够为访问者，甚至其他站点提供服务。门户站点提供国内和国际新闻、天气信息、体育比赛情况和其他个性化的内容。它通过编辑不同来源的信息，将其放在同一个地方，来为访问者提供服务。然而，除非该门户站点预算和人手都非常充足，否则几乎不可能撰写各方面最新的内容。

相反，该门户站点将依赖于其他站点的内容，它只需提供显示机制即可。这样，该门户站点仍

然为用户提供服务，但它依赖于其他站点提供的服务。

在新闻报道方面，这已经是一种普遍使用的方式。美联社提供新闻服务，供报纸摘录。下次您阅读报纸时，请注意有关联社撰写的文章，这些文章便是来自新闻服务的。

在 Internet 上，这种系统还没有得到广泛应用，因为在服务如何通信方面非常复杂。许多公司尝试过创建专用的通信系统，以便交换服务。但这种系统关系复杂和昂贵，难以被普遍采纳。另外，Internet 上安全系统的结构也带来了一些问题，许多这种专用系统难以跨越防火墙（设计用于阻断未经授权通信）传输数据。

.NET 框架提供的 Web 服务为上述问题（以及其他一些问题）提供了解决之道。Web 服务是一种可编程对象（就像业务对象一样），通过 Internet 提供了可供大量系统访问的功能。Web 服务得以其作用的原因在于，它使用的是用于对象通信的标准技术。Web 服务不需要使用定制系统和专用机制，您所需要的只是一条 Internet 连接。

Web 服务基于这样的事实，即任何系统或应用程序都可以使用 HTTP 并可以使用和转换 XML，前者是 Internet 通信的标准协议，而后者是一个通过 Web 递送数据的标准。Web 服务使用 XML 来发送命令以及发送（接收）服务器上的对象中的信息。使用数据和发送命令的应用程序可以使用任何语言编写，可运行在任何计算机体系结构上，可简单可复杂。应用程序只需要知道服务器的位置（即其 Internet 地址）即可。

Web 服务提供了一种新的计算方式。您可以将各种对象和组件组装成一个应用程序，以同样的方式，开发人员也可以将来自不同地方的 Web 服务组装起来，并在应用程序中使用它们。现在，完全不同的平台可以轻松地进行通信，将世界各地的不同的系统连接在一起。

### 16.2.1 Web 服务方案

假设您创建了一个能够执行简单计算的组件，类似于第 2 章创建的计算器。该计算机执行简单的算术运算。而另一个 Web 站点创建了一个可以向家装商店订货的组件。假设这两个组件都是 Web 服务，则明智的开发人员可以将它们链接起来，创建一个让用户可以设计其居家环境的应用程序。需要计算时，用户使用您的 Web 服务中的计算功能，然后通过另一个 Web 服务向家装商店订货，这些工作都是在一个应用程序中完成的，用户无需知道这些功能来自何方。图 16.3 说明了这个例子。

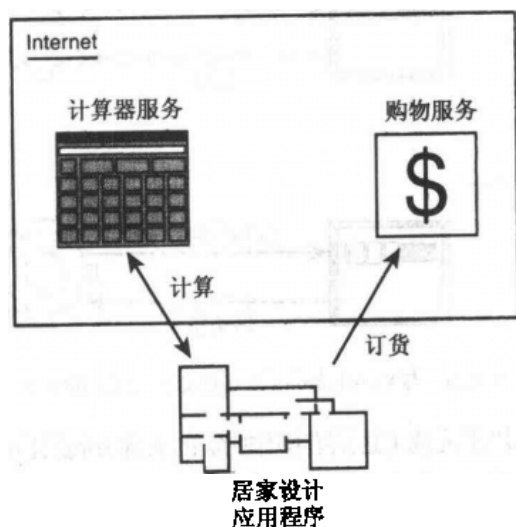


图 16.3 单个应用程序可以将多个 Web 服务连接在一起

Web 服务甚至可被 Internet 应用程序使用。假设某个电子商务站点需要计算客户订购的产品的运输费用。通常,该站点必须维护一个最新的表格,以便根据运货商、装货地点和优先次序等计算运输费用。有了 Web 服务,该站点可以使用一条简短的 XML 命令直接向运输公司发出请求,并立刻收到报价。Web 服务很轻松地将应用程序连接在一起,而这在以前是非常困难的。

### 16.2.2 Web 服务的编程模型

正如前面提到的,Web 服务使用 XML 进行通信。但通信到底是如何完成的呢?

新术语:通常发送的第一条信息涉及到一种名叫发现(discovery)的处理过程,即客户应用程序如何找到并检查 Web 服务。发现过程涉及到在服务和客户之间发送描述服务功能的消息。客户在使用该服务之前,需要知道这一点;而服务也告诉客户,它可以接受其他哪些消息。

发现过程并非必不可少的,例如,如果客户不希望其他应用程序了解它,则它可以禁用发现过程。这是一种保护措施,防止任何人都可以使用您创建的 Web 服务。

完成发现过程后,服务必须告诉客户,它期望收到什么(即它可以接受哪些命令)以及将发回什么。这一步是必不可少的,以便服务和客户都知道如何进行通信。这种数据被称为 Web 服务描述(service description)。在业务对象中,开发人员通常预先知道该对象将支持的命令(例如,通过提供的文档)。服务描述相当于 XML 格式的服务文档。

最后,将在客户和服务之间来回地发送 XML 格式的消息,发送给服务的消息中包括命令,而发送给客户的消息中则包括数据。图 16.4 说明了整个过程。

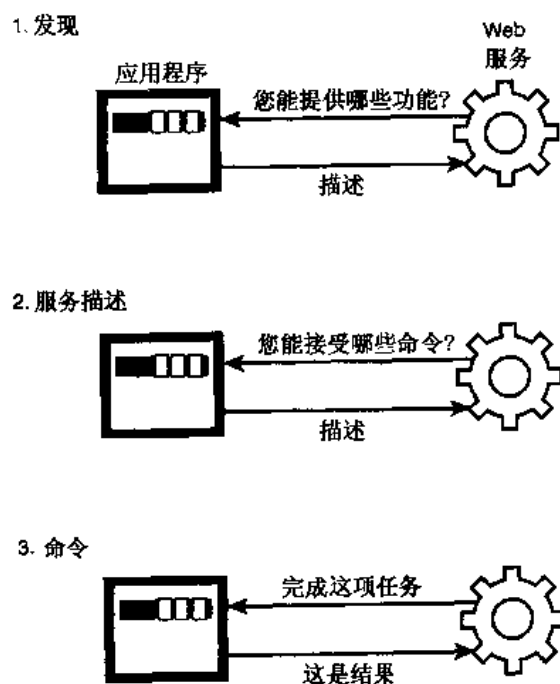


图 16.4 与 Web 服务的交互包括发现、描述和命令

所幸的是,ASP.NET 提供了完成上述所有操作所需的大部分底层设施。毕竟它能处理请求和应答、传输 XML 并使用业务对象,这使得它非常适合用于开发 Web 服务。您只需确定 Web 服务的功能并创建即可。

### 16.2.3 用于访问 Web 服务的协议

您知道, Web 服务依赖于 XML 格式的消息来发送数据和接收命令。具体地说, 它们在发现过程和服务描述中使用 XML, 但包含命令的消息并不一定要使用 XML。这种消息可以以其他两种方法发送: Http-Get 和 Http-Post, 这两种方法分别用于 ASP.NET 页面中的查询字符串和表单。

Web 服务支持三种与客户交互的协议: Http-Post、Http-Get 和简单对象访问协议 (Simple Object Access Protocol, SOAP)。深入讨论这些协议已超出本书的范围, 但您应该对这些协议及其对 Web 服务的影响有个基本的了解。

#### 1. Http-Get

Http-Get 是一种让客户能够通过 HTTP 与服务器通信的标准协议, 是您请求 Web 页面的方式。可以将 Http-Get 操作看作是客户从 Web 服务器那里获得网页。基本上, 客户向 Web 站点的 URL 发送 HTTP 请求, 而 Web 站点使用相应的 HTML 进行应答。处理请求所需的所有参数都通过查询字符串进行传递, 例如:

```
http://www.myserver.com/default.aspx?id=chris&sex=male
```

参数 id 和 sex 是作为 Web 服务器的输入传递的, 它们被附加在 URL 的后面。然后, ASP.NET 页面便可以使用下面的代码检索这些值:

```
Request.QueryString("id")
Request.QueryString("sex")
```

Web 服务可以使用 Http-Get 和查询字符串而不是 XML 消息, 来传递命令和方法。注意, 使用 Http-Get 发送的信息是 URL 的一部分。然而, Http-Get 的功能有限, 因为它只能发送名称/值对。

#### 2. Http-Post

该协议与 Http-Get 类似, 但不是将参数附加在 URL 的后面, 而是将其直接放置在 HTTP 请求消息中。通过 Http-Post 请求页面时, 客户发送一个包含其他消息的 HTTP 请求消息, 其中包括参数及其值。Http-Post 最常用于 HTML 表单。

例如, 假设有一个包含字段 id 和 sex 的 HTML 表单, 如下面的代码段所示:

```
<form method='post'>
  <input type="Text" id="id">
  <input type="Text" id="sex">
  <input type="Submit" id="btSubmit" Value="Submit" />
</form>
```

当用户提交该表单时, 浏览器将把文本框中的值加入到发送给服务器的 HTTP 请求消息中。这样, 服务器就可以使用下面的语法来取得这些值:

```
Request.Form("id")
Request.Form("sex")
```

和 Http-Get 一样, 该协议也只限于发送名称/值对。

**注意:** 也可以使用 Http-Get 来发送表单中的信息, 只需将 form method 指定为 Get, 这样表单便会将信息加入到查询字符串中, 而不是 HTTP 请求报头中。例如:

```
<form method="Get">
```

#### 3. SOAP

简单对象访问协议 (SOAP) 是一种相对较新的标准, 它使得客户和服务器能够相互发送数据。



不同于 Http-Post 和 Http-Get, SOAP 依赖于 XML 来转发其信息,而不是使用 HTTP 请求消息。这意味着 SOAP 不但能够发送名称/值对,还能发送更为复杂的对象,如各种复杂的数据类型、类、对象以及其他信息。

发送 SOAP 消息的方法与前面介绍的方法有很大的不同,Http-Get 通过查询字符串来发送信息,而 Http-Post 通过提交表单来发送。这两种方法都依赖于请求/应答操作模型。SOAP 信息也是通过 HTTP 传输的,但并不限于请求/应答模型。它可以用于发送任何类型的信息,而不管客户是否请求了信息。因此 SOAP 是一种非常灵活的发送数据的方法。

由于 SOAP 是基于 XML 的,因此可以在 Web 中使用它来传输数据。XML 只不过是纯文本,因此可以被发送到任何 HTML 页面可以到达的地方,包括通过防火墙。这是 Web 服务用来与客户通信的默认协议。

**注意:** 当对 SOAP 和 XML 进行比较时,许多人会感到迷惑。如果 SOAP 是一种如此好的发送消息的方法,为何不使用 SOAP,而是使用 XML 呢?它们之间的区别在于,XML 定义了数据格式,而 SOAP 定义了一种用于交换这些数据的协议。SOAP 依赖于 XML 来发送其消息。纯文本的 XML 对于传输大多数数据类型都是非常理想的。仅当需要传输诸如命令和指令等信息时,SOAP 才适用。

#### 16.2.4 为何使用 Web 服务

现在,您知道了 Web 服务是什么,那么为何要使用它们呢?

想象一下 10-15 年前,Internet 还未兴起时的情况。计算机系统通常是独立的实体,无法访问其他的系统。应用程序被设计为单机使用的,不具备任何共享数据的功能。公司内部或公司之间,很少通过计算机进行通信。即使是人与人之间的消息,也是通过手工递送的。

Internet 改变了全球的通信方式,同时改变了应用程序的设计方式。不以某种方式使用 Internet 的应用程序非常少,许多应用程序(如即时传信)依赖于 Internet 来为用户提供数据。

在通过 Internet 来连接用户方面,接下来要做的是以前面介绍的方式递送应用程序。一些公司已经尝试将传统的应用程序连接在一起,形成一个组合性实体。由于现在使用的传统应用程序的数目非常大,因此这项任务是非常艰巨的。

Web 服务为应用程序之间相互通信提供了一种非常简单的机制,它使得组件得以共享,而功能可以被递送给任何人,任何地方。想象一下,您再也不用在计算机上安装程序,而只需连接到 Web,并访问该服务即可。

所有这些都非常棒,但 Web 服务是如何使得 Web 开发更好的呢?其中的方法之一是通过代码重用。还记得吗,使用业务对象的原因之一是可以不断地重用这些代码,而不必每次都创建。诸如函数和子程序等分支逻辑也提高了代码的重用性(参见第 3 章)。使用 Web 服务,您可以重用其他人已经开发好的代码,而无需下载、复制和安装。省时省力,让您能够轻松地将功能加入到 ASP.NET 应用程序中。

Web 服务的另一个好处是,易于部署和维护。您不再需要在许多不同的系统上安装自定义对象或应用程序,用户将用于一个访问您的应用程序的标准框架——直接通过 Internet。另外,当需要修改时,您不必发布新的版本,只需修改 Web 服务,所有的客户都将自动使用修改后的 Web 服务(当然,除非客户依赖的功能被删除,在这种情况下,它们必须进行调整。但无论如何,对于开发人员而言,这一过程将更容易)。

## 16.3 创建 Web 服务

创建 Web 服务包括几个步骤，其中有创建实际的功能和服务描述。接下来的几节将通过一个简单的 Web 服务，介绍每一个步骤。

### 16.3.1 创建功能

Web 服务文件基本上是扩展名为 `.asmx` 的 VB.NET 源文件。Web 服务由一个从 `WebService` 类派生而来的类表示。清单 16.1 列出了一个非常简单的 Web 服务。

清单 16.1 一个简单的 Web 服务

```
1 <%@ WebService Language="VB" Class="Calculator" %>
2
3 Imports System.Web.Services
4
5 public Class Calculator : Inherits WebService
6     <WebMethod()> Public Function Add(intA As Integer, _
7         intB As Integer) As Integer
8         Return(intA + intB)
9     End Function
10 End Class
```

分析：请将该文件保存为 `c:\inetpub\wwwroot\tyaspnet21days\day16\Caluculator.asmx`。第 1 行包括一条编译指令 `@ WebService`，它与编译指令 `@ Page` 类似，声明该文件代表一个 Web 服务。该编译指令支持您熟悉的属性 `Language`（被设置为 `VB`）。该编译指令还支持另一个属性 `Class`，该属性用于指定要开发的 Web 服务类的名称。事实上，在第 5 行您可以看到，这个类的名称确实是 `Calculator`。稍后将介绍该属性的一个巧妙特性。

接下来，第 3 行导入了名称空间 `System.Web.Services`，这让您能够使用所有必须的 Web 服务方法和类。第 5 行定义的类必须是从该名称空间中的 `WebService` 类派生而来的。

注意：一个 `.asmx` 文件中可以包括许多类，但可能只有一个被用于 Web 服务——在最顶端使用编译指令 `WebService` 声明的那个类。

接下来，让我们看看类中的唯一一个函数。方法 `Add` 的声明非常标准，但有两个需要注意的地方。首先该方法必须声明为公用的，这意味着其他任何类或对象都可以使用它；否则，客户将无法访问该 Web 服务中的这个方法。

其次是属性 `<WebMethod()>`，它告诉 ASP.NET，该方法将作为一种服务供其他应用程序使用。这是一个非常重要的属性，必须加入，以便客户能够访问该方法，本章后面的“`WebMethod` 属性”一节将进一步介绍它。

因此希望客户能够访问的方法必须声明为公用的，并包含属性 `<WebMethod()>`。

让我们来看看该页面在浏览器中的显示结果，如图 16.5 所示。

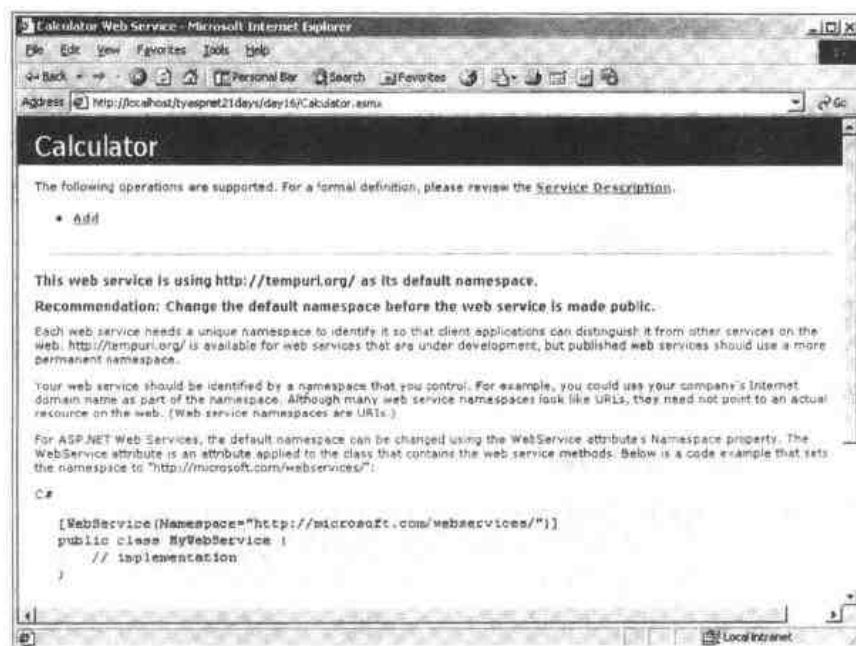


图 16.5 在 Web 浏览器中查看 .asmx 文件

这很有趣，代码发生了什么变化？其中的 UI 部分来自何方呢？

和 ASP.NET 页面一样，第一次被请求时，.asmx 文件也将被编译。然后每当页面被请求时，ASP.NET 便会提供服务描述。图 16.5 将是客户试图访问服务时看到的内容（XML 格式）。响应告诉客户类的名称（Calculator）以及公用的属性和方法。单击页面右边的 Service Description，您将发现地址后面将带一个查询字符串：

`http://localhost/tyaspnet21days/day16/Calculator.aspx?WSDL`

WSDL 告诉 ASP.NET，您想查看 XML 格式的服务描述。图 16.6 显示了该服务的 XML。

对于一个小型 Web 服务，其中的 XML 代码已经算是很多了。该 XML 文件使用一种叫做服务描述语言（Service Description Language, SDL）的标准格式告诉客户，使用该服务可以完成什么工作。这里不打算深入探讨该 XML 文件，但您应该注意其中一些熟悉的东西。例如，在最上面，有下述代码：

```

1  <s:element name='Add'>
2    <s:complexType>
3      <s:sequence>
4        <s:element name='intA' type='s:int' />
5        <s:element name='intB' type='s:int' />
6      </s:sequence>
7    </s:complexType>
8  </s:element>
9  <s:element name='AddResponse'>
10    <s:complexType>
11      <s:sequence>
12        <s:element name='AddResult' type='s:int' />

```

```

13      <s:sequence>
14      <s:complexType>
15      </s:element>

```

分析：第一个元素描述了方法Add及其接受的参数。第二个元素描述了方法的响应，被称为AddResponse。文件中的其余内容描述了如何使用Http-Get、Http-Post和SOAP协议访问该方法。

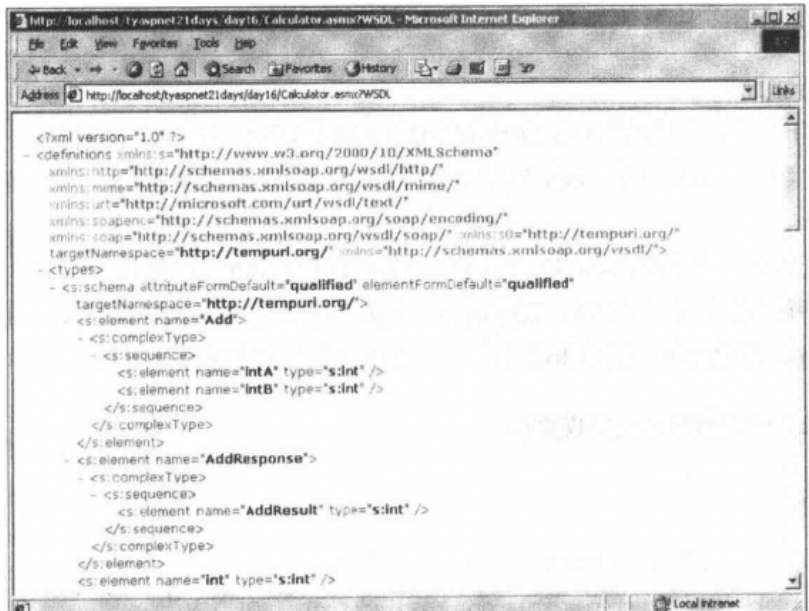


图 16.6 XML 格式的 Web 服务的描述

让我们回到.asmx 文件的常规视图。单击 Add，以查看 ASP.NET 为您准备的其他东西，如图 16.7 所示。

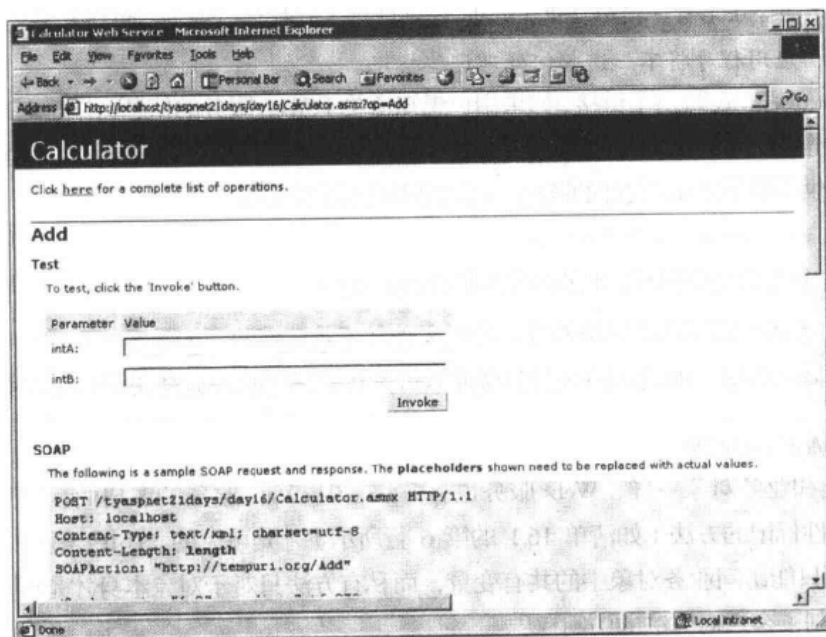


图 16.7 关于 Add 方法的详细描述

显然,其中有许多关于 Web 服务的信息。该页面被称为 HTML 描述页面 (Description Page),详细列出了许多有趣的东西,它甚至让您能够测试该方法。在文本框输入两个值,然后单击 Invoke 按钮,将弹出一个新窗口,其中包括 XML 格式的答案。这正是客户调用该 Web 方法后,将收到的响应。页面中接下来的三个部分 (SOAP、HTTP GET 和 HTTP POST) 列出了这些协议用来访问 Add 方法的方式。

该页面让您能够测试服务。注意,由于您是使用一个表单来提交数据,因此是使用 Http-Post 协议来与服务器通信。

### 16.3.2 启用发现功能

发现是客户应用程序找到 Web 服务并确定其功能的过程。这些信息来自服务描述,但大多数客户不知道 (也不应该知道) 该描述的文件名。因此,启用发现意味着让客户链接到这些描述。

发现是通过 Web 服务的 disco 文件启用的。这些文件是 XML 格式的文档,包含了到服务描述的链接。客户可以通过访问该文件,获得更详细的关于该 Web 服务的可用性方面的信息。

创建 disco 文件很简单,清单 16.2 是一个简单的计算器服务的发现文件。

**清单 16.2 计算器服务的发现文件**

```
1 <?xml version="1.0" ?>
2 <disco:discovery
3   xmlns:disco="http://schemas.xmlsoap.org/disco/"
4   xmlns:scl="http://schemas.xmlsoap.org/disco/scl">
5   <scl:contractRef ref="Calculator.asmx?WSDL"/>
6 </disco:discovery>
```

**分析:** 将该文件保存为 calculator.disco。正如您看到的,这个文件非常简单,因为它只包含一个到服务描述的链接,如第 5 行所示。Xmlns 标记提供的其他名称空间指定了 URL,后者定义了标记 scl 和 disco 可以使用的合法名称。换句话说,xmlns 标记提供了到标记 scl 和 disco 的标准定义的链接。没有这些名称空间,应用程序将不知道如何处理这些标记。

标记 disco:discovery 提供了链接,如果用户想知道关于 Web 服务的信息,沿着该链接即可找到。这些链接可以链接到其他 disco 文档或服务描述。到服务描述的链接是通过标记 scl 提供的,如第 5 行所示。要提供到其他 disco 文件的链接,可以使用下面的语法:

```
<disco:discoveryRef ref="link" />
```

在浏览器中请求该文件时,将得到相同的 XML 文件。

**注意:** 对于客户访问 Web 服务而言,发现文件并不是必需的。如果客户知道正确的 URL,便可以访问服务描述本身。发现文档只是帮助匿名用户找到您想供公众使用的 Web 服务而已。

### 16.3.3 WebMethod 属性

和常规类和业务对象一样,Web 服务也有方法。但 Web 服务的客户而言,只有那些使用 WebMethod 属性标记的方法 (如清单 16.1 的第 6 行所示) 才是可见的。还记得上一章介绍的吗,ASP.NET 页面只能访问业务对象中的共有变量,而私有方法只对于对象本身才是可见的。同样,WebMethod 限制了客户对方法的访问权。

Web 方法就像常规类中的方法一样。它们可以与 ASP.NET 的 Session 或 Cache 对象交互 (请参

见第 4 章和第 14 章), 就像 ASP.NET 页面一样, 它们也提供了缓存响应的功能, 并且可以与数据库和数据源进行交互。基本上, 它们与常规类中的方法相同, 唯一的差别是, 可以通过 Internet 访问它们。

这是一个非常重要的概念。毕竟, 类的方法是最重要的部分之一, 它们描绘了用户将如何和类进行交互。带 WebMethod 属性的方法将所有的调用者看作是本地的。换句话说, 在同一个目录下的 ASP.NET 页面中调用 Web 方法与在地球另一面的服务器上调用 Web 方法没有任何差别。这个属性是让 Web 方法能够被客户使用的关键, 让我们做进一步的讨论。WebMethod 属性有许多有趣的特性, 它由类 WebMethodAttribute 描绘, 这个类就像其他 .NET 类一样运行, 也有属性和方法。因此, 当您像下面那样插入属性时, 实际上是创建了 WebMethodAttribute 类的一个新实例。

```
<WebMethod> Public Function Add( intA As Integer)
```

给这个类的属性赋值的方法与您熟知的稍为有些不同。例如:

```
<WebMethod(Description:="Adds two numbers ")> Public Function _
    Add(intA As Integer,intB As Integer)
</WebMethod(Description:="Adds two numbers ",_EnableSession:=False)>
    Public Function Add(intA As Integer,intB As Integer)
```

您直接在函数声明的括号中指定属性和值。请注意等号前面的冒号, 这意味着您是在初始化一个属性, 而不是提供参数。如果您使用下面的代码:

则 ASP.NET 将把 "Description="Adds two number"" 看作是一个用于构造 WebMethod 的变量名。这使得该函数声明有些怪怪的, 但它让您能够设置 WebMethod 的属性; 要设置多个属性时, 只需使用逗号将其隔开:

```
Public Function<WebMethod(Description:="Adds two numbers",_
    EnableSession:=False)>Add(intA As Integer,intB As Integer)
```

表 16.1 描述了可以设置的属性。

表 16.1 WebMethod 的属性

属 性	描 述
BufferResponse	指定是否缓冲 Web 服务的输出。默认值为 true, 即生成的结果将缓存在服务器上, 然后才发送给客户。数据将成批的发回。  如果需要返回大量的数据, 则将该属性设置为 false 会好些。这样数据将分开返回给客户, 性能将稍微高些。否则, 总是应该将其设置为 true。
CacheDuration	Web 方法是可缓存的, 就像 ASP.NET 页面一样。该属性指定缓存响应的时间 (单位为秒), 默认值为 0, 即禁用缓存功能。  当该属性不被设置为 0 时, 结果将保存在高速缓存中一段时间, 接下来客户对该方法的所有调用都将取得高速缓存中的数据, 并不是再次执行该方法。
Description	该属性向客户提供 Web 方法的描述。该描述在服务描述中可以见到, 默认值为 String.Empty。
EnableSession	指定是否为当前方法启用会话状态。如果启用 (默认值), 则可以使用 Session 对象来存储数据。如果不需要将任何数据存储在会话变量中, 则应该关闭该选项, 这样可以提高性能。

续表

属 性	描 述
MessageName	该参数将是在服务和客户之间发送的数据中引用该 Web 方法时使用的名称。默认为 Web 方法的名称。 该参数常用于方法的名称是唯一的。例如，如果有两个重载的 Add 方法，它们接受不同的参数，则可以使用该属性来区分这两个方法。在服务描述中，该属性必须是唯一性的。
TransactionOption	和数据库一样，Web 服务也可以参与事务。在事务中，要么执行所有的代码，要么都不执行。如果其中某个代码执行失败，则整个事务便失败了，所作的修改必须撤销。有关事务的更详细的信息，请参见第 12 章。 该属性可能的取值有： Disable——方法运行时没有事务； NotSupported——不支持事务； Supported——支持事务，但方法不在事务中运行； Required——需要一个事务，将创建一个新事务； RequiresNew——需要一个事务，将创建一个新事务。
TypeID	用于标识该属性的唯一性 ID，用于区分两个属同一类型的属性。

#### 16.3.4 部署 Web 服务

部署 Web 服务很简单。由于所有的 ASP.NET 应用程序的管理工作都是由 CLR 处理的，因此只需将相应的 .asmx 文件、.disco 文件和自定义业务对象复制到合适的目录即可。部署应用程序从来没有这样简单过！

普遍的做法是，为包含服务的目录创建一个 web.config 文件。您通常总是要为这些服务实现安全机制，以防匿名用户进入并使用这些服务。例如，如果您公司开发 Web 服务，则公司肯定希望将其用来创造利润。如果任何人都可以进入并使用该服务，公司将无钱可赚。确保 Web 服务的安全（将在下一章介绍）可以避免出现这样的情况，并让您能够控制哪些人可以访问您的服务。

**警告：**用于部署服务的目录必须标记为一个 Internet 信息服务（IIS）应用程序目录，否则服务将无法执行，客户也无法使用。

您创建的、使得 ASP.NET 页面能够运行的目录都是 IIS 应用程序目录。

### 16.4 使用已有的业务对象创建 Web 服务

也可以使用已有的业务对象来创建 Web 服务，以避免重写所有的功能。但为启用服务支持，必须对已有的对象稍作修改。

让我们来修改前一章创建的 database 对象。为将其变成一个 Web 服务，需要做三项工作：从 System.Web.Services.WebService 类继承；给要暴露的每个方法加上 WebMethod 属性；将 OleDbDataReader 改为 DataSet，因为前者不能通过 XML 进行传输（更详细的信息，请参见后面）。清单 16.3 列出了修改后的类，它被重新命名为 DatabaseService。

**提示：**可以保留 database 类，并在同一个文件中创建这个类的副本，但使用不同的名称（例如，在后面加上“Service”）。只有从 WebService 继承而来的类才会作为服务暴露给客户。这使得您可以在同一个文件中创建对象的多个版本。

### 清单 16.3 修改业务对象 database，将其用作一个服务

```

1 Imports System
2 Imports System.Data
3 Imports System.Data.OleDb
4 Imports System.Web.Services
5
6 Namespace TV\SPNET
7     Public Class DatabaseService : Inherits WebService
8         private objConn as OleDbConnection
9         private objCmd as OleDbCommand
10
11     Public function <WebMethod()> SelectSQL(strSelect as _
12         string) as DataSet
13         try
14             objConn = new OleDbConnection("Provider=" & _
15                 "Microsoft.Jet.OLEDB.4.0;" & _
16                 "Data Source=C:\ASPNET\data\banking.mdb")
17             dim objDataCmd as OleDbDataAdapter = new _
18                 OleDbDataAdapter(strSelect, objConn)
19
20             Dim objDS as new DataSet
21             objDataCmd.Fill(objDS, "tblUsers")
22             return objDS
23         catch ex as OleDbException
24             return nothing
25         end try
26     end function
27
28     Public function <WebMethod()> ExecuteNonQuery(strQuery _
29         as string) as Boolean
30         try
31             objConn = new OleDbConnection("Provider=" & _
32                 "Microsoft.Jet.OLEDB.4.0;" & _
33                 "Data Source=C:\ASPNET\data\banking.mdb")
34             objCmd = new OleDbCommand(strQuery, objConn)
35             objCmd.Connection.Open()

```



```

36         objCmd.ExecuteNonQuery
37         objCmd.Connection.Close()
38         return true
39     catch ex as OleDbException
40         return false
41     end try
42 end function
43
44 End Class
45
46 End Namespace

```

第4行导入了另一个名称空间 `System.Web.Services`，在第8行的类声明中，继承了该名称空间中的 `WebService` 类。接下来，给要暴露为服务的函数加上了属性 `<WebMethod(>`。最后，在 `SelectSQL` 中，将 `OleDbDataReader` 改为 `DataSet`，同时将 `OleDbCommand` 对象改为 `OleDbDataAdapter`。这就是需要做的所有修改。使用下面的命令重新编译这个文件：

```

vbc /t:library /out:..\bin\TYASPNET.dll /r:System.dll
/r:System.Data.dll /r:System.Web.Services.dll
/r:System.Xml.dll DatabaseService.vb ..\day15\User.vb

```

该命令将新文件 `Database.vb` 和上一章的 `User.vb` 编译到库文件 `TYASPNET.dll` 中，另外还引用了两个新的 DLL：`System.web.Services.dll` 和 `System.xml.dll`。引用前一个文件的原因很明显；而引用第二个文件是由于它包含通过 XML 发送 `DataSet` 所需的方法。有关 VB.NET 编译器的更详细的信息，请参见前一章。

**注意：**在这个类（.vb 文件）中，您将属性 `<WebMethod(>` 放在函数名的前面，而在 .asmx 文件中，则放在函数声明的其他部分的前面。例如，在 .vb 文件中，使用下面的格式：

```

Public function <WebMethod(>ExecuteNonQuery(strQuery_
as string)as Boolean

```

而在 .asmx 文件中，则使用下面的格式：

```

<WebMethod(> Public function ExecuteNonQuery(strQuery _
as string)as Boolean

```

这是在 .NET 框架 beta2 版本中不一致的地方。只要知道这一点就可以了。

现在，新的对象将被保存在组合体仓库中，并从中被装载。让我们来创建 .asmx 文件，以便可以能够访问该对象，如清单 16.4 所示。

#### 清单 16.4 暴露 database 服务的 .asmx 文件

```

1 <%@ WebService Class="TYASPNET.DatabaseService" %>

```

至此，工作便完成了。您只需修改类名，以引用刚才编译的 `Database` 服务。将该文件保存为 `Database.asmx`，并从浏览器中请求它，您将看到两个被暴露的方法：`SelectSQL` 和 `ExecuteNonQuery`。

**警告：**如果将预先编译好的对象作为 Web 服务，则它必须位于不见仓库中，这样 ASP.NET 才能找到并使用它。

单击 `SelectSQL`，并在页面的测试部分输入一个 SQL 查询：`Select * FROM tblUsers`，然后单击

“Invoke”按钮，将得到与图 16.8 类似的结果。

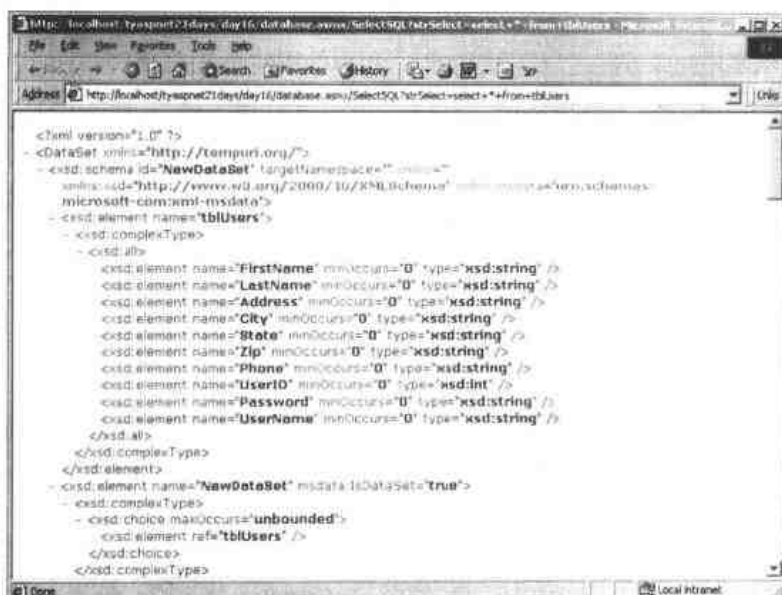


图 16.8 从 database 服务返回的数据

该 XML 文档定义了服务返回的 DataSet 的结构。向下滚动窗口，将看到来自数据表中的实际数据。如果需要，收到该 XML 文档的客户可以轻松地将该对象转换为 DataSet。

## 16.5 从服务返回数据

正如前面说明的，Web 服务可以返回任何类型的数据，这是因为 Web 服务是构建在 XML 之上的，而 XML 表示数据的功能非常强大。虽然 XML 是基于文本的，但使用一个类型命名系统，让其他应用程序能够很容易地确定数据类型。

例如，您知道，DataSet 基本上被表示为计算机内存中的 XML 数据，因此可以推断出，DataSet 可以通过 XML 发送。XML 还能够表示字符串、整数、数组和 DateTime 以及其他一些数据类型。表 16.2 列出了 Web 服务能够传输的各种数据类型。

表 16.2 支持的 Web 服务数据类型

类 型	描 述
数组	Web 服务可以发送后面描述的任何数据类型的数组，包括 DataSet、原语类型、类和 XmlNode。
类	带公有属性的类
DataSet	DataSet 在内部被 ADO.NET 表示为 XML，因此可能被 Web 服务传输。 DataSet 只不过是可传输的 ADO.NET 数据存储，而 DataReader 等则不是。
原语	原语类型包括 byte、Boolean、char、DateTime、Decimal、Double、GUID、int32、int16、int64、single、unit16、unit32、unit64 和 xmlQualifiedName 等。
xmlNodes	.NET XmlNode 类，XML 数据在内存中的一种表示方式，请参见第 11 章。

Web 服务可以返回表 16.2 中描述的所有数据类型,但对于其能够接受的参数数据类型,则要求更严格。

使用 SOAP 从客户向服务传递命令和参数时,可以使用前面描述的任何数据类型,因为 SOAP 是基于 XML 的。

但如果使用 Http-Get 或 Http-Post,则只能使用这些协议能够处理的数据类型。也就是说,只能使用原语类型的子集及其数组。另外,Http-Get 和 Http-Post 也只能发送名称/值对,因此不能发送较复杂的数据类型,如类或 DataSet。

## 16.6 这不是 ASP

Web 服务是 ASP.NET 中全新的特性,因此,如果您是传统的 ASP 开发人员,将发现本章介绍内容时,将您当成了一个新手。但 Web 服务基于的思想,如各种不同系统之间的相互通信,与 COM (组件对象模型)对象类似(关于 COM 对象的更详细的信息,请参见上一章)。

COM 旨在让开发人员能够使用各种不同的组件来创建应用程序。由于这些组件依赖于 COM 协议进行通信,因此可以使用不同的编程语言来开发。虽然如此,但 COM 的缺点在于被限制为特定的平台——Windows。

Web 服务之所以优于 COM,是由于它不使用任何专用的协议或对象间的二进制格式通信。相反,Web 服务是完全基于开放标准 XML 和 HTTP 的,这意味着真正能够在任何不同的系统之间进行通信。

通过 Internet 将应用程序作为服务进行部署的概念也不是全新的。以前的尝试像 COM 一样,依赖于复杂的专用标准,这限制了其用户。是 XML 和 HTTP 的简单性使得 Web 服务如此有效。

## 16.7 总 结

本章介绍了一种新的递送应用程序的方式:将其作为 Web 服务通过 Internet 进行递送。Web 服务是 ASP.NET 的有机组成部分,它使用 XML 和 HTTP 标准,使得组件能够通过 Web 相互进行通信。

Web 服务的工作时分三步进行:发现、服务描述和命令通信。发现是一个可选步骤,它让客户获悉关于 Web 的信息,具体的说,是 Web 位于什么位置。服务描述是使用一种叫做服务描述语言的、基于 XML 的标准提供的,它告诉客户,可以远程使用哪些方法以及服务期望和返回的数据类型。最后,客户使用这些描述将命令发送给服务器,而服务将数据发回给客户。

有三种协议被用于 Web 服务通信:Http-Get、Http-Post 和简单对象访问协议(SOAP)。前两种协议允许传递简单的名称/值对,而后一种协议是基于 XML 的,它能够提供更范围更广的数据类型。

您在.aspx 文件中创建 Web 服务,就像业务对象一样。区别在于 Web 服务是从 WebService 类派生而来的,同时要暴露为服务的所有方法都必须使用 WebMethod 属性。.aspx 文件必须包含以下代码:

```
<%@ WebService Class="class" %>
```

编译指令 WebService 告诉 ASP.NET,该文件包含或引用了一个应该被暴露为 Web 服务的类。Class 属性指定要暴露的类。这个类可以包含在.aspx 文件中,也可以单独编译为一个业务对象。

您可以使用.disco 文件来启用发现功能,这种文件是 XML 文档,其中包括到 Web 服务描述的

链接。这些文件可以手工创建，以定义某个特定的服务，或动态定义服务器上可用的一组服务。

最后，您可以通过浏览器来查看 .asmx 文件，以便看到 HTML 格式的描述页面，该页面描述了可用的方法，并让您能够直接通过 Http-Post 来测试功能。从 .asmx 页面，您还可以查看 XML 格式的服务描述。

阅读本章后，您应该熟悉 Web 服务的开发方法，并能够在服务上部署它们。下一章将继续讨论 Web 服务，介绍如何在客户端使用它们。

## 16.8 问与答

问：Web 服务将如何影响普通的家庭用户？

答：很可能是这样的，家庭用户不会马上受其影响。Web 服务需要到 Web 的高带宽连接，如 DSL 或线缆调制解调器。就目前而言，以这种方式连接的 Web 的人数还不足以使 Web 服务得到全面的应用。

最有可能使用 Web 服务的是维护强大的大型网络的公司。信息技术人员将发现，管理用户工作站将更容易、费用也将更低，因为不再需要在每台计算机上安装和维护应用程序。

问：有办法使用 Http-Get 的方式测试 Web 服务吗？SOAP 呢？

答：完全可以。只需在 .asmx 文件的地址后面加上要测试的函数名以及所需的参数。例如，您可以使用下面的 URL 来测试计算器服务的 Add 方法：

```
http://localhost/tyaspnet21days/day16/  
Calculator.asmx/Add?intA=1&intB=2
```

使用斜杠将函数名和 URL 分开，然后向查询字符串那样加上参数。这将通过 Http-Get 协议测试服务，而返回的 XML 数据将显示在相同的窗口中。

将在下一章介绍在客户端使用 Web 服务时，如何使用 SOAP 来测试服务。

问：.disco 文件的名称必须与 Web 服务文件名相同吗？

答：不。但习惯这样做。

## 16.9 作业

下面的作业帮助巩固本章介绍的概念。答案见附录 A。

### 16.9.1 小测验

1. Web 服务是什么？
2. SOAP 代表什么？有何功能？
3. 服务描述是什么？
4. Web 服务必须从那种类（包括名称空间）派生而来？
5. 随意编写一个 Web 方法，它启用缓存功能，并禁用会话状态。
6. 判断正误：Web 服务可以使用 ASP.NET 内置的对象，如 Session 和 Cache。
7. 判断正误：Web 方法可以返回 DataSet。
8. 要启用发现功能，.disco 文件必须包含何种代码？

### 16.9.2 练习

1. 查看本章创建的 `database` 服务的 XML 服务描述。尝试理解其中的每一个标记，以温习关于 XML 和 XML 模式的讨论。文件中的大部分内容描述的是各种不同的协议（`Http-Get`、`SOAP` 等）如何发送和接收数据。

2. 创建一个 Web 服务，将一种度量单位换算为另一种。该方法接受三个参数：一个值、原来的单位、目标单位。让用户能够换算以下单位：毫米、厘米、英寸、英尺、米、码、英里和千米。别忘了在 HTML 描述页面中测试该服务。

如果您不知道如何换算这些单位，请不用担心，只需虚构其转换方式即可。

## 第 17 章

# 使用 Web 服务并确保其安全

前一章全面介绍了如何创建 Web 服务：Web 服务是什么？为何使用它？如何创建它。本章将继续介绍 Web 服务，讲解如何通过 ASP.NET 页面使用 Web 服务。

本章将介绍以下内容：

- 如何使用发现功能获取关于某个 Web 服务的信息。
- 如何生成访问服务的代理类。
- 如何在 ASP.NET 页面中实现代理类。
- 如何使用 SOAP 报头确保服务的安全。

### 17.1 使用 Web 服务

前一章介绍了如何使用 Web 服务来创建能够通过 Internet 访问的服务。例如，某个站点可能想提供一组财务计算的函数。该 Web 服务设计好并被部署到该站点的 Web 服务器上后，任何人都可以使用其功能。

**新术语：**假设某开展银行业务的 Web 站点想使用其中的某个财务计算器服务，来计算存单的利息。则其银行业务应用程序必须消费（consume）您的服务，即访问并使用该方法。消费（Consumption）指的是客户使用 Web 服务中可用的方法。该站点的访问者可以使用该计算器，而无需知道它来自何方。

您去加油站时，服务员将为您提供服务，因此您是这种服务的消费者。您可以利用加油站提供的任何资源，如气泵、服务员的时间和服务以及它们销售的日用品（当然，您必须为这些服务付费，这与上述话题并不相干）。这样您可以省去自己建立气泵或自己动手修理汽车的麻烦。

Web 服务的消费者与此完全相同，它访问服务，并消费任何可用的资源。例如，您的计算机可能访问一个字处理服务。您可以使用该服务的所有特性，因此不用自己购买并安装字处理程序。

Web 服务的消费者几乎可以是任何应用程序：台式机、ASP.NET 页面，乃至移动设备（如手机）。本章重点介绍如何在 ASP.NET 页面中使用 Web 服务。

使用 Web 服务包括三个步骤：

1. 收集关于服务的信息，这一过程叫做发现。
2. 生成服务的代理。
3. 使用代理调用可用的服务。

**警告：**不要将Web服务客户和Internet客户混为一谈。前者是一个使用Web服务的应用程序或Web站点，而后者是用于浏览Web的应用程序，如浏览器。Web服务客户访问服务，而Web客户访问Web站点。它们是不同的，创建Web服务客户之前，必须弄清楚它们之间的差别。

前一章介绍过，发现是客户获取关于 Web 服务的信息的过程。当您去一个新餐厅时，您首先会看菜单，以便搞清楚餐厅提供的服务。Web 服务的消费者与此相同，它们在使用服务之前必须知道服务的功能。

这些信息是由 Web 服务的服务描述提供的（参见前一章）。服务描述是一个由服务生成的 XML 文件。客户访问该文件以获取关于服务的信息，换句话说，它查看关于特性的菜单。然后创建自己的菜单副本，供以后使用。

还记得上一章介绍的 disco 文件吗，这些文件的唯一目的是协助客户。它们提供到服务器上可用的服务描述的链接，客户使用它来找到道路。如果客户已经知道服务描述的 URL，则无需使用 disco 文件。

生成服务代理是一个很有趣的步骤。下面是一个比喻：您想去某个餐厅，但由于没有驾驶证，因此您无法驱车前往；不过您说服母亲，由她开车带您去。通过将您带到您去不了的地方，她代您完成了工作，换句话说，她是您和餐厅之间的代理（proxy）。

Web 服务的消费者与此极其类似。它们知道某个地方有服务可用，但自己到达不了这个地方。它们需要代理的帮助，代理提供了在服务 and 客户之间传输信息的机制。

在 ASP.NET 中访问服务时，您希望服务易于使用，而不用担心如何发送 XML 消息、将命令转换为合适的格式以及检索返回的数据等工作。最理想的情况是，您希望访问服务时，它们就像是自己计算机上的业务对象，即您只需创建一个对象实例。

因此，您生成一个服务代理。服务代理是一个位于您的计算机上的类，它封装了与服务通信所需的所有复杂的功能，让您能够像与其他任何对象交互一样与服务进行交互。事实上，代理包含与服务提供的方法和属性对应的方法和属性，而且名称相同。因此，如果您知道服务提供的某个方法，则可以使用相同的名称来调用代理。图 17.1 说明了使用代理的过程。

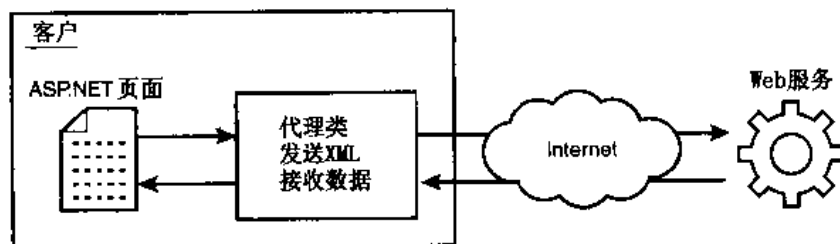


图 17.1 代理扮演了客户和服务的中间人的角色

代理是根据服务描述中的信息生成的，服务描述将关于如何发送命令以及将返回什么的所有信息告诉了代理。这将在本章后面的“通过 ASP.NET 页面使用 Web 代理”一节中更深入地介绍。

最后，消费者可以通过代理类使用 Web 服务的方法，就像使用其他任何对象一样。所有这一切对终端用户都是透明的，对于开发人员来说，这是非常难以实现的。图 17.2 说明了消费者一端涉及的步骤。

**注意：**Web 服务最普遍的消费者是 Web 页面和其他 Web 服务，虽然常规的、基于台式机的应用程序也经常访问 Web 服务。

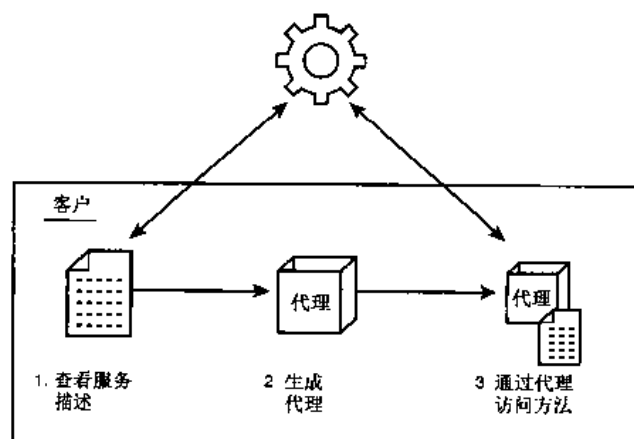


图 17.2 使用 Web 服务包括三个步骤

## 17.2 通过 ASP.NET 页面使用 Web 服务

您曾经是 Web 服务的消费者。上一章，您通过查看服务描述和 HTML 描述页面，您执行了客户将执行的方法，只不过您直接与服务打交道，而不是通过代理而已。

接下来的几节将介绍如何使用 SOAP 作为通信机制，通过 ASP.NET 页面完成发现、代理和实现等三个步骤。前两个步骤需要使用命令行工具，但所幸的是，您需要做的工作不多。第三步（实现）与在 ASP.NET 页面中使用业务对象完全相同，这都是代理的功劳。

### 17.2.1 发现

可以使用 Web 服务发现工具 disco.exe 来完成对 Web 站点的发现工作。这个工具检查 Web 站点，并返回服务器上的 disco 文件的副本。然后您就可以使用这个本地副本帮助生成代理。

我们来看一个例子。打开一个命令提示符窗口，并切换到目录 c:\inetpub\wwwroot\tyaspnet21days\day17。假设您已经创建了一个名为 Calculator.disco 的发现文件（见前一章），并将其保存在应用程序 tyaspnet21days 的根目录下，则请在命令窗口中输入以下命令：

```
disco http://localhost/tyaspnet21days/day16/calculator.disco
```

（本章后面将详细介绍这个工具的语法）disco.exe 将打印指定的服务器上可用的 disco 文件的 URL，如图 17.3 所示。

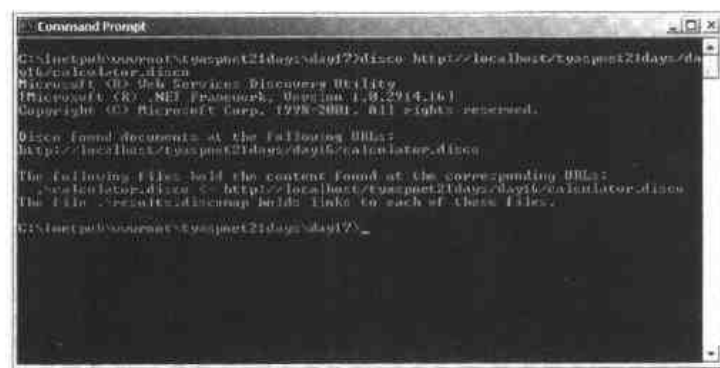


图 17.3 执行工具 disco.exe 得到的结果



默认情况下,该工具也会输出两个名叫 `results.discomap` 和 `services.disco` 的文件。它告诉您在服务器上发现了哪些 `.disco` 文件以及它将这些文件的副本保存在哪里。`Services.disco` 的内容和服务器上的 `.disco` 文件相同,它们都是 XML 文件,因此易于阅读。图 17.4 显示了一个典型的 `services.disco` 文件。

清单 17.1 列出了 `results.discomap` 的内容,该文件详细列出了发现过程的结果。

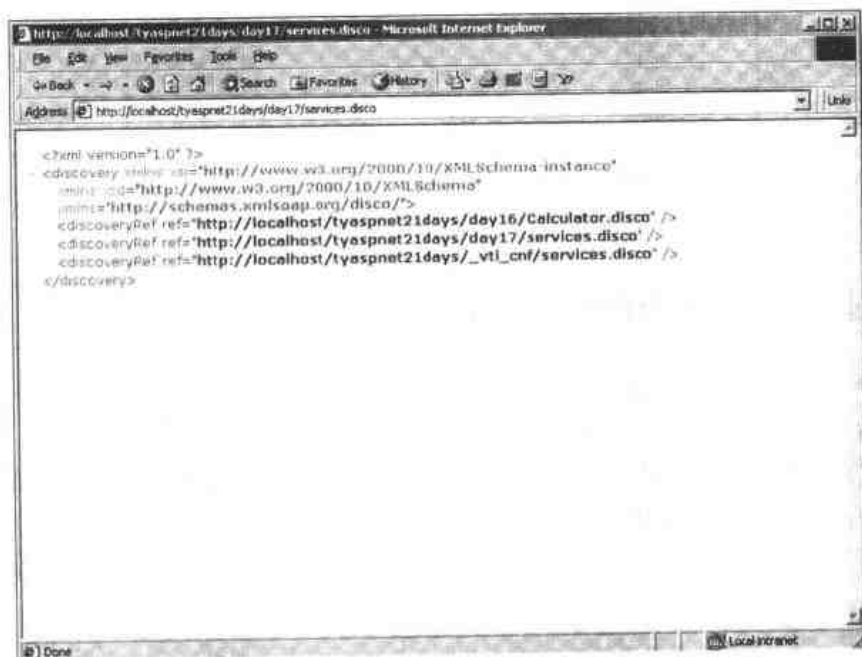


图 17.4 在浏览器中显示的 XML 文件 `services.disco`

#### 清单 17.1 `results.discomap` 的内容

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <DiscoveryClientResultsFile xmlns:xsi="http://www.w3.org/
3   2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/
4   2001/XMLSchema">
5   <Results>
6     <DiscoveryClientResult referenceType="System.Web.
7       Services.Discovery.DiscoveryDocumentReference"
8       url="http://localhost/tyaspnet21days/day16/
9       calculator.disco" filename="calculator.disco" />
10   </Results>
11 </DiscoveryClientResultsFile>

```

**分析:** 该文件包含在服务器上执行发现后获得的结果。您感兴趣的是标记 `Results` 中的内容。第 4 行的 `referenceType` 属性指出了您刚才查看的是哪类文件,即发现文档。`url` 属性指出了 `.disco` 文件的位置,即可以使用什么 URL 访问它。`Filename` 属性指出了 `.disco` 文件的副本被存储在您的计算机的什么位置。在创建代理时,可以使用文件 `results.discomap`。

工具 `disco.exe` 有 6 个命令行选项,如表 17.1 所示。

表 17.1 工具 disco.exe 的选项

选 项	描 述
/nologo	使用该工具时，不显示微软专用的消息。
/nosave	不将发现结果保存到文件中。
/out	指定保存结果的目录，默认为当前目录。
/username	指定访问服务器时需要使用的用户名。
/password	访问服务器使用的密码。
/domain	访问服务器需要的域。

例如，下面的命令不消失微软的徽标，并将结果保存到文件夹 c:\temp\disco 中：

```
disco /nologo /out:c:\temp\disco http://localhost/
► tyaspnet21days/services.disco
```

### 17.2.2 创建代理类

前面指出过，代理类充当了 Web 服务和消费者的中间人。它包含通过 Internet 传输数据所需的所有功能，因此您无需自己创建这些功能。稍后您将知道，代理与 Web 服务所在服务器上的.asmx 文件极其类似，但在调用方法方面有些不同。

使用另一个工具——Web 服务描述语言工具 wsdl.exe——来创建代理类。该工具查看服务器上的.discomap 文件或 XML 服务描述，并创建一个类，其中包括方法和属性与 Web 服务相同。这使得代理尽可能地不可见。可以像与服务直接交互一样，创建 ASP.NET 消费者。另外，该工具创建方法和添加属性，让代理能够通过 Internet 发送信息。

当然，您也可以自己创建代理，但既然 wsdl.exe 具备这样的功能，为何还要自己动手呢？（如果您决定自己做，则当您在本章后面查看一个实际的代理后，将知道所有需要完成的工作。）

**注意：**wsdl.exe 工具能够检查 Web 服务的描述，这意味着如果知道服务描述的 URL，则无需首先使用工具 disco.exe。

我们来看一个例子。从命令行执行下面的命令：

```
wsdl /language:VB http://localhost/tyaspnet21days/day16/
► Calculator.asmx?WSDL
```

本章后面将介绍该命令的语法。现在，您只需知道第一个参数/language 属性指定了生成的代理类将使用哪种编程语言（这里为 VB.NET）；第二个参数指定了服务描述的 URL（您在上一章以发现了该 URL）。

使用前面指定的参数运行 wsdl.exe 后，将看到与下面类似的配置消息：

```
H:\Inetpub\wwwroot\tyaspnet21days\day17\Calculator.vb
Writing file
'H:\Inetpub\wwwroot\tyaspnet21days\day17\Calculator.vb' .
```

注意到工具 wsdl.exe 已经创建了一个 VB.NET 类文件 calculator.vb，这便是您的代理类。请花一些时间来查看该文件的内容（该文件位于执行命令时的目录中）。清单 17.2 列出了该文件。

## 清单 17.2 生成的 Calculator.vb 文件

```

1 --- -----
2 <autogenerated>
3 ' This code was generated by a tool.
4 ' Runtime Version: 1.0.2914.16
5 '
6 ' Changes to this file may cause incorrect behavior and
7 ' will be lost if the code is regenerated.
8 </autogenerated>
9 -----
10
11 Option Strict Off
12 Option Explicit On
13
14 Imports System
15 Imports System.Diagnostics
16 Imports System.Web.Services
17 Imports System.Web.Services.Protocols
18 Imports System.Xml.Serialization
19
20 '
21 ' This source code was auto-generated by wsdl, version-1.0.2914.16.
22
23
24
25 <System.Web.Services.WebServiceBindingAttribute(Name:="CalculatorSoap",
26 [Namespace]:="http://tempuri.org/")> _
27 Public Class Calculator
28 Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
29 <System.Diagnostics.DebuggerStepThroughAttribute()> _
30 Public Sub New()
31 MyBase.New
32 Me.Url = http://localhost/tyaspnet21days/day16/Calculator.asmx"
33 End Sub
34 <System.Diagnostics.DebuggerStepThroughAttribute(), _
35 System.Web.Services.Protocols.
36 SoapDocumentMethodAttribute("http://tempuri.org/Add",
37 Use:=System.Web.Services.Description.SoapBindingUse.
38 Literal, parameterStyle:=System.Web.Services.Protocols.

```

```

38 SoapParameterStyle.Wrapped)>_
39     Public Function Add(ByVal intA As Integer, ByVal intB As Integer) As Integer
40         Dim results() As Object = Me.Invoke("Add", New Object() {intA, intB})
41         Return CType(results(0), Integer)
42     End Function
43
44     <System.Diagnostics.DebuggerStepThroughAttribute()>_
45     Public Function BeginAdd(ByVal intA As Integer, ByVal intB As Integer, ByVal callback
        As System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
46         Return Me.BeginInvoke("Add", New Object() {intA, intB}, callback, asyncState)
47     End Function
48
49     <System.Diagnostics.DebuggerStepThroughAttribute()>_
50     Public Function EndAdd(ByVal asyncResult As System.IAsyncResult) As Integer
51         Dim results() As Object = Me.EndInvoke(asyncResult)
52         Return CType(results(0), Integer)
53     End Function
54 End Class

```

分析：这个类是wsdl.exe为方便您使用Web服务而创建的。注意到除服务提供的一个方法外，该文件中还包括另外两个方法，同时还包括大量的属性和方法以及其他一些东西，这使得这些代码比较难理解。如果不能完全理解其中的代码，也不用担心，您无需深入研究其语法。

第 25 行声明了一个名叫 calculator 的类，它包含一个来自名称空间 System.Web.Services 的属性 WebServiceBindingAttribute。该属性定义了这个类必须使用的接口，即一组预先定义好的函数和属性。就本章的内容而言，您无需了解很多关于该属性的信息（更详细的信息，请参见.NET 框架 SDK 文档）。

另外，这个类是从 SoapHttpClientProtocol 继承而来的，如第 26 行所示，它提供了通过简单对象访问协议（SOAP）与 Calculator 服务远程通信的方法。

在 29 行，代理包含一个构造函数——一个用于初始化类的方法（参见前一章的“问与答”）。该构造函数设置 Web 服务的 URL。

从第 34 行起，是一些与上一章创建的 Add 方法类似的代码。该函数包含属性 SoapMethodAttribute，该属性提供了服务通过 SOAP 与消费者通信时使用的参数。在 37 行，该函数调用了 Invoke 方法，后者调用服务。Invoke 方法接受两个参数：要调用的服务中的方法的名称以及要发送给服务的参数。结果被存储在一个数组中，该数组在 49 行被返回。

接下来的两个函数定义了与服务通信时使用的异步方法，关于调用异步方法的更详细的信息，请参见第 13 章。实际上，这些方法让您能够在它们执行期间执行其他工作。常规方法（也叫同步方法）在执行时，不允许您执行其他操作。

该代理类几乎封装了使用各种不同的传输协议（如 SOAP 和 Http-Get），通过 Internet 调用服务所需的所有功能。可以像使用业务对象那样，使用该代理与服务进行交互。谢天谢地，您无需自己创建该代理。

Wsdl.exe 还有其他一些选项，可以通过指定它们来生成不同类型的代理。它支持表 17.1 中的所

有选项（/nosave 除外），另外还支持表 17.2 中的选项。

表 17.2 wsdl.exe 选项

选 项	描 述
/language	生成的代理类使用的语言，默认为 C#。
/namespace	生成的代理类所属的名称空间，默认为全局名称空间。
/protocol	用于与服务通信的协议，可以为 SOAP、Http-Get、Http-Post，默认为 SOAP。

**提示：**wsdl.exe 和 disco.exe 的许多选项都有简写，例如您可以输入 /l，而不用输入 /language。请在命令行输入 wsdl.exe /? 或 disco.exe /? 来查看更详细的信息。

下面的命令创建一个使用 VB.NET 编写的代理类，该代理类使用 SOAP 协议，并属于一个自定义的名称空间。

```
wsdl /l:VB /namespace:MyWebServiceConsumer /protocol:SOAP
    http://localhost/tyaspnet21days/day16/Calculator.asmx?WSDL
```

### 17.2.3 实现代理类

您差不多为在 ASP.NET 页面中使用 Web 服务做好了准备，现在代理类可以与服务交互，但您还不能使用它。就像业务对象一样，您首先必须将代理编译成一个组件（一个 DLL 文件）。这是使用 VB.NET 编译器（现在您对它应该非常熟悉）来完成的。请在命令提示窗口执行下面的命令：

```
vbc /t:library /out:..\bin\CalculatorServiceClient.dll
    /r:System.dll /r:System.XML.dll
    /r:System.Web.Services.dll Calculator.vb
```

**提示：**当需要在命令行执行多个命令时（如运行 disco.exe，然后运行 wsdl.exe，最后运行 vbc.exe），使用批处理文件（.bat）通常会很有帮助。这可以避免重复输入相同的命令。

创建一个文本文件，将其扩展名改为 .bat，然后将需要的命令复制到该文件中，（命令间用换行符隔开）。只需执行该文件，则所有的步骤将自动完成。

**警告：**当您编译该文件时，将出现很多错误，这是由于属性的位置引起的（有关原因的讨论，请参见上一章）。如果情况确实如此，则只需剪切属性，并将其粘贴到函数名的前面。另外，还可能需删除 Add 方法中的属性 Use:=System.Web.Services.Description.SoapBindingUse.Literal 和 ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped。

上述命令将把代理类编译成一个 DLL 文件，并将其放在您的组合体仓库 bin 目录中。然后，您便可以在 ASP.NET 页面中使用该组件来调用 Web 服务了。清单 17.3 列出了一个使用 Web 服务 Calculator 的简单 ASP.NET 页面。

清单 17.3 在 ASP.NET 页面中调用 Web 服务

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4     sub Page_Load(obj as object, e as EventArgs)
```

```
5    dim objCalc as new Calculator
6
7    lblMessage.Text = objCalc.Add(1,5)
8  end sub
9 </script>
10
11 <html><body>
12   The answer to 1 + 5 is:
13   <asp:Label id="lblMessage" runat="server"/>
14 </body></html>
```

分析：这个 ASP.NET 页面非常简单。在第 5 行的 Page\_Load 事件处理程序中，您创建了刚才编译的 Calculator 代理的实例。该对象代表 Web 服务 Calculator，您可以像直接与服务交互那样使用它。换句话说，如果您将这个编译后的代理交给另一个开发人员，它将对该代理是在与 Web 服务交互一无所知，他可以使用该代理，就像代理本身包含服务的方法一样。

第 7 行调用了 Add 方法，并将结果存储在第 13 行定义的标签中。该方法接受两个整数，将它们相加，并将结果返回。这里实际发生的情况是这样的，您调用代理的 Add 方法，后者接受两个整数，并通过 Internet 将它们传递给服务的 Add 方法，尤其执行实际的计算。然后，代理收到 XML 格式的结果，将其转换为合适的格式（这里为整数），并将结果返回给您。从开发人员的角度看，代理只是接受参数，执行计算，并返回结果。

在浏览器中查看时，该页面的结果如图 17.5 所示。



图 17.5 调用 Web 服务得到的结果

这个例子说明了使用代理类的好处——您无需完成任何特别的工作，便可以与 Web 服务 Calculator 进行通信。您需要做的只是调用 Web 服务中您对其功能感兴趣的方法。这看起来好像是

做了大量的工作，而得到的输出却如此简单，但您真正需要做的是执行三个步骤：使用工具 `wsdl.exe` 生成代理，编译它并将其放在组合体仓库中，以及创建 ASP.NET 页面以访问其方法。（因此，如果有人替您完成前两个步骤，则您只需提供实现）。您无需知道发现和代理生成两个步骤得到的结果——除非您要手工创建代理，否则这些结果跟您毫无关系。因此，代理隐藏了所有的复杂功能，您在 ASP.NET 页面中使用几行代码，便可以完成大量作用很大的工作。

注意：您无需导入其他任何名称空间，便可以使用该代理类。第15章讲过，ASP.NET将自动装载组合体仓库（目录\bin）中的对象，您可以访问这些对象，就像它们是页面中的一部分一样。

#### 17.2.4 另一个使用 Web 服务的例子

至此，您已经取得了很大的成就，通过 Internet 调用对象可是一项大本领。Web 服务为分布式计算提供了无限的可能性，但前面的例子只是一个非常简单的练习。它完成的计算功能在 ASP.NET 页面中便可以实现，而且不需要使用任何复杂的数据类型。

让我们来完成另一个例子。还记得吗，您也可以通过 Web 服务发送数据库结果，这都要归功于 XML。上一章创建了一个 Database 服务，它可以接受 SQL 语句，并检索相应的数据。让我们在 ASP.NET 页面中使用该服务。

我们略过发现的一步，直接生成代理。请执行下面的命令：

```
wsdl /language:VB /namespace:TYASPNET
```

```
http://localhost/tyaspnet21days/day16/Database.asmx?WSDL
```

前面您已经使用过工具 `wsdl.exe`，因此应该熟悉上述命令。该命令创建 Database 服务的一个代理类，并将其放在当前目录中。由于 database 服务包含大量的功能，因此生成的代理非常大，也非常复杂。所幸的是，您根本无需为此操心，只需编译并实现它即可。注意到上述命令将生成的代理放置在名称空间 TYASPNET 中，这是一种对对象进行逻辑分组的方式。生成的代理类似于图 17.6。



图 17.6 从 database 服务生成的代理

接下来，您需要在客户端编译该代理，请执行下面的命令：

```
C:\Inetpub\wwwroot\tyaspnet21days\day17>vbc /t:library
```

```
➤ /out:..\bin\DatabaseService.dll /r:System.dll
```

```
➤ /r:System.Xml.dll /r:System.Web.Services.dll
```

```
➤ /r:System.Data.dll DatabaseService.vb
```

别忘了引用名称空间 `System.Data`！因为代理使用了该名称空间中的对象，将返回的 XML 数据转换为可以在 ASP.NET 页面中使用的 `DataSet`。最后，让我们创建将使用该 Web 服务的 ASP.NET 页面，如清单 17.4 所示。

#### 清单 17.4 访问 database 服务

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3
4 <script runat="server">
5     sub Submit(obj as object, e as eventargs)
6         dim objService as new TYASPNET.DatabaseService
7         dim objDS as new DataSet
8
9         objDS = objService.SelectSQL(tbQuery.Text)
10        DataGrid1.DataSource = objDS
11        DataGrid1.DataMember = "tblUsers"
12
13        DataGrid1.DataBind()
14    end sub
15 </script>
16
17 <html><body>
18     <form runat="server">
19         Enter a query:
20         <asp:Textbox id="tbQuery" runat="server"/>
21         <asp:Button id="btSubmit" runat="server"
22             text="Submit"
23             OnClick="Submit" />
24     <p>
25     <asp:DataGrid id="DataGrid1"
26         runat="server" BorderColor="black"
27         GridLines="Vertical" cellpadding="4"
28         cellspacing="0" width="100%"
29         Font-Name="Arial" Font-Size="8pt"
30         HeaderStyle-BackColor="#cccc99"
```



```

31     ItemStyle-BackColor="#ffffff'
32     AlternatingItemStyle-BackColor="#cccccc' />
33
34 </form>
35 </body></html>

```

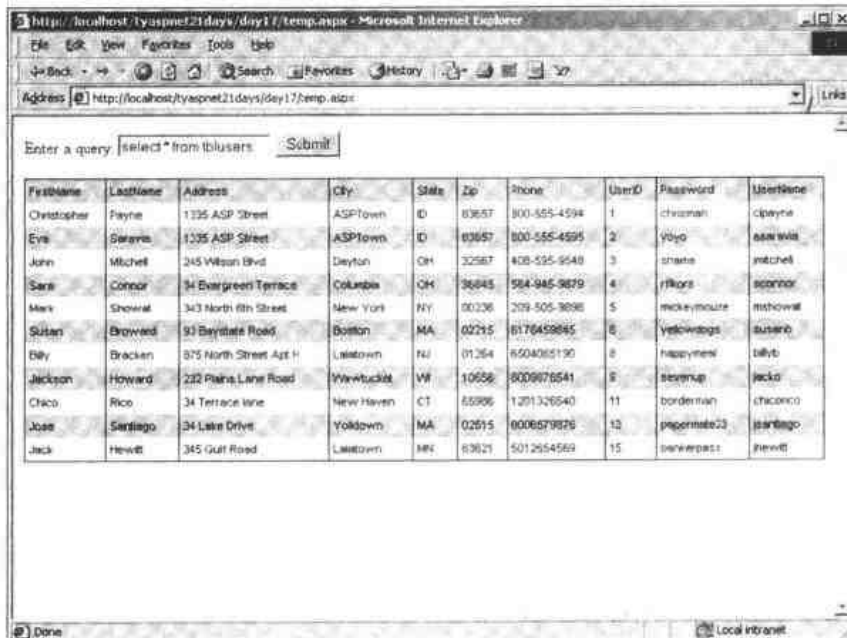
**分析：**这个页面非常简单，其唯一的函数是Submit方法。当用户在文本框中输入查询，并单击Submit按钮后，将新建代理类的一个实例，如第6行所示。还记得吗，您将该代理放置在名称空间TYASPNET中，因此您使用全名来声明变量（本章前一个例子中的代理没有被放置在某个特定的名称空间中，因此无需使用名称空间名）。

第9行调用SelectSQL方法，该方法返回一个DataSet。然后从第25行开始，您将返回的数据绑定到DataGrid。图17.7显示了返回的页面。

该ASP.NET页面可以位于世界的任何地方，而得到的结果不变。换句话说，该ASP.NET可以位于您在纽约的家中的计算机中，而服务可以位于东京的一台服务器上。

该服务能够接收来自任何地方的命令，因为这些命令是通过XML传输的，而XML只不过是纯文本而已，Internet上采用的安全措施通常被设计成允许纯文本通过。

Web服务让开发人员能够实现任何其认为合适的自定义功能（这不同于ASP.NET页面，在ASP.NET页面中，客户只能使用您创建的界面）。例如，使用上述Database服务及其返回的数据，开发人员可以创建在线通讯录应用程序，而另一个开发人员可以使用这些数据来确定统计信息。而您的服务只管返回数据，而不管数据如何被使用和显示。



Firstname	Lastname	Address	City	State	Zip	Phone	UserID	Password	Username
Christopher	Payne	1235 ASP Street	ASPTown	ID	03657	800-555-4594	1	chrismain	cpayne
Eve	Garera	1235 ASP Street	ASPTown	ID	03657	800-555-4595	2	Ydyo	emwila
John	Mitchell	245 Wilson Blvd	Dayton	OH	32567	408-555-9548	3	snare	mitchel
Sara	Connor	34 Evergreen Terrace	Columbus	OH	36845	584-945-9879	4	rflors	sconnor
Mark	Shenval	343 North 8th Street	New York	NY	00236	209-505-9898	5	mckeymouse	mshenval
Sultan	Broward	93 Baystate Road	Boston	MA	02215	6176459845	6	yellowdogs	sultanb
Billy	Bracken	875 North Street Apt H	Lakeland	FL	01264	8504065130	8	happyment	billyb
Jackson	Howard	222 Plains Lane Road	Weymouth	WI	10656	8009876541	9	stevemup	jacksn
Chico	Rico	34 Terrace Lane	New Haven	CT	65986	1201326540	11	bordenman	chicoico
Jose	Santiago	34 Lake Drive	Yokdown	MA	02515	6006579826	12	papermate23	santiago
Jack	Hewitt	345 Gulf Road	Lakelown	MS	63621	5012554569	15	benkepass	jhe Witt

图17.7 从服务返回的数据

**注意：**这些例子说明了业务对象和Web服务之间的相似性，它们要求在ASP.NET页面中实现编译后的对象。唯一的差别在于，对于Web服务，编译后的对象可以位于Internet的任何位置，然后由代理将实现和服务连接在一起。

17.3 关于使用 Web 服务的建议

如何使用 Web 服务随具体情形而异。还记得吗，前一章介绍过，可以使用三种不同的协议来访问 Web 服务：Http-Get、Http-Post 和 SOAP。前两种协议最容易实现，因为它们只需要 Web 服务的 URL。本章的例子使用的都是 SOAP，它为使用服务提供了更丰富的方法。

但使用 SOAP 有一个缺陷，您可能注意到了，在 SOAP 消息中的数据和命令进行编码非常复杂（虽然创建包含能够完成这项工作的方法的代理很简单）。由于 SOAP 必须发送 XML 模式的信息，因此发送的消息非常大。根据您或您的客户连接到 Internet 使用的连接类型，这可能影响性能。

另一种使用 Web 服务的方法是 Http-Get。如果知道服务和方法的 URL 以及方法接受的参数，便可以调用服务的方法。例如，您可以通过访问下面的 URL，来使用上一章创建的 Calculator 服务的 Add 方法：

```
http://localhost/tysapnet21days/day16/Calculator.asmx/  
➡Add?intA=1&intB=2
```

在 URL 后面加上斜杠和方法的名称，然后以查询字符串的方式加上参数。例如，服务的 URL 为：

```
http://localhost/tysapnet21days/day16/calculator.asmx  
后缀包含方法名和参数，为：  
/Add?intA=1&intB=2
```

被请求后，该 URL 将输出包含方法结果的 XML。在这个例子中，结果为：

```
<?xml version="1.0"?>  
<int xmlns="http://tempuri.org/">3</int>
```

也可以使用 Http-Post 操作来使用服务。您只需将 HTML 表单的 action 属性设置为 Add 方法：

```
http://localhost/tysapnet21days/day16/calculator.asmx/Add
```

只是一个后面带斜杠和方法名的 URL。当您提交该表单时，所有参数都将被发送给服务，并将生成相应的 XML 输出。只不过要记住的是，表单参数的名称必须与服务期待的参数相同。

那么，应该采用哪种方法呢？选择很简单，真的。当您要求服务发回复杂的数据时，如 DataSet 或类，则使用 SOAP。其他两种方法不能发送这类数据。

如果不要发回复杂的数据，而您的应用程序或客户可能对带宽比较敏感，则使用 Http-Get 或 Http-Post。这些方法发送的数据较少，因此性能更高。

应 该	不 应 该
需要复杂的数据或需要频繁访问 Web 服务时,应使用 SOAP。	当带宽有限时,不要使用 SOAP,而应该使用 Http-Get 或 Http-Post。

17.4 确保 Web 服务的安全

您经常需要确保 Web 服务的安全。毕竟即使是在一切东西共享的 Internet 时代，也必须对某些东西采取保护措施。假设您开发了一个 Web 服务，用户只需输入股票代码，便可获得该股票的实时

报价。通常，免费的股票报价会延迟 20 分钟，甚至更长时间，因此人们愿意为您提供的服务付费。因此您想将为付费者拒之门外。

前面介绍了如何创建公用的 Web 服务，任何知道该服务的人都可以使用（通过 disco 文件或直接通过 URL）。所幸的是，您可以采取措施来确保 Web 服务的安全，以便只有被授权者才能使用它们。

这种安全措施包括使用 SOAP（因此是 XML）来发送认证信息（作为命令的一部分）。由于信息被直接发送给 Web 服务，因此服务可以实现任何必要的认证。例如，可以访问数据库来确定用户的证书。

介绍技术细节之前，首先来看一个比喻。假设您要去一家非常高级的餐馆，该餐馆在您进门时将检查您的身份。为进入该餐馆，并享受其服务，您必须携带某种证件，如驾驶证。

为使用 Web 服务，您使用 SOAP 报头来传递身份信息。这是一条附加在能够提供自定义信息的服务的常规通信后面的 XML 消息，其作用与驾驶证类似。虽然它是随 SOAP 消息传输的，但却是一个独立的实体。SOAP 报头将被用来传递用户名和密码信息，以便只有您选择的用户才能访问该服务。

要传递 SOAP 报头，只需创建一个从 System.Web.Services.Protocols.SoapHeader 类派生而来的类。这样，Web 服务类便可以使用这个类来收集认证信息。例如，清单 17.5 显示了一个简单的 SOAP 报头类，它接受用户名和密码。

#### 清单 17.5 一个简单的 SOAP 报头类

```
1 Imports System
2 Imports System.Data
3 Imports System.Data.OleDb
4 Imports System.Web.Services
5 Imports System.Web.Services.Protocols
6
7 Namespace TYASPNRT
8
9     Public Class Authenticator : Inherits SoapHeader
10         Public Username as string
11         Public Password as string
12     End Class
```

**分析：**这个类叫做 Authenticator，可用于 Web 服务所在的文件中。它包含两个公有的字符串变量：Username 和 Password，这两个变量将包含来自客户的认证信息。

对于要保护的每个类，都应该声明一个 Authenticator 类的实例。您将使用该实例来显示对 Web 方法的访问。然后，除了 <WebMethod> 属性外，每个 Web 方法还必须使用属性 <SoapHeader>。

```
Public Class DataBase Service
    dim sHeader as Authenticator
    ...
    Public Function<WebMethod(), SoapHeader("sHeader")> MyMethod()
        as String
```

```

...
End Function
End Class

```

第 2 行创建了一个 `Authenticator` 类的实例，该实例被保存在变量 `sHeader` 中。然后，在函数声明中，您使用 `Authenticator` 实例的名称添加了 `<SoapHeader>` 属性。这告诉 ASP.NET，除了常规通信外，还有一个 SOAP 报头。没有这个报头，来自客户的任何消息都将被拒绝。

让我们使用上一章创建的 `Database` 服务来创建一个真实的例子。首先，修改 Web 服务类，使之能够利用自定义的 SOAP 报头。清单 17.6 列出了修改后的类。

**清单 17.6 一个安全的数据库服务类**

```

1 Imports System
2 Imports System.Data
3 Imports System.Data.OleDb
4 Imports System.Web.Services
5 Imports System.Web.Services.Protocols
6
7 Namespace TYASPNET
8
9     Public Class Authenticator : Inherits SoapHeader
10         Public Username as string
11         Public Password as string
12     End Class
13
14     Public Class SecureDatabaseService : Inherits WebService
15         private objConn as OleDbConnection
16         private objCmd as OleDbCommand
17         public sHeader as Authenticator
18
19         public function <WebMethod(), SoapHeader("sHeader")> _
20             SelectSQL(strSelect as string) as DataSet
21
22         if sHeader is Nothing then
23             throw new ArgumentNullException()
24         end if
25
26         if Authenticate(sHeader.UserName, sHeader.Password) _
27             then
28             try
29                 objConn = new OleDbConnection("Provider=" & _
30                     "Microsoft.Jet.OLEDB.4.0;" & _

```

```

31         'Data Source=C:\ASPNET\data\banking.mdb ')
32         dir, objDataCmd as OleDbDataAdapter = new _
33             OleDbDataAdapter(strSelect, objConn)
34
35         Dim objDS as new DataSet
36         objDataCmd.Fill(objDS, 'tblUsers')
37         return objDS
38     catch ex as OleDbException
39         return nothing
40     end try
41 end if
42 end function
43
44 Private Function Authenticate(strUser as String, strPass _
45     as string) as Boolean
46     try
47         dim intID as integer=0
48         objConn = new OleDbConnection("Provider=" & _
49             Microsoft.Jet.OLEDB.4.0;" & _
50             'Data Source=C:\ASPNET\data\banking.mdb")
51
52         dim strSelect as string="SELECT * FROM tblUsers WHERE UserID=" & _
53
54
55
56         objCmd = new OleDbCommand(strQuery, objConn)
57         objCmd.Connection.Open()
58         objCmd.ExecuteNonQuery()
59         objCmd.Connection.Close()
60         return true
61     catch ex as OleDbException
62         return false
63     end try
64 end function
65
66 private function Authenticate(strUser as string, _
67     strPass as string) as boolean
68     try
69         dim intID as integer = 0
70         objConn = new OleDbConnection("Provider=" & _

```

```

71         'Microsoft.Jet.OLEDB.4.0;' & _
72         "Data Source=C:\ASPNET\data\banking.mdb")
73
74         dim strSelect as string = "SELECT UserID FROM " & _
75         'tblUsers WHERE Username = ' & strUser & _
76         " AND Password = " & strPass & ""
77         objCmd = new OleDbCommand(strSelect, objConn)
78         objCmd.Connection.Open()
79
80         Dim objReader as OleDbDataReader
81         objReader = objCmd.ExecuteReader
82         do while objReader.Read
83             intID = objReader.GetInt32(0)
84         loop
85         objReader.Close
86
87         if intID <> 0 then
88             return true
89         else
90             return false
91         end if
92     catch ex as OleDbException
93         return false
94     end try
95 end function
96 End Class
97
98 End Namespace

```

**分析：**从第9行开始是您创建的认证类，它是从SoapHeader派生而来的。客户和服务通信时，这个类将作为额外的行装被发送。它只有两个变量：Username和Password，您将使用它们来验证用户的身份。

第 17 行创建了一个新的 SOAP 报头类实例，名叫 sHeader。该实例将包含客户传递的认证信息，而 Web 方法将使用这些信息来确定用户是否有必要的权限。

新的数据库服务类叫做 SecureDatabaseServices，这样您将很容易记住该服务。SelectSQL 函数从 19 行开始，但不同于上一章的同一个函数。首先，声明中包括 SoapHeader 属性，以设置 SOAP 报头类实例 sHeader。声明的其他部分则保持不变。

在该函数中，您首先客户提供的用户名和明码，如果没有这些参数，sHeader 将等于 nothing，您将退出该函数。在 23 行，您引发了一个自定义的异常错误，用于告诉客户，它没有提供密码和用户名。

注意：并不一定要引发某个特定的错误，因为如果没有传递合适的报头，ASP.NET将提醒用户，如图17.8所示。

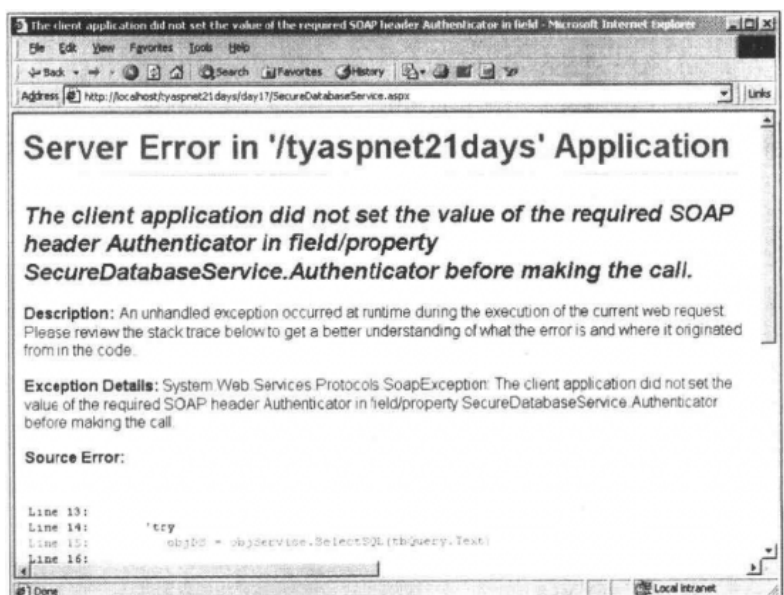


图 17.8 用户未提供 SOAP 报头时出现的错误

第 26 行调用自定义的 `Authenticate` 方法，它查询数据库，确定用户是否被允许访问该函数。如果证书有效，则 `Authenticate` 将返回 `true`，并跳到从 28 行开始的函数体中继续执行。这些代码您应该很熟悉，它是通过修改这个类的最后一个版本而得到的，它执行提供的查询，并返回一个 `DataSet` 对象。

从 44 行开始的 `Authenticate` 函数执行有效性检查，该函数查看数据库，以确定 SOAP 报头提供的用户名和密码描述的是否是一个真正的用户，如果是，则返回 `true`，否则返回 `false`。

使用下面的命令编译该服务：

```
vbc /t:library /out:..\bin\TYASPNET.dll /r:System.dll
  /r:System.data.dll /r:System.xml.dll /r:System.web.dll
  /r:System.Web.Services.dll Database.vb user2.vb
```

在服务器上创建一个 `.asmx` 文件，其中只需指定类名即可，如下所示：

```
<%@ WebService Class="TYASPNET.SecureDatabaseServices" %>
```

将该文件保存为 `SecureDatabaseService.asmx`。接着，需要在客户端创建一个代理类（在这个例子中，客户机和服务器是同一台计算机）。使用下来的命令来生成代理：

```
wsdl /language:VB /namespace:TYASPNET.Clients
  http://localhost/tyaspnet21days/day17/
  SecureDatabaseService.asmx?WSDL
```

注意到，您将该代理方在一个新的名称空间（`TYASPNET.Client`）中。由于所有的工作都是在同一台计算机上完成的，这可以防止名称空间中的类名重叠。

如果查看生成的代码，您将发现，代理从服务那里取得两样东西：认证器类和安全数据库类。在客户应用程序中，您将使用这两个类来访问服务。

最后，使用下面的命令编译代理：

```
vbc /t:library /out:..\bin\TYASPNET.Clients.dll /r:System.dll
➡/r:System.data.dll /r:System.XML.dll /r:System.Web.dll
➡/r:System.Web.Services.dll SecureDatabaseService.vb
```

剩下的工作是创建一个访问该服务的 ASP.NET 页面。该页面让用户输入用户名和密码以及用户想执行的查询。清单 17.7 列出了该页面的代码。

#### 清单 17.7 访问服务的 ASP.NET 页面

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3
4 <script runat="server">
5     sub Submit(obj as object, e as EventArgs)
6         dim objService as new TYASPNET.Clients.SecureDatabaseService
7         dim sHeader as new TYASPNET.Clients.Authenticator
8         dim objDS as new DataSet
9
10        sHeader.Username = tbUser.Text
11        sHeader.Password = tbPass.Text
12        objService.AuthenticatorValue = sHeader
13
14        try
15            objDS = objService.SelectSQL(tbQuery.Text)
16
17            DataGrid1.DataSource = objDS
18            DataGrid1.DataMember = "tblUsers"
19            DataGrid1.DataBind()
20        catch ex as exception
21            Response.Write("Invalid username or password")
22        end try
23    end sub
24 </script>
25
26 <html><body>
27     <form runat="server">
28         Username:
29         <asp:Textbox id="tbUser" runat="server" /><br>
30         Password:
31         <asp:Textbox id="tbPass" runat="server"
32             TextMode="password" /><p>
```



```

33      Enter a query:
34      <asp:Textbox id="tbQuery" runat="server" />
35      <asp:Button id="btSubmit" runat="server"
36          text="Submit"
37          OnClick="Submit" />
38      <p>
39      <asp:DataGrid id="DataGrid1"
40          runat="server" BorderColor="black"
41          GridLines="Vertical" cellpadding="4"
42          cellspacing="0" width="100%"
43          Font-Name="Arial" Font-Size="8pt"
44          HeaderStyle-BackColor="#cccc99"
45          ItemStyle-BackColor="#ffffff"
46          AlternatingItemStyle-BackColor="#cccccc" />
47
48      </form>
49 </body></html>

```

分析：页面中只有一个方法：Submit——Submit按钮的事件处理程序。当该方法被激发时，它创建一个认证器类对象和一个安全数据库类对象。11~12行将认证器对象的属性设置为文本框中的值，然后将安全数据库对象的AuthenticatorValue的值设置为该认证器对象。这样代理便知道需要随命令一起发送哪些信息。

最后，在 try 语句块中，执行服务的方法，并将数据绑定到一个 DataGrid。

请在浏览器中装载该页面，并输入用户名、密码和 SQL 查询。如果用户名和密码有效（即它们代表 tblUsers 表中的一个用户），则可以看到返回的数据，如图 17.9 所示。

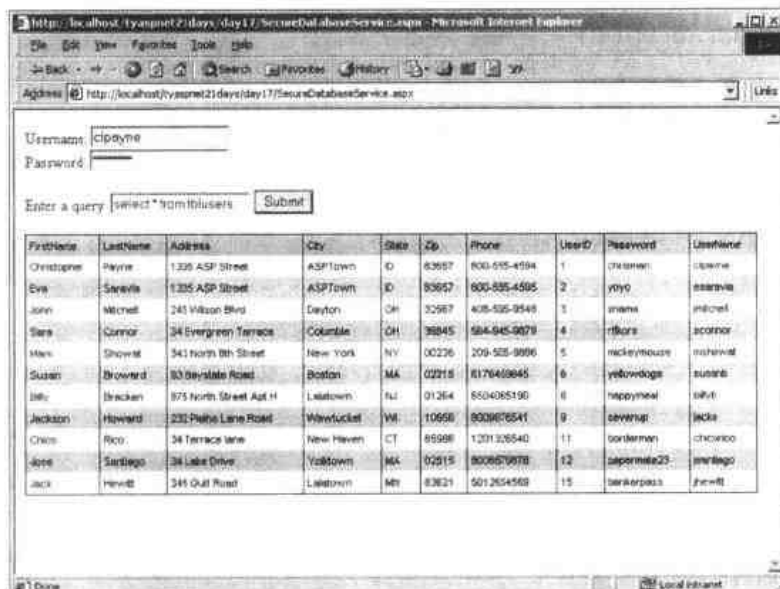


图 17.9 成功使用安全服务得到的结果

如果您输入的用户名和密码无效,则将看到一条错误消息,指出用户名或密码无效,如图 17.10 所示。这也是使用 try 语句块的原因所在,如果不这样做,则当输入的用户名或密码无效时,将返回一个空的 DataSet。当您将它绑定到一个 DataGrid 时,将引发错误。

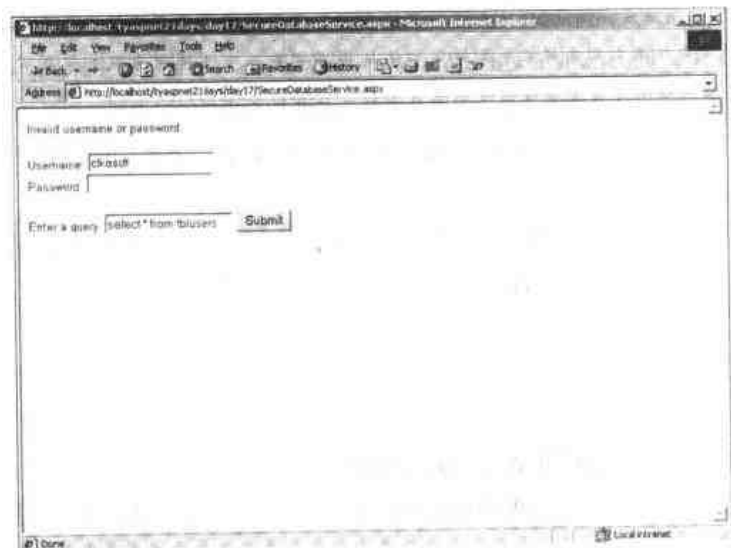


图 17.10 使用安全服务失败的情况

对实现安全 Web 服务来说, SOAP 报头是一个常用、高效的工具。虽然这里是使用数据库来验证认证信息,但您可以使用任何方式来验证客户提供的信息。

但使用 SOAP 报头有个缺点。通过 SOAP 报头在 Internet 上发送认证信息时,是以 XML 格式的纯文本发送的。这意味着它们容易被窃取。避免这种问题的方法之一是在发送之前对认证信息进行加密,并给 Web 服务提供对应的解密算法。但加密不再本书的讨论范围之内。有关确保应用程序安全的其他方式,请参见第 21 章“确保 ASP.NET 应用程序的安全”。

## 17.5 这不是 ASP

对 ASP 开发人员而言,让客户访问 Web 应用程序是一个新的概念。传统 ASP 让开发人员将应用程序放到网上,以便客户能够访问它们。而 Web 服务则是通过 Internet 访问应用程序的组件,并将其用于其他程序,这是一种全新的概念。

让应用程序的两个独立的部分远程相互通信是一个全新的概念。以前,业务逻辑和 ASP 页面必须位于同一台计算机上(至少是同一个网络上)。而在 ASP.NET 中,这些组件可以位于任何地方,使得能够创建真正的分布式应用程序。

使用 SOAP,通过标准语言(如 XML)来发送命令和数据是 Web 服务成功的关键因素。这些 SOAP 指令可以被发送到任何地方,甚至可以通过防火墙,因为它们依赖的是纯文本消息。

## 17.6 总 结

本章介绍了如何在 ASP.NET 页面中访问 Web 服务。对 ASP.NET 开发人员来说,使用 Web 服

务和开发 Web 服务同样重要。

使用 Web 服务包括三个步骤：发现、生成代理类和实现代理。

disco 工具用于执行服务发现工作，它生成两个文件：一个描述发现的结果（通常叫做 results.discomap），另一个包含的内容与服务器上的发现文件相同。

Web 服务描述语言工具 wsdl.exe 用于生成代理类。这个类通过提供内置的对使用 SOAP 发送和接收 XML 消息的支持，使方法能够远程执行 Web 服务命令。代理类被生成后，就可以像其他业务对象那样编译它。能够您便可以在应用程序中通过代理执行服务的方法。

本章还介绍了如何使用自定义 SOAP 报头确保 Web 服务的安全。除了消息外，这些报头也将被发送给服务。

下一章将介绍如何配置 ASP.NET 应用程序。在本书中，您已经见到过使用文件 web.config 和 global.asax 的例子，而下一章将介绍它们的功能以及如何使用它们。

## 17.7 问与答

问：在代理类中，也能看到非 Web 方法函数吗？

答：不能。代理只能看到包含 <WebMethod(> 属性的方法。通过 Web 根本看不到没有该属性的方法。

问：除了确保安全外，SOAP 报头还可以用于实现其他功能吗？

答：完全可以。SOAP 报头可以包含任何自定义信息。使用 <SoapHeader(> 属性的方法都要求随通信一起发送该报头。因此，这是一种方便的、发送不必包含在方法的参数列表中的信息的途径。

## 17.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 17.8.1 小测验

1. 代理类是什么？
2. 编写一个使用 disco.exe 工具的例子。
3. 编写一个使用 wsdl.exe 工具的例子。
4. 判断正误：生成代理之前，必须执行发现操作。
5. 使用 SOAP 报头来访问服务时，需要哪两个名称空间？
6. 创建一个简单的 SOAP 报头类。在 Web 方法中您将使用这个类做什么？在客户端呢？

### 17.8.2 练习

1. 为前一章的练习 2 创建的 ConvertUnits 服务创建一个代理类，并创建一个让用户输入值的界面。

## 第 18 章

# 配置和部署 ASP.NET 应用程序

通过前面的 17 章，您知道如何创建应用程序的大部分部件，但还不知道如何将应用程序作为一个整体进行控制。之前创建的应用程序主要由单独的页面组成，偶尔也与一些辅助文件进行交互。本章将介绍如何将这些页面组合成一个统一的应用程序，可以在一个地方控制所有的设置和时间。

对 ASP.NET 来说，Global.asax 和 web.config 是两个非常重要的文件。前者控制应用程序时间，让您能够在页面首次被请求，甚至每次被请求时执行某些代码；后者包含应用程序的设置，包括会话状态的行为或将安全措施加在什么地方。本章将介绍如何利用这些文件。

最后，您将知道在另一个服务器上部署应用程序是多么得容易。很多时候，开发人员在测试机器上创建应用程序，完成并消除所有的 bug 后，再将它转移到生产机器上。ASP.NET 将这一过程的复杂性降到了最低。

本章包含以下内容：

- 深入讨论 ASP.NET 应用程序；
- global.asax 是什么？如何使用它？
- 如何使用 web.config 配置应用程序？
- 如何使用一些常用的配置设置？
- 如果在 ASP.NET 中创建自定义的设置？
- 如何部署 ASP.NET 应用程序？

### 18.1 ASP.NET 应用程序简介

至此，您已经了解了 ASP.NET 中的大部分概念，从关于 Web 表单的基本知识到创建复杂的业务对象。使用这些知识，可以创建出功能强大的页面来执行几乎任何工作。接下来需要将所有的东西组合成一个应用程序，并配置和部署该应用程序。

您可以随便将一些页面放在一起，并将其称作是一个应用程序，但正式的定义远不止这些。ASP.NET 应用程序的定义如下：Web 应用程序服务器中某个虚拟目录下可运行或调用的全部文件、页面、事件处理程序、模块以及可执行代码。这包含的内容相当多。

第 1 章介绍过，虚拟目录是任何不包含在根目录下，但对客户来说就像是在根目录下的目录。例如，如果使用默认设置安装 IIS，则 c:\inetpub\wwwroot 将是一个虚拟目录（您的根目录）。对 Web 客户来说，它是 http://localhost；另一个虚拟目录是 c:\myweb\mysite，而客户看到的是

`http://localhost/mysite`。后一个目录不在根目录下，但 Web 客户不知道这种差别，可以通过 Internet 服务管理器很容易地创建虚拟目录（参见第 1 章）。

这些虚拟目录文件夹的子文件夹不一定非得也是虚拟目录，但也可设置为虚拟目录。图 18.1 显示了安装 Windows 2000 的计算机上的虚拟目录。

ASP.NET 应用程序是一个完全独立的实体。每个应用程序都在自己的 .NET 运行阶段应用程序域中进行处理（参见第 2 章的“处理技术”一节），这意味着每个应用程序都可以有自己的配置值和属性设置。因此，如果有两个虚拟目录：`http://localhost/MyASP` 和 `http://localhost/MyASP/App2`，则后者将是一个完全独立的实体，它不被包含或嵌入到前者中。

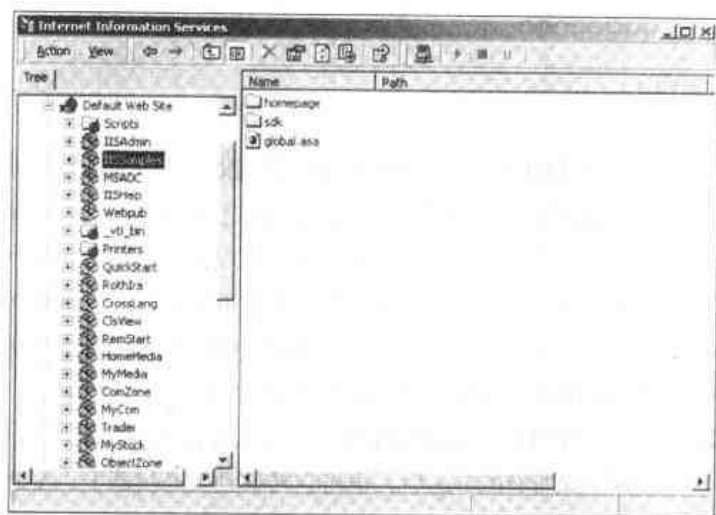


图 18.1 通过 Internet 服务管理器显示的 Windows 2000 机器上的 IIS 5.0 中的虚拟目录列表

ASP.NET 应用程序在用户首次向服务器提出请求时被创建（后面将详细介绍），之后您便可以通过 `global.asax` 文件（将在本章后面介绍），使用各种应用程序事件来控制程序的执行。

### 18.1.1 \bin 目录

**新术语：**ASP.NET 为其应用程序引入了一个新的特殊的目录：`\bin` 目录（也叫组合体仓库）。在关于业务对象的一章介绍过，该目录包含了应用程序使用的、编译后的二进制文件，如 DLL。每个应用程序都有自己的组合体仓库，但也可以从父类那里继承。

该目录对 ASP.NET 应用程序至关重要，它为应用程序使用的对象提供了易于访问的存储器。对 `.aspx` 文件而言，存放在这里的组合体是自动可用的，因此不用为复杂的创建过程而操心。关于如何使用该目录的更详细的信息，请参见第 15 章。

## 18.2 GlobalAsax

您可能没有把 ASP.NET 页面看作是一个真正的应用程序，人们很容易这样认为，因为页面与多组文本文件关联在一起。我们通常将传统的应用程序看作是一个可执行文件，当然还可能包括其他支持文件，如 DLL 或 `.inf` 文件。ASP.NET 应用程序与此不同，因为用户可以从任何页面进入——没有固定的起点，而且大部分关联文件都是人可以读懂的。

实际上，IIS 中的每个虚拟目录都是一个应用程序。应用程序总是在其页面第一次被请求时便

开始运行，而不管请求的是哪个页面。当用户从一个页面移到另一个页面时，处理工作将在后台进行，以处理应用程序。和任何传统应用程序一样，它也可能崩溃，并需要重新启动。区别在于传统应用程序在台式机上启动并运行，让用户直接与之交互，而 ASP.NET 应用程序在服务器上启动并运行，用户需要使用浏览器才能访问它。

.NET 框架中的任何东西都是对象，包括 ASP.NET 应用程序。与 ASP.NET 页面和服务器控件一样，HttpApplication 对象也提供了方法、属性和事件。再次阅读前面的段落，您将能推断出：虚拟目录的页面被首次请求时，将实例化一个 HttpApplication 对象。

**注意：**只有当虚拟目录的页面首次被请求时，应用程序才会启动。这不是说，其中的任何页面首次被请求时，应用程序都会启动。例如，如果应用程序由两个文件（default.aspx 和 exit.aspx）组成，则该应用程序将在用户首次请求页面（可以是其中的任何一个文件）时启动，而不是每个页面首次被请求时都启动。前者意味着应用程序只启动一次，而后者意味着每当用户请求新页面时，应用程序都要启动。

那么如何控制 HttpApplication 对象（从而控制 ASP.NET 应用程序）呢？在各个页面中进行的处理操作确实能够完成（某些）任务，但不能真正地将应用程序作为一个整体进行控制。一个页面无法直接影响另一个页面。必须有一个能够控制应用程序执行的中央位置：global.asax。

global.asax 文件被称为 ASP.NET 应用程序文件，让您能够规划 HttpApplication 对象：可以像对待其他任何对象一样，通过其方法和事件控制 ASP.NET 应用程序（介绍一些背景知识后，将介绍如何控制）。

**注意：**global.asax 完全是可选的，如果没有为应用程序创建该文件，则程序将以默认的方式运行，这足以能够完成工作。但如果您想实现其他的功能，则必须创建该文件。

global.asax 文件位于 ASP.NET 应用程序的根目录下。例如，如果您完成本书的范例，则应该创建了虚拟目录 `http://localhost/tyaspnet21days`，它对应的物理目录是 `c:\inetpub\wwwroot\tyaspnet21days`。global.asax 文件将位于该目录下，它控制应用程序的运行。

ASP.NET 将该文件配置成从浏览器中无法访问。这样，黑客便无法闯入并修改您的应用程序。如果访问者在浏览器中输入 `www.yoursite.com/global.asax`，则它将收到一个错误，如图 18.2 所示。

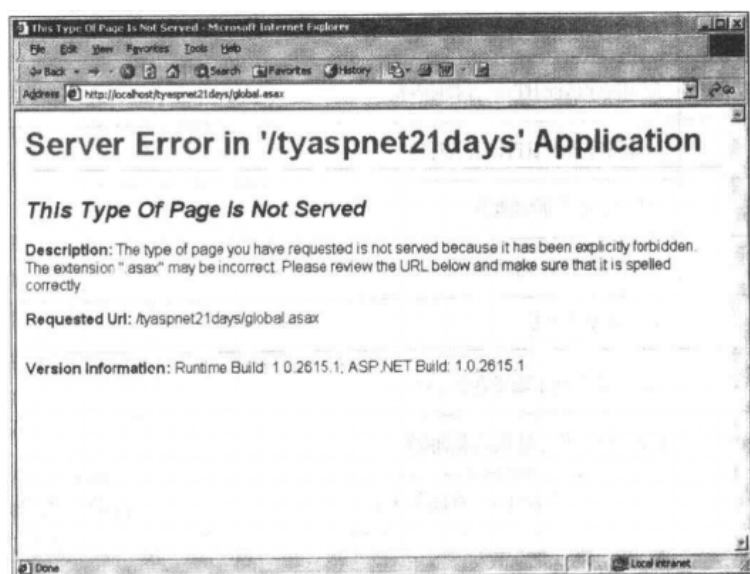


图 18.2 在浏览器中请求 global.asax 将引发错误

正如前面指出的, `global.asax` 是由 .NET 框架管理的, 每当该文件被修改时, CLR 都会重新启动应用程序, 以便使新的设置生效。这将影响用户, 因此应尽可能少地修改该文件。

### 18.2.1 HttpApplication 类

`HttpApplication` 对象让您能够控制 ASP.NET 应用程序, 但 `global.asax` 文件是如何与之协调的呢? 在运行阶段, `global.asax` 被编译成一个从 `HttpApplication` 类派生而来的动态类, 因此您能够使用 `global.asax` 文件来访问父对象 `HttpApplication` 能够使用的对象和事件。

应用程序初始化时, 大量的 `HttpApplication` 实例被创建, 而每个实例每次处理一个特定的 HTTP 请求。它处理所有请求的处理工作, 完成后, 便可以分配给另一个请求。该实例将负责管理该特定请求 (在请求的生存期内)。

假设您拥有一个高档餐馆, 每个服务员只同时为一桌客人提供服务。当客人到来后, 同一个服务员将服务满足客人的所有要求, 如安排座位、介绍特色服务以及端菜等 (虽然端菜的工作可能是由另一个人完成的, 但由该服务员负责)。同样, 一个 `HttpApplication` 实例负责“接待” HTTP 请求: 记录请求变量和交付页面 (实际的交付工作是由 `HttpHandler` 完成的, 但 `HttpApplication` 告诉它如何做)。

由于 `global.asax` 类是从 `HttpApplication` 类派生而来的, 因此它继承了后者的许多方法和事件。这些方法和事件让您能够访问 ASP.NET 应用程序。在 `global.asax` 中, 您定义处理 `HttpApplication` 实例引发的事件的处理程序, 它负责所有的应用程序处理工作。这些方法必须遵循事件处理程序的命名约定: `Application_EventName(event, arguments)`。例如, `Application_Start(obj as object, e as EventArgs)`。

### 18.2.2 编写 global.asax

`global.asax` 文件的运行方式与 `.aspx` 文件有许多类似的地方。您使用页面编译指令 (`@ Imports`、`@ Application` 和 `@ Assembly`)、服务器端包含 (参见下一章) 和代码声明块。您使用 `global.asax` 来同步 `HttpApplication` 类暴露的所有事件 (如表 18.1 所示)。

表 18.1 HttpApplication 类的事件

事 件	描 述
<code>AcquireRequestState</code>	应用程序为请求取得缓存时激发
<code>AuthenticateRequest</code>	应用程序认证 HTTP 请求时激发
<code>AuthorizeRequest</code>	应用程序批准 HTTP 请求时激发
<code>BeginRequest</code>	HTTP 请求开始时激发
<code>EndRequest</code>	HTTP 请求结束时激发
<code>Error</code>	发生错误时激发
<code>PostRequestHandlerExecute</code>	HTTP 处理程序处理请求后激发
<code>PreRequestHandlerExecute</code>	HTTP 处理程序处理请求前激发
<code>PreSendRequestContent</code>	如果请求额外的内容 (查询字符串、表单变量等), 则在这些内容收到之前激发
<code>PreSendRequestHeaders</code>	收到请求报头之前激发

续表

事 件	描 述
ReleaseRequestState	应用程序释放请求的会话状态时激发
ResolveRequestCache	应用程序为请求解决缓存时激发
UpdateRequestCache	应用程序为请求更新和释放缓存时激发

来看一个 global.asax 的例子，如清单 18.1 所示。

**清单 18.1 一个简单的 global.asax**

```

1:<script language='VB' runat="server">
2:
3:  Sub Application_Start(Sender As Object,E As EventArgs)
4:      Application("Time") = System.DateTime.Now
5:  End Sub
6:
7:  Sub Application_AcquireRequestState(Sender As Object, _
8:                                     E As EventArgs)
9:      Response.Write("Acquiring request session state" & _
10:                   "...<br>")
11:  End Sub
12:
13:  Sub Application_AuthenticateRequest(Sender As Object, _
14:                                     E As EventArgs)
15:      Response.Write("Authenticating request..."<br>")
16:  End Sub
17:
18:  Sub Application_AuthorizeRequest(Sender As Object, _
19:                                  E As EventArgs)
20:      Response.Write("Authorizing request..."<br>")
21:  End Sub
22:
23:  Sub Application_PostRequestHandlerExecute(Sender As _
24:                                             Object,E As EventArgs)
25:      Response.Write("Request handler executed..."<br>")
26:  End Sub
27:
28:  Sub Application_PreRequestHandlerExecute(Sender As _
29:                                           Object,E As EventArgs)
30:      Response.Write("Request handler executed..."<br>")

```



```
31: End Sub
32:
33: Sub Application_PreSendRequestContent(Sender As _
34:     Object, E As EventArgs)
35:     Response.Write("Receiving request content...<br>")
36: End Sub
37:
38: Sub Application_PreSendRequestHeaders(Sender As _
39:     Object, E As EventArgs)
40:     Response.Write("Receiving request headers...<br>")
41: End Sub
42:
43: Sub Application_ReleaseRequestState(Sender As Object, _
44:     E As EventArgs)
45:     Response.Write("Releasing request state...<br>")
46: End Sub
47:
48: Sub Application_ResolveRequestCache(Sender As Object, _
49:     E As EventArgs)
50:     Response.Write("Resolving request cache...<br>")
51: End Sub
52:
53: Sub Application_UpdateRequestCache(Sender As Object, E _
54:     As EventArgs)
55:     Response.Write("Updating request cache...<br>")
56: End Sub
57:
58: Sub Application_Error(Sender As Object, E As EventArgs)
59:     Response.Write(Sender.Request.IsAuthenticated & _
60:         " is authenticating request...<br>")
61: End Sub
62:
63: Sub Session_Start(Sender As Object, E As EventArgs)
64:     Response.Write("Session is starting...<br>")
65: End Sub
66:
67: Sub Application_BeginRequest(Sender As Object, E _
68:     As EventArgs)
69:     Response.Write("<b>Process</b><p>")
70:     Response.Write("Request is starting...<br>")
```

```

71: End Sub
72:
73: Sub Application_EndRequest(Sender As Object, E As _
74:                               EventArgs)
75:     Response.Write("Request is ending...<br>")
76: End Sub
77:
78:</script>

```

**分析:** 在这个文件中, 您为表 18.1 中的大部分时间提供了功能。第 3 行使用一个 `HttpApplication` 对象来记录应用程序开始运行的时间, 供以后使用(关于保存状态的更详细的信息, 请参考第 4 章)。每个事件都只是将一个字符串写入到页面中, 让您知道其功能——这让您能够知道处理请求的次序。

对于清单 18.1 中的 `global.asax` 文件, 当每个应用程序级事件发生时, 将输出一条简单的消息。注意, 其中的一些事件在 ASP.NET 页面每次被查看时都会发生。为证实这一点, 请创建如清单 18.2 所示的 ASP.NET 页面。

#### 清单 18.2 测试 `global.asax` 的事件处理程序

```

1 <%@ Page Language="VB" %>
2
3 <script language="VB" runat="server">
4     Sub Page_Load(Sender As Object, E As EventArgs)
5         lblOutput.text = "Page loading...<p>" & _
6             "Application started at: " & Application("Time") & "...<br>" & _
7             "Current time: " & DateTime.Now & "...<br>"
8     End Sub
9
10    Sub Click(obj As Object, E As EventArgs)
11        Session.Abandon()
12    End Sub
13 </script>
14
15 <html><body>
16     <form runat="server">
17         <table align="center">
18             <tr>
19                 <td valign="top" width="300"><b>Output</b></td>
20             </tr>
21             <tr>
22                 <td valign="top">
23                     <asp:label id="lblOutput" runat="server"/><p>
24                     <asp:button id="btSubmit" runat="server"

```

```

25         OnClick="Click" Text="End This Session"/>
26     </td>
27 </tr>
28 </table>
29 </form>
30 </body></html>

```

分析：将该文件保存为 Listing1802.aspx。被请求时，该页面将一些消息显示给用户，包括应用程序开始运行的时间和当前的时间（见 5—7 行）。第 25 行包含一个按钮控件，当用户单击该按钮时，将删除当前的会话。这有助于弄清楚 HttpApplication 对象是如何处理会话的。图 18.3 显示了请求结果。

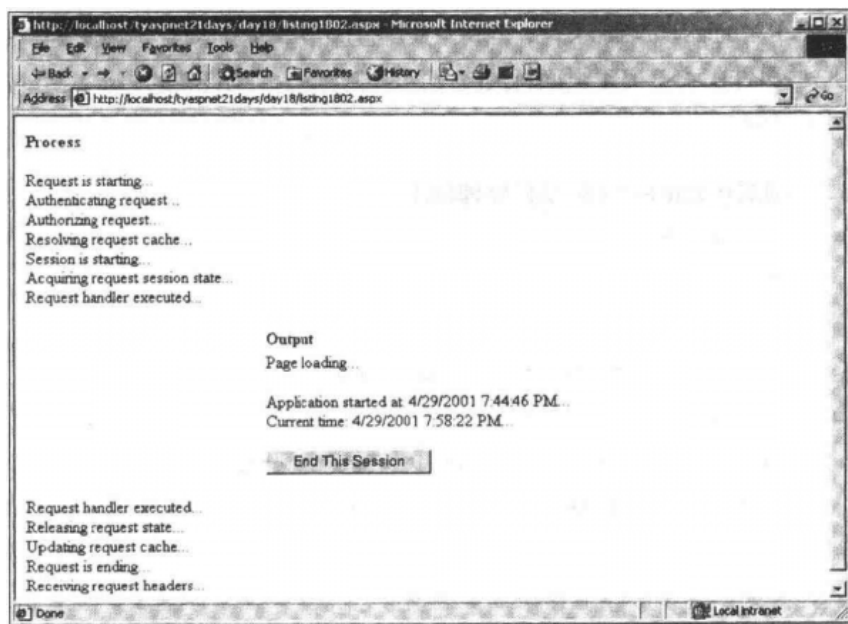


图 18.3 清单 18.2 生成的页面

当该页面首次被请求时，将发生多个 HttpApplication 事件发生，而 global.asax 通过输出文本来处理它们。在图 18.2 的左边可以看到 HttpApplication 对象的执行过程，而 Listing1802.aspx 文件的输出位于图中的右边。请注意从页面中间到底部的输出，应用程序文件处理请求并解决缓存和会话状态，然后 HTTP 请求处理程序（将在下一节讨论）接过控制权，并生成页面输出。最后，应用程序在请求结束前执行其他的一些功能。换句话说，方法按下面的次序执行：

1. Application\_OnStart
2. Application\_BeginRequest
3. Application\_AuthenticateRequest
4. Application\_AuthorizeRequest
5. Application\_ResolveRequestCache
6. Session\_Start
7. Application\_AcquireRequestState
8. Application\_PreRequestHandlerExecute
9. The Page\_Load event inside the.aspx file, followed by any other page outputs

- 10. Application\_PostRequestHandlerExecute
- 11. Application\_ReleaseRequestState
- 12. Application\_UpdateRequestCache
- 13. Application\_EndRequest
- 14. Application\_PreSendRequestHeaders

**注意：**实际发生的事件取决于请求页面时的情况。例如，如果是第二次请求页面，则方法 Session\_Start 不会执行，因此您看不到图 18.3 中的“Session is starting...”。

请注意应用程序开始（由 Application\_Start 设置）的时间和实际交付页面的时间之间的差异（实际的时间差取决于计算机的配置）。以后请求时，将不要这么长的时间，因为应用程序已经运行，而页面也已经被编译过。以任何方式修改文件 global.asax 都将重新启动应用程序。

图 18.3 中的 End This Session 按钮使用 Session 对象的 Abandon 方法重新开始会话，因此将再次出现“Session is starting...”。

因此，文件 global.asax 让您几乎能够控制 ASP.NET 处理的各个方面。您可以使用 HttpApplication 对象的事件来执行用户看不到的操作，如认证或其他个性化的操作——选择权在您手中。

## 18.3 配置 ASP.NET

就像需要控制应用程序的处理过程一样，也需要配置应用程序的设置。这包括处理页面如何被显示、如何被编译，控制对应用程序各个部分的访问等。您马上就会知道，ASP.NET 应用程序包含大量可配置的部件，需要一个很好的系统来管理所有哪些设置。

ASP.NET 提供一个层次式配置系统，其可扩展性非常好，最为重要的是，它是基于文本的，这意味着开发人员可以远程访问和修改配置。

**新术语：**在层次式配置（hierarchical configuration）中，根据 Web 站点的虚拟目录结构使用应用程序配置信息。子目录可以继承或覆盖父目录的配置选项。事实上，所有目录的配置都是从一个默认的系统配置文件 machine.config 派生而来的，该文件通常位于目录 WinNT\Microsoft.NET\Framework\<version>\CONFIG 中。这在设计应用程序方面给您提供了很大的灵活性。

这个健壮的系统是使用文件 web.config（一个基于文本的 XML 文件）实现的。下面详细地介绍该文件。

### 18.3.1 Web.config

本书曾多次提到过文件 web.config。到目前为止，您知道该文件包含了应用程序用来控制其行为的信息。这个文件并没有什么特别的：它只不过是保存了 ASP.NET 能够识别的关键字和值而已。这些值很容易修改，事实上，您可以加入自定义的关键字和值，以控制 ASP.NET 未为您处理的其他设置（后面将更详细地介绍）。

正如前面介绍过的，ASP.NET 配置系统是层次式的，就 web.config 文件而言，这意味着每个 web.config 文件的设置值应用其所在的目录及其子目录。因此，如果要为整个应用程序提供设置，只需将该文件放在应用程序根目录下。如果某个子目录对应用程序有特殊的要求（如不允许未经授权的用户使用），则只需另外创建一个 web.config 文件，其中包含新的设置。子目录将继承父目录的设置，并覆盖相同的选项。

然而，层次式结构是基于虚拟目录路径，而不是实际的物理路径的。为理解这一概念，假设某 web 站点的目录结构如图 18.4 所示。

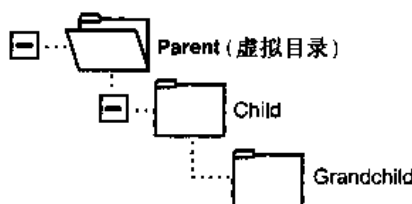


图 18.4 一个目录结构的范例

假设 Parent 是唯一的一个虚拟目录。Parent、Child 和 Grandchild 都继承了系统级 web.config 的设置。Parent 目录可以使用本地的 web.config 文件覆盖或添加设置。如果浏览器使用 URL `http://localhost/Parent/Child` 访问 Child 目录，则该目录将继承 Parent 的设置，而 Grandchild 也是如此。则两个子目录也可以使用自己的 web.config 文件覆盖 Parent 的设置。

现在假设 Child 也是一个虚拟目录，可以使用 `http://localhost/child` 进行访问。由于 web.config 设置是根据虚拟目录（而不是物理路径）被处理的，因此它不会继承 Parent 的任何设置，而 Grandchild 将随您访问它的方式的不同，而继承不同的设置。使用 `http://localhost/Parent/child/Grandchild` 和使用 `http://localhost /child/Grandchild` 访问 Grandchild 将导致它的设置不同，因此在组织虚拟目录的结构时一定要小心。

和 global.asax 文件一样，ASP.NET 也禁止通过 Web 客户访问文件 web.config。如果试图从浏览器访问该文件，则将返回与图 18.2 相同的错误。这对于防止黑客入侵应用程序很有帮助。

和 global.asax 一样，该文件被修改后，系统将自动发现，并重新启动应用程序，以使新的设置生效。

在理论上准备充分后，让我们来看看这个文件。Web.config 的基本格式非常简单：

```
<configuration>
  <configSections>
    <!--Handlers declared here -->
  </configSections>
  <system.web>
    <!--ASP.NET Configuration Settings Would Go Here -->
  </system.web>
  <system.net>
    <!--.NET Runtime Configuration Settings Would Go Here -->
  </system.net>
</configuration>
```

其中没有脚本块和 HTML 代码，只包括 XML。在标记 <configuration> 中，通常是两个不同的元素：配置段处理程序和配置段设置。前者声明了 web.config 文件中的数据类型；后者包括设置的关键字/值对。

由元素 <configSections> 表示的配置段处理程序处理 web.config 中的 XML 数据，并根据数据返回相应的对象。您必须指定这些处理程序，以便使 ASP.NET 知道有哪些类型的配置数据。例如，如果

名为“XmlFiles”的设置包含一系列 XML 文件，则需要一个对应的名叫“XmlFiles”的配置段处理程序，当它被调用时，将给 ASP.NET 页面返回一个 `XmlTextReader` 对象。所幸的是，ASP.NET 已经在默认的 `machine.config` 文件中提供了大量的段处理程序，因此您无需自己声明这些段处理程序。

例如，清单 18.3 列出了 `system.web` 段及其子段的默认配置段的部分内容。

**清单 18.3 配置段处理程序范例**

```

1: <configuration>
2:   <configSections>
3:     <sectionGroup name="system.web">
4:       <section name="browserCaps"
5:         type="System.Web.Configuration.
6:           HttpCapabilitiesSectionHandler, System.Web" />
7:       <section name="clientTarget"
8:         type="System.Web.Configuration.
9:           ClientTargetSectionHandler, System.Web" />
10:      <section name="compilation"
11:        type="System.Web.UI.CompilationConfigurationHandler,
12:          System.Web" />
13:      <section name="pages"
14:        type="System.Web.UI.PagesConfigurationHandler,
15:          System.Web" />
16:    </sectionGroup>
17:  </configSections>
18: </configuration>

```

**分析：**正如您看到的，该文件非常复杂，因此这里不打算深入地介绍。您只需知道，有 `name` 元素定义的段中包含 ASP.NET 使用的设置（下一节将介绍这些设置）。更详细的信息，请参考 .NET 框架 SDK 文档。

**注意：**请注意给 `web.config` 文件中的每个元素的格式。在由多个单词组成的元素（如 `sectionGroup`）中，第一个单词的首字母是小写的，而其余单词的首字母则为大写的。这被称为 *camel casing*，是 `web.config` 文件使用的标准。`web.config` 文件时区分大小写的，如果大小写不正确，应用程序将出错。

配置设置实际上是配置应用程序的数据：关键字/值对。配置段分两类：`system.net` 和 `system.web`。前者用于配置 .NET 运行阶段本身，因此您不用管它；后者用于控制 ASP.NET 应用程序的所有设置（有两个例外，这将在后面讨论）都被放置在标记 `<system.web>` 内。例如，清单 18.4 列出了一个配置文件的范例。

**清单 18.4 配置文件范例**

```

1: <configuration>
2:   <system.web>
3:     <trace
4:       enabled="false"

```

```

5:         requestLimit="10"
6:         pageOutput="false"
7:         traceMode="SortByTime"
8:         localOnly="true"
9:     />
10:    <globalization
11:        requestEncoding="iso-8859-1"
12:        responseEncoding="iso-8859-1"
13:    />
14:    <customErrors mode="RemoteOnly" />
15:    </system.web>
16: </configuration>

```

❏<!--Possible values for mode:On,Off,RemoteOnly -->

**分析：**该清单包含三个不同的ASP.NET设置。第一个（第3行）包含跟踪ASP.NET服务的设置（将在第20章介绍）；第二个设置（第10行）控制应用程序如何将输出发送给浏览器（将在下一章介绍）；最后一个配置段（customErrors）控制如何在应用程序中显示错误。

**注意：**同样，请注意每个元素的camel casing。

表 18.2 列出了 web.config 中可用的所有配置设置。我不打算在本书中介绍所有的配置，因此其中的一些很少使用，但下表将是非常好的参考资料。

**表 18.2** web.config 配置设置

段 名	描 述
<appSettings>	用于保存自定义的应用程序设置
<authentication>	配置 ASP.NET 如何验证用户的身份（见第 21 章）
<authorization>	配置如何在 ASP.NET 中授权使用资源（见第 21 章）
<browserCaps>	负责控制浏览器功能组件的设置
<compilation>	负责控制 ASP.NET 使用的编译设置。更详细的信息，请参见第 20 章
<customErrors>	告诉 ASP.NET 如何在浏览器中显示错误
<globalization>	负责配置应用程序的全局性设置
<httpHandlers>	负责将输入的 URL 映射为 IHttpHandlers 类。子目录不继承该设置。另外还负责将输入的 URL 映射为 IHttpHandlerFactory 类。<httpHandlerFactories>段中的数据将被子目录继承
<httpModules>	负责配置应用程序中的 Http 模块，Http 模块参与对进入应用程序的每一个请求的处理，常见的用途包括安全性和登录等
<identity>	控制 ASP.NET 如何访问资源（即以何种身份来访问这些资源），参见第 21 章
<location>	一个用于控制设置如何用于应用程序中目录的特殊标记
<pages>	控制页面特定的配置设置

续表

段 名	描 述
<processModel>	配置 IIS Web 服务器系统上的 ASP.NET 处理模型设置
<sessionState>	负责配置会话状态
<trace>	负责配置 ASP.NET 跟踪服务
<webServices>	控制 Web 服务的设置 (请参见第 16 章和第 17 章)

18.3.2 配置段

虽然您可以配置您选择的任何配置段,但 ASP.NET 自带了一些内置的配置,它们可能能够满足您的要求。接下来的几节介绍了常用的配置设置的语法和范例。

每个段都支持子元素、属性。子元素 (subelement) 是当前标记下的独立标记,可以有自己的子元素和属性。属性定义了标记的属性,如服务器控件的 id 属性。

记住,除了两个外 (遇到时,将进行讨论),所有这些设置都必须放置在配置段 <system.web> 中。

1. <appSettings>

该配置段让您能够为 ASP.NET 应用程序定义自定义的属性,如数据库连接字符串 (更详细的信息,请参见第 8 章)。它包含一个子元素: add, 该元素包含两个属性 key 和 value

该配置段专门用于放置自定义值,ASP.NET 根本不会使用这里的值 (当然,除非您命令它使用)。其语法很简单:

```
<appSettings>
  <add key="ApplicationName" value="MyApp" />
  <add key="DSN" value="Provider=Microsoft.Jet.OLEDB.4.0;
  Data Source=H:\ASP.NET\data\banking.mdb" />
</appSettings>
```

访问 <appSettings> 中存储的值的方法与访问其他配置值完全不同,ASP.NET 使得获取这些值更为容易。首先,该标记不必放在 <system.web> 段中,它可以直接位于 <configuration> 元素下。这样,ASP.NET 应用程序和常规 .NET 应用程序可以存取相同的值。

其次,无需使用 GetConfig 方法 (将在本章后面介绍) 来存取这些值,ASP.NET 为该配置段提供了一个更容易的属性: ConfigurationSettings.AppSettings。例如,使用前面的 <appSettings> 段,下面的 Page\_Load 方法将把 “MyApp” 写入到浏览器中。

```
sub Page_Load(obj as Object,e as EventArgs)
  Response.write(ConfigurationSettings.AppSettings _
    ("ApplicationName"))
end sub
```

只需在括号内提供关键字名称,便可以轻松地取得相应的值。

2. <browsersCaps>

该配置段包含许多不同的元素和设置。它描述了客户浏览器的功能,如是否支持 JavaScript、



浏览器版本等。这里不介绍所有的元素和设置。该元素的语法如下：

```
<configuration>
  <system.web>
    <browserCaps>
      browser='Unknown'
      version=0.0
      majorver=0
      minorver=0
      frames=false
      tables=false
      ...
    </browserCaps>
  </system.web>
</configuration>
```

有关每个属性的更详细的信息，请参考.NET 框架 SDK 文档。

### 3. <customErrors>

该配置段让您能够为应用程序定义自定义的错误，或覆盖内置的 ASP.NET 错误。例如，当用户试图访问一个并不存在的页面时，将看到与图 18.5 类似的错误。

虽然该页面足以指出用户所犯的错误，但它可能与您的站点的设计风格不符。因此，<customErrors>让您能够为这种错误（和其他错误）指定其他的页面。

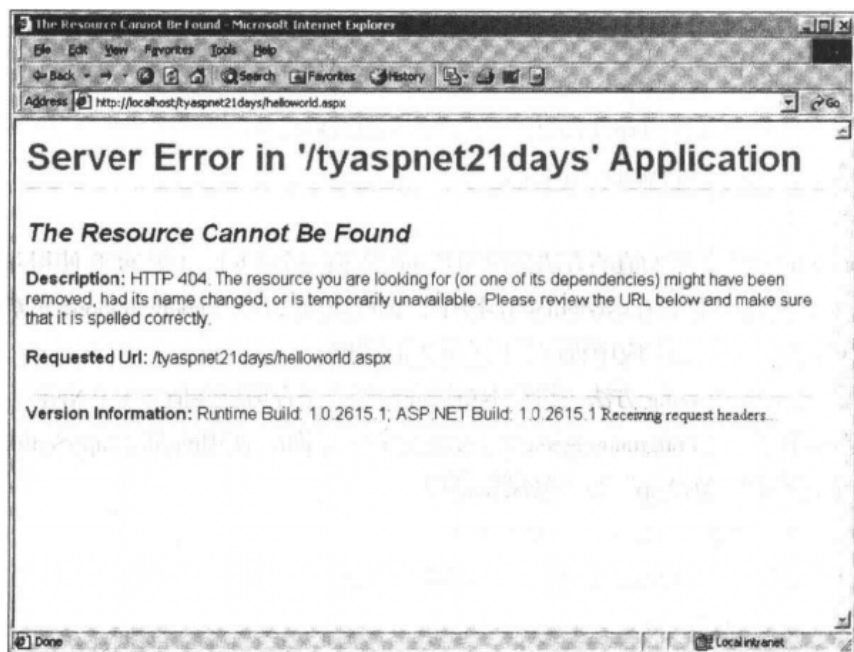


图 18.5 当用户请求服务器上不存在的 helloworld.aspx 时显示的错误页面

<customErrors>有两个属性：defaultRedirect 和 mode。前者指定发生错误时，浏览器应该跳到的 URL；后者指定自定义错误时开（on）、关（off），还是只向远程客户显示（remote）。换句话说，开

发应用程序时，您看不到自定义错误，但最终用户可以看到。

该配置段还支持一个子元素：`error`，该子元素让您能够为分别对待不同的错误。例如，请看下面的代码：

```
<customErrors defaultRedirect="error.htm" mode="on">
  <error statusCode="404" redirect="error404.htm" />
</customErrors>
```

上述代码规定，如果发现 404 错误（找不到资源，如图 18.5 所示），ASP.NET 应该将用户重定向到 `error404.htm`。该文件可以包含与您的设计风格相符的自定义错误消息。如果发生其他错误，则 ASP.NET 应该显示 `error.htm`，同样，该文件也可以其他更为通用的错误消息。

启用自定义错误处理功能后，如果发生错误，ASP.NET 将给重定向页面传递一个查询字符串参数，该参数包含引发错误的文件的虚拟路径。例如，在前面介绍的情况下，当用户请求文件 `helloworld.aspx` 时，用户将被重定向到下述 URL：

```
http://localhost/tyaspnet21days/error404.htm?
asperrorpath=/tyaspnet21days/helloworld.aspx
```

使用上述值的原因很多：您可以告诉用户 URL 无效，并指示它检查地址或更新书签；或者将错误记录到数据库中，帮助以后分析站点存在的问题。

#### 4. <location>

这是另一个不需要放置在 `<system.web>` 段中的配置段。其用途在于，让您能够将其他设置只应用于 Web 服务器的特定位置。它最常用于安全设置，因此将在第 21 章更深入地进行讨论。

想象一种简单的情形。站点包含两个页面：`default.aspx` 和 `members.aspx`。显然，前者是站点的默认页面，因此应该允许任何人访问它。而第二个页面只允许已经注册并登录的用户访问，因此您希望只将安全设置应用于该文件。实现这一目的的代码如下：

```
<configuration>
  <location path="members.aspx">
    <system.web>
      <authorization>
        <deny users="?" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

第 2 行的 `<location>` 标记中包含接下来的设置将应用于的文件的路径，这里是 `member.aspx`（也可以在该属性中指定目录名）。请不要担心属性 `<authorization>` 和 `<deny>`，这将在第 21 章进行介绍。注意，在标记 `<location>` 中，您必须指定 `<system.web>`，`<location>` 本身并不是 ASP.NET 或 .NET 专用的，但其内面的所有标记通常都是。

别忘了，`web.config` 是层次式的，因此标记 `<location>` 也必须遵守这些规则。例如，请看下面的代码：

```
<configuration>
  <location path="subdirectory1">
    <system.web>
```

```

    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
</configuration>

```

上述安全设置不但将被应用于目录 `subdirectory1`，还将被应用于该目录的所有子目录。

#### 5. <pages>

该配置段让您能够控制哪些通常需要使用页面编译指令（如 `@ Page`）指定的页面选项。区别在于，<pages>中的设置将被应用于站点中的所有页面，而不仅仅是某个特定的页面。其语法如下：

```

<configuration>
  <system.web>
    <pages
      buffer="on"
      enableSessionState="true"
      maintainState="true"
      autoEventWireup="true"
      clientScriptsLocation="/scripts"
      inherits="System.Web.UI.Page" />
    </system.web>
  </configuration>

```

上述代码片段包含<pages>元素的所有属性。它启用所有页面的缓冲、会话状态和视图状态（更详细的信息，请参考第 4 章）以及页面事件。它还指定了将在客户端运行的脚本的位置，并告诉 ASP.NET，所有页面都应该从 `System.Web.UI.Page` 类派生而来（这是默认情况）。

如果希望所有页面的行为都类似，则该配置段很有用。但使用时要谨慎，因为默认设置通常足够了，为每个页面修改这些值可能导致性能降低或错误。您通过针对各个页面修改这些设置。

#### 6. <sessionState>

ASP.NET 提供了许多定制会话状态的途径。例如，您可以将会话变量存储在一台完全独立的机器上，以备服务器崩溃或需要替换时的不时之需。这确保会话数据绝不会丢失。您也可以将会话状态存储在同一个系统的另一个进程中，其作用与前面描述的类似。该配置段包含以下属性：

- **mode**：该属性指定会话信息的存储位置。有效的选项有 `off`（禁用会话状态）、`inproc`（将会话信息存储在本地，为默认值）、`stateServer`（会话信息被存储在另一台机器上）和 `sqlServer`（将会话信息存储在一个 SQL Server 数据库中）。
- **cookieless**：指示是否使用 `cookieless` 会话（更详细的信息，请参见第 4 章）。
- **timeout**：空闲多少分钟后，会话将被 ASP.NET 删除（默认为 20 分钟）。
- **connectionString**：当 `mode` 被设置为 `stateServer` 时，指定存储状态信息的服务器名称和端口。
- **sqlConnectionString**：当 `mode` 被设置为 `sqlServer` 时，指定用于连接到 SQL 服务器（以存储会话信息）的连接字符串。

该配置段的语法如下：

```

<configuration>

```

```

<system.web>
  <sessionstate>
    mode='InProc'
    cookieless="false"
    timeout='10' />
  </system.web>
</configuration>

```

出于安全方面的考虑,将状态信息存储在另一台服务器上好像是一个不错的主意。但使用另一台服务器来存储状态信息时,性能将下降;ASP.NET 必须通过网络连接到远程机器,并与之通信,这极大地增加了延迟时间。将信息存储在同一台机器上可能同样安全,但速度要快得多。

### 18.3.3 自定义配置

web.config 并非只能包含表 18.2 中的配置段,您可以将任何类型的信息添加到该文件中。

使用<appSettings>如何?该配置段包含所有的自定义配置信息吗?从技术上说,您可以将所有的自定义信息放置到该配置段中,但只能是关键字/值对。当您使用属性 ConfigurationSettings.AppSettings (见本章前面的<appSettings>一节)时,ASP.NET 将创建一个 NameValueCollection 对象,并使用<appSettings>中的值填充它。可以很容易地使用该集合来引用这个配置段中的关键字/值对。

然而,有时候 NameValueCollection 对象无法满足您的需求。例如,假设应用程序依赖于一个 XML 文件,该文件包含关于图书馆中图书的信息。可以像下面这样很容易地将该文件名加入到<appSettings>配置段中:

```

<appSettings>
  <add key='Library' value='c:\inetpub\wwwroot\tyaspnet21days\
    day18\books.xml' />
</appSettings>

```

但当您检索 web.config 中的该配置段时,获得的不是一个 XmlTextReader,而是一个描述该文件名的关键字/值对。ASP.NET 允许您创建自己的配置段及其对象的段处理程序,这种端段处理程序能够返回任何类型的对象。让我们对此做进一步的讨论。首先,创建一个处理输入的段处理程序,然后在 web.config 中创建配置段以及一个使用该配置段的 ASP.NET 页面。

#### 1. 自定义段处理程序

正如前一节介绍的,有时候内置的 web.config 段处理程序无法满足需要,它们可能没有提供您的应用程序所需的功能。在这种情况下,您可以创建自己的处理程序,来执行所需的操作。让我们来看一个例子,如清单 18.5 所示。

**清单 18.5 自定义 Web.config 段处理程序类**

```

1 Imports System
2 Imports System.Configuration
3 Imports System.Xml
4
5 Namespace TYASPNET.Handlers
6   Public Class LibrarySectionHandler: Implements

```

```

7 IConfigurationSectionHandler
8     Public Function Create(parent as Object, configContext as Object, section as XmlNode)
as Object Implements IConfigurationSectionHandler.Create
9         dim strFileName as String
10        strFileName = section.Attributes.Item(0).Value
11
12        dim objReader as new XmlTextReader(strFileName)
13        return objReader
14    End Function
15 End Class
16
17 End Namespace

```

**分析：**将该文件保存为LibrarySectionHandler.vb。查看代码之前，先简要地介绍一下这个类。web.config文件可以将这个类声明为类似于清单18.3的配置段的处理程序。每当ASP.NET访问该配置段时（通常通过您编写的一些代码），这个类将被创建，以处理包含在该配置段中的数据。在这个例子中，类读取web.config中的文件路径，使用该路径创建一个XmlTextReader，并将其传递给请求者。然后，请求者便可以直接使用该XmlTextReader。

正如您看到的，自定义段处理程序是基于一个VB.NET类的，就像业务对象一样。区别在于，这个类必须实现IConfigurationSectionHandler接口。接口是一个类，它描述了其他类必须包含哪些方法和属性；IConfigurationSectionHandler告诉您，您的LibrarySectionHandler类必须包含一个Create方法。请不要过分纠缠接口，只需将它们看作是其他类的模板即可（更详细的信息，请参考.NET框架SDK文档）。另外，请注意这个类所属的名称空间：TYASPNET.Handlers。

第16行声明了段处理程序类LibrarySectionHandler，这个类负责检索web.config文件中的XML库文件名，并将合适的输出发送给ASP.NET页面。注意，这个类使用了关键字Implements。

接下来第7行声明了方法Create。还记得吗，IConfigurationSectionHandler为这个方法提供了模板，因此只需按模板做即可。该模板告诉您，上述方法包含三个参数。第一个是一个对象，包含关于该web.config段的父文件的信息。换句话说，如果该配置段是从另一个web.config文件继承而来的，则该参数将包含一个到该web.config文件中相应配置段的链接。这里不使用该参数。第二个参数configContext也是一个对象，它包含一些环境方面的信息，通常不会被使用，因此您也将忽略它。最后一个参数section是最重要的，它包含来自web.config文件中的XML数据。您将使用该参数来分析所需的信息，并生成合适的输出。

最后，请注意Create方法声明结尾处的关键字Implement。它告诉ASP.NET，您是想实现IConfigurationSectionHandler接口以及您要实现的是哪一个方法。使用接口时，一定要加上这个关键字，否则将出错。

接下来的部分应该很简单，因为第11章已经介绍过如何在ASP.NET中使用XML。第8~9行检索web.config文件中自定义段中的唯一一个属性，该属性包含XML文件的路径。第11~12行使用该文件路径创建一个XmlTextReader，然后将其返回给调用者。

这项工作看起来好像很可怕，但实际上很简单。该清单是一个样板，您可以将其剪切并粘贴到其他地方，并根据需要修改Create方法的实现。使用该清单，可以创建出大量的自定义段处理程序。

最后，还需要编译这个类，并将其放到组合体仓库中。对于这一过程，您应该非常熟悉。请在命令提示窗口中输入下述命令：

```

vbc /t:library /out:..\bin\TYASPNET.Handlers.dll /r:System.dll
/r:System.xml.dll /r:System.web.dll
/LibrarySectionHandler.vb

```

该命令假设您的组合体仓库（\bin 目录）位于当前目录（文件 LibrarySectionHandler.vb 所在的目录）下。接下来介绍如何使用这个新的处理程序。

## 2. 创建和检索自定义 web.config 配置段

要使用上述自定义的处理程序，首先需要在 web.config 文件中声明一个相应的配置段。这很简单，如清单 18.6 所示。

**清单 18.6 自定义 web.config**

```

1 <configuration>
2   <configSections>
3     <sectionGroup name="system.web">
4       <section name="Library" type="TYASPNET.Handlers.
LibrarySectionHandler,TYASPNET.Handlers" />
5     </sectionGroup>
6   </configSections>
7
8   <system.web>
9     <Library file="c:\inetpub\wwwroot\tyaspnet21days\day18\books.xml" />
10  </system.web>
11 </configuration>

```

**分析：**将该文家保存为 web.config，并将其放在文件 LibrarySectionHandler.vb 所在的目录中。web.config 文件由两部分组成：配置处理程序段和配置设置。前者由元素 <configSections> 表示，如第 2 行所示。第 3 行指出，该处理程序将应用于 <system.web> 段，您可以将其指定为 system.net，甚至可以让它空着。它只是一种给设置分组的方式而已。

第 4 行声明了一个名为 Library 的新段。type 属性指出了与该配置段相关联的处理程序，这里为刚才创建的 LibrarySectionHandler。必须指定其完整的名称空间名以及所属的组合体（TYASPNET.Handlers.dll）。

现在可以为应用程序创建实际的设置。第 9 行完成了配置段 Library 和指定 XML 文件路径的 file 属性（使用的是第 11 章的一个 XML 文件）。完成 web.config 文件后，让我们来创建 ASP.NET 页面，如清单 18.7 所示。

**清单 18.7 检索自定义的配置设置**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Xml" %>
3
4 <script runat="server">
5   sub Page_Load(obj as Object, e as EventArgs)

```

```

6      'retrieve information
7      dim i as integer
8
9      dim objLibrary as XmlTextReader = _
10         ConfigurationSettings.GetConfig("system.web/Library")
11
12      'display information
13      While objLibrary.Read()
14          Select Case objLibrary.NodeType
15              Case XmlNodeType.Element
16                  if objLibrary.HasAttributes then
17                      for i = 0 to objLibrary.AttributeCount - 1
18                          lblLibrary.Text += objLibrary. _
19                              GetAttribute(i) & " "
20                      next
21                      lblLibrary.Text += ", "
22                  end if
23              Case XmlNodeType.Text
24                  lblLibrary.Text += objLibrary.Value & "<br>"
25          End Select
26      End While
27
28      objLibrary.Close
29  end sub
30 </script>
31
32 <html><body>
33     <asp:Label id="lblLibrary" runat="server"/>
34 </body></html>

```

**分析：**该清单中唯一新的东西是第8~9行，让我们首先来分析它们。您使用GetConfig方法来检索web.config文件中的Library段，该方法返回自定义处理程序指定的对象。清单18.5的处理程序返回一个XmlTextReader。注意，Getconfig方法必须指定要检索的配置段的路径。由于Library被嵌入到<system.web>元素中，因此必须以相应的方式调用该方法：

```
Getconfig("System.web/Library")
```

第12~28行分析返回的XmlTextReader，然后在第33行将数据显示到标签中。有关这一过程的更详细的信息，请参考第11章。该清单的输出如图18.6所示。

让我们总结一下。您已经将XML文件的路径放在web.config的<appSettings>段中，然后在ASP.NET页面中使用属性ConfigurationSettings.AppSettings来检索。然后，ASP.NET页面将创建一个XmlTextReader，最后再显示这些数据。

但是,通过创建自定义的段处理程序,ASP.NET 页面无须为 web.config 中的数据以及创建 XmlTextReader 来操心,当您调用 Getconfig 方法时,自定义的处理程序将为您完成这些工作。ASP.NET 所需要做的只是显示数据。您也许可以少做许多工作,这取决于使用该 XML 文件的 ASP.NET 页面的数量。

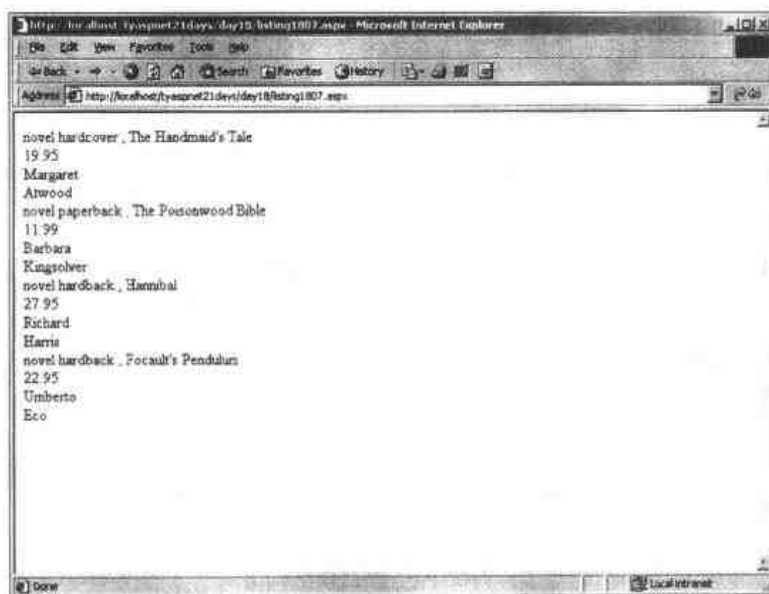


图 18.6 web.config 信息被分析,并作为一个 XmlTextReader 返回给您

## 18.4 部署应用程序

部署 Web 应用程序一度是 ASP 开发中最繁琐、最困难的部分。开发人员必须“亲自”在服务器上配置应用程序,而且使用 REGSVR32.exe 必须手工注册组件或通过微软事务服务器进行注册。这常常带来许多版本管理方面的问题,如 DLL 困境(库版本冲突将导致应用程序无法正常运行,甚至根本不运行)。

在 ASP.NET 中部署应用程序时,只需将所需的文本(ASP.NET 页面、web.config 文件和所需的 DLL)复制到要部署的 Web 服务器上,再也不需要注册组件或调整 Web 服务器的设置。让我们更深入地探讨一下这一神奇的机制。

### 18.4.1 组合体仓库

第 2 章介绍过,组合体是 CLR 中的共享和重用基本单元:命名的类部署单元。通常,组合体是单个 DLL 文件,但也可能包含多个文件(甚至文件的类型也不同)。

ASP.NET 应用程序通过将组合体放在一个全局或本地的组合体仓库中来使用它们。全局仓库存储了 ASP.NET 服务器上所有的应用程序都可以访问的代码。默认情况下,该仓库位于 \WinNT\Assembly 目录下。

本地仓库只供某个特定的应用程序使用,默认情况下,是应用程序根目录下的 bin 目录。该目录被配置成禁止任何客户请求访问,这是一项很有用的特性,可以防止他人窃取您的代码。让我们来看一个 Web 服务器目录结构的例子,如表 18.3 所示。



提示：可以在命令行运行全局仓库工具（gacutil.exe）来查看、添加、删除全局仓库中的组合体。该工具的语法如下：

用法：Gacutil <Option> [<parameters>]

选项：

-i

将组合体安装到全局组合体仓库中，将组合体的文件名作为参数。

范例：-i myDll.dll

-u

卸载全局仓库中的组合体，参数为要卸载的组合体的名称。

范例：-u myDll

-u myDll, ver=1.1.0.0, Loc=en, PK=874e23ab874e23ab

-l

列出全局仓库的内容。

-?

显示帮助文档

使用gacutil -u myDll将删除myDll的所有版本，因此一定要指定必要的信息。

表 18.3 Web 服务器目录结构范例

应用程序的 URL	物理路径
http://www.site.com	C:\inetpub\wwwroot
http://www.site.com:85	C:\inetpub\wwwroot\port85
http://www.site.com/games	D:\www\games
http://www.site.com/games/users	D:\www\users

表 18.3 中的每一个物理目录都是 IIS 中的虚拟目录，因此是一个独立的应用程序。每个应用程序都可以自己的本地不见仓库，它是子目录\bin，例如 www.site.com 的本地仓库为 c:\inetpub\wwwroot\bin。每个 bin 目录都只能供其本地应用程序使用。这让您能够在—一个站点中使用同—一个组合体的多个版本。

#### 18.4.2 影子组合体

通常，当 CLR 将组合体 DLL 装载到内存中时，将锁定对该文件的访问，以防其他应用程序访问或破坏。当引用该文件的应用程序域被释放时，该文件被解锁。这有助于提供稳定性，但增加了替换这些文件的成本。例如，在传统 ASP 中，这些 DLL 将一直被锁定，直到服务器重新启动，这对于管理员和访问者而言，简直是一场噩梦。

ASP.NET 通过不将实际的组合体装载到内存中解决了这种问题。相反，在使用组合体之前，ASP.NET 创建组合体的一个影子（shadowed）副本，并将其锁定和装载到内存中。由于实际的文件并没有被锁定，因此开发任何可以自由地替换它们，不再需要重新启动。

ASP.NET 监视并发现对实际文件所作的任何修改。当它发现文件被修改时，便创建新文件的一个影子副本，将其装载到内存中，并逐渐将请求转到新版本。当所有的请求引用的都是新版本时，

旧的版本将被释放。这对最终用户来说是透明的，并解决了很多令人头痛的问题！

## 18.5 这不是 ASP

本章介绍了许多与传统 ASP 不同的地方。ASP.NET 对配置和部署做了极大的改进，并且比 ASP 更容易实现，因此开发人员需要一段时间来适应这种新系统。

一个完全相同的方面是使用一个文件来控制应用程序事件（至少在实现方面是这样的）。在传统 ASP 中，这个文件是 `global.asa`；而在 ASP.NET 中，这是 `global.asax`。这两个文件是完全兼容的，将应用程序从 ASP 迁移到 ASP.NET 时，只需将 `global.asa` 的内容复制到新文件中，就可以确保应用程序的正常运行。但这些文件之间也有一些差别，例如在 ASP.NET 中，`global.asax` 被编译成一个 .NET 类；而在传统 ASP 中，`global.asa` 中的代码是解释型的，就像 `asp` 页面一样。

而配置系统之间则天壤之别。在 ASP 中，所有的设置都必须通过 IIS 或其他系统应用程序进行配置，这通常要求能够接触到服务器。另外，很难为不同的应用程序或应用程序的不同部分建立不同的设置。

在 ASP.NET 中，所有在 ASP 中可以修改的设置都被放在一个 XML 文件中，这意味着它们易于被修改。只需使用一个文本编辑器，便可以修改设置，然后将其上载到服务器并应用它们即可。

另外，与传统的 ASP 相比，层次式的 ASP.NET 配置系统让您能够更具体地控制应用程序。您可以将设置只应用于特定的文件或目录，控制 ASP.NET 的处理行为，乃至存储会话信息。`Web.config` 是可扩展的，因此您也可以加入自己的设置。

最后，新的部署模式也许是 ASP 开发人员梦寐以求的。不需要安装，也不需要注册组件，同时配置设置可以从一台服务器转移到另一台服务器。所需要做的只是复制合适的文件。应用程序的安装从来也没有这样容易过。

## 18.6 总 结

阅读本章后，您应该能够控制应用程序的每一个方面，从 UI 到会话状态的行为。本章介绍了很多东西，但大部分是以两个概念为中心：文件 `global.asax` 和 `web.config`。

通过本章的学习，您知道应用程序有虚拟目录中的所有文件组成。这包括组合体（DLL 文件）、`.asax` 页面、图像等等。在运行阶段，服务器上的应用程序是以 `HttpApplication` 对象描述的，该对象包含控制应用程序所需的属性和方法。

在 `global.asax` 中可以利用这些属性和方法。表 18.1 列出了您可以为之创建处理程序的应用程序事件。这些处理程序必须使用下面的语法：

```
Application_EventName(obj as Object, e as EventArgs)
```

接着本章介绍了文件 `web.config`。该 XML 文件包含大量的配置段，您可以使用它们来控制应用程序的行为。当这些预定义的配置段不能满足要求时，您知道如何创建自己的配置段以及用来分析配置段并生成合适输出的处理程序。

最后，本章介绍了部署应用程序是如何地简单。本章之所以没有花大量的内容来介绍这一主题，只是因为它很容易理解。毕竟，您只需知道将文件复制到哪里即可！

下一章将介绍如何在不修改任何代码的情况下操纵内容（页面的输出）。这包括本地化和 `code-behind` 表单等主题。

## 18.7 问与答

问：可以使用 Getconfig 方法检索 web.config 中的预定义段（如<sessionState>）吗？

答：不幸的是，这项工作很困难。ASP.NET 用来分析大部分预定义段的处理程序不能返回易于使用的对象。大多数处理程序被声明为私有的，因此无法在 ASP.NET 页面中访问它们。这阻止了大部分对其访问的企图。

更详细的信息，请参考.NET SDK 文档。

问：web.config 的安全设置能够保护所有的资源吗？

答：不能，它只能保护 ASP.NET 处理的资源。因此用户仍然可以访问.gif、.jpg、.txt 和其他文件。这一主题将在第 21 章进行讨论。

## 18.8 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 18.8.1 小测验

1. 判断正误：使用属性 configurationSection.AppSettings 可以访问<appsettings>配置段。
2. 可以在浏览器中查看文件 global.asax 吗？
3. 什么接口必须是自定义的处理程序实现？

下面的问题与表 18.4 的目录结构相关。

表 18.4 关于 Web 服务器目录结构的练习

应用程序的 URI	物理路径
http://www.site.com	C:\www\site
http://www.site.com/sales	C:\www\site\sales
http://www.site.com/hr	C:\www\site\sales\hr
http://www.site.com/users	D:\www\misc\users

4. 虚拟目录 sales 中有一个包含如下设置的 web.config 文件：

```
<httphandlers>
  <add verb="PUT,POST" path="index.aspx" type="System.Web.
    UI.PageHandlerFactory" />
</httphandlers>
```

如果 hr 目录是以下面的方式访问的，则它将继承该设置吗？

http://www.site.com/hr

5. 如果 hr 目录是以下面的方式访问的，则它将继承该设置吗？

http://www.site.com/sales/hr

6. 假设 sales 中的 web.config 文件也包含以下设置:

```
<location path="hr/*.aspx">
  <authorization>
    <deny users="?" />
  </authorization>
</location>
```

这对防止进入 hr 而言, 是一种很充分的安全措施吗? 请解释。

7. 对于上述问题, 提出一个更佳解决方案。

### 18.8.2 练习

1. 为应用程序创建一个 web.config 文件, 完成以下工作:

- 打开应用程序的调试模式;
- 将自定义错误设置为显示页面 errr.aspx;
- 声明一个名叫 authors 的配置段, 其中包含一个关键字 PrimaryAuthor, 其值为您的姓名。

该配置段应使用处理程序 DictionarySectionHandler。

## 第 19 章

# 将内容和代码分开

前 18 章中，介绍了多种将用户界面元素和程序代码（VB.NET 代码）分开的方法。例如，将非 UI 功能放在业务对象中，而 ASP.NET 页面中只保留 UI 代码。另外，在第 12 章介绍过，可以将数据库命令从 ASP.NET 页面中移出，而放在存储过程中。您甚至还可以将设置和变量放在诸如 web.config 等配置文件中。

本章介绍一些更先进的、将源代码和内容分开的方法。换句话说，将控制应用程序的代码与显示内容的代码（如 HTML 和 Web 控件）分开。ASP.NET 开发人员常努力实现上述分离目标，以避免混乱并以更合理的方式将代码分组。毕竟 ASP.NET 页面是用于提供用户界面的，为何要将其他类型的代码放在这些页面中呢？

本章还将介绍根据用户的位置定制 ASP.NET 页面的方法。这些方法让您能够独立于页面的其他代码修改页面的内容。

本章将介绍许多代码，我们开始吧！

本章包含以下内容：

- 何为 code-behind 表单？
- 如果在 ASP.NET 中使用 code-behind 表单？
- 如何在用户控件中使用 code-behind 表单？
- 如何确定用户所处的地理位置（他们使用的语言）？
- 如何检查用户的文化和地域信息？
- 如何将常用的字符串从 ASP.NET 页面中摘录出来，并将它们存储在独立的资源文件中？

### 19.1 为何要将代码和内容分开

第 2 章介绍过，为简化开发人员编写代码的工作，ASP.NET 允许开发人员将 ASP.NET 代码和 HTML 代码分开。例如大多数（如果不是全部的话）ASP.NET 代码应该放在脚本（SCRIPT）块（.aspx 文件的开始位置）中，将其与 HTML 代码分开。图 19.1 说明了这一点。

以这种方式编写代码是有益的，其原因很多：更容易修改代码和 HTML；从更为合理的角度来看待页面；不用在页面中四处查找代码交付块。但 ASP.NET 让您能够做的并不止这些。您可以使用 code-behind 表单和资源文件将代码和页面内容完全分开。code-behind 表单让您能够将代码放在一个单独的文件中，而资源文件让您能够将常用的值（自定义的页面描述）与页面分开，放在一个地方。

```

<VB Page Language = "VB" V>
<VB Import namespace="System.Drawing" V>

<HTML>
<HEAD>

<script language="VB" runat="server">    ASP.NET 代码
' This procedure handles the Change event of the
Sub tbMessage_Change(Sender As Object, I As Even
    lblMessage.Text = "Hello " & tbMessage.Text :
End Sub
</script>

</HEAD>
<BODY logocolor="sffffff">

    <font size="5">Sam's Teach Yourself ASP.NET in 1

    <V Response.Write("Hello" & |
        "World") V>

    <form runat="server">                                HTML 代码

        Please enter your name: <asp:textbox id="tbx
        <asp:button id="lotSubmit" Text="Submit" runat
        <asp:label id="lblMessage" font-size="10pt" :
        <V--
        <asp:label id="lblMessage1" font-size="10pt"
        ..V>

    </form>

```

图 19.1 组织 ASP.NET 页面的最佳方式

## 19.2 Code-behind 表单

回过头来看看图 19.1，便能知道将内容和代码分开的好处。如果能够将页面开始位置的 ASP.NET 代码移到其他地方，将会如何呢？这将真正地将代码和内容分开。ASP.NET 让您能够使用 code-behind 表单实现上述目标。code-behind 表单是一种将所有 ASP.NET 代码和用户界面分开的方法。现在，您可以使用两个文件（如图 19.2 所示），而不是像图 19.1 那样使用一个文件。

让我们再来探讨一下 ASP.NET 的工作原理，以便能够更好地理解 code-behind 模型。

当客户首次请求某个 ASP.NET 页面（.aspx 文件）时，ASP.NET 将分析该页面，并检查其所有组件，如服务器控件，然后基于.aspx 文件动态地创建一个新类。这个类将被编译并执行，以给客户返回 HTML。这个类必须从 System.Web.UIPage 类派生而来，后者为所有的 ASP.NET 页面提供了定义。这一过程是在页面被请求时发生的，并且是不可见的。

但是，.aspx 文件无需直接从 page 类派生而来，只要是 Page 类的后代便万事大吉。因此，可以创建一个从 Page 类派生而来的中间类，然后从该中间类派生.aspx 文件。这个中间类可以包含任何功能，而且这些功能对.aspx 文件来说是可用的。图 19.3 说明了这些类之间的关系。

该中间类位于 code-behind 表单中，它定义了 ASP.NET 可以使用的功能。涉及到的类这么多，可能让人有些困惑。基本上，可以归结为一点，即 ASP.NET 页面必须从 Page 类派生而来，但如何派生则由您决定。

加上 code-behind 类并未提供任何显而易见的好处。code-behind 页面不包含任何固有的功能，因此 ASP.NET 并未获得任何新的东西。同时，对于 ASP.NET 页面的执行也没有任何帮助。但使用 code-behind 页面，可以将代码放在一个中间类中，在 ASP.NET 只留下交付 UI 的 HTML，这使得页面得到了极大的简化。

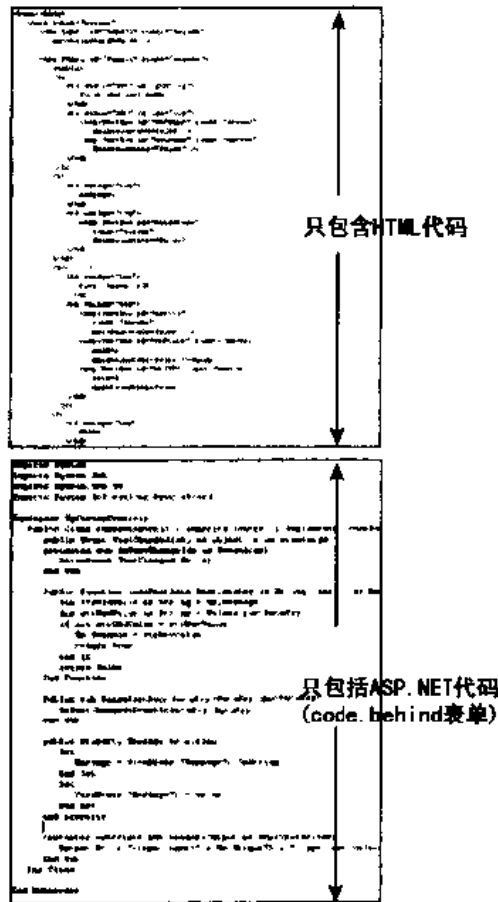


图 19.2 ASP.NET 让您能够使用 code-behind 表单将 UI 和逻辑完成分开

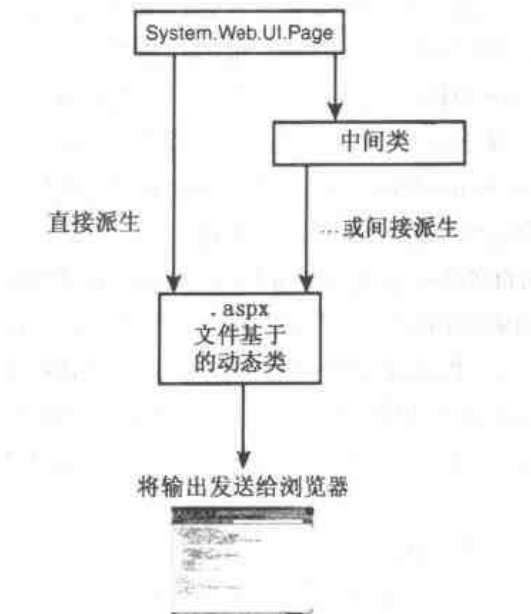


图 19.3 ASP.NET 页面必须直接或间接地从 System.Web.UIPage 类派生而来

## 19.2.1 在 ASP.NET 页面中使用 code-behind 表单

创建 code-behind 表单很简单，这与创建业务对象类似，只是不需要编译。您必须采取一些额外的预防措施，以确保 code-behind 表单能正常运行。清单 19.1 是一个典型的 ASP.NET 页面，它显示数据库中的信息。接下来您将使用该清单来生成一个 code-behind 表单。

清单 19.1 一个包含代码和 HTML 的典型 ASP.NET 页面

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     'declare connection
7     dim strConnString as string = "Provider=" & _
8         "Microsoft.Jet.OLEDB.4.0;" & _
9         "Data Source=C:\ASPNET\data\banking.mdb"
10    dim objConn as new OleDbConnection(strConnString)
11
12    sub Page_Load(obj as Object, e as EventArgs)
13        if Not Page.IsPostBack then
14            FillDataGrid()
15        end if
16    end sub
17
18    private sub FillDataGrid(Optional EditIndex as integer=-1)
19        'open connection
20        dim objCmd as OleDbCommand = new OleDbCommand _
21            ("select * from tblUsers", objConn)
22        dim objReader as OleDbDataReader
23
24        try
25            objCmd.Connection.Open()
26            objReader = objCmd.ExecuteReader
27        catch ex as OleDbException
28            lblMessage.Text = "Error retrieving from the database."
29        end try
30
31        DataGrid1.DataSource = objReader
32        DataGrid1.DataBind()
33
34        objReader.Close

```



```

35     objCmd.Connection.Close()
36 end sub
37 </script>
38
39 <html><body>
40   <form runat="server">
41     <asp:Label id="lblMessage" runat="server" />
42
43     <asp:DataGrid id="DataGrid1" runat="server"
44       BorderColor="black"
45       GridLines="Vertical"
46       cellpadding="4"
47       cellspacing="0"
48       width="100%"
49       Font-Name="Arial"
50       Font-Size="8pt"
51       HeaderStyle-BackColor="#cccc99"
52       ItemStyle-BackColor="#ffffff"
53       AlternatingItemStyle-BackColor="#cccccc"
54       AutoGenerateColumns="True" />
55   </asp:DataGrid><p>
56 </form>
57 </body></html>

```

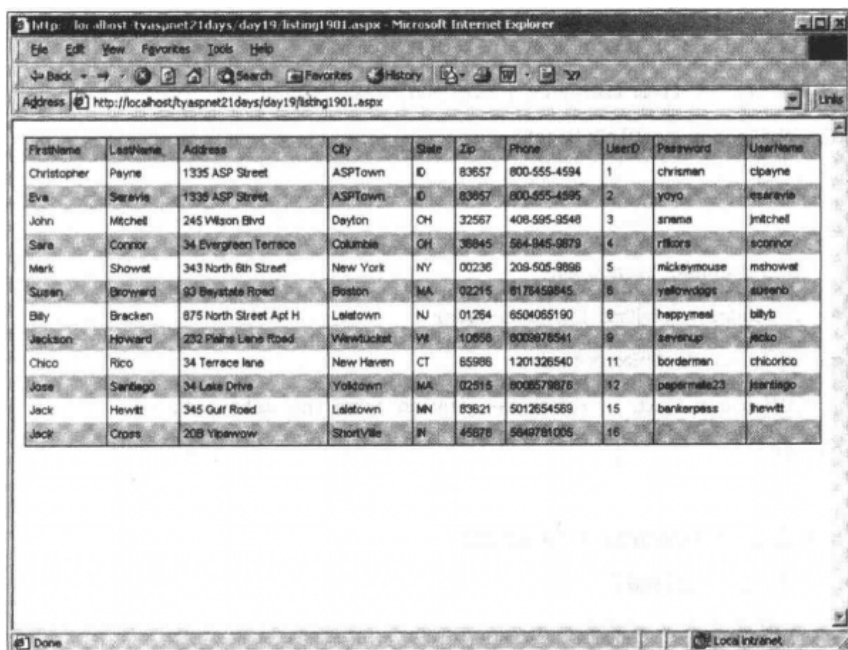


图 19.4 一个简单的数据驱动的 ASP.NET 页面

分析：该清单连接到第8章创建（并在第二部分复习的“附加工程2”中修改的）用户数据库第7~10行创建了一个连接字符串和一个OleDbConnection对象。在Page\_Load事件处理程序中，您调用FillDataGrid方法，将数据库中的信息取出，并将其绑定到DataGrid，如42~50行所示。在浏览器中查看时，该清单的输出如图19.4所示。

现在，以清单 19.1 为基础，来创建一个 code-behind 表单。首先需要创建一个从 System.Web.UI.Page 类派生而来的 VB.NET 类，如下述代码段所示。

```
Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Data
Imports System.Data.OleDb

Public Class CodeBehind1 : Inherits Page
    'Insert code here
End Class
```

这为将脚本块中的代码从清单 19.1 移出打下了基础。还记得吗，在 VB.NET 类中，您必须手工导入本来由 ASP.NET 自动导入的所有名称空间；另外，还必须导入名称空间 System.Data 和 System.Data.OleDb，因为您的 ASP.NET 页面将使用它们来显示数据。

接下来创建需要的所有公有变量。这与我们熟悉的情况稍有不同，这是由 code-behind 表单的特性决定的。让我们做更详细的介绍。code-behind 类将被用来控制 UI（即 Web 表单），它包含处理 UI 事件的逻辑；但 code-behind 类本身不包含任何 UI 元素——UI 包含在.aspx 文件中。这就带来了一个问题：code-behind 类必须控制位于另一个文件中的 UI。如何实现这一目的呢？

现在，您明白了 ASP.NET 页面从 code-behind 类派生而来的原因：如果您在 code-behind 类中声明所有合适的 UI 元素，则.aspx 文件可以继承这些元素。实际上，code-behind 类为所有 UI 元素创建并提供了逻辑，但不显示它们，显示的工作由 ASP.NET 文件负责。图 19.5 说明了这种概念。

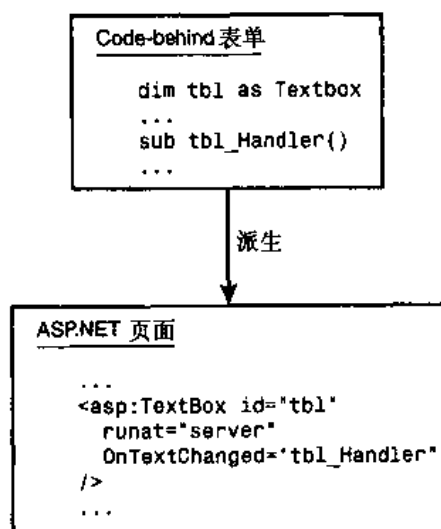


图 19.5 code-behind 类声明了 UI 元素及其逻辑，而 ASP.NET 页面只是显示它们

让我们将清单 19.1 中的代码移到一个 code-behind 表单中，如清单 19.2 所示。

**清单 19.2 包含所有 UI 功能的 code-behind 类**

```

1 Imports System
2 Imports System.Web
3 Imports System.Web.UI
4 Imports System.Web.UI.WebControls
5 Imports System.Data
6 Imports System.Data.OleDb
7
8 Public Class CodeBehind1 : Inherits Page
9     declare public variables for .aspx file to inherit
10    public lblMessage as Label
11    public DataGrid1 as DataGrid
12
13    'declare connection
14    private strConnString as string = "Provider=' & _
15        "Microsoft.Jet.OLEDB.4.0;" & _
16        "Data Source=C:\ASPNET\data\banking.mdb"
17    private objConn as new OleDbConnection(strConnString)
18
19    sub Page_Load(obj as Object, e as EventArgs)
20        if Not Page.IsPostBack then
21            FillDataGrid()
22        end if
23    end sub
24
25    private sub FillDataGrid(Optional EditIndex as integer=-1)
26        'open connection
27        dim objCmd as OleDbCommand = new OleDbCommand _
28            ("select * from tblUsers", objConn)
29        dim objReader as OleDbDataReader
30
31        try
32            objCmd.Connection.Open()
33            objReader = objCmd.ExecuteReader
34        catch ex as OleDbException
35            lblMessage.Text = "Error retrieving from the database."
36        end try
37    end sub

```

```

38     DataGrid1.DataSource = objReader
39     DataGrid1.DataBind()
40
41     objReader.Close
42     objCmd.Connection.Close()
43 end sub
44
45 End Class

```

**分析：**将该文件保存为 codeBehind1.vb。1~6 行导入了所需的名称空间，而第 8 行声明了 code-behind 类，其名称为 codebehind1，是从 Page 类派生（即继承）而来的。

在清单 19.1 中，您使用了两个服务器控件：一个名为 lblMessage 的标签和一个名为 DataGrid1 的 DataGrid。在清单 19.2 中，第 10~11 行将这些控件声明为变量，这样 ASP.NET 页面便可以继承这些控件并使用其功能。该文件的其他内容与清单 19.1 相同，只不过这里不包括 HTML。让我们来看看从该 code-behind 类派生而来的.aspx 文件，如清单 19.3 所示。

#### 清单 19.3 从 code-behind 类派生而来的.aspx 文件

```

1 <%@ Page Inherits="CodeBehind1" src="CodeBehind1.vb" %>
2
3 <html><body>
4     <form runat="server">
5         <asp:Label id="lblMessage" runat="server" />
6
7         <asp:DataGrid id="DataGrid1" runat="server"
8             BorderColor="black" GridLines="Vertical"
9             cellpadding="4" cellspacing="0"
10            width="100%" Font-Name="Arial"
11            Font-Size="8pt"
12            headerStyle-BackColor="#cccc99"
13            ItemStyle BackColor="#ffffff"
14            AlternatingItemStyle-Backcolor="#cccccc"
15            AutoGenerateColumns="True" />
16     </asp:DataGrid><p>
17 </form>
18 </body></html>

```

**分析：**将该文件保存到 code-behind 文件所在的目录中，并将其命名为 listing1903.aspx。除了第 1 行之外，该清单与清单 19.1 的 HTML 部分相同。第 1 行包含一些新的东西：@ Page 编译指令包含属性 Inherits 和 src。Inherits 指定该页面是从哪个类派生而来的，这里为刚才创建的 code-behind 类；src 属性指定了包含 code-behind 类的文件的路径（这意味着您不一定要将 code-behind 文件和.aspx 文件放在同一个目录下）。仅此而已。在浏览器中请求该页面，将看到如图 19.4 所示的输出。

清单 19.3 的服务器控件 DataGrid 和 Label 是清单 19.2 中声明的类变量 CodeBehind1 的实例。因

此, 虽然它们不在清单 19.3 中, 但 `code-behind` 类中引用该这些变量的任何代码都能够访问 Web 表单上的控件实例的属性, 如标签中包含的文本。

让我们来看一个处理表单事件的例子, 这次将从空白开始创建文件, 清单 19.4 是包含 UI 的 ASP.NET 页面。

**注意:** 您仍然可以在 ASP.NET 页面中创建 UI 元素和逻辑, 并没有规定所有的代码都应该放在 `code-behind` 表单中, 而 `code-behind` 类也可以包含 UI 元素。也可以在 `.aspx` 文件中加入代码, 来控制 `code-behind` 类中声明的 UI 元素。但既然创建了 `code-behind` 类, 为何还将代码加入到 ASP.NET 页面中呢?

**清单 19.4** 一个不包含任何 ASP.NET 代码的 `.aspx` 文件

```
1 <%@ Page Inherits="CodeBehind2" src="CodeBehind2.vb" %>
2
3 <html><body>
4   <form runat="server">
5     <asp:Calendar id="Calendar1" runat="server"
6       OnSelectionChanged='DateChanged'
7       Cellpadding="5" Cellspacing="5"
8       DayHeaderStyle-Font-Bold="True"
9       DayNameFormat="Short"
10      Font-Name="Arial" Font-Size="12px"
11      height="250px"
12      NextPrevFormat="ShortMonth"
13      NextPrevStyle-ForeColor='white'
14      SelectedDayStyle-BackColor="#ffcc66"
15      SelectedDayStyle-Font-Bold="True"
16      SelectionMode="DayWeekMonth"
17      SelectorStyle-BackColor="#99ccff"
18      SelectorStyle-ForeColor="navy"
19      SelectorStyle-Font-Size="9px"
20      ShowTitle="true"
21      TitleStyle-BackColor="#ddaa66"
22      TitleStyle-ForeColor='white'
23      TitleStyle-Font-Bold="True"
24      TodayDayStyle-Font-Bold="True" />
25   </form>
26   You selected:
27   <asp:Label id='lblMessage' runat="server"/>
28 </body></html>
```

该清单包含一个日历服务器控件和一个标签。虽然还没有创建 `code-behind` 文件, 但第 1 行已经指定了类和文件名, 因此您知道要创建什么类。第 6 行为日历控件的 `SelectionChanged` 事件提供了

处理程序。同时需要注意这两个控件的名称: calendar1 和 lblMessage。

如果您现在请求该页面,将发生错误,原因有两个:首先,ASP.NET 将查找编译指令@ Page 中的 Inherits 属性中指定的 code-behind 类,但找不到;其次,SelectionChanged 的事件处理程序被指定为 DateChanged,但该文件中并没有一个叫做 DateChanged 的方法。为修复上述问题,让我们来创建 code-behind 类,如清单 19.5 所示。

#### 清单 19.5 控制日历控件的事件

```

1 Imports System
2 Imports System.Web
3 Imports System.Web.UI
4 Imports System.Web.UI.WebControls
5
6 Public Class CodeBehind2 : Inherits Page
7     public lblMessage as Label
8     public Calendar1 as Calendar
9
10    Public sub Page_Load(obj as object, e as eventargs)
11        if not Page.IsPostBack then
12            Calendar1.SelectedDate = DateTime.Now
13            lblMessage.Text = Calendar1.SelectedDate. _
14                ToString("dddd, MMM dd yyyy")
15        end if
16    End Sub
17
18    Public sub DateChanged(obj as object, e as eventargs)
19        if Calendar1.SelectedDates.Count > 1 then
20            lblMessage.Text = Calendar1.SelectedDates(0). _
21                ToString("dddd, MMM dd yyyy") & " through " & _
22                Calendar1.SelectedDates(Calendar1.SelectedDates. _
23                    Count - 1).ToString("dddd, MMM dd yyyy")
24        else
25            lblMessage.Text = Calendar1.SelectedDate. _
26                ToString("dddd, MMM dd yyyy")
27        end if
28    End Sub
29 End Class

```

分析: 将该文件保存为CodeBehind2.vb。第6行将该code-behind类声明为CodeBehind2,这是由前面的ASP.NET页面指定的。第7~8行声明了两个公有变量: lblMessage和Calendar1,这些变量的名称与.aspx文件中相同。第10~16行的Page\_Load事件处理程序将当前的日期设置为选择的日期,并在标签中向用户显示一条消息。

最后，第 18 行的 `DateChanged` 方法是日历控件的 `SelectionChanged` 事件的处理程序，这是在清单 19.4 的第 6 行指定的。该方法以长格式将时间显示给用户。如果选中了多天（由属性 `SelectionDates.Count` 指出），则告诉用户，他选择的是一个时间段（第 19~24 行）；否则，显示被选择的日期。现在，请求该页面时，将得到如图 19.6 所示的结果。

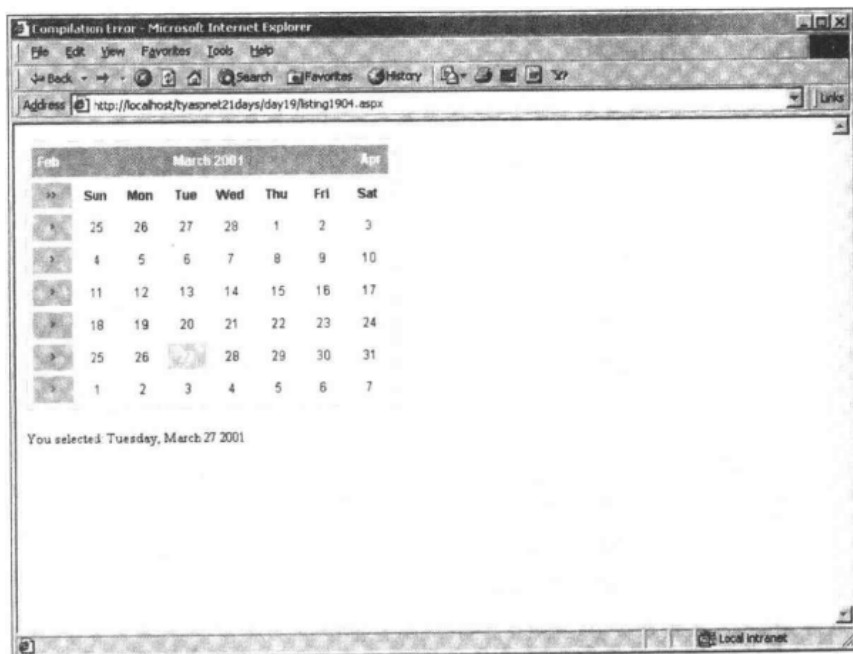


图 19.6 code-behind 表单处理 ASP.NET 页面的事件

这样，您便将 Web 表单的代码与其表示层完全分开了。

### 19.2.2 在用户控件中使用 code-behind 表单

在用户控件中使用 code-behind 表单不同于在 ASP.NET 页面中使用 code-behind 表单。具体地说，您必须继承 `System.Web.UI.UserControl` 类，而不是 `Page` 类（关于用户控件的更详细的信息，请参见第 6 章）。清单 19.6 到 19.8 列出了将第 2 章开发的计算器修改为用户控件和 code-behind 表单的情况。

#### 清单 19.6 用户控件

```
1 <%@ Control Inherits="CalculatorControl" src='Calculator.vb' %>
2
3 Number 1: <asp:textbox id="tbNumber1" runat="server"/><br>
4 Number 2: <asp:textbox id="tbNumber2" runat="server"/><p>
5 <asp:button id="btAdd" runat="server" Text="+"
6   OnClick='btOperator_Click' />
7 <asp:button id="btSubtract" runat="server" Text="-"
8   OnClick='btOperator_Click' />
9 <asp:button id="btMultiply" runat="server" Text="*"
10  OnClick='btOperator_Click' />
```

```

11 <asp:button id="btDivide" runat="server" Text=" / ">
12     OnClick="btOperator_Click" /></asp>
13 The answer is:
14 <asp:label id="lblMessage" runat="server" />

```

**分析：**将该文件保存为Calculator.aspx。这是一个非常简单的用户控件，它只包括显示两个文本框（让用户输入值）、四个按钮（让用户执行算术运算）和一个标签（用于显示结果）的HTML代码。第1行使用的是编译指令`@ Control`，而不是ASP.NET页面中使用的`@ Page`，但语法相同：使用属性`Inherits`和`src`来指定code-behind文件。请注意每个控件的名称，因为您将在code-behind表单中使用它们。清单19.7列出了该用户控件要使用的code-behind文件。

清单 19.7 code-behind 类

```

1 Imports System
2 Imports System.Web
3 Imports System.Web.UI
4 Imports System.Web.UI.WebControls
5
6 Public Class CalculatorControl : Inherits UserControl
7     public lblMessage as Label
8     public btAdd as Button
9     public btSubtract as Button
10    public btMultiply as Button
11    public btDivide as Button
12    public tbNumber1 as TextBox
13    public tbNumber2 as TextBox
14
15    Sub btOperator_Click(obj as object, e as EventArgs)
16        lblMessage.Text = Operate(CType(obj, Button).Text, _
17            tbNumber1.Text, tbNumber2.Text).ToString
18    End Sub
19
20    private function Operate(operator as string, number1 as string, optional number2 as string _
    - '_' as double) as double
21        select case operator
22            case "+"
23                Operate = CDbl(number1) + CDbl(number2)
24            case "-"
25                Operate = CDbl(number1) - CDbl(number2)
26            case "*"
27                Operate = CDbl(number1) * CDbl(number2)
28            case "/"

```



```

29         Operate = Cdbl(number1) / Cdbl(number2)
30     end select
31 end function
32 End Class

```

分析: 将该文件保存为 Calculator.vb。该清单的功能要比清单 19.5 复杂, 但格式是相同的。第 1~4 行导入了所需的名称空间。第 6 行声明了类 CalculatorControl, 它是从类 UserControl (而不是 Page) 派生而来的。然后, 7~13 行声明了与用户控件中的服务器控件对应的公有变量。

但用户单击用户控件中的按钮时, 将执行事件处理程序 htOperator\_Click。它调用 Operate 方法, 该方法接受三个参数: 要执行的运算以及两个参数数。当页面被请求时, 引发该事件的按钮将由 15 行的 obj 变量表示。我们来看看第 16 行的 Operate 方法的第一个参数:

```
CType(obj, Button).Text
```

该参数检索引发事件的按钮的 Text 属性 (例如, 如果用户单击的是 Add 按钮, 则值为 +)。如果是在 ASP.NET 页面中, 则可以使用 obj.Text 来检索这个值。但 VB.NET 类的行为与 ASP.NET 页面不同, 具体地说, 它不允许晚期联编。

**新术语:** 晚期联编 (Late binding) 指的是 Object 类型的变量不能等到运行阶段再进行处理。在运行阶段, 可以使用这种变量了代表任何类型的对象, 这就是即使 Object 数据类型没有 Text 属性, 也可以使用 obj.Text 的原因所在。ASP.NET 不会回避这一点, 因为当页面被请求时, obj 变量可能是一个按钮 (如果不是, 则可能是您犯了错误)。

不使用晚期联编时, Object 数据类型将被立刻处理, 当编译器发现您试图访问一个不存在的属性 (Object 没有 Text 属性) 时, 将引发错误。因此, 您必须使 VB.NET 认为 obj 确实是一个按钮 (即使是在页面被请求之前), 以便能够在代码中访问 Text 属性。CType 方法将一种数据类型强制转换为另一种, 而 16 行将一个 Object 类型的变量强制转换为一个按钮变量。这好像是一个令人费解的解决方案, 但当您开发更多的范例后, 便会了解晚期联编, 并对这种解决方案更为明白。

第 20~31 行的 Operate 方法使用 Select case 语句分析 htOperator\_Click 方法传递给它的运算符, 然后根据运算符对两个操作数执行不同的运算。结果将被返回给 htOperate\_Click, 并显示给用户。

最后来看一个简短的、实现该用户控件的 ASP.NET 页面, 如清单 19.8 所示。

#### 清单 19.8 实现用户控件和 code-behind 表单的 ASP.NET 页面

```

1 <%@ Register TagPrefix="TYASPNET" TagName="Calculator" src="Calculator.ascx" %>
2
3 <html><body>
4     <form runat="server">
5         <TYASPNET:Calculator id="Calc1" runat="server"/>
6     </form>
7 </body></html>

```

分析: 该页面很简单。第 6 章介绍过, 您必须使用编译指令 @Register 来注册页面上的用户控件, 然后便可以像使用其他服务器控件一样来使用该控件 (如第 5 行所示)。在浏览器中查看该页面, 尝试输入一些值并执行某些运算。该页面的输出与图 19.7 类似。

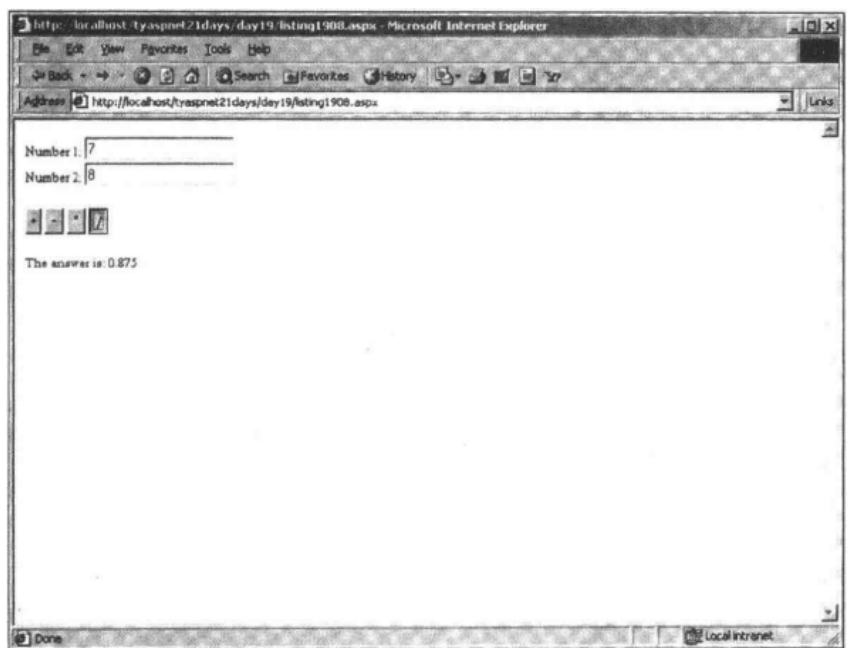


图 19.7 code-behind 表单处理 ASP.NET 页面上的用户控件的事件

## 19.3 资源文件和本地化

度假时，皮箱里装着衣服、化妆品以及其他个人用品。您可以将它们装在兜里，而不是皮箱中，但这太笨拙了（更不用说您的口袋有限）。对于旅行而言，皮箱是一个宝贵的资源，没有它，您的假期将是一场灾难。

ASP.NET 允许您将应用程序打包，可以这么说。任何自定义信息（如变量、消息、内容简介和图像等）都可以放在一个资源文件中。这样，页面便可以在需要时很容易地访问这些文件中的资源。

资源文件通常用于本地化应用程序。本地化指的是修改应用程序的输出，使之与不同的文化和语言相适应。可以为将使用您的应用程序的每一种文化创建一个资源文件。例如，您可以将欢迎消息存储在资源文件中，讲英语的人访问该页面时，显示“Hello!”；讲法语的人访问该页面时，则显示“Bonjour!”。通过将每种语言的消息存储在不同的资源文件中，无需为每种语言创建一个新的页面，而只需创建一个新的资源文件，这简单得多。

ASP.NET 使得本地化和资源归类非常简单。您不用修改任何代码，便可以修改输出内容，因此同样将内容和代码分开了。在接下来的两节中，将介绍如何确定谁将访问您的页面，并根据用户的情况调整页面，以及如何将资源包装到文件中。

### 19.3.1 应用程序的本地化

最理想的情况是，本地化 Web 应用程序时，您希望能够自动确定用户的地理位置，以便能够相应地设置输出语言和其他与文化相关的信息。不幸的是，由于 Web 的特性，并不存在能够轻松确定用户所在地理位置的标准方法。

所幸的是，还是有一些办法的。您可以通过浏览器确定访问者使用的语言。通常，浏览器指示了用户最为熟悉的语言和文化（虽然并非总是如此），并做相应的调整。这些信息是由对象 `Request.Languages` 提供的，该属性收集浏览器使用的语言，并根据用户设置的优先级对语言进行分

类。清单 19.9 是一个检测用户使用的首选语言的范例。

清单 19.9 使用 Request 对象确定用户的语言偏好

```
1 <%@Page Language="VB" %>
2 <%@Import Namespace= System.Globalization" %>
3
4
5 <script runat= server">
6     sub Page_Load(obj as object,e as EventArgs)
7         lblMessage.Text =Request.UserLanguages(0).ToString
8     end sub
9 </script>
10
11 <html><body>
12     Your primary language is:
13     <asp:Label id="lblMessage" runat="server"/>
14 </body></html>
```

分析：第2行导入了一个新的名称空间System.Globalization，它包含许多根据不同文化而修改输出时需要的对象。注意到该清单并未使用该名称空间中的任何对象，但您马上就要使用它们。第6行检索并显示用户使用的首选语言。UserLanguage属性返回一个由用户使用的语言（这是由其浏览器指定的）组成的数组，并按优先级进行排序（常用语言及其编码见表19.1）。UserLanguage(0)返回的是默认语言——优先级最高的语言。在浏览器中查看清单19.9时，将得到如图19.8所示的结果。



图 19.8 可以使用 Request 对象来确定用户使用的首选语言

表 19.1 常用的语言及其缩写

语 言	编 码
亚拉伯语 (埃及)	ar-eg
中文 (香港)	zh-hk
荷兰语 (比利时)	Nl-be
英语 (澳大利亚)	En-au
英语 (加拿大)	En-ca
英语 (英国)	En-gb
英语 (美国)	En-us
法语 (法国)	fr
德语 (德国)	De
意大利语 (意大利)	It
日语 (日本)	Ja
朝鲜语	Ko
葡萄牙语 (葡萄牙)	Pt
俄语	Ru
西班牙语 (墨西哥)	Es-mx
西班牙语 (传统)	es

知道用户使用的首选语言后，便可以据此来设置 ASP.NET 页面的文化信息。这是使用对象 `System.Globalization.CultureInfo` 来实现的，该对象包含了能够完整描述语言所代表的文化方面的信息，如使用的历法、国家的名称、日期和时间格式等等。清单 19.10 是一个使用该对象的范例。

清单 19.10 使用 `CultureInfo` 对象

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Globalization" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object, e as eventargs)
6         dim strLanguage as string = Request.UserLanguages(0).ToString
7
8         lblMessage.Text = "Primary language: " & strLanguage & "<br>"
9
10        dim objCulture as new CultureInfo(strLanguage)
11        lblMessage.Text += "Full name: " & objCulture._

```

```

12     EnglishName & "<br>"
13     lblMessage.Text += "Native name: ' & objCulture._
14     NativeName & "<br>"
15     lblMessage.Text += "Abbreviation: " & objCulture._
16     ThreeLetterISOLanguageName & "<br>"
17     lblMessage.Text += "Current Time: " & DateTime.Now._
18     ToString("D", objCulture) & "<br>"
19 end sub
20 </script>
21
22 <html><body>
23     <b>Your user information:</b> <p>
24     <asp:Label id="lblMessage" runat="server"/>
25 </body></html>

```

分析：该清单是在前一个清单的基础上修改后得到的。第2行导入了名称空间 System.Globalization。第6~7行像清单19.9一样，检索访问者使用的首选语言，并在26行显示出来。第12行根据第6行确定的首选语言，创建了一个新的 CultureInfo 对象。第13~20行显示了这种文化的各项属性，包括名称、缩写以及根据该文化格式化的日期。为正确地格式化日期，您使用了方法 ToString，该方法接受两个参数。第一个参数指示应该如何显示日期，而第二个参数是一个 CultureInfo 对象，它提供了自定义信息。图19.9时在首选语言为法语的浏览器中请求该页面时得到的结果。

CultureInfo 对象包含大量的属性，让您能够确定更详细的信息。表 19.2 列出了其中最常用的属性。

表 19.2 CultureInfo 对象的属性

属 性	描 述
Calendar	这种文化使用的历法（罗马历法、朝鲜历法等）
CurrentCulture	检索服务器（而不是用户的浏览器）使用的文化。返回一个 CultureInfo 对象，该属性是只读的
CurrentUICulture	检索服务器使用的文化。该属性用于指定该文化对应的资源文件，参见本章后面的“将资源包装到文件中”一节。该属性是只读的
DateTimeFormat	返回一个 DateTimeFormatInfo 对象，该对象指定了如何格式化日期和时间的信息。更详细的信息，请参考.NET SDK 文档
DisplayName	以 UI 使用的语言显示 CultureInfo 对象的全名
EnglishName	英语中，该 CultureInfo 对象的全名
Name	同 DisplayName 相同，但是只读的
NativeName	本国语言中，CultureInfo 对象的全名
NumberFormat	描述如何显示数字（即逗号、小数点等的位置），返回一个 NumberFormat 对象

续表

属 性	描 述
ThreeLetterISOLanguageName	在 ISO 3166 标准中, 文化对应的三个字母的编码
ThreeLetterWindowsLanguageName	文化对应的 Windows 版本的三字母编码
TwoLetterISOLanguageName	在 ISO 3166 标准中, 文化对应的两个字母的编码

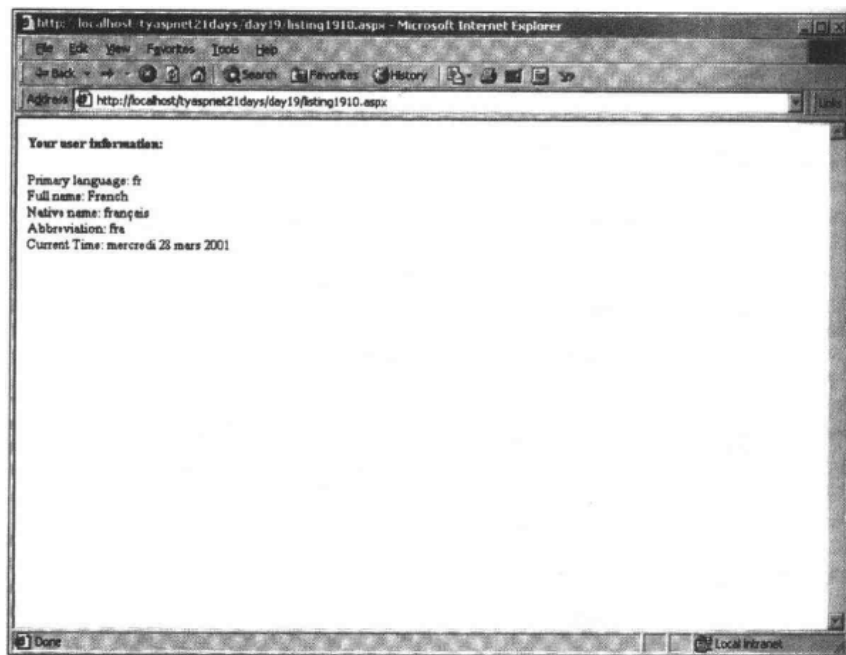


图 19.9 无需修改任何代码, 便可以根据用户使用的文化定制页面的输出

也可以使用编译指令 `@ Page` 的 `Culture` 属性为页面设置文化信息。下面的代码行将文化设置为德语 (de):

```
<%@ Page Language="VB" Culture="de" %>
```

每当您在 ASP.NET 页面中使用文化-特定的对象时, 都将把上述值 (而不是服务器指定的值) 作为默认值。例如, 当文化被设置为德语时, 代码 `DateTime.Now.ToString("D")` 将输出以下结果:

```
Mittwoch, 28. März 2001
```

当文化被设置为 en-us 时, 则上述代码的输出如下:

```
Wednesday, March 28, 2001
```

如果您预先知道要使用的是哪种文化, 则这是一种很容易的、在页面级设置文化的方法。

**注意:** `CultureInfo.CurrentCulture` 属性实际上是 `System.Threading.Thread.CurrentCulture` 对象的一种简捷方式。它们都返回一个 `CultureInfo` 对象, 可用于完成上述操作。区别在于, 前者是只读的, 而后者不是。因此需要修改当前的文化时, 必须使用后者。

文化是当前运行的线程（即当前应用程序中正在运行的部分）的一个属性。设置文化信息时，ASP.NET 针对的是整个应用程序。因此可以访问当前线程的文化信息，也可以访问当前 `CultureInfo` 对象的文化信息（本章后面将使用对象 `System.Threading.CurrentThread`）。

除了 `CultureInfo` 对象外，还可以使用 `RegionInfo` 对象，该对象提供了诸如货币符号以及该地区是否使用公制等信息。使用该对象的语法与 `CultureInfo` 类似，但它不使用语言来描述定制信息，而是使用国家名的缩写来描述。例如，美国为 `US`，而法国为 `FR`。清单 19.11 列出了一个范例。

**清单 19.11 显示关于用户所在地区的信息**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Globalization" %>
3
4 <script runat="server">
5     sub Page_Load(obj as object, e as EventArgs)
6         dim objRegion as RegionInfo
7
8         if Page.IsPostBack then
9             objRegion = new RegionInfo(txtRegion.Text)
10        else
11            objRegion = RegionInfo.CurrentRegion
12        end if
13
14        lblMessage.Text = "Region: " & objRegion.Name & "<br>"
15        lblMessage.Text += "Full name: " & objRegion._
16        EnglishName & "<br>"
17        lblMessage.Text += "Currency: " & objRegion._
18        CurrencySymbol & "<br>"
19        lblMessage.Text += "ISO Currency: " & objRegion._
20        ISOCurrencySymbol & "<br>"
21        lblMessage.Text += "Abbreviation: " & objRegion._
22        ThreeLetterISORegionName & "<br>"
23        lblMessage.Text += "Metric system: " & objRegion._
24        IsMetric
25    end sub
26 </script>
27
28 <html><body>
29     <form runat="server">
30         <b>Your information:</b> <p>
31         <asp:Label id="lblMessage" runat="server"/><p>
32         Change to (i.e. 'US', 'FR', 'JP', etc):

```

```

33     <asp:TextBox id="btRegion" runat="server"
34         AutoPostBack=true />
35 </form>
36 </body></html>

```

分析：当该页面首次被装载时，将使用CurrentRegion属性将默认区域装载到一个RegionInfo对象中，如第11行所示。14~22行显示区域的属性，如货币符号和缩写。当用户在如31行所示的文本框中输入信息时，页面将自动被发送，因为AutoPostBack属性被设置为True。然后，用户输入的区域编码被用来创建一个新的RegionInfo对象，如第9行所示。图19.10是区域为Jp（日本）时的输出。

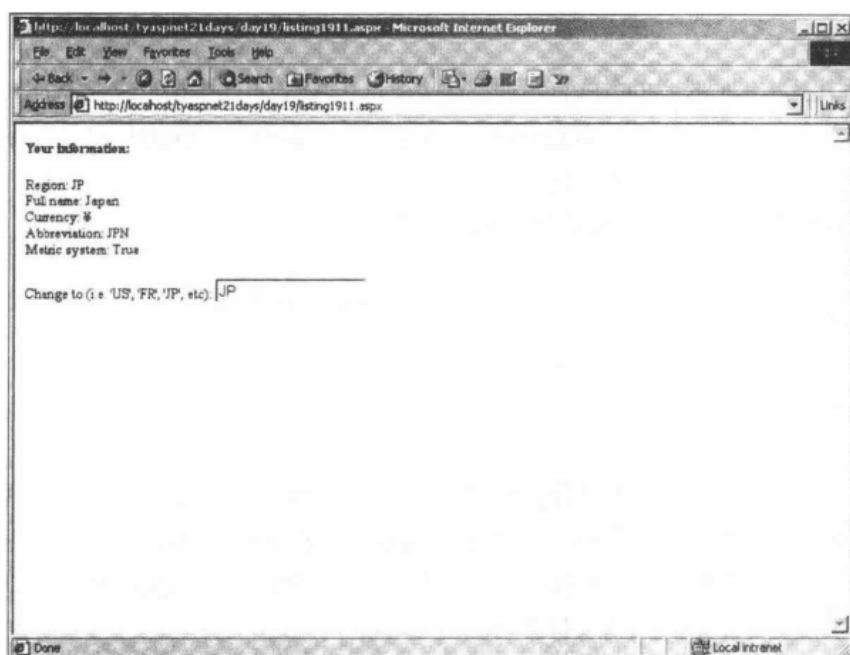


图 19.10 显示指定区域的信息

表 19.3 列出了 RegionInfo 对象的常用属性。

表 19.3 RegionInfo 对象的属性

属 性	描 述
CurrencySymbol	在指定区域中用于表示金额的符号
CurrentRegion	检索服务器（而不是用户）使用的区域，返回一个 RegionInfo 对象
DisplayName	UI 使用的语言表示的 RegionInfo 对象的全名
EnglishName	以英语表示的 RegionInfo 对象的全名
IsMetric	一个布尔值，指示当前区域是否使用公制
ISOCurrencySymbol	表示 CurrencySymbol 的 ISO 编码，例如的 ISO 编码为 USD
Name	与 DisplayName 相同，只不过是只读的



续表

属 性	描 述
ThreeLetterISORegionName	ISO 3166 标准中区域的三字母编码
ThreeLetterWindowRegionName	Window 版本的三字母区域编码
TwoLetterISORegionName	ISO 3166 标准中区域的两字母编码

**新术语：**最后，可以指定ASP.NET页面的输出使用的编码技术（encoding）。编码技术指的是计算机如何表示字符，例如Unicode或ASCII。默认情况下，ASP.NET使用Unicode，但编码方式随区域而异。

为应用程序指定编码技术的方式有两种：直接在 ASP.NET 页面中指定或在 Web.config 文件中指定。在 ASP.NET 页面中，只需使用编译指令@ Page 的 ResponseEncoding 属性指定即可：

```
<%@ Page Language="VB" ResponseEncoding="UTF-8" %>
```

在 web.config 文件中，指定编码技术的语法如下：

```
<configuration>
  <system.web>
    <globalization fileEncoding="utf-8" />
  </system.web>
</configuration>
```

至此，您已经了解了如何修改文化特定对象（如 DateTime 对象）的输出，那么如何修改纯文本呢？如何将纯文本转换为文化特定的语言呢？不幸的是，并没有简单的方法，必须手工翻译这些文本。当站点用户不都是使用同一种语言时，通常每一个页面都需要创建多个版本。幸运的是，在 ASP.NET 中不用这样做，使用资源文件即可。

### 19.3.2 将资源包装到文件中

资源文件用于存储应用程序数据，使之与应用程序本身分开。对于同一个资源文件，可以有多个版本，这样 ASP.NET 页面便可以显示不同的信息，而无需修改任何代码。以本地化为例，可以有多个资源文件：每个文件对应一种文化。每个文件都包含相同的信息，但这些信息被翻译为不同的语言。

让我们来看一个简单的 ASP.NET 页面，它包含能够被摘录到资源文件中的内容，如清单 19.12 所示。

**清单 19.12** 一个被用来提取资源的 ASP.NET 页面

```
1 <%@ Page Language="VB" %>
2
3 <script runat="server">
4   sub Page_Load(obj as Object, e as EventArgs)
5     lblMessage.Text = DateTime.Now.ToString("t")
6   end sub
7 </script>
```

```

8
9 <html><body>
10   <b>Welcome!</b> The time is now:
11   <asp:Label id="lblMessage" runat="server"/><p>
12
13   This page demonstrates using resource files with ASP.NET.<p>
14
15   <font size=1>Don't forget to try this at home!</font>
16 </body></html>

```

该清单并不复杂，它以 HTML 格式显示一些静态文本，并在 11 行定义的标签上显示当前的时间。图 19.11 为该页面的输出。

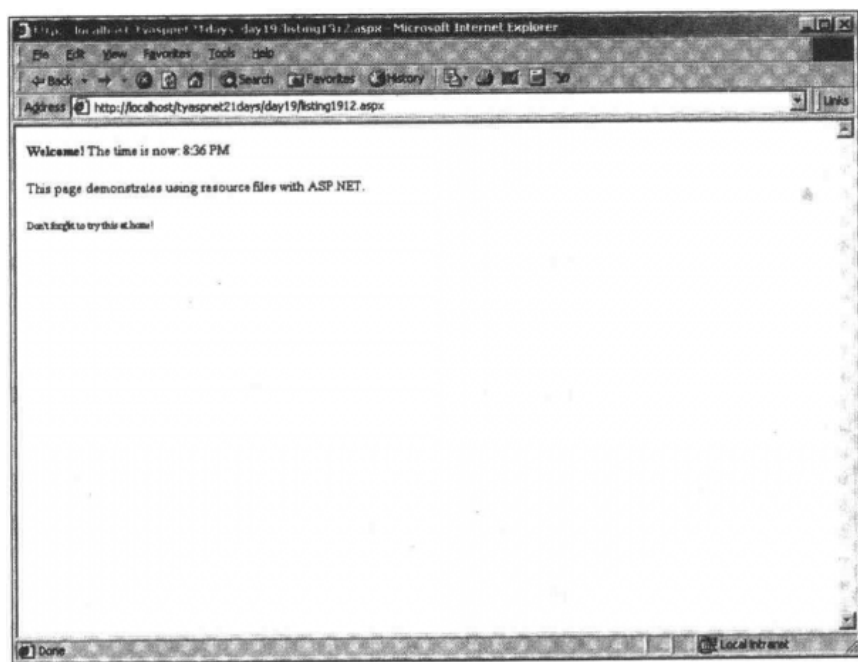


图 19.11 一个简单的、可用于制作资源文件的 ASP.NET 页面

假设您想将该页面翻译为法语，您可以新建一个格式相同、但文本内容不同的文件，但为何要如此麻烦呢？您可以创建两个独立的资源文件，它们是纯文本文件，可以使用任何文本编辑器进行创建。我们首先创建英文版的资源文件，在文本编辑器中输入以下代码：

```

[strings]
Greeting=Welcome!
Time=The time is:
Blurb=This page demonstrates using resource files with ASP.NET
Disclaimer=<font size=1>Don't forget to try this at home!
</font>

```

将该文件保存为 `data.en-us.txt`（后面将解释这样命名的原因）。该文件只包括一段：[strings]，其中包含了所有字符串值。这些信息被组织为关键字/值对，在下述代码行中：

```
Greeting=Welcome!
```

Greeting 是在 ASP.NET 页面中要引用的资源的名称，而 Welcome! 是将被显示给用户的实际值。法语版的资源文件如下：

```
[strings]
Greeting=Bonjour!
Time=L'heure maintenant est:
Blurb=Cette page demonstrate utiliser files de resource avec ASP.NET
Disclaimer=<font size=1>N'oubliez pas essayez de faire ceci chez
soi!</font>
```

将该文件保存为 data.fr.txt（每个字符串值都位于一行中，如果字符串中间有换行符，将发生错误）。注意到关键字名称没变。在不同版本的资源文件中，关键字名称必须一致，否则 ASP.NET 页面将找不到资源。

然而，ASP.NET 不能直接使用这些文本文件，您必须使用资源生成器工具（resgen.exe）将它们转换成 ASP.NET 能够理解的格式。该工具转换文件的格式，生成 resources 文件。请打开一个命令提示窗口，并切换到保存这些.txt 文件的目录下，然后在窗口中输入下面的命令（输入每个命令后，别忘了按 Enter 键）。

```
resgen data.en-us.txt
resgen data.fr.txt
```

执行每个命令后，您将看到与下面类似的信息：

```
Read in 4 resource from 'data.en-us.txt'
Writing resource file. Done.
```

现在，当前目录下将包含两个新文件：data.en-us.resources 和 data.fr.resources。ASP.NET 的资源管理器能够识别这些文件，让我们在页面中使用它们。

System.Resource.ResourceManager 对象负责为 ASP.NET 处理所有的资源文件，该对象找到与用户使用的语言对应的资源文件，并装载该文件中的所有资源。清单 19.13 说明了该对象的用法。

**清单 19.13 使用 ResourceManager 装载资源文件**

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Globalization" %>
3 <%@ Import namespace="System.Resources" %>
4 <%@ Import namespace="System.Threading" %>
5
6 <script runat="server">
7     sub Page_Load(obj as object, e as eventargs)
8         dim objRM as ResourceManager
9         dim strLanguage as string = Request.UserLanguages(0). _
10             ToString
11         dim objCulture as new CultureInfo(strLanguage)
12         Thread.CurrentThread.CurrentCulture = new _
13             CultureInfo(strLanguage)
14         Thread.CurrentThread.CurrentUICulture = new _
```

```

15     CultureInfo(strLanguage)
16
17     objRM = ResourceManager._
18     CreateFileBasedResourceManager("data", _
19     Server.MapPath("."), Nothing)
20
21     lblGreeting.Text = objRM.GetString("Greeting")
22     lblTime.Text = objRM.GetString("Time") & " " & _
23     DateTime.Now.ToString("t")
24     lblBlurb.Text = objRM.GetString("Blurb")
25     lblDisclaimer.Text = objRM.GetString("Disclaimer")
26
27     objRM.ReleaseAllResources
28 end sub
29 </script>
30
31 <html><body>
32     <b><asp:Label id="lblGreeting" runat="server"/></b>
33     <asp:Label id="lblTime" runat="server"/><p>
34
35     <asp:Label id="lblBlurb" runat="server"/><p>
36
37     <asp:Label id="lblDisclaimer" runat="server"/>
38 </body></html>

```

**分析：**第3~4行导入了另外两个名称空间。使用ResourceManager对象需要名称空间System.Resources；同时也需要导入名称空间System.Threading（稍后将解释其用途）。第8行声明了一个ResourceManager对象，其声明方法与其他对象相同。第9~10行通过浏览器检索访问者使用的首选语言，就像清单19.9所做的一样。

第12~15行根据用户使用的首选语言设置了页面的文化信息，前面的“应用程序的本地化”一节中介绍过，可以使用属性System.Threading.CurrentThread.CurrentCulture来设置文化信息，该属性接受并返回一个CultureInfo对象。第12行根据用户使用的首选语言将CurrentCulture属性设置为合适的文化。这样所有可本地化的对象都将使用您指定的文化（例如，在法文中时间将显示为22:06，而不是10:06 PM）。

但仅修改文化，并不能使ASP.NET从不同的资源文件中检索资源，还必须将属性System.Threading.CurrentThread.CurrentUICulture设置为新的文化，如第14~15行所示。ASP.NET使用该属性来确定应装载哪个资源文件。

第17~19行使用方法CreateFileBasedResourceManager实际装载资源文件。该方法接受三个参数：一个用作资源文件的前缀、资源文件的路径和一个用于分析资源的可选对象。在第二个参数中，您使用方法Server.MapPath来返回资源文件所在目录的物理路径（有关MapPath方法的更详细的信

息, 请参考第4章的“HttpServerUtility 对象”一节)。您并不要使用另一个对象来分析资源文件, 因此第三个参数为 `nothing`。

还记得吗, 您使用下面的格式来命令每一个资源文件: `data.culture.resources`。方法 `CreateFileBasedResourceManager` 搜索格式为 `prefix.culture.resources` 的文件, 因此在该方法的第一个参数中指定的前缀必须是资源文件的前缀。在这个例子中, 当文化为 `en-us` (美国英语) 时, 该方法将搜索文件 `data.en-us.resources`。同样, 如果文化为 `fr` (法语), 则该方法将搜索文件 `data.fr.resources`。前缀是一种按逻辑将资源文件分组的方式。

最后, 第 21 ~ 25 行使用 `ResourceManager` 的 `GetString` 方法来检索资源文件中的关键字/值对。其中的每个值都被显示在标签中, 如 30 ~ 35 行所示。打开资源文件时, 资源管理器将锁定该文件, 以防使用期间其他应用程序修改该文件。第 27 行调用 `ReleaseAllResource` 来解除对这些文件的锁定。图 19.12 和 19.13 分别是文化为 `en-us` 和 `fr-FR` 时, 该清单的输出。

根本不需要修改代码, 将根据用户的文化, 显示不同的内容。使用资源文件类似于将定制信息存储在数据库中, 并根据用户的偏好检索特定的数据。

**注意:** 也可以在编译指令 `@ Page` 中设置文化:

```
<%@ Page Language="VB" Culture="fr-FR" %>
```

如果您这样做, 则清单 19.13 中的代码将使用不正确的资源文件。这是因为第 9 行使用用户浏览器中的值, 而不是编译指令 `@ Page` 中的值, 来设置文化。为修复这种问题, 请将第 9 ~ 11 行改成如下的代码行:

```
dim strLanguage as string = CultureInfo.CurrentCulture.ToString
```

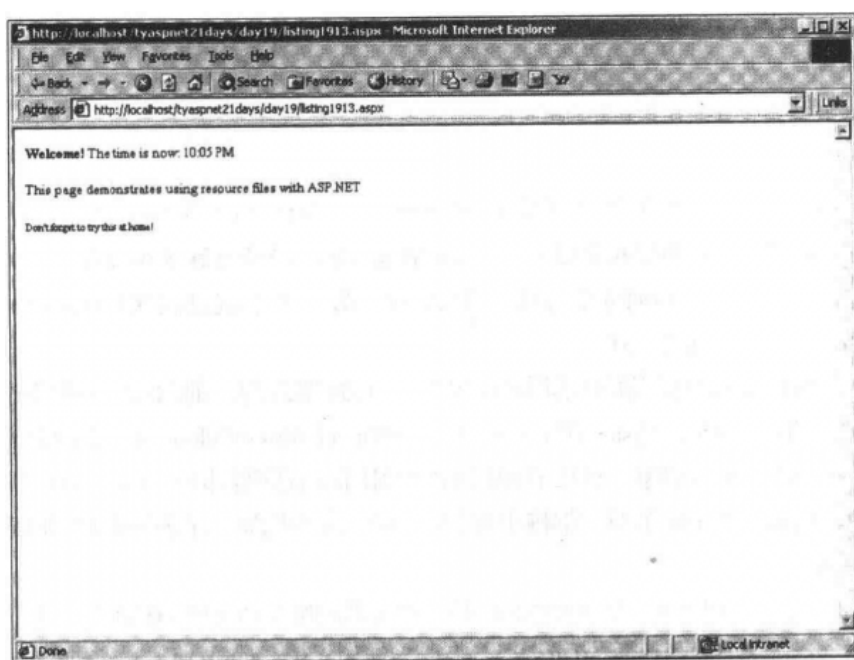


图 19.12 当文化被设置为 `en-us` 时, ASP.NET 使用资源文件 `data.en-us.resources`

**提示:** 由于文化信息必须在整个应用程序中保持一致, 因此可以考虑在 `global.asax` 文件的 `Application_BeginRequest` 方法中设置文化。这样, 每当服务器收到新的请求时, 文化都将被自动、正确地设置。

另外,您也可创建ResourceManager对象,并在Application\_OnStart事件中将其保存为一个应用程序级的变量,以免每收到一个新请求时都重新创建它。例如,可以将下面的代码加入到global.asax文件的Application\_OnStart事件中:

```
Application("RM")=New ResourceManager('data',_
    Server.MapPath("."),Nothing)
```

这样便拥有一个整个应用程序都可以使用的ResourceManager对象,可以通过变量Application("RM")来访问它。

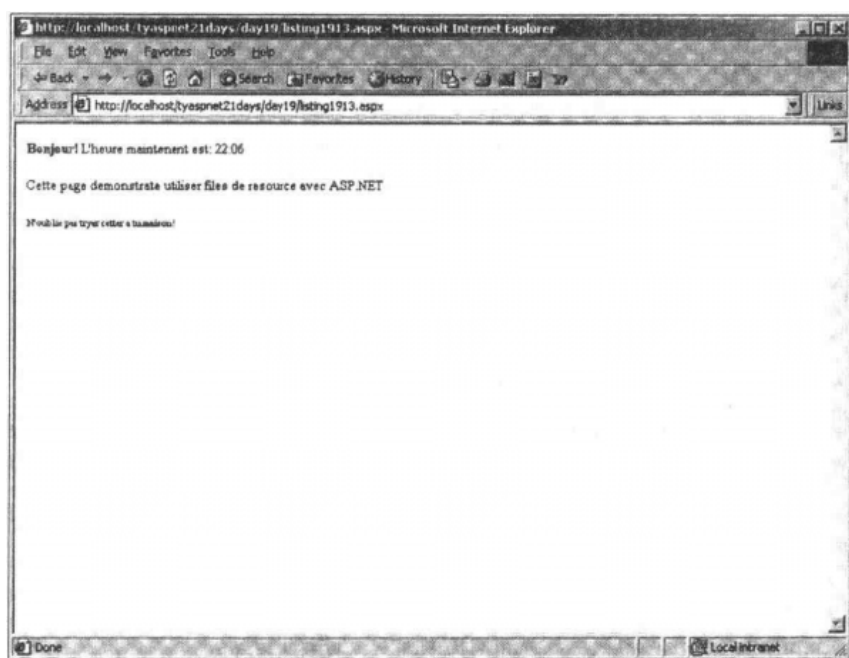


图 19.13 当文化被设置为 fr-FR 时, ASP.NET 使用资源文件 data.fr.resources

## 19.4 这不是 ASP

将代码和内容分开并从中受益并非 ASP.NET 新增的功能。事实上,开发人员致力于这样做已经有一段时间了,特别是在传统 ASP 中。传统 ASP 依赖于代码交付块,而不是代码声明块,这导致了被称为“代码混乱”的问题,即在整个页面中,代码和 HTML 交织在一起。这使得调试和修改页面非常困难,因为开发人员必须搜索整个页面,从中找到错误。通过使用代码声明块,这种问题基本上得到了解决,因为所有的代码都位于一个地方:页面的开始位置。

在传统 ASP 中,开发人员依赖于服务器端包含来将代码和内容分开。这些文件可以包含任何类型的代码(包括 HTML),因此它们常被用来存储通用的 UI 功能。在 ASP.NET 中,虽然服务器端包含仍然可用,但通常被用户控件和 code-behind 表单所取代。与服务器端包含相比,用户控件和 code-behind 表单都提供了更强的编程控制能力,而从技术上说,服务器端包含没有提供这样的能力。

资源文件是 ASP.NET 的新增特性。熟悉诸如 VB 和 C++ 等编译型语言的开发人员可能不会对这些文件感到陌生。在这些语言中使用资源文件的原因与本章介绍的相同:只是为了将代码和内容分开。

传统 ASP 本身并没有将可本地化的信息和页面分开的功能,因此对于整个站点的内容都必须

创建多个版本。另外，传统 ASP 不允许修改文化和区域信息。

所有这些新功能使得 ASP.NET 比传统 ASP 通用得多。现在开发人员在设计页面上的代码布局时，有更多的选择。

## 19.5 总 结

本章介绍了一些将页面和代码分开的新方法。对于 Web 表单框架，可以使用 code-behind 表单，它通过将代码放在单独的文件中来将页面和代码分开。通过使用本地化和资源文件，无需修改代码便可以显示不同的内容给用户。

为创建 code-behind 表单，需要创建一个从 System.Web.UIPage 类派生而来的类（如果 code-behind 表单是用于用户控件的，则从 System.Web.UIUserControl 类派生）。然后将.aspx 文件中的所有 ASP.NET 代码移到这个新类中，并声明页面中要使用的服务器控件的公有版本。例如，如果.aspx 文件中有一个名叫 lblMessage 标签，则在 code-behind 类中加入以下代码：

```
public lblMessage as Label
```

这确保您能够处理该服务器控件的属性和事件。然后在.aspx 文件中加入属性 Inherits 和 src，分别指定 code-behind 类的名称和文件的位置。这样，code-behind 类和 ASP.NET 页面的功效将于创建 code-behind 类之前相同。

您可以使用 Request.UserLanguage 属性来从浏览器那里检索用户使用的首选语言，然后使用 CultureInfo 对象根据该语言来设置文化信息。该对象提供了大量用于检索文化信息（如日期和时间格式）的方法。RegionInfo 对象与 CultureInfo 类似，但提供的是关于特定区域或国家的信息，如货币符号或国别的缩写。

使用 resgen.exe 工具，可以将资源文本文件生成.resources 文件。请务必将资源文本文件以 prefix.culture.tex 的格式命名。然后，为页面将以其显示的每种语言创建一个资源文件，这些文件以关键字/值对的格式存储信息。然后便可以使用 ResourceManager 对象来装载资源文件，并显示本地化后的字符串。

下一章将 ASP.NET 页面开发中另一个非常重要的部分：调试 ASP.NET 页面。没有人是十全十美的，因此创建页面时会犯错误。下一章介绍大量跟踪并修正错误的方法。

## 19.6 问与答

问：可以修改浏览器默认的首选语言吗？

答：可以。执行“开始/设置/控制面板”，然后双击“Internet”图标。在弹出的窗口中包含一个“语言”按钮，单击它，可以修改浏览器的首选语言。在新打开的窗口中，可以添加新语言、删除原有的语言并修改每种语言的优先级。

问：可以在编译的代码（如业务对象）中使用资源文件吗？可以编译资源文件吗？

答：都可以。可以通过下列两种方式在业务对象中使用资源文件：随业务对象一道编译资源文件或将资源文件作为单独的.resources 文件。对于后一种方法，可以像本章介绍的那样使用资源文件；对于前一种情况，则必须对代码稍作修改。

当资源文件被编译时，方法 CreateFileBasedResourceManager 不再管用，您必须使用下面的 ResourceManager 构造函数，指定包含编译后的资源文件的部件：

```
dim objRM as New ResourceManager('prefix',System.Reflection._
    Assembly.GetExecutingAssembly(),Nothing)
```

其中的第一和第三个参数与 `CreateFileBasedResourceManager` 方法相同，第二个参数 `System.Reflection.Assembly.GetExecutingAssembly` 返回一个 `Assembly` 对象，该对象代表从中执行代码的部件。这样，ASP.NET 便知道到哪里去找资源文件，而您则可以像以前那样从中检索值。

可以像 VB.NET 文件那样使用 `vbnc.exe` 来编译资源文件。编译后的、只包含资源的文件的文件被称为附属部件 (satellite assembly)，更详细的信息，请参考 .NET 框架 SDK 文档。

## 19.7 作业

下面的作业帮助巩固本章介绍的概念。继续阅读下一章前，完全理解其答案将很有帮助。答案见附录 A。

### 19.7.1 小测验

1. 判断正误：code-behind 类必须从 `System.Web.UIPage` 类派生而来。
2. 为什么必须在 code-behind 类中为 ASP.NET 页面中要使用的服务器控件声明公有变量。
3. 创建一个 @ Page 编译指令，该指令使用一个名叫 `MyControl` 的 code-behind 类，该类位于文件 `MyControl.vb` 中。
4. 何为资源文件？
5. 下述代码行的功能是什么？  

```
dim strVar as string = Request.UserLanguage()
```
6. 工具 `resgen.exe` 有何功能？
7. `CurrentCulture` 和 `CurrentUICulture` 的区别何在？
8. 判断正误：`CultureInfo.CurrentCulture` 是 `System.Threading.CurrentThread.CurrentCulture` 的简捷方式。

### 19.7.2 练习

1. 创建一个为多个国家的货币进行换算的计算器。让用户输入一种货币的金额，并从列表框中选择目标货币，然后换算成相应的货币。使用一个资源文件保存出汇率值（如果不知道汇率，也不用担心，只需虚构即可）。请使用 code-behind 技术。



## 第 20 章

# 调试 ASP.NET 页面

本章介绍一个非常重要的主题：调试。无论程序员的水平有多高，总会犯错误，因此调试应用程序是无法避免的。编写的代码越多，其中的 bug 也会越多。

您很可能在计算机应用程序中遇到了很多 bug。回忆一下您收到描述了某种您不懂的错误的消息，无论是来自桌面应用程序还是 Web 站点。大多数时候，这些错误将导致应用程序崩溃，而其他时候，bug 导致应用程序的输出结果与您的期望不符。无论是哪种情况，您的工作都将无法完成。最理想的情况是，代码编写后便能完成相应的任务，但情况往往并非如此。

ASP.NET 提供了两个强大的调试应用程序的工具：CLR 调试器和跟踪服务 (Trace Service)。本章将介绍这两个工具以及 try 和 catch 语句块。

本章包含以下内容：

- 如何使用 try 和 catch 语句进行调试；
- 如何使用 Response.Write 进行调试；
- 如何创建并引发自定义的错误；
- 跟踪工具是什么？如何使用它；
- 如果使用 CLR 调试器在运行阶段进行调试。

### 20.1 调试简介

您可能在开发 ASP.NET 页面时遇到过错误，从忘记加入属性 `runat="server"` 到花了几天的时间才解决的错误。遇到错误是令人沮丧的，看到如图 20.1 所示的屏幕时，可能会让您抓破头皮。

随着应用程序越来越复杂，避免潜在的 bug 以及找出并修复已有的 bug 也越来越困难。让我们来看一个典型的例子，如清单 20.1 所示。

#### 清单 20.1 找出 bug

```
1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
```

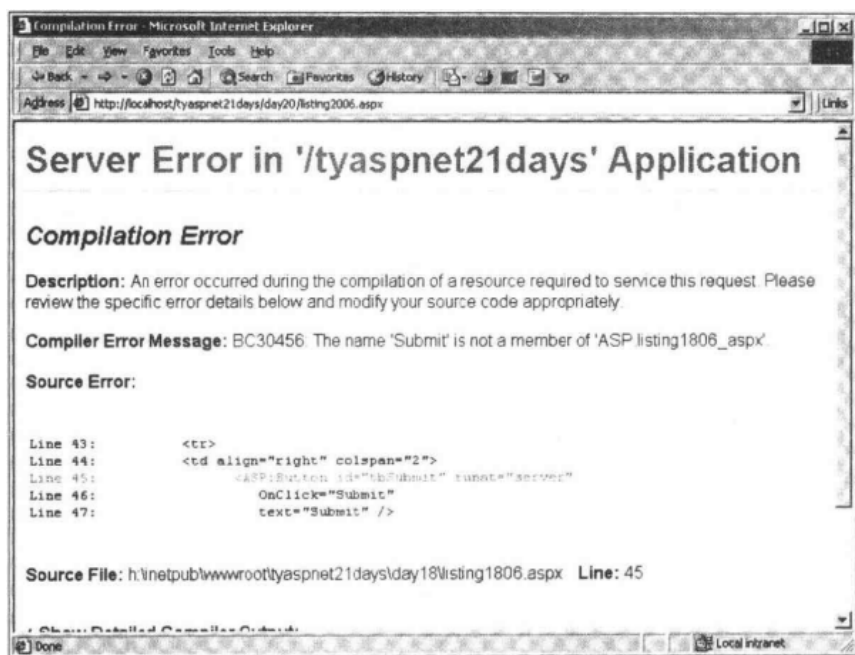


图 20.1 一个典型的 ASP.NET 错误

```

6  sub Page_Load(obj as Object, e as EventArgs)
7      'set up connection
8      dim objConn as new OleDbConnection _
9          ("Provider=Microsoft.Jet.OLEDB.4.0;" & _
10         "Data Source=C:\ASPNET\data\banking.mdb")
11
12      'open connection
13      dim objCmd as new OleDbDataAdapter _
14          ("select * from tblUsers", myConnection)
15
16      'fill dataset
17      dim ds as DataSet = new DataSet()
18      objComm.Fill(ds, "tblUsers")
19  end sub
20 </script>
21
22 <html><body>
23     <form runat='server'>
24
25     </form>
26 </body></html>

```

在浏览器中查看该文件（别忘了在 web.config 文件中打开调试模式），将看到类似于图 20.2 的结果。

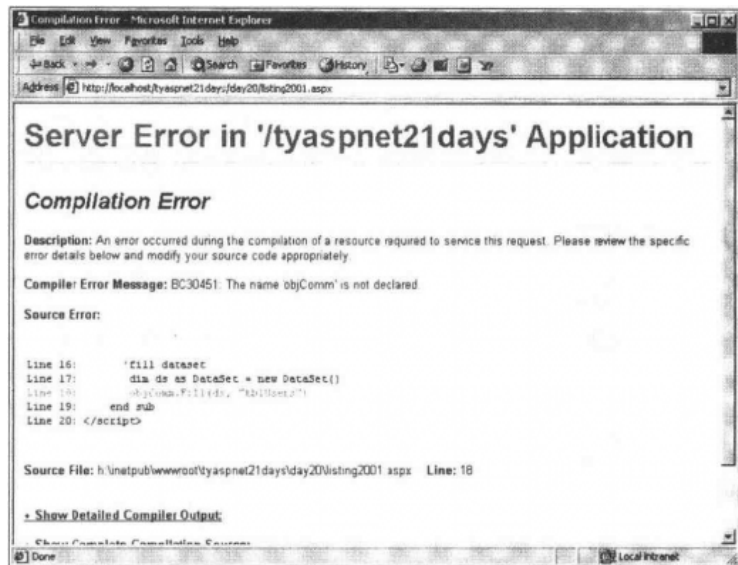


图 20.2 代码有错

显然，清单 20.1 中的代码中有错误，如果您还没有找到，则图 20.2 所示的页面将有所帮助。图 20.2 指出了是哪一行代码有错，并指出了文件和行号。好像是说第 18 行有拼写错误：您使用的是 `objComm`，而不是 `objCmd`。这种错误很容易修复，这要归功于上述错误明细页面。修改上述错误后，代码便能正确地运行了。

让我们来详细看看该页面，它提供了大量关于错误来源的信息。例如，向下滚动该错误页面，将看到两个链接：`Show Detailed Compiler Output` 和 `Show Complete Compilation Source`。展开这些链接，将得到如图 20.3 所示的结果。

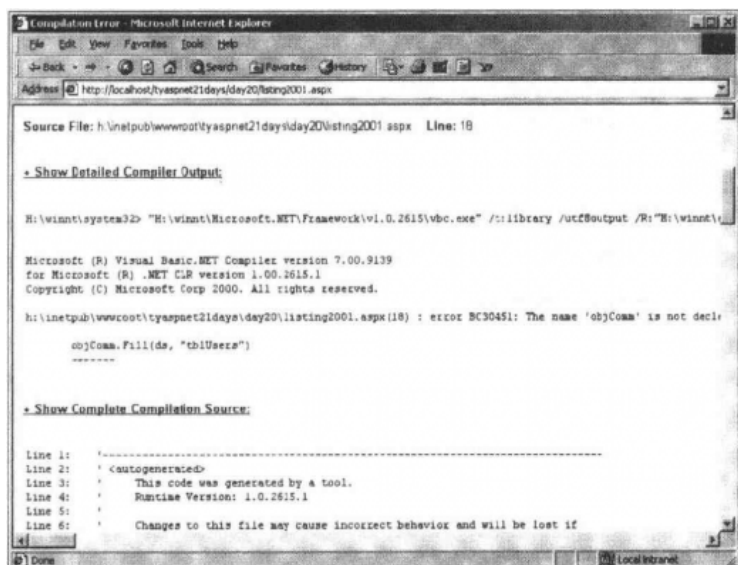


图 20.3 更详细的错误信息

**注意：**如果没有启用调试模式，则可能看不到这些额外信息的链接。更详细的信息，请参考第 18 章

在第一段 `Show Detailed Compiler Output` 中，ASP.NET 指出了想完成的工作以及未能完成的原因。

图 20.2 中的第 1 行显示了 ASP.NET 用来编译 ASP.NET 源代码的命令(正如您看到的,使用的是 `vbc.exe`——VB.NET 编译器)。由于前面介绍过编译器和业务对象,因此其中的一切对您而言都不会陌生。

然后,编译器生成输出,以帮助您确定错误的原因。图 20.2 也显示了这些消息。在图 20.3 的第二部分 Show Complete Compilation Source 中,显示了 ASP.NET 试图进行编译的整个源文件,包括所有的编译器命令和 CLR 函数。

通常,不应将这些内容显示给用户。首先,用户不希望在其应用程序中看到错误消息;其次,聪明的用户可能利用这些源代码做出对您不利的事。因此,必须使用开发阶段可用的调试机制来防止错误。

ASP.NET 提供了几种机制来帮助您解决上述难题。您已经在前面见到过第一种了: `try` 和 `catch` 语句,如上一章的清单 19.1。这些语句帮助捕获你的应用程序中的错误。跟踪工具也能帮助捕获错误,同时还提供了其他信息。最后,一旦错误发生,则 CLR 调试器将是一个发现并修复错误的强大工具。

## 20.2 Try 和 Catch 语句

通常,当代码段发生错误时,应用程序将停止执行,并显示一条错误消息,就像您在执行清单 20.1 时看到的。只要有可能防止这种错误,您就会这样做(毕竟,理想情况下不应看到任何错误)。

使用 `try` 语句时,您告诉 ASP.NET,希望它尝试性地执行代码块。如果 `try` 语句中的代码发生错误,不要停止应用程序并显示错误,而继续执行,让您能够解决问题。例如,如果将清单 20.1 的第 18 行放在 `try` 语句中,则可以避免错误终止应用程序,而继续执行其余的代码。

`Try` 语句用于控制可能导致错误的代码段。在传统的应用程序开发中,这是必不可少的,因为某些操作肯定会引发错误。例如,虽然您可能不知道,但单击对话框的 Cancel 按钮通常会导致错误,应用程序必须以某种方式对这种错误进行处理。`Try` 语句及其相关的语句被称为结构化异常处理(structured exception handling)。

在本书的一些例子中您已经看到了, `try` 语句的语法很简单。它有 `try` 语句、要控制的代码以及 `catch` 或 `finally` 语句组成,例如:

```
1: 'create OleDbCommand object here
2: try
3:     objCmd.Connection.Open
4:     ...
5: catch
6:     handle error
7: end try
```

这里控制的是第 2-5 行的代码,因为这些代码可能导致错误。如果 `try` 语句块中没有发生错误,将执行第 7 行的代码,结果与正常情况完成相同。如果确实发生了错误,则将执行第 5 行的 `catch` 语句,其中包含处理错误的代码。

**新术语:** 让我们更详细地讨论一下技术上的细节。当 VB.NET 应用程序发生错误时,将引发(thrown)异常(exception)。这意味着发生了不应该发生的事情,而 VB.NET 指出发生的错误,以便有人知道。如果不对异常采取任何措施,错误将导致应用程序崩溃,看到错误的将是最终用户。

引发异常让您能够在其导致应用程序崩溃之前发现它,并防止用户看到这种错误。如果能够及

时地发现错误，则可以防止应用程序崩溃，并修复出现的问题。

回到 try 语句。发生错误时，try 语句将提醒其配套语句 catch，后者将捕获并处理异常。清单 20.2 列出了一个例子。

**清单 20.2 一个调试范例**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     dim Conn as new OleDbConnection("Provider=" & _
7         "Microsoft.Jet.OLEDB.4.0;" & _
8         "Data Source=C:\ASPNET\data\banking.mdb")
9
10    sub GetData(obj as Object, e as EventArgs)
11        dim objCmd as OleDbCommand = new OleDbCommand _
12            ('select * from tblUsers where UserID = '&_
13                tblID.Text,Conn)
14        dim objReader as OleDbDataReader
15
16        objCmd.Connection.Open()
17        objReader = objCmd.ExecuteReader
18
19        dgData.DataSource = objReader
20        dgData.DataBind()
21
22        objReader.Close
23        objCmd.Connection.Close()
24    end sub
25 </script>
26
27 <html><body>
28     <form runat="server">
29         <asp:Label id="lblMessage" runat="server"
30             maintainstate=false /><br>
31         Enter an ID: <asp:TextBox id="tblID" runat="server"
32             AutoPostBack=True
33             OnTextChanged=GetData /><p>
34         <asp:DataGrid id="dgData" runat="server"
35             BorderColor="black"

```

```

36      GridLines="Vertical"
37      width="100%"
38      Font Name="Arial"
39      Font-Size="8pt"
40      HeaderStyle-BackColor="#cccc99"
41      ItemStyle-BackColor="#ffffff"
42      AlternatingItemStyle-BackColor="#cccccc"
43      AutoGenerateColumns="true" />
44  </form>
45 </body></html>

```

分析：这是一个简单的页面，它根据用户ID返回用户信息。当用户在文本框中输入一个数字时，数据库命令将被执行，并将返回的数据绑定到DataGrid。但如果用户不小心输入了字母，情况将如何呢？图20.4显示了这种情况下的结果。

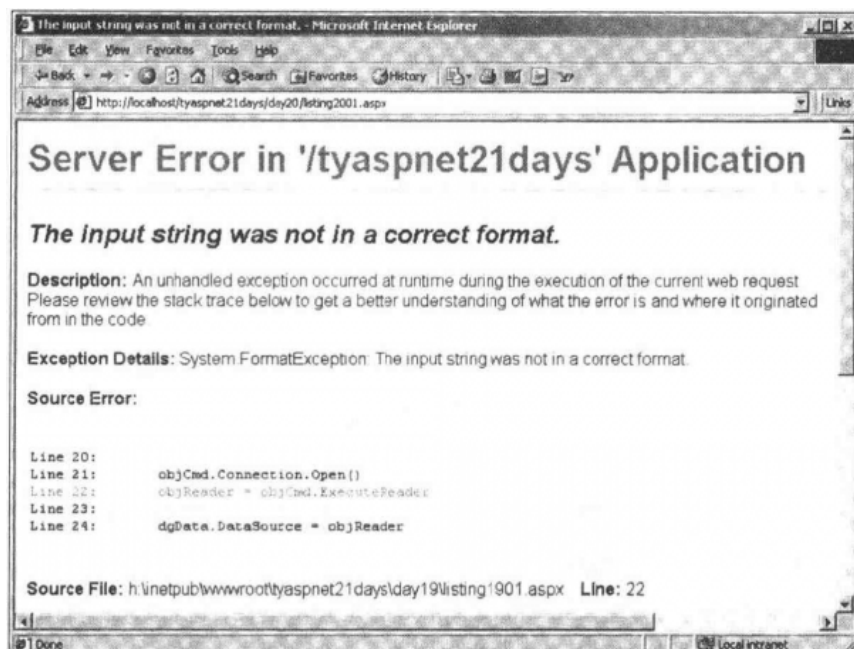


图 20.4 错误的用户输入引起的错误

不应该将这样的信息显示给用户。让我们在该清单中加入一条 `try` 语句，以便妥善处理这种错误。使用清单 20.3 替换第 16–23 行。

#### 清单 20.3 在清单 20.2 中使用 `try` 语句块

```

16 try
17     objCmd.Connection.Open()
18     objReader = objCmd.ExecuteReader
19
20     dgData.DataSource = objReader
21     dgData.DataBind()

```

```

22
23     objReader.Close
24     objCmd.Connection.Close()
25 catch
26     lblMessage.Text = "Invalid input!"
27 end try

```

再次请求该页面，将得到如图 20.5 所示的结果。

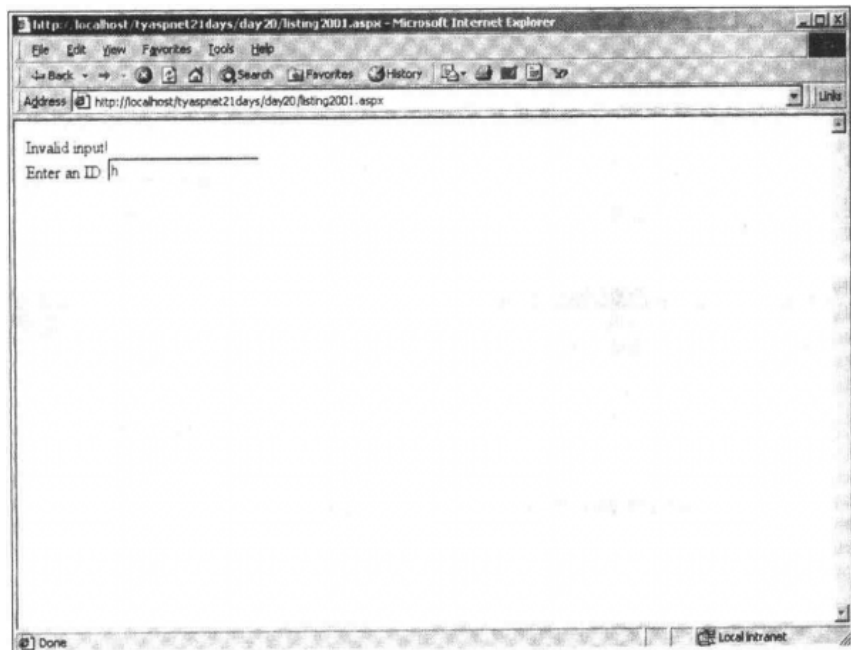


图 20.5 妥善地处理错误

Try 语句块在错误引发问题之前捕获它，并显示一条错误消息，如清单 20.3 的第 26 行所示。然后继续从第 27 行执行，程序正常地执行完毕。

异常被以层次的方式进行分组，System.Exception 类是异常基类，包含其他所有的异常。接着是 System.Exception，再下面是其他各种异常，如 OleDbException 和 FormatException。

层次结构的越下面，错误越具体。例如，System.Exception 包含所有内置的异常，而 FormatException 则只包含非法的用户输入引起的错误。就像以前讨论的其他对象一样，这些异常都是 .NET 类。图 20.6 是一个异常层次结构的范例。

**注意：**虽然在异常层次结构中，System.Exception 在 Exception 的下面，但它仍然属于名称空间 System。也就是说，您将使用 System.System.Exception，而不是 System.Exception.System.Exception 来引用它。

使用通用的 catch 语句（像前面那样）来捕获各种类型的错误。换句话说，它将捕获基类 Exception 下的所有异常。但可以指定要处理的错误类型，其语法如下：

```
catch VariableName as ExceptionType
```

例如，您可以用清单 20.4 替换清单 20.3。

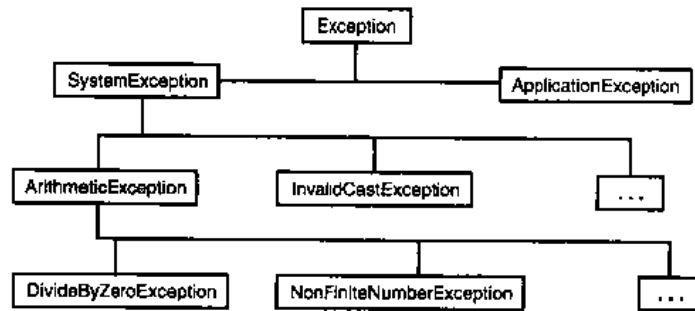


图 20.6 一个系统异常层次结构的例子

## 清单 20.4 修订后的 try 语句块

```

16 try
17     objCmd.Connection.Open()
18     objReader = objCmd.ExecuteReader
19
20     dgData.DataSource = objReader
21     dgData.DataBind()
22
23     objReader.Close
24     objCmd.Connection.Close()
25 catch objEx as FormatException
26     lblMessage.Text = objEx.Message
27 catch objEx as OleDbException
28     lblMessage.Text = "Database error!"
29 catch objEx as Exception
30     lblMessage.Text = "Unknown error!"
31 end try
  
```

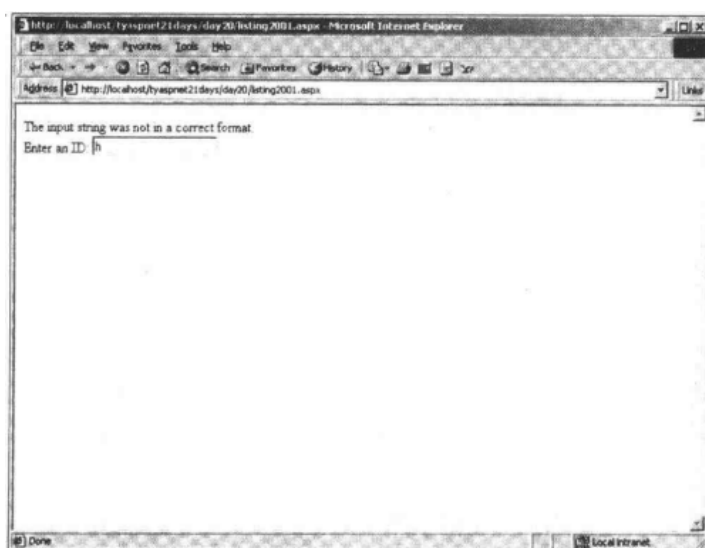


图 20.7 多个 catch 语句捕获更具体的错误，使用异常的 Message 属性来显示与异常相关联的错误文本



第 25 行捕获 `FormatException`（及其下面的）异常。请不要担心 26 行的 `Message` 属性，马上就会介绍它。第 27 行捕获 `OleDbException` 异常，而 29 行捕获所有的异常。由于异常是层次式的，因此 `catch` 语句将处理其声明的异常以及更为具体的异常。因此，应按具体次序编写 `Catch` 语句，越抽象的放在越后面。使用 `catch` 语句时，应该注意以下几点：

派生的异常类（如 `FormatException` 或 `OleDbException`）表示要检测的特定错误，因为这些错误可能发生或者当这些错误发生时需要采取特定的措施。

遇到相应的 `catch` 块后，`try...catch` 结构停止检测错误。

`Exception` 包含所有的异常，它类似于 `case` 语句中的 `case else`。

该清单的输出如图 20.7 所示。

还有一个与 `try` 语句向关联的语句：`finally` 语句。可以使用该语句块来执行一些清理代码或其他善后工作，而不管异常是否发生。可以将清单 20.5 中的代码插入到清单 20.2 中，以关闭数据库连接，而不管发生什么情况。

#### 清单 20.5 使用 `finally` 语句

```
16 objCmd.Connection.Open()
17 try
18     objReader = objCmd.ExecuteReader
19
20     dgData.DataSource = objReader
21     dgData.DataBind()
22
23     objReader.Close
24 catch objEx as Exception
25     lblMessage.Text = "Unknown error!"
26 finally
27     objCmd.Connection.Close()
28 end try
```

图 20.8 说明了使用 `try` 语句块时的处理过程。

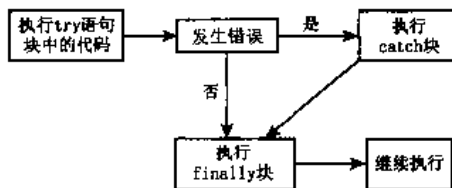


图 20.8 使用 `try` 语句块时的执行次序

每种异常类都有一些属性，可以使用它们来确定引发错误的原因。

- **HelpLink**：到一个文件的链接，该文件包含关于错误的更详细的信息（您必须自己创建链接和文件）。
- **InnerException**：一个内部异常的引用。如果另一个异常被捕获，并被传递给另一个异常处理程序，则该属性将返回到第一个异常的引用。

- **Message:** 描述异常的消息。
- **Source:** 一个字符串, 其中包含引起错误的对象的名称。
- **StackTrace:** 返回一个指出错误原因的栈跟踪(更详细的信息, 请参考本章后面的“跟踪”一节)。
- **TargetSite:** 引起错误的方法。

### 20.2.1 引发异常

异常很有用, 它不仅仅用于捕获错误。事实上, 有时候您可能想引发自定义的异常。

例如, 还记得在第 16 章和 17 章创建的 Web 服务吗? 假设您使用数据库来验证信息, 并且未能通过验证, 则可以在服务中引发一个自定义的异常, 让客户端应用程序使用 try 和 catch 语句来处理。

要引发异常, 只需使用关键字 **throw** 即可:

```
throw Exception("I'm taking exception to that!");
```

该语句以 **Exception** 类为基类创建一个新的异常, 括号内的字符串是错误消息。将这行代码加入到清单 20.1 的 **DataGet** 方法的开始位置后, 则提交表单后, 将得到如图 20.9 所示的页面。

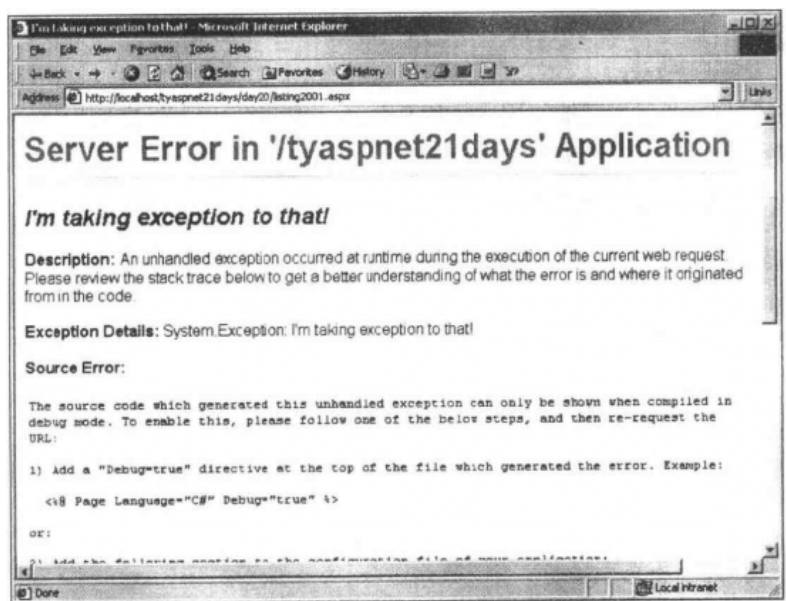


图 20.9 使用 throw 来引发自定义的异常

如果不想自己处理异常, 则引发自定义异常很有用。只要调用引起错误的代码的函数就能够处理这种异常, 您无需为此操心。

例如, 一种引发自定义异常很有用的情况是, 用户在表单中输入文本。当表单被提交后, 提交处理程序中的代码调用一个有效性验证方法, 来确定是否至少有一个条目是无效的(例如用户输入了一个数字, 而不是其姓名)。有效性验证子程序可以引发一个自定义异常, 来告诉调用者发生了错误。然后调用者便可以根据需要处理该异常。

您不一定非得引发一个自定义异常, 您可以引发任何可用类型的异常。例如, 下面的代码引发一个异常, 指出缺少必不可少的参数:

```
throw new ArgumentNullException();
```

在您创建业务对象时这很有用，它让实现者能够根据其需要来处理错误。

### 20.2.2 何时使用 try 语句

try 语句对于调试代码很有帮助，让您无需中断执行便能发现错误。使用 try 语句时，必须遵循一些通用的规则。

首先，访问 ASP.NET 环境之外的系统时（如访问数据库或文件），一定要使用 try。有很多因素是您无法控制的，而这些因素可能出现問題，例如，文件可能不存在；用户可能没有访问该文件的权限；数据库可能断开；要访问的数据库表可能被删除了等等。因此，必须使用 try 语句。

对代码的运行情况不确定时，特别是测试或调试时，应使用 try 语句块。例如，您创建的代码可能严重地依赖于应用程序中的其他因素（绑定到 DataGrid 是一个绝佳的例子）。

不要依赖于 try 语句来捕获用户输入错误，而应该采用专门为这种用途而创建的其他方法，如有效性验证控件。对于处理这种错误，这些控件提供了更为丰富的编程模型，比 try 语句更能准确地指出错误。

最后，不要到处使用 try 语句块，只在必要时才使用。该语句带来的处理开销可能很小，但不会任何时候都值得。

## 20.3 跟 踪

您可能已经发现，通过灵活地使用注释和 Response.Write 可以调试应用程序。如果怀疑某个代码段是引发错误的罪魁祸首，可以将其注释掉，并在合适的位置使用 Response.Write 方法来查看您的猜测是否正确。例如，请看清单 20.6。

**清单 20.6 测试部分代码**

```

1 <%@ Page Language="VB" %>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script runat="server">
6     sub Submit(obj as object, e as eventargs)
7         dim strSQL as string = _
8             "SELECT UserID FROM tblUsers WHERE " & _
9             "UserName = '" & tbUserName.Text & "' AND " & _
10            "Password = '" & tbPassword.Text & "'"
11
12         'do database stuff here
13     end sub
14 </script>
15
16 <html><body>
17     <form runat="server">
18         <asp:Label id="lblMessage" runat="server" />

```

```

19      Username:
20      <asp:TextBox id="tbUserName" runat="server" /><p>
21          Password:
22          <asp:TextBox id="tbPassword" runat="server"
23              TextMode="password" /><p>
24          <ASP:Button id="tbSubmit" runat="server"
25              OnClick="Submit"
26              Text="Submit" />
27      </form>
28 </body></html>

```

该表单接受用户名和密码，以便查询数据库。虽然这是一个非常简单的例子，但假设还是发生了错误——可能是未能从数据库返回正确的信息。在这种情况下，通常需要检查 SQL 是否正确。

您可以在第 10 行之后加入下述代码行，以验证 SQL 语句是否正确：

```
Response.Write(strSQL)
```

然后将怀疑的代码行注释掉，以防错误再次发生。图 20.10 显示了结果。

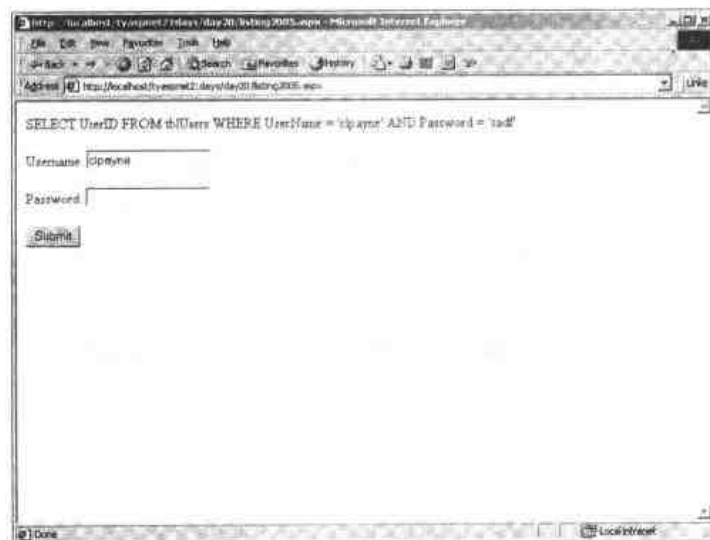


图 20.10 Response.Write 让您能够检测变量的值

**Response.Write** 对于跟踪变量并检其值非常有用。不幸的是，这种方法除了值之外，不能提供其他的信息，而且在不同的地方加入和删除这样的语句也非常麻烦。

ASP.NET 跟踪技术解决了这种问题，它让您能够输出调试语句并跟踪应用程序的执行。跟踪告诉您哪些方法已经执行；每个方法执行时花费的时间；变量的值等等。这些信息都是以易于阅读的格式提供的。

另外，结束调试后，您无需删除任何跟踪调试语句，只需使用一个命令就可以全部禁用它们。可以针对单个页面启用跟踪，也可以针对整个应用程序。无论采用哪种方式，当调试语句被禁用后，它们都不会对应用程序产生任何影响，也不会将任何输出发送给客户。

在 ASP.NET 有两种级别的跟踪方式：页面级和应用程序级。我们首先介绍页面级跟踪，因为

应用程序级跟踪只不过是这种跟踪的扩展而已。

### 20.3.1 页面级跟踪

要进行跟踪很容易。在清单 20.6 的第 1 行加入编译指令 `Trace="True"`，如下所示：

```
<%@ Page Language="VB" Trace="true" %>
```

这将打开页面的跟踪功能，图 20.11 和图 20.12 显示在浏览器中查看页面时的情况。

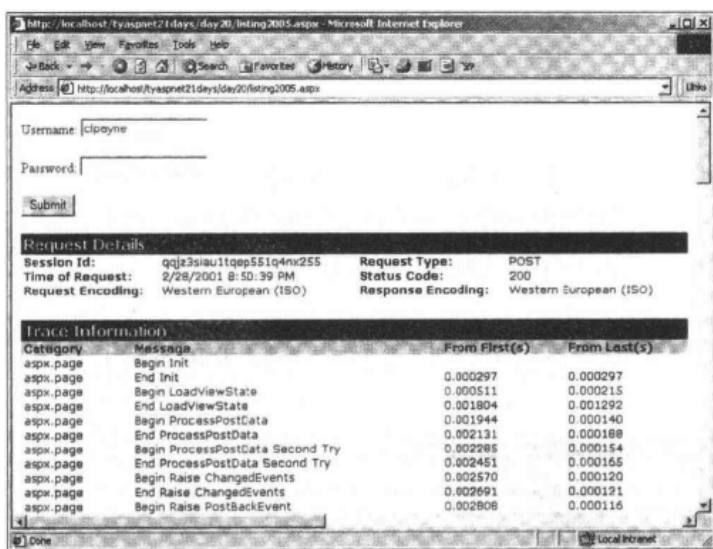


图 20.11 跟踪提供了大量的调试信息

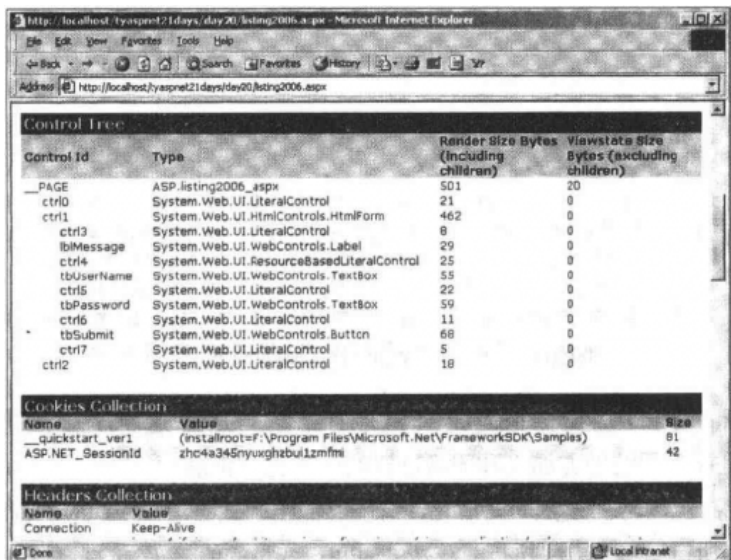


图 20.12 向下滚动以查看更多的跟踪输出

到底发生了什么情况？突然之间，页面便填满了新的信息。请滚动窗口，查看其中的每一项内容。这些跟踪信息差不多提供了关于该页面的所有信息！我们来看看其中的每一个部分，每次一个部分。

页面的开始与清单 20.6 的输出相同，您仍然可以与之进行交互，就像后面没有任何信息一样

当您启用跟踪功能时, ASP.NET 自动将这些信息加入到页面中。这些信息只是用于调试和提高性能的, 显然不能让最终用户看到它们。

接下来的 Request Details 部分指出客户是如何请求页面的。其中包含当前用户的唯一性会话 ID、请求时间、请求谓词(这里为 POST)、状态编码以及关于编码方法的信息。在这个例子中, 这些信息并没有多大用处, 但它可能很有用。

Trace Information 部分列出了页面的执行情况, 其中包括页面上发生的每个事件及其花费的时间(离页面请求开始的时间以及离前一个时间完成的时间)。图 20.11 的一些事件是您所熟悉的, 但也有几个是新的。类别指出了跟踪发生的位置。在接下来一节中, 将介绍如何添加自己的类别。

接下来的 Control Tree 部分按层次顺序列出了页面中使用的所有服务器控件, 甚至还列出了 LiteralControls(关于这些对象的描述, 请参见第 2 章)。这部分列出了每个控件的大小(单位为字节)以及维护视图状态信息需要的字节数。从中您可以清楚地知道维护表单需要的内存量并决定是否关闭视图状态维护。

Cookies Collection 部分列出了使用的所有 cookie 及其值。这是否包含信息取决于应用程序的特性。

Headers Collection 部分列出了 HTTP 报头中存储的信息, 其中包括主机服务器、浏览器版本和类型、引用的 URL 和内容的类型等等。

Form Collection 部分列出了通过发送(Post)操作发送的所有数据, 如果提交了页面上的输入表单, 则从中可以看到您输入的值以及视图状态信息。

最后, Server Variables 部分列出了诸如 HTTP 报头、服务器名称和端口、HTTP 连接的类型等信息。服务器将使用这些信息来设置应用程序。

您也可以按事件发生的时间或事件所属的类别(在属性 TraceMode 中指定)对页面的跟踪信息进行排序。例如:

```
<%@ Page Language="VB" Trace="true" TraceMode="SortByTime" %>
```

这是默认的排序方式。SortByCategory 以不同的方式组织输出, 但结果将与前面相同, 因为图 20.11 中只有一个类别。

该页面提供了大量关于应用程序性能方面的信息。您可以准确知道每个处理步骤所需的时间并确定是否需要进行调整。例如 Begin Render 和 End Render 之间的时间, 即在浏览器中显示页面所花的时间, 总是最长的时间值。

您可以使用 Trace 对象将自定义的调试信息写入到 ASP.NET 页面中。该属性实际上是一个类, 它包含两个方法: Write 和 Warn。这两个方法几乎完全相同, 前者输出调试信息, 而后者以红色输出同样的信息。

我们来看一个例子。将下述代码加入到清单 20.6 的第 11 行(替换 Response.Write, 如果还没有被删除的话):

```
Trace.Write("Custom category", strSQL)
```

第一个参数是要输出的信息所属的类别, 第二个参数是要显示的字符串。提交表单后, 您将看到如图 20.13 所示的结果。

请注意 Trace Information 中间的新类别, 该事件发生在 Begin RaisePostBack 之后, 但在 End RaisePostBack 之前, 它处理表单提交事件。该新项目包含 strSQL 的值。注意到实际的页面中并没有出现新文本, 这是因为 Trace.Write 和 Trace.Warn 只是将信息写入到跟踪日志中, 而不是客户端的输出中。

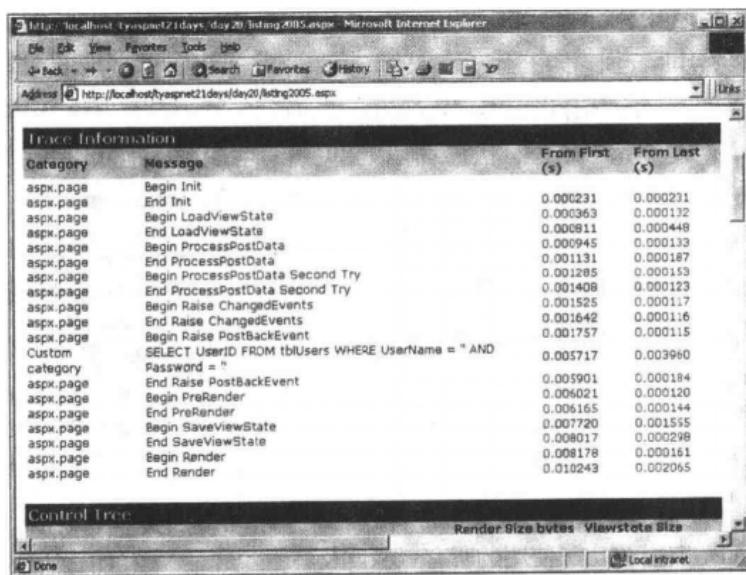


图 20.13 添加自定义的调试信息

我们来看一个稍微复杂一些的例子。清单 20.7 与第 14 章的练习 1 类似，创建了一个 DataSet，将其存储在高速缓存中，然后将数据绑定到一个 DataGrid。对于以后的请求，将从高速缓存（而不是数据库）中检索数据，以节省处理时间。同时包含一个按钮，让用户使高速缓存失效，从而重新从数据库中检索数据。

#### 清单 20.7 一个更复杂的使用跟踪的范例

```

1 <%@ Page Language="VB" Trace="true"%>
2 <%@ Import Namespace="System.Data" %>
3 <%@ Import Namespace="System.Data.OleDb" %>
4
5 <script language="VB" runat="server">
6   sub Page_Load(obj as Object, e as EventArgs)
7     Trace.Warn("Custom", "Page loading...")
8     if not Page.IsPostBack then
9       Trace.Warn("Custom", _
10        "No post, calling CreateData...")
11       CreateData()
12     end if
13   end sub
14
15   sub CreateData
16     dim source as DataView
17     source = Cache("DataView")
18
19     if source is nothing then
20       dim strConnString as string = "Provider=" & _

```

```
21         "Microsoft.Jet.OLEDB.4.0;" & _
22         "Data Source=C:\ASPNET\data\banking.mdb"
23
24         Trace.Warn("Custom", _
25         'Creating OleDbDataAdapter...')
26         dim objCmd as OleDbDataAdapter = new _
27         OleDbDataAdapter("select * from tblUsers", _
28         strConnString)
29         Trace.Warn("Custom", "SQL value: " & _
30         objCmd.SelectCommand.CommandText)
31
32         dim ds as DataSet = new DataSet()
33         objCmd.Fill(ds, "tblUsers")
34
35         source = new DataView(ds.Tables(0))
36         Trace.Warn("Custom", "Inserting into cache...")
37         Cache.Insert("DataView", source)
38
39         lblMessage.Text = "Data set created explicitly"
40     else
41         lblMessage.Text = "Data set retrieved from " & _
42         "cache<br>"
43     end if
44
45     Trace.Warn("Custom", 'Binding data...')
46     dgData.DataSource = source
47     dgData.DataBind()
48 end sub
49
50 sub ExpireCache(obj as object, e as eventargs)
51     Trace.Warn("Custom", _
52         "Removing from cache, call CreateData")
53     dim dv as dataview = Cache.Remove("DataVew")
54     CreateData()
55 end sub
56 </script>
57
58 <html><body>
59     <form runat="server">
60         <asp:Label id="lblMessage" runat="server">
```



```

61      main'ainState=false;><p>
62      <asp:Button id="btSubmit" runat="server"
63          text="Expire Cache"
64          OnClick="ExpireCache"><p>
65      <asp:DataGrid id="dgData" runat="server"
66          BorderColor="black" GridLines="Vertical"
67          cellpadding="4" cellspacing="0"
68          width="450" Font-Name="Arial"
69          Font-Size="8pt"
70      Header Style Back Color="#CCCC99"
71      Item Style Back Color="#ffffff"
72      Alternating Item Style-BackColor="#CCCC99"/>
73  </form>
74  </body></html>

```

分析：首先在第1行使用编译指令`Trace="true"`启用了跟踪功能。这样页面中将显示跟踪信息，而代码中所有的`Trace.Write`和`Trace.Warn`语句都将打印调试信息。

您在第7、9、24、29、36、45和51行分别放置了`Trace.Warn`语句，以便知道发生的具体事件。查看该页面后，单击`ExpireCache`按钮，将看到如图20.14所示的跟踪信息。

该页面提供了大量的信息。首先指出了事件的发生顺序，具体地说，`Page_Load`事件比其他任何Web表单事件发生的时间都早得多。在开发页面时未能注意到这一点，是一种常见的错误。

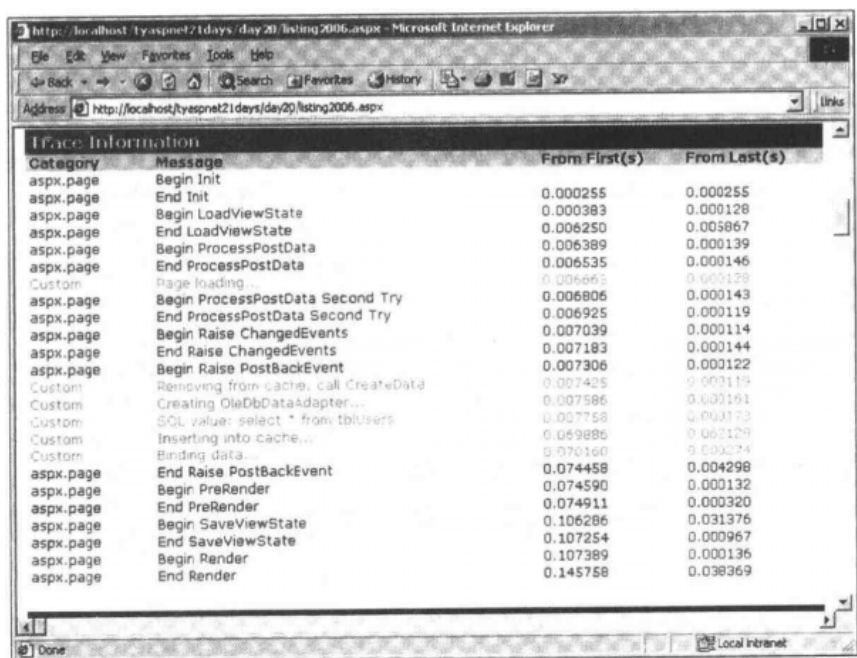


图 20.14 查看自定义调试信息

其次，该页面列出了您执行的SQL语句的值。这个例子中的SQL语句是静态的，因此这些信息对您的帮助不大；但当您需要动态地创建SQL语句时，这些信息将很有帮助。

接下来, 请注意 “Inserting into cache...” 和 “SQL Value: ...” 的时间之间的差异, 该时间差是从数据库检索信息所花费的时间, 它是其他任何自定义事件所花时间的 300 多倍, 甚至比其他所有时间花费的总时间还要长, 从中可以知道从数据库中检索信息的代价是非常高的。

刷新页面 (不重新提交表单数据), 并再次查看跟踪信息。您将发现花费的时间几乎少了一半。

最后, 向下滚动到 Control Tree 部分, 以查看维护视图状态 (参见第 2 章) 所需的内存量, 这些信息显示在 ViewState Size Bytes (不包括子视图状态) 栏中。将这些值相加, 可以知道该页面需要大约 6KB 内存来维护状态信息, 这大约是完全显示该页面所需内存 (显示在 Render Size Bytes 栏中) 的 40%。如果不需要维护视图状态, 则应使用页面编译指令 `EnableViewState="false"` 来关闭这项功能。

对于调试和提高应用程序的性能而言, 页面级跟踪是一个非常强大的工具。当您不再需要显示这些信息时, 可以在页面编译指令中使用 `Trace="false"` 来关闭跟踪功能, 并保留 `Trace.Warn` 语句不动即可。这些语句不会影响页面的执行, 也不会降低性能。

有时候您可能希望一些代码仅当跟踪功能被启用时才执行。例如, 您可能想显示自己创建的自定义调试信息。在这种情况下, 只需检查 `Trace.IsEnabled` 属性即可:

```
1: If Trace.IsEnabled then
2:     For I = 0 to ds.Tables("tblUsers").Rows.Count - 1
3:         Trace.Write("User Info", ds.Tables("tblUsers")._
4:             Rows(i)(0).ToString)
5:     next
6: end if
```

上述代码将返回的数据中的值写入到跟踪日志中, 但仅当 `Trace` 属性被设置为 `true` 时才这样做。因此, 您也可以使用 `Trace` 来避免代码的执行, 在调试期间经常需要这样做。

### 20.3.2 应用程序级跟踪

跟踪是一种非常有用的工具, 但如果需要针对站点中的每一个页面启用或关闭这项功能, 将是非常痛苦的事情。另外, 页面级跟踪提供的只是关于该页面的静态统计数据——它不能提供关于站点中其他页面的统计数据。使用应用程序级跟踪, 可以同时打开或关闭所有页面的跟踪功能, 同时能够在同一个地方查看跟踪结果。

在 `web.config` 文件中加入下面的编译指令, 可以启用对站点中所有页面的跟踪。

```
1: <configuration>
2:     <system.web>
3:         <trace enabled="true" />
4:     </system.web>
5: </configuration>
```

上述代码启用对整个站点的跟踪, 这意味着站点中的每一个页面都将输出跟踪信息 (除非显式地关闭该功能)。默认情况下, 应用程序级跟踪不显示关于每个页面的跟踪信息, 而是将累计信息存储在服务器的一个特殊文件 `trace.axd` 中 (您不用创建该文件, ASP.NET 将自动为您处理该文件, 事实上, 甚至在通过 Windows 资源管理器和 IIS 也看不到该文件——ASP.NET 自动创建它)。可以使用下面的 URL 来查看该文件:

`http://localhost/tyaspnet21days/trace.axd`

该页面的内容与图 20.15 类似, 这取决于您请求的是哪个页面。

**注意：**一定要在web.config文件中加入<trace enable="true">，否则在trace.axd中将看不到任何信息。

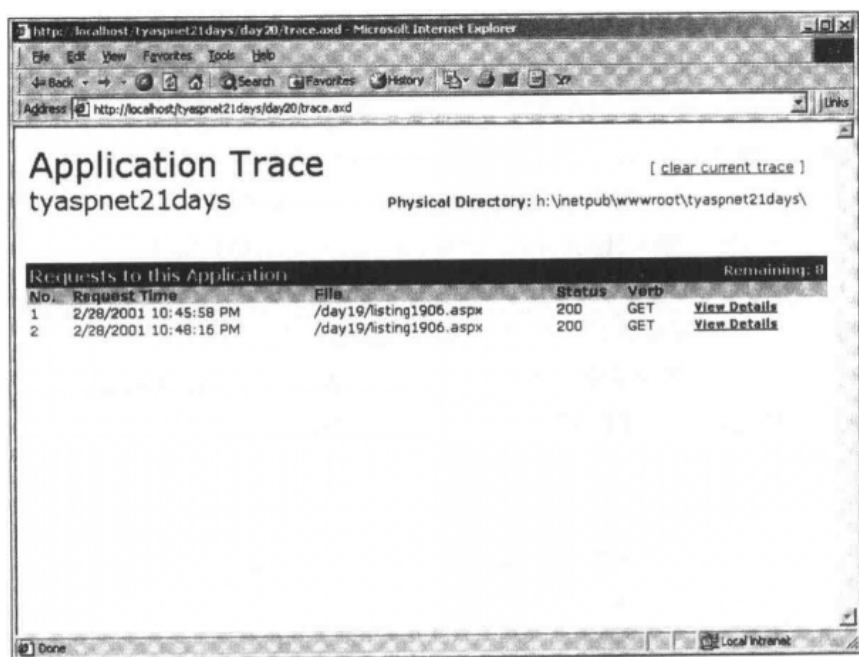


图 20.15 trace.axd 文件中的应用程序级跟踪信息

默认情况下，该页面将显示站点中最后 10 次请求的信息，包括请求时间、请求的文件、如何请求的（HTTP 谓词）和请求的状态。View details 链接将打开一个与图 20.11 类似的窗口（但不包含 UI 部分）。Clear current trace 链接清除应用程序日志中的所有数据。由于跟踪信息将占用一定的内存，因此清除这些数据使应用程序的性能稍微有所提高（只不过要知道的是，请求将很快增加日志中的数据）。

新建一个浏览器窗口，请求应用程序中的任何页面，并单击 Refresh 按钮几次，然后切换到 trace.axd，并单击 Refresh 按钮，您将看到其他一些请求的信息，同时每发出一个请求，Remaining 的值便减 1。当请求的次数超过限制后，最旧的请求的信息将被删除。

Trace.axd 的可配置性非常强，您可以修改存储的请求数目，跟踪显示的方法以及输出的显示位置。该 web.config 配置段的完整语法如下：

```
<Trace enabled="boolean" pageOutput="boolean"
  requestLimit="number" traceMode="mode" />
```

TraceMode 属性的功能与页面级跟踪下的情况类似，它告诉 ASP.NET 跟踪应如何运转。PageOutput 指定处理将跟踪信息存储到文件 trace.axd 外，是否还应显示在每个页面上，其默认值为 false。RequestLimit 指定内存中保存的请求数目，默认为 10 个。

**注意：**如果在特定页面中指定 Trace="false"，则就该页面而言，该设置将覆盖应用程序级的跟踪。也就是说，trace.axd 将不会显示该页面的任何信息。

应用程序级跟踪为快速查看关于站点中所有页面的统计信息，提供了一种简单的方法。当应用程序由于访问者过多而承受极大的压力时，您可以使用该工具来确定其瓶颈。

## 20.4 CLR 调试器

调试编译后的应用程序时，需要考虑一些特殊的因素。有时候这种应用程序无法生成开发人员能够看到的输出（例如，编译后的业务对象通常不显示任何数据）。通常，执行代码已经被转换为机器语言或 MSIL，这使得跟踪起来更为困难。对于这种应用程序，需要另一种不同的调试器——一种可以附加到运行的进程中的调试器。

新术语：运行应用程序时，计算机执行编译后的代码。这些代码执行指令、创建变量、给变量赋值，然后释放它们。还记得吗，这些变量被存储在内存中。附加调试器意味着另一个应用程序（一个调试器）将监视第一个应用程序使用的内存。调试器能够解释应用程序使用的指令和内存，并以人可以理解的方式，告诉开发人员正在发生的情况。

本章前面介绍的方法需要在执行前修改代码，并在执行后查看输出。当您需要在执行期间监视和修改命令时，附加调试器将很有帮助。图 20.16 说明了该方法与前面介绍的方法之间的差别。

请注意，您也可以在调试器中运行应用程序，因此无需在程序已经运行后附加调试器。

由于 ASP.NET 是编译型的，因此调试 ASP.NET 页面时，可以使用与传统应用程序调试工具类似的工具。具体地说，您将使用 .NET 框架 SDK 中自带的 CLR 调试器。

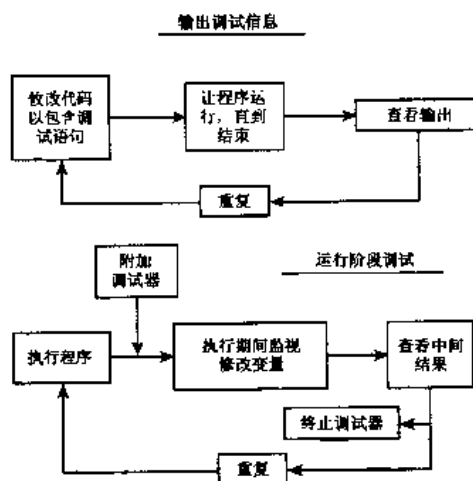


图 20.16 附加调试器让您能够监视执行过程中每一步的情况

### 20.4.1 使用 CLR 调试器

使用调试器之前，必须通过 web.config 文件在应用程序中启用调试功能，将下述代码行加入到应用程序目录下的 web.config 文件中。

```
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
</configuration>
```

上述代码为应用程序生成一个符号文件（symbol file，.pdb），该文件告诉调试器如何将必须解释的机器指令映射到源文件中的代码。这让您能够通过查看源代码而不是晦涩的机器代码来跟踪程序的执行情况。

当发生错误时,上述代码将在 ASP.NET 页面中显示一些基本的调试信息,这已经在本章前面的“调试简介”一节中介绍过了。

**警告:**一旦完成调试后,便应该在web.config文件中关闭调试选项。编译应用程序时,如果其调试选项被启用,将降低其运行速度。

让我们开始介绍 CLR 调试器。该文件名叫 DbgUrt.exe,默认情况下位于 C:\Program Files\Microsoft.NET\FrameworkSDK\GuiDebug。双击该文件,将打开如图 20.17 所示的窗口。

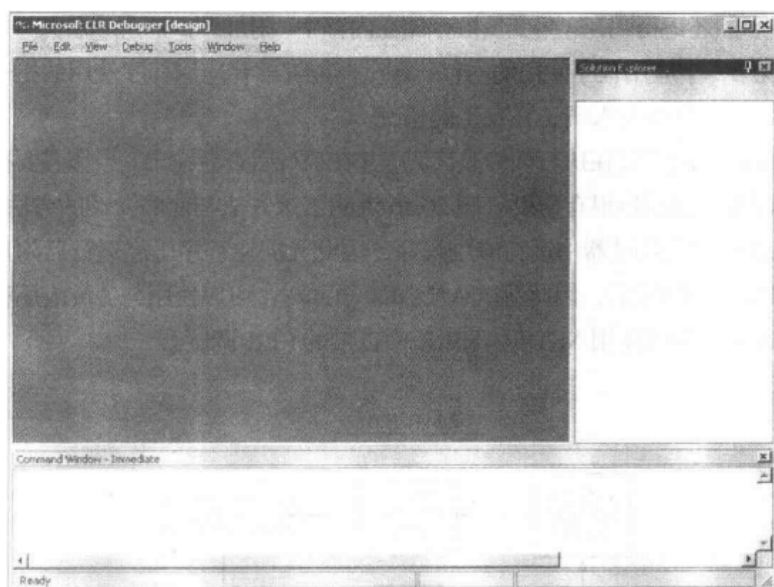


图 20.17 CLR 调试器的界面

打开调试器后,调试应用程序的工作将包括 4 个步骤:

1. 打开要调试的文件;
2. 将调试器附加到 ASP.NET 进程中;
3. 设置断点;
4. 使用调试器的工具来操纵应用程序。

假设您已经将清单 20.7 保存为 Listing2007.aspx,在调试器中打开该文件,并让它正常执行。这将运行 ASP.NET 进程和应用程序——如果它们还没有运行的话。在 CLR 打开该文件,方法是单击菜单 File/Miscellaneous/Open File,然后选择相应的源文件。您将在 CLR 调试器中看到源代码,而文件将出现在调试器窗口左边的 Solution Explorer 窗格中。

为将调试器附加到 ASP.NET 进程中,可以单击菜单 Tools/Debug Processes。您将看到如图 20.18 所示的窗口。

单击 Show System Processes,并找到进程 aspnet\_ewp.exe。然后选择该进程,并单击 Attach 按钮。在弹出的窗口中,选中复选框 Common Language Runtime。在窗口的底部,将可以看到应用程序的名称 (tyaspnet21days)。单击 OK 按钮,关闭 Processes 窗口。

现在 CLR 调试器中将出现一个名叫 Disassembly 的新窗口。该窗口显示了计算机见到的解释后的指令。

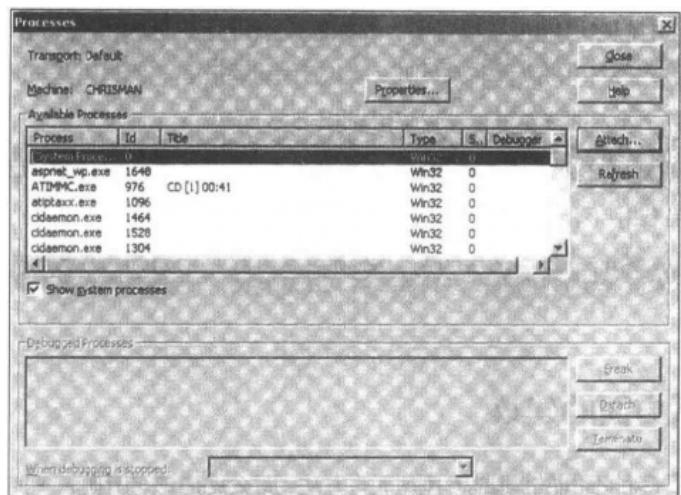


图 20.18 将调试器附加到 ASP.NET 进程中

**警告：**将调试器附加到进程中时，相应的应用程序将被冻结（不可用），同时未保存的信息可能丢失。ASP.NET 也一样。因此，将调试器附加到进程中时，一定要确保不封锁任何生产应用程序。

**新术语：**断点是代码中的一个位置，执行到这里时将暂停。断点对于调试代码中的特定行很有帮助。例如，如果您知道某一行会引发错误，则可以在该行之前设置一个断点，并使用调试器来分析指令和变量。

要设置断点，只需单击文件源代码左边的列。这样将出现一个红点，意味着执行到这里时将暂停（如果红点中出现问号，则可能是您没有在 web.config 中启用调试功能）。将鼠标指向断点，系统将显示该断点的准确位置，如图 20.19 所示。

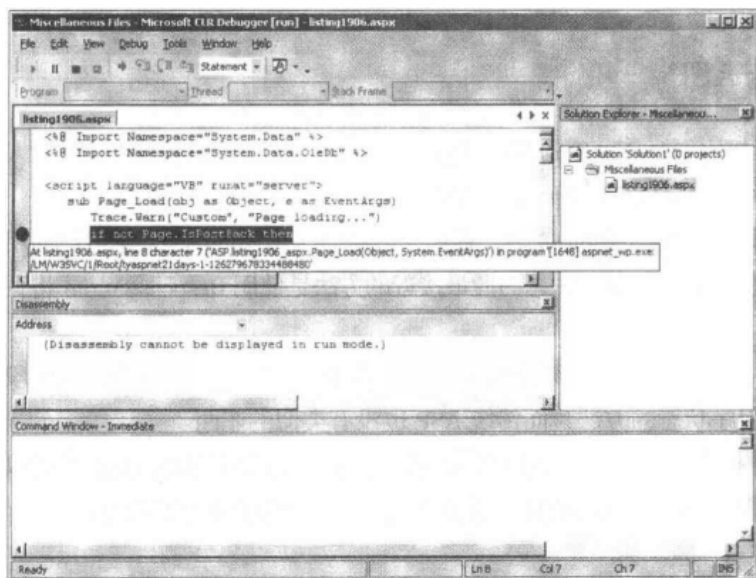


图 20.19 在源代码中设置一个断点后，则执行到这一行时将暂停

现在，在浏览器中再次请求 listing2007.aspx。执行到断点时，应用程序将停止，由调试器接管控制权。

在 CLR 调试器窗口的顶部，是一些让您能够控制程序执行的控件，如移到下一行或停止调试。调试器是一个非常复杂的工具，其功能非常多。更详细的信息，请参考 .NET 框架 SDK 文档。

调试编译后的应用程序可能是一场噩梦，因为您无法停止执行而不丢失信息，也无法实时地监视变量。这使得找到错误发生的位置非常困难，当应用程序非常复杂时更是如此。CLR 调试器为调试 ASP.NET 页面提供了上述所有功能，这使得调试非常简单。

注意：CLR 调试器并不能附加到所有的 ASP.NET 应用程序中，它能够附加到或调试任何已经运行的 .NET 框架应用程序（换句话说，应用程序必须是在 .NET 框架下开发的，例如您无法使用该调试器来调试 Windows 95）。需要记住的是，必须在应用程序中启用调试功能，即它必须有符号文件。

## 20.5 有关调试方面的建议

ASP.NET 提供了多种不同的调试应用程序的方法，那么您应该使用哪一种呢？在什么地方使用呢？

如果无法确定错误的位置，则最好使用跟踪。在代码的不同位置加入 `Trace.Write` 和 `Trace.Warn` 语句，并注意哪些语句执行了，哪些没有。通常这让您能够找到错误的位置，并做相应的处理。

如果对编写的代码没有把握，特别是当您不希望用户看到错误时，则应该使用 `try` 和 `catch` 捕获错误并做妥善的处理，以免用户感到意外。

最后，当您需要实时的跟踪程序的执行情况时，应使用 CLR 调试器。您可以进入到每一行代码中，查看变量以确定它们被赋给了正确的值。对于查找并更正错误而言，这是一种非常强大的工具，但它也有一些副作用，如丢失数据以及带来额外的开销。当您把调试器附加到某个进程（如 ASP.NET）中后，所有的执行都被冻结，这意味着它不能正确地运行——这可能给应用程序和用户带来问题。

## 20.6 这不是 ASP

与传统 ASP 相比，ASP.NET 的调试功能无疑要强一些。许多传统 ASP 开发人员熟悉调试方法 `Response.Write`，而在 ASP.NET 完全支持这种方法

但现在有更好的应用程序调试方法。跟踪工具提供了大量的信息，您可以使用它们来调试和分析应用程序的性能。在传统 ASP 中，与此最接近的方法是使用 `Response.Write` 将页面使用的信息输出。

另外，由于 ASP.NET 页面是编译型的，因此您可以使用诸如 CLR 等运行阶段调试器。该工具提供了监视应用程序状态的方法，而传统 ASP 也没有类似的东西。

与传统 ASP 相比，ASP.NET 支持的调试技术得到了极大的提高，现在您可以使用真正的调试工具，而不仅仅是 `Response.Write` 提供的有限的信息。这让您能够创建出更强大的、没有 bug 的应用程序。

## 20.7 总结

本章介绍了大量的内容。ASP.NET 中的调试工作非常复杂，但其提供的工具将帮助您轻松、迅速地找到并修复任何令人讨厌的 bug。

一种调试方法是使用 `Response.Write` 方法，在执行页面时将调试信息输出。这种方法可以帮助您跟踪程序的执行情况并找到错误，但不幸的是，您需要手工添加和删除这些语句，以免最终用户看到这些私有信息。

您可以使用 `try` 和 `catch` 语句来捕获错误并妥善地处理它们，以便程序能够继续执行，而不会有任何问题。您还可以使用关键字 `throw` 来引发自定义的错误。

与使用 `Response.Write` 相比，跟踪工具提供了一种更佳的调试方法。该工具提供了关于页面的详细统计信息，包括页面的执行时间、页面使用的对象、表单和 HTTP 报头信息以及服务器变量。可以在页面编译指令中加入 `trace="true"` 来启用跟踪功能。

您可以使用 `Response.Write` 和 `Trace.Warn` 将自定义的调试信息写入到跟踪日志中。这些语句与 `Response.Write` 类似，只不过它们不生成任何客户输出，因此调试完页面后无需删除它们。这些方法的输出只影响跟踪日志，而且只有在跟踪功能被启用时才会影响；否则根本不会影响应用程序。

您也可以在 `web.config` 文件中加入 `<trace enabled="true"/>` 来一次性启用整个应用程序的跟踪功能。您甚至可以通过在服务器上请求 `trace.axd` 来查看总体的统计信息和每个页面的跟踪输出。

最后，本章介绍了 CLR 调试器，一种非常有用的、用于实时监视程序执行情况的工具。使用该工具需要将其附加到要监视的 ASP.NET 应用程序中，这可能引发一些严重的副作用。要使用 CLR 调试器，必须首先在 `web.config` 文件中加入 `<compilation debug="true"/>`。

附加调试器后，便可以设置断点，并逐行执行代码，以监视任何变量及其值。

下一章介绍 ASP.NET 安全方面的知识，对于开发人员来说，这始终是一个非常重要的问题。ASP.NET 提供了大量不同的方法，来确保应用程序的安全，下一章将介绍其中的每一种方法。

## 20.8 问与答

问：在 `try` 语句块中所做的修改将被保留还是被撤销？

答：由于将在 `try` 语句块后继续执行，因此所做的修改将被保留。要撤销 `Try` 语句中所作的修改，请使用事务。

问：有办法将跟踪信息写入到一个永久性的日志中吗？

答：不幸的是，不能直接这样做。但可以使用第 13 章介绍的文件 I/O 方法可以捕获自定义的跟踪信息。只需将输出的 HTML 文件保存，即可捕获所有的跟踪信息。

## 20.9 作业

下面的作业帮助巩固本章介绍的概念，答案见附录 A。

### 20.9.1 小测验

1. 下面的代码段能运行吗？

```
try
    objCmd.Connection.Open
    objReader = objCmd.ExecuteReader
end try
```

2. 判断正误：`try...catch` 语句块中必须包含 `finally` 语句。



3. 跟踪工具是什么？如何在页面级启用它？如何在应用程序级启用它？
4. `Trace.Write` 和 `Trace.Warn` 之间的区别何在？
5. 判断正误：`Trace.Write` 和 `Trace.Warn` 将信息显示给客户。
6. 何为符号文件？
7. 判断正误：将调试器附加到应用程序中是非常安全的，不会影响应用程序的使用。
8. 何为断点？它有何用途？

#### 20.9.2 练习

1. 使用跟踪工具来比较应用程序的执行时间。确定禁用视图状态是否会加快执行速度，使用 `OleDbDataReader` 和 `DataSet` 之间的差别何在？注释是否会影响执行时间。
2. 使用 CLR 调试器以步进的方式执行几个页面。在应用程序执行时，使用 Watch 窗口来监视变量。尝试故意引入一些错误，以查看调试器中发生的情况。



## 第 21 章

# 确保 ASP.NET 应用程序的安全

Web 安全是一个复杂的主题，常令开发人员和最终用户感到困惑，但在当前的 Internet 中，这又是一个必须解决的问题。安全涉及到验证用户的证件（认证）以及确定其对资源的访问权限（授权）。在 ASP.NET 中实现上述措施的方法很多，而且其中的任何一种实现起来都不复杂。

ASP.NET 使得确保应用程序的安全非常容易。掌握安全方面的基础知识后，在 ASP.NET 中实现必须的安全措施将是小菜一碟。

本章包含以下内容：

- 何为 Web 安全？
- Windows 如何处理安全方面的问题？
- 三种不同的认证方式及其实现方法。
- 如何控制对服务器资源的访问？
- 何为模拟？如何使用它？

### 21.1 安全基础

默认情况下，大多数 Web 站点都允许匿名访问，也就是说，任何用户 Internet 连接的人都可以进入该站点并查看其中的页面。用户无需认证（验证其身份），便可以访问站点上的任何文件。假设您创建一个站点，成员必须支付订阅费后才能查看其中的内容（可能是权威人士提供的个股推荐服务）。如果不采取安全措施，则任何人都可以查看提供专家关于股票看法的页面。这对您的企业而言实在是太可怕了——在可以不付费的情况下，用户为何愿意付费呢？Web 安全就是为限制只有特定用户群才可以访问某些文件而设计的。

假设在一个极机密的政府机构供职，任何人都可以进入办公楼的大堂，但只有经过授权的人才能穿过大堂。每当需要进入机密房间时，都需要通过视网膜扫描来获得适当的授权。

**新术语：**这一过程说明了 Web 安全处理的基本原理。第一步是认证（authentication），即对于请求信息的用户验证其身份。用户使用其证件来表明其身份，证件的形态各种各样（最常用的是用户名和密码）。认证确保用户的身份是名副其实的。如果安全系统不能根据用户的证件确定其身份，则认证失败，用户将被拒绝访问。如果证件有效，则用户将被允许进入系统，并被赋予一个合法的、已知的身份。

**新术语：**用户被赋予身份后，系统将确定它能够访问的资源，这一过程被称为授权（authorization）。例如，在高度机密的政府机构的情形中，您必须获得授权才能进入某些房间。系统根据赋予您的身份的权限来进行授权。您可能被允许进入某些房间，而不允许进入其他的房间。在Web中，某些用户可能有权访问某些特定的文件，而其他用户则有权访问另一些文件。

**新术语：**最后，最后一步是模拟（impersonation）。假设政府大楼的安全系统是门口的警卫，警卫能够轻松地避开视网膜扫描，进入大楼的每一个房间，并且可以打开任何一个房间的大门。假设一个聪明的间谍毁坏了该安全系统，则他将可以进入任何一个房间，因此安全系统将失效。

作为另一种安全措施，安全系统的设计人员添加了模拟特性。当有人想进入大楼或计算机时，警卫将扮演这个人，从而限制自身的访问权限。

换句话说，警卫甘愿放弃其权利，防止黑客使用这种权力。这与军人被敌人抓获并审问时吞食一种神奇的药丸，以忘记一些高度机密的档案的情况类似。这个概念有些抽象，当您将其用于ASP.NET 页面时，其意义将更为明确。图 21.1 说明了用户为获得安全资源的访问权需要经历的步骤。

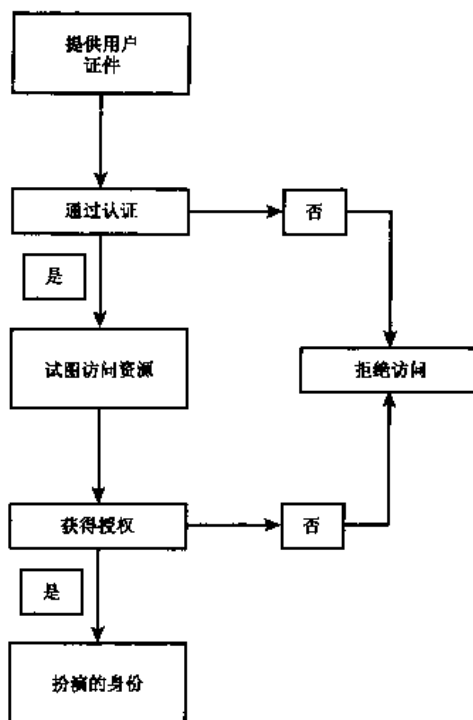


图 21.1 典型的安全规程

基本上，ASP.NET 中的安全性是通过两种不同的身份实现的。您可以通过将证件和操作系统的身份进行比较（通过 IIS），或将其与数据源中的权限（如通过 web.config）进行比较，来认证和审定证件。前一种方法所需的编码量（或对 ASP.NET 页面的修改量）非常少，但提供认证控制能力很小；而第二种方法需要的编码量更多，但提供了更大的灵活性。我们将在本章后面关于安全性的章节中讨论这些方法。

## 21.1 Windows 中的安全性

**新术语：**Windows 操作系统支持基于角色的安全（role-based security）。角色（role）定义了一类身份，例如，如果您负责维护计算机系统以及安装硬件和软件，则您扮演的是管理员的角色。如



















































































































































































































































































































































































































































































































