CHAPTER 6

.NET 4.0 Web 应用程序开发技术第 6 章

学习要点
（1）掌握 C#语言编程技术。 （2）了解 ASP.NET 的运行模式。 （3）熟悉 ASP.NET 的服务器对象。 （4）掌握 ASP.NET 基本控件的使用。 （5）学会配置 Web.config 文件。 （6）掌握 ADO.NET 数据库访问技术。 （7）学会使用 VS 2010 创建 Web 服务。 （8）掌握进行.NET 控件开发的技术。

ASP.NET 是微软推出的全新体系结构.NET 平台的一部分，它提供了一种以.NET Framework 为基础，开发 Web 应用程序的全新编程模式，和现有各种 Web 开发模式相比，ASP.NET 可以使 Web 开发人员更快捷和方便地开发 Web 应用程序。

ASP.NET 相比 ASP（Active Server Pages）脚本语言有了革命性的改进。从底层来说，ASP.NET 完全基于.NET 框架、完全面向对象，不仅更易于创建动态 Web 内容，还易于创建复杂和可靠的 Web 应用程序如 Web 服务等。

在 ASP 中，每一个功能都需要开发人员单独编写代码完成。为了构造一个完整的 Web 页面，并控制不同的部分完成相应的功能，开发人员需要编写大量的代码。而 ASP.NET 提供了丰富的服务器端控件，开发人员只需要选用合适的控件并且设置和调整其属性，就可以实现很多原来 ASP 中需要大量编码的功能。不仅如此，ASP.NET 还支持用户控件和自定义控件，进一步提供更加丰富完整的控件支持，简化开发人员的工作，使其把大量精力放在核心业务代码的处理上。

在 ASP 中使用 ADO（ActiveX Data Object）来访问数据库，它是面向连接的数据访问方式；而在 ASP.NET 中采用 ADO.NET 来访问数据库，它是一种无连接的基于消息机制的数据访问方式。在 ADO.NET 中，数据源的数据可以作为 XML 文档进行传输和存储；数据可以脱离源数据库进行各种操作，在需要时连接数据库将更新过的数据写到数据库中。

在 ASP 中采用弱化类型支持的 VBScript 和 JScript 两种脚本语言，而在 ASP.NET 中采

用强类型语言 VB.NET、C#等，采用完全面向对象方式编程。

在 ASP 中无法开发一个 Web Service，而在 ASP.NET 中建立一个 Web 服务是相当方便的，可用 Web 服务来为异构系统实现数据交换。

总之，ASP.NET 完全基于模块和组件，具有更好的可扩展性与可定制性，其特性远　远超越了 ASP，给 Web 开发人员提供了更好的灵活性，有效缩短了 Web 应用程序的开发周期。

图 6-1 所示为 ASP.NET 的体系结构。ASP.NET 和 ASP 一样通过 ISAPI（Internet Server Application Programming Interfaces）与 IIS 通信。事实上，ASP 和 ASP.NET 可以共存于 IIS 服务器上。ASP.NET 中有一个 cache，用来作为页面的缓存，用以提高性能。另外 ASP.NET 还包括一个跟踪用户会话的状态管理服务。

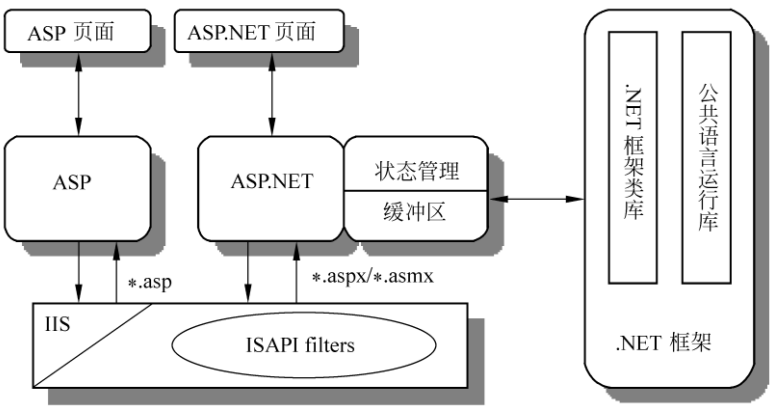


图 6-1　ASP.NET 体系结构

要学好 ASP.NET 的编程，必须掌握一种编程语言。本章将首先介绍 C#语言，为读者编程提供参考（详细的 C#编程方法请看专门书籍）；然后介绍了主要 Web 服务器标准控件、服务器端验证控件和 ASP.NET 内置服务器对象等的使用，介绍了 Web.config 文件的作用和常用配置参数；重点介绍了 ADO.NET 数据库访问技术以及执行存储过程、数据库的事务处理、跨数据库访问、数据绑定技术等内容；给出了创建和访问 Web 服务的实例；以案例方式对.NET 中的代码重用实现技术，包括 Web 开发中的类库构建与访问、Web 自定义控件和工具箱控件开发等进行了初步介绍；最后实例演示了如何在 ASP.NET 中进行 XML 编程。

6.1　C#语言初步

6.1.1　C#程序的基本结构

C#是由 C 和 C++发展而来的面向对象和类型安全的编程语言。C#读作 C Sharp，它和 Java 非常相近，其目标在于把 Visual Basic 的高生产力和 C++本身的能力结合起来。下面我们通过一个 C#语言的简单例子来了解一些概念和用 C#语言编写程序的方法。

【例 6.1】 将下列代码存为 C:\Windows\Microsoft.NET\Framework\v2.0.50727\hello.cs 文件。此处 Windows 目录为 Window 操作系统的安装目录。然后进入上述文件夹，在命令行输入 csc　hello.cs 进行编译后，就会生成一个名为 hello.exe 的可执行程序，运行它就会出现

"Hello, world"。

```
using System;
class Hello
{  static void Main() {
       Console.WriteLine("Hello, world");
   }
}
```

下面对这个程序进行一些说明：

（1）C#程序的源代存储在扩展名为.cs 的文件中。

（2）using System 指令涉及到一个叫做 System 的**名称空间**（Namespace），又翻译成**命名空间**，这是在 Microsoft.NET 类库中提供的。这个名称空间包括在 Main 方法中使用的 Console 类。名称空间提供了一种用来组织一个类库的分层方法。分层的类库之间用操作符“.”表示上下级的分层关系。使用“using”命令后，就可以无障碍地使用名称空间中的各种类型成员。.NET 中的名称空间概念和前面 XML 中所说的名称空间不太相似，XML 中所说的名称空间为了避免名称冲突，可以确保各个用户定义的相同标记元素在各个名称空间下具有不同的含义，而此处的名称空间除了具有避免名称冲突之功能外，更重要的是引用了名称空间后，你就可以在你的程序代码中方便地使用系统提供的各种类库成员，例如：用 C#语言实现复制一个文件的代码为：

```
System.IO.File.Copy("c:\\1.cs","c:\\2.cs",true );
```

可以理解为 System 名称空间下有一个 IO 名称空间，IO 名称空间下又有一个 File 名称空间，每个名称空间下都有许多系统提供的可直接调用的方法。调用方法时，如果你不想打入全称，可使用"using"指示符。这样上述代码可以写成：

```
using System.IO.File;
…
Copy("c:\\1.cs","c:\\2.cs",true );
```

同样，例 6.1 中 Console 是 System 名称空间中的成员类，该类有个 WriteLine 的方法。该方法的全称是：System.Console.WriteLine(...)，因为用了 using，所以上述代码就简写成 Console.WriteLine(...)。

（3）Main 方法是类 Hello 中的一个成员，它有 static 的说明符，所以它是类 Hello 中的一个方法而不是此类中的实例。 Main 方法是应用程序的主入口点，也称作开始执行应用程序的方法。

（4）C#语言编译器 CSC.EXE 只是将程序员编写的代码编译成 MSIL（Microsoft Intemediate Language）中间语言。中间语言在安装时被运行库编译成本机代码或者首次运行时被实时编译。因此例 6.1 中的 hello.exe 只是一个由中间语言可执行文件头组成的可移植执行文件。

6.1.2　C#中的数据类型

C#支持两种类型：数据类型和引用类型。数据类型包括一些简单类型（例如 char、int 和 float）、枚举类型和结构类型。引用类型包括类类型、接口类型、代表（delegate）类型和数组类型。

数据类型和引用类型的区别在于，数据类型变量直接包含它们的数据，而引用类型变量是存储对于对象的引用。对于引用类型，有可能两个变量引用相同的对象，因而可能出现对一个变量的操作影响到其他变量所引用对象的情况。对于数据类型，每个变量都有它们自己对数据的复制，所以不太可能因为对一个进行操作而影响到其他变量。

【例 6.2】 数据类型和引用类型示例。

```
1   using System;
2   class Class1 {
3      public int Value = 0;
4   }
5   class Test {
6      static void Main()
7      {
8         int val1 = 0;
9         int val2 = val1;
10        val2 = 123;
11        Class1 ref1 = new Class1();
12        Class1 ref2 = ref1;
13        ref2.Value = 123;
14        Console.WriteLine("Values: {0}, {1}", val1, val2);
15        Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
16     }
   }
```

此例分别输出 0，123 和 123，123。对局部变量 val1 的赋值没有影响到局部变量 val2，因为两个局部变量都是数据类型（int 类型），并且每个数据类型的局部变量都有它们自己的存储地址。与此相对的是，对于 ref2.Value 的赋值 ref2.Value=123 对 ref1 和 ref2 都有影响。从第 14 和 15 行可看到 Console.WriteLine 各使用了 2 个变量来实现字符串格式输出。其中占位符{0}和{1}各指向一个变量。占位符{0}指向第一个变量，占位符{1}指向第二个变量。在输出被送到控制台前，这些占位符会被它们相对应的变量所替换。

开发者可以通过枚举和结构声明定义新数据类型，可以通过类、接口和委派声明来定义新引用类型。

6.1.3　C#变量声明及其初始化

C#语言是一种强类型的语言，在使用变量前必须对该变量的类型进行声明。虽各种类型的变量都有默认值，但为了方便后面的程序调试，建议在声明变量时就对变量进行初始化。

变量声明及其初始化方法如表 6-1 所示。

表 6-1　C#变量声明及其初始化

（1）值类型

整　型	取 值 范 围	.NET 框架基类	声明及初始化
sbyte	–128～127	System.SByte	sbyte a=1,b=5,c; //有符号 8 位整数
byte	0～255	System.Byte	byte a=1,b=2; //无符号 8 位整数
short	–32768～32767	System.Int16	short a=60; //有符号 16 位整数
ushort	0～65535	System.UInt16	ushort a=60; //无符号 16 位整数
int	-2^{31}～$2^{31}-1$	System.Int32	int a=10,b=60; // 有符号 32 位整数
uint	0～$2^{32}-1$	System.UInt32	uint uInt1 = 123;// 无符号 32 位整数
long	-2^{63}～$2^{63}-1$	System.Int64	long l=4294967296L; //有符号 64 位整数
ulong	0～$2^{64}-1$	System.UInt64	ulong u=92233720; //无符号 64 位整数
浮点型	**取 值 范 围**	**.NET 框架基类**	**初　始　化**
float	±1.5*10^–45～±3.4*10^308	System.Single	float x=3.5F;//默认情况下实数被视为 double。使用后缀 f 或 F 可转化为浮点型
double	± 5.0*10^–324 ～±1.7*10^308	System.Double	double x=5.0,y = 3d; //希望整数被视为 double，使用后缀 d 或 D
decimal	± 1.0*10^–28 ～±7.9*10^28	System.Decimal	decimal myMoney = 300.5m; //希望实数被视为 decimal 类型，则使用后缀 m 或 M
布尔型	**取 值 范 围**	**.NET 框架基类**	**初　始　化**
bool	True 和 false	System.Boolean	bool test=true,flag=(100<110)
字符型	**取 值 范 围**	**.NET 框架基类**	**初　始　化**
char	特殊整型，16 位单 unicode 字符	System.Char	char c1='B',c2='\x0066' //16 进制转义 int myInt=(int)'A'

C#中采用转义字符来代替一些特殊的字符，常用的有单引号（\'）、反斜杠（\\）、退格（\b）、回车（\r）、换行（\n）、空字符（\0）、双引号（\"）、感叹号（\a）、制表符（\t）等

（2）引用类型

Object 对象是所有类型的父类型，是 C#中所有其他类的基类，可赋予任何类型的值，例如：
object　myobj=80; 或者　object obj=new object();

string 字符串类型表示 Unicode 字符的字符串，是.NET Framework 中预定义类的 System. String 的别名，例如：string a = "good " + "morning";

数组是一种包含若干变量的数据结构，所有数组元素可以通过数组名和下标来访问。下面举例说明数组的初始化方法。

```
string [] myarray = { "x", "y", "z" };                  //声明一个有 3 个元素的数组
int[] myarray = new int[100];                           //声明一个数组，没有初始化
int[,] myarray = {{100,10},{101,11},{102,12}};          //初始化一个二维数组,等价于下面两行
myarray[0,0]=100; myarray[1,0]=101; myarray[2,0]=102;
myarray[0,1]=10; myarray[1,1]=11; myarray[2,1]=12;
```

续表

类类型定义了一个数据结构，它包括数据成员和函数成员，它支持继承和多态。C#中预定义的类类型有：System.Object（所有其他类型的最终基类）、System.String（C#语言字符串类型）、System.Valtype（所有值类型的基类）、System.Enum（所有枚举类型的基类）、System.Array（所有数组类型的基类）、System.Delegate（所有委托类型的基类）、System.Exception（所有异常类型的基类）。下面代码定义了一个类 mycode，它包含一个变量和一个方法：

```
class mycode  {    //定义了一个类 mycode
  public double cd;  //定义了类 mycode 的一个变量
  public double getcd() {  //定义了类 mycode 的一个方法
    return this.cd;
    }
}
```

引用类型中的接口类型和委托类型此处不作介绍，读者可参看其他 C#方面的资料

变量名必须以英文字母或@开头，由字母、数字、下划线组成，不能有空格、标点、运算符号、C#中关键字名、C#中库函数名，且大小写敏感。C#中有静态变量、引用参数、数组变量、实例变量、值参数、输出参数和局部变量 7 种变量类型，如表 6-2 所示。

表 6-2　C#变量类型

变 量 类 型	说　　明
静态变量	用 static 修饰符声明的变量，如 static double=1.0。当静态变量所属的类被加载后，静态变量就一直存在，并且所有属于这个类的实例都共用同一个变量
实例变量	未用 static 修饰的变量。它们属于类的实例。当创建该类的新实例时，实例变量开始存在；当所有对该实力的引用结束后，该实例变量终止
引用参数	用 ref 修饰符声明的参数。引用参数的值与被引用的基础变量相同，所以引用参数不占用、不创建新的存储位置
输出参数	用 out 修饰符声明的参数。它表示函数调用中的基础变量，不创建新的存储位置
值参数	未用 ref 或 out 修饰符声明的参数。在调用参数所属的函数成员时开始存在，当返回该函数成员时值参数终止
局部变量	在某个独立的程序块中声明的变量。作用域仅限于此程序块。如 for、switch 语句等

C#中定义常量用 const 修饰符，例如“public const double y=1.234;”。枚举类型是由一组特定常量构成的一种数据结构，是值类型的特殊形式。当需要一个有指定常量集合组成的数据类型时，可使用枚举类型。枚举类型不能实现接口、不能定义方法、属性、事件。

【例 6.3】 枚举类型示例。

```
using System;
enum enumChoice { A = 1, B = 2, C = 3, D = 4 }  //定义了一个枚举类型
class Choice {
   public static void Main()  {
      choice = (int)enumChoice.A;        //演示了使用的方式
      Console.WriteLine("你选择的是: {0}", choice); //控制台输出"你选择的是: 1"
   }
}
```

6.1.4　C#表达式

表达式是可以计算且结果为单个值、对象、方法或命名空间的代码片段。表达式可以包含文本值、方法调用、操作符及其操作数或简单名称。与 C、C++相同，C#的表达式大致包含了算术表达式、赋值表达式、关系表达式和逻辑表达式。

（1）算术表达式

用算术操作符把数值连接在一起的、符合 C#语法的表达式称为算术表达式。算术操作符包括+、–、*、/、%、++、– –。其中模运算%表示整除取余数；++分为前缀增 1（先加 1 后再使用）和后缀增 1（先使用后再加 1）；– –分为后缀增 1（先减 1 后再使用）和后缀减 1（先使用后再减 1）。另外 C#还提供了专门针对二进制数据操作的运算符，包括以下六个：&（与）、|（或）、^（异或）、～（补）、<<（左移）、>>（右移）。

（2）赋值表达式

赋值操作符用于为变量、属性、事件或索引器元素赋予新值。赋值操作符的运算对象、运算法则及运算结果如表 6-3 所示。

表 6-3　C#赋值操作符的运算规则

操　作　符	名　　称	运 算 对 象	运 算 结 果
=	赋值	任意类型	任意类型
+=、– =、*=、/=	加、减、乘、除赋值	数值型（整型、实型等）	数值型（整型、实型等）
%=	模赋值	整型	整型
&=、!=、 >>=、<<=、~=	位与、位或、右移、左移、异或赋值	整型或字符型	整型或字符型

（3）关系表达式

= =、!=、<、>、<=和>=等操作符称为关系操作符。用关系操作符把运算对象连接起来并符合 C#语法的式子称为关系表达式。关系表达式要么返回 true 要么返回 false。C#中还定义了 is 操作符，其格式为：A(值) is B(类型)，意义是如果 A 是 B 类型或者 A 可以转化为 B 类型则返回 true，否则为 false。

（4）逻辑表达式

&&（and）、||（or）和!（not）操作符称为逻辑操作符。用逻辑操作符把运算对象连接起来并符合 C#语法的式子称为逻辑表达式。

6.1.5　C#控制语句

C#中控制语句主要包括分支和循环语句。分支语句有 3 种：①三元运算符，例如：a=(b>5)?100:10 表示 b>5 时 a=100，否则 a=10；②if 语句；③switch 语句。循环语句有 4 种：①已知步长的 for 语句；②foreach 语句；③while 语句；④do while 语句。它们的语法结构如表 6-4 所示。其中 switch 语句可一次将测试变量与多个值比较，而 if 仅仅测试一个条件。对于循环语句可用 break 和 continue 语句决定是否跳出循环或继续执行循环。foreach 语句可以遍历一个集合中的所有元素。

表 6-4　C#的分支/选择语句及循环语句

If	Switch	For	while	foreach	do while

if(…)
{
…
}
else
{
…
}
	switch(控制表达式) {
	case 测试值 1:
	语句 1
	break;
	case 测试值 2:
	语句 2
	break;
	default:
	默认语句
	break;
	}
		for(int i=0;i<10;i++)
		{
		…
		}
			int i=0;
			while(i<10)
			{
			…
			i++;
			}
				char[] person=new char[]{'0','1','2','3'};
				foreach(char i in person)
				{
				if(i=='1')
				{ … }
				else if(i=='2')
				{ … }
				else
				{ … }
				}
					do {
					内嵌语句
					} while(循环控制条件)

另外可用 try-catch-finally 语句来捕捉异常，其使用方法和 JavaScript 语言中的语法相类似。在 Web 开发中应尽量处理各种可能性，少用捕捉异常来实现某些功能。

6.1.6　C#类声明

类声明定义新的引用类型。一个类可以从其他类继承。类是一种将数据成员、函数成员和嵌套类型等进行封装的数据结构。它在面向对象基础上引入了接口、属性、方法、事件等组件特性。其数据成员可以是常量或域，函数成员可以是方法、属性、索引、事件、操作符或静态构造函数和析构函数。构造函数在创建对象时被自动调用，用来执行对象的初始化操作，其函数名总是与类名相同。析构函数在释放对象时被调用，用来删除对象前做一些清理工作。

类中的每个成员都必须定义其被访问的范围，用类的访问修饰符来表示访问这个成员的程序文本的区域。类的访问修饰符有五种可能形式，如表 6-5 所示。

表 6-5　类的访问修饰符

序　号	形　　式	直 观 意 义
1	public	访问不受限制
2	protected	访问只限于此程序或类中包含的类型
3	internal	访问只限于此程序
4	protectedinternal	访问只限于此程序或类中包含的类型
5	private	访问只限于所包含的类型

下面简要介绍类成员的有关概念。

（1）常数

一个常数是一个代表常数值的类成员即某个可以在编译时计算的数值。只要没有循环从属关系，允许常数依赖同一程序中的其他常数。

（2）域

域是一个代表和某对象或类相关的变量的成员。域可以是静态的。只读域可以用来避免错误的发生。对于一个只读域的赋值，只会在相同类中的部分声明和构造函数中发生。

（3）方法

方法是一个执行可以由对象或类完成的计算或行为的成员。方法有一个形式参数列表（可能为空），一个返回数值（或 void），并且可以是静态也可以是非静态。静态方法　　要通过类来访问。非静态方法，也称为实例方法，通过类的实例来访问。方法可以被重复调用。

（4）属性

属性是提供对对象或类的特性进行访问的成员。属性的例子包括字符串的长度、字体的大小、窗口的焦点、用户的名字，等等。属性是域的自然扩展。两者都是用相关类型成员命名，并且访问域和属性的语法是相同的。然而，与域不同，属性不指示存储位置。作为替代，属性有存取程序，它指定声明的执行来对它们进行读或写。

属性是由属性声明定义的。属性声明的第一部分看起来和域声明相当相似。第二部分包括一个 get 存取程序和一个 set 存取程序。

（5）事件

事件是使得对象和类提供通知的成员。一个类通过提供事件声明来定义一个事件，这看起来与域和事件声明相当类似，但是有一个 event 关键字。这个声明的类型必须是 delegate 类型。

（6）操作符

操作符是一个定义了可以用来使用在类的实例上的表达式。

（7）索引器（indexer）

索引器是使得对象可以像数组一样被索引的成员。属性使类似域的访问变得可能，索引器使得类似数组的访问变得可能。索引器的声明类似于属性的声明，最大的不同在于索引器是无名的（由于 this 是被索引，所以用于声明中的名称是 this）。class 或 struct 只允许定义一个索引器，而且索引器总是包含单个索引参数。索引参数在一对方括号中提供，用于指定要访问的元素。

（8）实例构造函数（constructor）

实例构造函数是实现对类中实例进行初始化的行为的成员，是一种特殊的方法。它与类同名，能获取参数，但不能返回任何值。每个类都必须至少有一个构造函数。如果类中没有提供构造函数，那么编译器会自动提供一个没有参数的默认构造函数。

（9）析构函数（destructor）

析构函数是实现破坏一个类的实例的行为的成员。析构函数完成对象被垃圾回收时需要执行的整理工作，在碎片收集时会被自动调用。在 C#中，没有提供一个 delete 操作符，由运行库控制何时摧毁一个对象。

析构函数的语法是首先写一个～符号，然后跟上类名。析构函数不能有参数，不能带任何访问修饰符（比如 public），而且不能被调用。不能在一个 struct 中声明一个析构函数。

（10）静态构造函数

静态构造函数是实现对一个类进行初始化的行为的成员。静态构造函数不能有参数，不能有修饰符而且不能被调用，当类被加载时，类的静态构造函数自动被调用。

（11）继承（Inheritance）

继承是面向对象的一个关键概念，它描述了类之间的一种关系。假如多个不同的类具有大量通用的特性，而且这些类相互之间的关系非常清晰，那么使用继承就能避免大量重复的工作。类支持单继承，System.Object 类是所有类的基类。所有类都是隐式地从 System.Object 类派生而来的。

方法、属性和索引器都可以是虚拟（virtual）的，这意味着它们可以在派生的类中被覆盖（override）。

可以通过使用 abstract 关键字来说明一个类是不完整的，只是用作其他类的基类。这样的类被称为抽象类。抽象类可以指定抽象函数－非抽象派生类必须实现的成员。

（12）接口

接口定义了一个连接。一个类或结构必须根据它的连接来实现接口。接口可以把方法、属性、索引器和事件作为成员。类和结构可以实现多个接口。因为通过外部指派接口成员实现了每个成员，所以用这种方法实现的成员称为外部接口成员。外部接口成员可以只是通过接口来调用。

（13）委派

委派（delegates）是指向一个方法的指针。委派与 C++中的函数指针相似，与函数指针不同，委派是类型安全并且可靠的。委派是引用类型，它从公共基类 System.Delegate 派生出来。一个委派实例压缩了一个方法——可调用的实体。对于静态方法，一个可调用实体由类和类中的静态方法组成。

委派的一个有趣而且有用的特性是它不知道或不关心与它相关的对象的类型。对象所要做的所有事情是方法的签名和委派的签名相匹配，这使得委派很适合“匿名”调用，而这是个很有用的功能。定义和使用委派分为三步：声明、实例化和调用。用 delegate 声明语法来声明委派：delegate void SimpleDelegate();声明了一个名为 SimpleDelegate 的委派，它没有任何参数并且返回类型为 void。

（14）枚举

枚举类型的声明为一个符号常数相关的组定义了一个类型名称。使用枚举可以使代码更可读还可以自归档，所以使用枚举比使用整数常数要好。

【例 6.4】 声明一个类 MyClass，并演示如何对它进行实例化。

```
using System;
class MyClass
{
public MyClass() {  //构造函数
      Console.WriteLine("Constructor");
   }
public MyClass(int value) {
      MyField = value;
```

```
      Console.WriteLine("Constructor");
```

```
   }
~MyClass() {  //析构函数
       Console.WriteLine("Destructor");
   }
public const int MyConstant = 12;  //定义常量
public int MyField = 34;
public void MyMethod(){   //定义了一个方法
   Console.WriteLine("MyClass.MyMethod");
   }
public int MyProperty {  //定义类的属性
       get  { return MyField; }
        set { MyField = value; }
}
public int this[int index] {
       get { return 0;}
       set {
           Console.WriteLine("this[{0}] was set to {1}", index, value);
       }
   }
public event EventHandler MyEvent;

public static MyClass operator+(MyClass a, MyClass b) {
   return new MyClass(a.MyField + b.MyField);
   }
internal class MyNestedClass{}
}
//下面代码将访问上面定义的 MyClass 类
class Test
{    static void Main() {
     MyClass a = new MyClass();  //调用构造函数
     MyClass b = new MyClass(123);
      //常量的使用
      Console.WriteLine("MyClass.MyConstant = {0}", MyClass.MyConstant);
     a.MyField++; //域的用法
     Console.WriteLine("a.MyField = {0}", a.MyField);

     a.MyMethod(); //调用方法
     a.MyProperty++;  //属性设置
     Console.WriteLine("a.MyProperty = {0}", a.MyProperty);

     a[3] = a[1] = a[2];   //索引的使用
     Console.WriteLine("a[3] = {0}", a[3]);
     a.MyEvent += new EventHandler(MyHandler);  //调用事件
// Overloaded operator usage
```

```
    MyClass c = a + b;
  }
static void MyHandler(object sender, EventArgs e) {
    Console.WriteLine("Test.MyHandler");
  }
internal class MyNestedClass{}
}
```

选择“开始”|“程序”|Microsoft VS 2010 | Visual Studio Tools |“VS 命令提示(2010)”菜单，出现 VS 2010 控制台，输入 notepad 后回车，将上述代码复制到记事本中，存盘为 ex_6_4.cs，用 CSC ex_6_4.cs 编译后运行结果如图 6-2 所示。

图 6-2　MyClass 类运行效果

6.2　常用 ASP.NET 控件的使用

VS 2010 集成开发环境中已为 Web 的开发提供了一个工具箱，工具箱的目的是将控件进行分类管理，以便方便用户使用，工具箱分类情况如表 6-6 所示。

表 6-6　工具箱控件

序　号	控 件 分 类	功 能 描 述
1	Web 标准控件	和界面设计制作有关的控件
2	数据控件	数据访问、操作以及数据可视化方面控件
3	验证控件	对用户输入的内容进行验证的控件
4	导航控件	提供站点导航、动态菜单、树型菜单的控件
5	登录控件	用户登录界面的设计制作控件
6	Web Parts 控件	Web 门户定制控件。用户可以拖动某一区域在屏幕上重新布局
7	HTML 控件	HTML 中的常规控件
8	CrystalReports 控件	提供 Web 页面上的报表处理

Web 开发过程中的控件区分为客户端控件和服务端控件。Web 页面中的客户端控件不需要 Web 服务器参与处理，Web 服务器将带有控件的网页传送到客户端，由客户端的浏览器来处理。Web 页面中的服务端控件是需要 Web 服务器参与处理的控件，它们需要占用 Web

服务器的内存、CPU、磁盘缓冲区、硬件资源等来处理这些控件，最终转变成客户端控件，传送到客户端浏览器，由客户端的浏览器来处理。

VS 2010 工具箱中的 HTML 控件既可以作为客户端控件，又可以作为服务器端控件，其他控件全部为服务器端控件。要将 HTML 控件从客户端控件变成服务器端控件，只要加上 runat 属性即可。例如定义一个在服务器端执行的按钮：

```
<input id="button1" type="button" value="button" runat="server" />
```

将客户端HTML控件变成服务器端控件后，该控件的使用效果将和Web标准控件　　中某些控件类似。可以用 Web 标准控件代替服务器端的 HTML 控件，它们在编程中稍有区别。

在使用控件时可以直接用鼠标左键双击或将某个控件元素用鼠标拖到工作区内进行设计。这里工作区既可以是“设计”也可以是“源”。对于每个控件元素我们可以单击该控件，按下 F4 键，出现对应的属性窗口，在属性窗口中来设置它的外观，例如颜色、大小、字体等；可以单击属性窗口中的“闪电”图标定义该控件的事件响应程序。

ASP.NET 的常用控件主要包括服务器端标准控件、服务器端数据控件、服务器端验证控件。在这里我们不对每个控件元素一一进行介绍，只是对在设计中经常会使用到的一些控件元素进行简单介绍。这方面的专业书籍比较多，有兴趣的读者可参阅其他资料详细了解各个控件的使用。

6.2.1　服务器端标准控件

服务器端标准控件是最常用的控件。在 ASP.NET 应用程序中，服务器端控件是 ASP.NET 内置控件。在添加一个服务器端控件时，在“源”中会自动添加“runat=server”属性，以便与客户端控件相区别，而且标记的前缀用“asp:”，例如定义一个标签控件的页面代码为：

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

服务器端的控件有一些共性的属性，具体说明如表 6-7 所示。

表 6-7　服务器端的控件的共性属性

序　号	属　　性	描　　述
1	Font	该属性为 Font 类型值，用于设置控件显示文本的字体属性。包括是否粗体 Bold、斜体 Italic、字体 Name、上划线 Overline、下划线 Underline、字体大小 Size、醒目线 Strikeout，等等
2	ForeColor	该属性为 Color 类型值，用于设置控件的前景颜色，即显示文字的颜色
	BackColor	设置或返回控件的背景颜色
3	Height,Width	分别获取或设置控件的高度和宽度
4	BorderColor	设置或返回控件的边框颜色
5	BorderStyle	设置或返回控件的边框样式
6	BorderWidth	设置或返回控件的边框的宽度

续表

序　号	属　　性	描　　述

7	AccessKey	获取或设置该控件的键盘快捷键。例如将按钮控件的该属性设为 D 后，用户可用 Alt+D 键盘直接运行按钮的单击事件
8	TabIndex	获取或设置控件的 Tab 键顺序。按下 Tab 键时，依据此值的大小顺序移动控件的焦点
9	Tooltip	获取或设置将鼠标放在该控件上所显示的一个动态提示框文本
10	Visible	获取或设置该控件是否可见
11	Enabled	获取或设置该控件是否处于激活状态
12	EnableViewState	在服务器端保存控件的状态，包括控件的属性以及在页面上的布局等
13	Text	该属性为 string 型，用于获取或设置需要在该控件上显示的文本
14	AutoPostBack	该属性为布尔型，用鼠标单击该控件或控件值发生变化时是否马上回传给服务器进行处理

当某控件的 EnableViewState 属性为 true 时，表示该控件的值在页面刷新或回传重新显示页面后不会丢失，但却耗费网络资源和服务器资源。因此当页面回传后无须保值处理时应设为 false。下面对一些常用服务器端控件进行简要介绍。

（1）Label 和 Literal 控件

使用 Label 控件在网页的设置位置上显示文本，可以通过 Text 属性自定义显示文本。Text 属性中可以包含其他 HTML 元素，Label 控件转换成客户端 HTML 代码后，Label 就成了，即该控件最终呈现一个 span 元素，例如：

```
<asp:Label ID="Label1" runat="server" Text="<B>I'm bold Font</B>">
</asp:Label>
```

Literal 控件和 Label 控件类似，但它不可向文本中添加任何 HTML 元素。因此，Literal 控件不支持包括位置属性在内的任何样式属性。转换成客户端 HTML 代码后，Literal 则是什么标记都不带，但 Literal 控件允许指定是否对内容进行编码。

通常情况下，当希望文本和控件直接呈现在页面中而不使用任何附加标记时，可使用 Literal 控件。

（2）TextBox 文本框控件

该控件用于获取用户输入的文本或显示文本。通过 ReadOnly 属性可设置该控件为只读，而不能对其中的文本进行编辑修改。Text 控件常用的属性如表 6-8 所示。

表 6-8　Text 控件的常用属性

序　号	属　　性	描　　述
1	Columns	以字符为单位的文本框的宽度
2	MaxLength	文本框中可输入的最大字符数
3	TextMode	确定文本框的行为模式是单行文本框(SingleLine)、多行文本框(MultiLine)还是密码编辑框(Password)
4	Rows	设置多行文本框显示的行数。该属性仅对多行文本框起作用
5	MaxLength	该属性是 int 类型值，用于设置 TextBox 控件中输入的最大字符数
6	ReadOnly	该属性为 bool 类型值，用于设置 TextBox 控件中的内容是否为只读
7	Text	该属性为 string 类型值，用于获取或设置 TextBox 控件中的文本

（3）Image 控件

该控件是用来插入图片的，常用的属性如表 6-9 所示。

表 6-9　Image 控件常用的属性

序　号	属　　性	描　　述
1	AlternateText	在图片不存在或尚未下载完的时候显示替换的文本
2	DescriptionUrl	指定更详细图像说明的 URL
3	ImageAlign	该属性用于设置或获取 Image 控件与网页其他对象的对齐方式。例如左对齐、右对齐、基底、顶端、中间等
4	ImageUrl	获取或设置图片来源的相对或绝对位置

（4）Button、LinkButton、ImageButton 控件

这三个控件分别表示普通按钮、超链接形式的按钮和图像按钮。它们是 Web 开发中相当重要的控件，允许用户通过单击来执行操作。每当用户单击按钮时，即调用 Click 事件处理程序。可将代码放入 Click 事件处理程序来执行所选择的操作。这三个控件常用的属性如表 6-10 所示。

表 6-10　Button、LinkButton、ImageButton 常用的控件属性

序　号	属　　性	描　　述
1	OnClientClick	输入客户端代码，以便单击按钮后先在客户端执行此代码后再执行服务器端的响应事件。例如输入 alert('ok')后，先显示一个对话框后，在执行服务器端事件程序
2	CommandName	为该按钮设定一个关联命令。具体使用方法见例 6.5
3	CommandArgument	为该按钮设定一个关联命令的参数

【例 6.5】 该例演示了如何使用 CommandName、CommandArgument 属性来识别用户按下了哪个按钮。每个按钮调用相同的 CommandBtn_Click 事件程序。文件 Ex_6_5.aspx 内容如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_5.aspx.cs"
Inherits="ex_6_5" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"> <title>无标题页</title></head>
<body>
   <form id="form1" runat="server">
      <h3>Button CommandName Example</h3>
      Click on one of the command buttons.<br><br>
      <asp:Button ID="Button1" Text="Sort Ascending" CommandName="Sort"
CommandArgument="Ascending" OnCommand="CommandBtn_Click" runat="server" />
 
      <asp:Button ID="Button2" Text="Sort Descending" CommandName="Sort"
CommandArgument="Descending" OnCommand="CommandBtn_Click" runat="server" />
<br><br>
```

```
      <asp:Button ID="Button3" Text="Submit" CommandName="Submit" OnCommand=
      "CommandBtn_Click" runat="server" />  
```

```
      <asp:Button ID="Button4" Text="Unknown Command Name" CommandName=
"UnknownName" CommandArgument="UnknownArgument" OnCommand=
"CommandBtn_Click" runat="server" />   <asp:Button ID="Button5"
Text="Submit Unknown Command Argument" CommandName="Submit"
      CommandArgument="UnknownArgument" OnCommand="CommandBtn_Click"
   runat="server" />
      <br><br><asp:Label ID="Message" runat="server" />
   </form>
</body>
</html>
```

文件 ex_6_5.aspx.cs 内容如下：

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

public partial class ex_6_5 : System.Web.UI.Page
{
    protected void CommandBtn_Click(Object sender, CommandEventArgs e)
   {
      switch (e.CommandName)
      {
         case "Sort":
            // Call the method to sort the list
            Sort_List((String)e.CommandArgument);
            break;
         case "Submit":
            // Display a message for the Submit button being clicked
            Message.Text = "You clicked the Submit button";
            // Test whether the command argument is an empty string ("")
            if ((String)e.CommandArgument == "")
            {  // End the message.
               Message.Text += ".";
            }
```

```
            else
            {  // Display an error message for the command argument
```

```
                Message.Text += ", however the command argument is not
                recogized.";
            }
            break;
        default:
            // The command name is not recognized. Display an error message
            Message.Text = "Command name not recogized.";
            break;
    }
  }
  protected void Sort_List(string commandArgument)
  {
    switch (commandArgument)
    {   case "Ascending":
            // Insert code to sort the list in ascending order here
            Message.Text = "You clicked the Sort Ascending button.";
            break;
        case "Descending":
            // Insert code to sort the list in descending order here
            Message.Text = "You clicked the Sort Descending button.";
            break;
        default:
    // The command argument is not recognized. Display an error message
            Message.Text = "Command argument not recogized.";
            break;
    }
  }
}
```

（5）HyperLink 控件

该控件用于制作文本或图片超级链接。常用的属性如表 6-11 所示。

表 6-11　HyperLink 控件常用的属性

序　号	属　　性	描　　述
1	ImageUrl	该属性用于获取或设置 HyperLink 控件链接源的来源，若设置它的属性，表示 HyperLink 控件为图片超链接
2	NavigateUrl	获取或设置 HyperLink 控件链接的网页或网址
3	Target	获取或设置 HyperLink 控件被单击时，其所链接的网页将在哪个框架或窗口打开（用于框架网页）

（6）RadioButton 单选按钮

该控件为用户提供由两个或多个互斥选项组成的选项集。当用户选择某单选按钮时，同一组中的其他单选按钮不能同时被选定。当单击 RadioButton 按钮时，其 Checked 属性设置为 true，并且调用 Click 事件处理程序。当 Checked 属性的值更改时，将引发 Checked- Changed

事件。用户可以通过用 Text 属性设置控件内显示的文本。RadioButton 控件常用的属性如表 6-12 所示。

表 6-12　RadioButton 控件的常用属性

序　号	属　　性	描　　述
1	GroupName	将多个单选钮指定为同一组的组名。这样就构成了互斥选项
2	Checked	该属性为 bool 类型，用于确定某一个单选按钮是否被选中
3	TextAlign	文本标签的对齐方式

（7）CheckBox 复选框

该控件通常是成组使用，完成多重选项的目的。这个控件与 RadionButton 控件相比，它们的相似之处在于都是用于指示用户所选的选项，不同之处在于，单选框一次只能选一个按钮，而复选框则可以选择任意数量。CheckBox 控件常用的属性如表 6-13 所示。

表 6-13　CheckBox 控件常用的属性

序　号	属　　性	描　　述
1	Checked	该属性为 bool 类型，用于确定某一个复选按钮是否被选中
2	Text	该属性是 string 类型值，用于设置与复选按钮相关的标签
3	TextAlign	文本标签的对齐方式

【例 6.6】 CheckBox 控件界面设计。ex_6_6.aspx 文件代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_6.aspx.cs"
Inherits="ex_6_6"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server"> <title>TextBox 控件输入样式效果</title>
</head>
<body> <center>
       <form id="form1" runat="server">
        <div>
           <asp:CheckBox ID="CheckBox1" runat="server" Text="学　院" /> 
            
           <asp:CheckBox ID="CheckBox2" runat="server" Text="专　业" /> 
           <asp:CheckBox ID="CheckBox3" runat="server" Text="班　级" /><br />
           <br />
           <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
           Text="选择" />
         </div>
       </form></center>
</body>
</html>
```

ex_6_6.aspx.cs 文件的代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
public partial class ex_6_6 : System.Web.UI.Page
{    //单击 <asp:Button>控件时触发事件
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (CheckBox1.Checked)
            Response.Write("您选择了" + "'" + CheckBox1.Text + "'");
        Response.Write("<br/>");
        if (CheckBox2.Checked)
            Response.Write("您选择了" + "'" + CheckBox2.Text + "'");
        Response.Write("<br/>");
        if (CheckBox3.Checked)
            Response.Write("您选择了" + "'" + CheckBox3.Text + "'");
    }
}
```

（8）DropDownList 控件

DropDownList 控件使用户可以从下拉列表框中进行选择。DropDownList 控件与下面将要介绍的 ListBox 控件非常相似，不同之处在于其选择项列表在用户单击下拉按钮前一直保持隐藏状态。同时它不支持多重选择。DropDownList 控件常用属性及说明如表 6-14 所示。

表 6-14　DropDownList 控件的常用属性及说明

序　号	属　　性	说　　明
1	DataSource	获取或设置对象数据源
2	DataTextField	获取或设置为列表项提供文本内容的数据源字段
3	DataValueField	获取或设置为各列表项提供值的数据源字段
4	SelectedIndex	获取或设置 DropDownList 控件中的选定项的索引
5	SelectedItem	获取列表控件中索引最小的选定项
6	SelectedValue	获取列表空间中选定项的值
7	Text	获取或设置 DropDownList 空间的 SelectedValue 属性

该控件在设计时就可以确定下拉选项内容,也可从其他数据源得到下拉列表内容。当添加该控件到“设计”界面时，会出现 DropDownList 任务对话框。在鼠标右击该控件出现的快捷菜单中，选择“显示智能标志”也会出现 DropDownList 任务对话框。对话框中具有“选择数据源”和“编辑项”两项功能。“选择数据源”可以以向导方式指定下拉列表的数据源。选择“编辑项”后出现“ListItem 集合编辑器”对话框，如图 6-3 所示。

图 6-3　ListItem 集合编辑器

单击“添加”实际上就增加了一条下拉列表行，该行被处理成 ListItem 对象，它有 4 个属性，如表 6-15 所示。

表 6-15　ListItem 对象的属性及说明

序　号	属　　性	说　　明
1	Enabled	如果设置成 false 则该条下拉行将被隐藏，不出现在下拉框中
2	Selected	是否将该行作为选定行
3	Text	该行的文本标签
4	Value	与该行文本标签对应的值，其值可与 text 相同，也可不同。例如 text 为男，而 value 为 0，也即用 0 表示性别为男

实际上，在 VS 2010 中可以使用 ListItem 集合编辑器的控件还有 ListBox、RadioButtonList、CheckBoxList、BulletedList 控件，它们的用法大致相似。

【例 6.7】 该例向 DropDownList 控件添加固定的 3 条选项，然后通过单击按钮 Add Item 为下拉控件动态增加一条可选项；单击按钮 Delete Item，删除一条选中的下拉项；单击按钮 Display Item 显示选中的下拉项内容。ex_6_7.aspx 文件代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_7.aspx.cs"
Inherits=" ex_6_7"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server"><title>DropDownList 控件输入样式效果</title>
</head>
<body>
  <form id="form1" runat="server">
```

```
            <asp:DropDownList ID="DropDownList1" runat="server">
            <asp:ListItem Value="1">财政与公共管理学院</asp:ListItem>
            <asp:ListItem Value="2">工商管理学院</asp:ListItem>
```

```
            <asp:ListItem Value="3">管理科学与工程管理学院</asp:ListItem>
            </asp:DropDownList><br /><br />
    <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
    Text="Add Item" />
    <asp:Button ID="Button2" runat="server" OnClick="Button2_Click"
    Text="Delete Item" />
    <asp:Button ID="Button3" runat="server" OnClick="Button3_Click"
    Text="Display Item" />
    </form>
</body>
</html>
```

ex_6_7.aspx.cs 文件代码如下：

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

public partial class ex_6_7 : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {  //添加一个新的下拉项
        ListItem li=new ListItem("计算机学院","5");
        DropDownList1.Items.Add(li);
    }
    protected void Button2_Click(object sender, EventArgs e)
    {   //删除下拉列表框中已被选中的项
        DropDownList1.Items.Remove(DropDownList1.SelectedItem);
    }
    protected void Button3_Click(object sender, EventArgs e)
    {  //显示下拉列表框中被选中项的文本和对应的值
    Response.Write(DropDownList1.SelectedItem.Text + DropDownList1.
    SelectedItem.Value);
    }
}
```

（9）ListBox 列表框

列表框通过显示多个选项供用户选择达到与用户对话的目的，如果候选项较多的时候它

还可以自动地加上滚动条。我们可以通过设置列表框的 Items 属性来添加列表框的内容。我们在属性菜单中找到 Items 选项，鼠标单击右边的按钮，会弹出如前所述的 ListItem 集合编辑器，就可方便地添加列表框中的每一项。ListBox 控件常用的属性如表 6-16 所示。

表 6-16　ListBox 控件的常用属性

序　号	属　　性	描　　述
1	DataSource	为 ListBox 控件设置数据源
2	SelectIndex	int 类型值，用于指示 ListBox 控件当前选中的索引值。注意索引是从 0 开始的
3	SelectedItem	Object 类型值，用于指示 ListBox 控件当前选中的项，它与 SelectIndex 属性的区别在于，SelectIndex 表示当前选中项的索引，而 SelectItem 表示的是当前选中项本身
4	SelectedValue	返回列表控件中选定项的值

（10）FileUpload 文件上载控件

该控件可实现让用户在客户端选择一个文件，然后放到 Web 服务器的某个指定的文件夹下。具体使用方法请看下例。

【例 6.8】 该例让用户从客户端选择一个图片文件，限定其只能是 jpg/bmp/gif/png/swf 文件，大小不超过 25KB，图片尺寸长宽在 190×130 以内。ex_6_8.aspx 文件内容如下：

```
<%@  Page  Language="C#"  AutoEventWireup="true"  CodeFile="ex_6_8.aspx.cs"
Inherits="ex_6_8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"> <title>上载文件示例</title></head>
<body>
   <form id="form1" runat="server">
      <asp:FileUpload ID="FileUp" runat="server" />
      <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
      Text="upload" />
   </form>
</body>
</html>
```

ex_6_8.aspx.cs 文件内容如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
```

```
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
using System.Web.UI.HtmlControls;

public partial class ex_6_8 : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {   //得到上载文件路径名和文件名
        string fullname = FileUp.PostedFile.FileName.ToString().Trim();
        if (fullname == "")
        {  this.Response.Write("<script language='javascript'>alert('请选择
        要上载的图片文件！')</script>");
           return;
        }

        //得到上载文件的文件名
        string filename = this.FileUp.FileName.ToString();
        //从上载文件名中得到文件类型
        string filetype = fullname.Substring(fullname.LastIndexOf(".") + 1);
        //将文件扩展名转化为小写字符
        string Ext = filetype.ToLower();
        //判断上载的文件是否为图片文件，若不是图片文件将不予处理
   if (Ext == "jpg" || Ext == "bmp" || Ext == "gif" || Ext == "png" || Ext
   == "swf")
        {
           //指定上载文件在 Web 服务器上的存放位置
           string UploadedFile = Server.MapPath(".") + "\\" + filename;
           //上传到 Web 服务器主目录下
           FileUp.PostedFile.SaveAs(UploadedFile);
           try
           {   //动态构建一个图像，目的是取得图片的宽度和高度
    System.Drawing.Image mylImage = System.Drawing.Image.FromFile
    (UploadedFile);
              int myLength = FileUp.PostedFile.ContentLength;  //图像文件大小
              int myWidth = mylImage.Width; //图像宽度
              int myHeight = mylImage.Height; //图像高度
              mylImage.Dispose();   //从内存中去除图像

              if (myLength > 25600 || myWidth > 190 || myHeight > 130)
              {   System.IO.File.Delete(UploadedFile);//删除已经上载的图片文件
                  Response.Write("<script language='javascript'>alert('上传
                  图片规格错误。图片应小于 25K,长宽在 190x130 以内。')</script>");
                  return;
              }
              else

              {
```

```
    Response.Write("<script language='javascript'>alert('上传图片成功！')
    </script>");
                return;
            }
        }
        catch
        {
    Response.Write("<script language='javascript'>alert('上传图片规格错误！')
    </script>");
            return;
        }
      }
      else
      {
        Response.Write("<script language='javascript'>
        alert('请选择图片文件(jpg/bmp/gif/png/swf)！')</script>");
         return;
      }
    }
}
```

（11）Panel 和 Placeholder 控件

Panel 和 Placeholder 控件都属于容器控件。容器控件是指该控件可以动态容纳其他控件或 HTML 元素。要在运行时刻向 Web 页面中动态添加内容，利用容器控件即可实现动态添加内容到 Web 页中。

Panel 和 Placeholder 控件（占位控件）转换成客户端 HTML 代码后，呈现为 div 元素。Placeholder Web 服务器控件可以将空的容器控件放置到页内，然后在运行时动态添加、删除子元素等。该控件只呈现其子元素，不具有自己的基于 HTML 的输出。

Panel 服务器控件最终在客户端呈现为 div 元素，但在 Web 开发时允许用户在该控件中添加其他控件，而且在运行过程中也允许动态添加控件。

【例 6.9】 在 aspx 页面中放置一个 Panel 控件，其中放入了一个 CheckBox 和 Placeholder 控件。单击 CheckBox 后，在 Placeholder 和 Panel 中增加了新的数目可变的控件。ex_6_9.aspx 文件内容如下：

```
<%@  Page  Language="C#"  AutoEventWireup="true"  CodeFile="ex_6_9.aspx.cs"
Inherits="ex_6_8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head runat="server"> <title>无标题页</title></head>
<body>
<form id="form1" runat="server">
  <asp:Panel   ID="Panel1"   runat="server"   Height="214px"   Width="208px"
```

```
BackColor="#FFFFC0"
         BorderStyle="Groove">
    <asp:CheckBox   ID="CheckBox1"   runat="server"   Text=" 查 看 回 复 "
OnCheckedChanged="CheckBox1_CheckedChanged" AutoPostBack="True" /><br />
     <asp:PlaceHolder ID="PlaceHolder1" runat="server"></asp:PlaceHolder>
     <br />
   </asp:Panel>
 </form>
</body>
</html>
```

ex_6_9.aspx.cs 文件代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

public partial class ex_6_8 : System.Web.UI.Page
{
    protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
    {   if (CheckBox1.Checked)
        {   //为 PlaceHolder 动态增加 Literal1、TextBox1 和 Button1 三个控件
            Literal Literal1 = new Literal();
            Literal1.Text = "<br>回复内容: ";
            PlaceHolder1.Controls.Add(Literal1);

            TextBox TextBox1 = new TextBox();
            TextBox1.Columns = 50;
            TextBox1.Rows = 20;
            TextBox1.TextMode = System.Web.UI.WebControls.TextBoxMode.
            MultiLine;
            TextBox1.Font.Bold = true;
            PlaceHolder1.Controls.Add(TextBox1);
            TextBox1.BorderColor = System.Drawing.Color.Red;
            TextBox1.Text = "This is my reply "; //可从数据库得到

            Button Button1 = new Button();
            Button1.Text = "我要回复";
            PlaceHolder1.Controls.Add(Button1); //可增加单击事件的编程实现
```

```
            //为 Panel 动态增加 Literal2 和 TextBox2 两个控件
            Literal Literal2 = new Literal();
            Literal2.Text = "<br>请输入姓名: ";
            Panel1.Controls.Add(Literal2);

            TextBox TextBox2 = new TextBox();
            TextBox2.Columns = 20;
            TextBox2.Font.Bold = true;
            Panel1.Controls.Add(TextBox2);
        }
    }
}
```

6.2.2　服务器端验证控件

服务器端验证控件是 ASP.NET 控件中新产生的一类验证控件。当用户输入错误时，验证控件可以显示错误信息。验证控件在正常工作情况下是不可见的，只有当用户输入数据有误时，它们才是可见的。服务器端验证控件包含如表 6-17 所示的六种验证控件。验证控件的公共属性如表 6-18 所示。

表 6-17　验证控件

序　号	控 件 名 称	说　　明
1	RequiredFieldValidator	输入值域是否为空
2	RangeValidator	输入值域是否在指定范围内
3	RegularExpressionValidator	输入值域是否符合某正则表达式要求的格式
4	CompareValidator	输入值和另外一个值满足什么关系
5	CustomValidator	定制的验证检查方式
6	ValidationSummary	检验其他验证控件的结果并集中显示

表 6-18　验证控件的公共属性

序　号	控 件 名 称	说　　明
1	ControlToValidate	指定一个控件 ID，该控件需要进行输入验证
2	ErrorMessage	用来显示错误信息
3	ForeColor	指定错误信息显示时的颜色
4	Display	指定验证控件的错误信息如何显示。Display="static"，即静态显示方式（系统默认方式）。当验证控件初始化时，需要在网页上有足够的空间来放置验证控件。当没有显示错误信息时，验证控件仍然占据一定的网页位置。Display="Dynamic"，即动态显示方式。当验证控件初始化时，控件不再占有网页上的位置，只有在需要显示错误信息时，控件才会占有一定的网页位置。Display="None"，即不在当前验证控件中显示错误信息，而在页面的总结验证控件 ValidationSummary 中显示错误信息
5	EnableClientScript	是否启动客户端验证，默认为 true。若为 false 则启动 Web 服务器来验证。采用客户端验证可得到较快的处理速度

（1）RequiredFieldValidator 控件

该控件又称非空验证控件，常用于文本输入框的非空验证。若在网页上使用此控件，则当用户提交网页到服务器端时，系统自动检查被验证控件的输入是否为空。如果为空，则网页显示错误信息。RequiredFieldValidator 控件使用方法如下：

```
<asp:RequiredFieldValidator id="控件名称" runat="server" ControlToValidate="
要检查的控件名" ErroeMessage ="错误消息" Display="Static|Dynamic|None">文本信息
</asp:RequiredFieldValidator>
```

（2）RangeValidator 控件

该控件又称范围验证控件。当用户输入不在验证范围内的值时将引发页面错误消息。该控件提供 Integer、String、Date、Double 和 Currency 五种验证。RangeValidator 控件的使用方法如下：

```
<asp:RangeValidator id="控件名称" runat="server" ControlToValidate = "要验证
的控件名" Type="数据类型" MinimumValue="最小值" MaximumValue="最大值"
ErrorMessage="错误信息" Display="Static|Dynamic|None">文本信息</asp:
RangeValidator>
```

（3）RegularExpressionValidator 控件

该控件又称正则表达式验证控件，它的验证功能比非空验证控件和范围验证控件更强大，用户可以自定义或书写自己的验证表达式。RegularExpressionValidator 控件的使用方法如下：

```
<asp:RegularExpressionValidator id="控件名称" runat="server"
ControlToValidate="要验证的控件名" ValidationExpression="正则表达式"
ErrorMessage="错误信息" Display="Static|Dynamic|None">文本信息</asp:
RegularExpressionValidator>
```

（4）CompareValidator 控件

该控件又称比较验证控件，主要用来验证 TextBox 控件内容或者某个控件的内容与某个固定表达式的值是否相同。CompareValidator 控件的使用方法如下：

```
<asp:CompareValidator id="控件名称" runat="server" ControlToValidate="要验证
的控件名" ValueToCompare="常值" ControlToCompare = "作比较的控件名" Type="输入值
" Operator = "操作方法" ErrorMessage = "错误信息" Display ="Static|Dynamic|
None">文本信息</asp:CompareValidator>
```

（5）CustomValidator 控件

该控件又称自定义验证控件，它使用自定义的验证函数来作为验证方式。CustomValidator 控件与其他验证控件的最大区别是该控件可以添加客户端验证函数和服务器端验证函数。客户端验证函数总是在 ClientValidatorFunction 属性中指定的，而服务器端验证函数总是通过 OnServerValidate 属性来设定，并指定为 ServerValidate 事件处理程序。CustomValidator 控件的使用方法如下：

```
<asp:CustomValidator id="控件名称" runat="server" ControlToValidate="要验证的
```

```
控件名" ErrorMessage="错误信息" Display="Static|Dynamic|None">文本信息
</asp:CustomValidator>
```

（6）ValidationSummary 控件

该控件又称错误总结控件，主要是收集本页中所有验证错误信息，并将它们组织好后显示出来，其使用方法如下：

```
<asp:ValidationSummary id="控件名称" runat="server" HeaderText="头信息"
ShowSummary="True|False"
Display="List|BulletList|SingleParagraph"></asp:ValidationSummary>
```

其中 ShowSummary="True"表示在该页中显示摘要信息，否则以对话框方式显示摘要信息。Display 确定摘要的显示方式。

【例 6.10】 使用 RequiredFieldValidator 控件控制必须输入班级编号，当班级编号为空时显示“班级编号不能为空”的字样；用 RangeValidator 控件控制用户输入的班级编号必须在 1～9 之间；利用 RegularExpressionValidator 控件检验用户输入的 URL 地址和 E-mail 地址是否规范；用 CompareValidator 控件检查用户输入两次密码的值是否相同；使用 ValidationSummary 控件将验证后的结果以对话框方式显示。Ex_6_10.aspx 文件如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_10.aspx.cs"
Inherits="ex_6_10" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server"><title>验证控件的运用</title></head>
<body>
   <form id="form1" runat="server">
      <asp:Literal ID="Literal1" runat="server" Text="学院编号"></asp:Literal>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:RangeValidator ID="RangeValidator1" runat="server" ErrorMessage=
"编号在 1 和 9 之间!" Display="Dynamic" ControlToValidate="TextBox1" MaximumValue=
"25" MinimumValue="1"></asp:RangeValidator><br />
      <asp:Literal ID="Literal2" runat="server" Text="学院名称">
</asp:Literal>
      <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox> 
      <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat=
"server" ErrorMessage="学院名称不能为空!" Display="Dynamic"
ControlToValidate="TextBox2"></asp:RequiredFieldValidator><br />
      <asp:Label ID="Label1" runat="server" ForeColor="MediumBlue" Text=
"请输入 URL 地址:"></asp:Label>
      <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
      <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat=
"server" ControlToValidate="TextBox3" ErrorMessage="输入的 URL 地址不正确!"
Display="Dynamic" ValidationExpression="http(s)?://([\w-]+\.)+[\w-]+(/[\w-
/?%&=]*)?"> </asp:RegularExpressionValidator><br />
      <asp:Label ID="Label2" runat="server" ForeColor="MediumBlue" Text=
"请输入 Email 地址:"></asp:Label>
```

```
        <asp:TextBox ID="TextBox4" runat="server"></asp:TextBox>
        <asp:RegularExpressionValidator     ID="RegularExpressionValidator2"
runat="server" ControlToValidate="TextBox4" ErrorMessage="输入的 Email 地址不
正 确 !"    Display="Dynamic"    ValidationExpression="\w+([-+.']\w+)*@\w+
([-.]\w+)*\.\w+([-.]\w+)*"> </asp:RegularExpressionValidator><br />
        <asp:Label ID="Label3" runat="server" ForeColor="MediumBlue" Text=
"请输入旧密码:"></asp:Label>
    <asp:TextBox    ID="TextBox5"    runat="server"    TextMode="Password">
</asp:TextBox><br />
        <asp:Label ID="Label4" runat="server" ForeColor="MediumBlue"
                Text="请输入新密码:"></asp:Label>
        <asp:TextBox    ID="TextBox6"    runat="server"    TextMode="Password">
</asp:TextBox><br />
        <asp:Label ID="Label5" runat="server" ForeColor="MediumBlue"
                Text="请确认新密码:"></asp:Label>
        <asp:TextBox    ID="TextBox7"    runat="server"    TextMode="Password">
</asp:TextBox>
        <asp:CompareValidator ID="CompareValidator1" runat="server"
                ErrorMessage="前后密码输入不一致!"
           ControlToCompare="TextBox6"          ControlToValidate="TextBox7"
Display="Dynamic"></asp:CompareValidator>
        <asp:ValidationSummary ID="ValidationSummary_Error"
          runat="server" ShowMessageBox="True" DisplayMode="SingleParagraph" />
        <br />
        <asp:Button ID="Button1" runat="server" Text="确  定" />
        <asp:Button ID="Button2" runat="server" Text="取  消" />
    </form>
</body>
</html>
```

其设计界面如图 6-4 所示。

一般情况下输入数据的有效性验证可通过客户端验证来进行，可避免服务器端验证所需要的信息往返，除非是对安全性要求高的场合采用服务器端验证。使用验证控件的 EnableClientScript 属性来指定是否启用客户端验证。

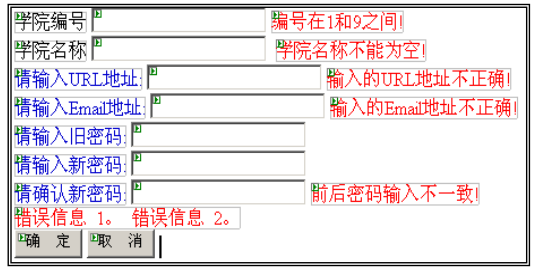


图 6-4　验证控件设计界面

6.2.3　服务器控件使用注意事项

使用 VS 2010 进行 Web 开发的初学者很容易产生对服务器控件的依赖性，几乎所有 Web 页面都采用服务器控件，HTML 控件（客户端控件）被束之高阁，当然这是一种简便的编程方法，但却是一种不考虑运行效率的做法。我们在本节开头已介绍了客户端控件和服务器控件的区别。服务器控件意味着要消耗更多的网络带宽和服务器资源，而客户端控件的处理逻辑全部在浏览器中进行。因此不能养成使用服务器控件的习惯性思维，在使用控件的时候有

必要思考一下采用客户端控件的可能性。

下面的代码演示了服务器端和客户端文本框控件将显示相同的值，还演示了数据的传递方式。将 ASPX 文件中的服务器控件 TextBox1 删除（删除 ASPX 文件中的第 8 行和 CS 文件中的第 11 行），运行后得到的 HTML 页面大小为 567 字节（单击浏览器文件菜单中的属性选项查看）；反之将服务器控件 TextBox1 保留，删除客户端控件，运行后得到的 HTML 页面大小为 729 字节。可以看出实现同样的功能，服务器文本框控件在网络上要比客户端控件多传输 162 字节，这还不算后台处理的时间和占用的系统资源。因此应根据情况，尽量使用客户端控件。

Default2.aspx 页面文件代码：

```
1  <%@          Page          Language="C#"          AutoEventWireup="true"
2  CodeFile="Default2.aspx.cs" Inherits="Default2" %>
3  <% string hisName = "Yangking"; %>
4  <html><head runat="server"><title>无标题页</title></head>
5  <body>
6      <form id="form1" runat="server">
7          姓名: <input id="Text1" type="text" value="<%=myName%>" /> <br
8  />
9          姓名: <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
10     </form>
11      <script>alert('<%=myName %>');</script>  <!—显示 "WangCL" -->
12      <script>alert('<%=hisName %>');</script> <!—显示 "Yangking" -->
13 </body>
   </html>
```

Default2.aspx.cs 文件代码：

```
1  using System;
2  using System.Data;
3  using System.Web;
4  using System.Web.UI;
5  using System.Web.UI.WebControls;
6  using System.Web.UI.HtmlControls;
7  public partial class Default2 : System.Web.UI.Page
8  {   protected  string myName = "WangCL";
9      protected void Page_Load(object sender, EventArgs e)
10     {   myName = "WangCL";  //可访问数据库得到相关数据
11         TextBox1.Text = myName;
12     }
13 }
```

6.2.4　服务器端数据访问控件

数据访问控件根据所实现的功能分为两大类：数据源控件和数据绑定控件。数据源控件可实现对不同数据源的数据访问，主要包括连接数据源，使用 SQL 语句获取和管理数据等。数据绑定控件主要用于以多种方式显示数据。通常情况下，可以将数据源控件与数据绑定控件结合起来，数据源控件负责获取和处理数据，数据绑定控件负责将数据显示在页面上。数

据源控件和数据绑定控件如表 6-19 所示。

表 6-19　数据源控件和数据绑定控件

	控 件 名 称	说　　明
数据源控件	SqlDataSource	用于连接 SQL 数据库，可以用来从任何 OLEDB 或者符合 ODBC 的数据源中检索数据，能够访问目前主流的数据库系统
	AccessDataSource	用于连接 Access 数据库，允许以声明方式将 Access 数据库中的数据绑定到指定对象中
	ObjectDataSource	用于连接自定义对象，允许以声明方式将对象绑定到自定义对象公开的数据，以用于多层 Web 应用结构
	XmlDataSource	该控件可装载 XML 文件作为数据源，并将其绑定到指定的对象中
	SiteMapDataSource	该控件装载一个预先定义好的站点布局文件作为数据源，Web 服务器控件和其他控件可通过该控件绑定到分层站点地图数据，以便制作站点的页面导航功能
数据绑定控件	Repeater	自由地控制数据的显示。即可以使用非表格的形式来显示数据，从而能够更灵活地定义其显示的风格
	GridView	.NET 中强大功能的数据控件，不需要编写代码就可实现数据的连接、绑定、编辑、删除、增加等功能
	DataList	通过定义模板或样式来灵活地显示数据
	DetailsView	用于显示表中数据源的单个记录，其中每个数据行表示记录中的一个字段。该控件通常与 GridView 控件组合使用，构成主-从显示方案
	FormView	用于显示表中数据源的单个记录。使用 FormView 控件时，需指定模板以显示和编辑绑定值。模板中包含用于创建窗体的格式、控件和绑定表达式。FormView 控件通常与 GridView 控件一起用于主控/详细信息方案
	ReportViewer	用于显示报表、工具栏和文档结构图的视图区域。工具栏是可配置的，它提供了运行时功能以支持多页报表中的导航、缩放、搜索、打印和导出功能。提供编程接口，以便可以自定义控件、配置控件，以及通过代码与控件进行交互，包括更改在运行时 ReportViewer 使用的数据源

下面分别对上述控件进行一一介绍。

1．数据源控件

数据源控件分为两种：普通数据源控件和层次化数据源控件（树型结构）。普通数据源控件包括 SqlDataSource、ObjectDataSource、AccessDataSource 主要检索带有行和列的基于数据表的数据源；层次化数据源控件包括 XmlDataSource 和 SiteMapDataSource，主要检索包含层次化数据的数据源。

（1）SqlDataSource 控件

SqlDataSource 控件的应用非常广泛，可以用来从任何 OLEDB 或者符合 ODBC 的数据源中检索数据，能够访问目前主流的数据库系统。该控件常用的属性及说明如表 6-20 所示。

表 6-20　SqlDataSource 控件常用属性及说明

序号	属　　性	说　　明
1	ConnectionString	用于设置连接数据源字符串
2	ProviderName	用于设置不同的数据提供程序，未设置该属性，则默认为 System.Data.SqlClient
3	SelectCommand	执行数据记录选择操作的 SQL 语句或者存储过程名称

4	UpdateCommand	执行数据记录更新操作的 SQL 语句或者存储过程名称
5	DeleteCommand	执行数据记录删除操作的 SQL 语句或者存储过程名称

续表

序号	属　　性	说　　明
6	InsertCommand	执行数据记录添加操作的 SQL 语句或者存储过程名称
7	DataSourceMode	用于获取或设置 SqlDataSource 控件获取数据时所使用的数据返回模式，包含 2 个可选枚举值：DataReader 和 DataSet

在 VS 2010“设计”中放入 SqlDataSource 控件后，出现智能标记“配置数据源”，单击后出现“配置数据源”对话框向导，即可选择一个现有的数据连接，也可创建一个新的数据连接。单击“新建连接”后出现“添加连接”对话框，如图 6-5 所示。

默认显示数据源连接方式为 Microsoft ODBC 数据源（ODBC），可以单击“更改”选择其他连接数据源的方式。单击“更改”后出现如图 6-6 所示对话框。

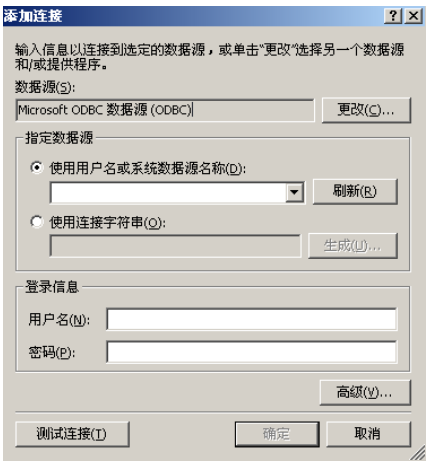


图 6-5　“添加连接”对话框（1）

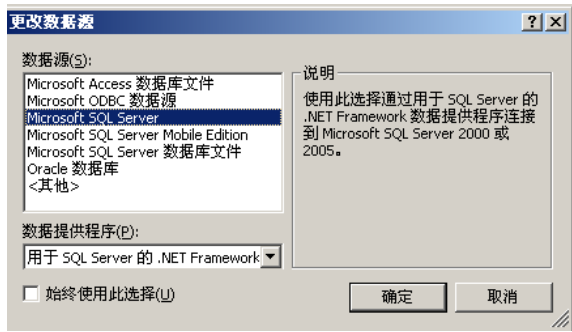


图 6-6　“更改数据源”对话框

从图 6-6 可知，.NET 支持多种数据源连接方式。若数据库为 SQL Server 2000/2005，采用 Microsoft SQL Server 专用接口方式可以获得最快的连接性能。单击“确定”按钮后出现如图 6-7 所示的“添加连接”对话框。

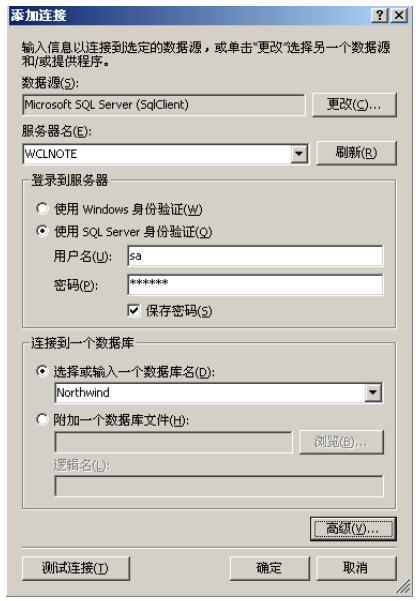


图 6-7　“添加连接”对话框（2）

输入数据库服务器名称，输入登录到数据库服务器的用户名和密码，再选择数据库（这里选择 SQL Server 中自带的 NorthWind 数据库），单击“测试连接”按钮看看是否连接成功。可单击“高级”来修改连接数据库的各项参数设置，以利提高数据访问性能。单击“确定”按钮后又回到“配置数据源”对话框（此时已在 VS 2010 的“服务器资源管理器”中生成了一个 wclnote.Northwind.dbo 连接），单击“下一步”按钮后，选择将此连接另存为 NorthwindConnectionString 后，连接字符串将被保存到应用程序配置文件 web.config 中（6.2.5 节中介绍）。其内容为：

```
<connectionStrings>
  <add        name="NorthwindConnectionString"        connectionString="Data
Source=WCLNOTE;Initial
Catalog=Northwind;Persist Security Info=True;User ID=sa;Password=docman"
   providerName="System.Data.SqlClient" />
</connectionStrings>
```

其中，connectionString 字串中 Data Source 参数设置数据库服务器名称；Initial Catalog 参数设置数据库名称；User ID 参数设置访问数据库的用户名；Password 参数设置该用户名对应的数据库访问密码。

再单击“下一步”后，出现“配置 Select 语句”对话框，其中有两个选项“指定自定义 SQL 语句或存储过程”和“指定来自表或视图的列”。当选中“指定来自表或视图的列”后，选择某个表或视图，选择要显示的列名，通过单击 Where 和 Order by 生成 where 子句和 order by 子句，再单击“下一步”后，单击“测试查询”可预先查看查询出来的数据，单击“完成”，即完成了利用 SqlDataSource 控件连接数据源的工作。当选中“配置 Select 语句”对话框中的“指定自定义 SQL 语句或存储过程”后，单击“下一步”后，可直接输入 SQL 语句或用“查询生成器”辅助生成一个 SQL 语句，也可选择一个存储过程作为数据源，最后单击“完成”按钮完成数据源的配置工作。

经过上述步骤后，在 VS 2010 中按下 Ctrl+Alt+S 组合键或单击主菜单中的“视图”| “服务器资源管理器”就会出现“服务器资源管理器”窗口，单击前面生成的 wclnote.Northwind.dbo 数据连接后，就可在 VS 2010 环境下管理 Northwind 数据库的表、视图、存储

过程等，可以在快捷菜单中发现很多有用的功能，例如新建或删除一个表/视图/存储过程/函数、显示选中表/视图的数据、创建一个表的触发器，等等。当然也可在“服务器资源管理器”中直接创建一个新的数据连接，方法是鼠标右击“数据连接”后选择“添加连接”，再在出现的对话框中进行配置。

SqlDataSource 控件的数据源配置工作完成后，就可被数据绑定控件使用，用来将数据显示出来。数据源配置后在 ASPX 文件中生成如下代码：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
  SelectCommand="SELECT    [CustomerID],    [CompanyName],    [ContactName],
[ContactTitle],  [Address],  [City],  [Region],  [PostalCode],  [Country],
[Phone], [Fax] FROM [Customers]">
</asp:SqlDataSource>
```

其中 <%$ ConnectionStrings:NorthwindConnectionString %> 就是从应用程序配置文件 web.config 中取回的数据库连接参数。

（2）AccessDataSource 控件

.NET 提供了一种访问 Access 数据库的专用数据源控件 AccessDataSource。该控件能够快速连接 Access 数据库，并且通过 SQL 语句等对数据库记录实现操作。通过 SqlDataSource 控件也可连接 Access 数据库，Access 数据库文件可放在服务器任意位置，但 AccessDataSource 控件在使用时要求事先将 Access 数据库文件放到 Web 服务器主目录下的某个位置，一般放在主目录的 App_Data 子目录下。该控件常用的属性及说明如表 6-21 所示。此控件的数据源配置过程同 SqlDataSource 控件。

表 6-21　AccessDataSource 控件常用属性及说明

序号	属　性	说　明
1	DataFile	该属性用于指定 Access 文件的虚拟路径或者 UNC 路径
2	ProviderName	用于设置不同的数据提供程序，未设置该属性，则默认为 System.Data.OleDb
3	SelectCommand	执行数据记录选择操作的 SQL 语句或者存储过程名称
4	UpdateCommand	执行数据记录更新操作的 SQL 语句或者存储过程名称
5	DeleteCommand	执行数据记录删除操作的 SQL 语句或者存储过程名称
6	InsertCommand	执行数据记录添加操作的 SQL 语句或者存储过程名称
7	DataSourceMode	用于获取或设置 SqlDataSource 控件获取数据时所使用的数据返回模式，包含 2 个可选枚举值：DataReader 和 DataSet

（3）ObjectDataSource 控件

多数 ASP.NET 数据源控件，如 SqlDataSource 等，都在两层应用程序层次结构中使用。在该层次结构中，表示层（ASP.NET 网页）可以与数据层（数据库和 XML 文件等）直接进行通信。但是，常用的应用程序设计原则是，将表示层与业务逻辑相分离，而将业务逻辑封装在业务对象中。这些业务对象在表示层和数据层之间形成一层，从而生成一种三层应用程序结构。ObjectDataSource 控件通过提供一种将相关页上的数据控件绑定到中间层业务对象的方法，为三层结构提供支持。在不使用扩展代码的情况下，ObjectDataSource 使用中间层

业务对象以声明方式对数据执行选择、插入、更新、删除、分页、排序、缓存和筛选操作。

ObjectDataSource 控件使用反射调用业务对象的方法，以对数据执行选择、更新、插入和删除操作。设置 ObjectDataSource 控件的 TypeName 属性来指定要用作源对象的类　　名称。

（4）XmlDataSource 控件

以上介绍的数据源控件是支持 SQL 语言的关系型数据库，其数据库中保存和组织数据的方式是采用数据表和列。通常以这种方式组织的数据被称为“表格化数据”，其特点是扁平存储和组织。然而，存储和组织数据不只是一种方式，还有一种常见的方式被称为“层次化数据”。该控件就是专门针对 XML 数据而发布的数据源控件。该控件常用的属性及说明如表 6-22 所示。

表 6-22　XmlDataSource 控件常用属性及说明

序号	属　　性	说　　明
1	DataFile	该属性用于获取或设置控件所绑定的 XML 文件
2	TransformFile	XML 转换文件的路径，即 XSL 文件
3	XPath	该属性用于获取或设置应用于 XML 数据中的 Xpath 查询值，默认值为空。Xpath 是一种查询语言，用于检索 XML 文档中包含的信息

XmlDataSource 控件在配置后生成的 ASPX 文件中生成如下代码：

```
<asp:XmlDataSource            ID="XmlDataSource1"            runat="server"
DataFile="~/XMLFile.xml" TransformFile="~/XSLTFile.xsl">
 </asp:XmlDataSource>
```

数据绑定控件可利用 XmlDataSource 控件连接层次化数据源进行相关处理。

（5）SiteMapDataSource 控件

SiteMapDataSource 控件用来连接包含来自站点地图的导航数据。此数据包括有关网站中的页的信息，如 URL、标题、说明和导航层次结构中的位置。该控件的使用较为复杂，有兴趣的读者可参阅其他专门书籍。

2．数据绑定控件

数据绑定控件负责将从数据库中获取的数据显示出来。

（1）GridView 控件

GridView 控件采用表格形式显示从数据库中获取的数据集合。通过使用 GridView 控件，用户可以显示、编辑、删除、排序和翻阅多种不同的数据源中的表格数据。该控件常用的属性及说明如表 6-23 所示。

表 6-23　GridView 控件属性及说明

序号	属　　性	说　　明
1	AllowPaging	获取或设置是否启用分页功能
2	AllowSorting	获取或设置是否启用排序功能
3	Columns	获取表示 GridView 控件中列字段的 DataControlField 对象集合
4	DataMember	当数据源包含多个不同的数据项列表时，获取或设置数据绑定控件绑定到的数

		据列表名称
5	DataSource	获取或设置数据绑定对象的数据源
6	DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
7	EditIndex	获取或设置要编辑的项的索引
8	GridLines	获取或设置 GridView 控件的网格线样式
9	PageIndex	获取或设置当前显示页的索引
10	Rows	获取表示 GridView 控件中数据行的 GridViewRow 对象的集合
11	SelectedIndex	获取或设置 GridView 控件中的选中行的索引
12	SelectedRow	获取对 GridViewRow 对象的引用
13	SelectedValue	获取 GridView 控件中选中行的数据值

【例 6.11】 该例说明对 SqlDataSource 数据源控件和 GridView 数据绑定控件的操作。这是一个使用 GridView 控件显示和编辑数据的示例。与 GridView 控件联系在一起的数据源控件是 SqlDataSource。本例所访问的数据来自 SQL Server 2000 示例数据库 pubs 的数据表 authors。示例效果如图 6-8 所示。单击“编辑”即可修改被选中的那行记录。

该示例主要包括两个过程：一是对 SqlDataSource 控件进行配置；二是对 GridView 控件进行配置。对 SqlDataSource 控件进行配置的过程已在前面作了详细介绍，注意应选择 pubs 数据库，而不是 Northwind。下面介绍对 GridView 控件进行配置。

首先将 GridView 控件放到页面的指定位置，如图 6-9 所示。

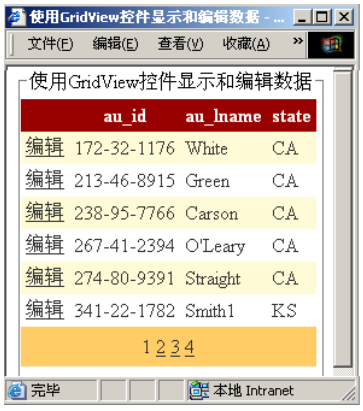


图 6-8　GridView 示例效果图

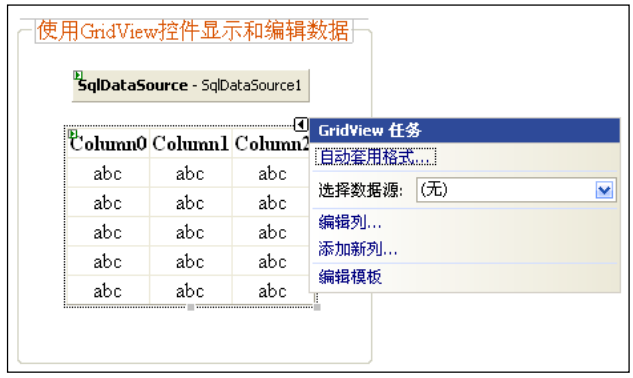


图 6-9　配置 GridView 控件（1）

在智能标记中的“选择数据源”中选择 SqlDataSource1 控件后，配置选项如图 6-10 所示。选中启用编辑和启用分页。然后用鼠标右击 GridView 控件后，在出现的快捷菜单中选择“自动套用格式”，在出现的“自动套用格式”对话框中选择“彩色型”。

图 6-10　配置 GridView 控件（2）

至此，完成了 GridView 控件的配置。生成的 ex_6_11.aspx 源代码如下所示：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_11.aspx.cs"
Inherits="ex_6_11" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head id="Head1" runat="server"><title>使用 GridView 控件显示和编辑数据
</title></head>
<body>
   <form id="form1" runat="server">
      <fieldset style="width: 250px">
         <legend>使用 GridView 控件显示和编辑数据</legend>
         <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:pubsConnectionString %>"
            SelectCommand="SELECT [au_id], [au_lname], [state] FROM
               [authors]"
            DeleteCommand="DELETE FROM [authors] WHERE [au_id] = @au_id"
InsertCommand="INSERT INTO [authors] ([au_id], [au_lname], [state]) VALUES
(@au_id, @au_lname, @state)"
            UpdateCommand="UPDATE [authors] SET [au_lname] = @au_lname,
[state] = @state WHERE [au_id] = @au_id">
            <DeleteParameters>
               <asp:Parameter Name="au_id" Type="String" />
            </DeleteParameters>
            <UpdateParameters>
```

```
                <asp:Parameter Name="au_lname" Type="String" />
                <asp:Parameter Name="state" Type="String" />
                <asp:Parameter Name="au_id" Type="String" />
            </UpdateParameters>
            <InsertParameters>
                <asp:Parameter Name="au_id" Type="String" />
                <asp:Parameter Name="au_lname" Type="String" />
                <asp:Parameter Name="state" Type="String" />
            </InsertParameters>
        </asp:SqlDataSource>  <br />
        <asp:GridView ID="GridView1" runat="server"
            AllowPaging="True" AutoGenerateColumns="False"
            CellPadding="4" DataKeyNames="au_id"
            DataSourceID="SqlDataSource1" Font-Size="Medium"
            ForeColor="#333333" GridLines="None" PageSize="6">
            <FooterStyle       BackColor="#990000"       Font-Bold="True"
ForeColor="White" />
            <Columns>
                <asp:CommandField ShowEditButton="True" />
                <asp:BoundField DataField="au_id" HeaderText="au_id"
                     ReadOnly="True" SortExpression="au_id" />
                <asp:BoundField DataField="au_lname"
                     HeaderText="au_lname" SortExpression="au_lname" />
                <asp:BoundField DataField="state"
                     HeaderText="state" SortExpression="state" />
            </Columns>
            <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
            <SelectedRowStyle BackColor="#FFCC66" Font-Bold="True"
     ForeColor="Navy" />
            <PagerStyle BackColor="#FFCC66"
                        ForeColor="#333333" HorizontalAlign="Center" />
            <HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor=
     "White" />
            <AlternatingRowStyle BackColor="White" />
        </asp:GridView>
        <br />
      </fieldset>
   </form>
</body>
</html>
```

（2）Repeater 控件

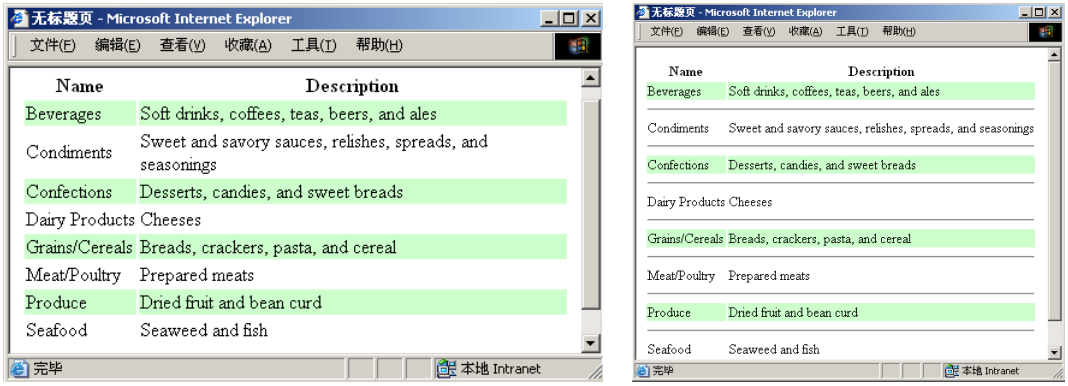
数据绑定控件 Repeater 的主要功能是以更自由的方式来控制数据的显示。它会按照所要求的样式严格地输出数据记录。Repeater 控件本身不具备内置的呈现功能，用户必须通过创建模板为 Repeater 控件提供布局。模板中可以包含标记和控件的任意组合。如果未定义模板，或者如果模板不包含元素，则当应用程序运行时，该控件不显示在页上。Repeater 控件支持的模板如表 6-24 所示。

表 6-24　Repeater 控件支持的模板

模 板 属 性	说　　明
HeaderTemplate	页头模板容纳数据列表开始处需要呈现的文本和控件
ItemTemplate	数据项模板容纳需要重复显示的数据记录，数据记录以 HTML 元素和控件来呈现
AlternatingItemTemplate	交替项模板功能同数据项模板。通常，可以使用此模板为交替项创建不同的外观，例如指定一种与在 ItemTemplate 中指定的颜色不同的背景色
SeparatorTemplate	分隔模板容纳每条记录之间呈现的元素。如用一条直线（hr 元素）实现数据记录间的分隔
FooterTemplate	页脚模板容纳在列表的结束处分别呈现的文本和控件

由于 Repeater 控件没有默认的外观，因此可以使用该控件的模板创建许多种列表，包括表布局、逗号分隔的列表、XML 格式的列表等。

【例 6.12】 该例说明了如何利用模板来实现数据的输出。图 6-11 为最终显示效果图。左图为没有交替项模板的情况；右图为具有交替项模板的情况。该例在“设计”中放置了一个 Repeater 控件和 SqlDataSource 控件，SqlDataSource 控件用来连接数据源，配置数据源方法同前。在 Repeater 控件中的 DataSourceID 属性中选择此 SqlDataSource 控件。进入“源”中，手工加入上述 5 个模板，分别在每个模板中编写代码。具体 Ex_6_12.aspx 文件代码如下：



（a）　　（b）

图 6-11　Repeater 控件示例效果图

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_12.aspx.cs"
Inherits="ex_6_12" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head id="Head1" runat="server"><title>无标题页</title></head>
<body>
   <form id="form1" runat="server">
      <div>
          <asp:SqlDataSource ID="SqlDataSource1" runat="server"
          ConnectionString="<%$ ConnectionStrings:
NorthwindConnectionString %>"
             SelectCommand="SELECT [CategoryID], [CategoryName],
[Description], [Picture] FROM [Categories]">
          </asp:SqlDataSource>
          <asp:Repeater ID="Repeater1" runat="server" DataSourceID=
              "SqlDataSource1">
             <HeaderTemplate>
                <table>
                   <tr>
                      <th>Name</th> <th>Description</th>
                   </tr>
             </HeaderTemplate>
             <ItemTemplate>
                <tr>
                   <td bgcolor="#CCFFCC">
                       <asp:Label runat="server" ID="Label1"
                       Text='<%# Eval("CategoryName") %>' />
                   </td>
                   <td bgcolor="#CCFFCC">
```

```
                       <asp:Label runat="server" ID="Label2"
                       Text='<%# Eval("Description") %>' />
                   </td>
                </tr>
             </ItemTemplate>
             <AlternatingItemTemplate>
                <tr>
                   <td>
                      <asp:Label runat="server" ID="Label3"
                      Text='<%# Eval("CategoryName") %>' />
                   </td>
                   <td>
                      <asp:Label runat="server" ID="Label4"
                      Text='<%# Eval("Description") %>' />
                   </td>
                </tr>
             </AlternatingItemTemplate>
```

```
            <SeparatorTemplate>
               <tr><td height="1" colspan="2"><hr /></td></tr>
            </SeparatorTemplate>
            <FooterTemplate></table></FooterTemplate>
         </asp:Repeater>
      </div>
   </form>
</body>
</html>
```

上述代码中，对放置 5 个模板的先后次序并无要求，建议按表 6-24 的顺序依次放置，便于阅读代码。ItemTemplate 是必需的，AlternatingItemTemplate 和 SeparatorTemplate 可有可无。需要重复显示的 HTML 元素（包括数据）放在 ItemTemplate、AlternatingItemTemplate 和 SeparatorTemplate 中，HeaderTemplate 和 FooterTemplate 中放置不重复显示的 HTML 元素。“<%# Eval("CategoryName") %>”表示从 SqlDataSource1 数据源中取出 CategoryName 列的值。

（3）DataList 控件

DataList 服务器控件使用模板和样式来显示数据。它需要连接到某个数据源控件，实现不同布局的数据显示。可以将 DataList 控件绑定到 ADO.NET 数据集（DataSet 类）、数据读取器（SqlDataReader 类或 OleDbDataReader 类）等。可使用 DataList 控件来显示模板定义的数据绑定列表。DataList 控件所支持的模板类型类似于 Reapter 控件，但增加了编辑和选择模板。编辑模板可用来删除和修改记录；选择模板可用来处理选中某个记录后的显示方式。模板类型如表 6-25 所示。

表 6-25　DataList 模板类型

模 板 属 性	说　　明
HeaderTemplate	页头模板容纳数据列表开始处需要呈现的文本和控件
ItemTemplate	数据项模板容纳需要重复显示的数据记录，数据记录以 HTML 元素和控件来呈现
AlternatingItemTemplate	交替项模板功能同数据项模板。通常，可以使用此模板为交替项创建不同的外观，例如指定一种与在 ItemTemplate 中指定的颜色不同的背景色
SelectedItemTemplate	选择模板包含一些元素，当用户选择 DataList 控件中的某一条记录时将呈现这些元素。通常，可用此模板来通过不同的背景色或字体颜色直观地区分选定的行。还可以通过显示数据源中的其他字段来展开该项
EditItemTemplate	编辑模板指定当某记录处于编辑模式中时的布局。此模板通常包含一些编辑控件，如 TextBox 控件
SeparatorTemplate	分隔模板容纳每条记录之间呈现的元素。如用一条直线（hr 元素）实现数据记录间的分隔
FooterTemplate	页脚模板容纳在列表的结束处分别呈现的文本和控件

DataList 控件常用属性及说明与 GridView 控件相似。DataList 控件的常用事件及说明如表 6-26 所示。

表 6-26　DataList 控件常用事件及说明

序号	事　　件	说　　明
1	CancelCommand	对 DataList 控件中的某个项单击 Cancel 按钮时发生
2	DeleteCommand	对 DataList 控件中的某个项单击 Delete 按钮时发生
3	EditCommand	对 DataList 控件中的某个项单击 Edit 按钮时发生
4	ItemCommand	当单击 DataList 控件中的任一按钮时发生
5	ItemDataBound	当某个记录的数据被绑定到 DataList 控件时发生
6	ItemCreated	当在 DataList 控件中创建记录时在服务器上发生
7	SelectedIndexChanged	在两次服务器发送之间，当 DataList 控件中选择了不同的项时发生
8	UpdateCommand	对 DataList 控件中的某个项单击 Update 按钮时发生

在使用 DataList 控件时，可以通过智能标记中的“自动套用格式”、“属性生成器”、“编辑模板”来进行控件的属性设置。对于分页、记录编辑、排序等必须通过手工编写代码完成。下面通过例子来说明该控件的使用。

【例 6.13】 该例说明了在 DataList 中如何利用模板来实现数据的输出。在例 6.12 所示代码中，将 Repeater 的控件标记换成 DataList 控件标记：

```
<asp:Repeater ID="Repeater1" runat="server"
DataSourceID="SqlDataSource1"> … </asp:Repeater>
=====〉
<asp:DataList   ID="DataList1"   runat="server"   DataKeyField="CategoryID"
DataSourceID="SqlDataSource1">   …  </asp:DataList>
```

最终数据显示效果相同。实际上 Repeater 控件是 DataList 控件的一个特例。

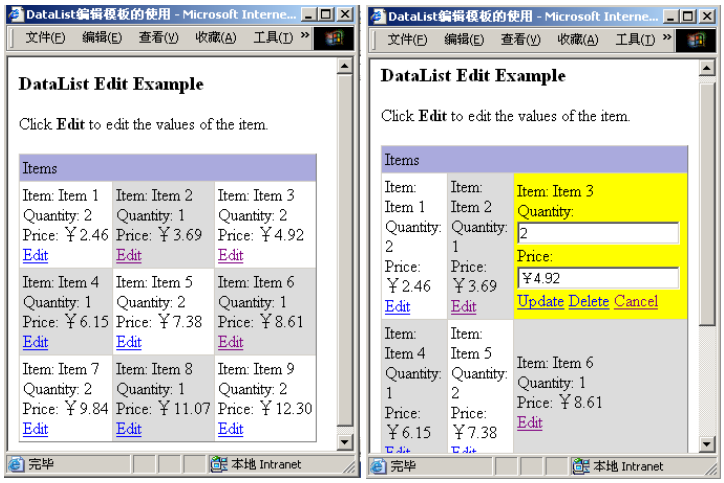
【例 6.14】 下面的代码示例演示如何使用数据项编辑模板来实现用户数据的编辑功能。该例中，DataList 控件 ItemsList 并没有绑定数据源，而是程序运行后动态生成一个 CartView 数据视图，通过以下代码绑定到 ItemsList 控件。

```
ItemsList.DataSource = CartView;
ItemsList.DataBind();
```

ex_6_14.aspx 中“<%# DataBinder.Eval(Container.DataItem, "Item") %>”表示在页中调用 DataBind 方法得到数据视图 Item 列的值。当在页上调用 DataBind 方法时，数据绑定表达式创建服务器控件属性和数据源之间的绑定。可以将数据绑定表达式包含在服务器控件开始标记中属性/值对的值一侧，或页中的任何位置。所有数据绑定表达式都必须包含在 <%# 和 %> 字符之间。ASP.NET 支持分层数据绑定模型，该模型创建服务器控件属性和数据源之间的绑定。几乎任何服务器控件属性都可以绑定到任何公共字段或属性，这些公共字段或属性位于包含页或服务器控件的直接命名容器上。数据绑定表达式使用 Eval 和 Bind 方法将数据绑定到控件，并将更改提交回数据库。Eval 方法是静态（只读）方法，该方法采用数据字段的值作为参数并将其作为字符串返回。Bind 方法支持读/写功能，可以检索数据绑定控件的值并将任何更改提交回数据库。

该例中先将一个 DataList 控件放于“设计”中，鼠标右击该控件，选择“编辑模板”|

“项模板”，在项模板中输入文本"Item:"、"Quantity:"和"Price:"，并放入一个 LinkButton 控件 EditButton，设置 Text 和 CommandName 属性设为"Edit"，用于将来单击该控件可编辑数据。在编辑模板中同样输入上述三个文本，对应三个文本放入 Label 控件 ItemLabel、TextBox 控件 QtyTextBox 和 PriceTextBox，另外放入三个 LinkButton 控件 UpdateButton、DeleteButton 和 CancelButton，用于数据的更新、删除和取消编辑功能。最终运行结果如图 6-12 所示。页面文件 ex_6_14.aspx 的代码为：



（a）　　（b）

图 6-12　DataList 控件示例效果图

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_14.aspx.cs"
Inherits="ex_6_14" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server"> <title>DataList 编辑模板的使用</title></head>
<body>
    <form id="form1" runat="server">
        <h3> DataList Edit Example</h3>
        Click <b>Edit</b> to edit the values of the item. <br><br>
        <asp:DataList ID="ItemsList" GridLines="Both"
            RepeatColumns="3" RepeatDirection="Horizontal"
            CellPadding="3" CellSpacing="0" OnEditCommand="Edit_Command"
            OnUpdateCommand="Update_Command" OnDeleteCommand="Delete_Command"
            OnCancelCommand="Cancel_Command" runat="server">
            <HeaderStyle BackColor="#aaaadd"></HeaderStyle>
            <AlternatingItemStyle BackColor="Gainsboro">
            </AlternatingItemStyle>
            <EditItemStyle BackColor="yellow"></EditItemStyle>
            <HeaderTemplate>
               Items
```

```
          </HeaderTemplate>
          <ItemTemplate>
             Item:
             <%# DataBinder.Eval(Container.DataItem, "Item") %><br>
             Quantity:
             <%# DataBinder.Eval(Container.DataItem, "Qty") %><br>
             Price:
             <%# DataBinder.Eval(Container.DataItem, "Price", "{0:c}") %><br>
             <asp:LinkButton ID="EditButton" Text="Edit"
                  CommandName="Edit" runat="server" />
          </ItemTemplate>
          <EditItemTemplate>
             Item:
             <asp:Label ID="ItemLabel"
             Text='<%# DataBinder.Eval(Container.DataItem, "Item") %>'
                   runat="server" />
             <br>Quantity:
             <asp:TextBox ID="QtyTextBox"
             Text='<%# DataBinder.Eval(Container.DataItem, "Qty") %>'
             runat="server" />
             <br>Price:
             <asp:TextBox ID="PriceTextBox"
             Text='<%# DataBinder.Eval(Container.DataItem, "Price",
             "{0:c}") %>'
             runat="server" /> <br>
             <asp:LinkButton ID="UpdateButton" Text="Update"
                  CommandName="Update" runat="server" />
             <asp:LinkButton ID="DeleteButton" Text="Delete"
                  CommandName="Delete" runat="server" />
             <asp:LinkButton ID="CancelButton" Text="Cancel"
                  CommandName="Cancel" runat="server" />
          </EditItemTemplate>
       </asp:DataList>
    </form>
</body>
</html>
```

ex_6_14.aspx.cs 文件代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
using System.Web.UI.HtmlControls;

public partial class ex_6_13 : System.Web.UI.Page
{   // The Cart and CartView objects temporarily store the data source
    // for the DataList control while the page is being processed.
    DataTable Cart = new DataTable();
    DataView CartView;

    protected void Page_Load(Object sender, EventArgs e)
    {
        // With a database, use an select query to retrieve the data.
        // Because the data source in this example is an in-memory
        // DataTable, retrieve the data from session state if it exists;
        // otherwise, create the data source.
        GetSource();

        // The DataList control maintains state between posts to the server;
        // it only needs to be bound to a data source the first time the
        // page is loaded or when the data source is updated.
        if (!IsPostBack){ BindList(); }
    }

    protected void BindList()
    {   // Set the data source and bind to the DataList control.
        ItemsList.DataSource = CartView;
        ItemsList.DataBind();
    }

    protected void GetSource()
```

```
    {
        // For this example, the data source is a DataTable that
        // is stored in session state. If the data source does not exist,
        // create it; otherwise, load the data.
        if (Session["ShoppingCart"] == null)
        {
            // Create the sample data.
            DataRow dr;

            // Define the columns of the table.
            Cart.Columns.Add(new DataColumn("Qty", typeof(Int32)));
            Cart.Columns.Add(new DataColumn("Item", typeof(String)));
            Cart.Columns.Add(new DataColumn("Price", typeof(Double)));

            // Store the table in session state to persist its values
            // between posts to the server.
            Session["ShoppingCart"] = Cart;
```

```
        // Populate the DataTable with sample data.
        for (int i = 1; i <= 9; i++)
        {
            dr = Cart.NewRow();
            if (i % 2 != 0)
            {dr[0] = 2;}
            else
            {dr[0] = 1;}
            dr[1] = "Item " + i.ToString();
            dr[2] = (1.23 * (i + 1));
            Cart.Rows.Add(dr);
        }
    }
    else
    {
        // Retrieve the sample data from session state.
        Cart = (DataTable)Session["ShoppingCart"];
    }
    // Create a DataView and specify the field to sort by.
    CartView = new DataView(Cart);
    CartView.Sort = "Item";
    return;
}

protected void Edit_Command(Object sender, DataListCommandEventArgs e)
{   // Set the EditItemIndex property to the index of the item clicked
```

```
    // in the DataList control to enable editing for that item. Be sure
    // to rebind the DataList to the data source to refresh the control.
    ItemsList.EditItemIndex = e.Item.ItemIndex;
    BindList();
}

protected void Cancel_Command(Object sender, DataListCommandEventArgs e)
{   // Set the EditItemIndex property to -1 to exit editing mode. Be sure
    // to rebind the DataList to the data source to refresh the control.
    ItemsList.EditItemIndex = -1;
    BindList();
}

protected void Delete_Command(Object sender, DataListCommandEventArgs e)
{   // Retrieve the name of the item to remove.
    String item = ((Label)e.Item.FindControl("ItemLabel")).Text;
```

```
        // Filter the CartView for the selected item and remove it from
        // the data source.
        CartView.RowFilter = "Item='" + item + "'";
        if (CartView.Count > 0)
        {CartView.Delete(0);}
        CartView.RowFilter = "";

        // Set the EditItemIndex property to -1 to exit editing mode. Be sure
        // to rebind the DataList to the data source to refresh the control.
        ItemsList.EditItemIndex = -1;
        BindList();
    }

    protected void Update_Command(Object sender, DataListCommandEventArgs e)
    {   // Retrieve the updated values from the selected item.
        String item = ((Label)e.Item.FindControl("ItemLabel")).Text;
        String qty = ((TextBox)e.Item.FindControl("QtyTextBox")).Text;
        String price = ((TextBox)e.Item.FindControl("PriceTextBox")).Text;

        // With a database, use an update command to update the data.
        // Because the data source in this example is an in-memory
        // DataTable, delete the old row and replace it with a new one.

        // Filter the CartView for the selected item and remove it from
        // the data source.
        CartView.RowFilter = "Item='" + item + "'";
        if (CartView.Count > 0)
        {CartView.Delete(0);}
```

```
        CartView.RowFilter = "";

        // ***************************************************************
        // Insert data validation code here. Make sure to validate the
        // values entered by the user before converting to the appropriate
        // data types and updating the data source.
        // ***************************************************************

        // Add a new entry to replace the previous item.
        DataRow dr = Cart.NewRow();
        dr[0] = qty;
        dr[1] = item;
        // If necessary, remove the '￥' character from the price before
        // converting the price to a Double.
        if (price[0] == '￥')
```

```
        {   dr[2] = Convert.ToDouble(price.Substring(1));}
        else
        {   dr[2] = Convert.ToDouble(price);}
        Cart.Rows.Add(dr);

        // Set the EditItemIndex property to -1 to exit editing mode.
        // Be sure to rebind the DataList to the data source to refresh
        // the control.
        ItemsList.EditItemIndex = -1;
        BindList();
    }
}
```

（4）DetailsView 控件

DetailsView 控件的主要功能是以表格形式显示和处理来自数据源的单条数据记录，其表格只包含两个数据列。一个数据列逐行显示数据列名，另一个数据列显示与对应列相关的详细数据值。它可与 GridView 控件结合使用，以用于主/详细信息方案。

DetailsView 控件支持以下功能：①绑定至数据源控件；②内置插入功能；③内置更新和删除功能；④内置分页功能；⑤以编程方式访问 DetailsView 对象模型从而动态设置属性、处理事件等；⑥可通过主题和样式自定义外观。

DetailsView 控件与 GridView 控件的功能，虽有很多相似，但也有不同。不同点表现在，DetailsView 控件内建数据添加的支持，而 GridView 控件不具备此项功能。另外，在数据显示方面也有区别。通常，GridView 控件习惯于使用表格形式将数据集合显示出来，其表格顶部显示了多个数据列名，具体的数据行都跟随在列名以下显示。DetailsView 虽然也显示数据，但是每次只显示一行具体的数据。

DetailsView 控件设置是否自动生成数据行的属性是 AutoGenerateRows，在自定义设置数据绑定行的过程中，DetailsView 控件使用了<Fields>标签，而 GridView 控件使用的是<Columns>。

表 6-27 列出了可以在 DetailsView 控件中使用的 7 种不同数据行字段类型。

表 6-27　DetailsView 控件使用的 7 种不同数据行字段类型

字 段 类 型	说　　明
BoundField	以文本形式显示数据源中某个字段的值
ButtonField	在 DetailsView 控件中显示一个命令按钮。允许显示一个带有自定义按钮（如“添加”或“移除”按钮）控件的行
CheckBoxField	在 DetailsView 控件中显示一个复选框。此行字段类型通常用于显示具有布尔值的字段
CommandField	在 DetailsView 控件中显示用来执行编辑、插入或删除操作的内置命令按钮
HyperLinkField	将数据源中某个字段的值显示为超链接。此数据行字段允许将另一个字段绑定到超链接的 URL
ImageField	在 DetailsView 控件中显示图像
TemplateField	根据指定的模板，为 DetailsView 控件中的行显示用户定义的内容。此数据行字段类型用于创建自定义的数据行字段

以上 7 个数据行字段只有当 AutoGenerateRows 属性设置为 false 时才能使用。在默认情况下，AutoGenerateRows 属性设置为 true，即允许 DetailsView 控件自动生成数据行。此时，每个数据行字段将以文本形式，按其出现在数据源中的顺序显示在表格中。自动生成行提供了一种显示记录中每个字段的快速简单的方式。Fields 对象集合允许以编程方式管理 DetailsView 控件中的数据行字段。需要注意的是，自动生成的行字段不会添加到 Fields 集合中。同时，显式声明的行字段可与自动生成的行字段结合在一起。两者同时使用时，先呈现显式声明的字段，再呈现自动生成的行字段。

DetailsView 同样支持模板功能。表 6-28 列出了它所支持的模板。

表 6-28　DetailsView 控件支持的模板

模 板 属 性	说　　明
EmptyDataTemplate	获取或者设置当 DetailsView 控件绑定空的数据源控件时，由开发人员定义的对于空数据所呈现模板的内容。默认值为空。可以将自定义模板内容放在<EmptyDataTemplate>和</EmptyDataTemplate>标签之中。如果 DetailsView 控件中同时设置了该属性与 EmptyDataText 属性，那么 EmptyDataTemplate 的优先级比 EmptyDataText 高
FooterTemplate	获取或者设置由开发人员自定义的对于表尾行所呈现模板的内容。默认值为空。可以将自定义模板内容放置在<FooterTemplate>和</FooterTemplate>标签之中。如果同时设置了 FooterText 属性，该属性将覆盖 FooterText 所设置的内容
HeaderTemplate	获取或者设置由开发人员自定义的对于表头行所呈现模板的内容。默认值为空。可以将自定义模板内容放置在<HeaderTemplate>和</HeaderTemplate>标签之中。如果同时设置 FooterText 属性，该属性将覆盖 HeaderText 所设置的内容
PagerTemplate	获取或者设置由开发人员自定义的对于分页行所呈现模板的内容。默认值为空。可以将自定义模板内容放在<PagerTemplate>和</PagerTemplate>标签之中

DetailsView 控件常用属性及说明与 GridView 控件相似，在此不再赘述。DetailsView 控件的常用事件及说明如表 6-29 所示。

表 6-29　DetailsView 控件常用事件及说明

序　号	事　　件	说　　明
1	ItemCommand	该事件发生在控件中某个按钮被单击时
2	ItemCreated	该事件发生在创建一个新数据记录时
3	ItemDeleted	该事件发生在单击删除按钮，在删除操作之后执行
4	ItemDeleting	该事件发生在单击删除按钮，在删除操作之前执行
5	ItemInserted	该事件发生在单击添加按钮，在添加操作之后执行
6	ItemInserting	该事件发生在单击添加按钮，在添加操作之前执行
7	ItemUpdated	该事件发生在单击更新按钮，在更新操作之后执行
8	ItemUpdating	该事件发生在单击更新按钮，在更新操作之前执行
9	ModeChanged	该事件发生在修改数据模式，CurrentMode 得到更新之后执行
10	ModeChanging	该事件发生在修改数据模式，CurrentMode 得到更新之前执行
11	PageIndexChanged	该事件发生在 PageIndex 属性的值在分页操作后更改时发生
12	PageIndexChanging	该事件发生在 PageIndex 属性的值在分页操作前更改时发生

【例 6.15】 该例说明了如何利用 DetailsView 控件来实现数据的输出。该例在“设计”中放置了一个 DetailsView 控件和 SqlDataSource 控件，SqlDataSource 控件用来连接数据源，配置数据源方法同前。在 DetailsView 控件中的 DataSourceID 属性中选择此 SqlDataSource 控件。进入“源”中，手工加入 2 个 Fields 对象，分别在每个 Fields 对象中编写代码。具体 Ex_6_15.aspx 文件代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_15.aspx.cs"
Inherits="ex_6_15" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server"><title>无标题页</title></head>
<body>
   <form id="form1" runat="server">
   <div>
   <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$
ConnectionStrings:NorthwindConnectionString %>"
         SelectCommand="SELECT [CategoryID], [CategoryName],
[Description], [Picture] FROM [Categories]">
      </asp:SqlDataSource>
      <asp:DetailsView ID="DetailsView1" runat="server" Height="84px"
Width="266px"
            DataSourceID="SqlDataSource1" AutoGenerateRows="False"
      HeaderText=" 目录信息" HeaderStyle-BackColor="#CC99FF"
            HeaderStyle-HorizontalAlign="Center" AllowPaging="True"
             PagerSettings-Position="Bottom" RowStyle-VerticalAlign="Top">
     <Fields>
      <asp:BoundField HeaderText="目录"
            NullDisplayText="没有目录" DataField="CategoryName" />
      <asp:BoundField HeaderText="描述"
            NullDisplayText="没有描述" DataField="Description"/>
      </Fields>
         <RowStyle VerticalAlign="Top" />
         <HeaderStyle BackColor="#CC99FF" HorizontalAlign="Center" />
      </asp:DetailsView>
   </div>
   </form>
</body>
</html>
```

上述代码中的<Fields>标签是必要的。DataField="CategoryName"表示从 SqlDataSource1 数据源中取出 CategoryName 列的值。最终执行效果如图 6-13 所示。

图 6-13　DetailsView 控件示例效果图

（5）FormView 控件

FormView 控件用于显示数据源中的单个记录。该控件与 DetailsView 控件类似，只是它显示用户定义的模板而不是行字段。FormView 控件支持的功能与 DetailsView 控件类似。不同点表现在 DetailsView 控件能够自动创建 HTML 表格结构代码，并且显示相关的数据字段名称和数据值。FormView 控件则默认创建一个空白的区域（实际上所创建的是一个只有一行一列的表格）。FormView 控件不具备自动创建表格显示数据的功能。FormView 控件需要开发人员自定义 ItemTemplate、PagerTemplate 等模板属性，以自定义方式显示各个字段。

FormView 控件支持以下功能：①支持绑定到数据源控件；②内置数据插入、更新和删除功能；③内置分页功能；④允许以编程方式访问 FormView 控件对象模型，以动态设置属性、处理事件等；⑤通过用户定义的模板、主题和样式自定义外观。

通常，FormView 控件用于更新和插入新数据记录，并且在主细表数据处理方面有较多应用。实现这些应用，FormView 控件需要数据源控件的支持，以便执行诸如更新、插入和删除记录的任务。即使 FormView 控件的数据源公开了多条记录，该控件一次也只显示一条数据记录。在该模板的自定义内容中都包含了数据绑定表达式，如：<%# Bind("columnName1")%>。

上述表达式用于绑定数据列的数据。如果在模板属性中包含了 Bind 表达式，那么表明 columnName1 是可读可写的。如果在模板属性中包含 Eval 表达式，则表明 columnName1 只用于显示数据，而不能修改该数据列的数据。

FormView 控件作为一个数据绑定控件，可与数据源控件结合实现各种数据操作。当 FormView 控件通过 DataSourceID 属性连接数据源控件后，该控件可利用数据源控件的内置功能，在自身内置功能的支持下，实现数据更新、删除、添加和分页等操作。与 GridView 和 DetailsView 控件不同的是，由于 FormView 控件使用模板属性，因此，没有提供自动生成命令按钮的功能。开发人员必须在模板属性中，自行定义各种命令按钮，这样才能实现数据操作功能。DetailsView 控件支持的模板见表 6-30。

表 6-30　FormView 控件支持的模板

模板属性	说　明
EditItemTemplate	获取或者设置在编辑模式下自定义项的内容
EmptyDataTemplate	获取或者设置当 DetailsView 控件绑定空的数据源控件时，由开发人员定义的对于空数据所呈现模板的内容。默认值为空。可以将自定义模板内容放在<EmptyDataTemplate>和</EmptyDataTemplate>标签之中。如果 DetailsView 控件中同时设置了该属性与 EmptyDataText 属性，那么 EmptyDataTemplate 的优先级比 EmptyDataText 高
FooterTemplate	获取或者设置由开发人员自定义的对于表尾行所呈现模板的内容。默认值为空。

	可以将自定义模板内容放置在<FooterTemplate>和</FooterTemplate>标签之中。如果同时设置了 FooterText 属性，该属性将覆盖 FooterText 所设置的内容
HeaderTemplate	获取或者设置由开发人员自定义的对于表头行所呈现模板的内容。默认值为空。可以将自定义模板内容放置在<HeaderTemplate>和</HeaderTemplate>标签之中。如果同时设置 FooterText 属性，该属性将覆盖 HeaderText 所设置的内容
InsertItemTemplate	获取或者设置在插入模式下自定义项的内容
ItemTemplate	获取或者设置在只读模式下自定义数据行的内容
PagerTemplate	获取或者设置由开发人员自定义的对于分页行所呈现模板的内容。默认值为空。可以将自定义模板内容放在<PagerTemplte>和</PagerTemplate>标签之中

FormView 控件的常用属性、常用事件及说明与 DetailsView 控件相似，在此不再赘述。

【例 6.16】 该例说明了如何利用 FormView 控件来实现数据的输出。该例在“设计”中放置了一个 FormView 控件和 SqlDataSource 控件，SqlDataSource 控件用来连接数据源，配置数据源方法同前。在 FormView 控件中的 DataSourceID 属性中选择此 SqlDataSource 控件。进入“源”中，手工加入上述模板中的 3 个模板，分别在每个模板中编写代码。具体 ex_6_16.aspx 文件代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_16.aspx.cs"
Inherits="ex_6_16" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server"><title>无标题页</title></head>
<body>
   <form id="form1" runat="server">
   <div>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
       ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
       SelectCommand="SELECT [CategoryID], [CategoryName], [Description],
[Picture] FROM [Categories]">
      </asp:SqlDataSource>
     <asp:FormView ID="FormView1" runat="server"
          DataSourceID="SqlDataSource1" AllowPaging="True">
```

```
     <HeaderTemplate>
               <table>
                  <tr>
                     <th>Name</th> <th>Description</th>
                  </tr>
           </HeaderTemplate>
         <ItemTemplate>
               <tr>
                  <td bgcolor="#CCFFCC">
                  <asp:Label runat="server" ID="Label1"
```

```
                    Text='<%# Eval("CategoryName") %>' />
                    </td>
                    <td bgcolor="#CCFFCC">
                    <asp:Label runat="server" ID="Label2"
                     Text='<%# Eval("Description") %>' />
                    </td>
                </tr>
            </ItemTemplate>
            <FooterTemplate></table></FooterTemplate>
        </asp:FormView>
    </div>
    </form>
</body>
</html>
```

上述代码中 ItemTemplate 是必需的，“<%# Eval("CategoryName") %>”表示从 SqlDataSource1 数据源中取出 CategoryName 列的值。显示效果如图 6-14 所示。

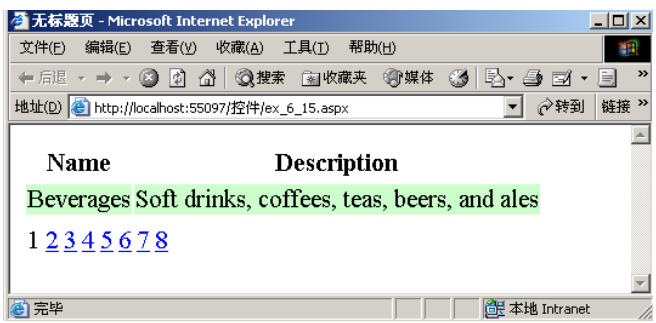


图 6-14　FormView 控件示例效果图

（6）ReportViewer 控件

VS 2010 已经内置了报表设计功能和用于显示报表的 ReportViewer 服务器控件。ReportViewer 服务器控件用来呈现表格格式数据、聚合数据和多维数据，它可以图表的形式显示数据。ReportViewer 服务器控件支持的图表有柱形图、条形图等。报表还可以导出 Excel 文件和 PDF 文件。ReportViewer 服务器控件可以配置成以本地处理模式或远程处理模式运行。

本地处理模式是指 ReportViewer 控件在客户端应用程序中处理报表。所有报表都是使用应用程序提供的数据作为本地过程处理的。若要创建本地处理模式下使用的报表，需要使用 Visual Studio 中的报表项目模板。该模式所有数据和报表的处理都是在客户端进行的，在处理大型或复杂的报表和查询时，应用程序性能可能会降低。

“远程处理模式”是指由 SQL Server 2005 Reporting Services 报表服务器处理报表。在远程处理模式下，ReportViewer 控件用作查看器，显示已经在 Reporting Services 报表服务器上发布的预定义报表。从数据检索到报表呈现的所有操作都是在报表服务器上处理的。若要使

用远程处理模式，则必须具有 SQL Server 2005 Reporting Services 的许可副本。

注意：Internet Explorer 6.0 以上版本浏览器是唯一保证支持用于报表的全套功能的浏览器，其他浏览器只支持部分功能。ReportViewer 控件中常用的属性如表 6-31 所示。

表 6-31　ReportViewer 控件中常用的属性

事　件	说　明
AsyncRendering	指明 ReportViewer 控件是否以异步方式呈现报表
BackColor	获取或设置控件报表区域的背景颜色
CurrentPage	获取或设置 ReportViewer 控件活动报表的当前页
DocumentMapCollapsed	获取或设置文档结构图的折叠状态
DocumentMapWidth	获取或设置文档结构图的宽度
ExportContentDisposition	指明内容应以内联方式存在还是作为附件存在
HyperlinkTarget	获取或设置单击报表中的超链接时返回网页内容的目标窗口或框架
LinkActiveColor	获取或设置控件中的活动链接的颜色
LinkActiveHoverColor	获取或设置当鼠标指针处于控件中的活动链接上时该链接的颜色
ProcessingMode	获取或设置 ReportViewer 控件的处理模式
PromptAreaCollapsed	折叠或还原 ReportViewer 控件的提示区域
ServerReport	获取报表查看器中的服务器报表
ShowCredentialPrompts	指明是否会显示要求用户凭据的提示
ShowExportControls	指明控件上的“导出”控件是否可见
WaitMessageFont	执行报表时显示的消息所采用的字体
ZoomMode	获取或设置控件的缩放模式
ZoomPercent	获取或设置显示报表时使用的缩放百分比

【例 6.17】该例说明了如何利用 ReportViewer 控件来实现数据的输出。该例在“设　计”中放置了一个 ReportViewer 控件和 SqlDataSource 控件，SqlDataSource 控件用来连接数据源，配置数据源方法同前。在 ReportViewer 控件中的 DataSourceID 属性中选择此 SqlDataSource 控件。按照 ReportViewer 控件向导一步步完成向导配置。

具体 Ex_6_17.aspx 文件代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_17.aspx.cs"
Inherits="ex_6_17" %>
<%@ Register Assembly="Microsoft.ReportViewer.WebForms, Version=8.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
```

```
   Namespace="Microsoft.Reporting.WebForms" TagPrefix="rsweb" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat ="server" >
protected void ReportViewer1_Load(object sender, EventArgs e)
    {
     this.ReportViewer1.LocalReport.EnableExternalImages = true;
    }</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>无标题页</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$
ConnectionStrings:NorthwindConnectionString %>"
          SelectCommand="SELECT        [CategoryID],        [CategoryName],
[Description], [Picture] FROM [Categories]">
       </asp:SqlDataSource>
       <rsweb:ReportViewer    ID="ReportViewer1"    runat="server"    Font-
Names="Verdana" Font-Size="8pt"
          Height="400px" OnLoad="ReportViewer1_Load" Width="400px">
          <LocalReport ReportPath="Report1.rdlc">
             <DataSources>
                <rsweb:ReportDataSource   DataSourceId="ObjectDataSource1"
Name="DataSet1_Categories" />
             </DataSources>
          </LocalReport>
       </rsweb:ReportViewer>
       <asp:ObjectDataSource      ID="ObjectDataSource1"      runat="server"
SelectMethod="GetData"
         TypeName="DataSet1TableAdapters.CategoriesTableAdapter">
</asp:ObjectDataSource>
       </div>
   </form>
</body></html>
```

显示效果如图 6-15 所示。

3．正确使用 GridView、Repeater 和 DataList 控件

数据绑定控件 GridView、Repeater 和 DataList 是显示数据的有力控件。其中 GridView 是迄今为止功能最为丰富的数据显示控件，大部分功能可通过属性设置来完成，甚至不需

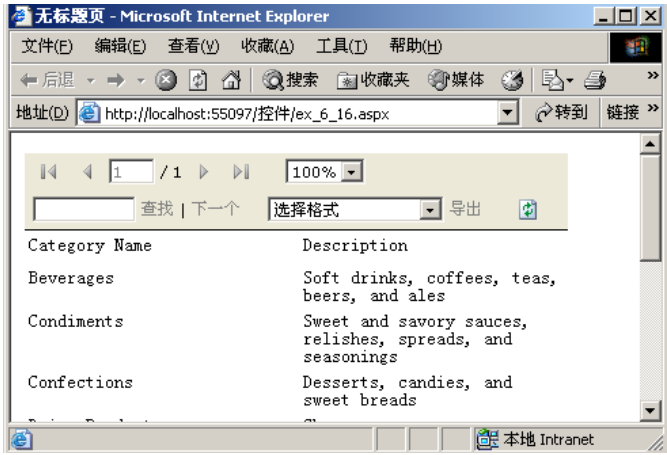


图 6-15　ReportViewer 控件示例效果图

要编写一行代码就能实现强大的数据处理功能。许多初学者在进行 Web 开发时，只要遇到数据处理或显示就习惯性地使用 GridView 控件。虽然使用 GridView 大大减少了开发者的编程工作量，但最大的问题就是该控件在处理数据时需要占用很多 Web 服务器资源，生成在客户端呈现的 HTML 文件也非常大，而且只能以表格形式输出数据，最终导致系统响应性能降低。我们建议在用户数据量不大且不需要出色的界面效果、特别是当需要编辑、分页、排序功能的时候，使用该控件有事半功倍的效果。但当数据量很大的时候，要求系统有较快的处理性能，具有更好的显示效果的时候推荐使用 Repeater 控件，Repeater 控件允许对用户通过模板自定义数据项的输出，处理速度非常快。但 Repeater 控件的缺点是，必须在各个模板中进行手工编写代码，对于分页、排序、编辑等都必须编写代码。DataList 控件的使用性能介于 GridView 和 Repeater 之间。可在 DataList 中通过属性设置定义各个模板，然后部分地通过编程实现分页、排序、数据编辑等功能。

6.3　Web.config 文件的配置

ASP.NET 的应用程序配置文件 web.config 是基于 XML 格式的纯文本文件，存在于应用的各个目录下。它决定了站点所在目录及其子目录的配置信息，并且子目录下的配置信息覆盖其父目录的配置。

Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG 下的 web.config 为整个机器的根配置文件，它定义了整个环境下的默认配置。实际上它继承了 Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\Machine.config 的配置设置和默认值。Machine.config 文件用于服务器级的配置设置。其中的某些设置不能在位于层次结构中较低级别的配置文件中被重写。

在运行状态下，ASP.NET 会根据远程 URL 请求，把访问路径下的各个 web.config 配置文件叠加，产生一个唯一的配置集合。例如，一个对 URL：http://localhost/webapp/mydir/test.aspx 的访问，ASP.NET 会根据以下顺序来决定最终的配置情况：

Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\web.config（默认配置）；

～/web.config（应用程序主目录下的配置）；

～/mydir/web.config（当前自己的配置）。

配置内容被置于 web.config 文件中的标记<configuration>和</configuration>之间。对于一般性 Web 开发应用来说，一般仅对～/web.config 文件进行配置。一个典型的配置文件内容如例 6.18 所示。

【例 6.18】 一个典型的配置文件内容如下：

```
<?xml version="1.0"?>
<configuration>
   <appSettings>
       <add key="dsn" value="localhost;uid=MyUserName;pwd=;"/>
       <add key="msmqserver" value="server\myqueue"/>
   </appSettings>
   <connectionStrings>
       <add     name="NorthwindConnectionString"     connectionString="Data
Source=WCLNOTE;Initial  Catalog=Northwind;Persist  Security  Info=True;User
ID=sa;Password=docman" providerName="System.Data.SqlClient"/>
   </connectionStrings>
   <system.web>
       <authentication mode="Forms">
           <forms loginUrl="logon.aspx" name=".FormsAuthCookie"/>
           <authorization>
              <allow users="*"/>   <!-- Allow all users -->
             <!-- Allow or deny specific users.
                   allow users="[comma separated list of users]"
                   roles="[comma separated list of roles]"/>
               <deny users="[comma separated list of users]"
                     roles="[comma separated list of roles]"/>  -->
       </authorization>
       </authentication>
       <customErrors     mode="RemoteOnly"     defaultRedirect="~/ErrorPage/
ErrorPage.htm">
           <error statusCode="403" redirect="NoAccess.htm"/>
           <error statusCode="404" redirect="FileNotFound.htm"/>
       </customErrors>
       <httpRuntime maxRequestLength="4096" executionTimeout="60"
                       appRequestQueueLimit="100"/>
       <pages buffer="true" enableViewState="true" enableSessionState="true" />
       <sessionState sqlConnectionString="data source=127.0.0.1;user
           id=sa;password=" cookieless="false" timeout="10"/>
       <trace enabled="false" requestLimit="10" pageOutput="false"
           traceMode="SortByTime" localOnly="true"/>
       <!--  全球化此节设置应用程序的全球化设置。-->
```

```
      <globalization requestEncoding="utf-8" responseEncoding="utf-8"/>
      <compilation debug="true"/></system.web>
</configuration>
```

要深刻理解和领会 Web.config 的使用对初学者来说是困难的。对于一般性的应用程序来说应掌握以下几个配置用法。

（1）<appSettings>节

作用：存储用户应用程序中的键值对。例如下面定义了一个键值对：

```
<add key="myURL" value="http://www.cqu.edu.cn"/>
```

在后台服务器程序中可以通过使用 ConfigurationManager.AppSettings 静态字符串集合来访问。例如在后台 C#程序中要取到键 myURL 的值，代码为：

```
string   myURL   =   System.Configuration.ConfigurationManager.AppSettings
["myURL"];
Response.Write(myURL);   //输出 "http://www.cqu.edu.cn"
```

可以将应用程序中的参数通过 appSettings 节放在配置文件中，以后参数值变化，只需要修改配置文件中的参数值即可。

（2）<connectionStrings>节

作用：存储应用程序中访问数据库的相关连接参数。例如下面定义了一个连接数据库 Northwind 的相关参数，包括三个部分，分别是 name、connectionString 和 providerName：

```
<add        name="NorthwindConnectionString"        connectionString="Data
Source=WCLNOTE;Initial
Catalog=Northwind;Persist Security Info=True;
User ID=sa;Password=docman"
providerName="System.Data.SqlClient"/>
```

在后台 C#程序中要取到上述三个参数的值，代码为：

```
string        MyName        =        ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].Name;
string        myConn        =        ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].ConnectionString;
string     myProviderName     =     ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].ProviderName;
Response.Write(MyName +","+ myConn +","+ myProviderName);
```

若存在多个数据库配置参数设置，可通过下面的 C#代码获得每个数据库连接参数：

```
  // Get the connectionStrings.
       ConnectionStringSettingsCollection connectionStrings =
```

```
                        ConfigurationManager.ConnectionStrings;
 // Get the collection enumerator.
       System.Collections.IEnumerator connectionStringsEnum =
                        connectionStrings.GetEnumerator();
  // Loop through the collection and display the connectionStrings key, value
pairs.
       int i = 0;
       Response.Write("<br/>Connection strings.<br/>");
       while (connectionStringsEnum.MoveNext())
       {   string name = connectionStrings[i].Name;
           Response.Write("Name:"+name+" Value:"+connectionStrings[name]);
           i += 1;
       }
```

（3）<authentication> 节

作用：配置 ASP.NET 身份验证支持，例如以下代码实现了当没有经过登录验证的用户访问需要身份验证的网页时，网页自动跳转到登录网页 logon.aspx。

```
<authentication mode="Forms" >
     <forms loginUrl="logon.aspx" name=".FormsAuthCookie"/>
</authentication>
```

（4）<authorization> 节

作用：控制对 URL 资源的客户端访问（如允许匿名用户访问）。此元素必须与<authentication>节配合使用。例如以下配置禁止匿名用户的访问：

```
<authorization>
     <deny users="?"/>
</authorization>
```

（5）<compilation>节

作用：配置 ASP.NET 使用的所有编译设置。默认的 debug 属性为 True。在程序编译完成交付使用之后应将其设为 True。

（6）<customErrors>节

作用：为 ASP.NET 应用程序提供有关自定义错误信息的信息。它不适用于 XML Web Services 中发生的错误。例如当网页发生错误时，将网页跳转到自定义的错误页面：

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

其中元素 defaultRedirect 表示自定义的错误网页的名称。mode 元素表示对不在本地 Web 服务器上运行的用户显示自定义信息。

（7）<httpRuntime>节

作用：配置 ASP.NET HTTP 运行库设置。例如控制用户上传文件最大为 4 MB，最长时间为 60s，最多请求数为 100：

```
<httpRuntime          maxRequestLength="4096"          executionTimeout="60"
appRequestQueueLimit="100"/>
```

（8）<pages>

作用：标识特定于页的配置设置，如是否启用会话状态、视图状态、是否检测用户的输入等。例如：

```
<pages buffer="true" enableViewState="true" enableSessionState="true" />
```

（9）<sessionState>

作用：为当前应用程序配置会话状态设置，如设置是否启用会话状态、会话状态保存位置等。例如：

```
<sessionState mode="InProc" cookieless="true" timeout="20"/>
```

注：mode="InProc"表示在本地储存会话状态（你也可以选择储存在远程服务器或不启用会话状态等）；cookieless="true"表示如果用户浏览器不支持 Cookie 时启用会话状态（默认为 False）；timeout="20"表示会话可以处于空闲状态的分钟数。

（10）<trace>

作用：配置 ASP.NET 跟踪服务，主要用来测试和判断程序哪里出错。例如：

```
<trace      enabled="false"      requestLimit="10"      pageOutput="false"
traceMode="SortByTime" localOnly="true" />
```

注：enabled="false"表示不启用跟踪；requestLimit="10"表示指定在服务器上存储的跟踪请求的数目。pageOutput="false"表示只能通过跟踪实用工具访问跟踪输出；traceMode="SortByTime"表示以处理跟踪的顺序来显示跟踪信息；localOnly="true" 表示跟踪查看器只用于宿主 Web 服务器。

6.4　ASP.NET 内置服务器对象与 Global.asax 文件

6.4.1　ASP.NET 内置服务器对象

ASP.NET 提供了如表 6-32 所示的常用内置服务器对象。这些对象提供了相当多的功能，例如可以在两个网页之间传递变量、输出数据以及记录变量值等。这些对象可以在后台服务器代码中直接使用，不需要作任何说明。

表 6-32　ASP.NET 常用内置服务器对象

对 象 名 称	说　明
Page 对象	用来指代 Web 窗体，设置或执行与 Web 窗体有关的属性、方法和事件
Response 对象	决定服务器在什么时候或如何输出数据到客户端
Request 对象	用来捕获由客户端返回到服务器的数据
Server 对象	获取 Web 服务器对象的各项参数
Application 对象	处理由不同客户端共享的变量
Session 对象	处理由各个客户端专用的变量
Cookies 对象	为 Web 应用程序保存访问者的信息

下面分别对 ASP.NET 的这些内置对象进行介绍。

（1）Page 对象

Page 对象指代 Web 窗体，它提供了许多属性、方法和事件。在执行一个 Web 窗体时，首先会进行网页的初始化，触发 Page 对象的 Init 事件；然后加载网页，触发 Load 事件；再加载服务器控件，根据客户端的请求触发服务器控件的事件；访问结束并且信息被写回客户端后，触发 Unload 事件；如果在访问过程中发生异常，则会触发 Error 事件。

Page 对象的一个常用的属性是 IsPostBack，它判断网页是否已被加载过，第一次加载网页的时候为 false，如果已经加载过，则为 true。利用此属性，可以在 Page 的 Load 事件中编写如下代码：

```
public partial class _default: System.Web.UI.Page
{   protected void Page_Load(object sender, EventArgs e)
    { //第一次加载此页面显示 OK 对话框，以后点击 Web 窗体按钮重新装载此页面将不再显示
        if (!Page.IsPostBack)   {
            string myScript = "<script>alert('ok');</script>";
            Page.RegisterClientScriptBlock("myscript1",myScript);
        }
    }
}
```

Page 对象的另一个常用的属性是 ErrorPage，可以获取或设置当页面发生未处理的异常情况时将用户请求定向到错误信息页面。

Page 对象的常用方法包括：

MapPath(VirtualPath)，将 VirtualPath 指定的虚拟路径转换成实际路径。

RegisterClientScriptBlock(key,script)，发送客户端脚本 script 给浏览器。

RegisterOnSubmitStatement(key,script)，设置当客户端发生 OnSubmit 事件时所要执行的 script 代码。

（2）Response 对象

Response 对象派生自 HttpResponse 类。它的主要功能是将服务器端的数据输出到客户端。它属于 Page 对象的成员，所以不用定义便可直接使用。Response 对象的常用属性及说明如表 6-33 所示，Response 对象常用方法及说明如表 6-34 所示。

表 6-33　Response 对象常用属性及说明

序号	属　性	说　明
1	Cache	获取 Web 页的缓存策略
2	Charset	设定或获取 HTTP 的输出字符编码
3	Expires	获取或设置在浏览器上缓存的页过期之前的分钟数
4	Cookies	获取当前请求的 Cookie 集合
5	IsClientConnected	传回客户端是否仍然和 Server 连接
6	SuppressContent	设定是否将 HTTP 的内容发送至客户端浏览器

表 6-34　Response 对象常用方法及说明

序号	方　法	说　明
1	AppendToLog	将自定义日志信息添加到 IIS 日志文件
2	Clear	清除缓冲区的内容
3	End	将目前缓冲区中所有的内容发送至客户端然后关闭
4	Flush	将缓冲区中的所有数据发送至客户端
5	Redirect	将当前页面重新导向至另一个地址的页面
6	Write	将数据输出到 HTTP 响应输出流
7	WriteFile	将指定的文件直接写入 HTTP 响应输出流

【例 6.19】 利用 Response 对象将本地的某个文件的内容写入 HTTP 输出流，并显示“写入成功”字样。在 VS 2010 中新建一个空的 Web 窗体 ex_6_19.aspx，ex_6_19.aspx.cs 文件的代码为：

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class _Default : System.Web.UI.Page
{    //加载页面_Default.aspx 时触发事件
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.WriteFile(@"E:\Example\Sample.txt");
        Response.Write("<br>");
        Response.Write("写入成功！");
    }
}
```

（3）Request 对象

Request 对象派生自 HttpRequest 类，用来捕获由客户端返回服务器端的数据，比如用户输入的表单数据、保存在客户机上的 Cookie 等。

① 读取表单数据

利用 Request 对象读取表单数据的方式取决于表单数据返回服务器的方式，其方式有两种：post 和 get，使用哪种方式是通过设置 Method 属性来实现的。当为 get 时，表单数据将

以字符串的形式附加在网址的后面返回服务器端，此时应用 Request 对象的 QueryString 属性来获取表单数据，例如 Request.QueryString("UserName")；当为 post 时，表单数据将存放在 HTTP 报头中，此时应使用 Request 对象的 Form 属性来获取表单数据，例如 Request.Form("UserName")；如果想避免因为 QueryString 属性和 Form 属性使用不当而造成错误，可以直接使用 Params，例如 Request.Params("UserName")；也可省略所有属性，直接读取表单数据 Form，例如 Request("UserName")。

【例 6.20】 用 Request 对象获取表单信息，并将信息在页面中输出 ex_6_20.aspx 代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_20.aspx.cs"
Inherits=" ex_6_20" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"> <title>无标题页</title></head>
<body style="text-align: center">
   <form id="form1" runat="server" method="post" action="test.aspx">
      <table>
         <tr><td style="width: 100px"> 姓名</td>
            <td style="width: 100px">
               <asp:TextBox ID="name" runat="server"></asp:TextBox></td>
         </tr><tr>
            <td style="width: 100px"> 邮箱</td>
            <td style="width: 100px">
               <asp:TextBox ID="email" runat="server"></asp:TextBox></td>
         </tr> <tr>
            <td colspan="2">
               <asp:Button ID="Button1" runat="server" OnClick=
               "Button1_Click" Text="Button" />
            </td> </tr>
      </table>
   </form>
</body>
</html>
```

ex_6_20.aspx.cs 代码如下：

```
using System;
using System.Data;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class ex_6_20 : System.Web.UI.Page
{
   protected void Button1_Click(object sender, EventArgs e)
   {  string name = Request["name"].ToString();//通过 Request 对象获取姓名信息;
      string email = Request["email"].ToString();//通过 Request 对象获取邮箱信息;
      Response.Write("您的姓名是: " + name + " 您的邮箱是" + email);//在页面输出信息;
   }
}
```

② 获取客户端浏览器信息

Request 对象的 Browser 属性可以获取 HttpBrowserCapabilities 对象，用来获取当前和服务器连接的浏览器的信息。

【例 6.21】 用 Request 对象获取浏览器信息。ex_6_21.aspx.cs 代码如下：

```
using System;
using System.Data;
using System.Web.UI;
public partial class Ex_6_21: System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {   string info = "浏览器信息:<br> 浏览器:" + Request.Browser.Type + "<br>"
+ "名称:" + Request.Browser.Browser + "<br> 版本:" + Request.Browser.Version
+ "<br>使用平台:" + Request.Browser.Platform;
        this.lab_info.Text = info;
    }
}
```

③ 获取服务器端的相关信息

Request 对象的 ServerVariables 属性可以获取服务器端有关信息，下面的代码输出服务器端所有信息和仅输出服务器的 IP 地址：

```
string s1 = Request.ServerVariables.ToString();
Response.Write(Server.Urldecode(s1)); //输出服务器端的所有信息
string s1 = Request.ServerVariables["REMOTE_ADDR"].ToString();
Response.Write(Server.Urldecode(s1)); //输出服务器的 IP 地址
```

ServerVariables 属性中的常用服务器环境变量如表 6-35 所示。

表 6-35　常用服务器环境变量

变　量	说　明
APPL_PHYSICAL_PATH	检索与元数据库路径相应的物理路径。IIS 通过将 APPL_MD_PATH 转换为物理（目录）路径以返回值
CONTENT_LENGTH	客户端发出内容的长度
CONTENT_TYPE	内容的数据类型。同附加信息的查询一起使用，如 HTTP 查询 GET、POST 和 PUT
GATEWAY_INTERFACE	服务器使用的 CGI 规格的修订。格式为 CGI/revision
LOCAL_ADDR	返回接受请求的服务器地址。如果在绑定多个 IP 地址的多宿主机器上查找请求所使用的地址时，这条变量非常重要
LOGON_USER	用户登录 Windows NT®的账号
QUERY_STRING	查询 HTTP 请求中问号（?）后的信息
REMOTE_ADDR	发出请求的远程主机的 IP 地址

续表

变　量	说　明
REMOTE_HOST	返回发出请求的主机名称。如果服务器无此信息，它将设置为空的 MOTE_ADDR 变量
REMOTE_USER	返回请求的用户名
REQUEST_METHOD	该方法用于得到请求方法。如 HTTP 的 GET、POST 等
SCRIPT_NAME	执行脚本的虚拟路径。
SERVER_NAME	服务器主机名或 IP 地址
SERVER_PORT	发送请求的端口号
URL	提供 URL 的基本部分

（4）Application 对象

Application 对象可以生成一个 Web 应用程序能共享的变量，所有访问这个网站的用户都可以共享此变量。Application 对象常用的属性及说明如表 6-36 所示。

表 6-36　Application 对象常用属性及说明

序　号	属　性	说　明
1	AllKeys	返回全部 Application 对象变量名到一个字符串数组中
2	Count	获取 Application 对象变量的数量
3	Item	允许使用索引或 Application 变量名称传回内容值

Application 对象常用方法及说明如表 6-37 所示。

表 6-37　Application 对象常用方法及说明

序号	方　法	说　明
1	Add	新增一个 Application 对象变量
2	Clear	清除全部 Application 对象变量
3	Lock	锁定一个或全部 Application 对象变量
4	Remove	使用变量名称移除一个 Application 对象变量
5	RemoveAll	移除全部 Application 对象变量
6	Set	使用变量名称更新一个 Application 对象变量的内容
7	Unlock	解除锁定的 Application 对象变量

【例 6.22】 通过使用 Application 对象向 Web 中添加三个对象变量，并将这些变量名称及值显示在 Web 页面。实现这一功能的 ex_6_22.Aspx.cs 文件代码如下：

```
using System;
using System.Data;
using System.Web;
using System.Web.UI;
public partial class _Default : System.Web.UI.Page
{   //加载页面_Default.aspx 时触发事件
```

```
    protected void Page_Load(object sender, EventArgs e)
    {   Application.Add("AppOne", "school");
        Application.Add("AppTwo", "university");
        Application.Add("AppThree", "college");
        for (int i = 0; i < 3; i++)
        {   Response.Write("变量名:" + Application.GetKey(i));
            Response.Write(",值:" + Application[i] + "<p>");   }
    }
}
```

运行后的效果如图 6-16 所示。

由于 Application 对象属于网站的全局变量，因此经常会出现多人同时访问和修改该对象的情况。为了避免多个用户同时修改 Application 对象，需要在修改前用 Lock 锁定 Application 对象，修改后再用 Unlock 解锁 Application 对象。用 Application 对象可实现网站访问记数器功能。由于 Application 对象一直会占用服务器资源，因此应用 Remove 等可及时清理无用的 Application 变量。

图 6-16　Application 对象的应用

（5）Session 会话对象

Session 对象的功能和 Application 对象的功能相似，都是用来存储跨网页程序的变量（包括对象）。但 Session 对象只针对单一访问网站的用户，用 Session 对象定义的变量可在某个用户打开的多个网页之间共享，相当于网站的“局部”变量。在服务器上将保存各个联机的机器的 Session 对象变量，不同的联机客户间无法相互存取。Application 对象变量终止于停止 IIS 服务，但是 Session 对象变量终止于联机机器离线时，也就是当网页用户关掉浏览器或超过设定 Session 变量对象的有效时间时，Session 对象变量就会消失。Session 对象属于 Page 对象的成员，可以直接使用。使用 Session 对象存放信息的语法如下：

```
Session["变量名"]="内容"
```

为了从会话中读取信息，可以使用以下语句：

```
VariableName=Session["变量名"];
```

【例 6.23】 使用 Session 对象记录访问页面的次数。Ex_6_23.aspx.cs 文件代码如下：

```
using System;
using System.Data;
using System.Collections;
using System.Web;
```

```
using System.Web.UI;
public partial class ex_6_23 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {  Session.Timeout = 1;//设置 Session 对象的有效期.本例是一分钟,在一分钟内不做任何操作将自动销毁
        if (Session["count"] != null)
        {  Session["count"]=(int)Session["count"]+1;
        }
        else
        {  Session["count"]=1;
        }
        Response.Write("您是第" + Session["count"].ToString() + "次访问本站");
    }
}
```

以间隔不超过 1min 的频率不停刷新，访问次数就会不停地加 1。

在应用程序中，人们需要在跳转的页面之间传递变量。例如采用 Session 对象记录用户登录的信息，在每个访问页面中检查 Session 对象变量，若没有登录信息，则转向登录界面。采用 Session 对象可以很方便地实现各个页面之间变量的共享，但 Session 变量的使用也给 Web 服务器增加了不少负荷，因此不可滥用 Session 对象。应注意以下几点：

① 不要在 Session 中存放大的数据对象，否则每次读写时服务器都要序列化，会大大影响系统性能；

② 当不需要 Session 时，应尽快用 Remove 将它从服务器内存中释放；尽量将 Session 的超时时间设短一点。超时的含义是指如果在限定的时间内没有使用该变量，则大于设定时间就将该变量从内存中释放。

（6）Server 对象

Server 对象也是 Page 对象的成员之一，主要提供一些处理网页请求时所需的功能，例如获取有关最新错误的信息、对 HTML 文本进行编码和解码、访问和读写服务器端文件等。Server 对象常用方法及说明如表 6-38 表示。

表 6-38　Server 对象常用方法及说明

方　法	说　明
HtmlEncode	对要在浏览器中显示的字符串进行编码。例如： Response.Write("hello"); 将在浏览器中输出"Hello"，而 Response.Write(Server.HtmlEncode("hello")); 将在浏览器中输出"hello"
HtmlDecode	对已被编码以消除无效 HTML 字符的字符串进行解码
MapPath	返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径

续表

方　法	说　明
UrlEncode	编码字符串。在使用网址进行数据传递时，一些特殊符号如"@"、"&"等将不能正常处理，必须经过编码后才可以顺利读取。例如： Response.Redirect("a.aspx?email="+Server.UrlEncode("pp@qq.com")); 在 a.aspx 的后台代码中可正确获取 email 的值： string email=Request["email"];
UrlDecode	对字符串进行解码
Transfer(url)	终止当前页的执行，用 URL 指定的路径执行新的页面
Execute(url)	终止当前页的执行，用 URL 指定的路径执行新的页面，执行完后又返回当前页

用 Response.Redirect 和 Server.Transfer 都能重定向网页，它们的用法是有区别的。Response.Redirect 是通过浏览器执行的，因此在服务器和浏览器之间产生额外的往返过程，可以实现将页面重定向到任何其他页面，包括非本网站的页面，并且浏览器网址会变成跳转后的页面地址；而 Server.Transfer 通过更改服务器端“焦点”和传输请求来代替告诉浏览器重定向，这就意味着不会占用较多的 HTTP 请求，因此这可以减轻服务器的压力，使服务器运行更快。它只能在本网站内切换到同目录或者子目录下的网页，用户浏览器的网址没有变化，即隐藏了新网页的地址及附带在地址后边的参数值，具有数据保密功能。

Server.Execute 方法允许当前的 ASPX 页面执行一个同一 Web 服务器上的指定 ASPX 页面，当指定的 ASPX 页面执行完毕，控制流程重新返回原页面后继续执行。这种页面导航方式类似于针对 ASPX 页面的一次函数调用，被调用的页面能够访问发出调用页面的表单数据和查询字符串集合，所以要把被调用页面 Page 指令的 EnableViewStateMac 属性设置成 False。

Server 对象的常用属性及说明如表 6-39 所示。

表 6-39　Server 对象常用属性及说明

属　性	说　明
MachineName	获取服务器的计算机名称
ScriptTimeout	获取和设置请求超时值（以秒来计算）

【例 6.24】利用 Server 对象获得服务器的计算机名称和服务器上某个文件的物理路　径。其 ex_6_24.aspx.cs 文件的代码如下：

```
using System;
using System.Data;
using System.Web;
using System.Web.UI;
public partial class ex_6_24: System.Web.UI.Page
{   //加载页面_Default.aspx 时触发事件
    protected void Page_Load(object sender, EventArgs e)
    {   Response.Write(Server.MachineName);
        Response.Write("<p>");
        Response.Write(Server.MapPath("ObjectWeb.aspx"));
    }
}
```

（7）Cookie 对象

Cookie 对象是 HttpCookieCollection 类的一个实例。它用于保存客户端浏览器请求的服务器页面，也可以用它存放非敏感性的用户信息，信息保存的时间可以根据需要设置。所有信息都将保存在客户端。要存储一个 Cookie 变量，可以通过 Response 对象的 Cookies 集合，其语法如下：

```
Response.Cookies[Name].Value="内容";
```

而要取回 Cookie，则要使用 Request 对象的 Cookies 集合，其语法如下：

```
VariableName=Request.Cookies[Name].Value;
```

Cookie 对象的常用属性及说明如表 6-40 所示。

表 6-40　Cookie 对象的常用属性及说明

序　号	属　　性	说　　明
1	Expires	设定 Cookie 变量的有效时间，默认为 1000 min
2	Name	获取 Cookie 变量的名称
3	Value	获取或设置 Cookie 变量的内容
4	Path	获取或设置 Cookie 使用的 URL

【例 6.25】 该例演示了如何设置一个 Cookie，如何得到设置的 Cookie。ex_6_25.aspx 页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ex_6_25.aspx.cs"
Inherits="ex_6_25"%>
<html><head runat="server"><title>无标题页</title></head>
<body>
   <form id="form1" runat="server">
      <asp:Button     ID="Button1"     runat="server"     Text="SetCookie"
OnClick="Button1_Click" />
      <asp:Button     ID="Button2"     runat="server"     Text="GetCookie"
OnClick="Button2_Click" />
   </form>
</body>
</HTML>
```

ex_6_25.aspx.cs 文件代码如下：

```
using System;
using System.Data;
using System.Configuration;
```

```
using System.Web;

public partial class ex_6_25 : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    { //定义新的 Cookie 对象，其名称为 LastVisit
        HttpCookie MyCookie = new HttpCookie("LastVisit");
        DateTime now = DateTime.Now;

        MyCookie.Value = now.ToString();
        MyCookie.Expires = now.AddHours(1); //设置失效日期
        Response.Cookies.Add(MyCookie);     //写入 Cookie

        MyCookie.Values.Add("email", "w@qq.com"); //在 Cookie 中添加新的内容
        Response.AppendCookie(MyCookie);
     }
    protected void Button2_Click(object sender, EventArgs e)
    {  //获取 Cookie 的内容
        string ss=Request.Cookies["LastVisit"].Values.ToString();
        Response.Write(Server.UrlDecode(ss));
    }
}
```

6.4.2　Global.asax 文件

在 VS 2010 中添加新项，选择“全局应用程序类”后将在应用程序根目录下生成一个 Global.asax 文件。它实际上是一个可选文件。删除它不会出问题——当然是在你没有使用它的情况下。Global.asax 文件存放 Web 应用程序的应用级事件处理程序，具体来说包括以下事件处理程序。

- Application_Init：在应用程序被实例化或第一次被调用时，该事件被触发。
- Application_Disposed：在应用程序被销毁之前触发。这是清除以前所用资源的理想位置。
- Application_Error：当应用程序中遇到一个未处理的异常时，该事件被触发。
- Application_Start：在应用程序第一个实例被创建时，该事件被触发。可以理解为网站被第一个用户访问时触发此事件，以后将不会再触发该事件。
- Application_End：当应用程序的最后一个实例被销毁时，该事件被触发。可以理解为网站的最后一个用户关闭浏览器停止访问时触发此事件。在一个应用程序的生命周期内它只被触发一次。
- Application_BeginRequest：在接收到一个应用程序请求时触发。对于一个请求来说，它是第一个被触发的事件，请求一般是用户输入的一个页面请求（URL）。
- Application_EndRequest：针对应用程序请求的最后一个事件。
- Session_Start：在一个新用户访问应用程序 Web 站点时，该事件被触发。

- Session_End：在一个用户的会话超时、结束或他们离开应用程序 Web 站点时，该事件被触发。

【例 6.26】 下面是一个 Global.asax 文件的例子。在应用程序启动时设置一个计数器，每当有一个用户访问网站时计数器加 1。代码如下：

```
<script runat="server">
    void Application_Start(object sender, EventArgs e)
    {  Application["count"] = 0;//设置初始值为 1
    }
    void Application_End(object sender, EventArgs e)
    {    //  在应用程序关闭时运行的代码
    }
    void Application_Error(object sender, EventArgs e)
    {  // 在出现未处理的错误时运行的代码
    }
    void Session_Start(object sender, EventArgs e)
    { Application.Lock;
      Application["count"] = (int)Application["count"] + 1;//每产生一个访问加 1
     Application.Unlock; }
    void Session_End(object sender, EventArgs e)
    {   // 在会话结束时运行的代码
        // 注意: 只有在 Web.config 文件中的 sessionstate 模式设置为
        // InProc 时，才会引发 Session_End 事件。如果会话模式设置为 StateServer
        // 或 SQLServer，则不会引发该事件
    }
</script>
```

6.5　ADO.NET 数据库访问技术

ADO.NET 是微软在.NET 平台下的一种全新的数据库访问机制。与 ADO 相比，ADO.NET 满足了 ADO 无法满足的三个重要需求：为适应 Web 环境的编程需要提供了断开的数据访问模型；提供了与 XML 的紧密集成；提供了与.NET 框架的无缝连接。在性能上，由于 ADO 使用 COM 来处理记录集，当记录集内的值转换为 COM 可识别的数据类型时会导致显著的处理开销，而 ADO.NET 的数据集不需要进行数据类型转换，其性能优于 ADO 的记录集。在应用程序可伸缩性上，由于 ADO.NET 为断开式 n 层编程环境提供了很好的支持，这使得应用程序的可伸缩性大为增强。ADO.NET 中有两个核心组成部分，分别是.NET 框架下的数据提供程序 Data Provider 和数据集 DataSet。数据提供程序包括以下 4 个核心对象。

- Connection：建立与特定数据源的连接。
- Command：对数据源执行数据库命令，用于返回和修改数据、运行存储过程等。
- DataReader：从数据源中获取高性能的数据流，例如只进且只读数据流。
- DataAdapter：用数据源填充 DataSet，并可处理数据的更新。

DataSet 是 ADO.NET 的断开式结构的核心组件。设计 DataSet 的目的是为了实现独立于任何数据源的数据访问。因此，它可以用于多种不同的数据源，用于 XML 数据，或用于管理应用程序本地数据。DataSet 是一个包含一个或多个 DataTable 对象的集合，这些对象由数据行和数据列以及主键、外键、约束和有关 DataTable 对象中数据关系组成。

ADO.NET 中有许多对象与 ADO 中的对象功能相似，但 ADO.NET 中的对象功能更加强大。同时，除了 Connection、Command 对象外，ADO.NET 还添加了许多新的对象和程序化接口，如 DataSet、DataReader、DataView、DataAdapter 等，使得对数据库的操作更加简单。图 6-17 为一个简化的 ADO.NET 对象模型。

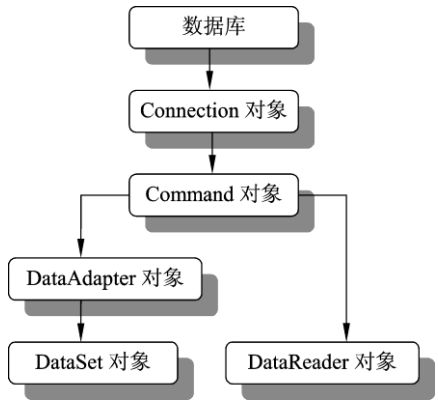


图 6-17　简化的 ADO.NET 对象模型

ASP.NET 数据库应用程序的开发流程有以下几个步骤：

① 利用 Connection 对象创建数据连接。

② 利用 Command 对象对数据源执行 SQL 命令并返回结果。

③ 利用 DataReader 对象读取数据源的数据。

④ DataSet 对象是 ADO.NET 的核心，与 DataAdapter 对象配合，完成数据库操作的增加、删除、修改、更新等操作。

在 ADO.NET 中，连接数据库进行访问通常有三种方式：ODBC、OLEDB、SQLClient。其中 ODBC 和 OLEDB 可用来连接各种数据库系统，而 SQLClient 提供专用连接方式，只能用来连接微软 SQL Server 7.0 及以上数据库。在应用程序中使用三种数据连接方式之一时，必须在后台代码中引用对应的名称空间，对象的名称也随之发生变化，如表 6-41 所示。

表 6-41　数据连接方式名称空间及对应的对象名称

名 称 空 间	对应的对象名称
System.Data.Odbc	OdbcConnection ； OdbcCommand ； OdbcDataReader ； OdbcDataAdapter
System.Data.OleDb	OleDbConnection ； OleDbCommand ； OleDbDataReader ； OleDbDataAdapter
System.Data.SqlClient	SqlConnection；SqlCommand；SqlDataReader；SqlDataAdapter

虽名称空间不同但编程方法相同。下面仅讨论在 System.Data.SqlClient 名称空间下数据库操作对象的使用。

6.5.1　Connection 对象

数据库应用程序与数据库进行数据交互，首先必须建立与数据库的连接。ADO.NET 中，使用 Connection 对象完成此项功能。在这里要指出的是在创建连接之间要在程序中添加 System.Data 和 System.Data.SqlClient 两个命名空间。建立与数据库的连接是通过数据库连接字符串来实现的，在连接字符串中要设置一些参数的值，连接 SQL Server 数据库的常用参数

如表 6-42 所示。

表 6-42　连接 Sql Server 数据库的常用参数说明

参 数 名 称	默 认 值	参 数 说 明
ConnectionTimeout	15	设置 SqlConnection 对象连接数据库的超时时间，单位为秒，若超时，则返回连接失败
Datasource	无	要连接的数据库服务器实例名称
Server	无	要连接的数据库服务器名称
Addr	无	要连接的数据库服务器名称的地址
Network Address	无	要连接的数据库服务器名称的地址
Initial Catalog	无	设置连接数据库的名称
Database	无	设置连接数据库的名称
Integrated Security	False	设置是否使用信任连接，可以设置的值有 True、False 和 SSPI，SSPI 和 True 相同，表示使用信任连接
Trusted_Connection	False	设置是否使用信任连接，可以设置的值有 True、False 和 SSPI，SSPI 和 True 相同，表示使用信任连接
Packet Size	8192	设置用来与数据库通信的网络数据包的大小，单位为 byte,有效值为 512-32767
Password	无	设置登录数据库的密码
User ID	无	设置登录数据库的用户账号
Workstation ID	计算机名称	设置连接到数据库服务器工作站的名称

例如，下面的连接字符串用来打开 SQL Server 中 NorthWind 数据库：

```
    SqlConnection conn = new SqlConnection();
   conn.ConnectionString="Data                    Source=127.0.0.1;Initial
Catalog=Northwind;Persist Security Info=True;User ID=sa;Password=docman ";
```

Connection 对象常用的方法有如下几种。

- BeginTransaction：开始记录数据库事务日志。
- ChangeDatabase(database)：将目前的数据库更改为参数 database 指定的数据库。
- Close：关闭数据库连接，数据源使用完毕后必须关闭数据库连接。
- CreateCommand：创建并返回与 Connection 对象有关的 Command 对象。
- Open：打开数据库连接。

常用的 Connection 对象的事件主要有两个，如表 6-43 所示。

表 6-43　Connection 对象的事件

事 件 名 称	说　明
InfoMessage	数据提供程序发送警告或其他信息时触发此事件。InfoMessage 事件的参数为 SqlInfoMessageEventArgs，有 Errors、Messages 和 Source 三个属性，分别用来获取数据源传出的警告集合、由数据源传出的错误全文以及发生错误的对象名称
StateChange	当数据连接状态改变时触发此事件。此事件的参数为 StateChangeEventArgs，它有两个属性：CurrentState 和 OriginalState。前者获取 Connection 对象连接的新状态，返回 1 表示 Open，返回 0 表示 Close。后者获取 Connection 对象连接的原始状态，返回 1 表示

	Open，返回 0 表示 Close

下面通过一个例子来讲述应用程序中如何建立与数据库的连接以及打开数据库的　连接。

【例 6.27】 Connection 对象的使用。在 VS 2010 中添加新项 ex_6_27.aspx，后台代码为：

```
    using System;
    using System.Data;
    using System.Data.SqlClient;
    using System.Web;
    using System.Web.UI;
    using System.Web.UI.WebControls;

    public partial class ex_6_27 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            SqlConnection conn = new SqlConnection(); //新建一个 Connection 对象
            conn.ConnectionString = "Data Source=127.0.0.1;Initial Catalog=
Northwind;Persist Security Info=True;User ID=sa;Password=docman";//设置连接
    //字符串
            //引发 StateChange 事件
            conn.StateChange                    +=                    new
StateChangeEventHandler(conn_StateChange);
            conn.Open();
            if (conn.State == ConnectionState.Open)
            {  //判断当前 Sqlconnection 对象的状态,如果打开则在页面输出 1
                Response.Write("1");
                conn.Close();
            }
            if (conn.State == ConnectionState.Closed)
            {   Response.Write("0");
            }
        }
        protected void conn_StateChange(object sender, StateChangeEventArgs
e)
        {   //事件处理程序
    Response.Write("StateChange from " + e.OriginalState + " to " +
    e. CurrentState+"<br>");
        }
    }
```

6.5.2　Command 对象

ADO.NET 提供了两个使用连接对象的类，一个是 DataAdapter 类，DataAdapter 对

象可以用来填充 DataTabel 或 DataSet，另一个就是 Command 类，Command 对象提供的 Execute 方法可以读取数据。Command 对象要与数据库连接方式相匹配，相对于 SqlConnection 采用的是 SqlCommand。常用的创建 Command 对象的语法有三种，如表 6-44 所示。

表 6-44　常用的创建 Command 对象的语法

1．使用无参数	SqlCommand cmd = new SqlCommand();
	cmd.Connection = conn;
	cmd.CommandText = strSQL;

2．使用一个参数	SqlCommand cmd = new SqlCommand(strSQL); cmd.Connection = conn;
3．使用两个参数	SqlCommand cmd = new SqlCommand(strSQL, conn);
4．使用 Connection 对象的 CreateCommand 方法	SqlCommand cmd = conn.CreateCommand(); cmd.CommandText = strSQL;

其中参数 strSQL 指定要执行的 sql 命令，参数 conn 为使用的数据连接对象名。常用的 Command 对象的属性如表 6-45 所示。

表 6-45　Command 对象的常用属性

属性名称	说　明
CommandText	获取或设置要对数据源执行的 sql 命令、存储过程或者数据表名称
CommandTimeout	获取或设置 Command 对象超时时间，单位为秒，默认为 30。如果 Command 对象在 30 秒时间内不执行 CommandText 属性设定的内容便返回失败
CommandType	获取或设置 CommandText 属性代表的意义，可以为 CommandType.StoreProcedure（存储过程）、CommandType.Text 等。默认为 Text
Parameters	获取 Parameter 对象的集合 ParameterCollection
Connection	获取或设置 Command 对象要使用的数据连接，值为 Connection 对象名

常用的 Command 对象的方法如表 6-46 所示。

表 6-46　常用的 Command 对象的方法

方　法	说　明
Cancle	取消 Command 对象的执行
CreateParameter	创建一个 Parameter 对象，对于 SqlCommand 对象创建 SqlParameter 对象；对于 OleDbCommand 对象创建 OleDbParameter
ExecuteNonQuery	执行 CommandText 属性指定的内容，并返回被影响的行数。只有 update、insert、delete 三个 sql 语句会返回被影响的行数，其他的如 Create 返回的都是–1
ExecuteReader	执行 CommandText 属性指定的内容，并创建 DataReader 对象
ExecuteScalar	执行 CommandText 属性指定的内容，返回结果的第一行第一列的值。此方法只用于执行 select 语句
ExecuteXMLReader	如果 Command 对象的 CommandType 属性为合法的包含 FOR XML 子句的 SQL 语句时，可通过此方法来返回一个 XMLReader 对象

下面通过例子演示创建 Command 对象以及执行 Command 对象的方法来对数据库数据执行操作。

【例 6.28】 Command 对象的使用。在 VS 2010 中添加新项 ex_6_28.aspx，后台代码为：

```
    using System;
```

```
    using System.Data;
    using System.Data.SqlClient;
    using System.Web;
    using System.Web.UI;
    public partial class ex_6_28 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {       string  ConnectionString  =  "Data  Source=127.0.0.1;Initial
Catalog=Northwind;Persist Security Info=True;User ID=sa;Password=docman";
    //设置连接字符串
            SqlConnection conn = new SqlConnection(ConnectionString);
            conn.Open();  //连接数据库
            //获取单一值
            GetSumofMoney(conn);
            //获取多行数据
            ReadOrderData(conn);
            //无数据返回,只返回被影响的行数
            UpdateCustomers(conn);
            conn.Close();  //断开数据库连接
        }
        protected void ReadOrderData(SqlConnection conn)
        {
            string StrSQL = "SELECT OrderID, CustomerID FROM dbo.Orders;";
            SqlCommand cmd = new SqlCommand(StrSQL, conn); //创建 Command 对象
            SqlDataReader reader = cmd.ExecuteReader();
            try  {
                while (reader.Read()) //循环读取一条记录，从首记录到末记录
                {     Response.Write(reader[0].ToString()     +     ","     +
reader[1].ToString() + "<br/>");
                }
            }
            finally
            {  // Always call Close when done reading.
                reader.Close();
            }
        }
        protected void GetSumofMoney(SqlConnection conn)
        {
            string strSQL = "SELECT SUM(UnitPrice * Quantity) " +
                  " FROM Orders INNER JOIN [Order Details]" +
                " ON Orders.OrderID = [Order Details].OrderID" +
```

```
            " WHERE CustomerID = 'ALFKI'";
        SqlCommand cmd = new SqlCommand(strSQL, conn); //创建 Command 对象
        decimal decOrderTotal = (decimal)cmd.ExecuteScalar();
        Response.Write(decOrderTotal.ToString() + "<p/>");
```

```
    }

    protected void UpdateCustomers(SqlConnection conn)
    {   string strSQL = "UPDATE Customers SET CompanyName = 'NewValue' "
+
              "WHERE CustomerID = 'ALFKI'";
        SqlCommand cmd = new SqlCommand(strSQL, conn); //创建 Command 对象
        int intRecordsAffected = cmd.ExecuteNonQuery();
        if (intRecordsAffected == 1)
            Response.Write("Update succeeded");
        else
        Response.Write("Update failed" + "<p/>");
    }
}
```

6.5.3　DataReader 对象

在数据库访问操作中，可通过 DataReader 对象和 DataSet 对象获取访问结果。DataReader 对象用来从数据库中检索只读、只进的数据流，它一次在内存中只存储一行记录，从而降低了系统开销，而 DataSet 是将数据一次性加载在内存中，读取完毕后即可断开数据库连接。虽然可以利用 DataSet 动态地添加行、列数据，并且可以对数据库进行回传更新操作，但它消耗的内存比较大，如果此时大量用户同时访问数据库，内存会因过度使用出现无法预料的问题。

应用程序可在下列情况下使用 DataReader 对象：不需要缓存数据；要处理的结果集太大，内存中放不下；一旦需要以只进、只读方式快速访问数据。注意：不能用 DataReader 修改数据库中的记录，它是采用向前的、只读的方式读取数据库。

常用的 DataReader 对象的属性如表 6-47 所示。

表 6-47　DataReader 对象的常用属性

属　性	说　明
FieldCount	表示由 DataReader 得到的一行数据中的字段数
HasRows	表示 DataReader 是否包含数据
IsClosed	表示 DataReader 对象是否关闭

DataReader 对象使用指针的方式来管理所连接的结果集，常用方法如表 6-48 所示。

表 6-48　DataReader 对象的常用方法

方　法	说　明

Close	Close 方法不带参数，无返回值，用来关闭 DataReader 对象。由于 DataReader 在执行 SQL 命令时一直要保持同数据库的连接，所以在 DataReader 对象开启的状态下，该对象所对应的 Connection 连接对象不能用来执行其他的操作。所以，在使用完 DataReader 对象后，要使用 Close 方法关闭该对象，否则既影响数据库连接的效率，也会阻止其他对象使用 Connection 连接对象来访问数据库

续表

方　法	说　明
bool Read()	让记录指针指向本结果集中的下一条记录，返回值是 true 或 false。当 Command 的 ExecuteReader 方法返回 DataReader 对象后，须用 Read 方法来获得第一条记录；当读好一条记录想获得下一条记录时，可用 Read 方法。如果当前记录已经是最后一条，调用 Read 方法将返回 false。也就是说，只要该方法返回 true，则可以访问当前记录所包含的字段
bool NextResult()	该方法会让记录指针指向下一个结果集。当调用该方法获得下一个结果集后，依然要用 Read 方法来开始访问该结果集
Object GetValue(int i)	该方法根据传入的列的索引值，返回当前记录行里指定列的值。由于事先无法预知返回列的数据类型，所以该方法使用 Object 类型来接收返回数据
int GetValues (Object[] values)	把当前记录行里所有的数据保存到一个数组里并返回。可以使用 FieldCount 属性来获知记录里字段的总数，据此定义接收返回值的数组长度
GetString、GetChar、GetInt32	获得指定字段的值，这些方法都带有一个表示列索引的参数，返回均是 Object 类型。用户可以根据字段的类型，通过输入列索引，分别调用上述方法，获得指定列的值。例如，在数据库里，id 的列索引是 0，通过 string id = GetString(0);代码可以获得 id 的值
GetDataTypeName(int i)	通过输入列索引，GetDataTypeName 方法返回列的数据类型
GetName(int i)	通过输入列索引，GetName 方法获得该列的名称
IsDBNull(int i)	该方法用来判断指定索引号的列的值是否为空，返回 True 或 False

【例 6.29】 通过一个用户登录的例子演示创建 DataReader 对象的使用。读取用户信息表，通过 DataReader 逐条将表中的记录与用户填写的登录信息（用户名和密码）相比较，用户合法则提示登录成功，否则提示失败。

```
   protected void Btn_confirm_Click(object sender, EventArgs e) //按钮单击事
件
       {    SqlConnection conn = new SqlConnection();  //创建连接对象
           con.ConnectionString          =          "user          id=sa;data
source=127.0.0.1;integrated                          security=false;initial
catalog=BookShop;pwd=123";//设置连接字符串
           conn.Open();
           SqlCommand cmd = new SqlCommand();
           cmd.Connection = conn;
           cmd.CommandText = "select * from userInfo";
           SqlDataReader myreader = cmd.ExecuteReader();
           string uName = this.txt_name.Text;  //得到用户输入的用户名
           string uPwd = this.txt_pwd.Text;   //得到用户输入的密码
```

```
            bool isTrue = false;
            while (myreader.Read())
            {  if (myreader["Name"].ToString() == uName &&
                   myreader["Password"].ToString() == uPwd)
               {
                   isTrue = true;
               }
            }
```

```
            if (isTrue == true)
              Response.Write("<script language=javascript>alert('登录成功!')
              </script>");
            else
              Response.Write("<script  language=javascript>alert('用户名不存在或
密码不正确!')</script>");
    conn.Close();}
```

6.5.4　DataSet 对象与 DataAdapter 对象

DataSet 对象是 ADO.NET 中最核心的成员之一，它是专门用来处理从数据存储中读出的数据，并以离线方式存于本地内存中。不管数据源是什么类型，DataSet 都使用相同的方式来操作从数据源中取得的数据，也就是说，DataSet 提供了一致的关系编程模型。应用程序主要在下列情况下需要使用 DataSet：在结果的多个离散表之间进行导航；操作来自多个数据源（例如，来自多个数据库、一个 XML 文件和一个电子表格的混合数据）的数据；在各层之间交换数据或使用 XML Web 服务；重用同样的记录行，以便通过缓存获得性能改善（例如排序、搜索或筛选数据）。

利用 DataSet 对象，用户可以先完成数据连接和通过数据适配器 DataAdapter 对象填充 DataSet 对象，然后客户端再通过读取 DataSet 来获得需要的数据，同样更新数据库中数据，也是首先更新 DataSet，然后再通过 DataSet 来更新数据库中对应的数据。DataSet 主要有三个特性：

（1）独立性。DataSet 独立于各种数据源。微软公司在推出 DataSet 时就考虑到各种数据源的多样性、复杂性。在.NET 中，无论什么类型数据源，它都会提供一致的关系编程　　模型。

（2）离线（断开）和连接。DataSet 既可以以离线方式，又可以以实时连接来操作数据库中的数据。这一点有点像 ADO 中的 RecordSet。

（3）DataSet 对象是一个可以用 XML 形式表示的数据视图，是一种数据关系视图。

DataAdapter 对象主要用来承接 Connection 和 DataSet 对象。DataSet 对象只关心访问操作数据，而不关心自身包含的数据信息来自哪个 Connection 连接到的数据源，而 Connection 对象只负责数据库连接而不关心结果集的表示。所以，在 ASP.NET 的架构中使用 DataAdapter

对象来连接 Connection 和 DataSet 对象。另外，DataAdapter 对象能根据数据库里的表的字段结构，动态地构造 DataSet 对象的数据结构。

DataAdapter 对象的工作步骤一般有两种，一种是通过 Command 对象执行 SQL 语句，将获得的结果集填充到 DataSet 对象中；另一种是将 DataSet 里更新数据的结果返回到数据库中。使用 DataAdapter 对象，可以读取、添加、更新和删除数据源中的记录。对于每种操作的执行方式，适配器支持以下 4 个属性，类型都是 Command，分别用来管理数据操作的“增”、“删”、“改”、“查”动作。常见的 DataAdapter 对象的属性如表 6-49 所示。

表 6-49　DataAdapter 对象常用属性

属　性	说　明
AcceptChangesDuringFill	获取或设置将 DataRow 对象置于 DataTable 对象时，DataRow 对象是否调用 AcceptChanges 方法，默认为 True
ContinueUpdateOnError	获取或设置执行 Update 方法更新数据库源时，若发生错误是否继续更新，默认为 False
DeleteCommand	获取或设置用来从数据源删除数据行的 SQL 命令，属性值必须为 Command 对象，并且此属性只有调用 Update 方法且从数据源删除数据行时使用，其主要用途是告知 DataAdapter 对象如何从数据源删除数据行
InsertCommand	获取或设置将数据行插入到数据源的 SQL 命令，属性值为 Command 对象
SelectCommand	获取或设置用来从数据源选取数据行的 SQL 命令，属性值为 Command 对象，使用原则与 DeleteCommand 属性一样
UpdateCommand	获取或设置用来更新数据源数据行的 SQL 命令，属性值为 Command 对象

DataAdapter 对象常用的方法如表 6-50 所示。

表 6-50　DataAdapter 对象常用的方法

方　法	说　明
Fill(DataSet dataset,string srcTable)	根据 dataTable 名填充 DataSet；DataSet：需要更新的 DataSet，srcTable：填充 DataSet 的 dataTable 名
Update(DataSet dataSet)	当程序调用 Update 方法时，DataAdapter 将检查参数 DataSet 每一行的 RowState 属性，根据 RowState 属性来检查 DataSet 里的每行是否改变和改变的类型，并依次执行所需的 INSERT、UPDATE 或 DELETE 语句，将改变提交到数据库中。这个方法返回影响 DataSet 的行数。更准确地说，Update 方法会将更改解析回数据源，但自上次填充 DataSet 以来，其他客户端可能已修改了数据源中的数据。若要使用当前数据刷新 DataSet，应使用 DataAdapter 和 Fill 方法

DataAdapter 对象常用的事件如表 6-51 所示。

表 6-51　DataAdapter 对象常用的事件

事　件	说　明
FillError	当执行 DataAdapter 对象的 Fill 方法发生错误时会触发此事件
RowUpdated	当调用 Update 方法并执行完 SQL 命令时会触发此事件
RowUpdating	当调用 Update 方法且在开始执行 SQL 命令之前会触发此事件

DataSet 的使用方法一般有三种：

（1）把数据库中的数据通过 DataAdapter 对象填充 DataSet。

（2）通过 DataAdapter 对象操作 DataSet 实现更新数据库。

（3）把 XML 数据流或文本加载到 DataSet。

DataSet 对象的常用属性如表 6-52 所示。DataSet 对象的常用方法如表 6-53 所示。

表 6-52　DataSet 对象的常用属性

属　　性	描　　述
CaseSentive	指示 DataTable 中的字符串进行比较时是否区分大小写
DataSetName	返回该 DataSet 的名称

续表

属　　性	描　　述
DefaultViewManager	返回一个 DataViewManager，后者包括 DataSet 中的数据组成的定制视图
EnforceConstraints	指出更新数据时是否遵守约束规则
ExtendedProperties	一个包含自定义用户信息的 PropertyCollection 对象
HasErrors	指出该 DataSet 中的记录是否有错误
Locale	用于比较字符串的区域信息，返回一个 CultureInfo 对象
Relations	一个 DataRelationCollection 对象，表示 DataSet 的表之间的所有关系
Tables	一个代表 DataSet 中所有表的 DataTableCollection 对象
XML	DataSet 中数据的 XML 格式
XMLSchema	关于 DataSet 数据的 XML 图表

表 6-53　DataSet 对象的常用方法

方　　法	描　　述
AcceptChange	提交对 DataSet 所做的所有修改
BeginInit	在运行阶段开始初始化 DataSet
Clear	删除 DataSet 中各个表中的所有记录
Clone	生成与当前 DataSet 相同且不包含数据的 DataSet
Copy	生成与当前 DataSet 相同且包含数据的 DataSet
EndInit	在运行阶段结束 DataSet 的初始化
GetChanges	生成一个包含修改过的数据的 DataSet
HasChange	返回当前 DataSet 中数据是否已修改
InferXMLSchema	使用 XML 数据源创建数据结构
Merge	合并数据集与 DataSet 指定的数据集
ReadXMLSchema	依据 XML 图表创建数据结构
RejectChange	撤销对 DataSet 所做的修改
ResetRelations	将 Relations 属性重置为默认值
ResetTables	将 Tables 属性重置为默认值
WriteXML	将 DataSet 的内容写成 XML 格式
WriteXMLSchema	将 DataSet 的数据结构写成 XML 模式

DataSet 对象和 DataAdapter 对象的区别和联系表现在：

（1）DataAdapter 对象是一种用来充当 DataSet 与实际数据源之间桥梁的对象，ADO.NET

通过该对象建立并初始化 DataTable（DataSet 中的数据表），从而与 DataSet 结合在一起。DataSet 对象是一种无连接的对象，它与数据源无关。而 DataAdapter 则正好充当 DataSet 与实际数据源之间的桥梁，可以用来向 DataSet 对象填充数据，并将对 DataSet 中的数据的修改更新到实际的数据库中。DataAdapter 对象在数据库操作中与 DataSet 配合使用，可以执行新增、查询、修改和删除等多种操作。

（2）DataSet 虽然拥有类似于数据库的结构，但并不等同于数据库。首先，DataSet 不仅可以存储来自于数据库中的数据，而且还可以存储其他类型的数据，如 XML 格式文档；其次 DataSet 与数据库之间没有直接的联系，操作 DataSet 并不等同于数据库中的数据也会发生改变。在 DataSet 上执行的 UPDATE（更新）或 Delete（删除）等操作，影响的仅仅是 DataSet 中存储的数据，而不会在数据库中执行相同的操作。DataSet 中有一个非常重要的对象 DataTable。它实际上是 DataSet 中的数据表，包含了 DataSet 中的所有数据。在 DataTable 中，不仅仅是列，它也包含了表的关系、关键字及其约束等信息。一个 DataTable 表是数据行（DataRow）和列（DataColumn）的集合。DataSet 对象的模型结构如图 6-18 所示。

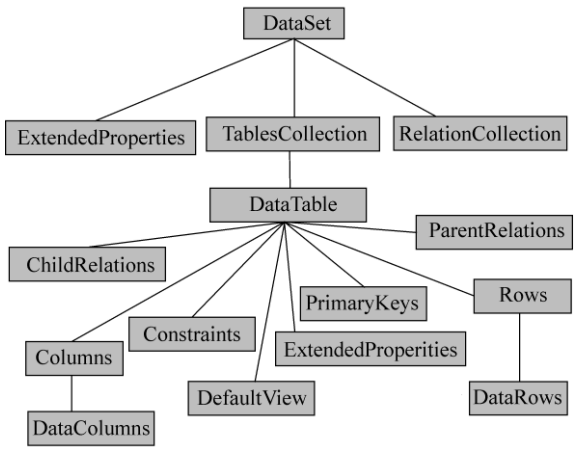


图 6-18　DataSet 对象的模型结构

【例 6.30】DataSet 对象与 DataAdapter 对象的使用。用 DataAdapter 实现对数据集 DataSet 的填充，最后将数据集作为 GridView 的数据源来显示数据信息。

```
    protected void Page_Load(object sender, EventArgs e)
        {  if (!this.IsPostBack)
            {
                SqlConnection con = new SqlConnection();
                conn.ConnectionString  =  "user  id=sa;data  source=127.0.0.1;
integrated security=false;initial catalog=BookShop;pwd=123";//设置连接字符串
                conn.Open();
                SqlDataAdapter  myda  =  new  SqlDataAdapter("select  *  from
BookInfo", conn);
                DataSet myset = new DataSet();
                myda.Fill(myset);
```

```
            this.GridView1.DataSource = myset;
            this.GridView1.DataBind();
        }
    }
```

【例 6.31】 独立使用 DataSet。创建一个 DataSet 对象，并在其中创建了一个表，同时添加了一条记录，并显示出来。

```
protected void Page_Load(object sender, EventArgs e)
```

```
    {
        DataSet ds = new DataSet();
        DataTable dt = new DataTable();
        dt.Columns.Add("Id", System.Type.GetType("System.Int32"));
        dt.Columns.Add("Uname", System.Type.GetType("System.String"));
        dt.Columns.Add("Pwd", System.Type.GetType("System.String"));
        dt.Columns["id"].AutoIncrement = true;
        ds.Tables.Add(dt);
        ds.Tables[0].PrimaryKey = new DataColumn[] { ds.Tables[0].Columns
["Id"] };
        DataRow dr = dt.NewRow();
        dr[0] = "1";
        dr[1] = "zhangsan";
        dr[2] = "123456";
        dt.Rows.Add(dr);
        this.GridView1.DataSource = ds;
        this.GridView1.DataBind();
    }
```

DataView 对象表示 DataSet 对象中数据的定制视图，它代表一个 DataTable 的数据查看方式。在通常情况下，默认的数据查看方式是按照数据库数据表中数据的排列顺序，以表格形式显示。对于用户而言，常常需要利用排序、筛选以及查询等属性来定义不同的数据查看方式，这正是 DataView 对象的价值所在。DataView 对象可以指定筛选条件来选择查看单个数据表或多个数据表中的部分数据，也可以指定与默认查看方式不同的排序方式。DataView 对象的常用属性如表 6-54 所示。DataView 对象的常用属性如表 6-55 所示。

表 6-54　DataView 对象的常用属性

属　　性	描　　述
AllowDelete	设置是否可以删除视图中的记录
AllowEdit	设置是否可以编辑视图
AllowNew	设置是否可以在视图中添加记录
ApplyDefaultSet	设置是否使用默认的排序方式

Item(Index)	返回视图中指定的记录
RowFilter	设置筛选条件，符合条件的记录添加到 DataView 中
Sort	设置或获取一个或多个排序的字段以及排序顺序
Table	从中取得数据的源 DataTable

表 6-55　DataView 对象的常用方法

方　法	描　述
AddNew	在视图中加一条新记录
BeginInit	开始初始化 DataView

续表

方　法	描　述
Delete(index)	删除由 index 所指定的记录
Dispose	释放对象的当前实例
EndInit	结束初始化过程
Find	在 DataView 中查找指定的记录

【例 6.32】 DataView 对象的使用。

```
   protected void Page_Load(object sender, EventArgs e)
       {
          SqlConnection              conn              =              new
SqlConnection("server=.;database=BookShop; uid=sa;pwd=123");
          conn.Open();//建立与数据库的连接
          SqlDataAdapter da = new SqlDataAdapter("select * from userInfo",
conn);
          DataSet ds = new DataSet();
          da.Fill(ds);
          DataView dview1 = new DataView(ds.Tables[0]);  //生成 DataView 对象
          dview1.RowFilter = "Id>5";//设置过滤条件
          DataView dview2 = new DataView(ds.Tables[0]);
          dview2.Sort = "names";//设置排序字段
          this.GridView1.DataSource = dview1;
          this.GridView1.DataBind();
          this.GridView2.DataSource = dview2;
          this.GridView2.DataBind();
          //分别绑定到不同的 GridView 控件上
       }
```

6.5.5　执行存储过程

存储过程是保存起来的可以接受和返回用户提供的参数的 Transact-SQL 语句的集合。在存储过程中可以使用数据存取语句、流程控制语句、错误处理语句等。其主要特点是执行效率高，可以重复使用。在创建存储过程时，SQL Server 会将存储过程编译成一个执行计划。一旦创建一个存储过程，很多需要该过程的应用程序都可以调用此存储过程，减少程序员可能出现的错误。

在 ADO.NET 应用程序中使用存储过程的一般步骤是：①创建存储过程；②建立与相应数据库的连接；③指定存储过程名称；④在程序中说明执行的类型是存储过程；⑤若要执行的存储过程有参数则完成填充 Parameters 集合，否则省略此步；⑥执行存储过程。

在上述过程中的第④步提到了 Parameters 集合，在使用带参数的存储过程时它是经常被用到的。Parameters 集合是一个 Parameter 类型的数组，用于给存储过程传递参数。

Parameter 类是为数据源控件提供一个绑定到应用程序变量、用户标识和选择其他数　据的机制。Parameter 参数除了动态提供传值的作用外，最重要的是它会执行以下工作：　①检查参数的类型；②检查参数的长度；③保证输入的参数在数据库中会被当成纯文字而不是可以执行的 SQL 命令。因此参数进行了以上三项安全性检查，将安全性强化到一定水平之上。

与 System.Data.SqlClient 名称空间相对应的 Parameter 对象是 SqlParameter 对象。其属性如表 6-56 所示。

表 6-56　SqlParameter 对象的属性

属 性 名 称	数 据 类 型	说　　明
DbType	DbType	指定该参数的数据库数据类型
Direction	ParameterDirection	指定该参数的方向：1-输入、2-输出、3-输入/输出，6-参数将包含存储过程的返回值
IsNullable	Boolean	指示该参数是否可以接受 Null
ParameterName	String	指定参数的名称
Precision	Byte	指定参数的精度
Scale	Byte	指定参数的位数
Size	Int32	指定参数的大小
SqlDbType	SqlDbType	指定该参数的 SQL 数据类型
SqlValue	Object	指定使用 SqlType 的参数的值
Value	Object	指定该参数的值

下面通过两个例子分别说明在程序中如何使用带参数的存储过程和不带参数的存储过程，每个例子分别使用 Command 对象和 DataAdapter 对象来实现调用。

【例 6.33】 在应用程序中执行不带参数的存储过程。

```
    /*在 SQL Server 中建立存储过程 p_Search*/
    create proc p_Search
    AS   select names,author,publisher,price from book
```

```
    protected void Page_Load(object sender, EventArgs e)
        {   ExecutebyCmd();    //1.通过 Command 对象调用存储过程;
            ExecutebyAdapter();//2.通过 DataAdapter 对象调用存储过程;
        }
    private void ExecutebyCmd()//通过 Command 对象调用存储过程;
        {
          SqlConnection conn = new SqlConnection("server=.;database=BookShop;
uid=sa;pwd=123");
```

```
          conn.Open();//建立与数据库的连接
        SqlCommand cmd = new SqlCommand("p_Search", conn);//调用存储过程
p_Search
          cmd.CommandType = CommandType.StoredProcedure;//说明执行的类型是存储过
程
          try
          {   cmd.ExecuteNonQuery();//执行存储过程
              Response.Write("<script language=javascript>alert('操作成功！')
</script>");
          }
          catch
```

```
          {    Response.Write("<script language=javascript>alert('操作失败')
</script>");
          }
      }
   private void ExecutebyAdapter()//通过 DataAdapter 对象调用存储过程;
      {           SqlConnection           conn           =           new
SqlConnection("server=.;database=BookShop;uid=sa;pwd=123");
          conn.Open();//建立与数据库的连接
          SqlDataAdapter da = new SqlDataAdapter();
          da.SelectCommand = new SqlCommand("p_Search ", conn);//调用存储过程
p_Search
          da.SelectCommand.CommandType = CommandType.StoredProcedure;//说明
类型是存储过程
          try
          {   da.SelectCommand.ExecuteNonQuery();// 执行存储过程
              Response.Write("<script language=javascript>alert('操作成功！')
</script>");
          }
          catch
          {    Response.Write("<script language=javascript>alert('操作失败')
</script>");
          }
      }
```

【例 6.34】 在应用程序中使用带参数的存储过程。在页面上由用户输入用户名、密码和邮件地址，单击“保存”按钮后调用存储过程将它们保存到数据表中。

```
   /*在 SQL Server 中建立存储过程 insertuserInfo*/
      create proc insertuserInfo
      @name varchar(30),@password varchar(50),@mail varchar(30)
      as insert into userinfo values(@name,@password,@mail)
```

```
   protected void Button1_Click1(object sender, EventArgs e)
```

```
    {   ExecutebyCmd();//通过 Command 对象调用存储过程;
        ExecutebyAdapter();//通过 DataAdapter 对象调用存储过程;
    }
  private void ExecutebyCmd()//通过 Command 对象调用存储过程;
    {  SqlConnection  conn  =  new  SqlConnection("server=.;  database=
BookShop; uid=sa;pwd=123");
        conn.Open();//建立与数据库的连接
        SqlCommand cmd = new SqlCommand("insertuserInfo", conn);//调用存储
过
                                                                  //      程
search
        cmd.CommandType  =  CommandType.StoredProcedure;//说明执行的类型是存储过
程
        //建立参数对象的数组，指定每个参数的类型、长度和值
        SqlParameter[] parameters =
                  {  new SqlParameter("@name", SqlDbType.VarChar,30),
                    new SqlParameter("@password", SqlDbType.VarChar,50),
                    new SqlParameter("@mail", SqlDbType.VarChar,30),
                   };
        parameters[0].Value = this.TextBox1.Text;
        parameters[1].Value = this.TextBox2.Text;
        parameters[2].Value = this.TextBox3.Text;
        //将参数对象传递给 SqlCommand 对象
        foreach (SqlParameter para in parameters)
        {   cmd.Parameters.Add(para);
        }
        try
        {   cmd.ExecuteNonQuery();//执行存储过程
           Response.Write("<script language=javascript>alert('操作成功！')
</script>");
        }
        catch
        {
           Response.Write("<script  language=javascript>alert('操作失败')
</script>");
        }
    }
    private void ExecutebyAdapter()//通过 DataAdapter 对象调用存储过程;
    {           SqlConnection           conn           =           new
SqlConnection("server=.;database=BookShop;uid=sa;pwd=123");
        conn.Open();//建立与数据库的连接
        SqlDataAdapter da = new SqlDataAdapter();
        da.InsertCommand = new SqlCommand("insertuserInfo", conn);//调用存
                                    储
```

```
                                                                    //过程 search
          da.InsertCommand.CommandType = CommandType.StoredProcedure;//说明
                                        类
                                                                    //型是存储过程
          SqlParameter[] parameters =
                    { new SqlParameter("@name", SqlDbType.VarChar,30),
                      new SqlParameter("@password", SqlDbType.VarChar,50),
                      new SqlParameter("@mail", SqlDbType.VarChar,30),
                     };//建立 parameter 数组
          parameters[0].Value = this.TextBox1.Text;
          parameters[1].Value = this.TextBox2.Text;
          parameters[2].Value = this.TextBox3.Text;
          foreach (SqlParameter para in parameters)//传递参数
          {     da.InsertCommand.Parameters.Add(para);
          }
          try
          {   da.SelectCommand.ExecuteNonQuery();// 执行存储过程
             Response.Write("<script language=javascript>alert('操作成功！')
</script>");
```

```
          }
          catch
          {    Response.Write("<script language=javascript>alert('操作失败')
</script>");
          }
       }
```

6.5.6　数据库事务处理

数据库事务处理是把一组数据库操作合并为一个逻辑上的工作单元。在系统中没有出现错误的情况下，开发人员可以使用事务处理来控制并保持事务处理中每一个动作的连续性和完整性。使用这样的方法可能导致向两个极端情况发展：要么在事务处理中的所有操作都得到执行，要么没有任何操作得到执行。这样的方法对于实时应用程序来说非常必要。事务处理需要一个数据库连接以及一个事务处理对象。

在 ADO.NET 中，可以使用 Connection 和 Transaction 对象启动、提交和回滚事务。若要执行事务，可执行下列操作：调用 Connection 对象的 BeginTransaction 方法来标记事务的开始。BeginTransaction 方法返回对 Transaction 的引用。该引用将分配给登记在事务中的 Command 对象。将 Transaction 对象分配给要执行的 Command 的 Transaction 属性。如果通过活动的 Transaction 对象对 Connection 执行 Command，但该 Transaction 对象尚未分配给 Command 的 Transaction 属性，则将引发异常。

与 System.Data.SqlClient 名称空间相对应的 Transaction 对象是 SqlTransaction 对象。它包括了两个属性。

- Connection：指示同事务处理相关联的 SqlConnection 对象；
- IsolationLevel：定义事务处理的锁定记录的级别。属性 IsolationLevel 是包括如　　表 6-57 所示成员的枚举对象。

表 6-57　IsolationLevel 枚举对象的成员

成 员 名 称	说　　明
Chaos	从高度独立的事务处理中出现的 pending changes 不能被覆盖
ReadCommitted	当数据需要被非恶意读取时，采用共享锁定（shared locks），但数据仍然可以在事务处理结束时被更新，这造成了非重复性的数据读取或 phantom data 的产生
ReadUncommitted	恶意读取数据是可能发生的，这表示没有使用共享锁定（shared locks），并且没有实现独占锁定（exclusive locks）
RepeatableRead	锁定查询中所用到的所有数据，由此避免其他用户对数据进行更新。在 phantom rows 仍然可用的状态下，这可避免非重复性的数据读取
Serializable	在 DataSet 中进行范围锁定，由此防止其他用户在事务处理结束之前更新数据或在数据库中插入行

SqlTransaction 对象包括三个主要方法。

（1）Commit：提交数据库事务；

（2）Rollback：从未决状态（pending state）回滚（roll back）事务处理。事务处理一旦被提交成功后即不能执行此操作；

（3）Save：在事务处理中通过指定 savepoint 保存点名称创建保存点，以便对事务处理的一部分进行回滚。

下面结合例子说明在 ADO.NET 中是如何完成事务处理的。

【例 6.35】 事务处理的例子。

```
    protected void Button1_Click(object sender, EventArgs e)
        {SqlConnection conn = new SqlConnection("server=.;database=BookShop;
uid=sa;pwd=123");
          conn.Open();
          //启用事务
          SqlTransaction tran = conn.BeginTransaction();
          SqlCommand cmd = new SqlCommand();
          cmd.Connection = conn;
          cmd.Transaction = tran;
          try
          {   cmd.CommandText = "update userinfo set mail='zhangsan@126.com'
where id=2";
              cmd.ExecuteNonQuery();
              cmd.CommandText = "update userinfo set mail='zhangsan@126.com'
where id=4";
              cmd.ExecuteNonQuery();
              tran.Commit();  //提交事务
        Response.Write("<script  language=javascript>alert('事务提交成功！')
```

```
</script>");
          }
          catch (Exception ex)
          {   tran.Rollback(); //回滚事务
              Response.Write("<script language=javascript>
                  alert('失败！'"+ex.ToString()+")</script>");
          }
      }
```

6.5.7　跨数据库访问

这里只给出针对 SQL Server 2000 的处理方法。进行跨库查询的时候会遇到两种情况：①被操作的多个数据库位于同一台物理机器上；②被操作的多个数据库位于不同的物理机器上。下面分别对这两种情况进行说明。

1．被操作的多个数据库位于同一台物理机器上

针对这种情况，只需要在构造的 sql 语句中的表名前加上“数据库.dbo”就可以，例如“select * from DBName1.dbo.Table1”。在程序中建立的与数据库的连接不需要改变。下面给出一个在应用程序中实现对多个库进行操作的示例代码。

【例 6.36】 同一台物理机器上的跨数据库操作。其中 ykdb 和 rjht 为不同的数据库名称。

```
    using System;
    using System.Data;
    using System.Data.SqlClient;
    using System.Web;
    using System.Web.UI;

    public partial class Default2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {   SqlConnection conn = new SqlConnection("server=.;database=ykdb;
uid=ykdb;pwd=123");
            conn.Open();//建立与数据库的连接
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "select a.user_name,b.realname from ykdb.dbo.
user_name a,rjht.dbo.userinfo b";
            cmd.Connection = conn;
            SqlDataReader reader = cmd.ExecuteReader();
            try
            {
                while (reader.Read()) //循环读取一条记录，从首记录到末记录
                {   Response.Write(reader[0].ToString() + "," + reader[1].
ToString() + "<br/>");
                }
            }
```

```
            finally
            {  reader.Close();
            }
        }
    }
```

2．被操作的数据库位于不同的物理机器上

针对这种情况，需要将其他数据库所在的机器连接到当前主操作数据库服务器中来。

具体操作过程为：首先进入主操作数据库服务器的企业管理器，选中左侧树型图中的“安全（Security）”，右击“连接服务器（Linked Severs）”，选中“新建连接服务器（New Linked Server…）”后弹出新建链接服务器窗口，选择安全性选项卡，选中“用此安全上下文进行（Be made using this security context）”选项，并填写登录名和密码。再选择常规选项卡，　　在服务器类型中选中 sql server，将要连接的服务器名填入到链接服务器，单击“确定”按钮，这样就完成了与另外一台物理机器的链接。通过新建连接服务器可链接多台数据库服务器到主操作数据库服务器上来。接下来要做的工作就是在程序中通过构造 sql 语句实现多库操作功能。假设当前服务器名为 MyCurrentServer，数据库为 MyDb，其中的一个表名为 MyCurrentTable，链接的数据库服务器名为 MyLinkedServer，数据库名为 MyLinkedDB，其中的一个表名为 MyLinkedTable，则你的 SQL 语句可以这样写：

```
    select   A.*,B.* from   MyCurrentTable  A,MyLinkedServer.MyLinkedDB.dbo.
MyLinkedTable  B
```

下面给出一个示例进一步加以说明。

【例 6.37】 不同物理机器上的跨库操作。另一台服务器名为 tyd，此处可为 IP 地址，数据库名为 news，被操作的表为 newsType。

```
    protected void Page_Load(object sender, EventArgs e)
        { SqlConnection conn = new SqlConnection("server=.;database=BookShop;
uid=sa;pwd=123");
          conn.Open();//建立与数据库的连接
          SqlCommand cmd = new SqlCommand();
          cmd.CommandText = "Select A.*,B.* from book A,tyd.news.dbo.newsType
B";
          cmd.Connection = conn;
           …
        }
```

6.5.8　数据绑定技术

数据绑定技术最通常的应用是把 Web 控件中用于显示的属性跟数据源绑到一起，从而在 Web 页面上显示数据。此外，也可以使用数据绑定技术设置 Web 控件的其他属性。因此可以说，ASP.NET 的数据绑定技术非常灵活，数据源可以是数据库中的数据，也可以是 XML

文档、其他控件的信息，甚至可以是其他进程的信息或其他进程的运行结果。

ASP.NET 控件既可以绑定到简单的数据源，如变量、属性、集合以及表达式等，也可以绑定到复杂的数据源，如数据集、数据视图等。

ASP.NET 引入了新的数据绑定方法，使用该语法可以轻松地将 Web 控件的属性绑定到数据源。语法如下：

```
<%#DataSource %>
```

其中 DataSource 表示各种数据源，如变量、属性、列表、表达式以及数据集等。下面给出两个数据绑定的例子。

【例 6.38】 绑定到简单数据源。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="test.aspx.cs"
Inherits="test" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server"> <title>无标题页</title>
<!-- 此段代码可以放在 test.aspx.cs 中,在这里这种写法没有使用代码分离技术 -->
   <script language="c#" runat="server">
      int num = 65108995;
      string name = "张强";
      string email = "zhangqiang@126.com";
      void Page_Load(Object sender, EventArgs e)
      {
         Page.DataBind();
      }
     </script>
</head>
<body>
   <form id="form1" runat="server">
   <b>姓名:<%#name%></b><br />
   <b>邮箱:<%#email %></b><br />
   <b>电话:<%#num.ToString() %></b>
   </form>
</body>
</html>
```

其运行效果如图 6-19 所示。

图 6-19　简单数据源绑定

【例 6.39】 绑定到复杂数据源。在页面上放一个 ListBox 控件，下面代码实现了将数据表中的数据绑定到该 ListBox 控件。

```
    using System;
    using System.Data;
    using System.Web;
    using System.Web.UI;
    using System.Data.SqlClient;
    public partial class test : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {  SqlConnection con = new SqlConnection("server=.;database=BookShop;
uid=sa;pwd=123");
           con.Open();
           SqlCommand cmd = new SqlCommand("select name from userinfo",con);
           ListBox1.DataSource = cmd.ExecuteReader();
           ListBox1.DataTextField = "name";
```

```
           ListBox1.DataValueField = "name";
           ListBox1.DataBind();
           con.Close();
        }
    }
```

6.5.9　LINQ 查询技术

LINQ（Language Integrated Query）提供了一种跨各种数据源和数据格式使用数据的一致模型。在 LINQ 查询中，始终使用对象而非针对某种具体数据源的操作命令。可以使用相同的基本编码模式来查询和转换 XML 文档、SQL 数据库、ADO.NET 数据集、.Net 集合中的数据及对其有 LINQ 提供程序可用的任何其他格式的数据。

LINQ 是一种全新的数据查询方式，LINQ 的查询有两种语法，一种是方法语法，即传统的 C#类和方法的语法进行查询；另外一种语法是查询语法，这是为 LINQ 而引入的新的 C＃语法。这两类语法形式不同，但是功能相同。实际上 C#在遇到 LINQ 的查询语法时，会翻译成相应的方法语法再执行。

1．创建查询数据源

LINQ 用于从各种数据源中查询数据。为了演示 LINQ 的语法和功能，需要创建一部分数据，以这些数据为基础进行查询。本节将使用一组随机生成的商品销售数据为数据源演示 LINQ 的使用。与此相关的有 3 个类：商品类别 Category、商品信息 Product 和销售记录 ProductSale。为了随机生成数据，3 个类中分别包含一个随机填充数据的方法。Category 类的代码如下：

```
public class Catagory
{
    public string id { get; set; }
    public string name { set; get; }
//<param name = "count">要生成商品类别数量</param>
    public static List<Catagory> randomCatagories(int count)
   {
     Random r = new Random(DateTime.Now.Millisecond);
     List<Catagory> catagories = new List<Catagory>();
     for (int i = 0; i < count; i++)
     {
         Catagory category = new Catagory();
         category.id = "C" + r.Next(1000);
         category.name = "C" + i.ToString();
         catagories.Add(category);
      }
      return catagories;
   }
}
```

Product 类的代码如下：

```
public class Product
{
    public string id { get; set; }
    public string name { set; get; }
    public int storage { set; get; }
    public double price { set; get; }
    public Catagory category { get; set; }
//<param name = "count">要生成商品类别数量</param>
   //<returns>所生成的商品列表</returns>
    public static List< Product> randomProducts(int count)
```

```
    {
        Random r = new Random(DateTime.Now.Millisecond);
        List<Product> products = new List<Product>();
        //随机生成类别,设类别数量为商品数量的1/10.但至少有3种,最多有20种。
        int categoryCount = Math.Min(20, Math.Max(3, count / 10));
        Catagory[] categories =
Catagory.randomCatagories(categoryCount).ToArray  ();
        for (int i = 0; i < count; i++)
        {
            Product p = new Product();
            p.id = "P" + r.Next(1000).ToString ("0000");
             p.name = "Product" + i.ToString();
             p.storage = r.Next(10, 1000);
             p.price = 1 + r.NextDouble() * 300;
             p.category = categories[r.Next(categoryCount)];
             products.Add(p);
        }
        return products;
     }
 }
```

ProductSale 类的代码如下：

```
public class ProductSale
{
    public Product product { get; set; }
    public DateTime  date { set; get; }
    public int quantity { set; get; }
//<param name = "count">要生成的销售记录的数量</param>
   //<returns>所生成销售数据列表</returns>
   public static List<ProductSale> randomSales(int count)
    {
        //随机生成商品列表,设商品种类为销售记录的1/5。
        int productCount = count / 5 + 1;
        Product[] products = Product.randomProducts(productCount).ToArray
();
        Random r = new Random(DateTime.Now.Millisecond);
        List<ProductSale> sales = new List<ProductSale>();
        for (int i = 0; i < count; i++)
```

```
        {
            ProductSale sale = new ProductSale();
            sale.product = products[r.Next(productCount)];
            sale.date = DateTime.Now.AddDays(r.Next(-30, 30));
            sale.quantity = r.Next(100);
            sales.Add(sale);
        }
        return sales;
    }
}
```

2．投影

在数据查询中，投影是指从一个数据源中查询某些字段或属性，例如 SQL 语句中的“select1 列名 1，列名 2，…，列名 N from 表名”就是一个投影操作。在 LINQ 中，IEnumberable<T>类的 Select 扩展方法可以实现投影操作，方法声明如下：

```
//摘要:将序列中的每个元素投影到新表中
//参数:
//source: 一个值序列，要对该序列调用转换函数
//selector: 应用于每个元素的转换函数
//类型参数:
//TSource: source中的元素的类型
//TResult: selector返回的值的类型
//返回结果: 一个System.Collections.Generic.IEnumberable<T>，其元素为对source
的每个元素调用转换函数的结果
//异常: System.ArgumentNullException:source或selector为null
public static IEnumberable<TResult> Select<TSource,TResult>(this
IEnumberable<TSource> source,Func<TSource,TResult> selector);
```

Select 方法用于从集合中查询数据，所查询到的数据并不一定与集合中原有数据类型相同，而是可以基于原有数据创建任意的数据类型。例如可以从一个学生信息集合中查询所有的学号。Select 方法参数中的 Func<TSource,TResult>实现了这个类型转换的功能，通常为这个参数传递一个 Lambda 表达式。

【例 6.40】 Select 查询。本例演示如何使用 IEnumberable<T>类的 Select 扩展方法查询商品信息。

```
public static void selectDemo()
{
    List<Product> products = Product.randomProducts(5);
    var query = products.Select(p => p);
```

```
    foreach (var item in query)
    {
Console.WriteLine("Id={0},Name={1},Price={2},storage{3},category={4}",
item.id, item.name, item.price, item.storage, item.category.name);
    }
//查询商品的一部分信息，这些信息构成一个匿名类型
    var query2 = products.Select(p => new { Id = p.id, Name = p.name,
Category = p.category.name });
    foreach (var item in query2)    {
Console.WriteLine("Id={0},Name={1},category={2}", item.Id, item.Name,
item.Category);
     }
}
```

如前所述，LINQ 有两种语法：方法语法和查询语法。LINQ 的查询语法如下：

```
from 变量名 in 数据源   select 表达式
```

【例 6.41】所使用的语法为扩展方法语法，所对应的等价查询语法如下：

```
public static void selectDemo()
{
   List<Product> products = Product.randomProducts(5);
   var query = from p in products
               select p;
   foreach (var item in query)
   {

Console.WriteLine("Id={0},Name={1},Price={2},storage{3},category={4}",
item.id,item.name, item.price, item.storage, item.category.name);
   }
   var query2 = from p in products
                select new { Id = p.id, Name = p.name, Category =
p.category.name };
   foreach (var item in query2)
   {
   Console.WriteLine("Id={0},Name={1},category={2}",item.Id,item.Name,
item.Category);
   }
}
```

3．选择

数据查询中的选择是指从数据源中查找符合条件的数据，过滤掉不符合条件的数据。SQL 语言中 Select 语句后面的 Where 条件所起的作用就是选择。在 LINQ 中 IEnumberable<T>类的 Where 扩展方法可以实现数据过滤功能。与 Where 扩展方法对应的查询语法为：

```
from 变量名   in 数据源     where 条件    select 表达式
```

【例 6.42】 Where 过滤数据。本例演示使用 Where 方法查询价格大于 180 元的商品信息。

```
public static void whereDemo()
{
    List<Product> products = Product.randomProducts(10);
    var query = from p in products
                where p.price > 180
                select p;
    foreach (var item in query)
    {
        Console.WriteLine("Id={0},Name={1},Price={2}", item.id,
item.name, item.price);
     }
     var query2 = from p in products
                  where p.price > 180
                  select new { Id = p.id, Name = p.name, Category =
p.category.name };
     foreach (var item in query2)
    {
    Console.WriteLine("Id={0},Name={1},category={2}",item.Id,item.Name,
item.Category);
     }
}
```

上述代码运行结果如图 6-20 所示。

```
Id=P0115,Name=Product3,Price=292.606874014999
Id=P0650,Name=Product5,Price=251.111771025747
Id=P0236,Name=Product9,Price=299.148720617475
Id=P0115,Name=Product3,category=C2
Id=P0650,Name=Product5,category=C2
Id=P0236,Name=Product9,category=C1
```

图 6-20 Where 语句的使用

4．排序

为了实现数据排序，IEnumberable<T>类包含 4 个扩展方法，分别为 OrderBy、OrderByDescending、ThenBy、ThenByDescending。其中 OrderBy 方法的功能是将数据源按照一个表达式升序排序。ThenBy 方法的作用是对一个已经按照某个表达式排序的集合再按照另一个表达式进行排序，ThenByDescending 方法的功能与之相似，只是按照降序排序。与 OrderBy 等 4 个排序方法相对应的查询语法如下：

```
From 变量名 in 数据源
Where 条件
Orderby 表达式1 [descending] ，表达式2，…，表达式n
Select 表达式
```

【例 6.43】数据排序。

本例演示按照单个关键字和按照多个关键字对商品进行排序。

```
public static void sortDemo() {
List<Product> products = Product.randomProducts(10);
var query = from p in products
             orderby p.price
             where p.price > 180
             select p;
   foreach (var item in query)
   {
    Console.WriteLine("Id={0},Name={1},Price={2}", item.id, item.name,
item.price);
   }
    var query2 = from p in products
              where p.price < 100
              orderby p.category .id,p.price  descending
              select p;
   foreach (var item in query2)
   {
    Console.WriteLine("Id={0},Name={1},Price={2}", item.id, item.name,
item.price);
}
}
```

5 数据分组

进行数据查询时，很多时候需要进行数据分组，IEnumberable<T>类的 GroupBy 方法实现了数据的分组。

【例 6.44】分组查询。

本例演示分组查找功能，将所有商品按照类别 ID 进行分组，并输出各组的键值（类别 ID）和各组中的商品列表。

```
public static void groupDemo()
{
   List<Product> products = Product.randomProducts(6);
   var query = from p in products
               group p by p.category .id
               into categoryGroup
               select new { categoryid = categoryGroup.Key , products =
categoryGroup};
   int i = 1;
  foreach (var group in query)
  {
   Console.WriteLine("=========第{0}组（类别id：{1}）
========",i++,group .categoryid );
   foreach (var item in group.products ) {
Console.WriteLine("Id={0},Name={1},Price={2}",item .id,item .name ,item
.price );
}
  }
}
```

上述代码输出结果如图 6-21 所示。

```
=========第1组（类别id：C882）========
Id=P0222,Name=Product0,Price=193.913795771503
Id=P0260,Name=Product5,Price=139.744492148396
=========第2组（类别id：C740）========
Id=P0863,Name=Product1,Price=129.954598553923
Id=P0962,Name=Product3,Price=227.930957160392
=========第3组（类别id：C790）========
Id=P0054,Name=Product2,Price=200.08264698418
Id=P0070,Name=Product4,Price=126.854281348108
```

图 6-21 分组查询

6.6 用 VS 2010 创建和访问 Web 服务实例

一个 Web 服务（Web Service）就是一个应用 Web 协议的可编程的应用程序逻辑。从表

面上看，Web 服务就是一个应用程序，它向外界提供了一个可以通过 Web 进行调用的 API，也就是说可以用编程的方法通过 Web 来调用这个程序。Web 服务平台是一套标准，它定义了应用程序如何在 Web 上实现互操作。可以用任何语言在任何平台上写 Web 服务，只要通过 Web 服务标准对这些服务进行查询和访问就行了。可将 Web 服务看做是 Web 上的组件编程。要实现这样的目标，Web 服务使用了两种技术——XML 技术和 SOAP 协议。

- XML 技术：XML 是在 Web 上传送结构化数据的有效方式，Web 服务要以一种可靠的自动的方式操作数据，XML 可以使 Web 服务方便地处理数据，十分理想地实现数据与表示的分离。
- SOAP 协议（Simple Object Access Protocol）：SOAP 是服务使用者向 Web 服务发送请求并接收应答的协议。SOAP 是基于 XML 和 XSD 的，XML 是 SOAP 的数据编码方式。

对 Web 服务进行简单介绍后，下面我们开始学习在 VS 2010 中如何创建 Web 服务。

【例 6.45】 单独创建一个 Web 服务，放在网上被其他应用系统共享使用。

（1）首先进入 VS 2010，选择新建网站。在项目类型中选择 Visual C# 项目（选择.NET Framework 4.0 以下框架），在模板中选择 ASP.NET Web 服务，在项目位置中选择文件系统，存放位置为 D:\Webservice。单击“确定”按钮后 VS 2010 就自动创建了.NET Web 服务框架。在“解决方案资源管理器”中生成了两个文件 Service.asmx 和 App_Code 目录下的 Service.cs。

（2）可以看到在 Service.cs 代码视图里已经存在一个名为 Hello world 的 Web 服务，这是 VS 2010 提供的一个示例 Web 服务方法，返回字符串“Hello World”。在每一个 Web 服务的方法前加上“WebMethod”就可以按自己的要求建立多个 Web 服务方法了。我们在上述基础上建立一个名为 MAX 和 getmyTable 的 Web 服务方法，其完整代码如下：

```
    using System;
    using System.Web;
    using System.Web.Services;
    using System.Web.Services.Protocols;
    using System.Data;
    using System.Data.SqlClient;   //由于 getmyTable 方法连接数据库，必须加上此名称空
间

    [WebService(Namespace = "http://tempuri.org/")]
```

```
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class Service : System.Web.Services.WebService
    {
        public Service()
        {   //如果使用设计的组件，请取消注释以下行
            //InitializeComponent();
        }

        [WebMethod]
```

```
        public string HelloWorld()
        {  return "Hello World";
        }
        [WebMethod(Description = "在输入的三个数中返回最大的一个")]
        public int Max(int a, int b, int c)
        {   int max;
            if (a > b)
                max = a;
            else
                max = b;
            if (c > max)
                return c;
            else
                return max;
        }

        [WebMethod(Description  =  "从数据表中读取数据，以表格方式显示出来。",
EnableSession = false)]
        public string getmyTable(string ConnectionString, string SelectSQL)
        {   string rsString;
            SqlConnection conn = new SqlConnection(ConnectionString);
            SqlDataAdapter da = new SqlDataAdapter(SelectSQL, conn);
            DataSet ds = new DataSet();
            da.Fill(ds, "myTable");
            DataTable tbl = ds.Tables["myTable"];
            rsString  =  @"<table  border='0'  bgcolor='blue'  cellpadding='1'
cellspacing='1'><tr bgcolor='white'>";
            for (int i = 0; i <= ds.Tables["myTable"].Columns.Count - 1; i++)
            { rsString += "<td>" + ds.Tables["myTable"].Columns[i].ColumnName
+ "</td>";
            }
            rsString += "</tr>";
            for (int i = 0; i < tbl.Rows.Count; i++)
            {   rsString += "<tr bgcolor=\"white\">";
                for (int j = 0; j <= ds.Tables["myTable"].Columns.Count - 1;
j++)
                {   rsString += "<td>" + tbl.Rows[i][j] + "</td>";
```

```
                }
                rsString += "</tr>";
            }
            rsString += "</table>";
            return rsString;
        }
    }
```

“WebMethod”中的 Description 是对该 Web 方法的一种描述，用来解释和说明该 Web 方法。此 Web 服务使用 http://tempuri.org/作为默认命名空间。在公开此 XML Web Services 之前，应更改默认命名空间。每个 Web 服务都需要一个唯一的命名空间，以便客户端应用程序能够将它与 Web 上的其他服务区分开。http://tempuri.org/可用于处于开发阶段的 Web 服务，而已发布的 Web 服务应使用更为永久的命名空间。可使用公司的 Internet 域名作为命名空间的一部分，它们不必指向 Web 上的实际资源。

（3）将 Service.asmx 设为起始页，按下 F5 运行此 Web 服务，运行结果如图 6-22 所示。运行结果中显示出了编写的三个 Web 服务方法。单击“服务说明”超链接，显示该 Web 服务的 WSDL 说明。WSDL（Web Services Description Language，Web 服务描述语言）用于描述服务端所提供服务的 XML 格式。WSDL 文件里描述了服务端提供的服务、提供的调用方法以及调用时所要遵循的格式，比如调用参数和返回值的格式，等等。WSDL 很像 COM 编程里的 IDL（Interface Description Language），是服务器与客户端之间的契约，双方必须按契约严格行事才能实现功能。它在创建 Web 服务时自动生成。

图 6-22　Web 方法的测试页面

（4）下面就可以测试所构建的 Web 服务方法的正确与否。单击 Max 后，分别输入 3、9、6，单击“调用”后，在弹出的新窗口中以 XML 文件格式显示最大值结果 9：

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri.org/">9</int>
```

若单击 getmyTable 方法，则出现如图 6-23 所示界面。

输入连接数据库的字符串"server=localhost;database=northwind;uid=sa;pwd=123;"和 SQL 查询语句"select * from employees"后，单击“调用”后如一切正常将以 XML 文件格式显示执行结果。实际上用 getmyTable 建立了一个通用方法，可连接不同的数据库，显示其中任一个表的数据。

图 6-23　getmyTable 方法的测试页面

（5）根据实际情况可对 Web.config 文件进行配置。该 Web 服务建好后，在 VS 2010 中分别单击主菜单中的“生成网站”和“发布网站”，在默认设置情况下，编译后生成的 Web 服务程序位于 d:\My Documents\ Visual Studio 2010\ Projects\ WebService\ Precompiled Web\WebService 文件夹中。

（6）需要将生成的 Web 服务部署在网络上的另一台 Web 服务器上。在该计算机上，新建一个文件夹 D:\Web_Service，将上面编译后生成的 Web 服务程序复制到该文件夹下。我们既可以将这个 Web 服务配置成单独一个网站来提供服务，也可以将它配置成某个现有网站下的一个虚拟目录来提供服务。本书第 2 章关于新建网站和新建虚拟目录方面的内容已作详细介绍。我们在“Internet 信息服务”管理器中新建一个网站，主目录设为 D:\Web_Service，网站端口号为 80，IP 地址为 192.168.0.22，选择 ASP.NET 2.0。此时 Web 服务已开始可以向 Web 客户端提供服务。

（7）下面在 ASP.NET Web 应用程序中调用这个 Web 服务。首先新建一个 ASP.NET Web 应用程序。在 VS 2010 中创建一个新网站后，在解决方案资源管理器中鼠标右击解决方案后选择“添加 Web 引用”菜单，弹出添加 Web 引用对话框，如图 6-24 所示。

图 6-24　添加 Web 引用对话框

对话框中有三种选项用来查找 Web 服务：①此解决方案中的 Web 服务。当前应用程序中自建一个 Web 服务供当前应用程序调用；②本地计算机上的 Web 服务。使用本机在其他网站中创建的 Web 服务供当前应用程序调用；③浏览本地网络上的 UDDI 服务器。查找本地网络中的 Web 服务来为当前应用系统调用。

在地址栏中添入刚才建立的 Web 服务的地址：http://192.168.0.22/Service.asmx，回车或单击“前往”按钮，结果如图 6-25 所示。默认的 Web 引用名为 WebReferences，这个名称很重要，它是程序使用 Web 服务对象名，修改它为 myWebService，单击“添加引用”按钮。此时会在解决方案资源管理器中自动添加一个 App_WebReferences 文件夹，其下新建 myWebService 文件夹，其中自动生成三个文件 Service.disco、Service.discomap 和 Service.wsdl 文件。disco 文件为静态发现文件，用来确定 Web 服务的位置。Service.wsdl 则为 Web 服务的页面描述文件。

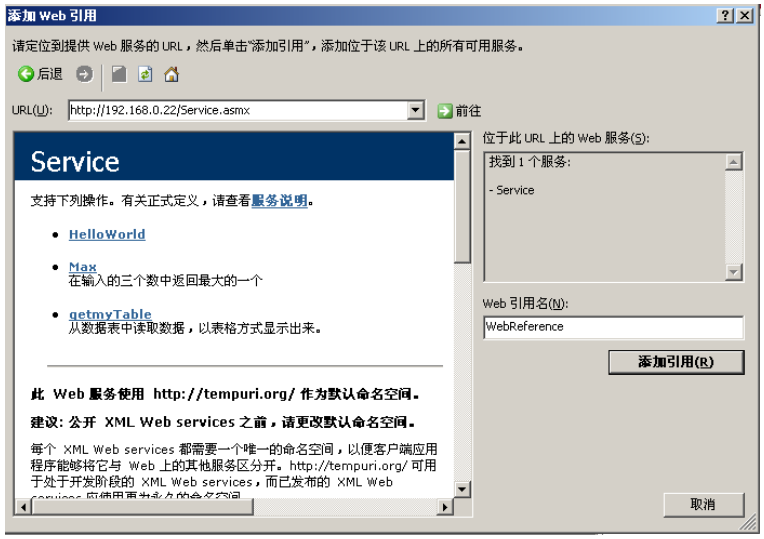


图 6-25　添加 Web 引用

（8）在 Web 窗体中调用 Web 服务。调用方法如下。

① 建立 Web 服务 myWebservice 的一个引用实例 myService：

```
myWebservice.Service myService=new myWebservice.Service();
```

② 调用 myService 对象的方法 helloWorld、Max 和 getmyTable 等，例如：

```
myService.Max(3,100,60)
```

下面在 Default.aspx 窗体的 Default.aspx.cs 后台代码中调用上述 Web 服务的代码为：

```
using System;
using System.Data;
using System.Web;
```

```
    using System.Web.UI;
    using System.Web.UI.WebControls;

    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {  // 建立 Web 服务 myWebservice 的一个引用实例
            myWebservice.Service myService=new myWebservice.Service();

            Response.Write(myService.HelloWorld()+"<br>");
            Response.Write(myService.Max(3,100,60)  + "<br>");

            string  ConStr="server=localhost;database=northwind;  uid=sa;pwd=
docman;";
            string Sql = "select * from employees";
            Response.Write(myService.getmyTable(ConStr,Sql) + "<br>");

        }
    }
```

将 Default.aspx 窗体设为起始页运行后就会在浏览器中看到调用 Web 服务的结果。如果在当前应用中建立 Web 服务来调用，可单击主菜单“网站”|“添加新项”，选择 Web 服务模板，在代码视图中输入 Web 服务的各种方法后，同样要通过添加 Web 引用来调用。

通过上述例子，读者可以体会 Web 服务具有如下技术特点。

（1）跨防火墙的通信

客户端和 Web 服务器之间通常会有防火墙或者代理服务器。采用传统的分布组件技术，例如 DCOM（Distribured Component Object Model）可能会通信失败。

（2）应用程序集成

企业中经常都要把用不同语言写成的、在不同平台上运行的各种程序集成起来。例如应用程序可能需要从运行在 IBM 主机上的程序中获取数据或者把数据发送到主机或 UNIX 应用程序中去。即使在同一个平台上，不同软件厂商生产的各种软件也常常需要集成起来。通过 Web Services，应用程序可以用标准的方法把功能和数据“暴露”出来，供其他应用程序使用。

例如，有一个订单录入程序，用于录入从客户来的新订单，包括客户信息、发货地址、数量、价格和付款方式等内容；还有一个订单执行程序，用于实际货物发送的管理。这两个程序来自不同软件厂商。一份新订单进来之后，订单录入程序需要通知订单执行程序　发送货物。通过在订单执行程序上面增加一层 Web Services，订单执行程序可以把“Add Order”函数“暴露”出来。这样，每当有新订单到来时，订单录入程序就可以调用这个函数来发送货物了。

（3）B2B 的集成

跨公司的商务交易集成通常叫做 B2B 集成。Web Services 是 B2B 集成成功的关键。通

过 Web Services，公司可以把关键的商务应用“暴露”给指定的供应商和客户。例如，把电子下单系统和电子发票系统“暴露”出来，客户就可以以电子的方式发送订单，供应商则可以以电子的方式发送原料采购发票。当然，这并不是一个新的概念，EDI（电子文档交换）早就是这样了。但是，Web Services 的实现要比 EDI 简单得多，而且 Web Services 运行在 Internet 上，在世界任何地方都可轻易实现，其运行成本就相对较低。用 Web Services 来实现 B2B 集成的最大好处在于可以轻易实现互操作性。只要把商务逻辑“暴露”出来，成为 Web Services，就可以让任何指定的合作伙伴调用这些商务逻辑，而不管他们的系统在什么平台上运行，使用什么开发语言。这样就大大减少了花在 B2B 集成上的时间和成本，让许多原本无法承受 EDI 的中小企业也能实现 B2B 集成。

（4）软件和数据重用

软件重用是一个很大的主题，重用的形式很多，重用的程度有大有小。最基本的形式是源代码模块或者类一级的重用，另一种形式是二进制形式的组件重用。

当前，像表格控件或用户界面控件这样的可重用软件组件，在市场上都占有很大的份额。但这类软件的重用有一个很大的限制，就是重用仅限于代码，数据不能重用。原因在于，发布组件甚至源代码都比较容易，但要发布数据就没那么容易，除非是不会经常变化的静态数据。

Web Services 在允许重用代码的同时，可以重用代码背后的数据。使用 Web Services，再也不必像以前那样，要先从第三方购买、安装软件组件，再从应用程序中调用这些组件；只需要直接调用远端的 Web Services 就可以了。例如要在应用程序中确认用户输入的地址，只需将这个地址直接发送给相应的 Web Services，这个 Web Services 就会帮你查阅街道地址、城市、省区和邮政编码等信息，确认这个地址是否在相应的邮政编码区域。Web Services 的提供商可以按时间或使用次数来对这项服务进行收费。这样的服务要通过组件重用来实现是不可能的，那样的话你必须下载并安装好包含街道地址、城市、省区和邮政编码等信息的数据库，而且这个数据库还是不能实时更新的。

另一种软件重用的情况是，把好几个应用程序的功能集成起来。例如，要建立一个局域网上的门户站点应用，让用户既可以查询联邦快递包裹，查看股市行情，又可以管理自己的日程安排，还可以在线购买电影票。现在 Web 上有很多应用程序供应商，都在其应用中实现了这些功能。一旦他们把这些功能都通过 Web Services“暴露”出来，就可以非常容易地把所有这些功能都集成到你的门户站点中，为用户提供一个统一的、友好的界面。

将来，许多应用程序都会利用 Web Services，把当前基于组件的应用程序结构扩展为组件/Web Services 的混合结构，可以在应用程序中使用第三方的 Web Services 提供的功能，也可以把自己的应用程序功能通过 Web Services 提供给别人。两种情况下，都可以重用代码和代码背后的数据。

6.7　Web 开发中的类库构建与访问

在 Web 开发过程中，可将许多通用的完成一定功能的方法或结构等独立出来，放在类库中，编译后形成一个动态链接库 DLL 文件，在 Web 应用程序中通过添加引用后，就可实现

代码的重用。

6.7.1　在 Web 开发中构建一个类库

创建类库过程如下：

（1）进入 VS 2010，新建一个项目，项目类型选择 Visual C#，选择类库模板，默认类库名称为 ClassLibrary1，改为 WebClassLib，单击“确认”后形成类库模板代码。

（2）在解决方案资源管理器中，鼠标右击“引用”后添加必要的.NET 组件名称，例如 System.Web、System.Configuration 等，以便在类代码中可以引用对应的命名空间。将类文件名 Class1.cs 改为 myClass.cs，方法是单击解决方案资源管理器中的文件 Class1.cs，鼠标右击选择重命名或直接按下 F2 来修改。在代码窗口中添加了对命名空间的引用，定义了三个方法 Query、CloseWindow、marriageCodeList，完整代码如下：

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.Data;
    using System.Data.SqlClient;
    using System.Collections;
    using System.Web.UI.WebControls;
    using System.Configuration;

    namespace WebClassLib  //此名称为将来访问此类库文件的命名空间
    {
       public class myClass
       {
          public static DataSet Query(string sql, string Proc_name)
          {
    #if (!DEBUG)      //程序调试过程中，参数传入的传出过程名无效，采用 p_test
             Proc_name ="p_test";
    #else
              Proc_name = "p_Query";
    #endif
              SqlConnection              conn              =              new
SqlConnection(ConfigurationManager.AppSettings["ConnString"]);

              SqlCommand cmd = new SqlCommand();
              cmd.Connection = conn;
              cmd.CommandType = CommandType.StoredProcedure;
              cmd.CommandText = Proc_name;

              SqlParameter paramSQL;
              paramSQL = new SqlParameter("@SQL", SqlDbType.VarChar, 1000);
              paramSQL.Value = sql;
```

```
            cmd.Parameters.Add(paramSQL);
```

```
            conn.Open();
            SqlDataAdapter sda = new SqlDataAdapter(cmd);
            DataSet ds = new DataSet();
            sda.Fill(ds);
            conn.Close();
            return ds;
        }

        public static void CloseWindow(System.Web.UI.Page page)
        {
            Literal lt = new Literal();
            lt.Text = "<script>window.close();</script>";
            page.Controls.Add(lt);
        }

        #region  //用 region 和 endregion 可以实现折叠程序代码
        public static ListItemCollection marriageCodeList()
        {
            ListItem li;
            ListItemCollection lic = new ListItemCollection();
            li = new ListItem("--", "0");
            lic.Add(li);
            li = new ListItem("未婚", "1");
            lic.Add(li);
            li = new ListItem("已婚", "2");
            lic.Add(li);
            li = new ListItem("离婚", "3");
            lic.Add(li);
            li = new ListItem("其他", "4");
            lic.Add(li);
            return lic;
        }
        #endregion
    }
}
```

（3）选择主菜单中的“生成”|“生成 WebClassLib”，如果程序没有错误的话，就会在默认文件夹“我的文档”下的 Visual Studio 2005\Projects\WebClassLib\ WebClassLib\ bin\ Debug 中生成 WebClassLib.dll 文件。这样一个类库文件就生成了。该类库文件就可分发给 Web 开发人员使用。

6.7.2　在 Web 开发中访问类库

下面介绍在 VS 2010 中，在现有的网站中使用上述类库进行 Web 开发。

（1）打开一个网站，选择菜单“网站”|“添加引用”，出现“添加引用”对话框，单击“浏览”标签，选择类库文件 WebClassLib.dll 的存放目录，将它加入到解决方案资源管理器中。可以看到在解决方案资源管理器中增加了一个 bin 目录，类库文件就放在该文件　　夹下。

（2）在 Web 窗体的后台代码中，必须引用 WebClassLib 命名空间，下面代码显示了如何调用类库：

```
using System;
using System.Data;
using System.Web;
using System.Web.UI;
using WebClassLib;  //引用 WebClassLib 命名空间
public partial class _Default : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
WebClassLib.myClass myc= new  WebClassLib.myClass();
DropDownList1.DataSource = myc.marriageCodeList(); //绑定数据
       DropDownList1.DataBind();
       //WebClassLib.myClass.CloseWindow(this); //调用类库中的方法
   }
}
```

6.8　Web 控件开发实例

基于组件的开发允许开发者把应用程序逻辑划分为离散的、独立的块。前面介绍的类库构建和访问是一种没有用户界面的组件开发方法。但在开发过程中也会遇到构建一些通用的具有界面的组件的需求，这种具有界面的组件称为控件。VS 2010 已经提供了很多 Web 开发控件，例如用户的登录控件 login 等，它们放在工具箱中供用户开发时使用。采用由可重用的代码单元构建的控件的好处就是可以创建模块化 Web 应用程序。在 VS 2010 中创建 Web 控件一般有两种方法：①自定义用户控件；②创建工具箱控件。

6.8.1　建立用户自定义控件

在 VS 2010 中可以创建自定义用户控件。自定义 Web 用户控件本质上就是构建一个 Web 窗体，把整个 Web 窗体当作一个控件来使用。将该 Web 窗体用一个 Ascx 为扩展名的文件保存起来，假设为 WebUserControl.ascx，关联的后台服务器代码就放在 WebUserControl.ascx.cs 中。用户控件不能作为一个 Web 窗体来运行，可以将用户控件放在其他 Web 窗体中作为一个独立控件来使用。这样一来，在多个 Web 窗体中可以多次使用用

户控件，实现界面和代码的重用。

（1）构建 Web 用户控件

在当前网站中添加新项，控件名称为 WebUserControl.ascx，选择“Web 用户控件”模板，单击“确定”后，在解决方案资源管理器中生成两个文件 WebUserControl.ascx 和 WebUserControl.ascx.cs。在“设计”中，放入工具箱中的各种 Web 控件构建其他 Web 窗体需要重用的界面，编写程序后台处理代码后保存。例如，设计一个简单的登录界面的 WebUserControl.ascx 文件内容为：

```
    <%@    Control    Language="C#"    AutoEventWireup="true"    CodeFile=
"WebUserControl. ascx.cs"
        Inherits="WebUserControl" %>
    <asp:Panel ID="Panel1" runat="server" Height="116px" Width="281px">
        用户名:<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox> <br />
        密码:  <asp:TextBox ID="TextBox2" runat="server"
               OnTextChanged="TextBox2_TextChanged"></asp:TextBox><p />
    <asp:Button ID="Button1" runat="server" Text="OK" Width="42px" OnClick=
"Button1_Click" />
    <asp:Button       ID="Button2"       runat="server"       Text="Reset"
OnClick="Button2_Click" /></asp:Panel>
```

WebUserControl.ascx.cs 代码为：

```
    using System;
    using System.Data;
    using System.Web;
    using System.Web.UI;
    using System.Web.UI.WebControls;
    using System.Web.UI.HtmlControls;

    public partial class WebUserControl : System.Web.UI.UserControl
    {
        protected void Button2_Click(object sender, EventArgs e)
        {   TextBox1.Text = "";
            TextBox2.Text = "";
        }
        protected void Button1_Click(object sender, EventArgs e)
        {   Response.Write(TextBox1.Text);
        }
    }
```

（2）使用 Web 用户控件

定义的用户控件可以多次被嵌入到不同的 Web 窗体中进行重用。在一个新建的 Web 窗体 Default1.aspx 中，用鼠标将解决方案资源管理器中的 ascx 控件文件拖放到 Web 窗体中，形成的 Web 窗体代码为：

```
1  <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default1.aspx.cs"
2  Inherits="Default1" %>
3  <%@    Register    Src="WebUserControl.ascx"    TagName="WebUserControl"
   TagPrefix= "uc1" %>
   <html xmlns="http://www.w3.org/1999/xhtml" >
4  <head runat="server"> <title>无标题页</title></head>
5  <body>
6     <form id="form1" runat="server">
7     <div>
8        <uc1:WebUserControl ID="WebUserControl1" runat="server" />
9     </div>
      </form>
10 </body>
11 </html>
12
13
```

代码中第 3 行指定该页面将引用一个 TagName 为 WebUserControl 的用户控件，Src 指定用户控件文件的位置。第 9 行是呈现用户控件的代码，其前缀 uc1 在第 3 行的 TagPrefix 中指定，可以更改成当前页面中的唯一的标识。

另外我们还可通过后台编程方式动态调用用户控件，方法是在需要填入控件的地方，放个容器，比如 td，并设定在服务器端运行，例如：

```
<TD  id="tdpan" runat=server></TD>
```

在后台.cs 代码中动态装入用户控件，代码为：

```
UserControl      myusercontrol      =      (UserControl)      LoadControl
("~/WebUserControl.ascx") ;
tdpan.Controls.Clear();
tdpan.Controls.Add(myusercontrol);
```

如果要在其他 Web 开发者之间共享定义的用户控件，可直接复制.ascx 文件和相应的 cs 文件到当前解决方案资源管理器中。

6.8.2　工具箱控件的创建与使用

ASP.NET 4.0 允许你自定义工具箱控件，其中新的自适应呈现模型减少了编写可专门识别其目标浏览器的控件的需要。换句话说，控件开发人员可以专注于设计控件，而让 ASP.NET 框架负责转换控件并针对不同类型的浏览器和设备呈现它。创建自定义服务器控件要用到 HtmlTextWriter 类，主要功能就是输出标记字符和文本输出，另外工具箱中控件的属性在编辑器上是分类的，如外观、行为、布局等，这些属性通过元数据来制定。元数据写

法如[CategoryAttribute("Appearance")]。要使用元数据，必须引用 System. ComponentModel 命名控件。

【例 6.46】 构建一个如图 6-26 所示的工具箱控件。新建一个类库项目 ToolBoxControl，通过添加引用 System.Web 命名空间，然后把下面的代码放在 Class1.cs 类文件中，编译后形成 ToolBoxControl.dll 文件。该文件默认位置在我的文档中的 Visual Studio 2010\Projects\ToolBoxControl\ToolBoxControl\bin\Debug 下。

图 6-26　工具箱控件的效果

接着在 Web 网站中的工具箱中，鼠标右击“常规”，选择“选择项”后，通过浏览找到生成的 DLL 文件后，会发现.NET Framework 组件列表框中已经增加了 CreditCardForm 组件，单击“确定”按钮后工具箱中已经自动帮你加上了自定义的控件。

自定义控件中放置了 6 个 HTML 控件。HTML 控件前面有 4 个用户可读写的文本标签属性。类文件 Class1.cs 代码如下：

```
    using System;
    using System.Web.UI;
    using System.ComponentModel;

    namespace CustomComponents  //命名空间的定义，可改为其他
    {
        [DefaultPropertyAttribute("CardholderNameText")]  //指定控件的默认属性
        //1.控件标签定义。{0}为将来用户自定义的控件标识。定义中指定了几个初始值
        [ToolboxData(@"<{0}:CreditCardForm PaymentMethodText='信用卡类型'
            CreditCardNoText='信用卡卡号' CardholderNameText='信用卡持有者姓名'
            SubmitButtonText = '提交' runat='server'></{0}:CreditCardForm>")]
        //2.构建继承了 Control 类的 CreditCardForm 控件类
        public class CreditCardForm : Control
        {
            private string paymentMethodText = "信用卡类型";
            private string creditCardNoText = "信用卡卡号";
            private string cardholderNameText = "信用卡持有者姓名";
            private string expirationDateText = "最后使用时间";
            private string submitButtonText = "提交";

            [BrowsableAttribute(true)] //该属性是否放在属性窗口中
            [DescriptionAttribute("获取和设置信用卡类型")] //该属性的描述说明
            [DefaultValueAttribute("信用卡类型")]  //指定属性的默认值
            [CategoryAttribute("Appearance")]  //指定该属性将放在外观类
            public virtual string PaymentMethodText  //定义该属性的名称和是否可读
写
            {   get { return this.paymentMethodText; }
                set { this.paymentMethodText = value; }
            }
```

```
        [BrowsableAttribute(true)]
        [DescriptionAttribute("获取或设置信用卡卡号")]
        [DefaultValueAttribute("信用卡卡号")]
        [CategoryAttribute("Appearance")]
        public virtual string CreditCardNoText
        {   get { return this.creditCardNoText; }
            set { this.creditCardNoText = value; }
        }
```

```
        [BrowsableAttribute(true)]
        [DescriptionAttribute("获取或设置信用卡持有者姓名")]
        [DefaultValueAttribute("信用卡持有者姓名")]
        [CategoryAttribute("Appearance")]
        public virtual string CardholderNameText
        {   get { return this.cardholderNameText; }
            set { this.cardholderNameText = value; }
        }

        [BrowsableAttribute(true)]
        [DescriptionAttribute("获取或设置最后使用时间")]
        [DefaultValueAttribute("最后使用时间")]
        [CategoryAttribute("Appearance")]
        public virtual string ExpirationDateText
        {   get { return this.expirationDateText; }
            set { this.expirationDateText = value; }
        }

        [BrowsableAttribute(true)]
        [DescriptionAttribute("获取或设置按钮标签")]
        [DefaultValueAttribute("提交")]
        [CategoryAttribute("Appearance")]
        public virtual string SubmitButtonText
        {
            get { return this.submitButtonText; }
            set { this.submitButtonText = value; }
        }

        protected override void Render(HtmlTextWriter writer)
        {    //以 HTML 表格方式呈现控件外形,也可设计成其他方式如 div
            writer.Write("<table style='width:287px;height:124px; border-
width:0;'>");
            writer.Write("<tr><td>" + PaymentMethodText + "</td>");
     writer.Write("<td><select   name='PaymentMethod'   id='PaymentMethod'
style= 'width:100%;'>");
```

```
            writer.Write("<option value='0'>Visa</option>");
            writer.Write("<option  value='1'>MasterCard</option></select>
</td></tr>");
            writer.Write("<tr><td>" + CreditCardNoText + "</td>");
   writer.Write("<td><input     name='CreditCardNo'     id='CreditCardNo'
type='text' /></td>");
            writer.Write("</tr>"<tr><td>" + CardholderNameText + "</td>");
   writer.Write("<td><input   name='CardholderName'   id='CardholderName'
type='text' /></td>");
            writer.Write("</tr><tr><td>" + ExpirationDateText + "</td>");
            writer.Write("<td><select name='Month' id='Month'>");
```

```
            for (int day = 1; day < 13; day++)
            {   if (day < 10)
   writer.Write("<option  value='"  +  day.ToString()  +  "'>"  +  "0"  +
day.ToString() + "</option>");
                else
   writer.Write("<option value='" + day.ToString() + "'>" + day.ToString()
+ "</option>");
            }
            writer.Write("</select> <select             name='Year'
id='Year'>");
            for (int year = 2005; year < 2015; year++)
            {
   writer.Write("<option value='" + year.ToString() + "'>" + year.ToString()
+ "</option>");
            }
            writer.Write("</select></td></tr><tr><td       align='center'
colspan='2'>");
            writer.Write("<input type='submit' value='" + SubmitButtonText
+ "' />");
            writer.Write("</td></tr></table>");
            base.Render(writer); //呈现控件
         }
      }
   }
```

代码中用大量的字符串来输出 HTML，而且容易输错，所以 HtmlTextWriter 类提供 AddStyleAttribute、AddAttribute、RenderBeginTag 和 RenderEndTag 几个有用的方法用来改善输出的复杂性，此处不再介绍，可参阅相关资料。

将自定义工具箱控件拖动到Web窗体设计界面中，可以和其他Web控件一样来使用它。其页面文件中增加了控件注册和标签输出两部分。控件注册中，程序集 Assembly 即为类库名，命名空间为类库代码中定义的，控件标签 cc1 可改为其他唯一的标识。

```
    <%@  Register  Assembly="ToolBoxControl"  Namespace="CustomComponents"
TagPrefix="cc1" %>
    …
    <cc1:creditcardform          id="CreditCardForm1"          runat="server"
cardholdernametext="信用卡持有者姓名"    creditcardnotext="信用卡卡号"
paymentmethodtext="信用卡类型" submitbuttontext="提交"></cc1:creditcardform>
```

6.9　ASP.NET 中 XML 编程基础

ASP.NET 通过 System.Xml 命名空间为开发者提供了操纵 XML 的所有功能。该命名空间包含许多类，主要类名称如表 6-58 所示。

表 6-58　System.Xml 命名空间中常用类名称

类	说　明
XmlAttribute	表示一个属性。此属性的有效值和默认值在文档类型定义(DTD)或架构中进行定义
XmlCDataSection	表示 CDATA 节
XmlCharacterData	提供多个类使用的文本操作方法
XmlComment	表示 XML 注释的内容
XmlConvert	对 XML 名称进行编码和解码并提供方法在公共语言类型库类型和 XML 架构定义语言（XSD）类型之间进行转换。当转换数据类型时，返回的值是独立于区域设置的
XmlDataDocument	允许通过相关的 DataSet 存储、检索和操作结构化数据
XmlDeclaration	表示 XML 声明节点：<?xml version='1.0'...?>
XmlDocument	表示 XML 文档
XmlDocumentType	表示文档类型声明
XmlElement	表示一个元素
XmlException	返回有关最后一个异常的详细信息
XmlNode	表示 XML 文档中的单个节点
XmlNodeList	表示排序的节点集合
XmlNodeReader	表示提供对 XmlNode 中的 XML 数据进行快速、非缓存的只进访问的读取器
XmlReader	表示提供对 XML 数据进行快速、非缓存、只进访问的读取器
XmlText	表示元素或属性的文本内容
XmlTextReader	表示提供对 XML 数据进行快速、非缓存、只进访问的读取器
XmlTextWriter	表示提供快速、非缓存、只进方法的编写器，该方法生成包含 XML 数据（这些数据符合 W3C 可扩展置标语言 (XML) 1.0 和“XML 中的命名空间”建议）的流或文件
XmlWriter	表示一个编写器，该编写器提供一种快速、非缓存和只进的方式来生成包含 XML 数据的流或文件

ASP.NET 中有关 XML 编程技术内容相当广泛。本书在此仅通过下面三个例子来让读者体会 XML 的编程过程。

【例 6.47】 将 SQL Server Northwind 数据库中的 Customers 数据表中的数据转换成一个 XML 文档。其完整代码如下：

```
    using System;
```

```
    using System.Data;
    using System.Collections;
    using System.Web;
    using System.Web.UI;
    using System.Web.UI.WebControls;
    using System.Data.SqlClient;
    using System.Xml;  //必须引用此命名空间

    public partial class Write2XML : System.Web.UI.Page
    {   protected void Page_Load(object sender, EventArgs e)
        {         string    conStr    =    "Server=(local);database=northwind;
Uid=sa;Pwd=docman";
          string mysql = "SELECT * FROM Customers";
```

```
          SqlConnection conn = new SqlConnection(conStr);
          conn.Open();
          try
          {   SqlDataAdapter da = new SqlDataAdapter(mysql, conn);
              DataSet ds = new DataSet();
              da.Fill(ds, "Customers");
              DataTable dt = ds.Tables["Customers"];

              //1. WriteXml 方法可直接将 DataTable 中的所有数据写入 XML 文档
              dt.WriteXml(@"D:\Customers1.xml"); //写到服务器的 D 盘上

              //2. 可有选择性地将数据写入 XML 文档
              XmlDocument  xdoc  =  new  XmlDocument();    //新建一个 XML 文档对象
xdoc
              XmlDeclaration xdecl = xdoc.CreateXmlDeclaration("1.0", "utf-
8", "yes");
              xdoc.AppendChild(xdecl); //写入 XML 文档标志
              XmlElement myCustomer = xdoc.CreateElement("Customers");//写入
                                                                    //根节
点
              xdoc.AppendChild(myCustomer);

              for (int i = 0; i < dt.Rows.Count; i++)  //对每一行循环
              {   XmlElement xdoc_Customer = xdoc.CreateElement("Customer");

                  for (int j = 0; j < dt.Columns.Count; j++) //对每一列循环
                  {   XmlElement ColName = xdoc.CreateElement(dt.Columns[j].
ColumnName);
                    ColName.InnerText = dt.Rows[i][j].ToString();
                    xdoc_Customer.AppendChild(ColName);
                  }
```

```
                xdoc.DocumentElement.AppendChild(xdoc_Customer);
            }
            xdoc.Save(@"D:\Customers.xml");  //写入 XML 文档到服务器的 D 盘
            Response.Write("XML document has generated sucessfully!");
        }
        finally
        { conn.Close();
        }
    }
}
```

上述代码执行后将生成两个 XML 文档。

【例 6.48】 将 XML 文档内容写入数据库中。在 SQL Server Northwind 数据库中创建一个表：create table tb_speciality(speciality_id int not null, speciality_name varchar(40))。对应的 XML 文档 D:\specialities.xml 为：

```
<?xml version="1.0" encoding="utf-8"?>
<specialities>
  <speciality>
    <speciality_id>100</speciality_id>
    <speciality_name>计算机软件与理论</speciality_name>
  </speciality>
  <speciality>
    <speciality_id>101</speciality_id>
    <speciality_name>计算机应用技术</speciality_name>
  </speciality>
  <speciality>
    <speciality_id>102</speciality_id>
    <speciality_name>通信工程</speciality_name>
  </speciality>
</specialities>
```

代码中给出了两种处理方式：

```
using System;
using System.Data;
using System.Web;
using System.Web.UI;
using System.Data.SqlClient;
using System.Xml;  //必须引用此命名空间

public partial class Read_fromXML : System.Web.UI.Page
{   protected void Page_Load(object sender, EventArgs e)
    {         string   conStr   =   "Server=(local);database=northwind;
Uid=sa;Pwd=docman";
```

```
        string mysql = "SELECT speciality_id,speciality_name FROM
tb_speciality";
        SqlConnection conn = new SqlConnection(conStr);
        conn.Open();
        try
        {   SqlDataAdapter da = new SqlDataAdapter(mysql, conn);
            DataSet ds = new DataSet();
            da.Fill(ds, "tb_speciality");
            DataTable dt = ds.Tables["tb_speciality"];

            //1. ReadXml 方法可直接将 XML 中的所有数据读入 DataTable
            //此处的 XML 文档放在服务器的 D 盘上，如放在网站中的某位置
            //用 dt.ReadXml(Server.MapPath(~/tb_speciality.xml"));
            dt.ReadXml(@"D:\tb_speciality.xml");
            SqlCommandBuilder objCB = new SqlCommandBuilder(da); //连接数据
库
            da.Update(ds, "tb_speciality");                      //更新数据
库
```

```
            Response.Write("Write Data from XML to Database has completed
sucessfully!");

            //2. 可有选择性地将 XML 文档中的数据写入数据库
            XmlDocument xdoc = new XmlDocument();
            xdoc.Load(@"D:\tb_speciality.xml");  //加载 XML 文档到内存
            XmlElement spe = xdoc.DocumentElement; //得到整个 XML 文档

            for (int i = 0; i < spe.ChildNodes.Count; i++)  //对所有子节点循
环
            {   XmlNode my_spe = spe.ChildNodes[i];

                int                         spe_id                        =
Convert.ToInt32(my_spe.ChildNodes[0].InnerText);
                string spe_name = my_spe.ChildNodes[1].InnerText;

                DataRow newSpeciality = dt.NewRow();  //定义一条新的行记录
                newSpeciality["speciality_id"] = spe_id;
                newSpeciality["speciality_name"] = spe_name;
                dt.Rows.Add(newSpeciality);

                //SqlCommandBuilder objCB = new SqlCommandBuilder(da);
                da.Update(ds, "tb_speciality");  //写入数据库
            }
            Response.Write("Write Data from XML to Database has completed
sucessfully!");
```

```
        }
        finally
        {   conn.Close();
        }
    }
}
```

【例 6.49】 将 golf.xml 文件按照 golf.xsl 文件的样式要求转换输出到浏览器上。最终结果如图 6-27 所示。Golf.xml 文件内容为：

```
<?xml version="1.0" ?>
<golfers>
  <golfer skill="excellent" handicap="10" clubs="Taylor Made" id="1111">
    <name>
      <firstName>Heedy</firstName>
      <lastName>Wahlin</lastName>
    </name>
    <favoriteCourses>
      <course city="Pinetop" state="AZ" name="Pinetop Lakes CC"/>
```

图 6-27　XML 文档的转换输出结果

```
      <course city="Phoenix" state="AZ" name="Ocotillo"/>
      <course city="Snowflake" state="AZ" name="Silver Creek"/>
    </favoriteCourses>
  </golfer>
  <golfer skill="moderate" handicap="12" clubs="Taylor Made" id="2222">
    <name>
      <firstName>Dan</firstName>
      <lastName>Wahlin</lastName>
```

```
      </name>
      <favoriteCourses>
        <course city="Pinetop" state="AZ" name="Pinetop Lakes CC"/>
        <course city="Pinetop" state="AZ" name="White Mountain CC"/>
        <course city="Springville" state="UT" name="Hobble Creek"/>
      </favoriteCourses>
    </golfer>
  </golfers>
```

Golf.xsl 文件内容为：

```
  <?xml version="1.0" ?>
  <xsl:stylesheet        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
    <xsl:output indent="yes" method="html"/>
    <xsl:template match="/">
      <html><head>
         <style type="text/css">
           .blackText {font-family:arial;color:#000000;}
           .largeYellowText               {font-family:arial;font-size:18pt;
color:#ffff00;}
```

```
           .largeBlackText                        {font-family:arial;font-
size:14pt;color:#000000;}
           .borders {border-left:1px solid #000000;
            border-right:1px solid #000000;
            border-top:1px solid #000000;
            border-bottom:1px solid #000000;}
         </style></head>
        <body bgcolor="#ffffff">
          <span class="largeBlackText"> <b>Golfers: </b> </span> <p/>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="golfers">
      <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="golfer">
      <table class="borders" border="0" width="640" cellpadding="4"
       cellspacing="0" bgcolor="#efefef">
        <xsl:apply-templates select="name"/>
        <tr class="blackText">
          <td width="12%" align="left"> <b>Skill: </b> </td>
```

```
          <td        width="12%"        align="left">        <xsl:value-of
select="@skill"/></td>
          <td width="12%" align="left"> <b>Handicap: </b> </td>
          <td  width="12%"   align="left"> <xsl:value-of  select="@handicap"/>
</td>
          <td width="12%" align="left"> <b>Clubs: </b> </td>
          <td  width="40%"  align="left">  <xsl:value-of  select="@clubs"/>
</td>
        </tr>
        <tr> <td colspan="6"> </td> </tr>
        <tr class="blackText"> <td colspan="6" class="largeBlackText">
           Favorite Courses   </td> </tr>
        <tr> <td colspan="2"> <b>City: </b>     </td>
            <td colspan="2">    <b>State: </b>  </td>
            <td colspan="2">    <b>Course: </b> </td> </tr>
        <xsl:apply-templates select="favoriteCourses"/>
      </table>  <p/>
    </xsl:template>

    <xsl:template match="name">
      <tr> <td colspan="6" class="largeYellowText" bgcolor="#02027a">
         <xsl:value-of             select="firstName"/> <xsl:value-of
select="lastName"/>
        </td> </tr>
    </xsl:template>
```

```
    <xsl:template match="favoriteCourses">
      <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="course">
      <tr class="blackText">
        <td colspan="2" align="left">
          <xsl:value-of select="@city"/>
        </td>
        <td colspan="2" align="left"> <xsl:value-of select="@state"/> </td>
        <td colspan="2" align="left"> <xsl:value-of select="@name"/>  </td>
      </tr>
    </xsl:template>
  </xsl:stylesheet>
```

Golf.aspx.cs 文件内容：

```
  using System;
  using System.Data;
  using System.Web;
```

```
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;
using System.Xml.Xsl;  //必须使用的命名空间
using System.Xml.XPath; //必须使用的命名空间
public partial class xslt_golf : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {   string xmlPath = Server.MapPath("golf.xml");
        string xslPath = Server.MapPath("golf.xsl");
        //Instantiate the XPathDocument Class
        XPathDocument doc = new XPathDocument(xmlPath);

        //Instantiate the XslTransform Class
        XslTransform transform = new XslTransform();
        transform.Load(xslPath); //加载 XSL 文件

        //Custom format the indenting of the output document by using an
        //XmlTextWriter
        XmlTextWriter writer = new XmlTextWriter(Response.Output);
        writer.Formatting = Formatting.Indented;
        writer.Indentation = 4;
        transform.Transform(doc, null, writer); //进行转换输出
    }
}
```

思考练习题

1．熟悉 ASP.NET 的各种常用控件。

2．简述 GridView 控件、DataList 控件和 Repeater 控件的优缺点及它们分别使用的场合。

3．写一个 ASP.NET 页面，该页面使用连接对象连接 Northwind 数据库，并使用 GridView 控件显示下面的信息：

① 所有供应商的地址、所在城市、联系人姓名和电话号码；

② 所有雇员的姓名和地址，按年龄降序显示。

4．分别使用 DataList 控件和 Repeater 控件实现对 Northwind 数据库中 Products 表的数据显示、分页和排序。

5．使用 FormView 控件实现对 Northwind 数据库中 Categories 表的数据显示，并且实现增加编辑和添加数据记录功能。效果如图 6-28 所示。

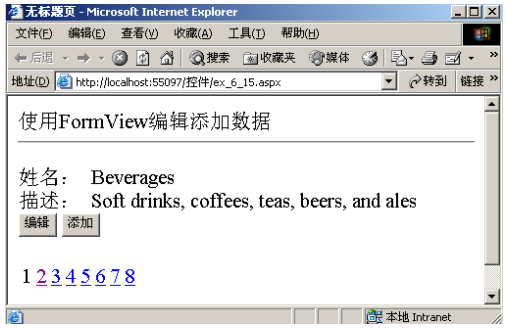


图 6-28　第 5 题的运行效果

6．请说出下列代码在浏览器中的输出形式。

```
    <%@          Page          Language="C#"          AutoEventWireup="true"
CodeFile="Default2.aspx.cs" Inherits="Default2" %>
    <% string hisName = "Yangking"; %>
    <html><head runat="server"><title>无标题页</title></head>
    <body>
        <form id="form1" runat="server">
            姓名: <input id="Text1" type="text" value="<%=myName%>" /></form>
         <script>alert('<%=myName %>');</script>
         <script>alert('<%= hisName %>');</script>
    </body></html>
    Default2.aspx.cs:
    public partial class Default2 : System.Web.UI.Page
    {   protected  string myName = "WangCL";
        protected void Page_Load(object sender, EventArgs e)
        {   myName = "WangCL";
        }
    }
```

7．结合前面几章介绍的知识开发一个简单的数据库应用系统。