单元 5　使用 ADO. NET 访问数据库

使用 ASP. NET 开发 Web 应用程序时，为了使客户端能够访问服务器中的数据库，经常需要用到对数据库的各种操作，而这其中，ADO. NET 技术是一种最常用的数据库操作技术。ASP. NET 技术是一组向. NET 开发人员公开数据访问服务的类，它为创建分布式数据共享应用程序提供了一组丰富的组件。本单元将对 ADO. NET 数据访问技术进行详细讲解。

本单元主要学习目标如下。

- 理解 ADO. NET 的相关概念及其结构。
- 掌握数据库的两种访问模式及其区别。
- 熟练掌握 Connection 对象的使用。
- 熟练掌握 Command 对象的使用。
- 掌握 DataReader 对象的使用。
- 熟练掌握 DataAdapter 对象的使用。
- 掌握 DataSet 对象中常用的方法，并高效使用 DataSet 开发。

5.1　ADO. NET 概述

ADO. NET 是. NET Framework 提供的数据访问服务的类库，ADO. NET 对 Microsoft SQL Server、Oracle 和 XML 等数据源提供一致的访问，应用程序可以使用 ADO. NET 连接到这些数据源，并检索和更新所包含的数据。

5.1.1　ADO. NET 简介

ADO. NET 的名称起源于 ADO（ActiveX Data Objects），ADO 用于在以往的 Microsoft 技术中进行数据的访问。所以微软公司希望通过使用 ADO. NET 向开发人员表明，这是在. NET 编程环境和 Windows 环境中优先使用的数据访问接口。

ADO. NET 提供了平台互用性和可伸缩的数据访问，增强了对非连接编程模式的支持，并支持 RICH XML。由于传送的数据都是 XML 格式的，因此任何能够读取 XML 格式的应用程序都可以进行数据处理。事实上，接收数据的组件不一定非要是 ADO. NET 组件，它可以是基于一个 Microsoft Visual Studio 的解决方案，也可以是运行在其他平台上的任何应用程序。

传统的 ADO 和 ADO. NET 是两种不同的数据访问方式，无论是在内存中保存数据，还是打开和关闭数据库的操作模式都不尽相同。

5.1.2 ADO.NET 的结构

1. ADO.NET 的模型

ADO.NET 采用层次管理模型，各部分之间的逻辑关系如图 5-1 所示。

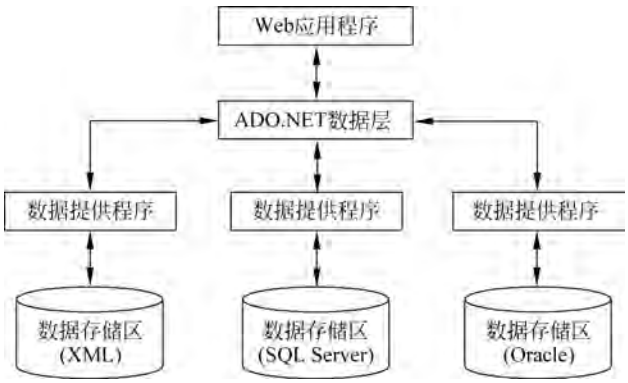

图 5-1　ADO.NET 的模型

ADO.NET 模型的最顶层是 Web 应用程序，中间是 ADO.NET 数据层和数据提供程序，在这个层次中数据提供程序相当于 ADO.NET 的通用接口，各种不同的数据源要使用不同的数据提供程序。

2. ADO.NET 的主要组件

ADO.NET 用于数据访问的类库包含.NET Framework 数据提供程序和 DataSet(数据集)两个组件。.NET Framework 数据提供程序和 DataSet 之间的关系如图 5-2 所示。

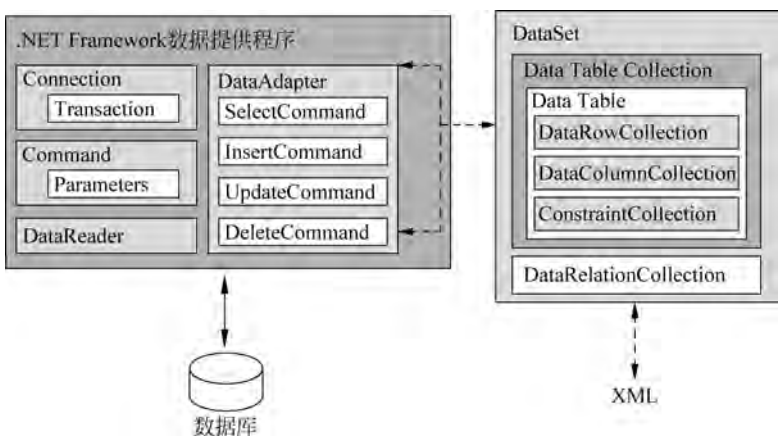

图 5-2　ADO.NET 的主要组件

.NET Framework 数据提供程序包括以下四个核心对象。

(1) Connection：建立与数据源的连接。

(2) Command：对数据源执行操作命令，用于修改、查询数据和运行存储过程等。

(3) DataReader：从数据源获取返回的数据。

(4) DataAdapter：用数据源数据填充 DataSet，并更新数据。

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，可以把它看成是内存中的数据库，是专门用来处理数据源中读出的数据的。

DataSet 的优点就是离线式，一旦读到数据库中的数据后，就在内存中建立数据库的副本，在此之后的操作，直到执行更新命令为止，所有的操作都是在内存中完成的。不管底层的数据库是哪种类型，DataSet 的行为都是一致的。

DataSet 是数据表(DataTable)的集合，它可以包含任意多个数据表，而且每个 DataSet 中的数据表对应一个物理数据库中的数据表(Table)或者数据视图(View)。

5.1.3 与数据有关的命名空间

在 ADO.NET 中，连接数据源的接口有以下四种：SQLClient、OracleClient、ODBC、OLEDB。其中，SQLClient 是 Microsoft SQL Server 数据库专用连接接口，OracleClient 是 Oracle 数据库专用连接接口，ODBC 和 OLEDB 可用于其他数据源的连接。在应用程序中使用任何一种连接接口时，必须在后台代码中引用相应的命名空间，类的名称也随之发生变化，如表 5-1 所示。

表 5-1 ADO.NET 的数据库命名空间及其说明

命名空间	说明
System.Data	ADO.NET 的核心，包含处理非连接的架构所涉及的类，如 DataSet
System.Data.SqlClient	SQL Server 的.NET 数据提供程序
System.Data.OracleClient	Oracle 的.NET 数据提供程序
System.Data.OleDb	OLE DB 的.NET 数据提供程序
System.Data.Odbc	ODBC 的.NET 数据提供程序
System.Xml	提供基于标准 XML 的类、结构等
System.Data.Common	由.NET 数据提供程序继承或者实现的工具类和接口

5.1.4 ADO.NET 数据提供者

ADO.NET 的一个核心成员——数据提供者(Data Provider)是一个类库，它可以被看成是数据库与应用程序的一个接口或中间件。由于现在使用的数据源种类很多，在编写应用程序时就要针对不同的数据源编写不同的接口代码，工作量很大且效率低下。数据提供者针对这一问题向应用程序提供了统一的编程界面，向数据源提供了多种数据源接口，即对数据源进行了屏蔽，可以使应用程序不必关心数据源的种类。

ADO.NET 提供与数据源进行交互的公共方法，但是对于不同的数据源要采用一组不同的类库，这些类库被称为数据提供者。数据提供者的命名通常是以与之交互的协议和数据源的类型来命名的。表 5-2 所示列出了一些常见的数据提供程序及其支持的数据源类型。

表 5-2　常见的数据提供程序及其支持的数据源类型

数据提供程序	支持的数据源类型
ODBC Data Provider	提供 ODBC 接口的数据源，包括 Access、Oracle、SQL Server、MySQL 和 Visual FoxPro 等老式数据源
OLE DB Data Provider	提供 OLE DB 接口的数据源，如 Access、Excel、Oracle 和 SQL Server
Oracle Data Provider	用于 Oracle 数据库
SQL Data Provider	用于 Microsoft SQL Server 7 或更高版本、SQL Express 或 MSDE
Borland Data Provider	许多数据库的公共存取方式，如 Interbase、SQL Server、IBM DB2 和 Oracle

具体使用哪种数据提供程序，要根据应用程序所使用的数据库来确定。

5.1.5　ADO.NET 对象模型

ADO.NET 对象是指包含在.NET Framework 数据提供程序和数据集 DataSet 中的对象，其中，DataSet 对象是驻留在内存中的数据库，位于 System.Data 命名空间下。ADO.NET 从数据库中抽取数据后数据就存放在 DataSet 中，故可以把 DataSet 看成一个数据容器。.NET Framework 数据提供程序包括 Connection、Command、DataReader、DataSet 和 DataAdapter 五个对象。ADO.NET 对象关系模型如图 5-3 所示。

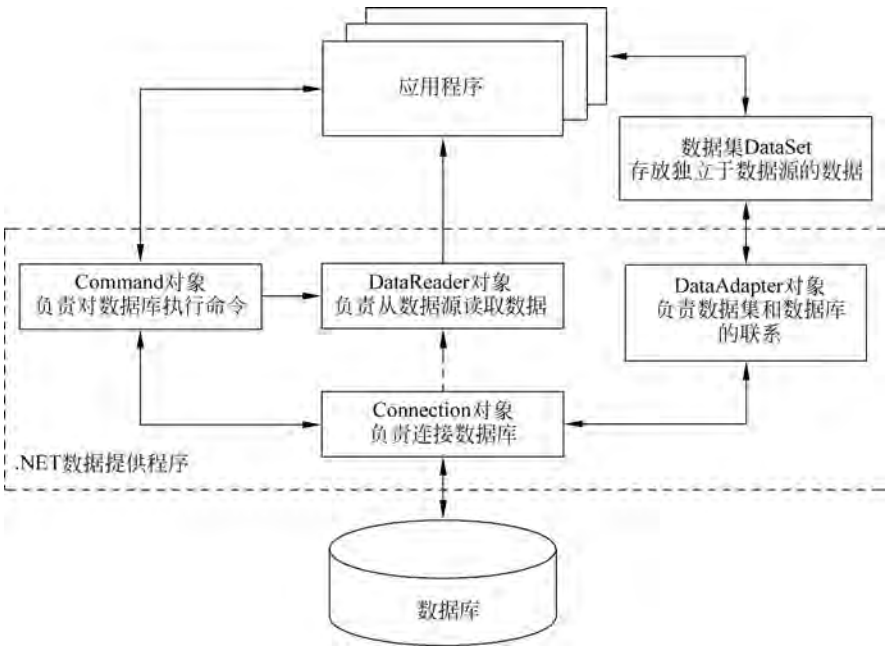

图 5-3　ADO.NET 对象关系模型

ADO.NET 的五大对象可以形象地记为连接 Connection、执行 Command、读取 DataReader、填充 DataSet、分配 DataAdapter。这正是 ADO.NET 对数据库操作的一般步骤。下面将详细介绍这些对象。

5.2 Connection 数据连接对象

5.2.1 Connection 对象概述

当应用程序要访问数据时,怎样才能够找到数据库呢? 这就需要使用 Connection 对象。在 ADO.NE 对象模型中,Connection 对象用于连接到数据库和管理数据库的事务。不同的数据源(.NET 数据提供程序)需要使用不同的类来建立连接。例如,要连接到 SQL Server,需要选择 SqlConnection 连接类。根据不同的数据源提供了表 5-3 所示的四种数据库连接方式。

表 5-3 .NET 数据提供程序及对应的连接类

数据访问提供程序	名 称 空 间	对应的连接类名称
SQL Server 数据提供程序	System.Data.SqlClient	SqlConnection
OLEDB 数据提供程序	System.Data.OleDb	OledbConnection
ODBC 数据提供程序	System.Data.Odbc	OdbcConnection
Oracle 数据提供程序	System.Data.OracleClient	OracleConnection

5.2.2 Connection 对象的常用属性和方法

ADO.NET 使用 SqlConnection 对象与 SQL Server 进行连接,下面以 SqlConnection 为例介绍 Connection 对象的使用。SqlConnection 对象提供了一些属性和方法,允许程序员与数据源建立连接或断开连接。SqlConnection 对象的常用属性和方法如表 5-4 所示。

表 5-4 SqlConnection 对象的常用属性和方法

属性和方法	说 明
ConnectionString 属性	获取和设置数据库的连接字符串
ConnectionTimeOut 属性	获取 SqlConnection 对象的超时时间,单位为秒,0 表示不限时。如果在这段时间之内无法连接数据源,则产生异常
Database 属性	获取当前数据库名称
DataSource 属性	获取数据源的完整路径和文件名,如果是 SQL Server 数据库,则获取所连接的 SQL Server 服务器名称
State 属性	获取数据库的连接状态,它的值为 ConnectionState 枚举值
Open 方法	打开与数据库的连接
Close 方法	关闭与数据库的连接
ChangeDatabase 方法	在打开连接的状态下,更改当前数据库
CreateCommand 方法	创建并返回与 SqlConnection 对象有关的 SqlCommand 对象
Dispose 方法	调用 Close 方法关闭与数据库的连接,并释放所占用的系统资源

注意:除了 ConnectionString 属性之外,其他属性都是只读属性,只能通过连接字符串的标记配置数据库连接。

在 ADO.NET 中,如果使用.NET Framework 数据提供程序操作数据库,必须显示关

闭与数据库的连接，也就是说在操作完数据库后，必须调用 Connection 对象的 Close 方法关闭连接。

5.2.3 使用 SqlConnection 对象连接数据库

建立应用程序与数据库连接需要以下三个步骤。

1. 定义数据库连接字符串

定义连接字符串的常用方式有以下两种。

1）使用 Windows 身份验证

该方式又称信任连接，这种连接方式有助于在连接到 SQL Server 时提供安全保护，因为它不会在连接字符串中公开用户 ID 和密码，是安全级别要求较高时推荐的数据库连接方法。其连接字符串的语法格式如下：

```
string ConnStr = "Server = 服务器名或 IP;Database = 数据库名;Integrated Security = true";
```

2）使用 SQL Server 身份验证

该方式又称非信任连接，这种连接方式把未登录的用户 ID 和密码写在连接字符串中，因此在安全级别要求较高的场合不要使用。其连接字符串的语法格式如下：

```
string ConnStr = "Server = 服务器名;Database = 数据库名;uid = 用户名;pwd = 密码";
```

或

```
string ConnStr = "Data Source = 服务器名;Initial Catalog = 数据库名; User ID = 用户名;Pwd = 密码";
```

数据库连接字符串由多个分号隔开的多个参数组成，其常用参数及其说明如表 5-5 所示。

表 5-5 Sqlconnection 对象的连接字符串参数及其说明

参　数	说　明
Data Source 或 Server	连接打开时使用的 SQL Server 数据库服务器名称，或者是 Microsoft Access 数据库的文件名，可以是"local""."localhost""127.0.0.1"，也可以是具体数据库服务器名称
Initial Catalog 或 Database	数据库的名称
Integrated Security	此参数决定连接是否是安全连接。可能的值有 true、false 和 SSPI（SSPI 是 true 的同义词）
User ID 或 uid	SQL Server 账户的登录账号
Password 或 pwd	SQL Server 登录密码

例如，“新知书店”应用程序与本机的 BookShopPlus 数据库连接的字符串可以写成：

```
String ConnStr = "Server = .; Database = BookShopPlus; uid = sa; pwd = 123456";
```

说明：如果数据库的密码为空，可以省略 pwd 这一项。

2. 创建 Connection 对象

使用定义好的连接字符串创建 Connection 对象，代码如下所示：

```
SqlConnection sqlconn = new SqlConnection(connStr);
```

3. 打开与数据库的连接

调用 Connection 对象的 Open 方法打开与数据库的连接，代码如下：

```
sqlconn.Open();
```

在上面的这三个步骤中，第 1、2 步的先后顺序可以调换，即可以先创建一个 Connection 对象，再设置它的 ConnectionString 属性，例如：

```
SqlConnection sqlconn = new SqlConnection();
String connStr = "Server = .; Database = BookShopPlus; Uid = sa; pwd = 123456";
sqlconn.ConnectionString = connStr;
```

注意：打开数据库连接，执行命令后，要确保关闭数据库连接。

【示例 5-1】 使用 SqlConnection 对象连接数据库。

使用 Connection 对象建立与 SQL Server 数据库 Student 的连接，并显示当前数据库的连接状态。

(1) 在 SQL Server 2014 中附加数据库文件 Student.mdf。

(2) 在 WebSite05 网站项目中创建文件夹 Ch5_1，在文件夹 Ch5_1 下新建 Web 页面 Default.aspx。在网页中添加一个 Label 标签控件和两个 Button 命令按钮，两个命令按钮的 Text 属性分别设置为“打开连接”和“关闭连接”。

(3) 在 Default.aspx.cs 文件中添加命名空间的引用，代码如下：

```
using System.Data.SqlClient;
```

在 Default.aspx.cs 文件的所有事件之外定义数据库连接字符串和连接对象，代码如下：

```
static string ConStr = " Server = .; Database = Student; Uid = sa; pwd = 123456";
SqlConnection conn = new SqlConnection(ConStr);
```

在 Default.aspx.cs 文件中添加页面载入时执行的 Page_Load 事件过程代码，如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    lblMsg.Text = "当前连接状态是: " + conn.State.ToString();
}
```

在 Default.aspx.cs 文件中分别添加单击“打开连接”和“关闭连接”按钮时执行的事件

过程代码，如下：

```
protected void btnConn_Click(object sender, EventArgs e)
{
    conn.Open();
    lblMsg.Text = "当前连接状态是：" + conn.State.ToString();
}
protected void btnClose_Click(object sender, EventArgs e)
{
    conn.Close();
    lblMsg.Text = "当前连接状态是：" + conn.State.ToString();
}
```

（4）浏览该页面，页面载入时，显示的连接状态是 Closed，打开连接时显示的连接状态是 Open，结果如图 5-4 所示。

图 5-4　使用 SqlConnection 对象连接数据库

5.3　Command 命令执行对象

5.3.1　Command 对象概述

使用 Connection 对象与数据源建立连接后，可以使用 Command 对象对数据源执行查询、添加、删除和修改等各种操作，操作的实现方式可以是使用 SQL 语句，也可以是使用存储过程。同 Connection 对象一样，Command 对象属于. NET Framework 数据提供程序，不同的数据提供程序（数据源）有各自的 Command 对象，如表 5-6 所示。

表 5-6　. NET 数据提供程序及对应的命令类

数据访问提供程序	名称空间	对应的命令类名称
SQL Server 数据提供程序	System. Data. SqlClient	SqlCommand
OLE DB 数据提供程序	System. Data. OleDb	OledbCommand
ODBC 数据提供程序	System. Data. Odbc	OdbcCommand
Oracle 数据提供程序	System. Data. OracleClient	OracleCommand

5.3.2　Command 对象的常用属性和方法

SqlCommand 对象的常用属性和方法如表 5-7 所示。

表 5-7 SqlCommand 对象的常用属性和方法

属性和方法	说　明
CommandText 属性	获取或设置要对数据源执行的 SQL 命令、存储过程或数据表名称
CommandType 属性	获取或设置命令类型,可取的值:CommandType.Text、CommandType.StoredProduce,分别对应 SQL 命令、存储过程,默认为 Text
Connection 属性	获取或设置 SqlCommand 对象所使用的数据连接属性
Parameters 属性	SQL 命令参数集合
Cancel 方法	取消 SqlCommand 对象的执行
CreateParameter 方法	创建 Parameter 对象
ExecuteNonQuery 方法	执行 CommandText 属性指定的内容,返回数据表被影响的行数。该方法只能执行 Insert、Update 和 Delete 命令
ExecuteReader 方法	执行 CommandText 属性指定的内容,返回 DataReader 对象。该方法用于执行返回多条记录的 Select 命令
ExecuteScalar 方法	执行 CommandText 属性指定的内容,以 object 类型返回结果表第一行第一列的值。该方法一般用来执行查询单值的 Select 命令

5.3.3 创建 Command 对象

Command 对象的构造函数的参数有两个,一个是需要执行的 SQL 语句,另一个是数据库连接对象。这里以它们为参数,调用 SqlCommand 类的构造方法创建 Command 对象,语法格式如下:

```
SqlCommand 命令对象名 new  SqlCommand(SOL 语句,连接对象)
```

用户也可以首先使用构造函数创建一个不含参数的 Command 对象,再设置 Command 对象的 Connection 属性和 CommandText 属性,其语法格式如下:

```
SqlCommand 命令对象名 = new SqlCommand();
命令对象名.Connection = 连接对象;
命令对象名.CommandText = SOL 语句;
```

5.3.4 使用 Command 对象操作数据

使用 Command 对象操作数据,必须有一个 Connection 对象,使用 Command 对象进行数据操作的步骤如下。

(1) 创建数据库连接:按照前面讲过的步骤创建一个 Connection 对象。

(2) 定义执行的 SQL 语句:将对数据库执行的 SQL 语句赋给一个字符串。

(3) 创建 Command 对象:使用已有的 Connection 对象和 SQL 语句字符串创建一个 Command 对象。

(4) 执行 SQL 语句:使用 Command 对象的某个方法执行命令。

1. 使用 Command 对象增加数据库的数据

使用 Command 对象向数据库增加数据的一般步骤如下:首先建立数据库连接;然后

创建 Command 对象，并设置它的 Connection 和 CommandText 属性；最后，使用 Command 对象的 ExecuteNoquery 方法执行数据库增加命令，ExecuteNoquery 方法表示要执行的是没有返回数据的命令。

【示例 5-2】 使用 Command 对象向数据库中添加新数据。

(1) 右击网站项目 WebSite05 新建文件夹 Images，用于存放上传的学生照片。

(2) 在网站项目 WebSite05 中新建文件夹 Ch5_2，并添加 Web 页面 Default.aspx。在 Default.aspx 中添加相应 Web 控件，使其设计外观如图 5-5 所示，页面主体部分代码如下：

```
<form id="form1" runat="server">
<div>
    <table style="width: 320px; height: 240px">
        <tr>
            <td style="width: 100px; text-align: right">学号：</td>
            <td style="width: 220px">
                <asp:TextBox ID="txtStuNo" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td style="width: 100px; text-align: right">姓名：</td>
            <td style="width: 220px">
                <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td style="width: 100px; text-align: right">性别：</td>
            <td style="width: 220px">
                <asp:DropDownList ID="DdSex" runat="server">
                    <asp:ListItem Selected="True">男</asp:ListItem>
                    <asp:ListItem>女</asp:ListItem>
```

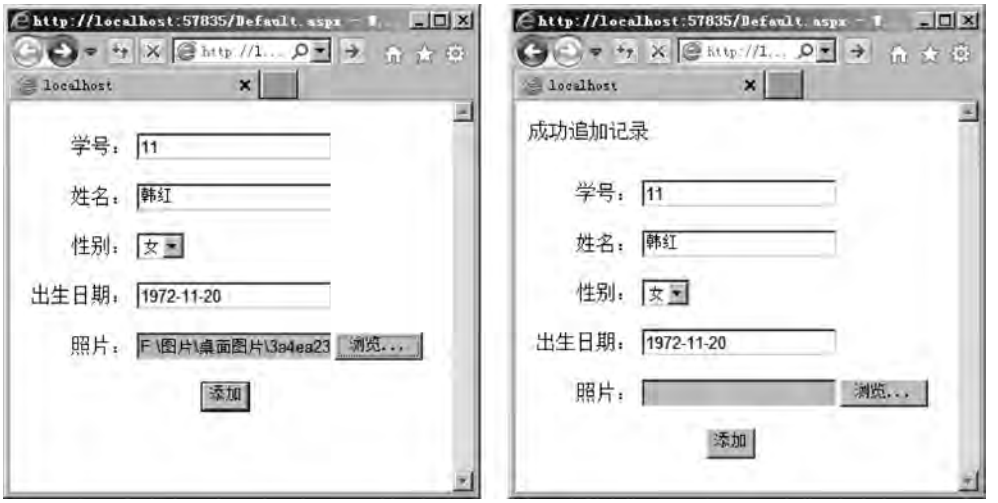

图 5-5 新增学生信息页面设计及运行效果

```
                    </asp:DropDownList>
                </td>
            </tr>
            <tr>
                <td style="width: 100px; text-align: right">出生日期: </td>
                <td style="width: 220px">
                    <asp:TextBox ID="txtBirth" runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td style="width: 100px; text-align: right">照片: </td>
                <td style="width: 220px"><asp:FileUpload ID="FileUpload1" runat="server" />
</td>
            </tr>
            <tr>
                <td colspan="2" style="text-align: center">
                    <asp:Button ID="btnAdd" runat="server" Text="提交" OnClick="btnAdd_
Click" />
                </td>
            </tr>
        </table>
    </div>
    </form>
```

(3) 在 Default.aspx.cs 文件的所有事件之外定义数据库连接字符串和连接对象,代码如下:

```
static string ConStr = "Server=.;Database=Student;Uid=sa;pwd=123456";
SqlConnection conn = new SqlConnection(ConStr);
```

(4) 编写 Default.aspx 页面中“添加”按钮的 Click 事件过程代码,如下:

```
protected void btnAdd_Click(object sender, EventArgs e)
{
    SqlCommand cmd = new SqlCommand(); //建立 Command 对象
    cmd.Connection = conn;
    //把 SQL 语句赋给 Command 对象
    cmd.CommandText = "insert into StuInfo(StuNo,Name,Sex,Birth,Photo) values (@StuNo,@
Name,@Sex,@Birth,@Photo)";
    //在执行之前告诉 Command 对象@StuNo、@Name、@Sex、@Birth、@Photo 将来用谁来代替,即给
    //参数赋值
    SqlParameter[] paras = new SqlParameter[] {
                new SqlParameter("@StuNo", txtStuNo.Text),
                new SqlParameter("@Name", txtName.Text),
                new SqlParameter("@Sex", DdSex.Text),
                new SqlParameter("@Birth", txtBirth.Text),
                new SqlParameter("@Photo", txtStuNo.Text + ".jpg")
                };
```

```
        cmd.Parameters.AddRange(paras);
        try
        {

            conn.Open(); //打开连接

            cmd.ExecuteNonQuery(); //执行 SQL 命令

            if (FileUpload1.HasFile == true) //把学生的照片上传到网站的"images"文件夹中,
                                             //以学号为名字进行保存
            {
                string fileName = this.txtStuNo.Text + ".jpg";
                FileUpload1.SaveAs(Server.MapPath(("images/") + fileName));
            }
            Response.Write("成功追加记录");
        }
        catch (Exception ex)
        {
            Response.Write("错误原因: " + ex.Message);
        }
        finally
        {
            cmd = null;
            conn.Close();
            conn = null;
        }
    }
```

(5) 程序运行效果如图 5-5 所示。

2. 使用 Command 对象删除数据库的数据

使用 Command 对象删除数据的一般步骤如下：首先建立数据库连接；然后创建 Command 对象，并设置它的 Connection 和 CommandText 属性，即使用 Command 对象的 Parameters 属性来设置输入参数；最后，使用 Command 对象的 ExecuteNoquery 方法执行数据删除命令。

【示例 5-3】 使用 Command 对象删除数据。

(1) 在网站项目 WebSite05 中新建文件夹 Ch5_3，并添加 Web 页面 CommDeleteDemo.aspx。在 CommDeleteDemo.aspx 中添加一个 TextBox 控件和一个 Button 控件，其中 Button 控件作为“删除”按钮，页面主体部分代码如下：

```
<form id="form1" runat="server">
<div>
    输入要删除学生的学号:<br />
   <asp:TextBox ID="txtStuNo" runat="server"></asp:TextBox>
   <asp:Button ID="btnDel" runat="server" Text="删除" OnClick="btnDel_Click" />
</div>
</form>
```

(2) 编写 CommDeleteDemo.aspx 页面中“删除”按钮的 Click 事件过程代码,如下:

```
protected void btnDel_Click(object sender, EventArgs e)
{
    int intDeleteCount;
    string ConStr = "Server=.; Database=Student; Uid=sa; pwd=123456";
    SqlConnection sqlconn = new SqlConnection(ConStr);
    //建立 Command 对象
    SqlCommand cmd = new SqlCommand();
    //给 Command 对象的 Connection 和 CommandText 属性赋值
    cmd.Connection = sqlconn;
    cmd.CommandText = "delete from StuInfo where StuNo=@no";
    //在执行之前告诉 Command 对象@no 将来用谁来代替,注意与【示例 5-2】的区别
    SqlParameter p1 = new SqlParameter("@No", txtStuNo.Text);
    cmd.Parameters.Add(p1);
    try
    {
        sqlconn.Open();
        intDeleteCount = cmd.ExecuteNonQuery();
        if (intDeleteCount > 0)
            Response.Write("删除成功!");
        else
            Response.Write("该记录不存在!");
    }
    catch (Exception ex)
    {
        Response.Write("删除失败,错误原因: " + ex.Message);
    }
    finally
    {
        cmd = null;
        sqlconn.Close();
        sqlconn = null;
    }
}
```

(3) 程序运行效果如图 5-6 所示。

图 5-6 页面 CommDeleteDemo.aspx 运行效果

3. 使用 Command 对象修改数据库的数据

使用 Command 对象修改数据库的数据和向数据库中添加数据的操作类似,在此不再举例,请读者自行完成。

任务 5-1　实现“新知书店”用户注册功能

【任务描述】

在任务 4-4 的基础上，新建“新知书店”用户信息注册 Web 页面 Register. aspx，实现用户注册功能，注册成功后，弹出对话框，单击“确定”按钮时跳转到“新知书店”首页 Default. aspx。注册页面运行效果如图 5-7 所示。

图 5-7　“新知书店”注册页面运行效果

【任务实施】

(1) 在文件夹 rw5-1 下创建网站项目 Web，解决方案名称为 BookShop，将任务 4-4 的站点目录下的文件及文件夹复制至新创建的网站项目 Web 下。

(2) 右击网站项目 Web，添加名为 Register. aspx 的用户注册 Web 页。在 Register. aspx 页面中引入外部样式文件，添加必要的 HTML 代码，实现 Web 页面的布局结构。在 Register. aspx 页中添加相应 Web 控件，使其设计外观如图 5-7 所示。

(3) 在 Register. aspx. cs 文件中，编写方法 IsExists，用于判断用户数据表“Users”中是否存在相同的注册信息，代码如下：

```
static bool IsExists(string txtLoginId, string txtEmail)
{
```

```
        string strSqlConn = "Server = .; Database = BookShopPlus; Uid = sa; pwd = 123456";
        string strSql = "Select * From Users Where LoginId = '" + txtLoginId + "' Or Mail = '" +
    txtEmail + "'";
        SqlConnection sqlConn = new SqlConnection();              //创建连接对象
        sqlConn.ConnectionString = strSqlConn;                   //给连接字符串赋值
        sqlConn.Open();                                          //打开数据库连接
        SqlCommand sqlComm = new SqlCommand(strSql, sqlConn);    //创建命令对象
        SqlDataReader sdr = sqlComm.ExecuteReader();             //读取数据表中的数据
        //使用 SqlDataReader 对象的 Read()方法判断是否存在记录,如果存在则返回 True
        if (sdr.Read())
        {
            sdr.Close();
            return true;
        }
        else
        {
            sdr.Close();
            return false;
        }
    }
```

(4)“确定了,马上提交”按钮的 Click 事件过程代码实现新增用户注册信息,编写 Register.aspx 页面中“确定了,马上提交”按钮的 btnSubmit_Click 事件过程代码,如下:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    string ConStr = "Server = .; Database = BookShopPlus; Uid = sa; pwd = 123456";
    SqlConnection conn = new SqlConnection(ConStr);
    SqlCommand cmd = new SqlCommand(); //建立 Command 对象
    if (!CeckCode())
    {
        Page.RegisterClientScriptBlock("alert", "< script > alert('验证码错误!')</script >");
        return;
    }
    try
    {
        cmd.Connection = conn;
        //把 SQL 语句赋给 Command 对象
         cmd.CommandText = " INSERT Users(LoginId, LoginPwd, Name, Address, Phone, Mail)
VALUES (@LoginId, @LoginPwd, @Name, @Address, @Phone, @Mail)";
        //在执行之前告诉 Command 对象@StuNo、@Name、@Sex、@Birth、@Photo 将来用谁来代替,
即给参数赋值
        SqlParameter[] para = new SqlParameter[]{
               new SqlParameter("@LoginId", txtLoginId.Text),
               new SqlParameter("@LoginPwd", txtLoginPwd.Text),
               new SqlParameter("@Name", txtName.Text),
               new SqlParameter("@Address", txtAddress.Text),
```

```
                new SqlParameter("@Phone", txtTele.Text),
                new SqlParameter("@Mail", txtEmail.Text)
            };
            cmd.Parameters.AddRange(para);
            //调用 IsExists()自定义方法判断数据表中是否存在相同的注册信息
            if (!IsExists(txtLoginId.Text, txtEmail.Text))
            {
                conn.Open(); //打开连接
                cmd.ExecuteNonQuery(); //执行 SQL 命令
                Page.RegisterClientScriptBlock("alert", "<script>alert('注册成功,请登录!');
window.location='../Default.aspx'</script>");
            }
            else
            {
                Page.RegisterClientScriptBlock("alert", "<script>alert('用户名已使用,请重新
输入!')</script>");
            }
        }
        catch (Exception ex)
        {
            Response.Write("错误原因: " + ex.Message);
        }
        finally
        {
            cmd = null;
            conn.Close();
            conn = null;
        }
    }
```

(5) 页面运行效果如图 5-7 所示。

5.4 DataReader 数据读取对象

5.4.1 DataReader 对象概述

当 Command 对象返回结果集时需要使用 DataReader 对象来检索数据。DataReader 对象返回一个来自 Command 的只读的、只能向前的数据集。DataReader 每次只能在内存中保留一行,所以开销非常小,提高了应用程序的性能。

由于 DataReader 只执行读操作,并且每次只在内存缓冲区里存储结果集中的一条数据,所以使用 DataReader 对象的效率比较高,如果要查询大量数据,同时不需要随机访问和修改数据,DataReader 是优先的选择。

DataReader 属于. NET 数据提供程序,每一种. NET 数据提供程序都有与之对应的 DataReader 类,如表 5-8 所示。

表 5-8 .NET 数据提供程序及对应的 DataReader 类

数据访问提供程序	名称空间	对应的 DataReader 类名称
SQL Server 数据提供程序	System.Data.SqlClient	SqlDataReader
OLEDB 数据提供程序	System.Data.OleDb	OledbDataReader
ODBC 数据提供程序	System.Data.Odbc	OdbcDataReader
Oracle 数据提供程序	System.Data.OracleClient	OracleDataReader

5.4.2 DataReader 对象的常用属性和方法

SqlDataReader 对象的常用属性和方法如表 5-9 所示。

表 5-9 SqlDataReader 对象的常用属性和方法

属性和方法	说　明
FieldCount 属性	获取由 DataReader 得到的一行数据中的字段数
isClosed 属性	获取 SqlDataReader 对象的状态,true 表示关闭,false 表示打开
HasRows 属性	表示查询是否返回结果。如果有查询结果,返回 true,否则返回 false
HasMoreRows 属性	只读,表示是否还有记录未读取
Close 方法	不带参数,无返回值,用来关闭 DataReader 对象
Read 方法	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
NextResult 方法	当返回多个结果集时,使用该方法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 方法来遍历访问该结果集
GetValue 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
GetValues 方法	该方法会把当前记录行里所有的数据保存到一个数组里。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
GetName 方法	通过输入列索引,获得该列的名称。综合使用 GetName 和 GetValue 两个方法,可以获得数据表里列名和列的字段
IsDBNull 方法	判断指定索引号的列的值是否为空,返回 true 或 false

5.4.3 创建 DataReader 对象

DataReader 对象不能直接实例化,而必须调用 Command 对象的 ExecuteReader 方法才能创建有效的 DataReader 对象。通过调用 Command 对象的 ExecuteReader 方法得到的结果集是一个 DataReader 对象,语法格式如下,可以调用 DataReader 对象的 Read 方法读取一行记录。

```
SqlDataReader 数据读取器对象名 new  命令对象名.ExecuteReader();
```

5.4.4 使用 DataReader 对象检索数据

使用 DataReader 对象读取数据,首先要使用其 HasRows 属性判断是否有数据可供读取,如果有数据,返回 true,否则返回 false; 然后再使用 DataReader 对象的 Read 方法来循环读取结果集中的数据; 最后通过访问 DataReader 对象的列索引来获取读取到的值,例如

dr["Id"]用来获取数据表中 Id 列的值。使用 SqlDataReader 对象检索数据的步骤如下。

(1) 创建 SqlConnection 对象,设置连接字符串。

(2) 创建 SqlCommand 对象,设置它的 Connection 和 CommandText 属性,分别表示数据库连接和需要执行的 SQL 命令。

(3) 打开与数据库的连接。

(4) 使用 SqlCommand 对象的 ExecuteReader 方法执行 CommandText 中的命令,并把返回的结果放在 SqlDataReader 对象中。假设已创建一个名为 cmd 的 Command 对象,下面的代码可以创建一个 DataReader 对象。

```
SqlDataReader dr = cmd.ExecuteReader();
```

(5) 通过调用 SqlDataReade 对象的 Read 方法循环读取查询结果集的记录。这个方法返回一个布尔值。如果能读到一行记录,返回 true,否则返回 false。代码如下:

```
dr.Read();
```

(6) 读取当前行的某列的数据。可以像使用数组一样,用方括号来读取某列的值,如(type)dr[ ],方括号中可以是列的索引(从 0 开始),也可以是列名。读取到的列值必须要进行类型转换,如下:

```
(string)dr["name"];
```

(7) 关闭与数据库的连接。

【示例 5-4】 使用 DataReader 对象读取数据库中的数据。

(1) 在网站项目 WebSite05 中新建文件夹 Ch5_4,在文件夹 Ch5_4 中添加一个名为 DataReaderDemo.aspx 的 Web 页。

(2) 在 DataReaderDemo.aspx.cs 文件的所有事件之外定义数据库连接字符串和连接对象,代码如下:

```
static string ConStr = " Server = .; Database = Student; Uid = sa; pwd = 123456";
SqlConnection conn = new SqlConnection(ConStr);
```

(3) 编写 DataReaderDemo.aspx 页面的 Page_Load 事件过程代码,如下:

```
protected void Page_Load(object sender, EventArgs e)
{
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = conn;
    cmd.CommandText = "select * from StuInfo";
    SqlDataReader dr = null;                    //创建 DataReader 对象的引用
    try
    {
        if (conn.State == ConnectionState.Closed)
            conn.Open();
```

```
            //执行 SQL 命令,并获取查询结果
            dr = cmd.ExecuteReader();
            //依次读取查询结果的字段名称,并以表格的形式显示
            Response.Write("<table border='1'><tr align='center'>");
            for (int i = 0; i < dr.FieldCount; i++)
            {
                Response.Write("<td>" + dr.GetName(i) + "</td>");
            }
            Response.Write("</tr>");
            //如果 DataRead 对象成功获得数据,返回 true,否则返回 false
            while (dr.Read())
            {
                //依次读取查询结果的字段值,并以表格的形式显示
                Response.Write("<tr>");
                for (int j = 0; j < dr.FieldCount; j++)
                {
                    Response.Write("<td>" + dr.GetValue(j) + "</td>");
                }
                Response.Write("</tr>");
            }
            Response.Write("</table>");
        }
        catch (Exception ex)
        {
            Response.Write("SqlDataReader 读取出错,原因: " + ex.Message);
        }
        finally
        {
            if (dr.IsClosed == false)
                dr.Close();                    //关闭 DataReader 对象
            if (conn.State == ConnectionState.Open)
                conn.Close();
        }
    }
```

(4) 页面运行效果如图 5-8 所示。

StuNo	Name	Sex	Birth	Photo	MajorId
1	刘备	男	1950/3/2 0:00:00	1.jpg	2
3	王五	男	1989/3/4 0:00:00	3.jpg	2
4	陈昊	男	1990/3/25 0:00:00	4.jpg	2
6	李勇	男	1988/4/6 0:00:00	6.jpg	3
7	王燕	女	1990/5/12 0:00:00	7.jpg	4
8	赵倩	女	1989/12/23 0:00:00	8.jpg	4
9	李兰	女	1990/1/20 0:00:00	9.jpg	4

图 5-8　页面 DataReaderDemo.aspx 运行效果

使用 SqlDataReader 对象时，应注意以下几点。

（1）读取数据时，SqlConnection 对象必须处于打开状态。

（2）必须通过调用 SqlCommand 对象的 ExecuteReader 方法产生 SqlDataReader 对象的实例。

（3）只能按向下的顺序逐条读取记录，不能随机读取，且无法直接获知读取记录的总数。

（4）SqlDataReader 对象管理的查询结果是只读的，不能修改。

任务 5-2　实现“新知书店”用户登录功能

【任务描述】

在任务 4-4 的基础上，创建名为 UserLogin. aspx 的用户登录 Web 页，该页面实现“新知书店”用户登录功能，其浏览效果如图 5-9 所示。如果用户名为空，则提示“请输入用户名！”，如果密码为空，则提示“请输入密码”。用户名和密码均输入，且与数据库查询验证通过后，则登录成功，自动弹出“登录成功”的提示信息对话框，并跳转到后台首页，在首页顶部显示登录的用户名，如图 5-10 所示，否则给出“登录失败”的提示信息。

图 5-9　“新知书店”用户登录页面

图 5-10　登录成功后的“新知书店”首页顶部

【任务实施】

(1) 在文件夹 rw5-2 下创建网站项目 Web,解决方案名称为 BookShop,将任务 4-4 的站点目录下的文件及文件夹复制至新创建的网站项目 Web 下。

(2) 右击网站项目 Web,基于母版页 common.master 添加名为 UserLogin.aspx 的用户登录 Web 页。在 UserLogin.aspx 页面中引入外部样式文件,添加必要的 HTML 代码,实现 Web 页面的布局结构。在 UserLogin.aspx 页中添加相应 Web 控件,使其设计外观如图 5-9 所示,主体部分代码如下:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/common.master" AutoEventWireup="
true" CodeFile="UserLogin.aspx.cs" Inherits="UserLogin" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphHeader" runat="Server">
<link href="Css/member.css" rel="stylesheet" type="text/css" />
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphContent" runat="Server">
<script type="text/javascript">
    function ValidateForm() {
        var txtLoginId = document.getElementById('<%= txtUserName.ClientID %>');
        var txtLoginPwd = document.getElementById('<%= txtPassword.ClientID %>');
        if (txtLoginId.value == "") {
            alert('请输入用户名!');
            return false;
        }
        else if (txtLoginPwd.value == "") {
            alert("请输入密码!");
            return false;
        }
        return true;
    }
    document.forms[0].onsubmit = function () {
        if (ValidateForm() == false) {
            return false;
        }
        else {
            document.forms[0].submit();
        }
    }
</script>
<div id="action_area" class="member_form">
    <h2 class="action_type">
        <img src="Images/login_in.gif" alt="会员登录" /></h2>
    <p class="state">
        欢迎光临新知书店网站!<br />
        您可以使用新知书店的用户名,直接登录.</p>
    <p>
        <label>
          用户名</label><asp:TextBox ID="txtUserName" runat="server" CssClass="opt_
input"></asp:TextBox></p>
    <p>
```

```
        <label>
          密     码</label><asp:TextBox ID="txtPassword" runat
="server" TextMode="Password" CssClass="opt_input"></asp:TextBox></p>
    <p class="form_sub">
        <input type="checkbox" name="" checked="checked" />
        在此计算机上保留我的密码</p>
    <p class="form_sub">
         <asp:Button runat="server" ID="btnLogin" CssClass="opt_sub" Text="登录"
TabIndex="1"
            OnClick="btnLogin_Click" />
        <a href="Register.aspx">还没有注册??</a></p>
</div>
</asp:Content>
```

(3)“登录”按钮的 Click 事件过程代码实现用户登录,编写 UserLogin.aspx 页面中“登录”按钮的 btnLogin_Click 事件过程代码,如下:

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    //声明一个连接数据库字符串
    string strSqlConn = "Server=.; Database=BookShopPlus; Uid=sa; pwd=123456";
    string strSqlComm = "Select * From Users Where LoginId='" + txtUserName.Text.Trim()
+"'And LoginPwd='" + txtPassword.Text.Trim() + "'" + "And UserRoleId=3";
    SqlConnection sqlConn = new SqlConnection(strSqlConn);
    sqlConn.Open();       //打开连接
    //建立 SqlCommand 命令对象 1
    SqlCommand sqlComm = new SqlCommand(strSqlComm, sqlConn);
    //使用 DataReader 对象执行命令
    SqlDataReader sdr = sqlComm.ExecuteReader();
    //读取查询数据
    if (sdr.Read())
    {
        Response.Write("<script>alert('登录成功!')</script>");
        Session["LoginUserName"] = txtUserName.Text.Trim();
        Session["LoginPassword"] = txtPassword.Text.Trim();
        Response.Redirect("Default.aspx?name=" + this.txtUserName.Text.Trim());
        Session["LoginTime"] = DateTime.Now.ToString();
    }
    else
    {
        Response.Write("<script>alert('登录失败!')</script>");
    }
    sdr.Close();
    sqlConn.Close();
}
```

(4) 页面运行效果如图 5-9 和图 5-10 所示。

说明:由于显示用户信息的首页顶部是在母版页中设计完成的,在 UserLogin.aspx 页面中登录后,用户登录信息被传递给网站首页 Default.aspx,因此,要在 Default.aspx 页中

添加<%@ MasterType VirtualPath ="common.master" %>标记,以便母版页获取传递给 Default.aspx 页的用户信息,具体参看 4.3.4 节【示例 4-4】。

5.5 DataSet 对象和 DataAdapter 对象

5.5.1 DataSet 对象

1. DataSet 对象概述

DataSet 对象是 ADO.NET 的核心组件之一。ADO.NET 从数据库抽取数据后数据就存放在 DataSet 中,故可以把 DataSet 看成一个数据容器,或称为"内存中的数据库"。

DataSet 从数据源中获取数据后就断开了与数据源之间的连接。用户可以在 DataSet 中对记录进行插入、删除、修改、查询、统计等,在完成了各项操作后还可以把 DataSet 中的数据送回数据源。

每一个 DataSet 都是一个或多个 DataTable 对象的集合,DataTable 相当于数据库中的表。DataTable 对象的常用属性主要有 Columns 属性、Rows 属性和 Default View 属性。

(1) Columns 属性:用于获取 DataTable 对象中表的列集合。

(2) Rows 属性:用于获取 DataTable 对象中表的行集合。

(3) Default View 属性:用于获取表的自定义视图。

2. DataSet 对象结构模型

DataSet 数据集对象的结构和我们熟悉的 SQL Server 非常相似,如图 5-11 所示。在 SQL Server 数据库中,有很多数据表,每个数据表都有行和列。DataSet 数据集中也包含多个表,这些表构成了一个数据表集合(DataTableCollection),其中每个数据表都是一个 DataTable 对象。在每个数据表中又有列和行,所有的列构成了数据列集合(DataColumnCollection),其中每个数据列是一个 DataColumn 对象;所有的行构成了数据行集合(DataRowCollection),每一行是一个 DataRow 对象。此外,在 DataSet 中还可以定义表之间的链接、视图等。

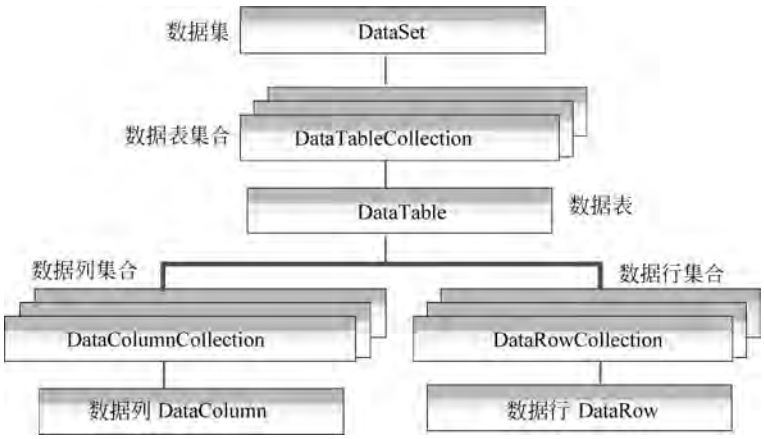

图 5-11 DataSet 数据集对象基本结构

注意：各个数据表 DataTable 之间的关系通过 DataRelation 来表示，这些 DataRelation 构成的集合就是 DataRelationColleciton 对象。

3. DataSet 对象工作原理

DataSet 数据集对象工作原理如图 5-12 所示。

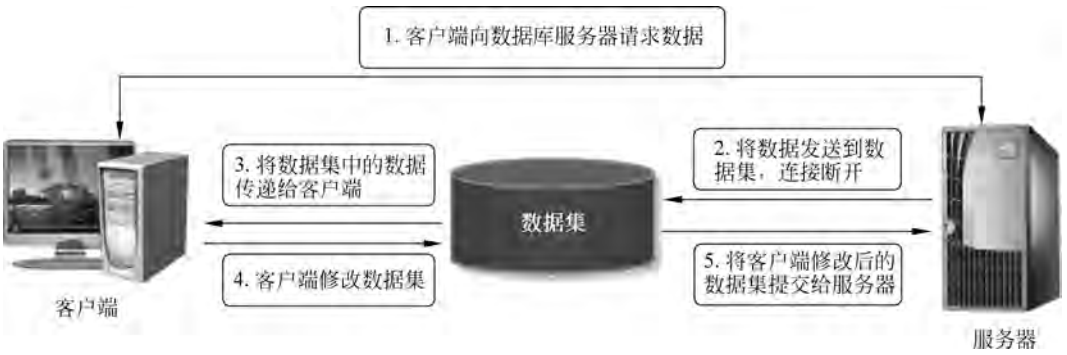

图 5-12　DataSet 数据集对象工作原理

当应用程序需要获取一些数据时，先向数据库服务器发出请求，要求获得数据。服务器将数据发送到 DataSet 数据集，然后再将数据集传递给客户端。客户端应用程序修改数据集中的数据后，统一将修改过的数据集发送到服务器，服务器接收数据集修改数据库中的数据。

4. 创建 DataSet（数据集）对象

创建 DataSet 的语法格式为：

```
DataSet 对象名 = new DataSet();
```

或

```
DataSet 对象名 = new DataSet("数据集名");
```

例如，创建数据集对象 dsStu，代码如下：

```
DataSet dsStu = new DataSet();
```

或

```
DataSet dsStu = new DataSet("Student");
```

注意：方法中的参数是数据集的名称字符串，可以有，也可以没有。如果没有写参数，创建的数据集名称就默认为 NewDataSet。

DataSet 对象的常用属性和方法如表 5-10 所示。

表 5-10　DataSet 对象的常用属性和方法

属性和方法	说　明
DataSetName 属性	获取或设置 DataSet 对象的名称
Tables 属性	获取包含在 DataSet 数据集中的数据表的集合
Clear 方法	删除 DataSet 对象中所有表
Copy 方法	复制 DataSet 的结构和数据,返回与本 DataSet 对象具有相同结构和数据的 DataSet 对象

5. 创建 DataTable(数据表)对象

DataSet 中的每个数据表都是一个 DataTable 对象。定义 DataTable 对象的语法格式为:

```
DataTable 对象名 = new DataTable();
```

或

```
DataTable 对象名 = new DataTable("数据表名");
```

例如,创建数据表对象 dtStuInfo,代码如下:

```
DataTable dtStuInfo = new DataTable();
dtStuInfo.TableName = "StuInfo";
```

或

```
DataTable dtStuInfo = new DataTable("StuInfo");
```

创建好的数据表对象可以添加到数据集对象中,例如把创建好的 dtStuInfo 数据表对象添加到数据集对象 dsStu 中,代码如下:

```
dsStu.Tables.Add(dtStuInfo);
```

DataTable 对象的常用属性和方法如表 5-11 所示。

表 5-11　DataTable 对象的常用属性和方法

属性和方法	说　明
Columns 属性	获取数据表的所有字段
DataSet 属性	获取 DataTable 对象所属的 DataSet 对象
DefaultView 属性	获取与数据表相关的 DataView 对象
PrimaryKey 属性	获取或设置数据表的主键
Rows 属性	获取数据表的所有行
TableName 属性	获取或设置数据表名
Clear()方法	清除表中所有的数据
NewRow()方法	创建一个与当前数据表有相同字段结构的数据行

6. 创建 DataRow(数据行)对象

DataTable 对象可以包含多个数据行,每行就是一个 DataRow 对象。定义 DataRow 对象的语法格式为:

```
DataRow 对象名 = DataTable 对象.NewRow();
```

注意:DataRow 对象不能用 New 来创建,而需要用数据表对象的 NewRow 方法创建。

例如,为数据表对象 dtStuInfo 添加一个新的数据行,代码如下:

```
DataRow dr = dtStuInfo.NewRow();
```

访问一行中某个单元格内容的方法为:

```
DataRow 对象名["字段名"]或 DataRow 对象名[序号]
```

DataRow 对象的常用属性和方法如表 5-12 所示。

表 5-12 DataRow 对象的常用属性和方法

属性和方法	说 明
RowState 属性	获取数据行的当前状态,属于 DataRowState 枚举型,分别为:Add、Delete、Detached、Modified、Unchanged
BeginEdit 方法	开始数据行的编辑
CancelEdit 方法	取消数据行的编辑
Delete 方法	删除数据行
EndEdit 方法	结束数据行的编辑

7. 创建 DataColumn(数据列)对象

DataTable 对象中包含多个数据列,每列就是一个 DataColumn 对象。定义 DataColumn 对象的语法格式为:

```
DataColumn 对象名 = new DataColumn();
```

或

```
DataColumn 对象名 = new DataColumn("字段名");
```

或

```
DataColumn 对象名 = new DataColumn("字段名",数据类型);
```

例如,创建数据列对象 stuNoColumn,代码如下:

```
DataColumn stuNoColumn = new DataColumn();
stuNoColumn.ColumnName = " StuNo";
stuNoColumn.DataType = System.Type.GetType("System.String");
```

或

```
DataColumn stuNoColumn = new DataColumn("StuNo", System.Type.GetType("System.String"));
```

创建好的数据列对象可以添加到数据表对象中,例如将创建的数据列对象 stuNoColumn 添加到刚才创建的数据表对象 dtStuInfo 中,代码如下:

```
dtStuInfo.Columns.Add(stuNoColumn);
```

DataColumn 对象的常用属性如表 5-13 所示。

表 5-13 DataColumn 对象的常用属性

属 性	说 明
AllowDBNull	设置该字段可否为空值,默认为 true
Caption	获取或设置字段标题。如果为指定字段标题,则字段标题与字段名相同
ColumnName	获取或设置字段名
DataType	获取或设置字段的数据类型
DefaultValue	获取或设置新增数据行时,字段的默认值

说明:通过 DataColumn 对象的 DataType 属性设置字段数据类型时,不可直接设置数据类型,而要按照以下语法格式:

```
对象名.DataType = System.Type.GetType("数据类型");
```

5.5.2 DataAdapter 对象

1. DataAdapter 对象概述

DataAdapter(即数据适配器)对象是一种用来充当 DataSet 对象与实际数据源之间桥梁的对象。DataSet 对象是一个非连接的对象,它与数据源无关。DataAdapter 正好负责填充它,并把它的数据提交给一个特定的数据源。DataAdapter 与 DataSet 配合使用可以执行新增、查询、修改和删除等多种操作。

DataAdapter 对象是一个双向通道,用来把数据从数据源中读到一个内存表中,以及把内存中的数据写回到一个数据源中。这两种情况下使用的数据源可能相同,也可能不相同。这两种操作分别称为填充(Fill)和更新(Update)。

DataAdapter 属于.NET 数据提供程序,每一种.NET 数据提供程序都有与之对应的 DataAdapter 类,如表 5-14 所示。

表 5-14 .NET 数据提供程序及对应的 DataAdapter 类

数据访问提供程序	名称空间	对应的 DataAdapter 类名称
SQL Server 数据提供程序	System.Data.SqlClient	SqlDataAdapter
OLE DB 数据提供程序	System.Data.OleDb	OledbDataAdapter
ODBC 数据提供程序	System.Data.Odbc	OdbcDataAdapter
Oracle 数据提供程序	System.Data.OracleClient	OracleDataAdapter

2. DataAdapter 对象的主要属性和方法

1) DataAdapter 对象的主要属性

(1) SelectCommand 属性：获取或设置一个语句或存储过程，用于在数据库中选择记录。

(2) InsertCommand 属性：获取或设置一个语句或存储过程，用于在数据库中插入记录。

(3) UpdateCommand 属性：获取或设置一个语句或存储过程，用于更新数据库中的记录。

(4) DeleteCommand 属性：获取或设置一个语句或存储过程，用于从 DataSet 数据集中删除记录。

2) DataAdapter 对象的主要方法

(1) Fill 方法：调用 Fill 方法会自动执行 SelectCommand 属性中提供的命令，获取结果集并填充数据集的 DataTable 对象。其本质是通过执行 SelectCommand 对象的 Select 语句查询数据库，返回 DataReader 对象，通过 DataReader 对象隐式地创建 DataSet 中的表，并填充 DataSet 中表行的数据。

(2) Update 方法：调用 InsertCommand、UpdateCommand 和 DeleteCommand 属性指定的 SQL 命令，将 DataSet 对象更新到相应的数据源。在 Update 方法中，逐行检查数据表每行的 RowState 属性值，根据不同的 RowState 属性，调用不同的 Command 命令更新数据库。

3. 创建 DataAdapter 对象

DataAdapter 对象构造函数的参数有两个，一个是需要执行的 SQL 语句，另一个是数据库连接对象。这里以它们为参数，调用 SqlDataAdapter 类的构造方法创建 DataAdapter 对象，其语法格式如：

```
SqIDataAdapter 数据适配器对象名 = new SqlDataAdapter(SOL 语句,连接对象);
```

用户也可以首先使用构造函数创建一个不含参数的 DataAdapter 对象，再设置 DataAdapter 对象的 Connection 属性和 CommandText 属性，其语法格式如下：

```
SqlDataAdapter 数据适配器对象名 = new SqlDataAdapter();
数据适配器对象名.Connection = 连接对象;
数据适配器对象名.CommandText = SQL 语句;
```

4. DataSet 和 DataAdapter 对象应用

使用 SqlDataAdapter 和 DataSet 对象操作数据库的步骤如下。

(1) 创建数据库连接对象。

(2) 利用数据库连接对象和 Select 语句创建 SqlDataAdapter 对象。

(3) 根据操作要求配置 SqlDataAdapter 对象中不同的 Command 属性。例如增加数据库数据，需要配置 InsertCommand 属性；修改数据库数据，需要配置 UpdateCommand 属性；删除数据库数据，需要配置 DeleteCommand 属性。

(4) 使用 SqlDataAdapter 对象的 Fill 方法把 Select 语句的查询结果放在 DataSet 对象

的一个数据表中或直接放在一个 DataTable 对象中。

(5) 对 DataTable 对象中的数据进行增加、删除、修改操作。

(6) 修改完成后,通过 SqlDataAdapter 对象的 Update 方法将 DataTable 对象中的修改更新到数据库。

说明:第(3)步中根据操作要求配置 SqlDataAdapter 对象中不同的 Command 属性,如果自己给 SqlDataAdapter 对象的 InsertCommand、UpdateCommand、DeleteCommand 属性定义 SQL 更新语句,过程比较复杂。可以通过建立 CommandBuilder 对象以便自动生成 DataAdapter 的 Command 命令。

1) 查询数据库的数据

使用 DataSet 和 DataAdapter 对象查询数据库数据的一般步骤如下:首先建立数据库连接;然后利用数据连接和 Select 语句建立 DataAdapter 对象,并使用 DataAdapter 对象的 Fill 方法把查询结果放在 DataSet 对象的一个数据表中;接下来,将该数据表复制到 DataTable 对象中;最后,实施对 DataTable 对象中数据的查询。

【示例 5-5】 使用 DataSet 和 DataAdapter 对象查询数据库的数据。

(1) 在网站项目 WebSite05 中新建文件夹 Ch5_5,在文件夹 Ch5_5 中添加一个名为 DataAdapterSelectDemo.aspx 的 Web 页。

(2) 编写 DataAdapterSelectDemo.aspx 页面的 Page_Load 事件过程代码,如下:

```
protected void Page_Load(object sender, EventArgs e)
{
    string ConStr = " Server = .; Database = Student; Uid = sa; pwd = 123456";
    SqlConnection sqlconn = new SqlConnection(ConStr);
    DataSet ds = new DataSet();          //建立 DataSet 对象
    DataRow dr;                          //建立 DataRow 数据行对象
    try
    {
        sqlconn.Open();                  //打开连接
        SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", sqlconn);
                                                                  //建立 DataAdapter 对象
        sda.Fill(ds, "StuTable");        //用 Fill 方法返回的数据,填充 DataSet,数据表取名为
                                         //"StuTable"
        DataTable dtable = ds.Tables["StuTable"]; //将数据表 StuTable 的数据复制到
                                                  //DataTable 对象
        DataRowCollection drc = dtable.Rows;   //用 DataRowCollection 对象获取 StuTable 数
                                               //据表的所有数据行
        for (int i = 0; i < drc.Count; i++)//逐行遍历,取出各行的数据
        {
            dr = drc[i];
            Response.Write("学号: " + dr["StuNo"] + " 姓名: " + dr["Name"] + " 性别: " +
dr["Sex"] + " 出生日期: " + dr["Birth"] + " 照片: " + dr["Photo"]);
            Response.Write("< br />");
        }
    }
    catch (Exception ex)
    {
```

```
            Response.Write("数据读取出错!原因: " + ex.Message);
        }
        finally
        {
            sqlconn.Close();
            sqlconn = null;
        }
    }
```

(3) 程序运行效果如图 5-13 所示。

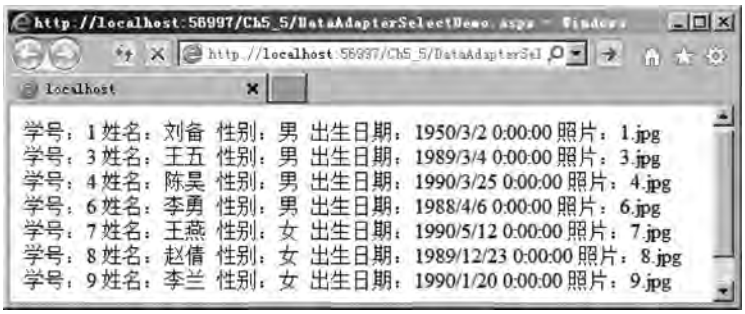

图 5-13　页面 DataAdapterSelectDemo.aspx 运行效果

在后续章节介绍完绑定控件 GridView 后，显示 DataSet 中的数据更加简单。

2) 新增数据库的数据

使用 DataSet 和 DataAdapter 对象增加数据库数据的一般步骤如下：首先建立数据库连接；然后利用数据连接和 Select 语句建立 DataAdapter 对象，并建立 CommandBuilder 对象以便自动生成 DataAdapter 的 Command 命令，否则，就要自己给 InsertCommand、UpdateCommand、DeleteCommand 属性定义 SQL 更新语句；使用 DataAdapter 对象的 Fill 方法把 Select 查询语句结果放在 DataSet 对象的一个数据表中；接下来，将该数据表复制到 DataTable 对象中；最后，向 DataTable 对象增加数据记录，并通过 DataAdapter 对象的 Update 方法向数据库提交数据。

【示例 5-6】 使用 DataSet 和 DataAdapter 对象向数据库增加一条学生记录。

(1) 在网站项目 WebSite05 中新建文件夹 Ch5_6，在文件夹 Ch5_6 中添加一个名为 DataAdapterInsertDemo.aspx 的 Web 页。

(2) 编写 DataAdapterInsertDemo.aspx 页面中“提交”按钮的 Click 事件过程代码，如下：

```
protected void btnAdd_Click(object sender, EventArgs e)
{
    string ConStr = " Server = .; Database = Student; Uid = sa; pwd = 123456";
    SqlConnection sqlconn = new SqlConnection(ConStr);
    DataSet ds = new DataSet();                              //建立 DataSet 对象
    try
    {
```

```
        sqlconn.Open();                                          //打开连接
        //建立 DataAdapter 对象
        SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", sqlconn);

        //设置 DataAdapter 的 InsertCommand 命令
         SqlCommand command = new SqlCommand("insert into StuInfo(StuNo,Name,Sex,Birth,
Photo)values (@StuNo,@Name,@Sex,@Birth,@Photo)", sqlconn);
        // Add the parameters for the InsertCommand.
        command.Parameters.Add("@StuNo", SqlDbType.NChar, 5, "StuNo");
        command.Parameters.Add("@Name", SqlDbType.NVarChar, 40, "Name");
        command.Parameters.Add("@Sex", SqlDbType.NChar, 5, "Sex");
        command.Parameters.Add("@Birth", SqlDbType.DateTime, 40, "Birth");
        command.Parameters.Add("@Photo", SqlDbType.NChar, 5, "Photo");
        sda.InsertCommand = command;

        //SqlCommandBuilder cb = new SqlCommandBuilder(sda);   //建立 CommandBuilder 对象
                                                               //来自动生成 DataAdapter 的
                                                               //Command 命令
        sda.Fill(ds, "stuTable");   //用 Fill 方法返回的数据,填充 DataSet,数据表取名为
                                    //"stuTable"
        DataTable dtable = ds.Tables["stuTable"];   //将数据表 stuTable 的数据复制到
                                                    //DataTable 对象
        if (FileUpload1.HasFile == true) //把学生的照片上传到网站的"images"文件夹中,
                                         //以学号为名字进行保存
        {
            string fileName = this.txtStuNo.Text + ".jpg";
            FileUpload1.SaveAs(Server.MapPath(("images/") + fileName));
        }
        DataRow dr = ds.Tables["stuTable"].NewRow();     //增加新记录
        //给该记录赋值
        dr["StuNo"] = txtStuNo.Text.Trim();
        dr["Name"] = txtName.Text.Trim();
        dr["Sex"] = DdSex.SelectedValue;
        dr["Birth"] = Convert.ToDateTime(txtBirth.Text.Trim());
        dr["Photo"] = txtStuNo.Text.Trim() + ".jpg";
        dtable.Rows.Add(dr);
        sda.Update(ds, "stuTable");                          //提交更新
        Response.Write("增加成功<hr>");
    }
    catch (Exception ex)
    {
        Response.Write("记录新增失败,原因:" + ex.Message);
    }
    finally
    {
        sqlconn.Close();
        sqlconn = null;
        Response.Write("<h7>成功关闭 SQL Server 数据库的连接</h7><hr>");
    }
}
```

(3) 程序运行效果如图 5-14 所示。

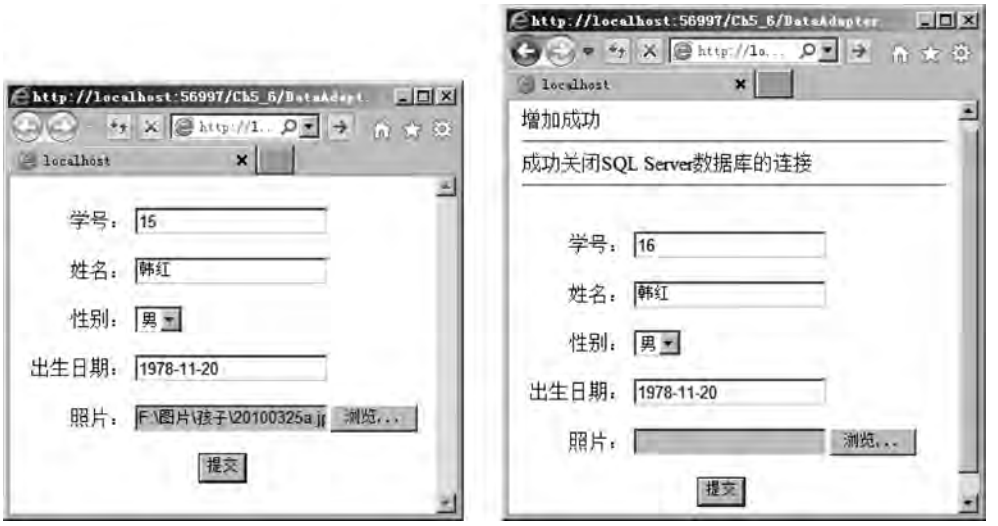

图 5-14　DataAdapterInsertDemo.aspx 页面设计及运行效果

3) 修改数据库的数据

使用 DataSet 和 DataAdapter 对象修改数据库数据和向数据库中添加数据的操作类似,在此不再举例,请读者自行完成。

4) 删除数据库的数据

使用 DataSet 和 DataAdapter 对象删除数据库数据的一般步骤如下：首先建立数据库连接；然后利用数据连接和 Select 语句建立 DataAdapter 对象；定义 DeleteCommand 属性,自定义 Delete 命令；使用 DataAdapter 对象的 Fill 方法把 Select 语句的查询结果放在 DataSet 对象的数据表中；接下来,将该数据表复制到 DataTable 对象中；最后,删除 DataTable 对象中的数据,并通过 DataAdpter 对象的 Update 方法向数据库提交数据。

【示例 5-7】 使用 DataSet 和 DataAdapter 对象删除符合条件的记录。

(1) 在网站项目 WebSite05 中新建文件夹 Ch5_7,在文件夹 Ch5_7 中添加一个名为 DataAdpterDeleteDemo.aspx 的 Web 页。

(2) 编写 DataAdpterDeleteDemo.aspx 页面中“删除”按钮的 Click 事件过程代码,如下：

```
protected void btnDel_Click(object sender, EventArgs e)
{
    string strCnn = " Server = .; Database = Student; Uid = sa; pwd = 123456";
    DataSet ds = new DataSet();
    using (SqlConnection cnn = new SqlConnection(strCnn))
    {
        SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", cnn);
        //定义 DeleteCommand 属性,自定义 Delete 命令,其中@StuNo 是参数
        sda.DeleteCommand = new SqlCommand("delete from StuInfo where StuNo = @StuNo",
cnn);
```

```
            //定义@StuNo 参数对应于 StuInfo 表的 StuNo 列
            sda.DeleteCommand.Parameters.Add("@StuNo", SqlDbType.VarChar, 8, "StuNo");
            sda.Fill(ds, "StuTable");
                                                    //调用 Fill 方法,填充 DataSet 的数据表 StuTable
            DataTable dtable = ds.Tables["StuTable"];   //将数据表 StuTable 的数据复制到
                                                        //DataTable 对象
            //设置 dtStuInfo 的主键,便于后面调用 Find 方法查询记录
            dtable.PrimaryKey = new DataColumn[] { dtable.Columns["StuNo"] };
            //根据 txtStuNo 文本框的输入查询相应的记录,以便修改
            DataRow dr = dtable.Rows.Find(txtStuNo.Text.Trim());
            if (dr != null)                 //如果存在相应记录,则删除并更新到数据库
            {
                dr.Delete();                //删除行记录
                sda.Update(dtable);         //提交更新
                Response.Write( "记录删除成功!" + "< hr >");
            }
            else
            {
                Response.Write("没有该记录!" + "< hr >");
            }
        }
    }
```

(3) 程序运行效果如图 5-15 所示。

图 5-15　DataAdpterDeleteDemo.aspx 页面设计及运行效果

单 元 小 结

本单元主要介绍了如何使用 ADO.NET 访问数据库。ADO.NET 是一个访问数据源通用接口,它允许以编程方式从 Web 窗体访问数据源。ADO.NET 允许使用 Command 和 DataReader 对象与数据库进行直接的交互,同时还以 DataAdapter 和 DataSet 对象的方式提供了一种高级、抽象的数据访问机制,实现断开式数据库操作。第一种方法需要的代码非常少,但可以完成大量工作;而第二种方法可以减少服务器的负担,使服务器获得更好的性能。ADO.NET 是一种强大的高性能数据库技术,在.NET 体系中占据着举足轻重的位置,读者要熟练掌握。

单元练习题

一、选择题

1.（　　）对象用于从数据库中获取仅向前的只读数据流，并且在内存一次只能存放一行数据。此对象具有较好的功能，可以简单地读取数据。

A. DataAdapter　　B. DataSet　　C. DataView　　D. DataReader

2. Command 对象执行查询语句时，调用（　　）方法会返回结果集中的第一条记录的第一个字段的值。

A. ExecuteNonQuery　　B. ExecuteReader

C. ExecuteScalar　　D. ExecuteXmlReader

3. 如果要从数据库中获取多行记录，应该使用 Command 对象的（　　）方法。

A. ExecuteNonQuery　　B. ExecuteReader

C. ExecuteScalar

4. 如果要对数据库执行修改、插入和删除操作，应该使用 Command 对象的（　　）方法。

A. ExecuteNonQuery　　B. ExecuteReader

C. ExecuteScalar

5.（　　）是开发人员要使用的第一个对象，被要求用于任何其他 ADO. NET 对象之前。

A. CommandBuilder 对象　　B. 命令对象

C. 连接对象　　D. DataAdapter 对象

6.（　　）表示一组相关表，在应用程序中这些表作为一个单元被引用。使用此对象可以快速从每一个表中获取所需的数据，当服务器断开时检查并修改数据，然后在下一次操作中使用这些修改的数据更新服务器。

A. DataTable 对象　　B. DataRow 对象

C. DataReader 对象　　D. DataSet 对象

7. 数据适配器 DataAdapter 填充数据集的方法是（　　）。

A. Fill　　B. GetChanges　　C. AcceptChanges　　D. Update

8. 如果 Command 对象执行的是存储过程，其属性 CommandType 应取（　　）。

A. CommandType. Text　　B. CommandType. StoredProcedure

C. CommandType. TableDirect　　D. 没有限制

9. 如果希望将 FlightNumber 字段的值在包含信息字段的表的第一个< td >元素中显示，要在表格的< td >元素添加（　　）代码以显示 FlightNumber 字段。

A. < td >< %＝FlightNumber %>< /td >

B. < td >< script runat＝"server"> FlightNumber < /script >< /td >

C. < td >< script > document. write("FlightNumber");< /scripts >< /td >

D. < td >＝FlightNumber < /td >

二、问答题

1. 列举常见的数据提供者,并且简单介绍对应的命名空间及作用。

2. 分别说明 SqlCommand 对象的 ExecuteReader、ExecuteNonQuery 和 ExecuteScalar 方法的作用。

3. 简述 DataSet 与 DataTable 的区别与联系。

4. 简述 SqlDataAdapter 对象查询数据库数据的步骤。