第 9 章　ASP.NET 操作数据库

通过对 ADO.NET 的基本讲解，以及讲解了一些数据源控件的基本用法后，本章将介绍一些 ASP.NET 操作数据库的高级用法，包括使用 SQLHelper，以及数据源控件对数据的操作。本章是对前面的数据库知识的一种补充和提升。

9.1　使用 ADO.NET 操作数据库

上一章中介绍了 ADO.NET 的基本概念、ADO.NET 的对象，以及如何使用 ADO.NET。使用 ADO.NET 能够极大的方便开发人员对数据库进行操作而无需关心数据库底层之间的运行，ADO.NET 不仅包括多个对象，同样包括多种方法，这些方法都可以用来执行开发人员指定的 SQL 语句，但是这些方法实现过程又不尽相同，本节将介绍 ADO.NET 中数据的操作方法。

9.1.1　使用 ExecuteReader()操作数据库

使用 ExecuteReader()操作数据库，ExecuteReader()方法返回的是一个 SqlDataReader 对象或 OleDbDataReader 对象。当使用 DataReader 对象时，不会像 DataSet 那样提供无连接的数据库副本，DataReader 类被设计为产生只读、只进的数据流。这些数据流都是从数据库返回的。所以，每次的访问或操作只有一个记录保存在服务器的内存中。

相比与 DataSet 而言，DataReader 具有较快的访问能力，并且能够使用较少的服务器资源。DataReader 对象提供了“游标”形式的读取方法，当从结果中读取了一行，则“游标”会继续读取到下一行。通过 Read 方法可以判断数据是否还有下一行，如果存在数据，则继续运行并返回 true，否则返回 false。示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";
SqlConnection con = new SqlConnection(str);
con.Open();                                                    //打开连接
string strsql = "select * from mynews";                        //SQL 查询语句
SqlCommand cmd = new SqlCommand(strsql, con);                  //初始化 Command 对象
SqlDataReader rd = cmd.ExecuteReader();                        //初始化 DataReader 对象
while (rd.Read())
{
    Response.Write(rd["title"].ToString());                    //通过索引获取列
}
```

DataReader 可以提高执行效率，有两种方式可以提高代码的性能，一种是基于序号的查询；第二种情况则是使用适当的 Get 方法来查询。一般来说，在数据库的设计中，需要设计索引键或主键来标识，在主键的设计中，自动增长类型是经常使用的，自动增长类型通常为整型，所以基于序号的查询可以使用 DataReader，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";    //设置连接字串
SqlConnection con = new SqlConnection(str);                               //创建连接对象
con.Open();                                                               //打开连接
```

```
string strsql = "select * from mynews where id=1 order by id desc";      //按标识查询
SqlCommand cmd = new SqlCommand(strsql, con);                            //创建 Command 对象
SqlDataReader rd = cmd.ExecuteReader();                                  //创建 DataReader 对象
while (rd.Read())                                                        //遍历数据库
{
    Response.Write(rd["title"].ToString());                              //读取相应行的信息
}
```

当使用 ExecuteReader()操作数据库时，会遇到知道某列的名称而不知道某列的号的情况，这种情况可以通过使用 DataReader 对象的 GetOrdinal()方法获取相应的列号。此方法接收一个列名并返回此列名所在的列号，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";   //创建连接字串
SqlConnection con = new SqlConnection(str);                              //创建连接对象
con.Open();                                                              //打开连接
string strsql = "select * from mynews where id=1 order by id desc";      //创建执行 SQL 语句
SqlCommand cmd = new SqlCommand(strsql, con);                            //创建 Command 对象
SqlDataReader rd = cmd.ExecuteReader();                                  //创建 DataReader 对象
int id = rd.GetOrdinal("title");                          //使用 GetOrdinal 方法获取 title 列的列号
while (rd.Read())                                                        //遍历 DataReader 对象
{
    Label1.Text = "新闻 id 是" + rd["id"];                               //输出对象的值
}
```

当完成数据库操作时，需要关闭数据库连接，DataReader 对象在调用 Close()方法即关闭与数据库的连接，如果在没有关闭之前又打开另一个连接，系统会抛出异常。示例代码如下所示。

```
rd.Close();                                                              //关闭 DataReader 对象
```

ExecuteReader()可以执行相应的 SQL 语句，例如插入、更新以及删除等，当需要执行插入、更新或删除时，可以使用 ExecuteReader()进行数据操作，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";   //创建连接字串
SqlConnection con = new SqlConnection(str);                              //创建连接对象
con.Open();                                                              //打开连接
string strsql = "insert into mynews values ('执行更新后的标题')";       //创建执行 SQL 语句
SqlCommand cmd = new SqlCommand(strsql, con);                            //创建 Command 对象
SqlDataReader rd = cmd.ExecuteReader();                                  //使用 ExcuteReader()方法
while (rd.Read())                                                        //读取数据库
{
    Response.Write(rd["title"].ToString() + "<hr/>");
}
rd.Close();                                                              //关闭 DataReader 对象
Response.Redirect("ExecuteReader.aspx");
```

当执行了插入、删除等数据库操作时，ExecuteReader 返回为空的 DataReader 对象。当使用 Read 方法遍历读取数据库时，并不会显示相应的数据信息，因为不是查询语句，则返回一个没有任何数据的 System.Data.OleDb.OleDbDataReader 类型的集（EOF），但是 ExecuteReader 方法可以执行 SQL 语句。如图 9-1 所示。

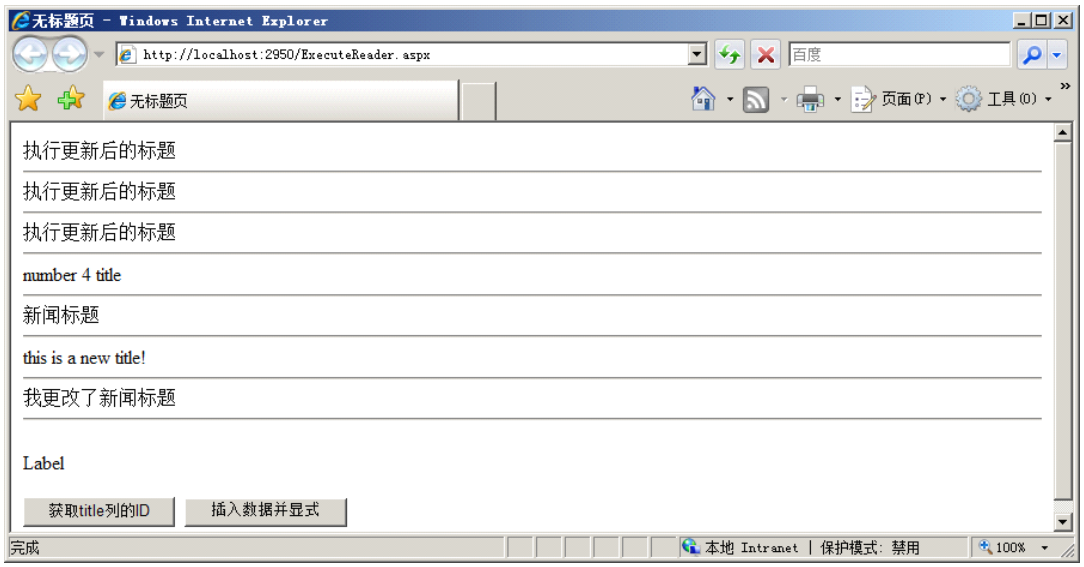

图 9-1　ExecuteReader()执行查询和事务处理

使用 ExecuteReader()操作数据库，通常情况下是使用 ExecuteReader()进行数据库查询操作，使用 ExecuteReader()查询数据库能够提升查询效率，而如果需要进行数据库事务处理的话，ExecuteReader()方法并不是理想的选择。

9.1.2　使用 ExecuteNonQuery()操作数据库

使用 ExecuteNonQuery()操作数据库时，ExecuteNonQuery()并不返回 DataReader 对象，返回的是一个整型的值，代表执行某个 SQL 语句后，在数据库中影响的行数，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";      //创建连接字串
SqlConnection con = new SqlConnection(str);                                 //创建连接对象
con.Open();                                                                 //打开连接
string strsql = "select top 5 * from mynews order by id desc";
SqlCommand cmd = new SqlCommand(strsql, con);                               //使用 ExecuteNonQuery
Label1.Text="该操作影响了"+cmd.ExecuteNonQuery()+"行";                       //执行 SQL 语句并返回行
```

上述代码使用了 SELECT 语句，并执行语句，返回受影响的行数。运行后，发现返回的结果为-1，说明，当使用 SELECT 语句时，并没有对任何行有任何影响。ExecuteNonQuery()通常情况下为数据库事务处理的首选，当需要执行插入、删除、更新等操作时，首选 ExecuteNonQuery()。

对于更新、插入和删除的 SQL 句，ExecuteNonQuery()方法的返回值为该命令所影响的行数。对于“CREATE TABLE”和“DROP TABLE”语句，返回值为 0，而对于所有其他类型的语句，返回值为-1。ExecuteNonQuery()操作数据时，可以不使用 DataSet 直接更改数据库中的数据，示例代码如下所示。

```
protected void Button1_Click(object sender, EventArgs e)
{
    string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";      //创建连接字串
    SqlConnection con = new SqlConnection(str);                                 //创建连接对象
    con.Open();                                                                 //打开连接
    string strsql = "delete from mynews where id>4";                            //编写执行删除的 SQL 语句
    SqlCommand cmd = new SqlCommand(strsql, con);                               //创建 Command 对象
    Label1.Text = "该操作影响了" + cmd.ExecuteNonQuery() + "行";                 //返回影响行数
}
```

运行上述代码后，会执行删除 id 号大于 4 的数据事务，当执行删除并删除完毕后，则 ExecuteNonQuery()方法返回受影响的行数，如图 9-2 所示。

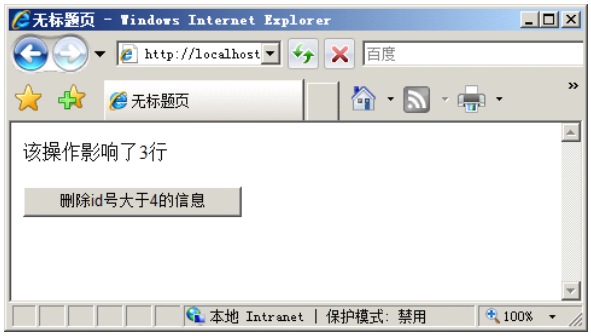

图 9-2　ExecuteNonQuery()方法

ExecuteNonQuery()操作主要进行数据库操作，包括更新、插入和删除等操作，并返回相应的行数。在进行数据库事务处理时或不需要 DataSet 为数据库进行更新时，ExecuteNonQuery()方法是数据操作的首选。因为 ExecuteNonQuery()支持多种数据库语句的执行。

注意：有些项目中，通过判断 ExecuteNonQuery()的返回值来判断 SQL 语句是否执行成功，这样是有失偏颇的，因为当使用创建表的语句时，就算执行成功也会返回-1。

9.1.3　使用 ExecuteScalar()操作数据库

ExecuteScalar()方法也用来执行 SQL 语句，但是 ExecuteScalar()执行 SQL 语句后的返回值与 ExecuteNonQuery()并不相同，ExecuteScalar()方法的返回值的数据类型是 Object 类型。如果执行的 SQL 语句是一个查询语句（SELECT），则返回结果是查询后的第一行的第一列，如果执行的 SQL 语句不是一个查询语句，则会返回一个未实例化的对象，必须通过类型转换来显示，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";      //创建连接字串
SqlConnection con = new SqlConnection(str);                                 //创建连接对象
con.Open();                                                                 //打开连接
string strsql = "select * from mynews order by id desc";
SqlCommand cmd = new SqlCommand(strsql, con);
Label1.Text = "查询出了 Id 为" + cmd.ExecuteScalar() \;                      //使用 ExecuteScalar 查询
```

通常情况下 ExecuteNonQuery()操作后返回的是一个值，而 ExecuteScalar()操作后则会返回一个对象，ExecuteScalar()经常使用于当需要返回单一值时的情况。例如当插入一条数据信息时，常常需要马上知道刚才插入的值，则可以使用 ExecuteScalar()方法。示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";      //创建连接字串
SqlConnection con = new SqlConnection(str);                                 //创建连接对象
con.Open();                                                                 //打开连接
string strsql = "insert into mynews values ('刚刚插入的 id 是多少?')
SELECT  @@IDENTITY  as  'bh'";                                              //插入语句
SqlCommand cmd = new SqlCommand(strsql, con);                               //执行语句
Label1.Text = "刚刚插入的行的 id 是" + cmd.ExecuteScalar();                  //返回赋值
```

上述代码使用了 SELECT　@@IDENTITY 语法获取刚刚执行更新后的 id 值，然后通过使用 ExecuteScalar()方法来获取刚刚更新后第一行第一列的值。

9.1.4 使用 ExecuteXmlReader()操作数据库

ExecuteXmlReader()方法用于操作 XML 数据库，并返回一个 XmlReader 对象，若需要使用 ExecuteXmlReader()方法，则必须添加引用 System.Xml。XmlReader 类似于 DataReader，都需要通过 Command 对象的 ExecuteXmlReader()方法来创建 XmlReader 的对象并初始化，示例代码如下所示。

```
XmlReader xdr = cmd.ExecuteXmlReader();                    //创建 XmlReader 对象
```

ExecuteXmlReader()返回 XmlReader 对象，XmlReader 特性如下所示：

- XMLReader 是面向流的，它把 XML 文档看作是文本数据流。
- XMLReader 是一个抽象类。
- XMLReader 使用 pull 模式处理流。
- 三个派生类：XMLTextReader、XMLNodeReader 和 XMLValidatingReader

下面代码实现了获取当前节点中属性的个数。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";    //创建连接字串
SqlConnection con = new SqlConnection(str);                               //创建连接对象
con.Open();                                                               //打开连接
string strsql = "select * from mynews order by id desc FOR XML AUTO, XMLDATA";
SqlCommand cmd = new SqlCommand(strsql, con);                             //创建 Command 对象
XmlReader xdr = cmd.ExecuteXmlReader();                                   //创建 XmlReader 对象
Response.Write(xdr.AttributeCount);                                       //读取节点个数
```

上述代码使用了 SQL 语言中的 FOR XML AUTO,、XMLDATA 关键字，当执行 ExecuteXmlReader()方法时，会返回 XmlReader 对象，若不指定 FOR XML AUTO,、XMLDATA 关键字，则系统会抛出异常 。

9.2 ASP.NET 创建和插入记录

在数据库操作中，经常需要对数据库中的内容进行插入操作。例如当有一个用户发布了评论，或者一个用户要购买某个商品，都需要插入记录来保存用户的相应的信息，以便当用户再次登录网站或应用时，能够及时获取自己购买的信息。

9.2.1 SQL INSERT 数据插入语句

使用 SQL INSERT 语句能够实现数据库的插入，SQL 语句必须遵照一些规范，SQL INSERT 语句的一般语法形式如下所示：

```
INSERT [INTO]
{table_name}
{
 [(column_list)]
 {
 VALUES({DEFAULT|NULL|expression} [,…n])
}
}
```

上述代码规范了 INSERT 语句的编写规范，其中：

- INSERT 是 SQL 插入关键字。
- [INTO]是表名称之前能够包含的可选关键字。

- Table_name 是相关的表名称。
- column_list 是列的集合，如果有多个列可用都好隔开。
- VALUES 是相应的列的值。

如果需要向表 mytables 插入数据，而 mytables 里包括自动增长的主键 id 和 title 两列，则 INSERT 语句可以编写为如下代码。

```
INSERT INTO mytables VALUES (‘新的新闻标题’)
```

上述代码向表 mytables 中插入了一条新记录，并将 title 赋值为“新的新闻标题”。值得注意的是，在这条语句中，并没有编写列的集合，是因为当不编写 column_list 时，则默认为每一个列插入数值。

注意：自动增长的主键类型的字段，无需向数据中插入数值，因为 SQL Server 会自动为该列赋值。

如果需要当插入数据时，需要指定插入相应的列的值，则可以将 SQL 语句代码编写如下。

```
INSERT INTO mytables (title) VALUES (‘新的新闻标题’)
```

上述代码指定了列 title，并对应了相应的值。若在表中存在多个列，列的顺序和列相应的值的顺序必须匹配。例如有 3 列并分别为 number1，string2，datetime3，当需要向其中插入数据时，则可以编写以下 SQL 语句。

```
INSERT INTO examtable (number1,string2,datetime3) VALUES (1,’this is a string’,’2008/9/18’)
```

上述代码编写了 INSERT 语句以便数据的插入，同样在插入语句中如果需要插入所有的列，可以简化 INSERT 语句以便快速进行数据插入，示例代码如下所示。

```
INSERT INTO examtable VALUES (1,’this is a string’,’2008/9/18’)
```

值得注意的是，无论按照何种方法编写 SQL 语句，值和列都应该相互匹配。

9.2.2 使用 Command 对象更新记录

编写了 SQL 语句后，必须执行 SQL 语句，在 ADO.NET 中，执行 SQL 语句有很多方法，其中推荐使用 Command 命令的 ExecuteNonQuery()。执行 SQL 语句的命令的必要步骤如下所示。

- 打开数据连接。
- 创建一个新的 Command 对象。
- 定义一个 SQL 命令。
- 执行 SQL 命令。
- 关闭连接。

从上面的步骤可以发现执行 SQL 语句是非常容易的，首先必须要打开到数据库的连接，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";
SqlConnection con = new SqlConnection(str);                    //创建连接对象
con.Open();                                                    //打开连接
```

其中，str 是数据连接字串，用来初始化 Connection 对象，说明如何连接数据库，当数据库连接完毕后，可以使用 Open 方法打开数据连接。完成数据库连接后，需创建一个新的 Command 对象，示例代码如下所示。

```
SqlCommand cmd = new SqlCommand("insert into mynews value ('插入一条新数据')", con);
```

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，令一个是数据库连接对象。创建 Command 对象后，就可以执行 SQL 命令，执行后完成并关闭数据连接，示例代码如下所示。

```
cmd.ExecuteNonQuery();                                         //执行 SQL 命令
con.Close();                                                   //关闭连接
```

上述代码使用了 ExecuteNonQuery()方法执行了 SELECT 语句的操作，当执行完毕后就需要对现有

的连接进行关闭，以释放系统资源。

9.2.3 使用 DataSet 数据集插入记录

使用 INSERT 语句能够完成数据插入，使用 DataSet 对象也可以完成数据插入。为了将数据库的数据填充到 DataSet 中，则必须先使用 DataAdapter 对象的方法实现填充，当数据填充完成后，开发人员可以将记录添加到 DataSet 对象中，然后使用 Update 方法将记录插入数据库中。使用 DataSet 更新记录的步骤如下所示：

- 创建一个 Connection 对象。
- 创建一个 DataAdapter 对象。
- 初始化适配器。
- 使用数据适配器的 Fill 方法执行 SELECT 命令，并填充 DataSet。
- 使用 DataTable 对象提供的 NewRow 方法创建新行。
- 将数据行的字段设置为插入的值。
- 使用 DataRowAdd 类的 Add 方法将数据行添加到数据表中。
- 把 DataAdapter 类的 InsertCommand 属性设置成需要插入记录的 INSERT 语句。
- 使用数据适配器提供的 Update 方法将新记录插入数据库。
- 使用 DataSet 类提供的 AcceptChanges 方法将数据库与内存中的数据保持一致。

当使用 DataSet 插入记录前，需要创建 Connection 对象以保证数据库连接，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";          //创建连接字串
SqlConnection con = new SqlConnection(str);                                    //创建连接对象
con.Open();                                                                    //打开连接
```

上述代码创建了一个数据库连接，并打开了数据库连接。完成数据连接后，就需要查询表中的数据并使用 DataAdapter 对象初始化适配器，示例代码如下所示。

```
string strsql = "select * from mynews";                                        //编写 SQL 语句
SqlDataAdapter da = new SqlDataAdapter(strsql, con);                           //创建适配器
```

DataAdapter 对象默认构造函数包括两个参数，其中一个参数是需要执行的 SQL 语句，另一个是 Connection 对象。在初始化适配器后，需要对适配器的相应的属性做设置，使用 SqlCommandBuilder 对象可以让系统构造 InsertCommand 属性，示例代码如下所示。

```
SqlCommandBuilder build = new SqlCommandBuilder(da);                           //构造 SQL 语句
```

使用适配器的 Fill 方法能够填充 DataSet 数据集，示例代码如下所示。

```
DataSet ds = new DataSet();                                                    //创建数据集
da.Fill(ds, "datatable");                                                      //填充数据集
DataTable tb = ds.Tables["datatable"];                                         //创建表
tb.PrimaryKey = new DataColumn[] { tb.Columns["id"] };                         //创建表的主键
```

上述代码创建了一个 DataSet 数据集对象，被填充数据后，数据集中表的名称被命名为 datatable，该命名与数据库中的表的名称并不冲突。填充了 DataSet 数据对象后，需要使用 DataRow 对象为 DataSet 添加数据，示例代码如下所示。

```
DataRow row = ds.Tables["datatable"].NewRow();                                 //创建 DataRow
row["title"] = "使用 DataSet 插入新行";                                        //赋值新列
row["id"] = "15";
```

上述代码使用了 NewRow 方法创建新行返回 DataRow 对象，当 DataRow 对象中的相应的元素被赋值后，则需要使用 Rows.Add 方法增加新行，因为只对 DataRow 对象赋值，并不能自动的在数据库中增加新行。示例代码如下所示。

```
ds.Tables["datatable"].Rows.Add(row);                                    //添加新行
```

上述代码将数据更新到 DataSet 数据集中，为了保持数据集中的数据和数据库的数据的一致性，需使用 Update 方法，示例代码如下所示。

```
da.Update(ds, "datatable");                                              //更新数据
```

当执行了 Update 方法后，数据库中的数据就会同步 DataSet 数据集中的数据进行数据更新。

9.3 ASP.NET 更新数据库

在应用程序的开发中，常常会需要对数据库中现有的内容进行更新操作。ADO.NET 提供了若干不同的更新数据库中记录的方法，如果需要更新数据库中的某列的值或者某几列的值，则需要使用 SQL UPDATE 命令进行数据库更新。

9.3.1 SQL UPDATE 数据更新语句

使用 SQL UPDATE 语句能够实现数据库中数据的更新，SQL UPDATE 语句的一般语法格式如下所示。

```
UPDATE
{table_name}
{
  SET column1_name=expression1,
      column2_name=expression2,
      .
      .
      columnN_name=expressionN
{WHERE condition1 AND|OR condition2}
}
```

上述代码规范了 UPDATE 语句的编写规范，其中：

- UPDATE 是 SQL 更新关键字。
- table_name 是需要更新的表的名称。
- columnN_name 是需要更新的列的名称。
- expression 是列相应的值。
- WHERE 是 SQL 语句关键字。
- condition 是条件。

如果需要更新表 mytable 中的某行的数据，则可以编写 SQL UPDATE 语句进行更新，示例代码如下所示。

```
UPDATE mytable SET title='修改后的数据' where id=3
```

上述代码更新了 id 为 3 的数据中的 title，并将 title 字段的值修改为“修改后的数据”。

9.3.2 使用 Command 对象更新记录

如需要执行 UPDATE 语句时，同样可以使用 Command 对象执行语句。Command 对象基本上能够执行所有需要进行数据更新的 SQL 语句。使用 Command 对象进行数据库操作的步骤基本如下所示。

- 创建数据库连接。

- 创建一个 Command 对象，并指定一个 SQL UPDATE（或存储过程）。
- 使用 Command 对象的 ExecuteNonQuery()方法执行 UPDATE（或存储过程）。
- 关闭数据库连接。

当需要执行 UPDATE 语句时，首先必须要打开到数据库的连接，打开连接后，使用 Command 对象执行 SQL 语句，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";
SqlConnection con = new SqlConnection(str);                    //创建连接对象
con.Open();                                                    //打开连接
```

其中，str 同样是数据连接字串，用来初始化 Connection 对象，说明如何连接数据库，当数据库连接完毕后，可以使用 Open 方法打开数据连接。完成数据库连接后，需创建一个新的 Command 对象进行数据更新，示例代码如下所示。

```
SqlCommand cmd = new SqlCommand("UPDATE mynews SET title='修改后的数据'
where id=3", con);                                             //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，令一个是数据库连接对象。创建 Command 对象后，就可以执行 SQL 命令，执行后完成并关闭数据连接，示例代码如下所示。

```
cmd.ExecuteNonQuery();                                         //执行 SQL 命令
con.Close();                                                   //关闭连接
```

上述代码使用了 ExecuteNonQuery()方法进行 SQL UPDATE 语句的执行，从而能够更新数据库中的相应数据。

9.3.3 使用 DataSet 数据集更新记录

ADO.NET 的 DataSet 对象提供了更好的编程实现数据库的更新功能。因为 DataSet 对象与数据库始终不是连接的，开发人员可以向脱离数据库的 DataSet 对象中增加列、删除列或更新列。当完成了修改后，则可以通过将 DataSet 对象连接到 DataAdapter 对象来将记录传输给数据库。DataSet 更新记录的步骤如下所示。

- 创建一个 Connection 对象。
- 创建一个 DataAdapter 对象。
- 初始化适配器。
- 使用数据适配器的 Fill 方法执行 SELECT 命令，并填充 DataSet。
- 执行 SqlCommandBuilder 方法生成 UpdataCommand 方法。
- 创建 DataTable 对象并指定相应的 DataSet 中的表。
- 创建 DataRow 对象并查找需要修改的相应行。
- 更改 DataRow 对象中的列的值。
- 使用 Update 方法进行数据更新。

在更新记录前，首先需要查询出相应的数据，查询相应的数据后才能够填充 DataSet，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";   //创建连接字串
SqlConnection con = new SqlConnection(str);                             //创建连接对象
con.Open();                                                             //打开连接
string strsql = "select * from mynews";                                 //执行查询
SqlDataAdapter da = new SqlDataAdapter(strsql, con);                    //使用 DataAdapter
DataSet ds = new DataSet();                                             //使用 DataSet
da.Fill(ds, "datatable");                                               //使用 Fill 方法填充 DataSet
```

上述代码将查询出来的数据集保存在表为 datatable 的 DataSet 记录集中，DataSet 记录集的表的名称可以按照开发人员的喜好来编写，从而区分内存中表的数据和真实的数据库的区别。当需要处理数据时，只需要处理相应名称的表即可，示例代码如下所示。

```
DataTable tb = ds.Tables["datatable"];
```

当需要执行更新时，可直接使用 DataSet 对象进行更新操作来修改其中的一行或多行记录，示例代码如下所示。

```
DataTable tb = ds.Tables["datatable"];
tb.PrimaryKey = new DataColumn[] { tb.Columns["id"] };
DataRow row = tb.Rows.Find(1);
row["title"] = "新标题";
```

当需要更新某个记录时，必须在更新之前查找到该行的记录。可以使用 Rows.Find 方法查找到相应的行，然后将数据集表中的该行的列值进行更新。使用 DataAdapter 的 Update 方法可以更新 DataSet 数据集，并保持数据集和数据库中数据的一致性，示例代码如下所示。

```
da.Update(ds, "datatable");
```

在执行以上代码，可能会抛出异常“当传递具有已修改行的 DataRow 集合时,更新要求有效的 UpdateCommand”。这是因为在更新时，并没有为 DataAdapter 对象配置 UpdateCommand 方法，可以通过 SqlCommandBuilder 对象配置 UpdateCommand 方法，示例代码如下所示。

```
SqlCommandBuilder build = new SqlCommandBuilder(da);
```

上述代码为 DataAdapter 对象自动配置了 UpdateCommand，DeleteCommand 等方法，当执行更新时，无需手动配置 UpdateCommand 方法。

9.4 ASP.NET 删除数据

当数据库中的数据过多，或需要对数据库进行数据优化时，则可能需要对数据库中的数据进行删除，例如用户的操作，长期不上线的用户资料，都可以删除。ADO.NET 提供多种数据库的删除方法，并且同样支持 DataSet 方法删除数据库。

9.4.1 SQL DELETE 数据删除语句

使用 SQL DELETE 语句能够实现数据库中数据的更新，SQL DELETE 语句的一般语法格式如下所示。

```
DELETE [FROM]
{table_name}
[WHERE condition1 AND|OR condition2]
```

上述代码规范了 DELETE 语句的编写规范，其中：

- DELETE 是 SQL 删除关键字。
- FORM 是一个可以选择的关键字。
- table_name 是表的名称。
- WHERE 是一个 SQL 关键字。
- conditionN 是执行 DELETE 命令中需要达成的若干条件。

SQL DELETE 相对来说比较简单，当需要对某个表中的数据进行删除时，可以使用 DELETE 语句来执行删除操作，在编写 DELETE 语句时，需要指定表，并且指定相应的条件，示例代码如下所示。

```
DELETE FROM mynews WHERE ID=3
```

上述代码指定删除了 mynews 表中 ID 为 3 的行。如果不编写 WHERE 子句，则该表中所有的行都能够达成删除的条件，则会删除表中所有的行，示例代码如下所示。

```
DELETE FROM mynews
```

在编写删除语句时，可以通过编写相应的条件来提高执行的效率。

9.4.2 使用 Command 对象删除记录

当需要执行删除语句，可以使用 Command 对象来删除数据库中的记录。Command 对象的使用方法在前面的 SQL 语句介绍中已经讲的比较多了，在删除记录时，其使用方法基本相同。使用 Command 对象进行数据库操作的步骤基本如下所示。

- 创建数据库连接。
- 创建一个 Command 对象，并指定一个 SQL DELETE（或存储过程）。
- 使用 Command 对象的 ExecuteNonQuery()方法执行 DELETE（或存储过程）。
- 关闭数据库连接。

当需要执行 DELETE 语句时，首先必须要打开到数据库的连接，打开连接后，使用 Command 对象执行 SQL 语句，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";
SqlConnection con = new SqlConnection(str);
con.Open();                                                    //打开连接
```

完成数据库连接后，需创建一个新的 Command 对象，示例代码如下所示。

```
SqlCommand cmd = new SqlCommand("Delete mynews where id=3", con);
```

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，令一个是数据库连接对象。创建 Command 对象后，就可以执行 SQL 命令，执行后完成并关闭数据连接，示例代码如下所示。

```
cmd.ExecuteNonQuery();                                         //执行 SQL 命令
con.Close();                                                   //关闭连接
```

9.4.3 使用 DataSet 数据集删除记录

使用 DataSet 删除记录和使用 DataSet 更新记录非常的相似，DataSet 删除记录的步骤如下所示。

- 创建一个 Connection 对象。
- 创建一个 DataAdapter 对象。
- 初始化适配器。
- 使用数据适配器的 Fill 方法执行 SELECT 命令，并填充 DataSet。
- 执行 SqlCommandBuilder 方法生成 UpdataCommand 方法。
- 创建 DataTable 对象并指定相应的 DataSet 中的表。
- 创建 DataRow 对象并查找需要修改的相应行。
- 使用 Delete 方法删除该行。
- 使用 Updata 方法进行数据更新。

在删除记录前，首先需要创建连接，示例代码如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='sa'";
SqlConnection con = new SqlConnection(str);
con.Open();
```

```
string strsql = "select * from mynews";
```

上述代码创建了与数据库的连接，并编写 SQL 查询语句来填充 DataSet。填充 DataSet 对象需使用 DataAdapter，示例代码如下所示。

```
SqlDataAdapter da = new SqlDataAdapter(strsql, con);
SqlCommandBuilder build = new SqlCommandBuilder(da);
DataSet ds = new DataSet();
da.Fill(ds, "datatable");
```

编写完成后，需要创建 DataTable 对象对 DataSet 中相应的数据进行操作，其代码和更新记录基本相同，示例代码如下所示。

```
DataTable tb = ds.Tables["datatable"];
tb.PrimaryKey = new DataColumn[] { tb.Columns["id"] };
DataRow row = tb.Rows.Find(3);
```

在进行删除之前，同样需要找到相应的行，来指定删除语句所需要删除的行，示例代码如下所示。

```
row.Delete();
```

读者可以看到，DataSet 删除方法与更新方法不同的地方只操作语句的不同，在更新中使用的是 Update()方法，而在删除中使用的是 Delete()方法。

注意：当使用 Delete 方法删除记录行的时候，可以通过调用 DataRow 对象的 RejectChanges 方法取消对记录的删除，当使用该方法删除记录行时，该行的状态会恢复为 Unchanged。

在删除完毕后，同样需要保持 DataSet 中的数据和数据库中的数据的一致性，示例代码如下所示。

```
da.Update(ds, "datatable");
```

使用 Update 方法能够使 DataSet 中的数据和数据库中的数据保持一致性，在 ASP 中，这种方法也比较常见。

9.5 使用存储过程

存储过程在开发过程中经常被使用，因为存储过程能够将数据操作和程序操作在代码上分离，而且存储过程相对于 SQL 语句而言，具有更好的性能和安全性，使用存储过程能够提高应用程序的性能和安全性。

9.5.1 存储过程的优点

在数据库操作中，已经有了 SQL 语句，为何还需要存储过程。因为存储过程有 SQL 语句不能具备的特点和优点，以至于存储过程能在严格的数据库驱动的应用程序中起到重要的作用。存储和过程有点包括：

- 事务处理。
- 速度和性能。
- 过程控制。
- 安全性。
- 减少网络流量和通信。
- 模块化。

1．事务处理

存储过程中，包括多个 SQL 语句，存储过程中的 SQL 语句属于事务处理的范畴。也就是说，存储

过程类似于一个函数，当执行存储过程时，存储过程中的 SQL 语句要不都执行，要不都不执行。

2．速度和性能

存储过程由数据库服务器编译和优化，优化包括使用存储过程在运行时所必须的特定数据库的结构信息，这样在执行过程中会节约很多时间。存储过程完全在数据库服务器上执行，避免了大量的 SQL 语句代码的传递，对于循环使用 SQL 语句而言，存储过程在速度和性能上都被优化。

3．过程控制

在编写存储过程中，可以使用 IF ELSE、FOR 以及 WHILE 循环，这些语句并不能在 SQL 语句中编写，但是可以在存储过程中编写。当需要进行大量的和复杂的操作时，SQL 语句需要通过和编程语言一同编写才能实现，而且实现复杂。相比之下，存储过程可以对过程进行控制。

4．安全性

存储过程也可以作为额外的安全层。开发人员或者用户，都只能对数据库中的存储过程进行使用，而无法直接对表进行数据操作，这样封装了数据操作，提高安全性。

5．减少网络流量和通信

存储过程是在数据库服务器上运行的，在使用存储过程中，无需将大量的 SQL 语句代码传递给数据库服务器，而只需告诉数据库服务器执行哪个存储过程即可，而数据库服务器则会自行执行中间处理操作，而不会通过网络传递不必要的数据。

6．模块化

正如代码编写规范和设计模式一样，通常情况下，开发团队或者公司需要严谨的代码编写风格和良好的协调能力，例如一个团队有人专门负责编码，有人专门负责数据库开发，那么可以让数据库开发人员负责数据库的开发，而编码的程序员只需要使用数据库开发人员设计的存储过程即可。在这种情况下，数据库操作和应用程序编码的操作被分开，在维护、管理中，也非常方便，如果数据库存储过程的代码出现问题，则只需要修改存储过程中的代码即可。

9.5.2 创建存储过程

存储过程可以通过 SQL Server Management Studio 创建，也可以使用.NET 框架通过编程实现.SQL Server Management Studio 创建存储过程比较方便，右击【对象资源管理器】中的相应的数据库，在下拉菜单中选择【可编程性】选项并选择【存储过程】选项。单击右键，选择【新建存储过程】选项，系统会自动创建一个新的标签（tab）窗口，以提供输入存储过程语句，如图 9-3 所示。

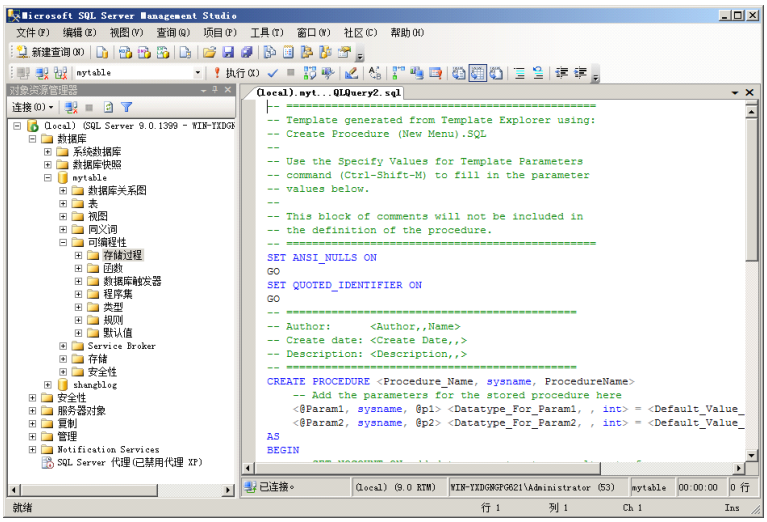

图 9-3　使用 Server Management Studio 创建存储过程

在 tab 窗口中输入存储过程，代码如下所示。

```
CREATE PROC myproc
(
@id int,
@title varchar(50) OUTPUT
)
AS
SET NOCOUNT ON
DECLARE @newscount int
SELECT @title=mynews.title,@newscount=COUNT(mynews.id)
FROM mynews
WHERE (id=@id)
GROUP BY mynews.title
RETURN @newscount
```

上述存储过程返回了数据库中新闻的标题内容。“@id”表示新闻的 id，@title 表示新闻的标题，此存储过程将返回“@title”的值，并且返回新闻的总数。在 C#中同样可以使用编程实现存储过程的创建，示例代码如下所示。

```
string str = "CREATE PROC myproc" +
"(" +
"@id int," +
"@title varchar(50) OUTPUT" +
")" +
"AS" +
"SET NOCOUNT ON" +
"DECLARE @newscount int" +
"SELECT @title=mynews.title,@newscount=COUNT(mynews.id)" +
"FROM mynews" +
"WHERE (id=@id)" +
"GROUP BY mynews.title" +
"RETURN @newscount";
SqlCommand cmd = new SqlCommand(str, con);
cmd.ExecuteNonQuery();                    //使用 cmd 的 ExecuteNonQuery 方法创建存储过程
```

上述代码通过使用 SqlCommand 对象的 ExecuteNonQuery()方法在数据库中创建了一个存储过程，该存储过程用于返回了数据库中新闻的标题内容。

9.5.3 调用存储过程

创建存储过程之后，可以在.NET 应用程序中使用存储过程。存储过程可以看成是一个函数，可以对存储过程进行调用，传递参数，接受返回值。在调用存储过程前，首先要与数据库建立连接，示例代码如如下所示。

```
string str = "server='(local)';database='mytable';uid='sa';pwd='Sa'";
SqlConnection con = new SqlConnection(str);
con.Open();                                                    //打开连接
```

建立与数据库连接后，需要使用 Command 对象使用存储过程，Command 对象接受的两个参数分别为 SQL 语句和 Connection 对象，在使用存储过程时，其中表示 SQL 语句的参数可以直接编写为存储过程名，代码如下所示。

```
SqlCommand cmd = new SqlCommand("getdetail", con);             //使用存储过程
```

默认情况下，Command 对象的类型是 SQL 语句，必须将 Command 对象的 CommandType 属性设置为存储过程，系统才会调用存储过程，示例代码如下所示。

```
cmd.CommandType = CommandType.StoredProcedure;                 //设置 Command 对象的类型
```

设置执行类型后，需要为存储过程增加参数，示例代码如下所示。

```
SqlParameter spr;                                              //表示执行一个存储过程
spr = cmd.Parameters.Add("@id", SqlDbType.Int);                //增加参数 id
spr = cmd.Parameters.Add("@title", SqlDbType.NChar,50);        //增加参数 title
spr.Direction = ParameterDirection.Output;                     //该参数是输出参数
spr = cmd.Parameters.Add("@count", SqlDbType.Int);             //增加 count 参数
spr.Direction = ParameterDirection.ReturnValue;                //该参数是返回值
cmd.Parameters["@id"].Value = 1;                               //为参数初始化
cmd.Parameters["@title"].Value = null;                         //为参数初始化
```

参数设置完毕后，执行 ExecuteNonQuery 方法能够执行存储过程，就相当于开始调用函数，示例代码如下所示。

```
cmd.ExecuteNonQuery();                                         //执行存储过程
```

当存储过程执行完毕后，能够获取参数和返回值，示例代码如下所示。

```
Label1.Text = cmd.Parameters["@count"].Value.ToString();       //获取返回值
```

使用 SQL Server Management Studio 同样能够执行存储过程，单击存储过程，单击右键，选择执行存储过程，系统会提示输入参数，如图 9-4 所示。输入相应的参数，单击确定，系统会执行存储过程并返回相应的值，如图 9-5 所示。

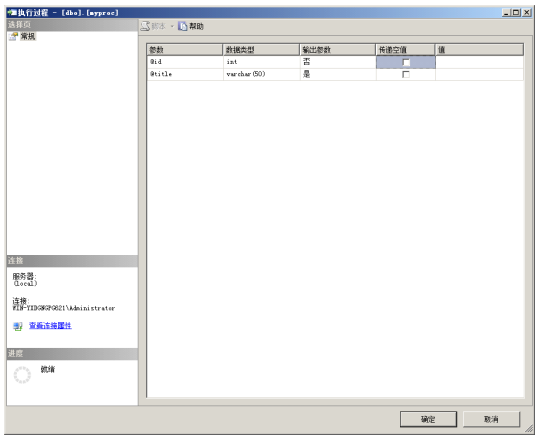

图 9-4　为存储过程传递参数

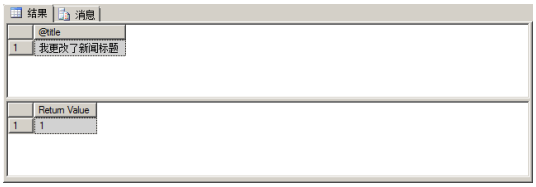

图 9-5　执行完成

使用 SQL Server Management Studio 能够快速的创建和使用存储过程，同样，能够通过编程的方法实现存储过程的创建、参数的传递以及执行。存储过程的优点就在于速度比较快，能够控制过程、减少网络通信和模块化，熟练的使用存储过程能够提高应用程序的性能和复用性。

9.6　ASP.NET 数据库操作实例

在了解了数据源控件和数据绑定控件的功能和使用方法，并且了解了 ADO.NET 的基本知识后，就可以使用控件和 ADO.NET 来操作数据库。ASP.NET 提供了强大的数据源控件和数据绑定控件，能够迅速的对数据库进行操作，同时，使用 ADO.NET 对数据进行操作，能够加深对 ADO.NET 的认识。

9.6.1　制作用户界面（UI）

使用数据控件和数据源控件显式数据，则需要为控件制作相应的用户界面，让数据控件对用户呈现的效果更好。首先，需要使用创建数据绑定控件 GridView 和数据源控件，并配置数据源控件，如图 9-6 所示。

显然，对于用户而言，该数据源控件和数据绑定控件显然很不友好，这里就需要对数据绑定控件的界面进行修改。通过配置数据绑定控件的相应格式可以修改数据绑定控件的外观，如图 9-7 所示。

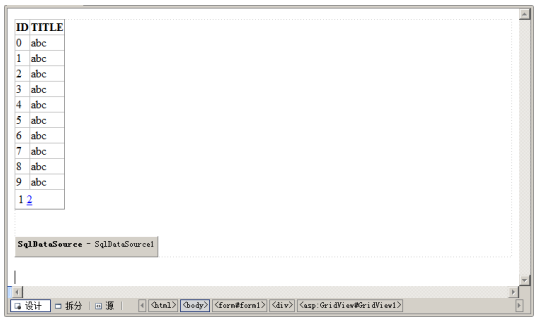

图 9-6　配置数据源控件和数据绑定控件

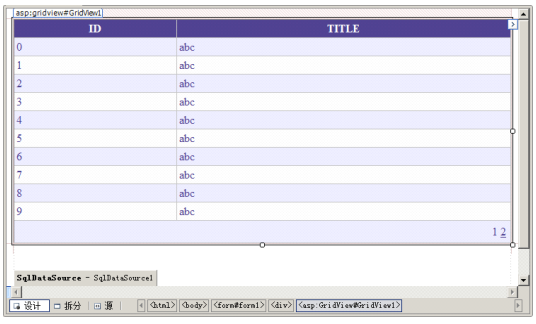

图 9-7　编辑数据绑定控件界面

开发人员能够自定义数据绑定控件的样式，并且修改某些列的顺序，这里使用了自动套用格式，并将数据绑定控件的 width 属性设置为 100%，这样编写宽度就能够适应浏览器的大小，从而随着浏览器

的大小而改变。数据绑定控件配置完成后，值得注意的是，需要勾选 SQL 语句的高级选项，让数据绑定控件支持编辑、删除和选择，如图 9-8 所示。

图 9-8　SQL 高级选项

配置 SQL 高级选项后，数据源控件就会自动生成 INSERT、UPDATE、DELETE 语句，示例代码如下所示。

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:mytableConnectionString %>"
    DeleteCommand="DELETE FROM [mynews] WHERE [ID] = @ID"
    InsertCommand="INSERT INTO [mynews] ([TITLE]) VALUES (@TITLE)"
    SelectCommand="SELECT * FROM [mynews]"
    UpdateCommand="UPDATE [mynews] SET [TITLE] = @TITLE WHERE [ID] = @ID">
    <DeleteParameters>
        <asp:Parameter Name="ID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="TITLE" Type="String" />
        <asp:Parameter Name="ID" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
        <asp:Parameter Name="TITLE" Type="String" />
    </InsertParameters>
</asp:SqlDataSource>
```

在完成用户界面的配置后，系统生成的 HTML 代码如下所示。

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True"
    AutoGenerateColumns="False" BackColor="White" BorderColor="#E7E7FF"
    BorderStyle="None" BorderWidth="1px" CellPadding="3" DataKeyNames="ID"
    DataSourceID="SqlDataSource1" GridLines="Horizontal" Width="100%">
    <FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C" />
    <RowStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" />
    <Columns>
        <asp:BoundField DataField="ID" HeaderText="ID" InsertVisible="False"
            ReadOnly="True" SortExpression="ID" />
        <asp:BoundField DataField="TITLE" HeaderText="TITLE" SortExpression="TITLE" />
    </Columns>
    <PagerStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" HorizontalAlign="Right" />
    <SelectedRowStyle BackColor="#738A9C" Font-Bold="True" ForeColor="#F7F7F7" />
    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True" ForeColor="#F7F7F7" />
    <AlternatingRowStyle BackColor="#F7F7F7" />
```

```
    </asp:GridView>
```

开发人员可以编写以上 HTML 实现更多的效果，当确定用户界面编写完毕后，就可以为数据绑定控件选择操作了。

9.6.2 使用 GridView 显示、删除、修改数据

配置完成用户界面，则需要选择 GridView 控件的属性并配置 GridView 任务，如图 9-9 和图 9-10 所示。

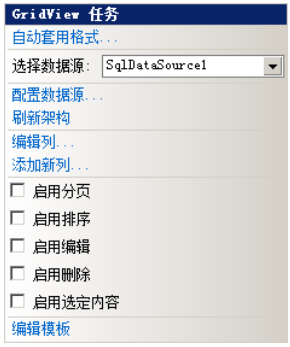

图 9-9 默认 GridView 任务

图 9-10 选择 GridView 任务

GridView 控件支持分页、排序、编辑、删除和选定内容等操作。在 GridView 控件中，首先必须勾选【分页】复选框，然后再配置 PageSize 属性才能够让 GridView 控件支持分页功能。在 GridView 控件属性中如果勾选了【分页】复选框而不配置 PageSize 属性，则默认按 10 条数据分页。勾选了以启用分页、启用编辑、启用删除和启用选定内容后，GridView 控件的界面如图 9-11 所示。

因为在数据源控件配置的过程中，已经配置了支持编辑、删除和选择，所以在数据绑定控件中可以选择启用编辑，启用删除和选定内容等操作，并且系统默认支持更新、插入、删除等操作，运行后如图 9-12 所示。

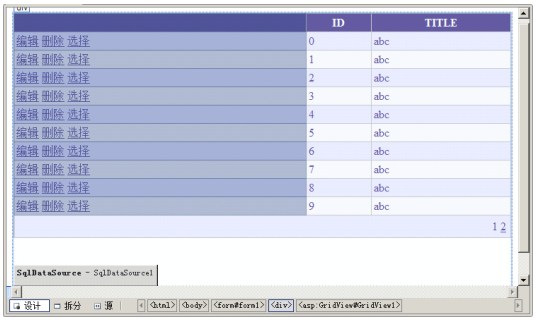

图 9-11 编辑数据绑定控件界面

图 9-12 GridView 控件显式

GridView 控件支持编辑、删除和选择，当单击编辑时，能够对选择的行进行数据编辑，如图 9-13 所示。编辑完成后，单击更新按钮则可以执行更新操作，而无需手动编写 UPDATE 操作，如图 9-14 所示。

图 9-13　编辑数据

图 9-14　执行更新操作

当单击删除时，则会执行 DELETE 命令，而无需手动编写 DELETE 命令。GridView 控件支持分页、排序、编辑、删除和选定内容，开发人员无需手动编写更新、删除、编辑、也无需手动编写分页，对GridView 控件进行缓存设置能够提高应用程序性能，在对数据库的操作，编辑及更新中，GridView 控件能够方便开发人员，简化代码。

9.6.3　使用 DataList 显示数据

DataList 控件需要编辑 HTML 模板来显式数据，虽然在开发上，DataList 控件比 GridView 更加复杂，但是 DataList 控件能够实现更多效果。相比之下，DataList 控件比 GridView 控件更加灵活，能够进行复杂的事件编写和样式控制。选择【自动套用格式】复选框并将 DataList 控件的宽度设置为 100%，编辑基本的用户界面，如图 9-15 所示。

通过编辑 ItemTemplate 能够实现自定义模板，而无需像 GridView 一样，以表格形式呈现，编辑后运行如图 9-16 所示。

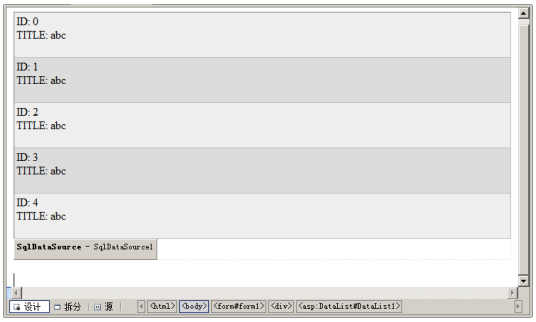

图 9-15　DataList 控件显式数据

图 9-16　编写 ItemTemplate 模板

DataList 控件执行数据操作基本上同 GridView 一样，DataList 控件与 GridView 相比只下，有着更灵活的模板方案，能够实现更多的显示效果。

9.6.4　DataList 分页实现

DataList 控件本身并不带分页实现，如果需要 DataList 能够实现分页效果，则需要通过代码实现 DataList 控件的分页。DataList 控件分页需要增加若干标签（Label）控件来显式“上一页”，“下一页”等分页所需要的连接，示例代码如下所示。

```
<asp:Label ID="Label4" runat="server" Text="Label"></asp:Label>
<asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
```

```
<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
```

上述代码创建了三个 Label 控件，这三个控件并无需初始化，这三个控件通过编程实现上一页，下一页的分页形式。如果需要执行分页，则需要编写 cs 页面代码，cs 页面代码如下所示。

```
PagedDataSource objPds = new PagedDataSource();
objPds.DataSource = this.SqlDataSource1.Select(new DataSourceSelectArguments());
objPds.AllowPaging = true;                                  //设置是否允许分页
objPds.PageSize = 3;                                        //设置分页条目数
int CurPage;                                                //设置当前页码
Label2.Visible = false;                                     //隐藏标签
Label4.Visible = false;
```

上述代码初始化 PagedDataSource 对象，并将分页控件默认初始化属性 Visible 为 false。其中 PagedDataSource 是封装分页相关属性的类。

```
if (Request.QueryString["Page"] != null)                    //如果传递的页面不为空
{
    CurPage = Convert.ToInt32(Request.QueryString["Page"]); //获取传递的参数
}
else
{
    CurPage = 1;                                            //页面的值为 1
}
objPds.CurrentPageIndex = CurPage - 1;                      //设置索引
Label2.Visible = true;                                      //显式标签
Label4.Visible = true;
Label3.Text = "<a href=\"datalist.aspx\">首页</a>";          //编写分页
Label2.Text = "<a href=\"datalist.aspx?page=" + Convert.ToString(CurPage + 1) + "\">下一页</a>";
Label4.Text = "<a href=\"datalist.aspx?page=" + Convert.ToString(CurPage - 1) + "\">上一页</a>";
```

上述代码通过传递的 Page 的值进行分页操作，如果传递的 Page 的值为不为空，则从数据源控件中读取相应的数据，并显示到数据绑定控件中。

```
if (CurPage == 1)                                           //如果只有一个页面
{
    Label4.Visible = false;                                 //隐藏标签
}
if (objPds.IsLastPage)
{
    Label2.Visible = false;
}
DataList1.DataSourceID = "";                                //重新绑定数据
DataList1.DataSource = objPds;                              //编写 DataList 的数据源
DataList1.DataBind();                                       //绑定数据源
```

上述代码通过 PagedDataSource 对象实现了分页效果，并且将分页条目数设置为 3，当数据超过 3 条时，则会实现分页。运行后如图 9-17 和图 9-18 所示。

图 9-17　实现下一页效果

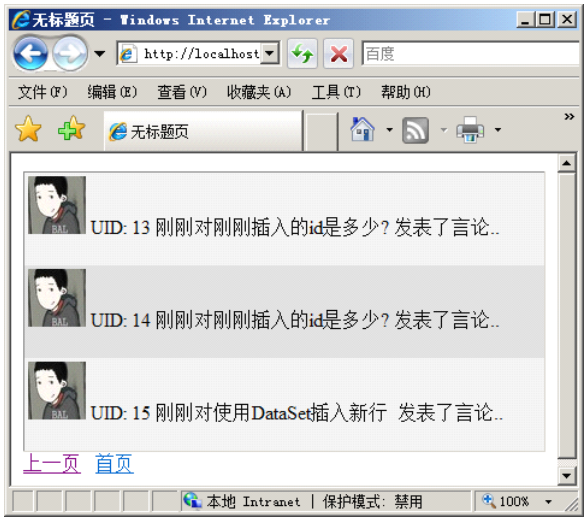

图 9-18　实现上一页效果

DataList 控件虽然不支持分页，但是能够通过编程实现 DataList 控件的分页效果。DataList 控件在模板编辑和代码开发上虽然没有 GridView 方便，但是却提高了灵活性，能够自定义分页和数据显示。

9.6.5　使用 SQLHelper 操作数据库

使用控件，能够方便开发人员的开发和使用，但是很多情况下，不能使用控件来实现，所以很多情况都需要使用 ADO.NET 操作数据库中的数据，SQLHelper 是将 ADO.NET 中对数据操作的类和对象进行的封装的一个类库，使用 SQLHelper 能够提高数据库操作的效率。

1．创建 SQLHelper

SQLHelper 类经常在数据库开发中使用，不仅封装了数据库操作，也提高了数据库操作的安全性，SQLHelper 在微软的开发中和 DEMO 中经常被使用，SQLHelper 通常用于多层设计，如果需要使用 SQLHelper 类，可以到微软官方下载最新的 SQLHelper 类，也可以自行编写 SQLHelper 类。如果自行创建 SQLHelper 类，则在解决方案管理器中新建一个类库，如图 9-19 所示。

创建类库后，删除自动生成的 Class1 类，并创建一个新类，类名为 SQLHelper，如图 9-20 所示。

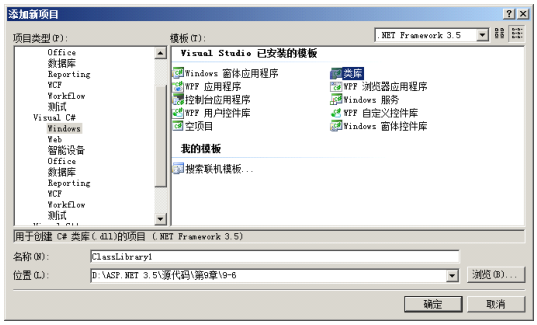

图 9-19　添加类库

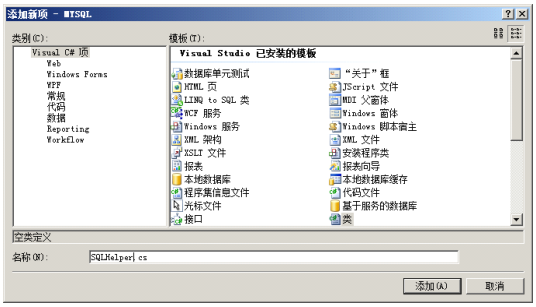

图 9-20　创建 SQLHelper 类

如果使用下载的 SQLHelper 类，则可以单击解决方案管理器，单击右键，选择添加现有项，然后选择现有项目添加即可。在 SQLHelper 类下对数据操作进行封装，开发人员能够使用自己封装的类进行数据操作，示例代码如下所示。

```
#region //数据库连接串
private static readonly string database = “数据库”;                                    //配置数据库信息
private static readonly string uid = “用户名”;                                         //配置用户名信息
private static readonly string pwd = “密码”;                                           //配置密码信息
private static readonly string server = “服务器”;                                      //配置服务器信息
private static readonly string condb = "server='"+ server +"';database='" + database + "';uid=
'" + uid + "';pwd='" + pwd + "';Max Pool Size=100000;Min Pool Size=0;
Connection Lifetime=0;packet size=32767;Connection Reset=false; async=true";//设置连接字串
#endregion
#region//DataAdapter 方法  返回 DataSet 数据集
/// <summary>
/// DataAdapter 方法  返回 DataSet 数据集
/// </summary>
/// <param name="sqlCmd">SQL 语句</param>
/// <param name="command">操作参数  枚举类型</param>
/// <returns></returns>
public static DataSet DataAdapter(string sqlCmd, SDACmd command,                       //实现适配器
string tabName, params SqlParameter[] paraList)
{
    SqlConnection con = new SqlConnection(condb);                                      //创建连接对象
    SqlCommand cmd = new SqlCommand();                                                 //创建 Command 对象
    cmd.Connection = con;                                                              //使用连接对象
    cmd.CommandText = sqlCmd;                                                          //配置连接字串
    if (paraList != null)
    {
        cmd.CommandType = CommandType.Text;                                            //配置 Command 类型
        foreach (SqlParameter para in paraList)                                        //遍历参数
        { cmd.Parameters.Add(para); }                                                  //添加参数
    }
    SqlDataAdapter sda = new SqlDataAdapter();                                         //创建适配器
    switch (command)                                                                   //查找条件
    {
        case SDACmd.select:                                                            //如果为 select 执行
            sda.SelectCommand = cmd;
            break;
        case SDACmd.insert:                                                            //如果为 insert 执行
            sda.InsertCommand = cmd;
            break;
        case SDACmd.update:                                                            //如果为 update 执行
            sda.UpdateCommand = cmd;
            break;
        case SDACmd.delete:                                                            //如果为 delete 执行
            sda.DeleteCommand = cmd;
            break;
    }
    DataSet ds = new DataSet();                                                        //创建数据集
    sda.Fill(ds, tabName);                                                             //填充数据集
    return ds;                                                                         //返回数据集
}
```

在上述代码中，还需要通过一个枚举类型进行 switch 操作，枚举类型用于判断执行的操作，示例代

码如下所示。

```
public enum SDACmd { select, delete, update, insert }                    //定义枚举类型
```

定义的枚举类型用于在程序中进行筛选操作，用于指定 SQL 语句执行的操作。在 SQLHelper 类中，还需要封装 DataReader 方法进行 DataReader 的封装和实现，开发人员能够使用 SQLHelper 类中的 DataReader 方法进行数据库的读取，示例代码如下所示。

```
public static SqlDataReader ExecReader(string sqlcmd, params SqlParameter[] paraList)
{
    SqlConnection con = new SqlConnection(condb);                        //创建连接对象
    SqlCommand cmd = new SqlCommand();                                   //创建 Command 对象
    cmd.Connection = con;                                                //使用连接
    cmd.CommandText = sqlcmd;                                            //配置 SQL 语句
    if (paraList != null)
    {
        cmd.CommandType = CommandType.Text;                              //配置 Command 类型
        foreach (SqlParameter para in paraList)
        { cmd.Parameters.Add(para); }                                    //添加参数
    }
    con.Open();                                                          //打开连接
    SqlDataReader sdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
    return sdr;
}
```

上述代码实现了 DataReader 对象，使用 DataReader 能够填充 SqlDataReader 对象并进行数据的循环输出。在 ADO.NET 中，通常需要执行 SQL 语句进行数据库的操作，在 SQLHelper 类中，同样需要封装执行 SQL 语句的操作以便能够快速执行数据操作。

```
public static void ExecNonQuery(string sqlcmd, params SqlParameter[] paraList)
{
    using (SqlConnection con = new SqlConnection(condb))                 //创建连接对象
    {
        SqlCommand cmd = new SqlCommand();                               //创建 Command 对象
        cmd.Connection = con;                                            //使用连接
        cmd.CommandText = sqlcmd;                                        //配置执行类型
        if (paraList != null)
        {
            cmd.CommandType = CommandType.Text;                          //配置执行类型
            foreach (SqlParameter para in paraList)
            { cmd.Parameters.Add(para); }                                //添加参数
        }
        con.Open();                                                      //打开数据连接
        cmd.ExecuteNonQuery();                                           //执行 SQL 语句
    }
}
```

上述代码编写了 SQLHelper 类操作数据库的函数，通过执行函数并传递参数，即可实现数据库的插入、更新和删除。

2. 使用 SQLHelper

创建完成 SQLHelper 类后，需要为应用程序配置 SQLHelper 的基本属性，代码如下所示。

```
private static readonly string database = "mytable";                    //配置数据库
private static readonly string uid = "sa";                               //配置用户名
```

```
        private static readonly string pwd ="sa";                                    //配置用户会密码
        private static readonly string server = "local";                             //配置服务器的值
```

上述代码为 SQLHelper 类配置了属性，当使用 SQLHelper 类时，系统会自动连接数据库，在完成使用后，系统会自动关闭数据库。如果需要在当前项目使用 SQLHelper 类，则需要添加引用来使用 SQLHelper 类，右击现有项目，在下拉菜单中选择【添加】选项，在【添加】选项中选择【现有项】选项，在弹出窗口中选择【项目】标签栏，就可以添加相同项目的类库，如图 9-21 所示。

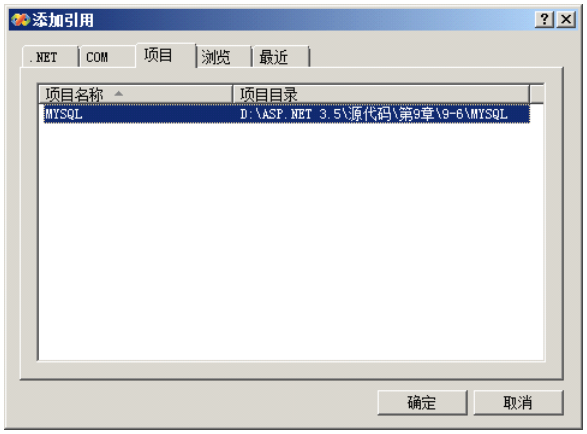

图 9-21　添加引用

引用添加完毕，在使用 SQLHelper 页面的 CS 页面中，需要添加命名空间，命名空间的名称和创建类库的名称相同，如果需要更改名称，可以通过修改类库的属性来修改。示例代码如下所示。

```
using MYSQL;
```

引用完毕后，就可以执行 SQL 语句，使用 SQLHelper 执行 SQL 语句非常方便，下面代码演示了如何执行插入、删除操作。

```
        string strsql = "insert into mynews values ('SQLHelper 插入标题')";    //编写 SQL 语句
        SQLHelper.ExecNonQuery(strsql);                                          //执行 SQL 语句
```

上述代码运行后，则会执行插入操作，相比于 ADO.NET，封装后的代码更加简便易懂，删除操作代码如下所示。

```
        string strsql2 = "delete form mynews where id=3";                        //编写 SQL 语句
        SQLHelper.ExecNonQuery(strsql2);                                         //执行 SQL 语句
```

当需要执行 SELECT 语句时，可以通过 SQLHelper.DataAdapter 获取数据，示例代码如下所示。

```
        string strsql = "select * from mynews where id=3";                       //编写 SQL 语句
        DataSet ds = SQLHelper.DataAdapter(strsql, SQLHelper.SDACmd.select, "mydatatable");
```

上述代码通过 SQLHelper.DataAdapter 获取数据，并创建了一个 mydatatable 虚拟表，填充 DataSet 对象。当需要获取 DataSet 对象中的数据时，和普通的 DataSet 对象一样。SQLHelper 封装了 ADO.NET 中的许多方法，为开发人员提高了效率，同时也增加了安全性和模块化的特性。

注意：上述代码中的 SQLHelper 类能够执行的是 SQL 语句，如果需要执行存储过程，则需要更改 SQLHelper 类中的相应的 CommandType 属性的值。

9.7　小结

本章介绍了 ADO.NET 中操作数据库和执行数据库的一些方法，还介绍了如何编写和执行 SQL 语句，包括 SQL INSERT、SQL UPDATE、SQL DELETE 等数据操作语句，另外，本章还介绍了如何通过 DataSet

数据集实现插入、更新、删除等操作来深入了解 ADO.NET。本章通过演示使用控件更新和操作数据库，加强了控件操作数据库的示例，本章还包括：

- 使用 ADO.NET 操作数据库，介绍了 ADO.NET 操作数据库的方法。
- ASP.NET 创建和插入记录，介绍了 SQL INSERT 和数据操作。
- ASP.NET 更新数据库，介绍了 SQL UPDATE 和数据操作。
- ASP.NET 删除数据，介绍了 SQL DELETE 和数据操作。
- 使用存储过程，介绍了如何使用存储过程。
- 数据库操作实例，介绍了企业应用中常用的 SQLHelper，以及用户操作数据库的配置。

本章还介绍了如何创建和使用企业应用中常用的 SQLHelper 类、SQLHelper 类能够简化数据使用，提高开发效率。下一章中将会讲解如何连接其他数据库。