10.2 Connection 与 Command 对象的使用

10.2.1 Connection 对象

在 ADO.NET 中，Connection（连接对象）用于连接数据库，是应用程序访问和使用数据源数据的桥梁。表 10-5 列出了 Connection 的主要成员。

表 10-5 Connection 的主要成员

属性或方法	说　明
ConnectionString	连接字符串
Open()	打开数据库连接
Close()	关闭数据库连接

使用 Connection 对象连接数据库的一般步骤如下：

（1）定义连接字符串。

连接字符串用来描述数据源的连接方式，不同的数据源使用不同的连接字符串。在定义连接字符串时必须参考数据源的帮助手册。以 SQL Server 为例，它既支持 SQL Server 身份验证的连接方式，也支持 Windows 集成身份验证的连接方式。

其中，在 SQL Server 身份验证方式中连接字符串的一般格式如下：

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

在 Windows 身份验证方式中连接字符串的一般格式如下：

```
string connString = "Data Source = 服务器名; Initial Catalog = 数据库名; Integrated Security = True";
```

其中，“服务器名”是数据库的服务器名称或 IP 地址。当应用程序与 SQL Server 服务器在同一台计算机上运行时，SQL Server 服务器是本地服务器，其服务器名可以有以下几种写法：.（圆点）、(local)、127.0.0.1、本地服务器名称。

扫描全能王 3亿人都在用的扫描App

(2) 创建 Connection 对象。

```
SqlConnection conn = new SqlConnection(connString);
```

(3) 打开与数据库的连接。

```
conn.Open();
```

(4) 使用该连接进行数据访问。

(5) 关闭与数据库的连接。

```
conn.Close();
```

【注意】 不同数据提供者的连接对象及其命令空间是不相同的。表 10-6 列出了不同命名空间的 Connection 对象。

表 10-6 不同命名空间的 Connection 对象

命名空间	对应的 Connection 对象	命名空间	对应的 Connection 对象
System.Data.SqlClient	SqlConnection	System.Data.Odbc	OdbcConnection
System.Data.OleDb	OleDbConnection	System.Data.OracleClient	OracleConnection

10.2.2 Command 对象

Command(命令对象)用于封装和执行 SQL 命令并从数据源中返回结果,命令对象的 CommandText 属性用来保存最终由数据库管理系统执行的 SQL 语句。对于不同的数据源需要使用不同的命令对象。例如,用于 SQL Server 的命令对象为 SqlCommand,用于 ODBC 的命令对象为 OdbcCommand,用于 OLE DB 的命令对象为 OleDbCommand。表 10-7 列出了 Command 的主要成员。

表 10-7 Command 的主要成员

属性或方法	说明
Connection	Command 对象使用的数据库连接
CommandText	执行的 SQL 语句
ExecuteNonQuery()	执行不返回行的语句,如 UPDATE 等,执行后返回受影响的行数
ExecuteReader()	返回 DataReader 对象
ExecuteScalar()	执行查询,并返回查询结果集中第一行的第一列

虽然不同数据源的命令对象的名字略有不同,但使用方法是相同的,通常按以下步骤访问数据库源。

(1) 创建数据库连接。

(2) 定义 SQL 语句。

(3) 创建 Command 对象,一般形式如下:

```
SqlCommand comm = new SqlCommand(SQL 语句, 数据库连接对象);
```

也可以采用以下形式创建 Command 对象。

扫描全能王 3亿人都在用的扫描App

```
SqlCommand comm = new SqlCommand();
comm.Connection = 数据库连接对象;
comm.CommandText = "SQL 语句";
```

(4) 执行命令。

【注意】 在执行命令前，必须打开数据库连接，执行命令后，应该关闭数据库连接。

10.2.3 应用实例——实现用户登录

【例 10-1】 在项目 MyAccounting 中，连接 Financing 数据库，使用 Command 的 ExecuteScalar()方法完成用户登录功能。Financing 数据库的用户表(User)的结构如表 10-8 所示。

表 10-8 User 表结构

字 段 名	类 型	其他属性	说 明
UserId	int	主键，标识列，非空	用户编号
UserName	nchar(20)	非空	用户名
Password	nvarchar(20)	非空	密码

【操作步骤】

(1) 启动 VS2017，打开在本书第 9 章中创建的 Windows 应用程序 MyAccounting。

(2) 在 Login.cs 的代码视图中，添加以下命名空间的引用。

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

(3) 双击“登录”按钮，更新该按钮的 Click 事件方法，实现用户登录验证，代码如下。

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string userName = txtName.Text.Trim();          //Trim()用于去除文本框中的前后空格
    string password = txtPwd.Text.Trim();
    string connString = "Data Source = .; Initial Catalog = Financing; User ID = sa; Pwd =
123456";
    SqlConnection conn = new SqlConnection(connString);     //创建 Connection 对象
    //获取用户名和密码匹配的行数的 SQL 语句
    string sql = String.Format("select count(*) from User where UserName = '{0}' and Password = '{1}
'", userName, password);
    try
    {
        conn.Open();                                        //打开数据库连接
        SqlCommand comm = new SqlCommand(sql, conn); //创建 Command 对象
        int num = (int)comm.ExecuteScalar();            //执行查询语句，返回匹配的行数
        if (num == 1)                                   //在 User 表中指定用户名和密码的记录
                                                        //最多只有一行
        {
            //如果有匹配的行，则表明用户名和密码正确
            MessageBox.Show("欢迎进入个人理账系统!", "登录成功", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            MainFrm mainForm = new MainFrm();           //创建主窗体对象
```

```
                mainForm.Show();                                    //显示主窗体
                this.Visible = false;                               //登录窗体隐藏
            }
            else
            {
                txtPwd.Text = "";
                MessageBox.Show("您输入的用户名或密码错误!", "登录失败", MessageBoxIcon.
OK, MessageBoxIcon.Exclamation);
            }
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message, "操作数据库出错!", MessageBoxButtons.OK, MessageBoxIcon.
Exclamation);
        }
        finally
        {
            conn.Close();                                           //关闭数据库连接
        }
    }
```

(4) 在解决方案资源管理器中双击 Program.cs 文件,将 Main()方法中的最后一行代码改为

```
Application.Run(new Login());
```

【分析】 本程序的数据库服务器是 SQL Server,所使用的数据源是根据表 10-2、表 10-3、表 10-4 和表 10-5 创建的 Financing 数据库。本程序采用 SQL Server 身份验证方式连接数据源,并且直接使用 SQL Server 的系统管理账户(sa),不过建议读者视情况灵活设置连接字符串。本程序调用了 SqlCommand 对象的 ExecuteScalar()方法来执行数据库的查询,并返回查询结果集中第一行的第一列数据,所执行的 SQL 命令"select count(*) from [User] where UserName='……' and Password='……'"将返回符合条件的数据记录的行数,由于指定用户名和密码的记录最多只有 1 行,因此如果用户输入的用户名和密码正确,则返回结果为 1,否则为 0,检查该值,就可以判断是否成功登录。执行本程序时,当用户成功登录之后,系统将弹出个人理财系统的主窗体。程序运行效果如图 10-3 所示。

图 10-3 "用户登录"运行效果

扫描全能王
3亿人都在用的扫描App

10.3 DataReader 对象的使用

10.3.1 DataReader 对象

DataReader(数据读取对象)提供一种从数据库只向前读取行的方式。表 10-9 列出了 DataReader 的主要成员。

表 10-9 DataReader 的主要成员

属性或方法	说　明
HasRows	DataReader 中是否包含一行或多行
Read()	前进到下一行记录,如果下一行有记录,则读出该行并返回 true;否则返回 false
Close()	关闭 DataReader 对象

使用 DataReader 检索数据的步骤如下。

(1) 创建 Command 对象。

(2) 调用 Command 对象的 ExecuteReader()方法创建 DataReader 对象。

```
SqlDataReader dr = Command 对象.ExecuteReader();
```

【注意】 DataReader 类在.NET Framework 中被定义为抽象类,因此不能直接实例化,只能使用 Command 对象的 ExecuteReader()方法来创建 DataReader 对象。

(3) 调用 DataReader 对象的 Read()方法逐行读取数据。

```
while (dr.Read())
{
```

扫描全能王 3亿人都在用的扫描App

```
    //读取某列数据
)
```

(4) 读取某列的数据

获取某列的值，可以指定列的索引，从 0 开始，也可以指定列名，一般形式如下

```
(数据类型)dr[索引或列名]
```

【注意】 在执行程序时，DataReader 对象的 Read 方法会自动把所读取的一行数据的每个列值通过装箱操作保存到 DataReader 内部的集合中，因此要想获得某一列的值，就必须进行拆箱操作，即强制类型转换。

(5) 关闭 DataReader 对象。

```
dr.Close();
```

CS 扫描全能王 3亿人都在用的扫描App

10.4 DataSet 与 DataAdapter 对象的使用

10.4.1 DataSet 与 DataAdapter 对象

DataReader(数据读取对象)是一种快速的、轻量的、单向只进的数据访问对象,结合命令对象,可以较快地查询和修改少量的数据,但当数据量较大,想要大批量地查询和修改数据,或者想在断开数据库连接的情况下操作数据时,DataReader 就无法做到了,这时可以使用 DataSet(数据集对象)。

DataSet 可以简单理解为一个临时数据库,DataSet 将数据源的数据保存在内存中并独立于任何数据库,此时 DataSet 中的数据相当于数据源的数据的一个副本,应用程序与内存中的 DataSet 数据进行交互,在交互期间不需要连接数据源,因此可以极大地加快数据访问和处理速度,同时也节约了资源。这一切就像在现实生活中工厂仓库、临时仓库与生产线的关系。生产线上需要的材料和生产的成品都来自于或存放在工厂仓库,但如果频繁同仓库交互会降低效率,可以在生产线和工厂仓库之间建立一个临时仓库,存储常用的材料成品,这样可以大大地加快生产的速度。这里的生产线就是应用程序,临时仓库就是 DataSet,而工厂仓库就是数据源。

DataSet 保存了从数据源读取的数据信息,以 DataTable 为单位,自动维护表间关系和数据约束。DataSet 的基本结构如图 10-6 所示。其中,DataTable 是数据源执行一次 SELECT 操作得到的,它同数据源中表的区别在于,它可能是数据源的某个表的所有数据或部分数据,也可能是对数据源的多个表进行联合查询得到的结果。DataSet 内部的所有 DataTable 构成 DataTableCollection 对象。每一个 DataTable 由一个 DataColumnCollection 对象和一个 DataRowCollection 对象组成,前者是表的所有列组成的集合,后者是表的所有行组成的集合。DataTable 还保存数据的当前状态,根据 DataTable 的状态就可知道数据是否被更新或被删除。

DataSet 内部的各个 DataTable 之间的关系是通过 DataRelation 来表达,这些 DataRelation 形成一个集合,称为 DataRelationCollection。每个 DataRelation 表示表之间

扫描全能王 3亿人都在用的扫描App

的主键—外键关系。当两个 DataTable 存在主键—外键关系时，只要其中的一个表的记录指针移动，另一个表的记录指针也将随之移动；当一个表的记录更新时，如果不满足主键—外键约束，则更新就会失败。例如，假设收支项目 Item 表与收支明细 List 表存在主键—外键约束，当收支明细表包含了"餐饮"方面的支出性的数据记录，此时试图删除收支项目表中的与"餐饮"对应的数据记录就会失败。

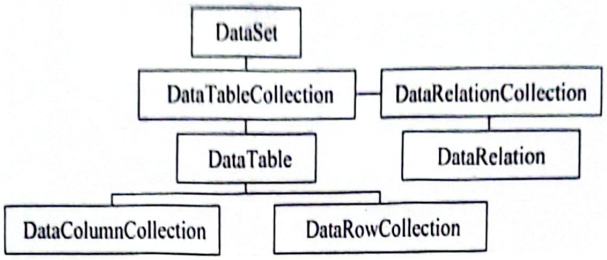

图 10-6　DataSet 的基本结构

图 10-7 显示了数据集的工作原理：客户端向数据库服务器请求数据后，数据库服务器会从数据库中将数据发送给 DataSet，由 DataSet 存储这些数据，并在需要时将数据传递给客户端。客户端对数据进行修改后，先将修改后的数据放入 DataSet 中，然后统一由 DataSet 将修改后的数据提交到数据库服务器中。

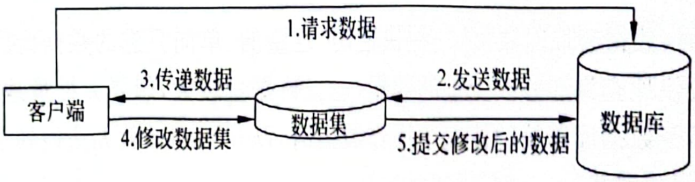

图 10-7　数据集的工作原理

要使用 DataSet，必须先创建 DataSet 对象。在创建 DataSet 对象时可以指定一个数据集的名称，如果不指定名称，则默认为 NewDataSet，一般形式如下。

```
DataSet 数据集对象 = new DataSet("数据集的名称字符串");
```

其中，数据集的名称字符串可以省略。

例如：

```
DataSet dst = new DataSet();                    //创建一个名为"NewDataSet"的数据集
DataSet ds = new DataSet("MyData");             //创建一个名为"MyData"的数据集
```

接下来，需要在数据集和数据源之建立一个桥梁，用于将数据源中的数据放入数据集中，当数据集中的数据发生改变时能够把修改后的数据再次回传到数据源中。DataAdapter 是 DataSet 和数据源之间的桥接器，用于检索和保存数据。这如同在工厂仓库和临时仓库之间先建立一个道路，用一个大货车在两个仓库之间来回运送货物一样，这里的仓库与临时仓库之间的路相当于数据库连接，而运货车相当于数据适配器 DataAdapter。

在使用不同数据提供程序的 DataAdapter 对象时，对应的命名空间不同。表 10-10 列出了不同命名空间的 DataAdapter 对象。

扫描全能王
3亿人都在用的扫描App

表 10-10 不同命名空间的 DataAdapter 对象

命名空间	对应的 DataAdapter 对象
System. Data. SqlClient	SqlDataAdapter
System. Data. OleDb	OleDbDataAdapter
System. Data. Odbc	OdbcDataAdapter
System. Data. OracleClient	OracleDataAdapter

表 10-11 列出了 DataAdapter 的主要成员。

表 10-11 DataAdapter 的主要成员

属性或方法	说　明
SelectCommand	从数据库检索数据的 Command 对象，该对象封装了 SQL 的 SELECT 语句
InsertCommand	向数据库插入数据的 Command 对象，该对象封装了 SQL 的 INSERT 语句
UpdateCommand	修改数据库中数据记录的 Command 对象，该对象封装了 SQL 的 UPDATE 语句
DeleteCommand	删除数据库中的数据记录的 Command 对象，该对象封装了 SQL 的 DELETE 语句
Fill()	向 DataSet 对象填充数据
Update()	将 DataSet 中的数据提交到数据库

使用 DataAdapter 对象填充数据集时，先使用 Connection 连接数据源，然后使用 Fill() 方法填充 DataSet 中的表，一般格式如下。

（1）创建 SqlDataAdapter 对象。

```
SqlDataAdapter 对象名 = new SqlDataAdapter(SQL 语句, 数据库连接);
```

（2）填充 DataSet。

```
DataAdapter 对象. Fill(数据集对象, "数据表名");
```

把数据集中修改过的数据提交到数据源时，需要使用 Update()方法，在调用 Update() 方法前，要先设置需要的相关命令，包括 INSERT、UPDATE 和 DELETE 命令，可通过表 10-11 的相关的属性完成设置，也可以使用 SqlCommandBuilder 对象来自动生成更新所需要的相关命令，简化操作，步骤如下。

（1）自动生成用于更新的相关命令。

```
SqlCommandBuilder builder = new SqlCommandBuilder(已创建的 DataAdapter 对象);
```

（2）将 DataSet 的数据提交到数据源。

```
DataAdapter 对象. Update(数据集对象, "数据表名称字符串");
```

扫描全能王
3亿人都在用的扫描App