

第3章 三层架构项目开发实战

第2章学习的信息管理系统可以理解为二层架构的项目,由用户(界面)直接访问数据库,这种项目运行效率高,但是如果用户需求发生改变,那么需要改动的地方往往会很多,甚至项目需要重新开发,也就是说,这种架构的项目不利于后续的维护。本章学习三层架构项目,三层架构项目有利于后续维护。

学习目标

- (1) 能够理解三层架构的思想。
- (2) 能够熟练开发三层架构的项目。

思政目标及设计建议

根据新时代软件工程师应该具备的基本素养,挖掘课程思政元素,有机融入教学中,本章思政目标及设计建议如表3-1所示。

表 3-1 第3章思政目标及设计建议

思政目标	思政元素及融入
培养学生职业道德和职业素养	讲解项目为什么要分层时类比到随着企业的发展壮大,也要分层,以更好地对企业进行管理。三层架构的项目层与层之间访问是有严格顺序的,启发学生在学习工作中有问题要先找直接领导,一般不宜越级找领导,培养学生职业道德和职业素养
培养自主探索、敬业、专注的工匠精神	通过课前自主学习,培养自主探索、敬业、专注的工匠精神

3.1 三层架构的基础知识

3.1.1 三层架构的理解和作用

在生活中经常会见到分层的现象,例如,公司人员结构会分层,楼房是分层的,甚至做包子的笼屉都是分层的。虽然分层的目的各有不同,但都是为解决某一问题而产生的。所以,分层架构其实是为了解决某一问题而产生的一种解决方案。

从社会的发展来看,社会分工是人类进步的表现。社会分工的优势就是让适合的人做自己喜欢的事情,使平均社会劳动时间大大缩短。



程序三层架构就是在项目开发过程中根据代码的不同功能,分别对代码进行存储与调用。通常分为表现层、业务逻辑层、数据访问层,这三层如何理解?我们过去大饭店吃饭来形象的理解。如果到小餐馆吃饭,可能直接叫老板给我们炒菜,最后也是老板自己收钱,这种情况可以理解为二层架构。如果到大饭店吃饭,我们就是把需求(要点的菜)告诉服务员,由服务员将顾客的需求转交到厨房,然后由厨师做出相应的菜肴,并由服务员送到顾客餐桌上。这一过程中,厨师、服务员和顾客这三个角色可以分别表示为数据访问层、业务逻辑层和表现层,如图 3-1 所示。在这个过程中,根据顾客的需求不同,更换服务员和厨师都不影响顾客的需求,同样在项目开发中也是这样。

表现层(User Interface Layer,UIL): 主要用于存放与用户交互的展示页面。主要实现和用户的交互,接收用户请求或返回用户请求的数据结果的展现,而具体的数据处理则交给业务逻辑层和数据访问层去处理。

业务逻辑层(Business Logic Layer,BLL): 主要用于存放针对具体问题对数据进行逻辑处理的代码,起到承上启下的作用。

数据访问层(Data Access Layer,DAL): 主要用于存放对原始数据进行操作的代码,它封装了所有与数据库交互的操作,并为业务逻辑层提供数据服务。

因此,三层架构可以用图 3-2 表示,这也是一种最简易的三层架构。

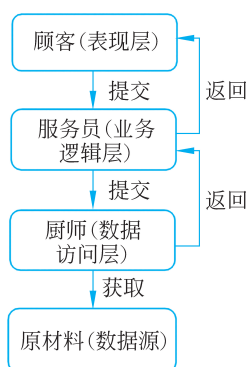


图 3-1 三层架构形象比喻



图 3-2 简易三层架构图

实际开发中,很多情况下为了复用一些共同的“东西”,会把各层都用的“东西”抽象出来。例如,将数据对象实体分离,以便在多个层中传递,称为实体类(Model)。一些共性的通用辅助类和工具方法,如数据校验、生成验证码、加解密处理等,为了让各个层之间复用,也单独分离出来,作为独立的模块使用,称为通用类库(Common)。这时三层架构可以用图 3-3 表示。

说明:

(1) 什么是业务实体(Model)?

用于封装实体类数据结构,一般用于映射数据库的数据表或视图,用以描述业务中客观存在的对象。Model 分离出来是为了更好地解耦,为了更好地发挥分层的作用,更好地进行复用和扩展,增强灵活性。

业务实体也称为实体类或实体模型类,通常类名与映射的数据库的数据表名称一致,该类中包含一系列属性,这些属性与数据库中的字段一一对应,从数据库中查询出来的数据都

使用该类的对象来保存,以便在程序中使用。

(2) 为了使数据访问层的代码更加简洁及操作数据库的代码复用,通常数据访问层会分离出通用数据访问类,即第 2 章封装的 SqlHelper 工具类。此时就形成了一个完整经典的三层架构图,如图 3-4 所示。

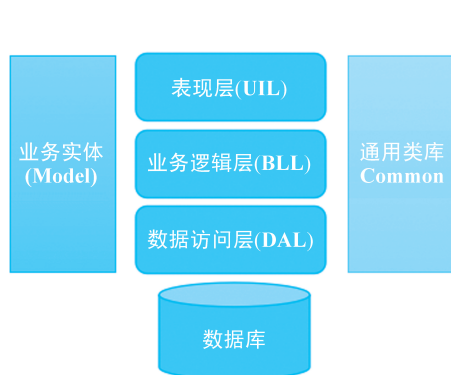


图 3-3 三层架构演变图

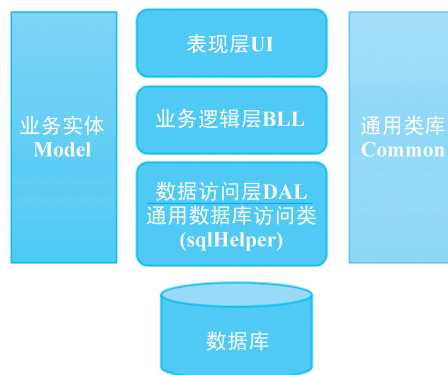


图 3-4 完整经典的三层架构图

3.1.2 三层架构的优缺点

三层架构是一种通用的项目开发方式,可以极大地提高项目的可扩展性和可维护性。但是使用三层架构也会存在一些缺陷,实际开发中根据项目的大小及客户的需求综合考虑是否选择三层架构。三层架构的优缺点如表 3-2 所示。

表 3-2 三层架构的优缺点

优 点	缺 点
代码结构清晰	增加了开发成本
耦合度降低,可维护性和可扩展性提高	降低了系统的性能
适应需求的变化,降低维护的成本和时间	在表现层中增加一个功能,为保证其设计符合分层式结构,就需要在相应的业务逻辑层和数据访问层中都增加相应的代码

3.2 三层架构项目实战——登录设计与实现

在学习完三层架构的基础知识后,接下来根据前面所学的知识结合 ADO.NET 以及简单的 WebForm 窗体来实现一个三层架构的学生信息管理系统,本节来完成三层架构的登录功能。

3.2.1 创建数据库

在 ASPNETDemoDataBase 数据库下创建一个名为 UserLogin 的用户登录表,该表的结构设计如图 3-5 所示。其中,ID 为自增的主键。

为了后面的测试,为 UserLogin 表输入一条记录,如 UserName 为 admin,Pwd 为 123456。

	列名	数据类型	允许 Null 值
🔑	ID	int	☐
	UserName	nvarchar(20)	☐
	Pwd	varchar(20)	☐

图 3-5 UserLogin 表结构

3.2.2 搭建三层架构的基本结构

三层架构的项目数据库访问层、业务逻辑层、实体模型层项目类型都要为类库(.NET Framework),表现层根据项目的需要可以为 WebForm 项目、Windows 窗体应用程序等。

1. 创建解决方案及数据访问层(DAL)

打开 Visual Studio 2022,创建新项目,项目模板选择类库(.NET Framework),如图 3-6 所示。



图 3-6 项目模板选择类库(.NET Framework)

单击“下一步”按钮,打开如图 3-7 所示的“配置新项目”窗口,在此设置解决方案名称、项目名称,此项目作为数据库访问层,因此这里项目名称设置为 DAL,实际开发中数据访问层项目名称一般以 DAL 作为后缀,但是前面可以加解决方案名称之类的。

之后单击“创建”按钮,稍后即可创建好解决方案及项目 DAL,如图 3-8 所示。

2. 创建业务逻辑层(BLL)

在图 3-8 界面中右击解决方案,选择“添加”→“新建项目”,弹出添加新项目的模板,继续选择“类库(.NET Framework)”,单击“下一步”按钮之后在弹出的界面中设置项目名称为 BLL,作为业务逻辑层。实际开发中,业务逻辑层项目的名称一般以 BLL 作为后缀,但是前面可以加解决方案名称之类的。

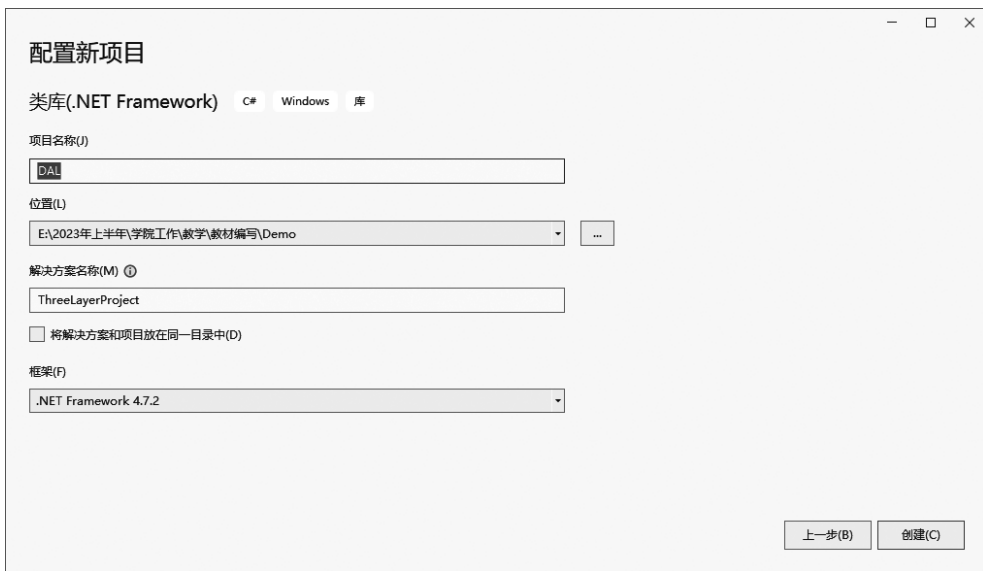


图 3-7 设置解决方案名称及项目名称

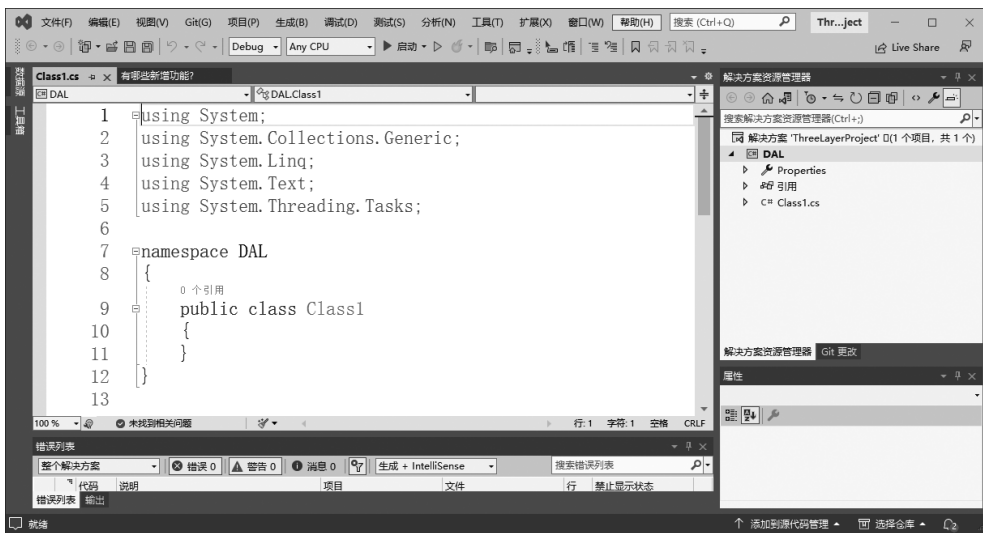


图 3-8 创建了 DAL 项目的 Visual Studio 界面

3. 创建实体模型层 (Model)

在图 3-8 界面中右击解决方案,选择“添加”→“新建项目”,弹出添加新项目的模板,继续选择“类库(.NET Framework)”,单击“下一步”按钮之后在弹出的界面中设置项目名称为 Model,作为实体模型层。实际开发中,实体模型层项目的名称一般就叫 Model。

4. 创建表现层 (UI)

在图 3-8 界面中右击解决方案,选择“添加”→“新建项目”,弹出添加新项目的模板,选择“ASP.NET Web 应用程序(.NET Framework)”,如图 3-9 所示。单击“下一步”按钮,配置项目名称为 UI,然后单击“下一步”按钮,打开如图 3-10 所示的对话框,这里选择“空”,即创建空的 ASP.NET Web 应用程序。



图 3-9 创建 ASP.NET Web 应用程序(.NET Framework)模板项目



图 3-10 创建空的 ASP.NET Web 应用程序

至此,三层架构的项目基本结构就搭建好了。

3.2.3 添加各层之间的引用

三层架构各项目之间有严格的访问关系。表现层(UI)只能访问实体模型层(Model)和业务逻辑层(BLL);业务逻辑层(BLL)只能访问实体模型层(Model)和数据访问层(DAL);数据访问层(DAL)只能访问实体模型层(Model)。下面添加各层之间的引用。

(1) 展开解决方案下面的 UI 层项目,右击“引用”,选择“添加引用”,弹出如图 3-11 所示对话框,在左侧选择项目,然后在右侧勾选 BLL 和 Model,就表示 UI 层可以访问 BLL 和 Model 层。单击“确定”按钮之后可以看到引用下面多了对 BLL 和 Model 程序集的引用。

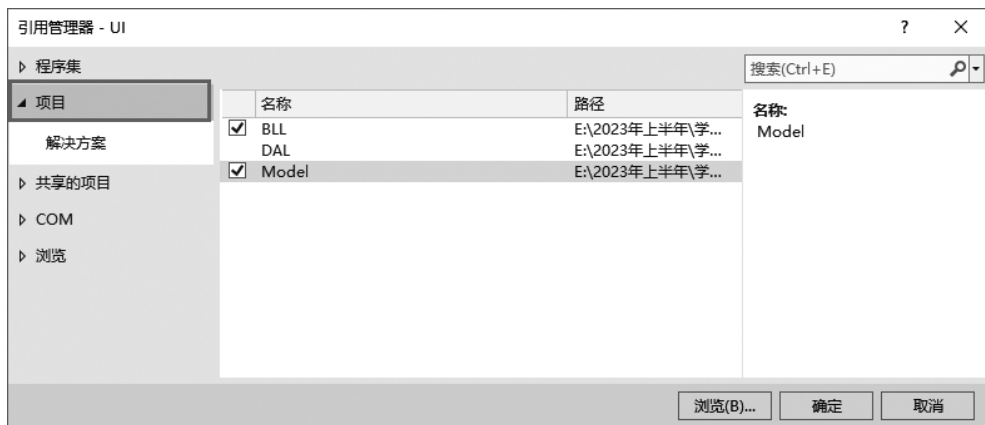


图 3-11 为 UI 层添加引用

(2) 同理展开解决方案下面的 BLL 项目,右击“引用”,选择“添加引用”,在弹出的对话框左侧选择项目,然后在右侧勾选 DAL 和 Model,就表示 BLL 可以访问 DAL 和 Model 层。单击“确定”按钮之后可以看到引用下面多了对 DAL 和 Model 程序集的引用。

(3) 同理展开解决方案下面的 DAL 项目,右击“引用”,选择“添加引用”,在弹出的对话框左侧选择项目,然后在右侧勾选 Model,就表示 DAL 可以访问 Model 层。单击“确定”按钮之后可以看到引用下面多了对 Model 程序集的引用。

3.2.4 编写实体模型层 Model 代码

实体模型层就是把数据表转换成对应的实体类,本节只用到 UserLogin 表,即只要把该表转换成实体类。在 Model 层下添加一个类,类名一般与数据表名称相同,即 UserLogin.cs。

说明:原有的默认类 Class1.cs 一般删除。

根据数据表 UserLogin 的字段在 UserLogin.cs 中添加对应的属性。同时把类的修饰符改为 public,方便其他项目调用。UserLogin 代码如下。

```
public class UserLogin
{
    public int ID { get; set; }
    public string UserName { get; set; }
    public string Pwd { get; set; }
}
```




3.2.5 编写数据访问层代码

1. 添加 SqlHelper 工具类到 DAL 项目下

为了提高编写代码效率及操作数据库代码的复用,从而使代码也更加简洁,数据访问层一般需要把访问数据库操作的工具类 SqlHelper.cs 添加进来。直接复制第2章完成的 SqlHelper 工具类到 DAL 项目下。之后还要做以下两个处理。

1) 为 SqlHelper 工具类添加引用 System.Configuration

即打开复制过来的 SqlHelper 工具类,然后安装 System.Configuration.ConfigurationManager 包,之后会自动添加引用 System.Configuration。

2) 为项目配置文件添加连接数据库信息

即打开 UI 层下的 web.config 文件,在<configuration>节中添加下面的代码。

```
<connectionStrings>
  <add name="connectionStr" connectionString="server=.;uid=sa;pwd=123456;
  database=ASPNETDemoDataBase"/>
</connectionStrings>
```

同时检查下<add>标签下的 name 属性值与 SqlHelper 工具类下加粗名称是否相同,需要保持一致。

```
public static string constr = ConfigurationManager.ConnectionStrings
["connectionStr"].ConnectionString;
```

2. 添加数据访问层类

在 DAL 下添加一个类,类名一般以 DAL 作为结尾,前面部分一般是表名称,本数据访问层主要是针对 UserLogin 表进行操作,所以这里类名取名为 UserLoginDAL.cs。

说明:原有的默认类 Class1.cs 一般删除。

接下来在该类中添加对 UserLogin 表进行操作的方法。因为本节要实现的是登录功能,所以这里编写一个根据用户名去查找用户的方法,返回值为 UserLogin 对象,具体代码如下。

```
public UserLogin SelectUserLogin(string username)
{
    string sql = "select * from UserLogin where userName=@username";
    UserLogin user = null;
    using (SqlDataReader reader = SqlHelper.ExecuteReader(sql, new
    SqlParameter("@username", username)))
    {
        if (reader.Read())
        {
            user = new UserLogin();
            user.ID = reader.GetInt32(0);
            user.UserName = reader.GetString(1);
            user.Pwd = reader.GetString(2);
        }
    }
}
```




```
        return user;  
    }  
}
```

提示：在上述类中需要添加实体类 UserLogin 所在命名空间，如果没有默认引用，则需要手动引用。

3.2.6 编写业务逻辑层代码

数据访问层编写完成后，接下来编写业务逻辑层代码。在 BLL 项目下创建一个类，命名为 UserLoginBLL.cs，一般以 BLL 结尾，前面部分为数据表名，同时把默认修饰符改为 public。

由于业务逻辑层肯定需要用到数据访问层对象，所以就需要用数据库访问层类实例化出一个数据访问层对象。然后再创建业务逻辑层的方法，该方法结构一般可以与数据访问层下的相应方法结构保持一致，如果修改一般就是修改方法名称和返回值类型。方法体里面一般需要用到数据访问层对象调用相应方法，具体代码如下。

```
public class UserLoginBLL  
{  
    //实例化数据访问层对象  
    UserLoginDAL dal=new UserLoginDAL();  
    public UserLogin SelectUserLogin(string username)  
    {  
        return dal.SelectUserLogin(username);  
    }  
}
```

3.2.7 实现 UI 层

右击 UI 层项目，选择“添加”→“Web 窗体”，窗体文件名取名为 Login，单击“确定”按钮之后即创建了文件名为 Login.aspx 的 Web 页面。

1. 表现层界面设计

表现层运行效果如图 3-12 所示。

图 3-12 登录效果图

通过表格布局，输入相关文字及引入相关控制，代码如下。

```
<form id="form1" runat="server">  
    <div id="divLogin">  
        <table id="tab">
```




```
        text-align:center;
    }
    .td1
    {
        height: 45px;
        width:90px;
        color:white;
        font-size:18px;
        font-family:微软雅黑;
        text-align:right;
    }
    .td2
    {
        text-align:center;
    }
</style>
```

2. 表现层功能实现

UI层的界面设计好了,接下来就实现表现层的功能。单击“登录”按钮时检测用户输入的用户名在数据库中是否存在,如果存在再看用户输入的密码是否与数据库中该用户的密码一致,如果一致则可以进入系统,否则给出相应的提示。

由于表现层需要用到业务逻辑层对象,所以需要先实例化出业务逻辑层对象。

```
UserLoginBLL bll=new UserLoginBLL();
```

然后编写“登录”按钮的单击事件如下。

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    //获取用户输入的用户名和密码
    string name =txtusername.Text.Trim();
    string pwd =txtpwd.Text.Trim();
    if (string.IsNullOrEmpty(name) || string.IsNullOrEmpty(pwd))
    {
        Response.Write("<script>alert('用户名或者密码不能为空!')</script>");
    }
    else
    {
        //获取到登录对象
        UserLogin user =bll.SelectUserLogin(name);
        if (user !=null)
        {
            //拿数据库里获取的密码与输入的密码对比
            if (user.Pwd ==pwd)
            {
                //将用户名和密码写入 Session 中
                Session["UserName"] =user.UserName;
                Session["Pwd"] =user.Pwd;
            }
        }
    }
}
```



```
//跳转到主页 StudentList.aspx,并传递用户名过去
Response.Redirect("StudentList.aspx?UserName=" + name);
}
else
{
    Response.Write("<script>alert('密码错误!')</script>");
}
}
else
{
    Response.Write("<script>alert('用户名不存在!')</script>");
}
}
}
```

注意：需要导入 Model 和 BLL 命名空间。

说明：

(1) Get 和 Post 的请求方式。

Get 和 Post 是向服务器发送请求的两种方式,其中,Get 请求是将需要提交给服务器的数据放在 URL 中,而 Post 请求则是将请求数据封装到请求报文中进行发送。

请求报文由请求行、请求头部、空行和请求数据 4 部分组成,其中,请求行中包括请求方式、URL 和 HTTP 版本三个字段;请求头部是通知服务器有关于客户端请求的信息;空行用于通知服务器以下不再是请求头;请求数据是使用 Post 方式发送的数据。

(2) Response 对象的使用——用于输出数据的对象。

Response 对象用于将服务器响应的数据发送到客户端,此对象中包含有关该响应的信息,并且通过 Response 对象的方法可以执行一些特殊操作。例如,通过该对象的 Write() 方法可以向页面输出内容。如果传递的参数是普通的字符串,则会被直接输出到页面;当传递的是“<script>alert('输入的内容')</script>”等内容的字符串参数时,浏览器会将该字符串当作脚本解析。

通过 Response 对象的 Redirect() 方法可以跳转到另一个页面,使用该方法时需要将重定向的 url 作为实参传递到该方法中,跳转的 url 可以通过“?”来传递参数,多个参数之间使用“&”连接,示例代码如下。

```
Response.Write("江西服装学院欢迎您!");
Response.Write("<script>alert('密码错误!')</script>");
Response.Redirect("http://www.jift.edu.cn");
Response.Redirect("http://www.jift.edu.cn?Username='admin'");
```

(3) Request 对象的使用——用于接收数据的对象。

Request 对象的作用是获取从客户端向服务器端发出的请求信息。根据请求方式的不同,可以通过三种方式来接收客户端的值,当使用 Get 方式发送请求时可以通过 QueryString 属性来获取值;当用户通过 Post 方式发送请求时,可以通过 Form 属性来获取值;当不确定请求方式时,可以通过 Request 对象直接获取值,具体示例代码如下。



```
string name=Request.QueryString["Name"]; //Get 请求
string name=Request.Form["Name"]; //Post 请求
string name=Request["Name"]; //Get 和 Post 请求
```

接下来为“取消”按钮编写单击事件代码,只需要把两个文本框置空即可,代码如下。

```
protected void btnCancel_Click(object sender, EventArgs e)
{
    txtusername.Text=string.Empty;
    txtpwd.Text=string.Empty;
}
```

3.2.8 设置启动项和测试项目运行结果

由于三层架构的解决方案下肯定有很多项目,所以运行测试之前要先设置启动项,右击解决方案,选择“设置启动项”,弹出如图 3-13 所示对话框,在此设置“单启动项目”为 UI。



图 3-13 设置解决方案下的启动项目

然后再右击 UI 项目下的 Login.aspx,选择“设为起始页”。

设置后启动运行出现“检测到在集成的托管管道模式下不适用的 ASP.NET 设置”错误提示,只需要在项目的 Web.config 配置文件中添加如下代码。

```
<system.webServer>
    <validation validateIntegratedModeConfiguration="false" />
</system.webServer>
```

为了测试运行效果,需要在 UI 层添加一个名为 StudentList.aspx 的 Web 窗体,当用户名和密码正确时进入该页面。

再次启动运行后,输入正确的用户名和密码后,能够进入主页面 StudentList.aspx。其他情况会给出相应的提示。



3.3 三层架构项目实战——学生信息列表展示页设计与实现

3.2 节完成了三层架构的登录,进入的主页面 StudentList.aspx 是一个空的页面,接下来把这个页面作为学生信息列表展示页面。所使用的数据表是 student 表,表结构如图 3-14 所示。

	列名	数据类型	允许 Null 值
PK	ID	int	☐
	Num	varchar(10)	☐
	Name	nvarchar(12)	☐
	Sex	nvarchar(2)	☒
	Age	int	☒
	Class	nvarchar(30)	☐
	Speciality	nvarchar(20)	☐
	Phone	varchar(11)	☒

图 3-14 student 表结构

3.3.1 在 Model 层添加学生表(student)实体类

在 Model 层中添加一个名为 Student.cs 的类文件,并在该文件中创建一个与 student 数据表对应的实体类,即 Student 类中的属性与 student 表中的字段一一对应。同时把类的修饰符改为 public,方便其他项目调用,具体代码如下。

```
public class Student
{
    public int ID { get; set; }
    public string Num { get; set; }
    public string Name { get; set; }
    public string Sex { get; set; }
    //int? 表示可空的值类型
    public int? Age { get; set; }
    public string Class { get; set; }
    public string Speciality { get; set; }
    public string Phone { get; set; }
}
```

3.3.2 在数据访问层查询学生表(student)数据

在 DAL 中添加一个名为 StudentDAL.cs 的类文件,类的修饰符改为 public,在该类中封装对 student 表的增、删、改、查操作的代码。在该类中定义一个 GetAllStudent() 方法,该方法用于查询 student 表中的所有数据,具体代码如下。

```
public List<Student> GetAllStudent()
{
    string sql = "select * from student";
    List<Student> studentList = new List<Student>();
    using (SqlDataReader reader = SqlHelper.ExecuteReader(sql))
    //提示:正常该方法有两个参数,即(sql,null),但可变参数为空时可以不写
}
```



```
{
    if (reader.HasRows)
    {
        while (reader.Read())
        {
            Student stu = new Student();
            stu.ID = reader.GetInt32(0);
            stu.Num = reader.GetString(1);
            stu.Name = reader.GetString(2);
            stu.Sex = Convert.IsDBNull(reader[3]) ? null : reader.
GetString(3);
            stu.Age = Convert.IsDBNull(reader[4]) ? null : (int?)
reader.GetInt32(4);
            stu.Class = reader.GetString(5);
            stu.Speciality = reader.GetString(6);
            stu.Phone = Convert.IsDBNull(reader[7]) ? null : reader.
GetString(7);
            studentList.Add(stu);
        }
    }
}
return studentList;
}
```

在上述代码中, GetAllStudent() 方法调用了 SqlHelper 的 ExecuteReader() 方法查询数据, 并将查询到的数据封装到泛型集合 List<Student> 中, 并作为返回值返回。其中, 在查询数据时取出数据表中可为空的类型列的数据赋给对象属性时, 需要先通过 Convert.IsDBNull() 方法判断再赋值。

提示: int 类型与 null 值不是同类型, 在进行三元运算时需要将 int 类型强制转换成 int? 类型。

3.3.3 在业务逻辑层利用数据访问层查询学生表(student)数据

在 BLL 中添加一个名为 StudentBLL.cs 的类文件, 类的修饰符改为 public, 在该类中也是封装对 student 表的增、删、改、查操作的代码, 但是是通过数据访问层对象调用数据访问层的方法, 只不过是在业务逻辑层对数据访问层的方法进行了一次封装。在该类中定义的方法结构可以与数据访问层的保持一致, 具体代码如下。

```
StudentDAL dal = new StudentDAL();
public List<Student> GetAllStudent()
{
    return dal.GetAllStudent();
}
```

3.3.4 在表现层调用业务逻辑层

在 UI 层的 StudentList.aspx.cs 文件的 Page_Load() 事件中编写代码, 该事件在页面加载时触发, 触发时将所有数据加载到页面中, 具体代码如下。



```
["UserName"].ToString() %>, 欢迎使用学生信息管理系统</div>
    <div id="right">
        <a href="StudentList.aspx"> 网站主页 </a> &nbsp; &nbsp; &nbsp; &nbsp;
&nbsp; &nbsp;
        <a href="UpdatePassword.aspx">修改密码</a>&nbsp; &nbsp; &nbsp; &nbsp;
&nbsp; &nbsp;
        <a href="temp.aspx">注销退出</a>&nbsp; &nbsp; &nbsp; &nbsp; &nbsp; &nbsp;
    </div>
</div>
</form>
</body>
```

说明：“<%=%>”用于读取 Session 中的用户名并展示到页面上。

添加布局样式,在 UI 层下创建文件夹 Style,再在该文件夹下创建一个样式表文件 NavigateStyle.css,并添加如下样式。

```
body {
    margin: 0px;
}
#box {
    width: 100%;
    background-color: cornflowerblue;
    height: 80px;
    line-height: 80px;
    font-family: "微软雅黑";
}
#left {
    width: 60%;
    float: left;
}
#right {
    text-align: right;
    width: 40%;
    float: left;
}
a:link {
    color: #fff;
    text-decoration: none;
    font-family: 微软雅黑;
}
a:hover {
    color: #f00;
    text-decoration: none;
    font-family: 微软雅黑;
}
```

启动运行,登录成功后看到的效果如图 3-15 所示。

您好: admin,欢迎使用学生信息管理系统

网站首页 修改密码 注销退出

编号	学号	姓名	性别	年龄	班级	专业	电话	操作
1	1001	吴花花	女	20	20软本1班	软件工程	13467654589	删除 修改
2	1003	张丽花	女	20	20数据科学1班	数据科学与大数据技术	13565890976	删除 修改
3	1005	高君发	女	21	21软本2班	软件工程	13898098768	删除 修改
4	1002	王前坤	女	22	21物联网1班	物联网工程	13987655670	删除 修改
添加用户								

图 3-15 学生信息管理系统主页面效果

3.4 三层架构项目实战——添加学生信息设计与实现

本节实现三层架构的添加学生信息功能。在 UI 层项目下添加一个名为 AddStudent.aspx 的 Web 窗体。

3.4.1 设计添加学生信息的界面

添加学生信息界面如图 3-16 所示。

您好: ,欢迎使用学生信息管理系统

网站首页 修改密码 注销退出

添加信息

学号:		姓名:	
班级:		专业:	
年龄:		电话:	
性别:	☒ 男 ☐ 女		
保存 清空			

图 3-16 添加学生信息界面

1. 导航栏部分设计

该部分设计与主页 StudentList.aspx 的导航栏一样,直接复制相关代码与样式即可。

2. 引入添加学生信息的控件及布局

采用表格布局,从工具箱中拖入 6 个 TextBox 控件、2 个 RadioButton 控件和 2 个 Button 控件,并修改 ID 等属性。

然后编写样式,最终参考代码如下。

(1) 样式代码,其中,NavigateStyle.css 样式见 3.3 节。

```
<link href="Style/NavigateStyle.css" rel="stylesheet" />
<style type="text/css">
    #divLogin
    {
        background-color:cornflowerblue;
```



```
        width: 550px;
        height: 300px;
        left: 50%;
        margin-left: -275px;
        top: 50%;
        margin-top: -150px;
        position: absolute;
    }

    #tb
    {
        width: 100%;
        height: 100%;
    }

    #td1
    {
        height: 70px;
        font-family: 微软雅黑;
        color: white;
        font-size: 28px;
        text-align: center;
    }

    .td2
    {
        height: 40px;
        font-family: 微软雅黑;
        color: white;
        font-size: 20px;
        text-align: right;
    }

    #td3
    {
        height: 56px;
        text-align: center;
    }
</style>
```

(2) HTML 代码。

```
<form id="form1" runat="server">
    <div id="box">
        <div id="left"> &nbsp; &nbsp; &nbsp; &nbsp; 您好: <% = Session [ "
UserName"].ToString() %>, 欢迎使用学生信息管理系统</div>
        <div id="right">
            <a href="StudentList.aspx"> 网站主页 </a> &nbsp; &nbsp; &nbsp;
&nbsp; &nbsp; &nbsp; &nbsp;
```

