

计算机与信息学院

《Java高级技术》

结课报告

题目：学生管理系统

开课学期：2019年春季学期

网选班号： 2 班

姓 名： 冯阳

学 号： 2017112319

目录

[一. 系统分析 2](#_Toc12286174)

[1.1. 开发背景 2](#_Toc12286175)

[1.2. 目的和要求 2](#_Toc12286176)

[1.3. 可行性研究 2](#_Toc12286177)

[二．开发技术及开发环境 3](#_Toc12286178)

[2.1. Java语言及J2EE技术介绍 3](#_Toc12286179)

[2.2. Spring框架介绍 3](#_Toc12286180)

[2.3. SpringMVC框架介绍 5](#_Toc12286181)

[2.4. MyBatis框架介绍 6](#_Toc12286182)

[2.5. easyUI框架介绍 7](#_Toc12286183)

[三．需求分析 7](#_Toc12286184)

[3.1. 需求 7](#_Toc12286185)

[3.2. 开发环境 9](#_Toc12286186)

[四．概要设计 9](#_Toc12286187)

[4.1. view层设计 9](#_Toc12286188)

[4.2. 数据库设计 13](#_Toc12286189)

[4.3. Service层设计 15](#_Toc12286190)

[4.4. Dao层设计 16](#_Toc12286191)

[五．详细设计 17](#_Toc12286192)

[5.1. Dao层 17](#_Toc12286193)

[5.2. Service层 18](#_Toc12286194)

[5.3. Controller层 19](#_Toc12286195)

[5.4 Mapper映射 20](#_Toc12286196)

[六. 附录 21](#_Toc12286197)

[6.1. 全局配置文件web.xml 21](#_Toc12286198)

[6.2. SpringMVC配置文件springmvc.xml 25](#_Toc12286199)

[6.3. Spring配置文件applicationContext.xml 28](#_Toc12286200)

[6.4. 日志文件log4j.properties 33](#_Toc12286201)

[6.5. 数据库配置文件dp.properties 33](#_Toc12286202)

[七．参考文献 36](#_Toc12286203)

## 一. 系统分析

### 开发背景

随着互联网技术的飞速发展，各种应用涌入人们的生活中，使人们的生活更加便捷，生活质量越来越高。人们不断地从互联网中获取各种讯息，不断地使用着用互联网技术实现的应用。可以毫不夸张地说，互联网技术正在改变人们生活的方方面面。

在生活中，信息处理是互联网技术、应用应用到人们生活过程中的一个很重要的，需要处理的方式，这也是利用计算机的一个原因。因为计算机最大的有事在于它快速处理数据的能力，同时保证了正确性、安全性，尤其是对于复杂的数据而言，他的优势将会非常明显。使用计算机进行信息管理与信息管理系统的开发密切相关，系统的开发是系统管理的前提。

### 目的和要求

原来是利用手工的方式进行信息的存储、修改等操作，但是很明显，这种方式有一个很大的弊端，即费时费力费财，特别是当数据量特别大的时候，人工几乎不可能完成，因此需要解决计算机开发一个管理系统来代替人工的方式。

该文档是继系统设计文档之后而开发出来能够详尽的介绍学生管理系统的核心界面和核心代码的实现，该系统的开发具有指导性的意义。学生管理系统可以很好地代表各种管理系统，其他的管理系统可以在本系统上修改得到。

### 可行性研究

设计的这个系统，可以从根本上解决传统方式进行学生系统管理的缺点，使得效率大大提升，方便了管理。从实用性来讲，由于这个系统是基于Web的，在浏览器普及的情况下，使用它将会非常方便，用户只需要一个浏览器就可以实现登录等操作。同时如今非常主流的编程语言和开发工具都可以实现这样的一个系统，特别方便。从技术上来讲，目前也有许多成功的管理网站的例子可以参考，故实现起来不会特别复杂。

## 二．开发技术及开发环境

### 2.1. Java语言及J2EE技术介绍

Java语言是美国Sun公司于1995年推出的一种简单的、面向对象的、分布式的、可解释的、键壮的、安全的、结构的、中立的、可移植的、性能很优异的多线程的、动态的语言。其前身为OAK语言，是SUN公司为一些消费性电子产品而设计的一个通用环境。他们最初的目的只是为了开发一种独立于平台的软件技术。经过Sun公司的工程师的不懈努力以及全世界无数的编程爱好者的使用，Java终于发展成为今天这样一个集桌面(J2SE)、网络(J2EE)、移动平台(J2ME)应用为一体的功能强大的编程语言。目前Java由于其平台无关及分布式特性，最重要的应用是在网络应用上。

J2EE技术是Java网络应用的技术，广泛应用于企业级的应用。它是由一套规范（shannin,2001）和建立在J2SE平台上的应用编程接口(API)组成。J2EE拥有Sun公司提供免费开发工具，称为J2EE软件开发工具包(Software Development Kit,SDK )（http://java.sun.com/j2ee）。J2EE技术是一种Java网络应用技术的组合，包含有远程方法调用(Remote Method Invocation,RMI)、Java消息服务(Java Message Service,JMS)、JavaMail等通信服务，JDBC、Java命名和目录接口(Java naming and Directory Interface,JNDI)、Java连接器体系结构(Java Connector Architecture,JCA)、Java事务API/Java事务服务(JTA/JTS)、XML处理API等通用服务以及Servlet、Java服务器页面(JSP)、企业级JavaBean(EJB)等组件技术。任何J2EE的实现都必须要在其特定容器中实现。

### 2.2. Spring框架介绍

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许使用者选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。Spring的核心是控制反转（IoC）和面向切面（AOP）。

简单来说，Spring是一个分层的JavaSE/EE full-stack(一站式) 轻量级开源框架。如图2.1所示：

****

图2.1 Spring框架图

组成 Spring 框架的每个模块（或组件）都可以单独存在，或者与其他一个或多个模块联合实现。每个模块的功能如下：

核心容器：核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转 （IOC） 模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。

**Spring 上下文：**Spring 上下文是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。

**Spring AOP：**通过配置管理特性，Spring AOP 模块直接将面向方面的编程功能集成到了 Spring 框架中。所以，可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

**Spring DAO：**JDBC DAO 抽象层提供了有意义的异常层次结构，可用该结构来管理异常处理和不同数据库供应商抛出的错误消息。异常层次结构简化了错误处理，并且极大地降低了需要编写的异常代码数量（例如打开和关闭连接）。Spring DAO 的面向 JDBC 的异常遵从通用的 DAO 异常层次结构。

**Spring ORM：**Spring 框架插入了若干个 ORM 框架，从而提供了 ORM 的对象关系工具，其中包括 JDO、Hibernate 和 iBatis SQL Map。所有这些都遵从 Spring 的通用事务和 DAO 异常层次结构。

**Spring Web 模块：**Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文。所以，Spring 框架支持与 Jakarta Struts 的集成。Web 模块还简化了处理多部分请求以及将请求参数绑定到域对象的工作。

**Spring MVC 框架：**MVC 框架是一个全功能的构建 Web 应用程序的 MVC 实现。通过策略接口，MVC 框架变成为高度可配置的，MVC 容纳了大量视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI。

### 2.3. SpringMVC框架介绍

实际上Spring MVC是Spring框架的一部分，Spring框架成为Java EE开发主流框架后，Spring开发小组又在Spring框架的基础上推出了MVC架构，主要用于支持WEB应用程序的开发。

MVC是Model（模型，也称为数据模型）、View（视图）、Controll（控制器）三个英文单词首字母的缩写。从MVC组合的三个单词也可以看出，MVC是一种设计模型，它使用控制器将数据模型和视图进行分离，也就是将视图和数据解耦。这样的好处是后端处理的数据模型和前端视图显示的数据格式无关，实现一个数据模型可以对应多个视图以不同的方式来展现数据，当数据模型或视图发生变化时，相互之间的影响也会降低到最低。

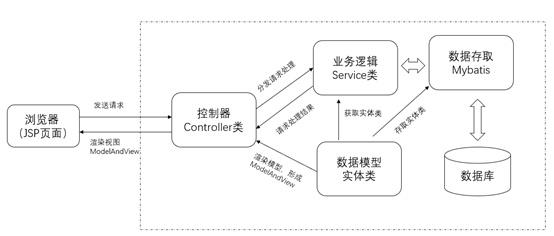
MVC中最重要的核心就是控制器，控制器与视图和模型相对独立，它起到一个负责分发请求和返回处理结果的作用，对请求和数据模型的处理一般由JavaBean负责。

在MVC架构中，JSP页面就是视图，用户通过JSP页面发出请求后，Spring MVC会根据请求路径，将请求发给与请求路径对应的Controller类，Controller类调用Service类对请求进行处理，Service类会调用数据持久层MyBatis完成对实体类的存取和查询工作，并将处理结果返回到Controller类，Controller类将处理结果转换为ModelAndView对象，JSP接收ModelAndView对象并进行渲染。

Controller类是在普通Java类的头部加入@Controller注解，使其变为Controller类，在Controller类中使用@RequestMapping注解标记在Controller类方法的头部，使该方法可以响应一个WEB请求地址，当JSP页面向这个地址发送请求时，该方法会被调用。

Service类为业务逻辑处理类，在类的头部使用注解@Service标注，被注解@Service标注的类会被Spring框架自动注入到Controller类。

**SpringMVC架构图如图2.2所示：**

****

**图2.2 Spring MVC架构图**

### 2.4. MyBatis框架介绍

MyBatis本是[apache](https://baike.baidu.com/item/apache" \t "https://baike.baidu.com/item/MyBatis/_blank)的一个开源项目[iBatis](https://baike.baidu.com/item/iBatis" \t "https://baike.baidu.com/item/MyBatis/_blank), 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。

MyBatis是一个开源的数据持久层框架，内部封装了通过JDBC访问数据库的操作，支持普通的SQL查询、存储过程和高级映射。作为持久层框架，主要思想是将程序中的大量的SQL语句分离出来，配置在相应的配置文件中，这样可以在不修改代码的情况下，直接在配置文件中的修改SQL。MyBatis 消除了几乎所有的[JDBC](https://baike.baidu.com/item/JDBC" \t "https://baike.baidu.com/item/MyBatis/_blank)代码和参数的手工设置以及[结果集](https://baike.baidu.com/item/%E7%BB%93%E6%9E%9C%E9%9B%86" \t "https://baike.baidu.com/item/MyBatis/_blank)的检索。MyBatis 使用简单的 XML或注解用于配置和原始映射，将接口和 Java 的POJOs（Plain Ordinary Java Objects，普通的 Java对象）映射成数据库中的记录。

### 2.5. easyUI框架介绍

easyUI是一种基于jQuery的用户界面插件集,它是个完美支持HTML5网页的完整框架 ，能够节省网页开发的时间和规模。

开发者不需要编写复杂的javascript，也不需要对css样式有深入的了解，开发者需要了解的只有一些简单的html标签。

这个框架主要有以下特点：

1）jquery用户界面插件的集合

2）一些前用于交互的js应用提供必要的功能

3）esyUI支持两种渲染方式分别为javascript方式（如：$('#p').panel({...})）和html标记方式（如：class="easyui-panel"）

4）支HTML5（通过data-options属性）

5）产品时可节省时间和资源

6）简单很强大

7）支持，可根据自己的需求扩展控件

8）各项不足正以版本递增的方式不断完善

## 三．需求分析

### 3.1. 需求

#### 3.1.1 用户（管理员）管理

1）用户添加： 根据用户的用户名和密码可以新建一个用户。添加成功之后自动跳转到用户列表页面。

2）用户修改： 选定一个用户之后，可以修改其用户名或者密码。修改成功之后自动跳转到用户列表页面。

3）用户删除： 在用户列表界面，可以选择一个或者多个用户进行删除，删除成功后自动更新用户列表界面。

4）用户查询： 在用户列表界面可以根据用户的用户名进行查询，搜索到满足条件的用户即会显示于表格之中，未查到则显示为空；

#### 3.1.2 学生管理

1）学生添加： 根据学生的头像、姓名、登录密码、所属班级、性别、和备注可以新建一个学生。添加成功之后自动跳转到学生列表页面。

2）学生修改： 选定一个学生之后，可以修改其头像、姓名、登录密码、所属班级、性别、和备注。修改成功之后自动跳转到学生列表页面。

3）用户删除： 在学生列表界面，可以选择一个或者多个学生进行删除，删除成功后自动更新学生列表界面。

4）学生查询： 在学生列表界面可以根据学生的姓名进行查询，同时可以根据班级进行筛选。搜索到满足条件的学生即会显示于表格之中，未查到则显示为空；

#### 3.1.3 年级管理

1）年级添加： 根据年级的年级名和备注可以新建一个年级。添加成功之后自动跳转到年级列表页面。

2）年级修改： 选定一个年级之后，可以修改其年级名或备注。修改成功之后自动跳转到用户列表页面。

3）年级删除： 在年级列表界面，可以选择一个或者多个年级进行删除，删除成功后自动更新年级列表界面。

4）年级查询： 在年级列表界面可以根据年级的年级名进行查询，搜索到满足条件的年级即会显示于表格之中，未查到则显示为空

#### 3.1.4 班级管理

1）班级添加： 根据班级的班级名、所属年级和备注可以新建一个班级。添加成功之后自动跳转到班级列表页面。

2）班级修改： 选定一个班级之后，可以修改其班级名、所属年级或备注。修改成功之后自动跳转到班级列表页面。

3）班级删除： 在班级列表界面，可以选择一个或者多个班级进行删除，删除成功后自动更新班级列表界面。

4）班级查询： 在班级列表界面可以根据班级的班级名进行查询，搜索到满足条件的班级即会显示于表格之中，未查到则显示为空

### 3.2. 开发环境

系统环境：Windows

开发工具：Eclipse、Navicat

Java版本：JDK 10

服务器：tomcat 8.5

数据库：MySQL 8

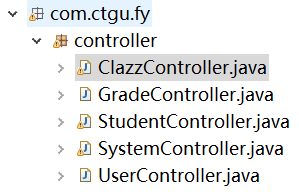
## 四．概要设计

### 4.1. view层设计

#### 4.1.1. 后台设计

##### 4.1.1.1. 框图设计

框图设计如图4.1所示：

****

**图4.1 后台controller包图**

[注]： 图示中展开的包均为接口

##### 4.1.1.2. 框图设计说明

**类名：**ClazzController

功能：此类主要实现了在接到对班级的处理结果后，根据处理结果找到要作为向客户端发回的响应View页面。页面经渲染（数据填充）后，再发送给客户端。

**类名：**GradeController

功能：此类主要实现了在接到对年级的处理结果后，根据处理结果找到要作为向客户端发回的响应View页面。页面经渲染（数据填充）后，再发送给客户端。

**类名：**StudentController

功能：此类主要实现了在接到对学生的处理结果后，根据处理结果找到要作为向客户端发回的响应View页面。页面经渲染（数据填充）后，再发送给客户端。

**类名：**UserController

功能：此类主要实现了在接到对用户（管理员）的处理结果后，根据处理结果找到要作为向客户端发回的响应View页面。页面经渲染（数据填充）后，再发送给客户端。

#### 4.1.2. 前端展示

1）登录界面

登陆界面如图4.2所示：

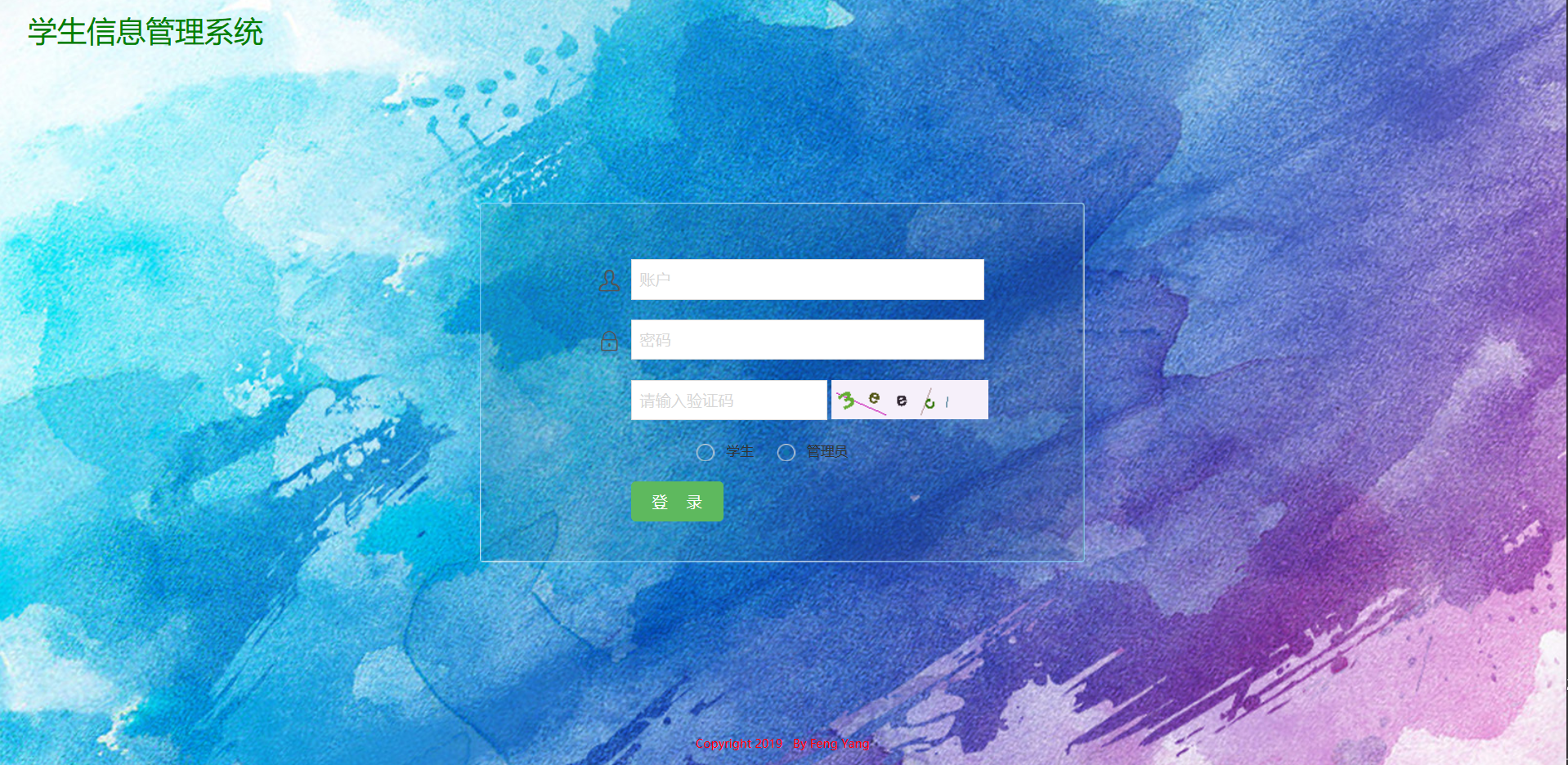
****

图4.2 登录界面

2）主界面

主界面如图4.3所示：

****

图4.3 主界面

3）查询界面

查询界面如图4.4所示：

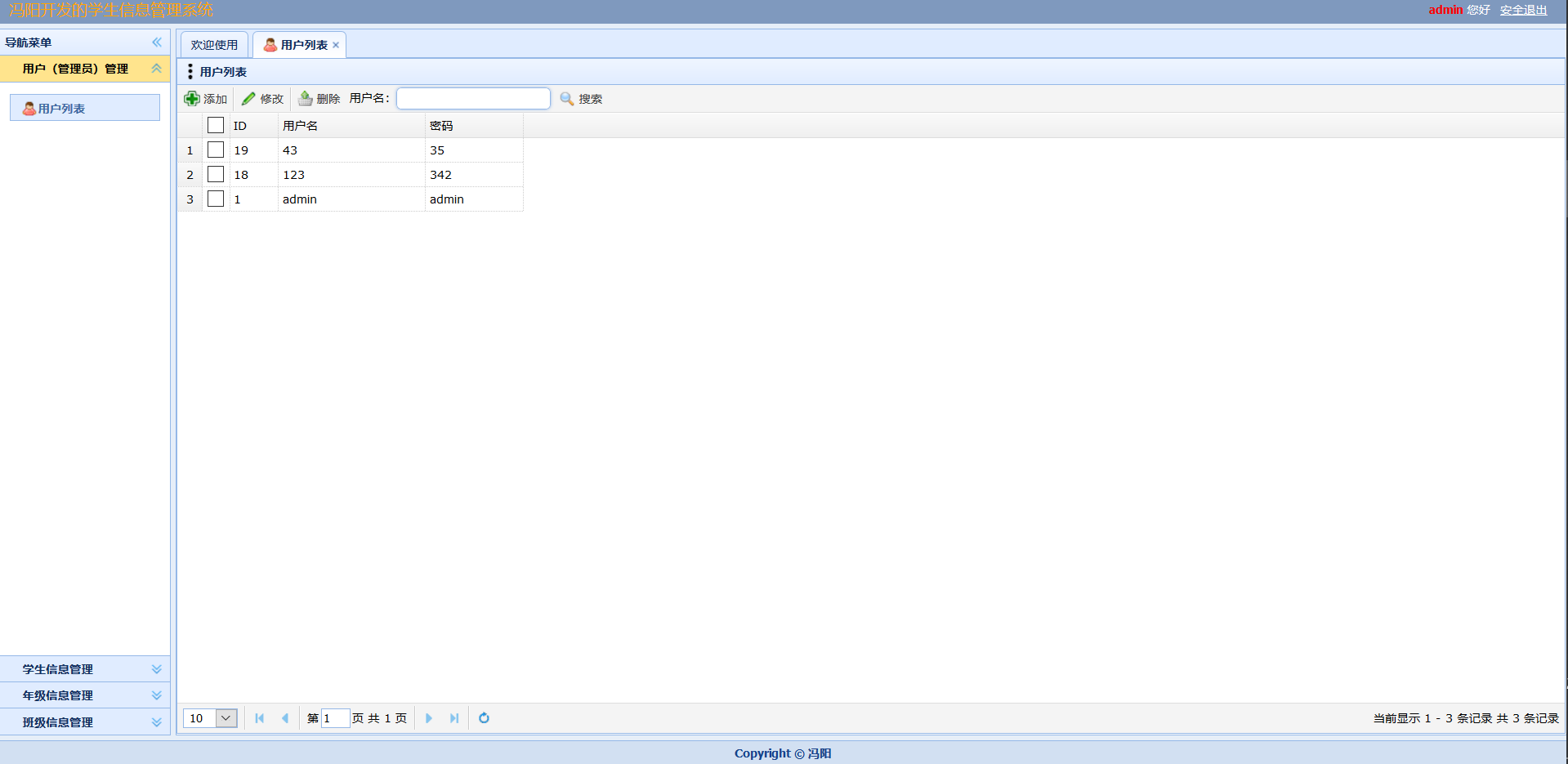
****

图4.4 查询界面

4）添加界面

添加界面如图4.5所示：

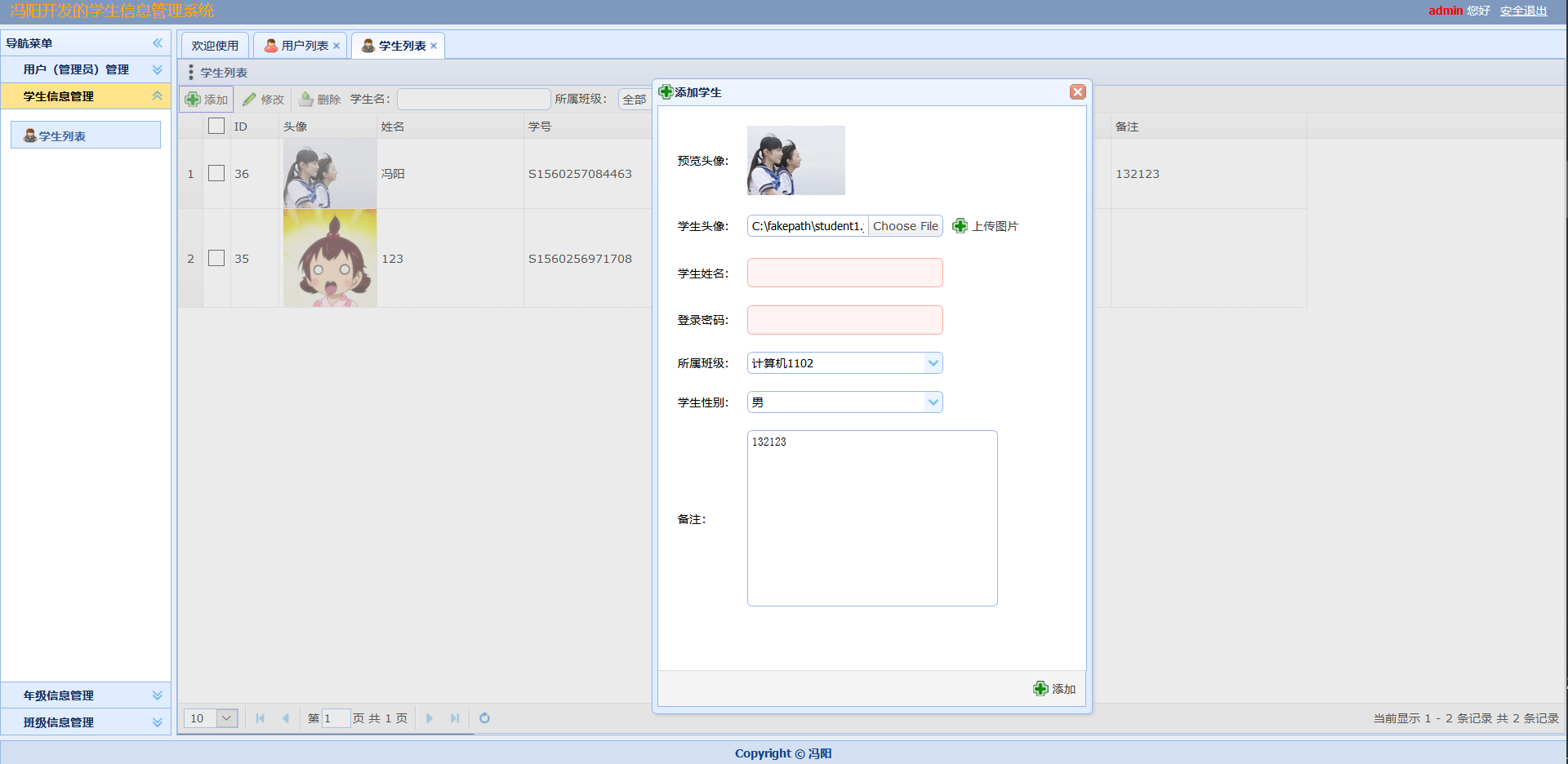
****

图4.5 添加界面

5）修改界面

修改界面如图4.6所示：

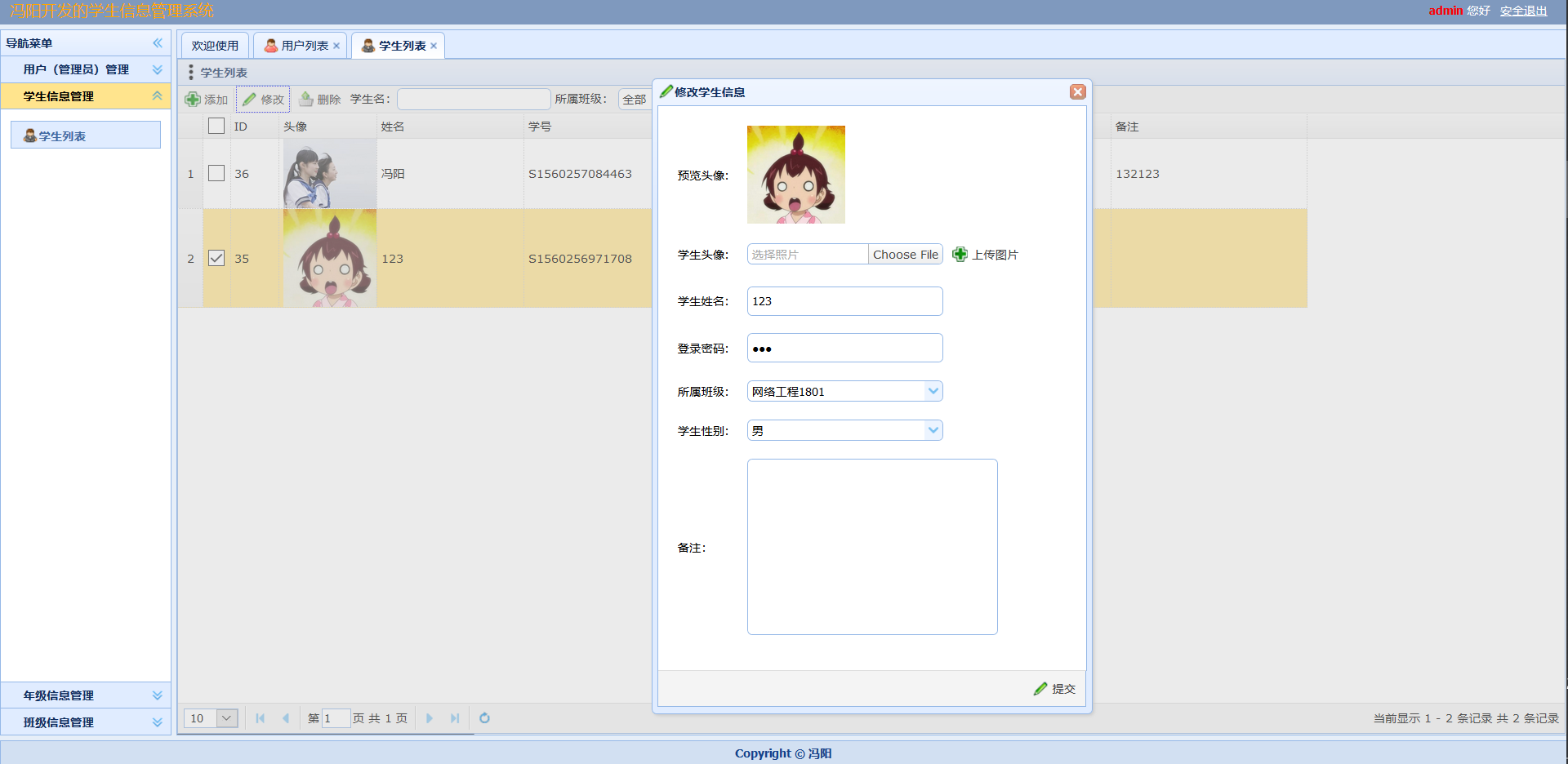
****

图4.6 修改界面

### 4.2. 数据库设计

通过对用户需求的分析，设计出以下实体，如图4.7：

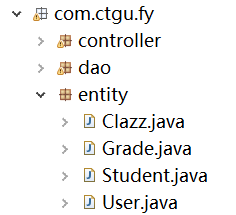
****

图4.7 后台entity包图

#### 4.2.1. 用户表/管理员表 (user)

user表如表4-1：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 长度 | 主键 | 解释 |
| id | int | 11 | √ | 列表中第几个用户 |
| username | varchar | 32 |  | 用户名 |
| password | varchar | 32 |  | 密码 |

表4-1 用户表/管理员表(user)

#### 4.2.2. 学生表 (student)

student表如表4-2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 长度 | 主键 | 解释 |
| id | int | 11 | √ | 列表中第几个学生 |
| clazzId | int | 11 |  | 班级ID |
| sn | varchar | 32 |  | 学号 |
| username | varchar | 32 |  | 姓名 |
| password | varchar | 32 |  | 密码 |
| sex | varchar | 8 |  | 性别 |
| photo | varchar | 128 |  | 头像 |
| remark | varchar | 512 |  | 备注 |

表4-2 学生表(student)

#### 4.2.3. 年级表 (grade)

grade表如表4-3：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 长度 | 主键 | 解释 |
| id | int | 11 | √ | 列表中第几个年级 |
| name | varchar | 32 |  | 年级名 |
| remark | varchar | 512 |  | 备注 |

表4-3 年级表(grade)

#### 4.2.4. 班级表 (clazz)

clazz表如表4-4：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 长度 | 主键 | 解释 |
| id | int | 11 | √ | 列表中第几个班级 |
| gradeId | int | 11 |  | 所属年级id |
| name | varchar | 32 |  | 班级名 |
| remar | varchar | 512 |  | 备注 |

表4-4 班级表(clazz)

### 4.3. Service层设计

#### 4.3.1 框图设计

框图设计如图4.8

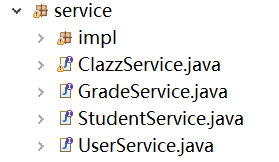
****

图4.8 service包图设计

[注] ：impl为相应实现

#### 4.3.2 框图设计说明

类名：ClazzService

功能：此类主要封装了对班级进行增删查改、获取总数的动态查询语句，组装数据访问层，逻辑性的操作。

类名：GradeService

功能：此类主要封装了对年级进行增删查改、获取总数的动态查询语句，组装数据访问层，逻辑性的操作。

类名：StudentService

功能：此类主要封装了对学生进行增删查改、获取总数的动态查询语句，组装数据访问层，逻辑性的操作。

类名：UserService

功能：此类主要封装了对用户（管理员）进行增删查改、获取总数的动态查询语句，组装数据访问层，逻辑性的操作。

### 4.4. Dao层设计

#### 4.4.1 框图设计

框图设计如图4.9：

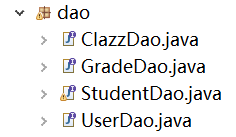
****

图4:9 dao包图设计

#### 4.4.2 框图设计说明

类名：ClazzDao

功能：此类主要封装了对班级进行增删查改、获取总数的动态查询语句，直接访问数据库，原子性的操作。

类名：GradeDao

功能：此类主要封装了对年级进行增删查改、获取总数的动态查询语句，直接访问数据库，原子性的操作。

类名：StudentDao

功能：此类主要封装了对学生进行增删查改、获取总数的动态查询语句，直接访问数据库，原子性的操作。

类名：UserDao

功能：此类主要封装了对用户（管理员）进行增删查改、获取总数的动态查询语句，直接访问数据库，原子性的操作。

## 五．详细设计

### 5.1. Dao层

DAO层主要是做数据持久层的工作，负责与数据库进行联络的一些任务都封装在此，DAO层的设计首先是设计DAO的接口，然后在Spring的配置文件中配置扫描 dao 包，动态实现 dao 接口，注入到 spring 容器中。然后就可在模块中调用此接口来进行数据业务的处理，而不用关心此接口的具体实现类是哪个类，显得结构非常清晰，DAO层的数据源配置，以及有关数据库连接的参数都在Spring的配置文件中进行配置。

数据库中的每一个实体对应一个dao接口，每一个接口都拥有对应实体的CRUD等操作，Dao层只是负责和数据库打交道，具体包括以下几个接口，如图5.1：

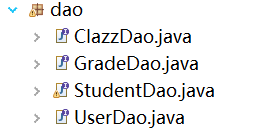
****

图5.1 dao包图设计

#### 5.1.1. ClazzDao

对应Clazz实体，封装了对Clazz实体的一系列增删查改等操作，是直接访问数据库的操作。

#### 5.1.2. GradeDao

对应Grade实体，封装了对Grade实体的一系列增删查改等操作，是直接访问数据库的操作。

#### 5.1.3. StudentDao

对应Student实体，封装了对Student实体的一系列增删查改等操作，是直接访问数据库的操作。

#### 5.1.4. UserDao

对应User实体，封装了对User实体的一系列增删查改等操作，是直接访问数据库的操作。

### 5.2. Service层

与Dao层不同，Service层主要负责业务模块的逻辑应用设计。同样是首先设计接口，再设计其实现的类，接着在Spring的配置文件中配置其实现的关联。这样我们就可以在应用中调用Service接口来进行业务处理。Service层的业务实现，具体要调用到已定义的DAO层的接口，封装Service层的业务逻辑有利于通用的业务逻辑的独立性和重复利用性，程序显得非常简洁。

在这个项目中，我将每一个角色写成一个java文件，每一个独立的功能写成一个单独的方法，方便其他的代码进行调用，具体有以下几个Service接口，如图5.2

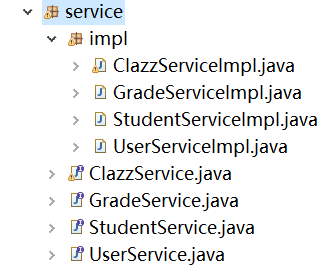
****

图5.2 service包图计

#### 5.2.1. ClazzService

建立在ClazzDao上面的（有了相应的Dao层才能有对应的Service层），封装了对Clazz实体的一系列操作的业务逻辑，调用ClazzDao接口，同时提供接口给相应的Controller调用。ClazzService接口的具体实现类为ClazzServiceImpl.

#### 5.2.2. GradeService

建立在GradeDao上面的（有了相应的Dao层才能有对应的Service层），封装了对Grade实体的一系列操作的业务逻辑，调用GradeDao接口，同时提供接口给相应的Controller调用。GradeService接口的具体实现类为GradeServiceImpl.

#### 5.2.3. StudentService

建立在StudentDao上面的（有了相应的Dao层才能有对应的Service层），封装了对Student实体的一系列操作的业务逻辑，调用StudentDao接口，同时提供接口给相应的Controller调用。StudentService接口的具体实现类为StudentServiceImpl.

#### 5.2.4. UserService

建立在UserDao上面的（有了相应的Dao层才能有对应的Service层），封装了对User实体的一系列操作的业务逻辑，调用UserDao接口，同时提供接口给相应的Controller调用。UserService接口的具体实现类为UserServiceImpl.

### 5.3. Controller层

Controller层负责具体的业务模块流程的控制，是View层的后台，是Model与View之间沟通的桥梁，可以分派用户的请求并选择恰当的视图以用于显示，同时可以解释用户的输入并映射为模型层可执行的操作，在此层里面要调用Service层的接口来控制业务流程，控制的配置也同样是在Spring的配置文件里面进行，针对具体的业务流程，会有不同的控制器，我们具体的设计过程中可以将流程进行抽象归纳，设计出可以重复利用的子单元流程模块，这样不仅使程序结构变得清晰，也大大减少了代码量。

在这个项目中，我将每一个角色写成一个java文件，每一个独立的功能写成一个单独的方法，方便其他的代码进行调用，具体有以下几个Controller接口，如图5.3：

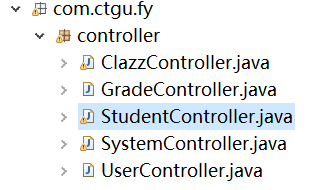
****

图5.3 controller包图设计

#### 5.3.1. ClazzController

Clazz控制器，调用ClazzService接口来控制业务逻辑。

#### 5.3.2. GradeController

Grade控制器，调用GradeService接口来控制业务逻辑。

#### 5.3.3. StudentController

Student控制器，调用StudentService接口来控制业务逻辑。

#### 5.3.4. UserController

User控制器，调用UserService接口来控制业务逻辑。

### 5.4 Mapper映射

#### 5.4.1 映射框图设计

映射框图设计如图5.4：

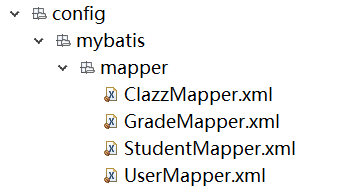
****

图5.4 mapper包图设计

#### 5.4.2 映射框图设计说明

类名：ClazzMapper

功能：使用Mapper代理的方式，生成ClazzDao的实现类的对象。

类名：GradeMapper

功能：使用Mapper代理的方式，生成GradeDao的实现类的对象。

类名：StudentMapper

功能：使用Mapper代理的方式，生成StudentDao的实现类的对象。

类名：UserMapper

功能：使用Mapper代理的方式，生成UserDao的实现类的对象。

## 六. 附录

### 6.1. 全局配置文件web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">

<!-- 中文乱码处理 -->

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter>

<filter-name>HiddenHttpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>HiddenHttpMethodFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- Spring配置文件信息 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:config/spring/applicationContext.xml</param-value>

</context-param>

<!-- ContextLoaderListener监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 日志配置 -->

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>classpath:config/log4j.properties</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

<!-- 配置前端控制器 -->

<servlet>

<servlet-name>DispatcherServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:config/springmvc/springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>DispatcherServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<error-page>

<error-code>404</error-code>

<location>/WEB-INF/errors/404.jsp</location>

</error-page>

<error-page>

<error-code>500</error-code>

<location>/WEB-INF/errors/500.jsp</location>

</error-page>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

### 6.2. SpringMVC配置文件springmvc.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:task="http://www.springframework.org/schema/task"

xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.2.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.2.xsd

http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task-3.2.xsd">

<!-- 只需要扫描包中的 Controller 注解 -->

<context:component-scan base-package="com.ctgu.fy.controller">

<context:include-filter type="annotation"

expression="org.springframework.stereotype.Controller" />

</context:component-scan>

<!-- 启动 mvc 注解驱动 -->

<mvc:annotation-driven></mvc:annotation-driven>

<!-- 启动定时任务 -->

<task:annotation-driven/>

<!-- 静态资源处理 -->

<mvc:default-servlet-handler/>

<!-- 配置视图解析器 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/"></property>

<property name="suffix" value=".jsp"></property>

</bean>

<!-- 文件上传 -->

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<!-- 上传文件大小限制 -->

<property name="maxUploadSize">

<value>10485760</value>

</property>

<!-- 请求的编码格式, 和 jsp 页面一致 -->

<property name="defaultEncoding">

<value>UTF-8</value>

</property>

</bean>

<!-- 后台访问拦截器 -->

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path="/\*\*"/>

<mvc:exclude-mapping path="/system/login"/>

<mvc:exclude-mapping path="/system/get\_cpacha"/>

<mvc:exclude-mapping path="/h-ui/\*\*"/>

<mvc:exclude-mapping path="/easyui/\*\*"/>

<bean class="com.ctgu.fy.interceptor.LoginInterceptor"></bean>

</mvc:interceptor>

</mvc:interceptors>

</beans>

### 6.3. Spring配置文件applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.2.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.2.xsd">

<context:component-scan base-package="com.ctgu.fy">

<context:include-filter type="annotation"

expression="org.springframework.stereotype.Component" />

<context:include-filter type="annotation"

expression="org.springframework.stereotype.Repository" />

<context:include-filter type="annotation"

expression="org.springframework.stereotype.Service" />

</context:component-scan>

<!-- 加载配数据源配置文件 db.properties -->

<context:property-placeholder location="classpath:config/db.properties" />

<!-- 配置 C3P0 数据源 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">

<property name="driverClass" value="${datasource.connection.driver\_class}"/>

<property name="jdbcUrl" value="${datasource.connection.url}"/>

<property name="user" value="${datasource.connection.username}"/>

<property name="password" value="${datasource.connection.password}"/>

<property name="minPoolSize" value="${datasource.connection.minPoolSize}"/>

<!--连接池中保留的最大连接数。Default: 15 -->

<property name="maxPoolSize" value="${datasource.connection.maxPoolSize}"/>

<!--最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->

<property name="maxIdleTime" value="${datasource.connection.maxIdleTime}"/>

<!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->

<property name="acquireIncrement" value="${datasource.connection.acquireIncrement}"/>

<!--JDBC的标准参数，用以控制数据源内加载的PreparedStatements数量。但由于预缓存的statements 属于单个connection而不是整个连接池。所以设置这个参数需要考虑到多方面的因素。

如果maxStatements与maxStatementsPerConnection均为0，则缓存被关闭。Default: 0 -->

<property name="maxStatements" value="${datasource.connection.maxStatements}"/>

<!--maxStatementsPerConnection定义了连接池内单个连接所拥有的最大缓存statements数。Default: 0 -->

<property name="maxStatementsPerConnection"

value="${datasource.connection.maxStatementsPerConnection}"/>

<!--初始化时获取三个连接，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->

<property name="initialPoolSize" value="${datasource.connection.initialPoolSize}"/>

<!--每60秒检查所有连接池中的空闲连接。Default: 0 -->

<property name="idleConnectionTestPeriod"

value="${datasource.connection.idleConnectionTestPeriod}"/>

<!--定义在从数据库获取新连接失败后重复尝试的次数。Default: 30 -->

<property name="acquireRetryAttempts"

value="${datasource.connection.acquireRetryAttempts}"/>

<!--获取连接失败将会引起所有等待连接池来获取连接的线程抛出异常。但是数据源仍有效 保留，并在下次调用getConnection()的时候继续尝试获取连接。如果设为true，那么在尝试

获取连接失败后该数据源将申明已断开并永久关闭。Default: false -->

<property name="breakAfterAcquireFailure"

value="${datasource.connection.breakAfterAcquireFailure}"/>

<!--因性能消耗大请只在需要的时候使用它。如果设为true那么在每个connection提交的 时候都将校验其有效性。建议使用idleConnectionTestPeriod或automaticTestTable

等方法来提升连接测试的性能。Default: false -->

<property name="testConnectionOnCheckout"

value="${datasource.connection.testConnectionOnCheckout}"/>

<property name="checkoutTimeout" value="${datasource.connection.checkoutTimeout}"/>

<property name="testConnectionOnCheckin"

value="${datasource.connection.testConnectionOnCheckin}"/>

<property name="automaticTestTable" value="${datasource.connection.automaticTestTable}"/>

<property name="acquireRetryDelay" value="${datasource.connection.acquireRetryDelay}"/>

<!--自动超时回收Connection-->

<property name="unreturnedConnectionTimeout" value="${datasource.connection.unreturnedConnectionTimeout}"/>

<!--超时自动断开-->

<property name="maxIdleTimeExcessConnections" value="${datasource.connection.maxIdleTimeExcessConnections}"/>

<property name="maxConnectionAge" value="${datasource.connection.maxConnectionAge}"/>

</bean>

<!-- 事务管理器 （JDBC） -->

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"></property>

</bean>

<!-- 启动声明式事务驱动 -->

<tx:annotation-driven transaction-manager="transactionManager" />

<!-- spring 通过 sqlSessionFactoryBean 获取 sqlSessionFactory 工厂类 -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource"></property>

<!-- 扫描 po 包，使用别名 -->

<property name="typeAliasesPackage" value="com.ctgu.fy.entity"></property>

<!-- 扫描映射文件 -->

<property name="mapperLocations" value="classpath:config/mybatis/mapper/\*.xml"></property>

</bean>

<!-- 配置扫描 dao 包，动态实现 dao 接口，注入到 spring 容器中 -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="com.ctgu.fy.dao" />

<!-- 注意使用 sqlSessionFactoryBeanName 避免出现spring 扫描组件失效问题 -->

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />

</bean>

<bean id="gson" class="com.google.gson.Gson" scope="prototype"></bean>

</beans>

### 6.4. 日志文件log4j.properties

### direct log message to stdout ###

log4j.appender.stdout.Target = System.out

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout = org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern = %d{ABSOLUTE} %5p %c{1}:%L - %m%n

log4j.rootLogger=INFO, stdout

### 6.5. 数据库配置文件dp.properties

datasource.connection.driver\_class=com.mysql.cj.jdbc.Driver

datasource.connection.url=jdbc:mysql://localhost:3306/db\_student\_ssm?useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC

datasource.connection.username=root

datasource.connection.password=loveisyou0

#连接池保持的最小连接数,default : 3（建议使用）

datasource.connection.minPoolSize=3

#连接池中拥有的最大连接数，如果获得新连接时会使连接总数超过这个值则不会再获取新连接，而是等待其他连接释放，所以这个值有可能会设计地很大,default : 15（建议使用）

datasource.connection.maxPoolSize=15

#连接的最大空闲时间，如果超过这个时间，某个数据库连接还没有被使用，则会断开掉这个连接。如果为0，则永远不会断开连接,即回收此连接。default : 0 单位 s（建议使用）

datasource.connection.maxIdleTime=0

#连接池在无空闲连接可用时一次性创建的新数据库连接数,default : 3（建议使用）

datasource.connection.acquireIncrement=3

#连接池为数据源缓存的PreparedStatement的总数。由于PreparedStatement属于单个Connection,所以这个数量应该根据应用中平均连接数乘以每个连接的平均PreparedStatement来计算。同时maxStatementsPerConnection的配置无效。default : 0（不建议使用）

datasource.connection.maxStatements=0

#连接池为数据源单个Connection缓存的PreparedStatement数，这个配置比maxStatements更有意义，因为它缓存的服务对象是单个数据连接，如果设置的好，肯定是可以提高性能的。为0的时候不缓存。default : 0（看情况而论）

datasource.connection.maxStatementsPerConnection=0

#连接池初始化时创建的连接数,default : 3（建议使用）

datasource.connection.initialPoolSize=3

#用来配置测试空闲连接的间隔时间。测试方式还是上面的两种之一，可以用来解决MySQL8小时断开连接的问题。因为它保证连接池会每隔一定时间对空闲连接进行一次测试，从而保证有效的空闲连接能每隔一定时间访问一次数据库，将于MySQL8小时无会话的状态打破。为0则不测试。default : 0(建议使用)

datasource.connection.idleConnectionTestPeriod=0

#连接池在获得新连接失败时重试的次数，如果小于等于0则无限重试直至连接获得成功。default : 30（建议使用）

datasource.connection.acquireRetryAttempts=30

#如果为true，则当连接获取失败时自动关闭数据源，除非重新启动应用程序。所以一般不用。default : false（不建议使用）

datasource.connection.breakAfterAcquireFailure=false

#性能消耗大。如果为true，在每次getConnection的时候都会测试，为了提高性能,尽量不要用。default : false（不建议使用）

datasource.connection.testConnectionOnCheckout=false

#配置当连接池所有连接用完时应用程序getConnection的等待时间。为0则无限等待直至有其他连接释放或者创建新的连接，不为0则当时间到的时候如果仍没有获得连接，则会抛出SQLException。其实就是acquireRetryAttempts\*acquireRetryDelay。default : 0（与上面两个，有重复，选择其中两个都行）

datasource.connection.checkoutTimeout=30000

#如果为true，则在close的时候测试连接的有效性。default : false（不建议使用）

datasource.connection.testConnectionOnCheckin=false

#配置一个表名，连接池根据这个表名用自己的测试sql语句在这个空表上测试数据库连接,这个表只能由c3p0来使用，用户不能操作。default : null（不建议使用）

datasource.connection.automaticTestTable=c3p0TestTable

#连接池在获得新连接时的间隔时间。default : 1000 单位ms（建议使用）

datasource.connection.acquireRetryDelay=1000

#为0的时候要求所有的Connection在应用程序中必须关闭。如果不为0，则强制在设定的时间到达后回收Connection，所以必须小心设置，保证在回收之前所有数据库操作都能够完成。这种限制减少Connection未关闭情况的不是很适用。建议手动关闭。default : 0 单位 s（不建议使用）

datasource.connection.unreturnedConnectionTimeout=0

#这个配置主要是为了快速减轻连接池的负载，比如连接池中连接数因为某次数据访问高峰导致创建了很多数据连接，但是后面的时间段需要的数据库连接数很少，需要快速释放，必须小于maxIdleTime。其实这个没必要配置，maxIdleTime已经配置了。default : 0 单位 s（不建议使用）

datasource.connection.maxIdleTimeExcessConnections=0

#配置连接的生存时间，超过这个时间的连接将由连接池自动断开丢弃掉。当然正在使用的连接不会马上断开，而是等待它close再断开。配置为0的时候则不会对连接的生存时间进行限制。default : 0 单位 s（不建议使用）

datasource.connection.maxConnectionAge=0

## 七．参考文献

[1] 陈强，《精通Java开发技术》，清华大学出版社.2016

[2] 李刚，《轻量级Java EE企业用用实战》第5版，电子工业出版社.2017

[3] 黄俊，《Java程序设计与应用开发》第2版，机械工业出版社.2016

[4] 杨开振，《Java EE互联网轻量级框架整合开发》，电子工业出版社.2017

[5] 张剑飞，《Java EE开发技术》，哈尔滨工业大学出版社.2015

[6] 疯狂软件，《Spring+MyBatis企业应用实战》，电子工业出版社.2017

[7] 李燕飞，《MVC模式在J2EE中的应用》，电脑知识与技术.2017

[8] 向昌成，Java程序设计项目化教程，清华大学出版社.2016

[9] 霍斯特曼，《JAVA核心技术 卷I：基础知识》，机械工业出版社，2016

[10] Bruce Eckel，《Java编程思想》，机械工业出版社，2007