|  |  |
| --- | --- |
| **项目编号** |  |
| **文档编号** |  |
| **密级** |  |

**课程资料自主学习系统详细设计**

**V1.0**

**广西民族大学**

评审日期： 2021年6月16日

目录

[1．导言 4](#_Toc444248412)

[1.1 目的 4](#_Toc444248413)

[1.2 范围 4](#_Toc444248414)

[1.3 引用标准 5](#_Toc444248415)

[1.4 参考资料 5](#_Toc444248416)

[1.5 版本更新信息 5](#_Toc444248417)

[2．详细设计简述 6](#_Toc444248418)

[2.1 设计简介 6](#_Toc444248419)

[2.2 模块简介 6](#_Toc444248420)

[3、界面详细设计 7](#_Toc444248421)

[3.1、页面设计说明 7](#_Toc444248422)

[3.2、静态页面详细设计 7](#_Toc444248423)

[4、客户端模块详细设计 7](#_Toc444248424)

[4.1、公用模块的详细设计 7](#_Toc444248425)

[4.1.1 表现层 8](#_Toc444248426)

[4.1.2 控制层 8](#_Toc444248427)

[4.2、登录模块 12](#_Toc444248429)

[4.2.1 表现层 12](#_Toc444248430)

[4.2.2 控制层 14](#_Toc444248431)

[4.n、XXXX模块 20](#_Toc444248435)

[4.n.1 表现层 20](#_Toc444248436)

[4.n.2 控制层 21](#_Toc444248437)

[4.n.3 业务逻辑层 21](#_Toc444248438)

[4.n.4 域模型层 21](#_Toc444248440)

[5、管理端模块详细设计 21](#_Toc444248441)

[5.1、XXXXX模块详细设计 21](#_Toc444248442)

[5.1.1 表现层 21](#_Toc444248443)

[5.1.2 控制层 21](#_Toc444248444)

[5.1.3 业务逻辑层 21](#_Toc444248445)

[5.1.4 域模型层 22](#_Toc444248447)

[5.2、XXXXX模块详细设计 22](#_Toc444248448)

[5.n.1 表现层 22](#_Toc444248449)

[5.n.2 控制层 22](#_Toc444248450)

[5.n.3 业务逻辑层 23](#_Toc444248451)

[5.n.4 域模型层 23](#_Toc444248453)

[6、公共部分模块详细设计 23](#_Toc444248454)

[6.1 公共页面 23](#_Toc444248455)

[6.2 安全模块的详细设计 23](#_Toc444248456)

[6.n XXXXX 23](#_Toc444248457)

# 1．导言

## 1.1 目的

该文档是对课程资料自主学习系统概要设计进行的详细说明，是为了指导和规范课程资料自主学习系统概开发而制定的详细开发设计文档

本文档的预期读者包括：

* 设计开发人员
* 项目管理人员
* 测试人员
* 用户

## 1.2 范围

该文档内容涵盖课程资料自主学习系统的所有功能模块的详细设计描述

## 1.3 引用标准

[1] 《软件工程案例教程 第2版》 韩万江等 机械工业出版社

## 1.4 参考资料

[1] 《软件项目管理案例教程 第3版》 韩万江等 机械工业出版社

## 1.5 版本更新信息

本文档的更新记录如下表1-1所示。

表1-1 文档更新记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修改编号 | 修改日期 | 修改后版本 | 修改位置 | 修改内容概述 |
| 001 | 2021.6.12 | 0.1 | 全部 | 初始发布版本 |
| 002 | 2021.6.13 | 1.0 | 4章节 | 增加 |
| 003 | 2021.6.13 | 2.0 | 4章节 | 修改 |
| 004 | 2021.6.16 | 3.0 | 4、6章节 | 修改 |
| 005 | 2021.6.18 | 4.0 | 6、7章节 | 修改、增加 |
| 006 | 2021.6.19 | 5.0 | 5、7章节 | 修改、增加 |
| 007 | 2021.6.20 | 6.0 | 4、6章节 | 修改 |
| 008 | 2021.6.22 | 7.0 | 5、7章节 | 修改 |
| 009 | 2021.6.23 | 8．0 | 全部 | 修改，增加 |

# 2．详细设计简述

## 2.1 设计简介

分层结构图如图2-1所示

图 2-1 课程资料自主学习系统结构示意图

## 2.2 模块简介

系统功能模块图如图2.-2所示。

# 

表2.2 课程资料自主学习系统

# 3、界面详细设计

[界面详细设计 包括界面设计的技术，以及静态页面设计]

## 3.1、页面设计说明

系统页面的设计采用了HTML/CSS/JS的开发方式来进行网页界面的开发，并且使用了layui、echarts、jQuery、fullcalendar、tinymce等第三方库支持开发

## 3.2、静态页面详细设计

课程资料自主学习系统静态页面见表3-1

表3-1 页面表现层设计

|  |  |  |
| --- | --- | --- |
| 界面 | Html | 功能描述 |
| 系统登录页 | Login.html | 提示登录信息，提供用户登录系统的入口 |
| 用户系统主页 | Users/Index.html | 显示系统为用户提供的功能菜单与客户查看个人信息与选择课程等各项功能的按钮 |
| 用户订单页面 | Order.html | 显示用户已购买的课程的订单号以及课程的信息 |
| 课程资料首页 | Default.html  Course.html | 显示可供客户选择的课程，并且提供标签与搜索框供客户筛选寻找目标课程 |
| 课程选择页面 | Detail/x.html（x为每门课程对应的编号） | 点击后进入课程的详细信息页面，客户可从该页面进行课程的选择与购买 |
| 管理员/教师系统页面 | Index.html | 提供系统管理、课程管理、教师管理、新闻管理等模块 |
| 权限管理 | Teacher.html | 对教师账号权限进行修改 |
| 专业课程管理 | Zykc.html | 对专业课程进行管理 |
| 免费课程管理 | Mfkc.html | 对免费课程进行管理 |
| 公共课程管理 | Ggkc.htm | 对公共课程进行管理 |
| 课程管理 | Course.html | 对课程进行增删查改等操作 |
| 类别管理 | classify.html | 对课程的类别进行修改 |
| 视频管理 | Vdio.html | 提供删除视频按钮 |

# 4、客户端模块详细设计

客户端模块主要包括学生、教师/管理员两大模块模块。

## 4.1、公用模块的详细设计

[公用模块是指在系统在运行过程中将会被其他模块调用的基础资源]。

### 4.1.1 表现层

公用模块的表现层主要是一些页面出错时的跳转页面，显示当前错误的结果处理报告。具体见表？-？。

表 4-1 课程资料自主学习系统错误提示页面

|  |  |  |
| --- | --- | --- |
| 界面 | Html | 功能描述 |
| 登录出错页面 | 403.html 404.html | 登入时出现严重错误时跳转到该页面。 |
| 统一异常页面 | 500.html | 在系统运行时出现未知错误时跳转到该页面。 |

### 4.1.2 控制层

[公共模块的控制层: ]

主要是通过 getLoginUser()方法来获取当前尝试登录的用户，然后通过getLoginUserId()方法判断登录的用户命是否为空，如果为空则返回错误页面

public class BaseController {  
 */\*\*  
 \* 获取当前登录的user  
 \*/* public User getLoginUser() {  
 Subject subject = SecurityUtils.*getSubject*();  
 if (subject == null) return null;  
 Object object = subject.getPrincipal();  
 if (object != null) return (User) object;  
 return null;  
 }  
 */\*\*  
 \* 获取当前登录的userId  
 \*/* public Integer getLoginUserId() {  
 User loginUser = getLoginUser();  
 return loginUser == null ? null : loginUser.getUserId();  
 }  
}

## 4.2、登录模块

在登录模块时，系统内部的相应响应操作示意图如图4-2所示。

图 4-2 系统响应示意图

### 4.2.1 表现层

登录模块的表现层主要完成不同用户的登录功能，登录页面时要求用户输入账号密码的基本信息，确认后页面给出响应消息，提示登录成功或失败的提示。表现层对应的Html页面列表见表4-3所示。

表 4-3 登入模块表现层Html列表

|  |  |  |
| --- | --- | --- |
| 界面 | Html | 功能描述 |
| 登录页面 | login.html | 用户（教师、学生）登入功能，当登入出错时给出提示。 |

login.html的流程图如图4-3所示。

进入游首页



图 4-3 登录模块详细设计流程图

### 4.2.2 控制层

登录模块的控制层负责接受来自login.Html的用户输入，同时调用登入模块的业务逻辑接口，将用户名与密码等用户关键信息传递到业务逻辑层进行判定。等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并决定显示页面。登入模块控制层列表见表4-4所示。

表 4-4 登入模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 登录 | MainController.java  UserrController.java | SUCCESS | mainFrame.Html—登入成功，显示提示窗口。 |
| ERROR | login.Html---登入失败，显示提示窗口。 |

在控制层中MainController.java的描述如下所示：

@Controller  
public class MainController extends BaseController implements ErrorController {  
 @Autowired  
 private MenuService menuService;  
 @Autowired  
 private LoginRecordService loginRecordService;  
 */\*\*  
 \* 用户登录  
 \*/* @ResponseBody  
 @PostMapping("/login")  
 public JsonResult login(String username, String password, String code, Boolean remember, HttpServletRequest request) {  
 if (username == null || username.trim().isEmpty()) return JsonResult.*error*("请输入账号");  
 if (!CaptchaUtil.*ver*(code, request)) {  
 loginRecordService.saveAsync(username, LoginRecord.*TYPE\_ERROR*, "验证码错误", request);  
 return JsonResult.*error*("验证码不正确");  
 }  
 try {  
 if (remember == null) remember = false;  
 SecurityUtils.*getSubject*().login(new CustomizedToken(username, password, remember,"Admin"));  
 loginRecordService.saveAsync(username, request);  
 return JsonResult.*ok*("登录成功");  
 } catch (IncorrectCredentialsException ice) {  
 loginRecordService.saveAsync(username, LoginRecord.*TYPE\_ERROR*, "密码错误", request);  
 return JsonResult.*error*("密码错误");  
 } catch (UnknownAccountException uae) {  
 loginRecordService.saveAsync(username, LoginRecord.*TYPE\_ERROR*, "账号不存在", request);  
 return JsonResult.*error*("账号不存在");  
 } catch (LockedAccountException e) {  
 loginRecordService.saveAsync(username, LoginRecord.*TYPE\_ERROR*, "账号被锁定", request);  
 return JsonResult.*error*("账号被锁定");  
 } catch (ExcessiveAttemptsException eae) {  
 loginRecordService.saveAsync(username, LoginRecord.*TYPE\_ERROR*, "操作频繁", request);  
 return JsonResult.*error*("操作频繁，请稍后再试");  
 }  
 }  
 */\*\*  
 \* 登录页  
 \*/* @GetMapping("/login")  
 public String login() {  
 if (getLoginUser() != null) return "redirect:index";  
 return "login.html";  
 }  
 */\*\*  
 \* 主页  
 \*/* @RequestMapping({"/", "/index"})  
 public String index(Model model) {  
 // 左侧菜单  
 List<Menu> menus = menuService.getUserMenu(getLoginUserId(), Menu.*TYPE\_MENU*);  
 model.addAttribute("menus", menuService.toMenuTree(menus, 0));  
 return "index.html";  
 }  
 */\*\*  
 \* 图形验证码  
 \*/* @RequestMapping("/assets/captcha")  
 public void captcha(HttpServletRequest request, HttpServletResponse response) {  
 try {  
 CaptchaUtil.*out*(5, request, response);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* 主页弹窗页面  
 \*/* @RequestMapping("/tpl/{name}")  
 public String tpl(@PathVariable("name") String name) {  
 return "index/" + name + ".html";  
 }  
 */\*\*  
 \* 错误页  
 \*/* @RequestMapping("/error")  
 public String error() {  
 return "error/404.html";  
 }  
 @Override  
 public String getErrorPath() {  
 return "/error";  
 }  
}

在控制层中UserrController.java的描述如下所示：

@Controller  
@RequestMapping("/users")  
public class UserrController extends BaseController{  
 @Autowired  
 private MenurService menurService;  
 @Autowired  
 private UserTwoService userTwoService;  
 @RequestMapping("/index")  
 public String index(Model model) {  
 Subject subject = SecurityUtils.*getSubject*();  
 Object object = subject.getPrincipal();  
 if(object != null) {  
 model.addAttribute("user", (UserTwo)object);  
 model.addAttribute("menu", menurService.list());  
 return "/user/index.html";  
 }else {  
 UserTwo user2 = new UserTwo();  
 user2.setUsername("游客");  
 user2.setUserId(-1);  
 model.addAttribute("user", user2);  
 model.addAttribute("menu", menurService.list());  
 return "default/login.html";  
 }   
 }  
 @RequestMapping("/order")  
 public String order(Model model) {  
 Subject subject = SecurityUtils.*getSubject*();  
 Object object = subject.getPrincipal();  
 if(object != null) {  
 model.addAttribute("user", (UserTwo)object);  
 model.addAttribute("menu", menurService.list());  
 return "/user/order.html";  
 }else {  
 UserTwo user2 = new UserTwo();  
 user2.setUsername("游客");  
 user2.setUserId(-1);  
 model.addAttribute("user", user2);  
 model.addAttribute("menu", menurService.list());  
 return "default/login.html";  
 }  
 }  
 @ResponseBody  
 @PostMapping("/repass")  
 public JsonResult repass(String username, String pass,String repass) {  
 UserTwo userTwo = new UserTwo();  
 userTwo.setUserId(userTwoService.getByUsername(username).getUserId());  
 if(!pass.equals(repass)){  
 return JsonResult.*error*("两次密码输入不一值");  
 }else {  
 userTwo.setPassword(userTwoService.encodePsw(pass));  
 userTwoService.updateById(userTwo);  
 }  
 return JsonResult.*ok*("修改成功");  
 }  
}

## 4.3、学生模块

**4.3.1** 个人信息模块

学生模块的个人信息模块，系统内部的相应响应操作示意图如图4-4所示。

图 4-4 系统响应示意图

### 4.3.1.1 表现层

个人信息页面的表现层主要完成显示用户个人信息，登录页面时要求用户输入账号密码的基本信息，确认后页面给出响应消息，提示登录成功或失败的提示。表现层对应的Html页面列表见表?-?所示。

表 4-5 登入模块表现层Html列表

|  |  |  |
| --- | --- | --- |
| 界面 | Html | 功能描述 |
| 学生个人信息页面 | Users/Index.Html | 学生点击个人信息页面按钮时进入个人信息页面并且显示个人信息 |

Users/Index.Html的流程图如图4-5所示。



### 4.3.1.2 控制层

。。。。。。。

### 4.3.1.3 业务逻辑层

。。。。。。

### 4.3.1.4 域模型层

。。。。。。

**4.3.2** 课程订单模块

学生模块的课程订单模块，系统内部的相应响应操作示意图如图4-4所示。

图 4-6 系统响应示意图

### 4.3.2.1 表现层

个人信息页面的表现层主要完成显示用户个人信息，登录页面时要求用户输入账号密码的基本信息，确认后页面给出响应消息，提示登录成功或失败的提示。表现层对应的Html页面列表见表4-5所示。

表 4-5 登入模块表现层Html列表

|  |  |  |
| --- | --- | --- |
| 界面 | Html | 功能描述 |
| 学生个人信息页面 | Users/Index.html | 学生点击个人信息页面按钮时进入个人信息页面并且显示个人信息 |

Users/Index.html的流程图如图4-4所示。



### 4.3.2.2 控制层

个人信息查询模块的控制层负责接受来自index.html的用户请求，同时调用查询用户个人信息模块的业务逻辑接口，等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并决定显示页面。个人信息查询模块控制层列表见表4-4所示。

表 4-4 个人信息查询模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 登录 | UserrController.java | SUCCESS | 拉取个人信息成功后显示信息窗口 |
| ERROR | 个人信息拉取失败，显示提示窗口。 |

在控制层中UserrController.java的描述如下所示：

@Controller  
@RequestMapping("/users")  
public class UserrController extends BaseController{  
 @Autowired  
 private MenurService menurService;  
 @Autowired  
 private UserTwoService userTwoService;  
 @RequestMapping("/index")  
 public String index(Model model) {  
 Subject subject = SecurityUtils.*getSubject*();  
 Object object = subject.getPrincipal();  
 if(object != null) {  
 model.addAttribute("user", (UserTwo)object);  
 model.addAttribute("menu", menurService.list());  
 return "/user/index.html";  
 }else {  
 UserTwo user2 = new UserTwo();  
 user2.setUsername("游客");  
 user2.setUserId(-1);  
 model.addAttribute("user", user2);  
 model.addAttribute("menu", menurService.list());  
 return "default/login.html";  
 }   
 }  
 @RequestMapping("/order")  
 public String order(Model model) {  
 Subject subject = SecurityUtils.*getSubject*();  
 Object object = subject.getPrincipal();  
 if(object != null) {  
 model.addAttribute("user", (UserTwo)object);  
 model.addAttribute("menu", menurService.list());  
 return "/user/order.html";  
 }else {  
 UserTwo user2 = new UserTwo();  
 user2.setUsername("游客");  
 user2.setUserId(-1);  
 model.addAttribute("user", user2);  
 model.addAttribute("menu", menurService.list());  
 return "default/login.html";  
 }  
 }  
 @ResponseBody  
 @PostMapping("/repass")  
 public JsonResult repass(String username, String pass,String repass) {  
 UserTwo userTwo = new UserTwo();  
 userTwo.setUserId(userTwoService.getByUsername(username).getUserId());  
 if(!pass.equals(repass)){  
 return JsonResult.*error*("两次密码输入不一值");  
 }else {  
 userTwo.setPassword(userTwoService.encodePsw(pass));  
 userTwoService.updateById(userTwo);  
 }  
 return JsonResult.*ok*("修改成功");  
 }  
}

### 4.3.2.3 域模型层

个人信息查询模块用到域模型层中User.java，User.java是一个公用域模型，在涉及到用户信息查询等操作时，就会调用到该模型，个人信息查询域模型层列表如表4-7所示。

表 4-7登入模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| User.java | 对用户信息的增、删、改、查操作 |

User.java主要属性与方法：

@TableName("sys\_user")  
public class User implements Serializable {  
 private static final long *serialVersionUID* = 242146703513492331L;  
 */\*\*  
 \* 用户id  
 \*/* @TableId(value = "user\_id", type = IdType.*AUTO*)  
 private Integer userId;  
 */\*\*  
 \* 账号  
 \*/* private String username;  
 */\*\*  
 \* 密码  
 \*/* private String password;  
 */\*\*  
 \* 昵称  
 \*/* private String nickName;  
 */\*\*  
 \* 头像  
 \*/* private String avatar;  
 */\*\*  
 \* 性别  
 \*/* private Integer sex;  
 */\*\*  
 \* 手机号  
 \*/* private String phone;  
 */\*\*  
 \* 邮箱  
 \*/* private String email;  
 */\*\*  
 \* 邮箱是否验证,0否,1是  
 \*/* private Integer emailVerified;  
 */\*\*  
 \* 真实姓名  
 \*/* private String trueName;  
 */\*\*  
 \* 身份证号  
 \*/* private String idCard;  
 */\*\*  
 \* 出生日期  
 \*/* private Date birthday;  
 */\*\*  
 \* 个人简介  
 \*/* private String introduction;  
 */\*\*  
 \* 机构id  
 \*/* private Integer organizationId;  
 */\*\*  
 \* 状态，0正常，1冻结  
 \*/* private Integer state;  
 */\*\*  
 \* 注册时间  
 \*/* private Date createTime;  
 */\*\*  
 \* 修改时间  
 \*/* private Date updateTime;  
 */\*\*  
 \* 是否删除,0否,1是  
 \*/* @TableLogic  
 private Integer deleted;  
 */\*\*  
 \* 权限列表  
 \*/* @TableField(exist = false)  
 private List<String> authorities;  
 */\*\*  
 \* 角色列表  
 \*/* @TableField(exist = false)  
 private List<Role> roles;  
 */\*\*  
 \* 角色id  
 \*/* @TableField(exist = false)  
 private List<Integer> roleIds;  
 */\*\*  
 \* 机构名称  
 \*/* @TableField(exist = false)  
 private String organizationName;  
 */\*\*  
 \* 性别名称  
 \*/* @TableField(exist = false)  
 private String sexName;  
 public Integer getUserId() {  
 return userId;  
 }  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
 public String getUsername() {  
 return username;  
 }  
 public void setUsername(String username) {  
 this.username = username;  
 }  
 public String getPassword() {  
 return password;  
 }  
 public void setPassword(String password) {  
 this.password = password;  
 }  
 public String getNickName() {  
 return nickName;  
 }  
 public void setNickName(String nickName) {  
 this.nickName = nickName;  
 }  
 public String getAvatar() {  
 return avatar;  
 }  
 public void setAvatar(String avatar) {  
 this.avatar = avatar;  
 }  
 public Integer getSex() {  
 return sex;  
 }  
 public void setSex(Integer sex) {  
 this.sex = sex;  
 }  
 public String getPhone() {  
 return phone;  
 }  
 public void setPhone(String phone) {  
 this.phone = phone;  
 }  
 public String getEmail() {  
 return email;  
 }  
 public void setEmail(String email) {  
 this.email = email;  
 }  
 public Integer getEmailVerified() {  
 return emailVerified;  
 }  
 public void setEmailVerified(Integer emailVerified) {  
 this.emailVerified = emailVerified;  
 }  
 public String getTrueName() {  
 return trueName;  
 }  
 public void setTrueName(String trueName) {  
 this.trueName = trueName;  
 }  
 public String getIdCard() {  
 return idCard;  
   
 public void setIdCard(String idCard) {  
 this.idCard = idCard;  
 }  
 public Date getBirthday() {  
 return birthday;  
 }  
 public void setBirthday(Date birthday) {  
 this.birthday = birthday;  
 }  
 public String getIntroduction() {  
 return introduction;  
 }  
 public void setIntroduction(String introduction) {  
 this.introduction = introduction;  
 }  
 public Integer getOrganizationId() {  
 return organizationId;  
 }  
 public void setOrganizationId(Integer organizationId) {  
 this.organizationId = organizationId;  
 }  
 public Integer getState() {  
 return state;  
 }  
 public void setState(Integer state) {  
 this.state = state;  
 }  
 public Date getCreateTime() {  
 return createTime;  
 }  
 public void setCreateTime(Date createTime) {  
 this.createTime = createTime;  
 }  
 public Date getUpdateTime() {  
 return updateTime;  
 }  
 public void setUpdateTime(Date updateTime) {  
 this.updateTime = updateTime;  
 }  
 public Integer getDeleted() {  
 return deleted;  
 }  
 public void setDeleted(Integer deleted) {  
 this.deleted = deleted;  
 }  
 public List<String> getAuthorities() {  
 return authorities;  
 }  
 public void setAuthorities(List<String> authorities) {  
 this.authorities = authorities;  
 }  
 public List<Role> getRoles() {  
 return roles;  
 }  
 public void setRoles(List<Role> roles) {  
 this.roles = roles;  
 }  
 public List<Integer> getRoleIds() {  
 return roleIds;  
 }  
 public void setRoleIds(List<Integer> roleIds) {  
 this.roleIds = roleIds;  
 }  
 public String getOrganizationName() {  
 return organizationName;  
 }  
 public void setOrganizationName(String organizationName) {  
 this.organizationName = organizationName;  
 }  
 public String getSexName() {  
 return sexName;  
 }  
 public void setSexName(String sexName) {  
 this.sexName = sexName;  
 }  
 @Override  
 public String toString() {  
 return "User{" +  
 ", userId=" + userId +  
 ", username=" + username +  
 ", password=" + password +  
 ", nickName=" + nickName +  
 ", avatar=" + avatar +  
 ", sex=" + sex +  
 ", phone=" + phone +  
 ", email=" + email +  
 ", emailVerified=" + emailVerified +  
 ", trueName=" + trueName +  
 ", idCard=" + idCard +  
 ", birthday=" + birthday +  
 ", introduction=" + introduction +  
 ", organizationId=" + organizationId +  
 ", state=" + state +  
 ", createTime=" + createTime +  
 ", updateTime=" + updateTime +  
 ", deleted=" + deleted +  
 ", organizationName=" + organizationName +  
 "}";  
 }  
}

**4.3.1** 课程选择模块

学生模块的课程选择模块，系统内部的相应响应操作示意图如图4-5所示。

图 4-7 系统响应示意图

### 4.3.3.1 表现层

课程选择页面的表现层主要完成显示所有可选课程的信息提供客户进行课程选择。表现层对应的Html页面列表见表4-5所示。

表 4-7 登入模块表现层Html列表

|  |  |  |
| --- | --- | --- |
| 界面 | Html | 功能描述 |
| 课程页面 | Course.html | 学生点击选择课程页面按钮时进入课程选择页面并且可选课程信息 |

Course.html的流程图如图4-8所示。

### 

### 4.3.3.2 控制层

课程选择模块的控制层负责接受来自index.html的用户请求，向服务器请求可选课程的数据，等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并显示选课页面。课程选购模块控制层列表见表4-8所示。

表 4-8 个人信息查询模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 课程选择 | OrderController.java | SUCCESS | 拉取可选购课程信息成功后显示信息窗口 |
| ERROR | 课程信息拉取失败，显示提示窗口。 |

在控制层中OrderController.java的描述如下所示：

@Controller  
@RequestMapping("/course/order")  
public class OrderController extends BaseController {  
 @Autowired  
 private OrderService orderService;  
 @RequestMapping()  
 public String view() {  
 return "course/order.html";  
 }  
 */\*\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Order> page(HttpServletRequest request) {  
 PageParam<Order> pageParam = new PageParam<>(request);  
 return new PageResult<>(orderService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Order> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(orderService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(orderService.getById(id));  
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Order order) {  
 if (orderService.save(order)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Order order) {  
 if (orderService.updateById(order)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (orderService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Order> list) {  
 if (orderService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Order> batchParam) {  
 if (batchParam.update(orderService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (orderService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
}

@Controller  
@RequestMapping("/course/order")  
public class OrderController extends BaseController {  
 @Autowired  
 private OrderService orderService;  
 @RequestMapping()  
 public String view() {  
 return "course/order.html";  
 }  
 */\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Order> page(HttpServletRequest request) {  
 PageParam<Order> pageParam = new PageParam<>(request);  
 return new PageResult<>(orderService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
   
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Order> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(orderService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(orderService.getById(id));  
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Order order) {  
 if (orderService.save(order)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Order order) {  
 if (orderService.updateById(order)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (orderService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Order> list) {  
 if (orderService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Order> batchParam) {  
 if (batchParam.update(orderService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (orderService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
}

### 4.3.3.3 业务逻辑层

课程选购模块的业务逻辑层主要完成对用户登录逻辑的判定，同时调用课程选购模块的业务逻辑接口。课程选购模块业务逻辑层列表如图4-5所示。

表 4-5课程选购模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 显示可选购课程页面 | OrderService.java  OrderServiceImpl.java | 无 | 无 |

在登入模块的业务逻辑层是调用了公用的OrderService接口，同时在实现该接口。

OrderService.java接口主要方法：

public interface OrderService extends IService<Order> {  
 */\*\*  
 \* 分页查询  
 \*/* PageResult<Order> listPage(PageParam<Order> page)  
 */\*\*  
 \* 查询所有  
 \*/* List<Order> listAll(Map<String, Object> page);  
}

OrderServiceImpl.java实现了OrderService.java接口

OrderServiceImpl.java主要实现属性与方法:

@Controller  
@RequestMapping("/course/order")  
public class OrderController extends BaseController {  
 @Autowired  
 private OrderService orderService;  
 @RequestMapping()  
 public String view() {  
 return "course/order.html";  
 }  
 */\*\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Order> page(HttpServletRequest request) {  
 PageParam<Order> pageParam = new PageParam<>(request);  
 return new PageResult<>(orderService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 //return classifyService.listPage(pageParam); // 使用关联查询  
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Order> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(orderService.list(pageParam.getOrderWrapper()));  
 //List<Classify> records = classifyService.listAll(pageParam.getNoPageParam()); // 使用关联查询  
 //return JsonResult.ok().setData(pageParam.sortRecords(records));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(orderService.getById(id));  
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Order order) {  
 if (orderService.save(order)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Order order) {  
 if (orderService.updateById(order)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (orderService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Order> list) {  
 if (orderService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Order> batchParam) {  
 if (batchParam.update(orderService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (orderService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
}

### 4.3.3.4 域模型层

课程选购模块用到域模型层中Order.java，Order.java是一个公用域模型，在涉及到用户课程选购操作时，就会调用到该模型，课程选购模块域模型层列表如表4-9所示。

表 4-9课程选购域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Order.java | 拉取可选购课程信息并且展示给用户 |

Order.java主要属性与方法：

@TableName("t\_order")  
public class Order implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* 主键  
 \*/* @TableId(value = "id", type = IdType.*AUTO*)  
 private Integer id;  
 */\*\*  
 \* 名称  
 \*/* private String name;  
 */\*\*  
 \* 排序  
 \*/* private Integer userId;  
 */\*\*  
 \* 备注  
 \*/* private String orderNo;  
 private String type;  
 private Date createTime;  
 private String jine;  
 private Integer courseId;  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public Integer getUserId() {  
 return userId;  
 }  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
 public String getOrderNo() {  
 return orderNo;  
 }  
 public void setOrderNo(String orderNo) {  
 this.orderNo = orderNo;  
 }  
 public String getType() {  
 return type;  
 }  
 public void setType(String type) {  
 this.type = type;  
 }  
 public Date getCreateTime() {  
 return createTime;  
 }  
 public void setCreateTime(Date createTime) {  
 this.createTime = createTime;  
 }  
 public String getJine() {  
 return jine;  
 }  
 public void setJine(String jine) {  
 this.jine = jine;  
 }  
 public Integer getCourseId() {  
 return courseId;  
 }  
 public void setCourseId(Integer courseId) {  
 this.courseId = courseId;  
 }  
 @Override  
 public String toString() {  
 return "Classify{" +  
 ", id=" + id +  
 ", name=" + name +  
 ", userId=" + userId +  
 ", orderNo=" + orderNo +  
 ", type=" + type +  
 ", createTime=" + createTime +  
 ", jine=" + jine +  
 ", courseId=" + courseId +  
 "}";  
 }  
}

# 5、管理端模块详细设计

## 5.1、教师/管理员模块详细设计

**5.1.1 系统管理模块**

### 5.1.2 控制层

系统管理模块的控制层负责接受来自index.html的用户请求，向服务器请求获取用户数据，等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并显示系统管理页面。课程选购模块控制层列表见表4-8所示。

表 5-1 个人信息查询模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 系统管理 | UserController.java  RoleController.java | SUCCESS | 显示用户/角色管理页面 |
| ERROR | 用户信息拉取失败，显示提示窗口。 |

在控制层中UserController.java的描述如下所示：

@Controller  
@RequestMapping("/sys/user")  
public class UserController extends BaseController {  
 @Autowired  
 private UserService userService;  
 @Autowired  
 private UserTwoService userTwoService;  
 @Autowired  
 private DictionaryDataService dictionaryDataService;  
 @Autowired  
 private RoleService roleService;  
 @Autowired  
 private OrganizationService organizationService;  
 @RequiresPermissions("sys:user:view")  
 @RequestMapping()  
 public String view(Model model) {  
 model.addAttribute("sexList", dictionaryDataService.listByDictCode("sex"));  
 model.addAttribute("organizationTypeList", dictionaryDataService.listByDictCode("organization\_type"));  
 model.addAttribute("rolesJson", JSON.*toJSONString*(roleService.list()));  
 return "system/user.html";  
 }  
 */\*\*  
 \* 个人中心  
 \*/* @RequestMapping("/info")  
 public String userInfo(Model model) {  
 model.addAttribute("user", userService.getFullById(getLoginUserId()));  
 model.addAttribute("sexList", dictionaryDataService.listByDictCode("sex"));  
 return "index/user-info.html";  
 }  
 */\*\*  
 \* 分页查询用户  
 \*/* @OperLog(value = "用户管理", desc = "分页查询")  
 @RequiresPermissions("sys:user:list")  
 @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<User> page(HttpServletRequest request) {  
 PageParam<User> pageParam = new PageParam<>(request);  
 pageParam.setDefaultOrder(null, new String[]{"create\_time"});  
 return userService.listPage(pageParam);  
 }  
 */\*\*  
 \* 查询全部用户  
 \*/* @OperLog(value = "用户管理", desc = "查询全部")  
 @RequiresPermissions("sys:user:list")  
 @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<User> pageParam = new PageParam<>(request);  
 List<User> records = userService.listAll(pageParam.getNoPageParam());  
 return JsonResult.*ok*().setData(pageParam.sortRecords(records));  
 }  
 */\*\*  
 \* 根据id查询用户  
 \*/* @OperLog(value = "用户管理", desc = "根据id查询")  
 @RequiresPermissions("sys:user:list")  
 @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 PageParam<User> pageParam = new PageParam<>();  
 pageParam.put("userId", id);  
 List<User> records = userService.listAll(pageParam.getNoPageParam());  
 return JsonResult.*ok*().setData(pageParam.getOne(records));  
 }  
 */\*\*  
 \* 添加用户  
 \*/* @OperLog(value = "用户管理", desc = "添加", param = false, result = true)  
 @RequiresPermissions("sys:user:save")  
 @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(@RequestBody User user) {  
 user.setState(0);  
 user.setPassword(userService.encodePsw(user.getPassword()));  
 UserTwo userTwo = new UserTwo();  
 userTwo.setState(0);  
 userTwo.setPassword(userService.encodePsw(user.getPassword()));  
 userTwo.setUsername(user.getUsername());  
 userTwo.setNickName(user.getNickName());  
 userTwo.setSex(user.getSex());  
 userTwo.setPhone(user.getPhone());  
 userTwo.setRoleIds(user.getRoleIds());  
 if(user.getRoleIds().contains(3)){  
 if (userService.saveUser(user)&&userTwoService.saveUser(userTwo)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 }else {  
 if (userService.saveUser(user)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改用户  
 \*/* @OperLog(value = "用户管理", desc = "修改", param = false, result = true)  
 @RequiresPermissions("sys:user:update")  
 @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(@RequestBody User user) {  
 user.setState(null); // 状态不能修改  
 user.setPassword(null); // 密码不能修改  
 user.setUsername(null); // 账号不能修改  
 if (userService.updateUser(user)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除用户  
 \*/* @OperLog(value = "用户管理", desc = "删除", result = true)  
 @RequiresPermissions("sys:user:remove")  
 @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (userService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量修改用户  
 \*/* @OperLog(value = "用户管理", desc = "批量修改", param = false, result = true)  
 @RequiresPermissions("sys:user:update")  
 @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<User> batchParam) {  
 // 不能修改的字段  
 batchParam.getData().setPassword(null);  
 batchParam.getData().setState(null);  
 batchParam.getData().setUsername(null);  
 batchParam.getData().setPhone(null);  
 batchParam.getData().setEmail(null);  
 if (batchParam.update(userService, "user\_id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除用户  
 \*/* @OperLog(value = "用户管理", desc = "批量删除", result = true)  
 @RequiresPermissions("sys:user:remove")  
 @ResponseBody  
 @RequestMapping("/deleteBatch")  
 public JsonResult deleteBatch(@RequestBody List<Integer> ids) {  
 if (userService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 修改用户状态  
 \*/* @OperLog(value = "用户管理", desc = "修改状态", result = true)  
 @RequiresPermissions("sys:user:update")  
 @ResponseBody  
 @RequestMapping("/state/update")  
 public JsonResult updateState(Integer id, Integer state) {  
 if (state == null || (state != 0 && state != 1)) {  
 return JsonResult.*error*("状态值不正确");  
 }  
 User user = new User();  
 user.setUserId(id);  
 user.setState(state);  
 if (userService.updateById(user)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量修改用户状态  
 \*/* @OperLog(value = "用户管理", desc = "批量修改状态", result = true)  
 @RequiresPermissions("sys:user:update")  
 @ResponseBody  
 @RequestMapping("/state/updateBatch")  
 public JsonResult updateStateBatch(@RequestBody BatchParam<User> batchParam) {  
 User user = new User();  
 user.setState(batchParam.getData().getState());  
 if (user.getState() == null || (user.getState() != 0 && user.getState() != 1)) {  
 return JsonResult.*error*("状态值不正确");  
 }  
 if (batchParam.update(userService, "user\_id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 重置密码  
 \*/* @OperLog(value = "用户管理", desc = "重置密码", param = false, result = true)  
 @RequiresPermissions("sys:user:update")  
 @ResponseBody  
 @RequestMapping("/psw/reset")  
 public JsonResult resetPsw(Integer id,String username,String password) {  
 User user = new User();  
 UserTwo userTwo = new UserTwo();  
 user.setUserId(id);  
 user.setUsername(username);  
 userTwo.setUsername(username);  
 userTwo.setUserId(userTwoService.getByUsername(username).getUserId());  
 user.setPassword(userService.encodePsw(password));  
 userTwo.setPassword(userService.encodePsw(password));  
 if(userService.getByUsername(username)!=null) {  
 userService.updateById(user);  
 }  
 if(userTwoService.getByUsername(username)!=null){  
 userTwoService.updateById(userTwo);  
 }  
 return JsonResult.*ok*("重置成功");  
  
 }  
 */\*\*  
 \* 批量重置密码  
 \*/* @OperLog(value = "用户管理", desc = "批量重置密码", param = false, result = true)  
 @RequiresPermissions("sys:user:update")  
 @ResponseBody  
 @RequestMapping("/psw/resetBatch")  
 public JsonResult resetPswBatch(@RequestBody BatchParam<User> batchParam) {  
 User user = new User();  
 user.setPassword(userService.encodePsw(batchParam.getData().getPassword()));  
 if (batchParam.update(userService, "user\_id")) {  
 return JsonResult.*ok*("重置成功");  
 } else {  
 return JsonResult.*error*("重置失败");  
 }  
 }  
 */\*\*  
 \* 修改自己密码  
 \*/* @OperLog(value = "用户管理", desc = "修改自己密码", param = false, result = true)  
 @ResponseBody  
 @RequestMapping("/psw/update")  
 public JsonResult updatePsw(String oldPsw, String newPsw) {  
 if (StrUtil.*hasBlank*(oldPsw, newPsw)) {  
 return JsonResult.*error*("参数不能为空");  
 }  
 if (getLoginUserId() == null) {  
 return JsonResult.*error*("未登录");  
 }  
 if (!userService.comparePsw(userService.getById(getLoginUserId()).getPassword(), oldPsw)) {  
 return JsonResult.*error*("原密码输入不正确");  
 }  
 User user = new User();  
 user.setUserId(getLoginUserId());  
 user.setPassword(userService.encodePsw(newPsw));  
 if (userService.updateById(user)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 修改自己资料  
 \*/* @OperLog(value = "用户管理", desc = "修改个人信息", param = false, result = true)  
 @ResponseBody  
 @RequestMapping("/info/update")  
 public JsonResult updateInfo(User user) {  
 user.setUserId(getLoginUserId());  
 // 不能修改的字段  
 user.setState(null);  
 user.setPassword(null);  
 user.setUsername(null);  
 user.setOrganizationId(null);  
 if (userService.updateById(user)) {  
 User loginUser = getLoginUser();  
 if (user.getNickName() != null) loginUser.setNickName(user.getNickName());  
 if (user.getAvatar() != null) loginUser.setAvatar(user.getAvatar());  
 return JsonResult.*ok*("保存成功");  
 }  
 return JsonResult.*error*("保存失败");  
 }  
 */\*\*  
 \* excel导入用户  
 \*/* @Transactional  
 @OperLog(value = "用户管理", desc = "excel导入", param = false, result = true)  
 @RequiresPermissions("sys:user:save")  
 @ResponseBody  
 @RequestMapping("/import")  
 public JsonResult importBatch(MultipartFile file) {  
 StringBuilder sb = new StringBuilder();  
 try {  
 // 读取excel  
 int startRow = 1;  
 ExcelReader reader = ExcelUtil.*getReader*(file.getInputStream(), 0);  
 List<List<Object>> list = reader.read(startRow);  
 // 进行非空和重复检查  
 sb.append(CoreUtil.*excelCheckBlank*(list, startRow, 0, 1, 2, 3, 4, 7));  
 sb.append(CoreUtil.*excelCheckRepeat*(list, startRow, 0, 5, 6));  
 if (!sb.toString().isEmpty()) return JsonResult.*error*(sb.toString());  
 // 进行数据库层面检查  
 List<User> users = new ArrayList<>();  
 for (int i = 0; i < list.size(); i++) {  
 List<Object> objects = list.get(i);  
 String username = String.*valueOf*(objects.get(0)); // 账号  
 String password = String.*valueOf*(objects.get(1)); // 密码  
 String nickName = String.*valueOf*(objects.get(2)); // 用户名  
 String sexName = String.*valueOf*(objects.get(3)); // 性别  
 String roleName = String.*valueOf*(objects.get(4)); // 角色名  
 String phone = String.*valueOf*(objects.get(5)); // 手机号  
 String email = String.*valueOf*(objects.get(6)); // 邮箱  
 String orgName = String.*valueOf*(objects.get(7)); // 组织机构  
 if (userService.count(new QueryWrapper<User>().eq("username", username)) > 0) {  
 sb.append("第");  
 sb.append(i + startRow + 1);  
 sb.append("行第1");  
 sb.append("列账号已存在;\r\n");  
 }  
 if (StrUtil.*isNotBlank*(phone) && userService.count(new QueryWrapper<User>().eq("phone", phone)) > 0) {  
 sb.append("第");  
 sb.append(i + startRow + 1);  
 sb.append("行第6");  
 sb.append("列手机号已存在;\r\n");  
 }  
 if (StrUtil.*isNotBlank*(email) && userService.count(new QueryWrapper<User>().eq("email", email)) > 0) {  
 sb.append("第");  
 sb.append(i + startRow + 1);  
 sb.append("行第7");  
 sb.append("列邮箱已存在;\r\n");  
 }  
 User user = new User();  
 user.setUsername(username);  
 user.setNickName(nickName);  
 user.setPassword(userService.encodePsw(password));  
 user.setState(0);  
 user.setPhone(phone);  
 user.setEmail(email);  
 DictionaryData sexDictData = dictionaryDataService.listByDictCodeAndName("sex", sexName);  
 if (sexDictData == null) {  
 sb.append("第");  
 sb.append(i + startRow + 1);  
 sb.append("行第4");  
 sb.append("列性别不存在;\r\n");  
 } else {  
 user.setSex(sexDictData.getDictDataId());  
 }  
 Role role = roleService.getOne(new QueryWrapper<Role>().eq("role\_name", roleName), false);  
 if (role == null) {  
 sb.append("第");  
 sb.append(i + startRow + 1);  
 sb.append("行第5");  
 sb.append("列角色不存在;\r\n");  
 } else {  
 user.setRoleIds(Collections.*singletonList*(role.getRoleId()));  
 }  
 Organization org = organizationService.getOne(new QueryWrapper<Organization>().eq("organization\_full\_name", orgName), false);  
 if (org == null) {  
 sb.append("第");  
 sb.append(i + startRow + 1);  
 sb.append("行第8");  
 sb.append("列角色不存在;\r\n");  
 } else {  
 user.setOrganizationId(org.getOrganizationId());  
 }  
 users.add(user);  
 }  
 if (!sb.toString().isEmpty()) return JsonResult.*error*(sb.toString());  
 // 开始添加用户  
 int okNum = 0, errorNum = 0;  
 for (User user : users) {  
 if (userService.saveUser(user)) okNum++;  
 else errorNum++;  
 }  
 return JsonResult.*ok*("导入完成，成功" + okNum + "条，失败" + errorNum + "条");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return JsonResult.*error*("导入失败");  
 }  
}

在控制层中RoleController.java的描述如下所示：

@Controller  
@RequestMapping("/sys/role")  
public class RoleController extends BaseController {  
 @Autowired  
 private RoleService roleService  
 @RequiresPermissions("sys:role:view")  
 @RequestMapping()  
 public String view() {  
 return "system/role.html";  
 }  
 */\*\*  
 \* 分页查询角色  
 \*/* @OperLog(value = "角色管理", desc = "分页查询")  
 @RequiresPermissions("sys:role:list")  
 @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Role> page(HttpServletRequest request) {  
 PageParam<Role> pageParam = new PageParam<>(request);  
 return new PageResult<>(roleService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 }  
 */\*\*  
 \* 查询全部角色  
 \*/* @OperLog(value = "角色管理", desc = "查询全部")  
 @RequiresPermissions("sys:role:list")  
 @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Role> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(roleService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 根据id查询角色  
 \*/* @OperLog(value = "角色管理", desc = "根据id查询")  
 @RequiresPermissions("sys:role:list")  
 @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(roleService.getById(id));  
 }  
 */\*\*  
 \* 添加角色  
 \*/* @OperLog(value = "角色管理", desc = "添加", param = false, result = true)  
 @RequiresPermissions("sys:role:save")  
 @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Role role) {  
 if (roleService.count(new QueryWrapper<Role>().eq("role\_code", role.getRoleCode())) > 0) {  
 return JsonResult.*error*("角色标识已存在");  
 }  
 if (roleService.count(new QueryWrapper<Role>().eq("role\_name", role.getRoleName())) > 0) {  
 return JsonResult.*error*("角色名称已存在");  
 }  
 if (roleService.save(role)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改角色  
 \*/* @OperLog(value = "角色管理", desc = "修改", param = false, result = true)  
 @RequiresPermissions("sys:role:update")  
 @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Role role) {  
 if (roleService.count(new QueryWrapper<Role>().eq("role\_code", role.getRoleCode())  
 .ne("role\_id", role.getRoleId())) > 0) {  
 return JsonResult.*error*("角色标识已存在");  
 }  
 if (roleService.count(new QueryWrapper<Role>().eq("role\_name", role.getRoleName())  
 .ne("role\_id", role.getRoleId())) > 0) {  
 return JsonResult.*error*("角色名称已存在");  
 }  
 if (roleService.updateById(role)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除角色  
 \*/* @OperLog(value = "角色管理", desc = "删除", result = true)  
 @RequiresPermissions("sys:role:remove")  
 @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (roleService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加角色  
 \*/* @OperLog(value = "角色管理", desc = "批量添加", param = false, result = true)  
 @RequiresPermissions("sys:role:save")  
 @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Role> list) {  
 // 对集合本身进行非空和重复校验  
 StringBuilder sb = new StringBuilder();  
 sb.append(CoreUtil.*listCheckBlank*(list, "roleCode", "角色标识"));  
 sb.append(CoreUtil.*listCheckBlank*(list, "roleName", "角色名称"));  
 sb.append(CoreUtil.*listCheckRepeat*(list, "roleCode", "角色标识"));  
 sb.append(CoreUtil.*listCheckRepeat*(list, "roleName", "角色名称"));  
 if (sb.length() != 0) return JsonResult.*error*(sb.toString());  
 // 数据库层面校验  
 if (roleService.count(new QueryWrapper<Role>().in("role\_code",  
 list.stream().map(Role::getRoleCode).collect(Collectors.*toList*()))) > 0) {  
 return JsonResult.*error*("角色标识已存在");  
 }  
 if (roleService.count(new QueryWrapper<Role>().in("role\_name",  
 list.stream().map(Role::getRoleName).collect(Collectors.*toList*()))) > 0) {  
 return JsonResult.*error*("角色名称已存在");  
 }  
 if (roleService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量删除角色  
 \*/* @OperLog(value = "角色管理", desc = "批量删除", result = true)  
 @RequiresPermissions("sys:role:remove")  
 @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (roleService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
}

### 5.1.3 业务逻辑层

系统管理模块的业务逻辑层主要完成对用户登录逻辑的判定，同时调用系统管理模块的业务逻辑接口。系统管理模块业务逻辑层列表如图4-5所示。

表 5-2系统管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 显示系统管理页面 | UserService.java  UserRoleService.java | 无 | 无 |

在用户管理的业务逻辑层是调用了公用的UserService接口，同时在实现该接口。

UserService.java接口主要方法：

public interface UserService extends IService<User> {  
 */\*\*  
 \* 根据账号查询用户  
 \*/* User getByUsername(String username);  
 */\*\*  
 \* 根据id查询用户(关联查询)  
 \*/* User getFullById(Integer userId);  
 */\*\*  
 \* 查询用户角色和权限  
 \*/* User selectRoleAndAuth(User user);  
 */\*\*  
 \* 关联分页查询用户  
 \*/* PageResult<User> listPage(PageParam<User> page);  
 */\*\*  
 \* 关联查询全部用户  
 \*/* List<User> listAll(Map<String, Object> page);  
 */\*\*  
 \* 添加用户(包含角色)  
 \*/* boolean saveUser(User user);  
 */\*\*  
 \* 修改用户(包含角色)  
 \*/* boolean updateUser(User user);  
 */\*\*  
 \* 比较用户密码  
 \*  
 \** ***@param*** *dbPsw 数据库存储的密码  
 \** ***@param*** *inputPsw 用户输入的密码  
 \** ***@return*** *boolean  
 \*/* boolean comparePsw(String dbPsw, String inputPsw);  
 */\*\*  
 \* md5加密用户密码  
 \*/* String encodePsw(String psw);  
}

UserServiceImpl.java实现了OrderService.java接口

UserServiceImpl.java主要实现属性与方法:

@Service  
public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements UserService {  
 @Autowired  
 private UserRoleMapper userRoleMapper;  
 @Autowired  
 private MenuMapper menuMapper;  
 @Override  
 public User getByUsername(String username) {  
 return baseMapper.selectOne(new QueryWrapper<User>().eq("username", username));  
 }  
 @Override  
 public User getFullById(Integer userId) {  
 List<User> userList = baseMapper.listAll(new PageParam<User>().put("userId", userId).getNoPageParam());  
 if (userList == null || userList.size() == 0) return null;  
 return selectRoleAndAuth(userList.get(0));  
 }  
 @Override  
 public User selectRoleAndAuth(User user) {  
 user.setRoles(userRoleMapper.listByUserId(user.getUserId()));  
 List<Menu> menus = menuMapper.listByUserId(user.getUserId(), null);  
 List<String> auths = new ArrayList<>();  
 for (Menu menu : menus) {  
 auths.add(menu.getAuthority());  
 }  
 user.setAuthorities(auths);  
 return user;  
 }  
 @Override  
 public PageResult<User> listPage(PageParam<User> page) {  
 List<User> users = baseMapper.listPage(page);  
 // 查询用户的角色  
 selectUserRoles(users);  
 return new PageResult<>(users, page.getTotal());  
 }  
 @Override  
 public List<User> listAll(Map<String, Object> page) {  
 List<User> users = baseMapper.listAll(page);  
 // 查询用户的角色  
 selectUserRoles(users);  
 return users;  
 }  
 @Transactional  
 @Override  
 public boolean saveUser(User user) {  
 if (user.getUsername() != null && baseMapper.selectCount(new QueryWrapper<User>()  
 .eq("username", user.getUsername())) > 0) {  
 throw new BusinessException("账号已存在");  
 }  
 if (user.getPhone() != null && baseMapper.selectCount(new QueryWrapper<User>()  
 .eq("phone", user.getPhone())) > 0) {  
 throw new BusinessException("手机号已存在");  
 }  
 if (user.getEmail() != null && baseMapper.selectCount(new QueryWrapper<User>()  
 .eq("email", user.getEmail())) > 0) {  
 throw new BusinessException("邮箱已存在");  
 }  
 boolean result = baseMapper.insert(user) > 0;  
 if (result && user.getRoleIds() != null) {  
 addUserRoles(user.getUserId(), user.getRoleIds(), false);  
 }  
 return result;  
 }  
 @Transactional  
 @Override  
 public boolean updateUser(User user) {  
 if (user.getUsername() != null && baseMapper.selectCount(new QueryWrapper<User>()  
 .eq("username", user.getUsername()).ne("user\_id", user.getUserId())) > 0) {  
 throw new BusinessException("账号已存在");  
 }  
 if (user.getPhone() != null && baseMapper.selectCount(new QueryWrapper<User>()  
 .eq("phone", user.getPhone()).ne("user\_id", user.getUserId())) > 0) {  
 throw new BusinessException("手机号已存在");  
 }  
 if (user.getEmail() != null && baseMapper.selectCount(new QueryWrapper<User>()  
 .eq("email", user.getEmail()).ne("user\_id", user.getUserId())) > 0) {  
 throw new BusinessException("邮箱已存在");  
 }  
 boolean result = baseMapper.updateById(user) > 0;  
 if (result && user.getRoleIds() != null) {  
 addUserRoles(user.getUserId(), user.getRoleIds(), true);  
 }  
 return result;  
 }  
 @Override  
 public boolean comparePsw(String dbPsw, String inputPsw) {  
 return dbPsw != null && dbPsw.equals(encodePsw(inputPsw));  
 }  
 @Override  
 public String encodePsw(String psw) {  
 if (psw == null) return null;  
 return DigestUtils.*md5DigestAsHex*(psw.getBytes());  
 }  
 */\*\*  
 \* 查询用户的角色  
 \*/* private void selectUserRoles(List<User> users) {  
 if (users != null && users.size() > 0) {  
 List<Integer> userIds = new ArrayList<>();  
 for (User one : users) {  
 userIds.add(one.getUserId());  
 }  
 List<Role> userRoles = userRoleMapper.listByUserIds(userIds);  
 for (User user : users) {  
 List<Role> roles = new ArrayList<>();  
 for (Role userRole : userRoles) {  
 if (user.getUserId().equals(userRole.getUserId())) {  
 roles.add(userRole);  
 }  
 }  
 user.setRoles(roles);  
 }  
 }  
 }  
 */\*\*  
 \* 添加用户角色  
 \*/* private void addUserRoles(Integer userId, List<Integer> roleIds, boolean deleteOld) {  
 if (deleteOld) {  
 userRoleMapper.delete(new UpdateWrapper<UserRole>().eq("user\_id", userId));  
 }  
 if (roleIds.size() > 0) {  
 if (userRoleMapper.insertBatch(userId, roleIds) < roleIds.size()) {  
 throw new BusinessException("操作失败");  
 }  
 }  
 }  
}

在角色管理的业务逻辑层是调用了公用的UserRoleService接口，同时在实现该接口。

UserRoleService.java接口主要方法：

public interface UserRoleService extends IService<UserRole> {  
 */\*\*  
 \* 查询用户的角色id  
 \*/* Integer[] getRoleIds(String userId);  
}

UserRoleServiceImpl.java实现了UserRoleService.java接口

UserRoleServiceImpl.java主要实现属性与方法:

@Service  
public class UserRoleServiceImpl extends ServiceImpl<UserRoleMapper, UserRole> implements UserRoleService {  
 @Override  
 public Integer[] getRoleIds(String userId) {  
 List<UserRole> userRoles = baseMapper.selectList(new QueryWrapper<UserRole>().eq("user\_id", userId));  
 Integer[] roleIds = new Integer[userRoles.size()];  
 for (int i = 0; i < userRoles.size(); i++) {  
 roleIds[i] = userRoles.get(i).getRoleId();  
 }  
 return roleIds;  
 }  
}

### 5.1.4 域模型层

系统管理模块用到域模型层中User.java，UserRole.java是一个公用域模型，在涉及到角色/用户管理操作时，就会调用到该模型，系统管理模模块域模型层列表如表5-3所示。

表 5-3系统管理模域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| UserRole.java  User.java | 拉取用户和角色信息信息并且展示 |

UserRole.java主要属性与方法：

@TableName("sys\_user\_role")  
public class UserRole implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* 主键id  
 \*/* @TableId(value = "id", type = IdType.*AUTO*)  
 private Integer id;  
 */\*\*  
 \* 用户id  
 \*/* private Integer userId;  
 */\*\*  
 \* 角色id  
 \*/* private Integer roleId;  
 */\*\*  
 \* 创建时间  
 \*/* private Date createTime;  
 */\*\*  
 \* 修改时间  
 \*/* private Date updateTime;  
 */\*\*  
 \* 角色名称  
 \*/* @TableField(exist = false)  
 private String roleName;  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public Integer getUserId() {  
 return userId;  
 }  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
 public Integer getRoleId() {  
 return roleId;  
 }  
 public void setRoleId(Integer roleId) {  
 this.roleId = roleId;  
 }  
 public Date getCreateTime() {  
 return createTime;  
 }  
 public void setCreateTime(Date createTime) {  
 this.createTime = createTime;  
 }  
 public Date getUpdateTime() {  
 return updateTime;  
 }  
 public void setUpdateTime(Date updateTime) {  
 this.updateTime = updateTime;  
 }  
 public String getRoleName() {  
 return roleName;  
 }  
 public void setRoleName(String roleName) {  
 this.roleName = roleName;  
 }  
 @Override  
 public String toString() {  
 return "UserRole{" +  
 ", id=" + id +  
 ", userId=" + userId +  
 ", roleId=" + roleId +  
 ", createTime=" + createTime +  
 ", updateTime=" + updateTime +  
 ", roleName=" + roleName +  
 "}";  
 }  
}

## 5.2、课程管理模块详细设计

**5.2.1** 课程管理**模块**

### 5.2.2 控制层

课程管理模块的控制层负责接受来自index.html的用户请求，向服务器请求获取课程数据，等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并显示课程管理页面。课程选购模块控制层列表见表4-8所示。

表 5-4 个人信息查询模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 课程管理 | ClassifyController.java  CourseController.java | SUCCESS | 显示课程管理页面以及功能 |
| ERROR | 用户信息拉取失败，显示提示窗口。 |

在控制层中ClassifyController.java的描述如下所示：

@Controller  
@RequestMapping("/course/classify")  
public class ClassifyController extends BaseController {  
 @Autowired  
 private ClassifyService classifyService;  
 @RequestMapping()  
 public String view() {  
 return "course/classify.html";  
 }  
 */\*\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Classify> page(HttpServletRequest request) {  
 PageParam<Classify> pageParam = new PageParam<>(request);  
 return new PageResult<>(classifyService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Classify> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(classifyService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(classifyService.getById(id));  
 }   
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/getById")  
 public String getById(Integer id) {  
 Classify classify = classifyService.getById(id);  
 return classify.getName();  
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Classify classify) {  
 if (classifyService.save(classify)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Classify classify) {  
 if (classifyService.updateById(classify)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (classifyService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Classify> list) {  
 if (classifyService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Classify> batchParam) {  
 if (batchParam.update(classifyService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (classifyService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
}

在控制层中CourseController.java的描述如下所示：

@Controller  
@RequestMapping("/course/course")  
public class CourseController extends BaseController {  
 @Autowired  
 private CourseService courseService;  
 @Autowired  
 private ClassifyService classifyService;  
 @Autowired  
 private TeacherService teacherService;  
 @RequestMapping()  
 public String view(Model model) {  
 model.addAttribute("classifyList", classifyService.list());  
 model.addAttribute("teacherList", teacherService.list());  
 return "course/course.html";  
 }  
 @RequestMapping("/ggkc")  
 public String ggkc() {  
 return "course/ggkc.html";  
 }  
 @RequestMapping("/zykc")  
 public String zykc() {  
 return "course/zykc.html";  
 }  
 @RequestMapping("/mfkc")  
 public String mfkc() {  
 return "course/mfkc.html";  
 }  
 */\*\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Course> page(HttpServletRequest request) {  
 PageParam<Course> pageParam = new PageParam<>(request);  
 return new PageResult<>(courseService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Course> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(courseService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(courseService.getById(id));  
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Course course) {  
 course.setCreateTime(new Date());  
 if (courseService.save(course)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Course course) {  
 if (courseService.updateById(course)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 @ResponseBody  
 @RequestMapping("/updateggkc")  
 public JsonResult updateggkc(Integer id,String type) {  
 Course course = courseService.getById(id);  
 if(type.equals("add")) {  
 course.setGgkc(1);  
 }else {  
 course.setGgkc(0);  
 }  
 if (courseService.updateById(course)) {  
 if(type.equals("add")) {  
 return JsonResult.*ok*("添加成功");  
 }else {  
 return JsonResult.*ok*("移除成功");  
 }  
 }  
 if(type.equals("add")) {  
 return JsonResult.*error*("添加失败");  
 }else {  
 return JsonResult.*error*("移除失败");  
 }   
 }  
 @ResponseBody  
 @RequestMapping("/updatezykc")  
 public JsonResult updatezykc(Integer id,String type) {  
 Course course = courseService.getById(id);  
 if(type.equals("add")) {  
 course.setZykc(1);  
 }else {  
 course.setZykc(0);  
 }  
 if (courseService.updateById(course)) {  
 if(type.equals("add")) {  
 return JsonResult.*ok*("添加成功");  
 }else {  
 return JsonResult.*ok*("移除成功");  
 }  
 }  
 if(type.equals("add")) {  
 return JsonResult.*error*("添加失败");  
 }else {  
 return JsonResult.*error*("移除失败");  
 }  
 }   
 @ResponseBody  
 @RequestMapping("/updatemfkc")  
 public JsonResult updatemfkc(Integer id,String type) {  
 Course course = courseService.getById(id);  
 if(type.equals("add")) {  
 course.setMfkc(1);  
 }else {  
 course.setMfkc(0);  
 }  
 if (courseService.updateById(course)) {  
 if(type.equals("add")) {  
 return JsonResult.*ok*("添加成功");  
 }else {  
 return JsonResult.*ok*("移除成功");  
 }  
 }  
 if(type.equals("add")) {  
 return JsonResult.*error*("添加失败");  
 }else {  
 return JsonResult.*error*("移除失败");  
 }  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (courseService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Course> list) {  
 if (courseService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Course> batchParam) {  
 if (batchParam.update(courseService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (courseService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 @ResponseBody  
 @PostMapping("/edit\_upload")  
 public JsonResult edit\_upload(@RequestParam MultipartFile file) {  
 return FileUploadUtil.*edit\_upload*(file);  
 }  
}

### 5.2.3 业务逻辑层

课程管理模块的业务逻辑层主要完成对用户登录逻辑的判定，同时调用课程管理模块的业务逻辑接口。课程管理模块业务逻辑层列表如图4-5所示。

表 5-5课程管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 显示课程管理页面 | CourseService.java  ClassifyService.java | 无 | 无 |

在用户管理的业务逻辑层是调用了公用的CourseService接口，同时在实现该接口。

CourseService.java接口主要方法：

public interface CourseService extends IService<Course> {  
 */\*\*  
 \* 分页查询  
 \*/* PageResult<Course> listPage(PageParam<Course> page);  
 */\*\*  
 \* 查询所有  
 \*/* List<Course> listAll(Map<String, Object> page);  
 List<Course> listGgkc();  
 List<Course> listZykc();  
 List<Course> listMfkc();  
 List<Course> listXie();  
}

CourseServiceImpl.java实现了CourseService.java接口

CourseServiceImpl.java主要实现属性与方法:

@Service  
public class CourseServiceImpl extends ServiceImpl<CourseMapper, Course> implements CourseService {  
 @Override  
 public PageResult<Course> listPage(PageParam<Course> page) {  
 List<Course> records = baseMapper.listPage(page);  
 return new PageResult<>(records, page.getTotal());  
 }  
 @Override  
 public List<Course> listAll(Map<String, Object> page) {  
 return baseMapper.listAll(page);  
 }  
 @Override  
 public List<Course> listGgkc() {  
 return baseMapper.listGgkc();  
 }  
 @Override  
 public List<Course> listZykc() {  
 return baseMapper.listZykc();  
 }  
 @Override  
 public List<Course> listMfkc() {  
 return baseMapper.listMfkc();  
 }  
 @Override  
 public List<Course> listXie() {  
 return baseMapper.listXie();  
 }  
}

在用户管理的业务逻辑层是调用了公用的ClassifyService接口，同时在实现该接口。

ClassifyService.java接口主要方法：

public interface ClassifyService extends IService<Classify> {  
 */\*\*  
 \* 分页查询  
 \*/* PageResult<Classify> listPage(PageParam<Classify> page);  
 */\*\*  
 \* 查询所有  
 \*/* List<Classify> listAll(Map<String, Object> page);  
}

ClassifyServiceImpl.java实现了ClassifyService.java接口

ClassifyServiceImpl.java主要实现属性与方法:

@Service  
public class ClassifyServiceImpl extends ServiceImpl<ClassifyMapper, Classify> implements ClassifyService {  
 @Override  
 public PageResult<Classify> listPage(PageParam<Classify> page) {  
 List<Classify> records = baseMapper.listPage(page);  
 return new PageResult<>(records, page.getTotal());  
 }  
 @Override  
 public List<Classify> listAll(Map<String, Object> page) {  
 return baseMapper.listAll(page);  
 }  
}

### 5.2.4 域模型层

课程管理模块用到域模型层中Classify.java，Course.java是一个公用域模型，在涉及到课程管理操作时，就会调用到该模型，课程管理模模块域模型层列表如表5-3所示。

表 5-6课程管理模域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Classify.java  Course.java | 拉取课程信息信息并且展示 |

Classify.java主要属性与方法：

@TableName("t\_classify")  
public class Classify implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* 主键  
 \*/* @TableId(value = "id", type = IdType.*AUTO*)  
 private Integer id;  
 */\*\*  
 \* 名称  
 \*/* private String name;  
 */\*\*  
 \* 排序  
 \*/* private Integer sort;  
 */\*\*  
 \* 备注  
 \*/* private String remark;  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public Integer getSort() {  
 return sort;  
 }  
 public void setSort(Integer sort) {  
 this.sort = sort;  
 }  
 public String getRemark() {  
 return remark;  
 }  
 public void setRemark(String remark) {  
 this.remark = remark;  
 }  
 @Override  
 public String toString() {  
 return "Classify{" +  
 ", id=" + id +  
 ", name=" + name +  
 ", sort=" + sort +  
 ", remark=" + remark +  
 "}";  
 }  
}

Course.java主要属性与方法：

@TableName("t\_course")  
public class Course implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* 主键  
 \*/* @TableId(value = "id", type = IdType.*AUTO*)  
 private Integer id;  
 */\*\*  
 \* 名称  
 \*/* private String name;  
 */\*\*  
 \* 缩略图  
 \*/* private String imageUrl;  
 */\*\*  
 \* 类别  
 \*/* private String type;  
 */\*\*  
 \* 内容  
 \*/* private String content;  
 */\*\*  
 \* 教师id  
 \*/* private Integer teacherId;  
 */\*\*  
 \* 创建时间  
 \*/* private Date createTime;  
 */\*\*  
 \* 分类id  
 \*/* private Integer classifyId;  
 private Integer ggkc;  
 private Integer zykc;  
 private Integer mfkc;  
 public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public String getImageUrl() {  
 return imageUrl;  
 }  
 public void setImageUrl(String imageUrl) {  
 this.imageUrl = imageUrl;  
 }  
 public String getType() {  
 return type;  
 }  
 public void setType(String type) {  
 this.type = type;  
 }  
 public String getContent() {  
 return content;  
 }  
 public void setContent(String content) {  
 this.content = content;  
 }  
 public Integer getTeacherId() {  
 return teacherId;  
 }  
 public void setTeacherId(Integer teacherId) {  
 this.teacherId = teacherId;  
 }  
 public Date getCreateTime() {  
 return createTime;  
 }  
 public void setCreateTime(Date createTime) {  
 this.createTime = createTime;  
 }  
 public Integer getClassifyId() {  
 return classifyId;  
 }  
 public void setClassifyId(Integer classifyId) {  
 this.classifyId = classifyId;  
 }  
 public Integer getGgkc() {  
 return ggkc;  
 }  
 public void setGgkc(Integer ggkc) {  
 this.ggkc = ggkc;  
 }  
 public Integer getZykc() {  
 return zykc;  
 }  
 public void setZykc(Integer zykc) {  
 this.zykc = zykc;  
 }  
 public Integer getMfkc() {  
 return mfkc;  
 }  
 public void setMfkc(Integer mfkc) {  
 this.mfkc = mfkc;  
 }  
 @Override  
 public String toString() {  
 return "Course{" +  
 ", id=" + id +  
 ", name=" + name +  
 ", imageUrl=" + imageUrl +  
 ", type=" + type +  
 ", content=" + content +  
 ", teacherId=" + teacherId +  
 ", createTime=" + createTime +  
 ", classifyId=" + classifyId +  
 ", ggkc=" + ggkc +  
 ", zykc=" + zykc +  
 ", mfkc=" + mfkc +  
 "}";  
 }  
}

## 5.3、教师管理模块详细设计

**5.2.1 教师管理模块**

### 5.3.2 控制层

教师管理模块的控制层负责接受来自index.html的用户请求，向服务器请求获取教师数据，等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并显示教师管理页面。课程选购模块控制层列表见表4-8所示。

表 5-7 教师管理模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 课程选择 | TeacherController.java | SUCCESS | 显示教师管理页面 |
| ERROR | 用户信息拉取失败，显示提示窗口。 |

在控制层中TeacherController.java的描述如下所示：

@Controller  
@RequestMapping("/course/teacher")  
public class TeacherController extends BaseController {  
 @Autowired  
 private TeacherService teacherService;  
 @Autowired  
 private UserRoleService userRoleService;  
 @Autowired  
 private UserService userService;  
 @RequestMapping()  
 public String view() {  
 return "course/teacher.html";  
 }  
 */\*\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Teacher> page(HttpServletRequest request) {  
 PageParam<Teacher> pageParam = new PageParam<>(request);  
 return new PageResult<>(teacherService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Teacher> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(teacherService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/listuser")  
 public JsonResult listuser() {  
 List<UserRole> userRoles = userRoleService.list(new QueryWrapper<UserRole>().eq("role\_id", 2));  
 String ids = "";  
 for(UserRole userrole : userRoles) {  
 ids += userrole.getUserId()+",";  
 }  
 ids = ids.substring(0,ids.length() - 1);  
 List<User> users = userService.list(new QueryWrapper<User>().in("user\_id", ids).eq("state", 0).eq("deleted", 0));  
 return JsonResult.*ok*().setData(users);  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(teacherService.getById(id));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/getById")  
 public String getById(Integer id) {  
 Teacher teacher = teacherService.getById(id);  
 return teacher.getNickName();  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/userId")  
 public String userId(Integer userId) {  
 User user = userService.getById(userId);  
 return user.getUsername();  
 }  
 @ResponseBody  
 @RequestMapping("/updateByuserId")  
 public JsonResult updateByuserId(Integer id,Integer userId) {  
 Teacher teacher = teacherService.getById(id);  
 teacher.setUserId(userId);  
 if (teacherService.updateById(teacher)) {  
 return JsonResult.*ok*("授权成功");  
 }  
 return JsonResult.*error*("授权失败");   
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Teacher teacher) {  
 if (teacherService.save(teacher)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Teacher teacher) {  
 if (teacherService.updateById(teacher)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (teacherService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Teacher> list) {  
 if (teacherService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Teacher> batchParam) {  
 if (batchParam.update(teacherService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (teacherService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 @ResponseBody  
 @PostMapping("/upload")  
 public JsonResult upload(@RequestParam MultipartFile file) {  
 return FileUploadUtil.*upload*(file);  
 }  
}

### 5.3.3 业务逻辑层

教师管理模块的业务逻辑层主要完成对用户登录逻辑的判定，同时调用教师管理模块的业务逻辑接口。教师管理模块业务逻辑层列表如图4-5所示。

表 5-8系统管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 显示教师管理页面 | TeacherService.java | 无 | 无 |

在教师管理的业务逻辑层是调用了公用的TeacherService接口，同时在实现该接口。

TeacherService.java接口主要方法：

public interface TeacherService extends IService<Teacher> {  
 */\*\*  
 \* 分页查询  
 \*/* PageResult<Teacher> listPage(PageParam<Teacher> page);  
 */\*\*  
 \* 查询所有  
 \*/* List<Teacher> listAll(Map<String, Object> page);  
}

TeacherServiceImpl.java实现了TeacherService.java接口

TeacherServiceImpl.java主要实现属性与方法:

@Service  
public class TeacherServiceImpl extends ServiceImpl<TeacherMapper, Teacher> implements TeacherService {  
 @Override  
 public PageResult<Teacher> listPage(PageParam<Teacher> page) {  
 List<Teacher> records = baseMapper.listPage(page);  
 return new PageResult<>(records, page.getTotal());  
 }  
 @Override  
 public List<Teacher> listAll(Map<String, Object> page) {  
 return baseMapper.listAll(page);  
 }  
}

### 5.3.4 域模型层

教师管理模块用到域模型层中Teacher.java是一个公用域模型，在涉及到教师管理操作时，就会调用到该模型，教师管理模模块域模型层列表如表5-3所示。

表 5-12教师管理模域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Teacher.java | 拉取教师信息信息并且展示 |

Teacher.java主要属性与方法：

@TableName("t\_teacher")  
public class Teacher implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* 主键  
 \*/* @TableId(value = "id", type = IdType.*AUTO*)  
 private Integer id;  
 */\*\*  
 \* 真实姓名  
 \*/* @TableField("nickName")  
 private String nickName;  
 */\*\*  
 \* 备注  
 \*/* private String remark;  
 */\*\*  
 \* 图片  
 \*/* @TableField("imageName")  
 private String imageName;  
 private Integer userId  
 public Integer getId() {  
 return id;  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getNickName() {  
 return nickName;  
 }  
 public void setNickName(String nickName) {  
 this.nickName = nickName;  
 }  
 public String getRemark() {  
 return remark;  
 }  
 public void setRemark(String remark) {  
 this.remark = remark;  
 }  
 public String getImageName() {  
 return imageName;  
 }  
 public void setImageName(String imageName) {  
 this.imageName = imageName;  
 }  
 public Integer getUserId() {  
 return userId;  
 }  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
 @Override  
 public String toString() {  
 return "Teacher{" +  
 ", id=" + id +  
 ", nickName=" + nickName +  
 ", remark=" + remark +  
 ", imageName=" + imageName +  
 ", userId=" + userId +  
 "}";  
 }  
}

## 5.4、新闻管理模块详细设计

**5.2.1 新闻管理模块**

### 5.4.2 控制层

新闻管理模块的控制层负责接受来自index.html的用户请求，向服务器请求获取新闻数据，等到业务逻辑处理完成之后，将来自业务逻辑层的相应信息传到表现层，并显示新闻管理页面。新闻管理模块控制层列表见表5-10所示。

表 5-10 新闻管理查询模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 新闻管理 | ReleaseController.java | SUCCESS | 显示新闻管理页面 |
| ERROR | 新闻信息拉取失败，显示提示窗口。 |

在控制层中ReleaseController.java的描述如下所示：

@Controller  
@RequestMapping("/course/release")  
public class ReleaseController extends BaseController {  
 @Autowired  
 private ReleaseService releaseService;  
 @Autowired  
 private UserTwoService userTwoService;  
 @RequestMapping()  
 public String view() {  
 return "course/release.html";  
 }  
 */\*\*  
 \* 分页查询  
 \*/* @ResponseBody  
 @RequestMapping("/page")  
 public PageResult<Release> page(HttpServletRequest request) {  
 PageParam<Release> pageParam = new PageParam<>(request);  
 return new PageResult<>(releaseService.page(pageParam, pageParam.getWrapper()).getRecords(), pageParam.getTotal());  
 }

*/\*\*  
 \* 查询全部  
 \*/* @ResponseBody  
 @RequestMapping("/list")  
 public JsonResult list(HttpServletRequest request) {  
 PageParam<Release> pageParam = new PageParam<>(request);  
 return JsonResult.*ok*().setData(releaseService.list(pageParam.getOrderWrapper()));  
 }  
 */\*\*  
 \* 根据id查询  
 \*/* @ResponseBody  
 @RequestMapping("/get")  
 public JsonResult get(Integer id) {  
 return JsonResult.*ok*().setData(releaseService.getById(id));  
 }  
 @ResponseBody  
 @RequestMapping("/shenhe")  
 public JsonResult ShenHe(Integer id,Integer state,Integer present,Integer userId) {  
 Release release = releaseService.getById(id);  
 UserTwo usertwo = userTwoService.getById(userId);  
 release.setState(state);  
 if (releaseService.updateById(release)) {  
 int fenzhi = usertwo.getPresent() + present;  
 usertwo.setPresent(fenzhi);  
 if(userTwoService.updateById(usertwo)) {  
 return JsonResult.*ok*("添加成功");  
 }   
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 添加  
 \*/* @ResponseBody  
 @RequestMapping("/save")  
 public JsonResult save(Release release) {  
 if (releaseService.save(release)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 修改  
 \*/* @ResponseBody  
 @RequestMapping("/update")  
 public JsonResult update(Release release) {  
 if (releaseService.updateById(release)) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 删除  
 \*/* @ResponseBody  
 @RequestMapping("/remove")  
 public JsonResult remove(Integer id) {  
 if (releaseService.removeById(id)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
 */\*\*  
 \* 批量添加  
 \*/* @ResponseBody  
 @RequestMapping("/saveBatch")  
 public JsonResult saveBatch(@RequestBody List<Release> list) {  
 if (releaseService.saveBatch(list)) {  
 return JsonResult.*ok*("添加成功");  
 }  
 return JsonResult.*error*("添加失败");  
 }  
 */\*\*  
 \* 批量修改  
 \*/* @ResponseBody  
 @RequestMapping("/updateBatch")  
 public JsonResult updateBatch(@RequestBody BatchParam<Release> batchParam) {  
 if (batchParam.update(releaseService, "id")) {  
 return JsonResult.*ok*("修改成功");  
 }  
 return JsonResult.*error*("修改失败");  
 }  
 */\*\*  
 \* 批量删除  
 \*/* @ResponseBody  
 @RequestMapping("/removeBatch")  
 public JsonResult removeBatch(@RequestBody List<Integer> ids) {  
 if (releaseService.removeByIds(ids)) {  
 return JsonResult.*ok*("删除成功");  
 }  
 return JsonResult.*error*("删除失败");  
 }  
}

### 5.4.3 业务逻辑层

新闻管理模块的业务逻辑层主要完成对用户登录逻辑的判定，同时调用新闻管理模块的业务逻辑接口。登录模块业务逻辑层列表如图4-5所示。

表 5-11新闻管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 显示新闻管理页面 | ReleaseService.java | 无 | 无 |

在新闻管理的业务逻辑层是调用了公用的ReleaseService接口，同时在实现该接口。

ReleaseService.java接口主要方法：

ReleaseServiceImpl.java实现了ReleaseService.java接口

ReleaseServiceImpl.java主要实现属性与方法:

@Service  
public class ReleaseServiceImpl extends ServiceImpl<ReleaseMapper, Release> implements ReleaseService {  
 @Override  
 public PageResult<Release> listPage(PageParam<Release> page) {  
 List<Release> records = baseMapper.listPage(page);  
 return new PageResult<>(records, page.getTotal());  
 }  
 @Override  
 public List<Release> listAll(Map<String, Object> page) {  
 return baseMapper.listAll(page);  
 }  
}

### 5.4.4 域模型层

新闻管理模块用到域模型层中Release.java是一个公用域模型，在涉及到教师管理操作时，就会调用到该模型，教师管理模模块域模型层列表如表5-3所示。

表 5-12教师管理模域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Release.java | 拉取新闻信息并且展示 |

Release.java主要属性与方法：

@TableName("t\_teacher")  
public class Teacher implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
 */\*\*  
 \* 主键  
 \*/* @TableId(value = "id", type = IdType.*AUTO*)  
 private Integer id;  
 */\*\*  
 \* 真实姓名  
 \*/* @TableField("nickName")  
 private String nickName;  
 */\*\*  
 \* 备注  
 \*/* private String remark;  
 */\*\*  
 \* 图片  
 \*/* @TableField("imageName")  
 private String imageName;  
 private Integer userId  
 public Integer getId() {  
 return id;  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getNickName() {  
 return nickName;  
 }  
 public void setNickName(String nickName) {  
 this.nickName = nickName;  
 }  
 public String getRemark() {  
 return remark;  
 }  
 public void setRemark(String remark) {  
 this.remark = remark;  
 }  
 public String getImageName() {  
 return imageName;  
 }  
 public void setImageName(String imageName) {  
 this.imageName = imageName;  
 }  
 public Integer getUserId() {  
 return userId;  
 }  
 public void setUserId(Integer userId) {  
 this.userId = userId;  
 }  
 @Override  
 public String toString() {  
 return "Teacher{" +  
 ", id=" + id +  
 ", nickName=" + nickName +  
 ", remark=" + remark +  
 ", imageName=" + imageName +  
 ", userId=" + userId +  
 "}";  
 }  
}

# 6、公共部分模块详细设计

## 安全模块的详细设计

安全模块使用了Apache Shiro安全框架,执行身份验证、授权、密码和会话管理。使用Shiro可以快速、轻松地获得任何应用程序,从最小的移动应用程序到最大的网络和企业应用程序。

