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

**XXX详细设计**

**V1.0**

**广西民族大学**

评审日期： 2016年2月28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)

[4.1.3 数据持久层 9](#_Toc444248428)

[4.2、登录模块 12](#_Toc444248429)

[4.2.1 表现层 12](#_Toc444248430)

[4.2.2 控制层 14](#_Toc444248431)

[4.2.3 业务逻辑层 15](#_Toc444248432)

[4.2.4 数据持久层 17](#_Toc444248433)

[4.2.5 域模型层 19](#_Toc444248434)

[4.3、XXXX模块 20](#_Toc444248435)

[4.3.1 表现层 20](#_Toc444248436)

[4.3.2 控制层 21](#_Toc444248437)

[4.3.3 业务逻辑层 21](#_Toc444248438)

[4.3.4 数据持久层 21](#_Toc444248439)

[4.3.5 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 数据持久层 22](#_Toc444248446)

[5.1.5 域模型层 22](#_Toc444248447)

[5.n、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 数据持久层 23](#_Toc444248452)

[5.n.5 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

该文档是对作业管理系统概要设计进行的详细说明，是为了指导和规范作业管理系统开发而制定的详细开发设计文档,作业是一个强有力的手段.对于计算机类课程的作业提交及管理相对于传统课目而言,有它的特殊性.传统的作业提交方式应用于计算机类课程作业的提交有着很多的弊病,因而建立一套方便,快捷,适用,安全可靠,基与网络的作业管理系统就显得十分的必要。

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

该文档内容涵盖作业管理系统的所有功能模块的详细设计描述，包括以下几个模块：

1.题库管理

2.作业布置

3.作业提交

4.作业批改

5.分数统计

6.作业情况查阅

7.作业范例学习和点评

## 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。

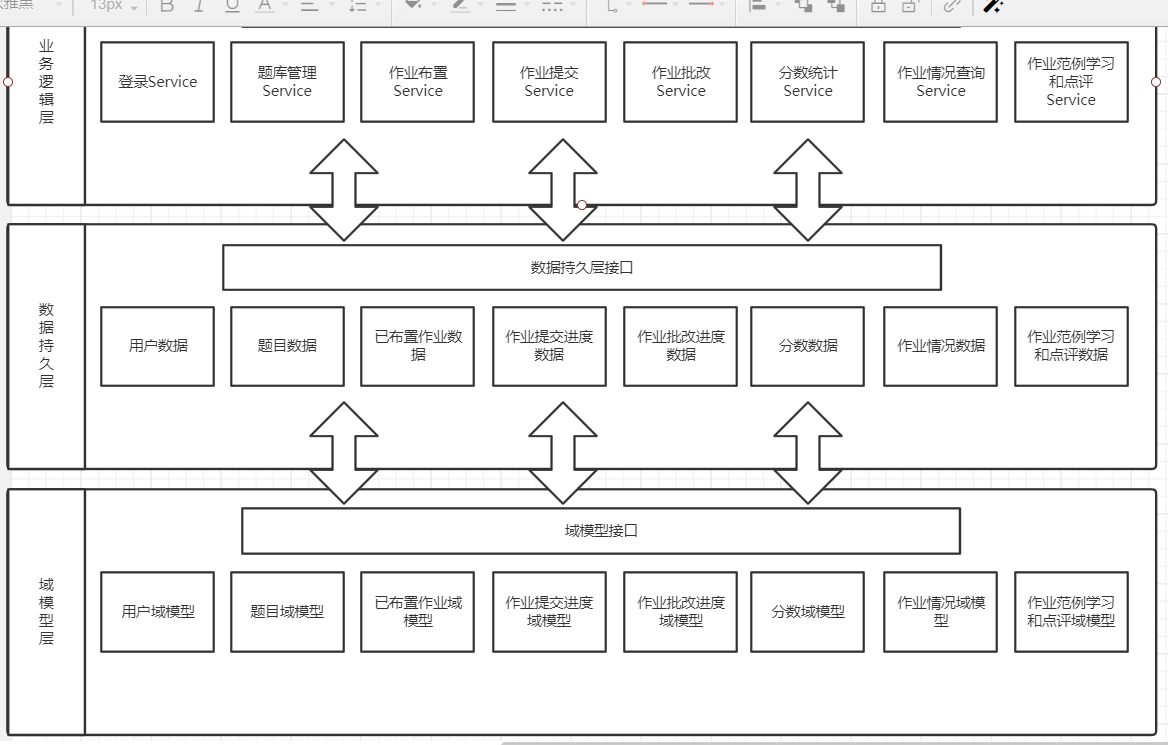
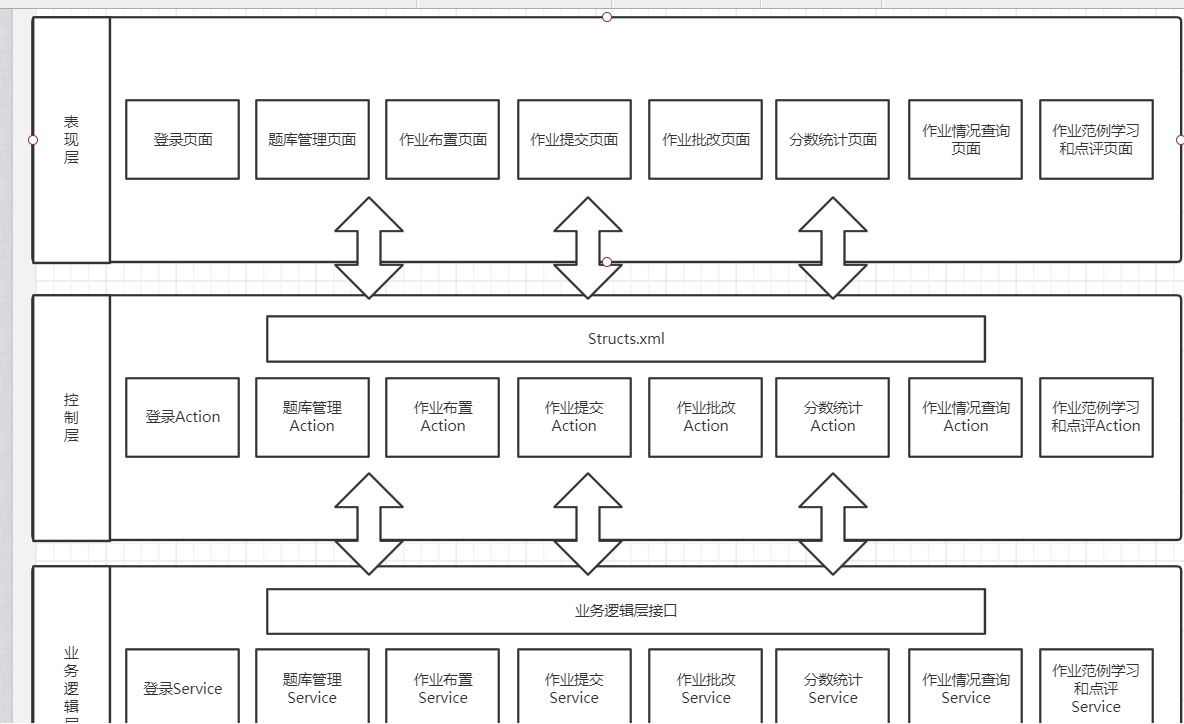
表1-1 文档更新记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修改编号 | 修改日期 | 修改后版本 | 修改位置 | 修改内容概述 |
| 001 | 2021.6.15 | 0.1 | 全部 | 初始发布版本 |
| 002 | 2021.6.16 | 1.0 | 4章节 | 增加 |
| 003 | 2021.6.17 | 2.0 | 4章节 | 增加 |
| 004 | 2021.6.18 | 3.0 | 4、5章节 | 增加 |
| 005 | 2021.6.21 | 4.0 | 5章节 | 修改、增加 |
| 006 | 2021.6.22 | 5.0 | 5章节 | 修改、增加 |
| 007 | 2021.6.23 | 6.0 | 6章节 | 增加 |

# 2．详细设计简述

## 2.1 设计简介

作业管理系统分层结构图如图2-1所示。

图 2-1 作业管理系统示意图

## 2.2 模块简介

该系统主要分为题库管理、作业布置、作业提交、作业批改、分数统计、作业情况查阅、作业范例学习和点评这几个模块。题库管理是用于存储学生使用的题库，由教师进行添加，修改等操作。作业布置是教师用来发布作业给学生完成的模块，其中可以调整发布时间。作业提交是学生用来提交已完成的作业的模块，学生需要在教师规定时间内完成并提交作业，否则记为未完成，作业批改是教师用来给学生提交作业评分的模块，教师可以使用该模块进行评分，也可以在评分后修改评分。分数统计是教师用来对学生进行相关评价的模块，在教师提交学生分数后，系统会自动进行相关计算和统计，然后把统计后的分数情况呈现给教师进行判断。作业情况查询是教师用来检查学生完成作业情况的模块，在结束提交作业后，系统会自动记录已提交作业的学生和未提交作业的学生，然后在此模块展示出来给教师评阅。作业范例学习和评阅是教师用来提供给学生学习范例与查看个人评阅的模块，教师可以选择批改后的优秀作业进行展示，并且可以对所有作业进行评阅，学生在教师评阅完成后可以查看自身个人评阅。

# 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

页面采用vue框架 ，是一套构建用户界面的渐进式框架。Vue 只关注视图层， 采用自底向上增量开发的设计。Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件

## 3.2、静态页面详细设计

作业管理系统静态页面见表3-1

表3-1 页面表现层设计

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 题库管理页面 | exam/form.vue | 用于存储学生使用的题库，由教师进行添加，修改等操作 |
| 作业布置页面 | exam/index.vue | 教师用来发布作业给学生完成的模块，其中可以调整发布时间 |
| 作业提交页面 | paper/exam/list.vue | 学生用来提交已完成的作业的模块，学生需要在教师规定时间内完成并提交作业，否则记为未完成 |
| 作业批改页面 | paper/paper/index.vue | 教师用来给学生提交作业评分的模块，教师可以使用该模块进行评分，也可以在评分后修改评分 |
| 分数统计页面 | paper/exam/result.vue | 教师用来对学生进行相关评价的模块，在教师提交学生分数后，系统会自动进行相关计算和统计，然后把统计后的分数情况呈现给教师进行判断 |
| 作业情况查询页面 | user/exam/paper.vue | 教师用来检查学生完成作业情况的模块，在结束提交作业后，系统会自动记录已提交作业的学生和未提交作业的学生，然后在此模块展示出来给教师评阅 |
| 作业范例学习和点评页面 | user/exam/my.vue | 教师用来提供给学生学习范例与查看个人评阅的模块，教师可以选择批改后的优秀作业进行展示，并且可以对所有作业进行评阅，学生在教师评阅完成后可以查看自身个人评阅 |

# 4、客户端模块详细设计

客户端模块主要包括登录、主页、个人信息等模块，登录模块可以分学生登录和教师登录，拥有不同的界面，主页主要有题库管理、作业布置、作业提交、作业批改、分数统计、作业情况查询、作业范例学习和点评等功能组成的链表和退出登录按钮和个人信息按钮构成，个人信息模块主要可以查看和修改姓名，班级，年龄等个人基本信息。

## 4.1、公用模块的详细设计

### 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4-1。

表 4-1 公用模块表现层

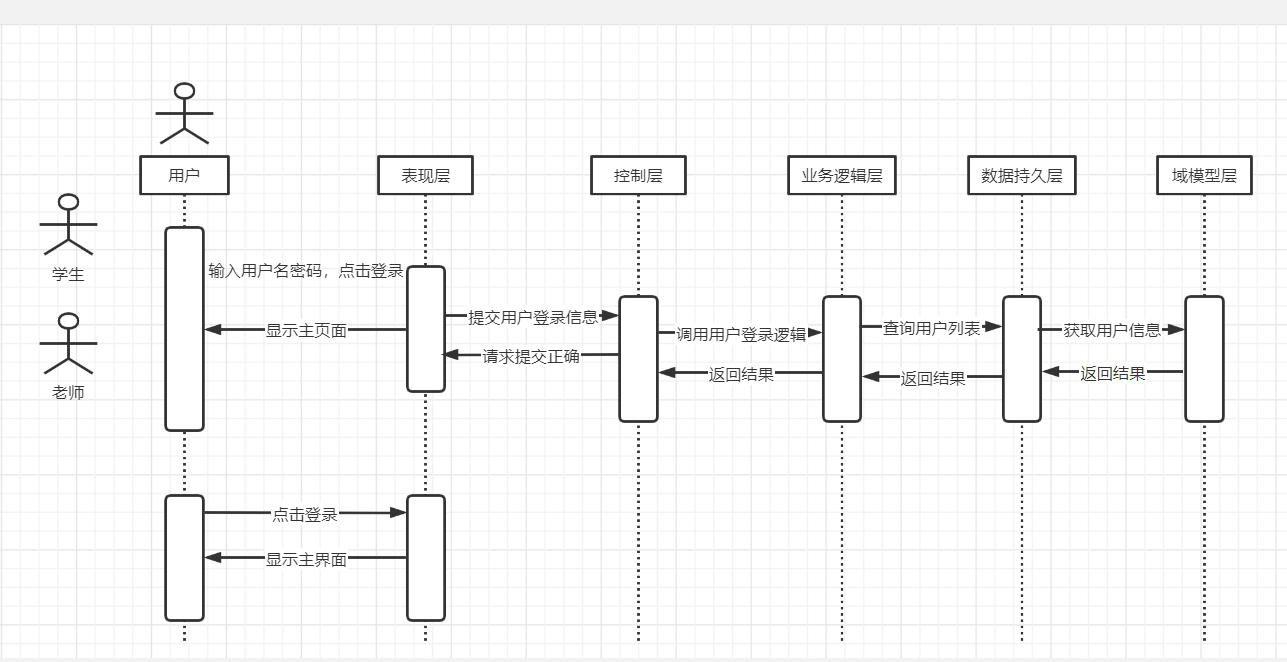
|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 网页错误页面 | error-page/404.vue | 输入网页时出现严重错误时跳转到该页面。 |
| 权限错误页面 | error-page/401.vue | 在进入其他模块时权限不足出时跳转到该页面。 |

### 4.1.2 控制层

public class BaseController {  
  
 private static final Integer CODE\_SUCCESS = 0;  
 private static final String MSG\_SUCCESS = "操作成功！";  
  
 private static final Integer CODE\_FAILURE = 1;  
 private static final String MSG\_FAILURE = "请求失败！";  
  
 protected <T> ApiRest<T> message(Integer code, String message, T data){  
 ApiRest<T> response = new ApiRest<>();  
 response.setCode(code);  
 response.setMsg(message);  
 if(data!=null) {  
 response.setData(data);  
 }  
 return response;  
 }  
  
 protected <T> ApiRest<T> success(){  
 return message(0, "请求成功！", null);  
 }  
  
 protected <T> ApiRest<T> success(String message, T data){  
 return message(CODE\_SUCCESS, message, data);  
 }  
  
 protected <T> ApiRest<T> success(T data){  
 return message(CODE\_SUCCESS, MSG\_SUCCESS, data);  
 }  
  
 protected <T> ApiRest<T> failure(Integer code, String message, T data){  
 return message(code, message, data);  
 }  
  
 protected <T> ApiRest<T> failure(String message, T data){  
 return message(CODE\_FAILURE, message, data);  
 }  
  
 protected <T> ApiRest<T> failure(String message){  
 return message(CODE\_FAILURE, message, null);  
 }  
  
 protected <T> ApiRest<T> failure(T data){  
 return message(CODE\_FAILURE, MSG\_FAILURE, data);  
 }  
  
 protected <T> ApiRest<T> failure(){  
 return message(CODE\_FAILURE, MSG\_FAILURE, null);  
 }  
  
 protected <T> ApiRest<T> failure(ApiError error, T data){  
 return message(error.getCode(), error.msg, data);  
 }  
  
 protected <T> ApiRest<T> failure(ServiceException ex){  
 ApiRest<T> apiRest = message(ex.getCode(), ex.getMsg(), null);  
 return apiRest;  
 }  
}

## 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VUE页面列表见表4-2所示。

表 4-2 登入模块表现层VUE列表

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

Register.vue的流程图如图4-3所示。

进入游首页

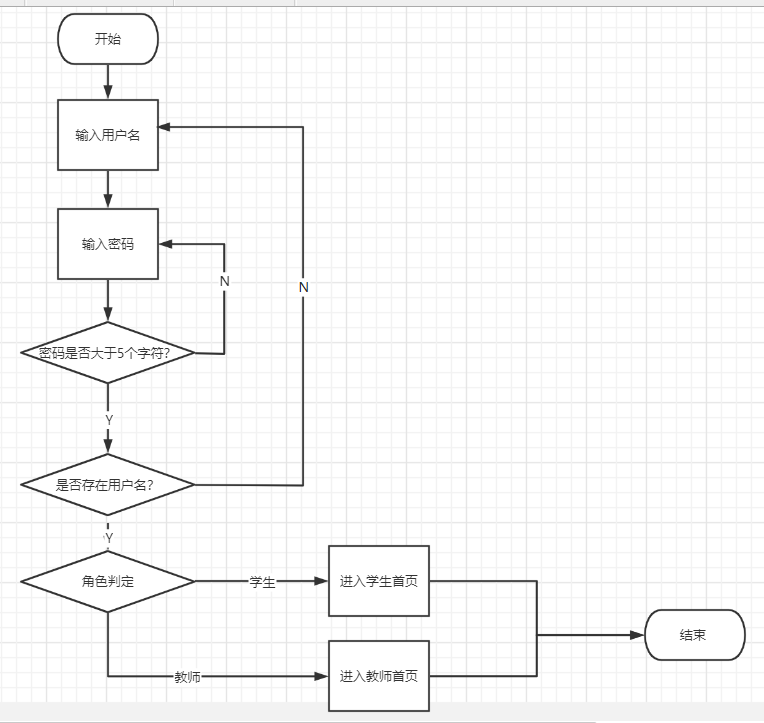


图 4-3 登录模块详细设计流程图

### 4.2.2 控制层

登录模块的控制层负责接受来自login包中的index.vue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 | ShiroRealm.java | SUCCESS | dashboard/index.jsp—登入成功，显示提示窗口。 |
| ERROR | login/index.vue---登入失败，显示提示窗口。 |

在控制层中ShiroRealm.java的描述如下所示：

public class ShiroRealm extends AuthorizingRealm {  
  
 @Autowired  
 @Lazy  
 private SysUserService sysUserService;  
  
 @Autowired  
 @Lazy  
 private SysUserRoleService sysUserRoleService;  
  
  
 @Override  
 public boolean supports(AuthenticationToken token) {  
 return token instanceof JwtToken;  
 }  
  
 @Override  
 protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  
  
 String userId = null;  
 if (principals != null) {  
 SysUserLoginDTO user = (SysUserLoginDTO) principals.getPrimaryPrincipal();  
 userId = user.getId();  
 }  
 SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();  
  
 List<String> roles = sysUserRoleService.listRoles(userId);  
 info.setRoles(new HashSet<>(roles));  
  
 log.info("++++++++++校验详细权限完成");  
 return info;  
 }  
  
 @Override  
 protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken auth) throws AuthenticationException {  
 String token = (String) auth.getCredentials();  
 if (token == null) {  
 throw new AuthenticationException("token为空!");  
 }  
  
 log.error("++++++++++校验用户："+token);  
  
 SysUserLoginDTO user = this.checkToken(token);  
 return new SimpleAuthenticationInfo(user, token, getName());  
 }  
  
 public SysUserLoginDTO checkToken(String token) throws AuthenticationException {  
  
 log.debug("++++++++++校验用户token： "+ token);  
  
 String username = JwtUtils.getUsername(token);  
 log.debug("++++++++++用户名： "+ username);  
  
 if (username == null) {  
 throw new AuthenticationException("无效的token");  
 }  
 SysUserLoginDTO user = sysUserService.token(token);  
  
 if (!JwtUtils.verify(token, username)) {  
 throw new AuthenticationException("登陆失效，请重试登陆!");  
 }  
  
 return user;  
 }  
  
 @Override  
 public void clearCache(PrincipalCollection principals) {  
 super.clearCache(principals);  
 }  
  
}

## 4.3、题库管理模块

题库管理模块的操作示意图如图4-3所示。

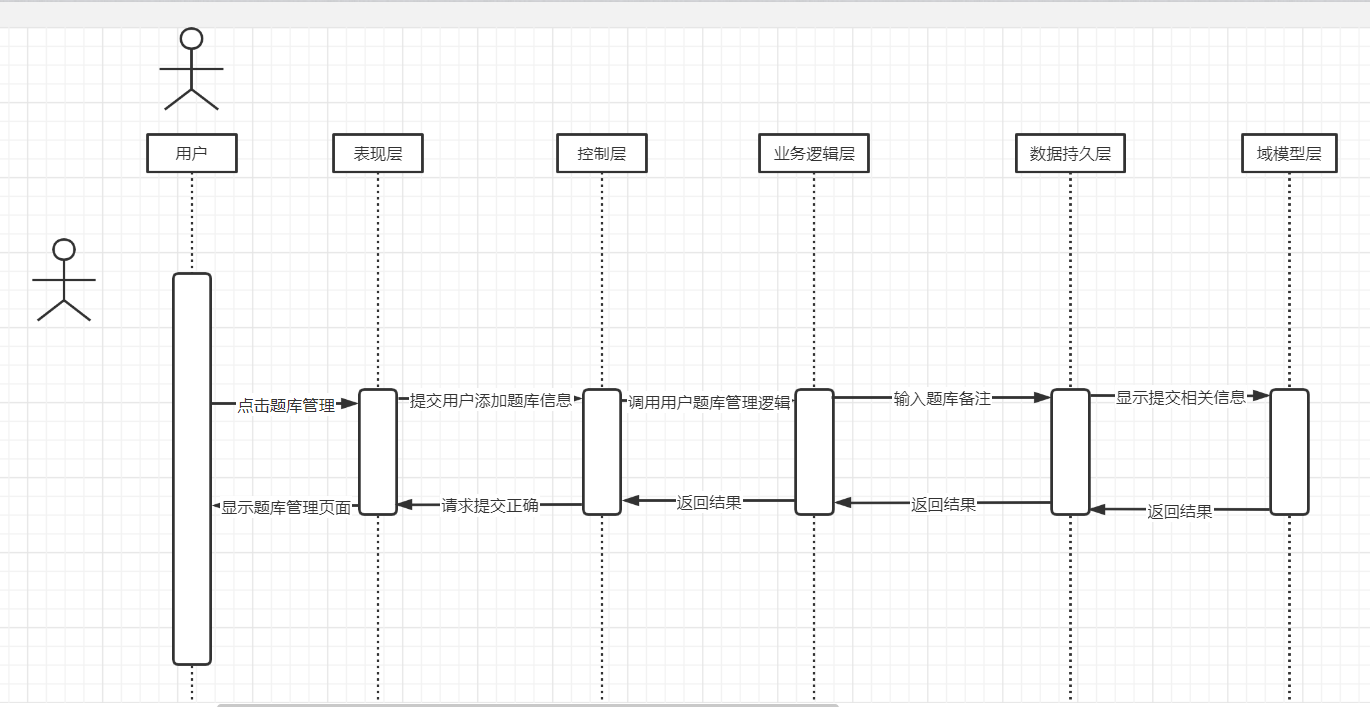


图 4-3 题库管理操作示意图

### 4.3.1 表现层

题库管理模块的表现层主要完成不同用户的查看功能。表现层对应的VUE页面列表见表4-3所示。

表 4-3 题库管理模块表现层VUE列表

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 题库管理页面 | Qu/index.vue | 用户查看题库功能 |

### 4.3.2 控制层

登录模块的控制层负责接受来自qu包中的index.vue的用户输入，同时调用题库模块的业务逻辑接口，将用户查看的相关题目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。题库管理模块控制层列表见表4-4所示。

表 4-4 题库管理模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 登录 | QuController.java | SUCCESS | qu/index.vue |

在控制层中QuController.java的描述如下所示：

public class QuController extends BaseController {  
  
 @Autowired  
 private QuService baseService;  
  
 @ApiOperation(value = "添加或修改")  
 @RequestMapping(value = "/save", method = {RequestMethod.POST})  
 public ApiRest<BaseIdRespDTO> save(@RequestBody QuDetailDTO reqDTO) {  
 baseService.save(reqDTO);  
 return super.success();  
 }  
  
 @ApiOperation(value = "批量删除")  
 @RequestMapping(value = "/delete", method = {RequestMethod.POST})  
 public ApiRest edit(@RequestBody BaseIdsReqDTO reqDTO) {  
  
 baseService.removeByIds(reqDTO.getIds());  
 return super.success();  
 }  
  
 @ApiOperation(value = "查找详情")  
 @RequestMapping(value = "/detail", method = {RequestMethod.POST})  
 public ApiRest<QuDetailDTO> detail(@RequestBody BaseIdReqDTO reqDTO) {  
 QuDetailDTO dto = baseService.detail(reqDTO.getId());  
 return super.success(dto);  
 }  
  
 @ApiOperation(value = "分页查找")  
 @RequestMapping(value = "/paging", method = {RequestMethod.POST})  
 public ApiRest<IPage<QuDTO>> paging(@RequestBody PagingReqDTO<QuQueryReqDTO> reqDTO) {  
  
 IPage<QuDTO> page = baseService.paging(reqDTO);  
  
 return super.success(page);  
 }  
  
 @ApiOperation(value = "查找列表")  
 @RequestMapping(value = "/list", method = {RequestMethod.POST})  
 public ApiRest<List<QuDTO>> list(@RequestBody QuDTO reqDTO) {  
  
 QueryWrapper<Qu> wrapper = new QueryWrapper<>();  
  
 List<Qu> list = baseService.list(wrapper);  
  
 List<QuDTO> dtoList = BeanMapper.mapList(list, QuDTO.class);  
  
 return super.success(dtoList);  
 }  
  
 @ResponseBody  
 @RequestMapping(value = "/export")  
 public ApiRest exportFile(HttpServletResponse response, @RequestBody QuQueryReqDTO reqDTO) {  
  
 String fileName = "导出的试题-" + System.currentTimeMillis() + ".xlsx";  
  
 try {  
  
 int no = 0;  
 String quId = "";  
 List<QuExportDTO> list = baseService.listForExport(reqDTO);  
 for (QuExportDTO item : list) {  
 if (!quId.equals(item.getQId())) {  
 quId = item.getQId();  
 no += 1;  
 } else {  
 item.setQuType("0");  
 item.setQContent("");  
 item.setQAnalysis("");  
 item.setRepoList(null);  
 item.setQImage("");  
 item.setQVideo("");  
 }  
 item.setNo(String.valueOf(no));  
 }  
 new ExportExcel("试题", QuExportDTO.class).setDataList(list).write(response, fileName).dispose();  
 return super.success();  
 } catch (Exception e) {  
 return failure(e.getMessage());  
 }  
 }  
  
 @ResponseBody  
 @RequestMapping(value = "import")  
 public ApiRest importFile(@RequestParam("file") MultipartFile file) {  
  
 try {  
  
 ImportExcel ei = new ImportExcel(file, 1, 0);  
 List<QuExportDTO> list = ei.getDataList(QuExportDTO.class);  
  
 this.checkExcel(list);  
  
 baseService.importExcel(list);  
  
 return super.success();  
  
 } catch (IOException e) {  
  
 } catch (InvalidFormatException e) {  
  
 } catch (IllegalAccessException e) {  
  
 } catch (InstantiationException e) {  
  
 }  
  
 return super.failure();  
 }  
  
 private void checkExcel(List<QuExportDTO> list) throws ServiceException {  
  
 int line = 3;  
 StringBuffer sb = new StringBuffer();  
  
 if (CollectionUtils.isEmpty(list)) {  
 throw new ServiceException(1, "您导入的数据似乎是一个空表格！");  
 }  
  
 Integer quNo = null;  
 for (QuExportDTO item : list) {  
  
 System.out.println(item.getNo());  
 if (StringUtils.isBlank(item.getNo())) {  
 line++;  
 continue;  
 }  
  
 System.out.println(item.getQContent());  
 Integer no;  
  
 try {  
 no = Integer.parseInt(item.getNo());  
 } catch (Exception e) {  
 line++;  
 continue;  
 }  
  
 if (no == null) {  
 sb.append("第" + line + "行，题目序号不能为空！<br>");  
 }  
  
 if (quNo == null || !quNo.equals(no)) {  
  
 if (item.getQuType() == null) {  
 sb.append("第" + line + "行，题目类型不能为空<br>");  
 }  
  
 if (StringUtils.isBlank(item.getQContent())) {  
 sb.append("第" + line + "行，题目内容不能为空<br>");  
 }  
  
 if (CollectionUtils.isEmpty(item.getRepoList())) {  
 sb.append("第" + line + "行，题目必须包含一个题库<br>");  
 }  
 }  
  
 if (StringUtils.isBlank(item.getAIsRight())) {  
 sb.append("第" + line + "行，选项是否正确不能为空<br>");  
 }  
  
 if (StringUtils.isBlank(item.getAContent()) && StringUtils.isBlank(item.getAImage())) {  
 sb.append("第" + line + "行，选项内容和选项图片必须有一个不为空<br>");  
 }  
  
 quNo = no;  
 line++;  
 }  
  
 if (!"".equals(sb.toString())) {  
 throw new ServiceException(1, sb.toString());  
 }  
 }  
  
 @ResponseBody  
 @RequestMapping(value = "import/template")  
 public ApiRest importFileTemplate(HttpServletResponse response) {  
 try {  
 String fileName = "试题导入模板.xlsx";  
 List<QuExportDTO> list = Lists.newArrayList();  
  
 QuExportDTO l1 = new QuExportDTO();  
 l1.setNo("正式导入，请删除此说明行：数字，相同的数字表示同一题的序列");  
 l1.setQContent("问题内容");  
 l1.setQAnalysis("整个问题的解析");  
 l1.setQuType("只能填写1、2、3、4；1表示单选题，2表示多选题，3表示判断题，4表示主观题");  
 l1.setQImage("题目图片，完整URL，多个用逗号隔开，限制10个");  
 l1.setQVideo("题目视频，完整URL，只限一个");  
 l1.setAImage("答案图片，完整URL，只限一个");  
 l1.setRepoList(Arrays.asList(new String[]{"已存在题库的ID，多个用逗号隔开，题库ID错误无法导入"}));  
 l1.setAContent("候选答案1");  
 l1.setAIsRight("只能填写0或1，0表示否，1表示是");  
 l1.setAAnalysis("这个项是正确的");  
  
  
 QuExportDTO l2 = new QuExportDTO();  
 l2.setQContent("找出以下可以被2整除的数（多选）");  
 l2.setQAnalysis("最基本的数学题，不做过多解析");  
 l2.setQuType("2");  
 l2.setNo("1");  
 l2.setAIsRight("1");  
 l2.setAContent("数字：2");  
 l2.setAAnalysis("2除以2=1，对的");  
  
 QuExportDTO l3 = new QuExportDTO();  
 l3.setNo("1");  
 l3.setAIsRight("0");  
 l3.setAContent("数字：3");  
 l3.setAAnalysis("3除以2=1.5，不能被整除");  
  
 QuExportDTO l4 = new QuExportDTO();  
 l4.setNo("1");  
 l4.setAIsRight("1");  
 l4.setAContent("数字：6");  
 l4.setAAnalysis("6除以2=3，对的");  
  
  
  
 list.add(l1);  
 list.add(l2);  
 list.add(l3);  
 list.add(l4);  
  
 new ExportExcel("试题数据", QuExportDTO.class, 1).setDataList(list).write(response, fileName).dispose();  
 return super.success();  
 } catch (Exception e) {  
 return super.failure("导入模板下载失败！失败信息："+e.getMessage());  
 }  
 }  
}

### 4.3.3 业务逻辑层

题库管理模块的业务逻辑层主要完成对用户题库管理逻辑的判定，同时调用题库管理模块的业务逻辑接口。比如：用户到题库管理是查询题目是否存在。题库管理模块业务逻辑层列表如图4-5所示。

表 4-5题库管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 题库管理 | QuService.java  QuServiceImpl.java | 调用QuMapper | 返回给Action |

在题库管理模块的业务逻辑层是调用了公用的QuService接口，同时在实现该接口。

QuService.java接口主要方法：

public interface QuService extends IService<Qu> {  
  
 IPage<QuDTO> paging(PagingReqDTO<QuQueryReqDTO> reqDTO);  
  
 List<Qu> listByRandom(String repoId,  
 Integer quType,  
 Integer level,  
 List<String> excludes,  
 Integer size);  
  
 QuDetailDTO detail(String id);  
  
 void save(QuDetailDTO reqDTO);  
  
 List<QuExportDTO> listForExport(QuQueryReqDTO query);  
  
 int importExcel(List<QuExportDTO> dtoList);  
}

### 4.3.4 数据持久层

题库管理模块的是数据持久层能对库内题目进行增删改查。题库管理模块数据持久层列表见表4-6所示。

表 4-6 题库管理模块数据持久层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Mapper | 调用数据模型 | 说明 |
| 题库管理 | QuMapper.java  UserMapperImpl.java | 调用 Qu.Java | 对库内题目进行增删改查操作 |

QuMapper.java定义了库内题目进行增、删、改、查的接口：

public interface QuMapper extends BaseMapper<Qu> {  
  
 List<Qu> listByRandom(@Param("repoId") String repoId,  
 @Param("quType") Integer quType,  
 @Param("level") Integer level,  
 @Param("excludes") List<String> excludes,  
 @Param("size") Integer size);  
  
 List<QuExportDTO> listForExport(@Param("query") QuQueryReqDTO query);  
  
 IPage<QuDTO> paging(Page page, @Param("query") QuQueryReqDTO query);  
  
  
}

### 4.3.5 域模型层

题库管理模块用到域模型层中Qu.java，Qu.java是一个公用域模型，在涉及到库内题目信息查询等操作时，就会调用到该模型，题库管理模块域模型层列表如表4-7所示。

表 4-7题库管理模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Qu.java | 对库内题目的增、删、改、查操作 |

Qu.java主要属性与方法：

public class Qu extends Model<Qu> {  
  
 private static final long serialVersionUID = 1L;  
  
 @TableId(value = "id", type = IdType.ID\_WORKER\_STR)  
 private String id;  
  
 @TableField("qu\_type")  
 private Integer quType;  
  
 private Integer level;  
  
 private String content;  
  
 @TableField("create\_time")  
 private Date createTime;  
  
 @TableField("update\_time")  
 private Date updateTime;  
  
 private String remark;  
  
 private String analysis;  
   
}

# 5、管理端模块详细设计

## 5.1、作业布置模块详细设计

### 5.1.1 表现层

作业布置模块的表现层主要完成教师布置作业功能。表现层对应的VUE页面列表见表5-1所示。

表 5-1 作业布置模块表现层VUE列表

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 作业布置页面 | exam/list.vue | 教师布置作业功能 |

### 5.1.2 控制层

作业布置模块的控制层负责接受来自exam包中的index.vue的用户输入，同时调用作业布置模块的业务逻辑接口，将用户查看的布置题目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。作业布置模块控制层列表见表5-2所示。

表 5-2 作业布置模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 作业布置 | ExamController.java | SUCCESS | exam/list.vue |

在控制层中ExamController.java的描述如下所示：

public class ExamController extends BaseController {  
  
 @Autowired  
 private ExamService baseService;  
  
 @ApiOperation(value = "添加或修改")  
 @RequestMapping(value = "/save", method = { RequestMethod.POST})  
 public ApiRest save(@RequestBody ExamSaveReqDTO reqDTO) {  
 baseService.save(reqDTO);  
 return super.success();  
 }  
 @ApiOperation(value = "批量删除")  
 @RequestMapping(value = "/delete", method = { RequestMethod.POST})  
 public ApiRest edit(@RequestBody BaseIdsReqDTO reqDTO) {  
 baseService.removeByIds(reqDTO.getIds());  
 return super.success();  
 }  
  
 @ApiOperation(value = "查找详情")  
 @RequestMapping(value = "/detail", method = { RequestMethod.POST})  
 public ApiRest<ExamDTO> find(@RequestBody BaseIdReqDTO reqDTO) {  
 ExamSaveReqDTO dto = baseService.findDetail(reqDTO.getId());  
 return super.success(dto);  
 }  
  
 @ApiOperation(value = "查找详情")  
 @RequestMapping(value = "/state", method = { RequestMethod.POST})  
 public ApiRest state(@RequestBody BaseStateReqDTO reqDTO) {  
 QueryWrapper<Exam> wrapper = new QueryWrapper<>();  
 wrapper.lambda().in(Exam::getId, reqDTO.getIds());  
 Exam exam = new Exam();  
 exam.setState(reqDTO.getState());  
 exam.setUpdateTime(new Date());  
 baseService.update(exam, wrapper);  
 return super.success();  
 }  
  
 @ApiOperation(value = "考试视角")  
 @RequestMapping(value = "/online-paging", method = { RequestMethod.POST})  
 public ApiRest<IPage<ExamOnlineRespDTO>> myPaging(@RequestBody PagingReqDTO<ExamDTO> reqDTO) {  
 IPage<ExamOnlineRespDTO> page = baseService.onlinePaging(reqDTO);  
 return super.success(page);  
 }  
  
 @ApiOperation(value = "分页查找")  
 @RequestMapping(value = "/paging", method = { RequestMethod.POST})  
 public ApiRest<IPage<ExamDTO>> paging(@RequestBody PagingReqDTO<ExamDTO> reqDTO) {  
 IPage<ExamDTO> page = baseService.paging(reqDTO);  
 return super.success(page);  
 }  
 @ApiOperation(value = "待阅试卷")  
 @RequestMapping(value = "/review-paging", method = { RequestMethod.POST})  
 public ApiRest<IPage<ExamReviewRespDTO>> reviewPaging(@RequestBody PagingReqDTO<ExamDTO> reqDTO) {  
 IPage<ExamReviewRespDTO> page = baseService.reviewPaging(reqDTO);  
 return super.success(page);  
 }  
  
  
}

### 5.1.3 业务逻辑层

作业布置模块的业务逻辑层主要完成对用户作业布置逻辑的判定，同时调用作业布置模块的业务逻辑接口。比如：用户到作业布置模块布置作业。作业布置模块业务逻辑层列表如图5-3所示。

表 5-3作业布置模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 作业布置 | ExamService.java  ExamServiceImpl.java | 调用ExamMapper | 返回给Action |

在作业布置模块的业务逻辑层是调用了公用的ExamService接口，同时在实现该接口。

ExamService.java接口主要方法：

public interface ExamService extends IService<Exam> {  
  
 void save(ExamSaveReqDTO reqDTO);  
  
 ExamSaveReqDTO findDetail(String id);  
  
 ExamDTO findById(String id);  
  
 IPage<ExamDTO> paging(PagingReqDTO<ExamDTO> reqDTO);  
  
 IPage<ExamOnlineRespDTO> onlinePaging(PagingReqDTO<ExamDTO> reqDTO);  
  
 IPage<ExamReviewRespDTO> reviewPaging(PagingReqDTO<ExamDTO> reqDTO);  
}

### 5.1.4 数据持久层

作业布置模块的是数据持久层能对布置的作业进行增删改查。作业布置模块数据持久层列表见表5-4所示。

表 5-4 作业布置模块数据持久层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Mapper | 调用数据模型 | 说明 |
| 作业布置 | ExamMapper.java  ExamMapperImpl.java | 调用 Exam.Java | 对库内题目进行增删改查操作 |

ExamMapper.java定义了布置的作业进行增、删、改、查的接口：

public interface ExamMapper extends BaseMapper<Exam> {  
  
 IPage<ExamDTO> paging(Page page, @Param("query") ExamDTO query);  
  
 IPage<ExamReviewRespDTO> reviewPaging(Page page, @Param("query") ExamDTO query);  
  
 IPage<ExamOnlineRespDTO> online(Page page, @Param("query") ExamDTO query);  
}

### 5.1.5 域模型层

作业布置模块用到域模型层中Exam.java，Exam.java是一个公用域模型，在涉及到布置的作业信息查询等操作时，就会调用到该模型，作业布置模块域模型层列表如表5-5所示。

表 5-5作业布置模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Exam.java | 对布置的作业的增、删、改、查操作 |

Exam.java主要属性与方法：

public class Exam extends Model<Exam> {  
  
 private static final long serialVersionUID = 1L;  
  
 @TableId(value = "id", type = IdType.ASSIGN\_ID)  
 private String id;  
  
 private String title;  
  
 private String content;  
  
 @TableField("open\_type")  
 private Integer openType;  
  
 @TableField("join\_type")  
 private Integer joinType;  
  
 private Integer level;  
  
 private Integer state;  
  
 @TableField("time\_limit")  
 private Boolean timeLimit;  
  
 @TableField("start\_time")  
 private Date startTime;  
  
 @TableField("end\_time")  
 private Date endTime;  
  
 @TableField("create\_time")  
 private Date createTime;  
  
 @TableField("update\_time")  
 private Date updateTime;  
  
 @TableField("total\_score")  
 private Integer totalScore;  
  
 @TableField("total\_time")  
 private Integer totalTime;  
  
 @TableField("qualify\_score")  
 private Integer qualifyScore;  
   
}

## 5.2、作业提交模块详细设计

### 5.2.1 表现层

作业提交模块的表现层主要完成学生提交作业功能。表现层对应的VUE页面列表见表5-6所示。

表 5-6 作业提交模块表现层VUE列表

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 作业提交页面 | book/index.vue | 学生提交作业功能 |

### 5.2.2 控制层

作业提交模块的控制层负责接受来自book包中的index.vue的用户输入，同时调用作业提交模块的业务逻辑接口，将用户查看的提交作业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。作业提交模块控制层列表见表5-7所示。

表 5-7 作业提交模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 作业提交 | UserBookController.java | SUCCESS | book/index.vue |

在控制层中UserBookController.java的描述如下所示：

public class UserBookController extends BaseController {  
  
 @Autowired  
 private UserBookService baseService;  
  
 @ApiOperation(value = "批量删除")  
 @RequestMapping(value = "/delete", method = { RequestMethod.POST})  
 public ApiRest delete(@RequestBody BaseIdsReqDTO reqDTO) {  
  
 baseService.removeByIds(reqDTO.getIds());  
 return super.success();  
 }  
  
 @ApiOperation(value = "分页查找")  
 @RequestMapping(value = "/paging", method = { RequestMethod.POST})  
 public ApiRest<IPage<UserBookDTO>> paging(@RequestBody PagingReqDTO<UserBookDTO> reqDTO) {  
  
 IPage<UserBookDTO> page = baseService.paging(reqDTO);  
  
 return super.success(page);  
 }  
  
 @ApiOperation(value = "查找列表")  
 @RequestMapping(value = "/next", method = { RequestMethod.POST})  
 public ApiRest<BaseIdRespDTO> nextQu(@RequestBody UserBookDTO reqDTO) {  
  
 String quId = baseService.findNext(reqDTO.getExamId(), reqDTO.getQuId());  
 return super.success(new BaseIdRespDTO(quId));  
 }  
}

### 5.2.3 业务逻辑层

作业提交模块的业务逻辑层主要完成对用户作业提交逻辑的判定，同时调用作业提交模块的业务逻辑接口。比如：用户到作业提交模块提交作业。作业提交模块业务逻辑层列表如图5-8所示。

表 5-8作业提交模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 作业提交 | UserBookService.java  UserBookServiceImpl.java | 调用UserBookMapper | 返回给Action |

在作业提交模块的业务逻辑层是调用了公用的UserBookService接口，同时在实现该接口。

UserBookService.java接口主要方法：

public interface UserBookService extends IService<UserBook> {  
  
 IPage<UserBookDTO> paging(PagingReqDTO<UserBookDTO> reqDTO);  
  
 void addBook(String examId, String quId);  
  
 String findNext(String examId, String quId);  
}

### 5.2.4 数据持久层

作业提交模块的是数据持久层能对提交的作业进行增删改查。作业提交模块数据持久层列表见表5-9所示。

表 5-9作业提交模块数据持久层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Mapper | 调用数据模型 | 说明 |
| 作业提交 | UserBookMapper.java  UserBookMapperImpl.java | 调用 UserBook.Java | 对提交的作业进行增删改查操作 |

UserBookMapper.java定义了布置的作业进行增、删、改、查的接口：

public interface UserBookMapper extends BaseMapper<UserBook> {  
  
}

### 5.2.5 域模型层

作业提交模块用到域模型层中UserBookr.java，UserBook.java是一个公用域模型，在涉及到提交的作业信息查询等操作时，就会调用到该模型，作业提交模块域模型层列表如表5-10所示。

表 5-10题库管理模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| UserBook.java | 对提交的作业的增、删、改、查操作 |

UserBook.java主要属性与方法：

public class UserBook extends Model<UserBook> {  
  
 private static final long serialVersionUID = 1L;  
  
 @TableId(value = "id", type = IdType.ASSIGN\_ID)  
 private String id;  
  
 @TableField("exam\_id")  
 private String examId;  
  
 @TableField("user\_id")  
 private String userId;  
  
 @TableField("qu\_id")  
 private String quId;  
  
 @TableField("create\_time")  
 private Date createTime;  
  
 @TableField("update\_time")  
 private Date updateTime;  
  
 @TableField("wrong\_count")  
 private Integer wrongCount;  
  
 private String title;  
  
 private Integer sort;  
   
}

## 5.3、作业批改模块详细设计

### 5.3.1 表现层

作业批改模块的表现层主要完成教师批改作业功能。表现层对应的VUE页面列表见表5-11所示。

表 5-11 作业批改模块表现层VUE列表

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 作业批改页面 | exam/index.vue | 教师批改作业功能 |

### 5.3.2 控制层

作业批改模块的控制层负责接受来自exam包中的index.vue的用户输入，同时调用作业批改模块的业务逻辑接口，将用户查看的批改题目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。作业批改模块控制层列表见表5-2所示。

表 5-2 作业批改模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 作业批改 | UserExamController.java | SUCCESS | exam/index.vue |

在控制层中UserExamController.java的描述如下所示：

public class UserExamController extends BaseController {  
  
 @Autowired  
 private UserExamService baseService;  
  
 @ApiOperation(value = "分页查找")  
 @RequestMapping(value = "/paging", method = { RequestMethod.POST})  
 public ApiRest<IPage<UserExamRespDTO>> paging(@RequestBody PagingReqDTO<UserExamReqDTO> reqDTO) {  
  
 IPage<UserExamRespDTO> page = baseService.paging(reqDTO);  
  
 return super.success(page);  
 }  
  
 @ApiOperation(value = "分页查找")  
 @RequestMapping(value = "/my-paging", method = { RequestMethod.POST})  
 public ApiRest<IPage<UserExamRespDTO>> MyPaging(@RequestBody PagingReqDTO<UserExamReqDTO> reqDTO) {  
  
 IPage<UserExamRespDTO> page = baseService.myPaging(reqDTO);  
  
 return super.success(page);  
 }  
}

。。。。。。。。。。

### 5.3.3 业务逻辑层

作业批改模块的业务逻辑层主要完成对用户作业批改逻辑的判定，同时调用作业批改模块的业务逻辑接口。比如：用户到作业批改模块批改作业。作业批改模块业务逻辑层列表如图5-8所示。

表 5-8作业批改模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 作业批改 | UserExamService.java  UserExamServiceImpl.java | 调用UserExamMapper | 返回给Action |

在作业批改模块的业务逻辑层是调用了公用的UserExamService接口，同时在实现该接口。

public interface UserExamService extends IService<UserExam> {  
  
 IPage<UserExamRespDTO> paging(PagingReqDTO<UserExamReqDTO> reqDTO);  
  
 IPage<UserExamRespDTO> myPaging(PagingReqDTO<UserExamReqDTO> reqDTO);  
  
 void joinResult(String userId, String examId, Integer score, boolean passed);  
}

### 5.3.4 数据持久层

作业批改模块的是数据持久层能对批改的作业进行增删改查。作业批改模块数据持久层列表见表5-9所示。

表 5-9作业批改模块数据持久层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Mapper | 调用数据模型 | 说明 |
| 作业批改 | UserExamMapper.java  UserExamMapperImpl.java | 调用 UserExam.Java | 对批改的作业进行增删改查操作 |

UserExamMapper.java定义了批改的作业进行增、删、改、查的接口：

public interface UserExamMapper extends BaseMapper<UserExam> {  
  
   
 IPage<UserExamRespDTO> paging(Page page, @Param("query") UserExamReqDTO query);  
  
}

### 5.3.5 域模型层

作业批改模块用到域模型层中UserExam.java，UserExam.java是一个公用域模型，在涉及到批改的作业信息查询等操作时，就会调用到该模型，作业批改模块域模型层列表如表5-10所示。

表 5-10题库管理模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| UserExam.java | 对批改的作业的增、删、改、查操作 |

UserExam.java主要属性与方法：

public class UserExam extends Model<UserExam> {  
  
 private static final long serialVersionUID = 1L;  
   
 @TableId(value = "id", type = IdType.ASSIGN\_ID)  
 private String id;  
   
 @TableField("user\_id")  
 private String userId;  
   
 @TableField("exam\_id")  
 private String examId;  
   
 @TableField("try\_count")  
 private Integer tryCount;  
  
 @TableField("max\_score")  
 private Integer maxScore;  
   
 private Boolean passed;  
   
 @TableField("create\_time")  
 private Date createTime;  
   
 @TableField("update\_time")  
 private Date updateTime;  
   
}

# 6、公共部分模块详细设计

## 6.1 公共页面

公共页面主要为登录成功后的首页，当输入网址错误后的404错误页面和权限不足时的401页面。

## 6.2 安全模块的详细设计

安全模块主要为网关的全局设置，通过CorsConfig.java来实现，详细实现代码为：

public class CorsConfig {  
  
 private CorsConfiguration buildConfig() {  
 CorsConfiguration corsConfiguration = new CorsConfiguration();  
 corsConfiguration.addAllowedOrigin("\*");  
 corsConfiguration.addAllowedHeader("\*");  
 corsConfiguration.addAllowedMethod("\*");  
 return corsConfiguration;  
 }  
  
 @Bean  
 public CorsFilter corsFilter() {  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 source.registerCorsConfiguration("/\*\*", buildConfig());  
 return new CorsFilter(source);  
 }  
  
}