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

**课程资料自主学习系统详细设计**

**V6.0**

**广西民族大学**

评审日期： 2021年6月25日

目录

[1．导言 4](#_Toc527)

[1.1 目的 4](#_Toc3271)

[1.2 范围 4](#_Toc20008)

[1.3 引用标准 5](#_Toc22109)

[1.4 参考资料 5](#_Toc17888)

[1.5 版本更新信息 5](#_Toc22366)

[2．详细设计简述 6](#_Toc22914)

[2.1 设计简介 6](#_Toc27436)

[2.2 模块简介 6](#_Toc31261)

[3、 界面详细设计 7](#_Toc7404)

[3.1、页面设计说明 7](#_Toc13135)

[3.2、静态页面详细设计 7](#_Toc27298)

[4、客户端模块详细设计 8](#_Toc12387)

[4.1、公用模块的详细设计 8](#_Toc19044)

[4.1.1 表现层 8](#_Toc28480)

[4.1.2 控制层 8](#_Toc18942)

[4.2、登录模块 9](#_Toc14338)

[4.2.1 表现层 9](#_Toc9389)

[4.2.2 控制层 12](#_Toc26166)

[4.2.3 业务逻辑层 14](#_Toc18411)

[4.2.4 数据持久层 14](#_Toc14604)

[4.2.5 域模型层 14](#_Toc18269)

[4.3、个人中心模块 17](#_Toc18939)

[4.3.1 表现层 17](#_Toc351)

[4.3.2 控制层 17](#_Toc1452)

[4.3.3 业务逻辑层 19](#_Toc197)

[4.3.4 数据持久层 19](#_Toc8495)

[4.3.5 域模型层 19](#_Toc19908)

[4.4、资料上传下载模块 21](#_Toc5561)

[4.4.1 表现层 21](#_Toc12077)

[4.4.2 控制层 21](#_Toc16261)

[4.4.3 业务逻辑层 23](#_Toc9814)

[4.4.4 数据持久层 23](#_Toc24715)

[4.4.5 域模型层 23](#_Toc864)

[5、管理端模块详细设计 24](#_Toc3631)

[5.1、新闻管理模块详细设计 24](#_Toc7372)

[5.1.1 表现层 24](#_Toc2318)

[5.1.2 控制层 25](#_Toc19564)

[5.1.3 业务逻辑层 27](#_Toc14486)

[5.1.4 数据持久层 27](#_Toc9530)

[5.1.5 域模型层 27](#_Toc4116)

[5.2、课程信息管理模块详细设计 28](#_Toc12451)

[5.2.1 表现层 28](#_Toc4431)

[5.2.2 控制层 29](#_Toc14134)

[5.2.3 业务逻辑层 31](#_Toc240)

[5.2.4 数据持久层 31](#_Toc30693)

[5.2.5 域模型层 31](#_Toc17226)

[5.3、用户管理模块详细设计 32](#_Toc2618)

[5.3.1 表现层 32](#_Toc19520)

[5.3.2 控制层 33](#_Toc19234)

[5.3.3 业务逻辑层 35](#_Toc9037)

[5.1.4 数据持久层 35](#_Toc29001)

[5.1.5 域模型层 35](#_Toc17674)

[6、安全模块模块详细设计 38](#_Toc3354)

# 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开发而制定的详细开发设计文档，说明程序模块的设计考虑。

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，包括程序的描述和流程逻辑。

## 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所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 导言 | 初始发布版本 |
| 002 | 2021.6.16 | 1.0 | 1、2、3章节 | 增加 |
| 003 | 2021.6.17 | 2.0 | 1、2、3章节 | 修改 |
| 004 | 2021.6.18 | 3.0 | 4、5、6章节 | 增加 |
| 005 | 2021.6.19 | 4.0 | 5、6章节 | 修改 |
| 006 | 2021.6.21 | 5.0 | 4、5章节 | 修改、增加 |
| 007 | 2021.6.22 | 6.0 | 4、5、6章节 | 修改，增加 |

# 2．详细设计简述

## 2.1 设计简介

随着计算机技术的普及和计算机网络的普遍应用，学生自主学习，教师精心辅导也成趋势。建立一个网站方便更多的人享用宝贵的教育资料源，方便教师教学，提高学生自学能力，已成趋势。

课程资料自主学习系统分层结构图如图2-1所示。

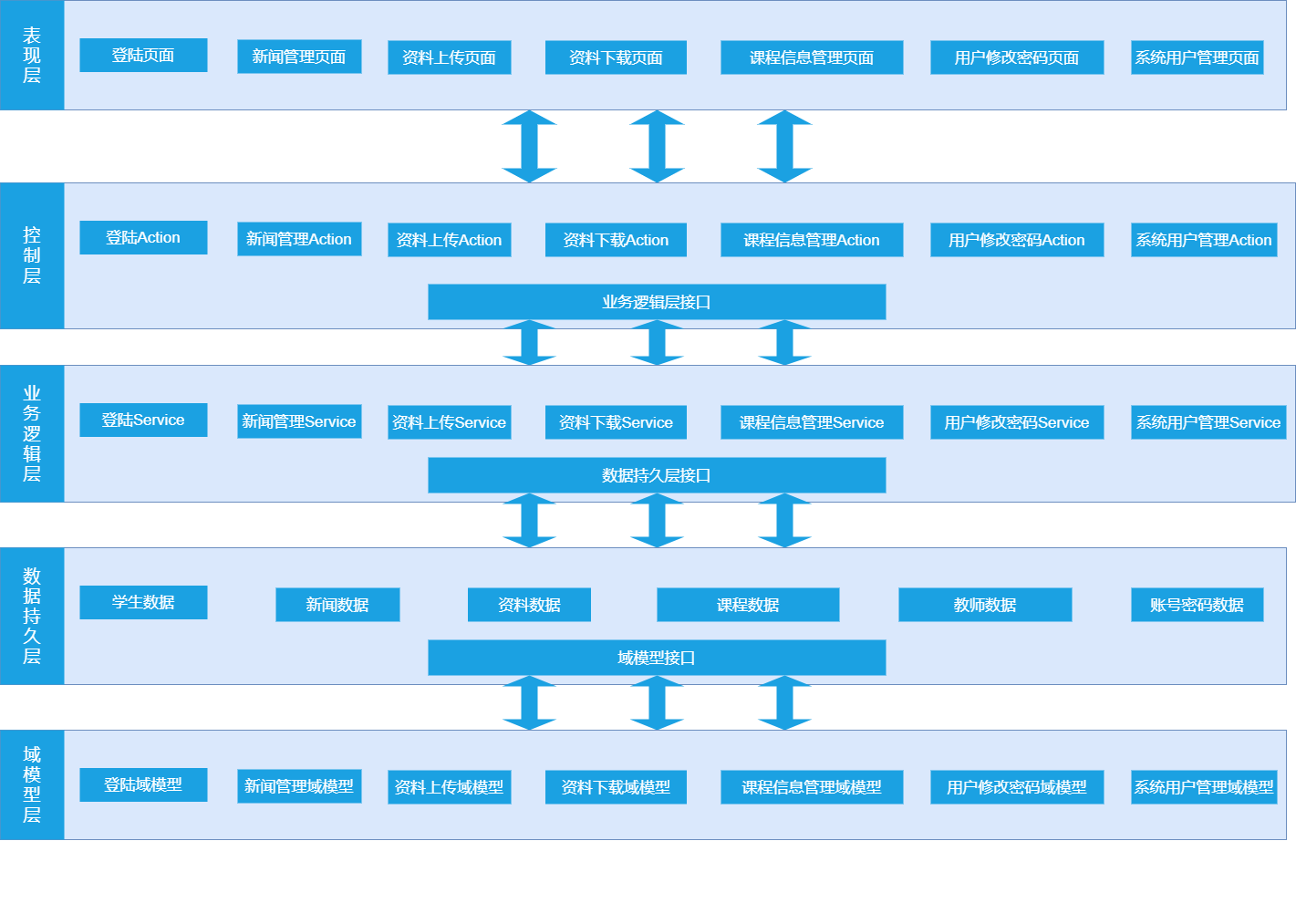
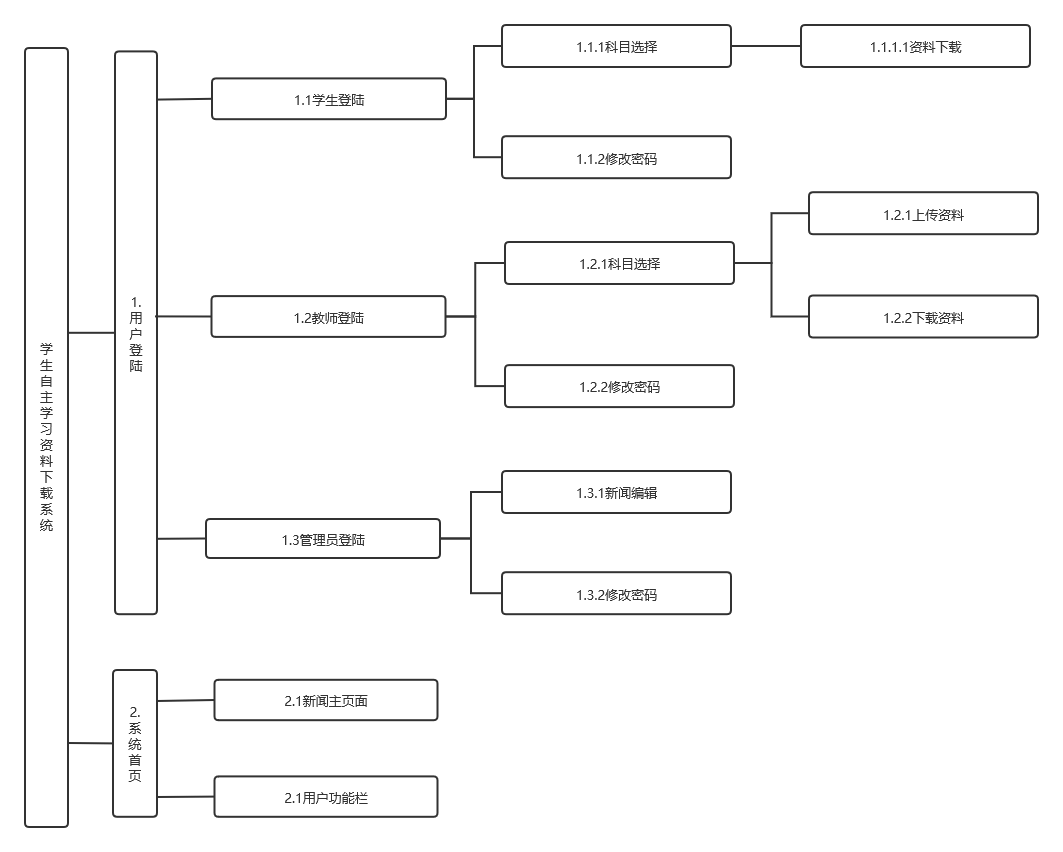


图 2-1分层结构图示意图

## 2.2 模块简介

系统功能模块图如图所示。



# 界面详细设计

# 3.1、页面设计说明

系统有学生、教师、管理员三个角色，每个角色对应的页面都不同。三角色份共用一个登录页，选择登录后进入不同的页面。系统的前后端分离，前端发起请求，后端返回数据。

## 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下表

页面表现层设计

|  |  |
| --- | --- |
| 界面 | 功能描述 |
| 登录 | 选择角色，输入账号密码登录系统 |
| 学生首页 | 学生浏览发布新闻 |
| 学生课程信息 | 学生查看所修课程及下载该课程资料 |
| 学生个人中心 | 学生查看及修改个人基本信息 |
| 教师首页 | 教师浏览发布新闻 |
| 教师课程信息 | 教师查看所授课程及上传下载该课程资料 |
| 教师个人中心 | 教师查看及修改个人基本信息 |
| 学生管理 | 管理员对学生资料进行增删查改（包括密码） |
| 教师管理 | 管理员对教师资料进行增删查改（包括密码） |
| 新闻列表 | 管理员浏览已发布的新闻 |
| 新闻添加 | 管理员创建并发布新闻 |
| 课程管理 | 管理员匹配课程名、课程号和授课教师 |

# 4、客户端模块详细设计

客户端模块主要包括登录、个人中心、资料上传下载模块。

## 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

表现层公用模块

|  |  |
| --- | --- |
| 界面 | 功能描述 |
| 登录出错页面 | 登入时出现错误及时提示错误 |
| 全部页面 | 错误显示提示框 |

### 4.1.2 控制层

功能描述：获取页面模型

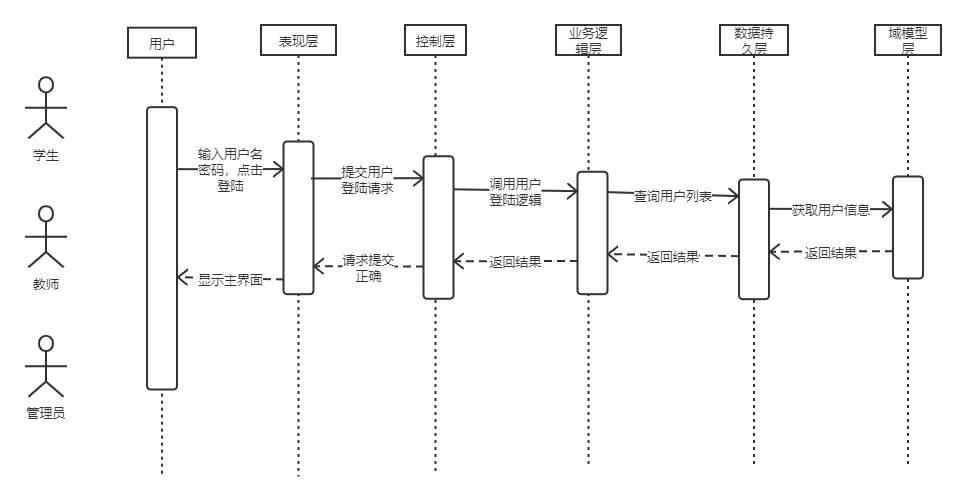
代码描述：

//函数返回页面模型

public class BaseController {  
    @Autowired  
    HttpServletRequest req;  
    /\*\*  
     \* 获取页面  
     \* @return  
     \*/  
    public Page getPage() {  
        int current = ServletRequestUtils.getIntParameter(req, "cuurent", 1);  
        int size = ServletRequestUtils.getIntParameter(req, "size", 10);  
  
        return new Page(current, size);  
    }  
}

## 4.2、登录模块

在登录模块时，系统操作示意图如图所示。

系统操作示意图

### 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。

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 登录页面 | Login.vue | 用户（教师、学生、管理员）登入 |

登录的流程图如图所示。

进入游首页



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.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所示。

登入模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 登录 | AccountController.java | SUCCESS | Login.vue—登入成功，跳转到对应界面。 |
| ERROR | login.vue---登入失败，显示提示窗口。 |

在控制层中AccountController.java的描述如下所示：

@RestController  
  
public class AccountController {  
    @Autowired  
    JwtUtils jwtUtils;  
    @Autowired  
    TeacherService teacherService;  
    @Autowired  
    StudentService studentService;  
    @Autowired  
    AdminService adminService;  
    @CrossOrigin  
    @PostMapping("/login")  
    public Result login(@Validated @RequestBody LoginDto loginDto, HttpServletResponse response) {  
        String roleType = loginDto.getRole();  
        String jwt;  
        JwtToken jwtToken;  
        switch (roleType){  
            case "Student":  
                Student student = studentService.getOne(new QueryWrapper<Student>().eq("sno", loginDto.getUsername()));  
                Assert.notNull(student, "用户不存在");  
                if(!student.getPwd().equals(loginDto.getPassword())) {  
                    return Result.fail("密码错误！");  
                }  
                jwt = jwtUtils.generateToken(student.getSno(),"Student");  
                JwtToken token = new JwtToken(jwt);  
                response.setHeader("Authorization", jwt);  
                response.setHeader("Access-control-Expose-Headers", "Authorization");  
                // 用户可以另一个接口  
                return Result.success(MapUtil.builder()  
                        .put("id",student.getId())  
                        .put("sno", student.getSno())  
                        .put("sex", student.getSex())  
                        .put("sname", student.getSname())  
                        .put("role","Student")  
                        .map()  
                );  
  
            case "Teacher":  
                Teacher teacher = teacherService.getOne(new QueryWrapper<Teacher>().eq("tno", loginDto.getUsername()));  
                Assert.notNull(teacher, "用户不存在");  
                if(!teacher.getPwd().equals(loginDto.getPassword())) {  
                    return Result.fail("密码错误！");  
                }  
                jwt = jwtUtils.generateToken(teacher.getTno(),"Teacher");  
                response.setHeader("Authorization", jwt);  
                response.setHeader("Access-Control-Expose-Headers", "Authorization");  
                  
                // 用户可以另一个接口  
                return Result.success(MapUtil.builder()  
                        .put("id",teacher.getId())  
                        .put("sno", teacher.getTno())  
                        .put("sex", teacher.getSex())  
                        .put("sname", teacher.getTname())  
                        .put("role","Teacher")  
                        .map()  
                );  
            case "Admin":  
                Admin admin = adminService.getOne(new QueryWrapper<Admin>().eq("ano", loginDto.getUsername()));  
                Assert.notNull(admin, "用户不存在");  
                if(!admin.getPwd().equals(loginDto.getPassword())) {  
                    return Result.fail("密码错误！");  
                }  
  
                jwt = jwtUtils.generateToken(admin.getAno(),"Admin");  
                response.setHeader("Authorization", jwt);  
                response.setHeader("Access-Control-Expose-Headers", "Authorization");  
                // 用户可以另一个接口  
                return Result.success(MapUtil.builder()  
                        .put("id",admin.getId())  
                        .put("sno", admin.getAno())  
                        .put("role","Admin")  
                        .map()  
                );  
        }  
        return Result.fail("未选择角色");  
    }  
  
    @RequiresAuthentication  
    @GetMapping("/logout")  
    public Result logout() {  
        SecurityUtils.getSubject().logout();  
        return Result.success(null);  
    }  
}

### 4.2.3 业务逻辑层

登录模块的业务逻辑层主要完成对用户登录逻辑的判定，同时调用登录模块的业务逻辑接口。比如：用户到登录是输入的用户名是否存在、密码是否正确，同时在对用户的身份进行判定。登录模块业务逻辑层列表如图所示。

登入模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 登录 | TeacherService.java  StudentService.java  AdminService.java | 分别调用  TeacherMapper.java  StudentMapper.java  AdminMapper.java | 返回给Login.vue |

在登入模块的业务逻辑层是分别调用了TeacherMapper.java，StudentMapper.java，AdminMapper.java接口。

对应的接口Mybatis plus映射生成实现。

### 4.2.4 数据持久层

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | DAO | 调用数据模型 | 说明 |
| 登录 | TeacherMapper.java  StudentMapper.java  AdminMapper.java | 调用 Teacher.java  Student.java  Admin.java | 调用数据库数据装配对应数据模型 |

对应实现Mybatis Plus映射数据库生成。

### 4.2.5 域模型层

登录模块用到域模型层中Teacher.java，Student.java，Admin.java，在涉及到用户信息查询等操作时，就会调用到该模型，登录模块域模型层列表如表所示。

登录模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Teacher.java  Student.java  Admin.java | Teacher实体拥有tno tname sex pwd四个属性用于装配  Student实体拥有sno sname sex pwd四个属性用于装配  Admin实体拥有id ano pwd三个属性用于装配 |

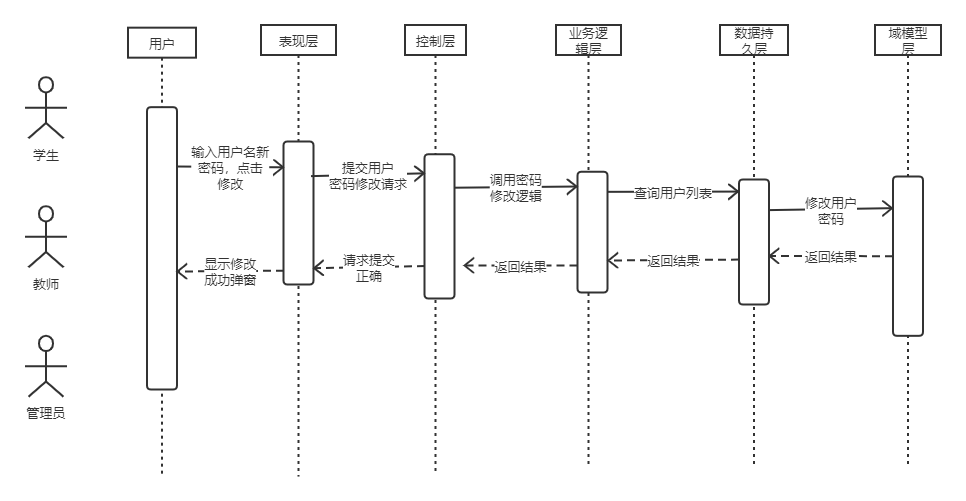
Teacher.java Student.java Admin.java属性与方法：

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Admin implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String ano;  
  
    private String pwd;  
  
  
}

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Student implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String sno;  
  
    private String sname;  
  
    private String sex;  
  
    private String pwd;  
  
  
}

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Teacher implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String tno;  
  
    private String tname;  
  
    private String sex;  
  
    private String pwd;  
  
  
}

## 4.3、个人中心模块



### 4.3.1 表现层

个人中心模块的表现层主要显示和编辑个人信息。

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 个人中心页面 | StudentInfo.vue  TeacherInfo.vue | 教师、学生查看、编辑个人信息 |

### 4.3.2 控制层

个人中心模块的控制层负责接受来自StudentInfo.vue,TeacherInfo.vue页面输入，调用StudentController和TeacherController中的update方法。个人中心模块控制层列表见表所示。

登入模块控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 修改密码 | StudentController.java  TeacherController.java | SUCCESS | 当前页面，修改成功 |
| ERROR | 当前页面，修改失败 |

在控制层中StudentController.java TeacherController.java的相关方法描述如下所示：

@CrossOrigin  
    @PostMapping("/update")  
    public Result update(@Validated @RequestBody Teacher teacher) {  
        if (teacher.getSex().equals("1")){  
            teacher.setSex("女");  
        }  
        else {  
            teacher.setSex("男");  
        }  
        if (StrUtil.isNotBlank(teacher.getPwd())) {  
            teacherService.updateById(teacher);  
        }  
        return Result.success(teacher);  
    }

@CrossOrigin  
    @PostMapping("/update")  
    public Result update(@Validated @RequestBody Student student) {  
        if (student.getSex().equals("1")){  
            student.setSex("女");  
        }  
        else {  
            student.setSex("男");  
        }  
        if (StrUtil.isNotBlank(student.getPwd())) {  
            studentService.updateById(student);  
        }  
        return Result.success(student);  
    }

### 4.3.3 业务逻辑层

个人中心模块的业务逻辑层主要完成对用户修改面逻辑的判定，同时调用个人中心模块的业务逻辑接口。

个人中心模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 修改密码 | TeacherService.java  StudentService.java | 分别调用  TeacherMapper.java  StudentMapper.java | 返回当前页面 |

在个人中心模块的业务逻辑层是分别调用了TeacherMapper.java，StudentMapper.java接口。

对应的接口Mybatis plus映射生成实现。

### 4.3.4 数据持久层

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | DAO | 调用数据模型 | 说明 |
| 登录 | TeacherMapper.java  StudentMapper.java | 调用 Teacher.java  Student.java | 调用数据库数据装配对应数据模型 |

对应实现Mybatis Plus映射数据库生成。

### 4.3.5 域模型层

个人中心模块用到域模型层中Teacher.java，Student.java，Admin.java，在涉及到用户信息查询等操作时，就会调用到该模型，个人中心模块域模型层列表如表所示。

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Teacher.java  Student.java | Teacher实体拥有tno tname sex pwd四个属性用于装配  Student实体拥有sno sname sex pwd四个属性用于装配 |

Teacher.java Student.java属性与方法：

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Admin implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String ano;  
  
    private String pwd;  
  
  
}

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Student implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String sno;  
  
    private String sname;  
  
    private String sex;  
  
    private String pwd;  
  
  
}

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Teacher implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String tno;  
  
    private String tname;  
  
    private String sex;  
  
    private String pwd;  
  
  
}

## 4.4、资料上传下载模块

### 4.4.1 表现层

资料上传下载模块的表现层用于学生、教师下载资料和教师上传资料。

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 资料上传下载页面 | StuCourse.vue  TeacherCourse.vue | 学生、教师下载资料和教师上传资料 |

### 4.4.2 控制层

资料上传下载模块的控制层负责接受来自StuCourse.vue,TeacherCouruse.vue页面输入，调用TeacherController中的方法。资料上传下载模块控制层列表见表所示。

资料上传下载控制层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 上传 | TeacherController.java | SUCCESS | 当前页面，上传成功 |
| ERROR | 当前页面，上传失败 |
| 下载 | TeacherController.java | SUCCESS | 当前页面，下载成功 |
| ERROR | 当前页面，下载失败 |

在控制层中TeacherController.java的相关方法描述如下所示：

 private  final static String rootPath = "D:/downloadTest/";  
    @RequestMapping("/file/upload")  
    public Object uploadFile(MultipartFile[] multipartFiles,String cno){  
        File fileDir = new File(rootPath);  
        if (!fileDir.exists() && !fileDir.isDirectory()) {  
            fileDir.mkdirs();  
        }  
        try {  
            if (multipartFiles != null && multipartFiles.length > 0) {  
                for(int i = 0;i<multipartFiles.length;i++){  
                    try {  
                        //以原来的名称命名,覆盖掉旧的，这里也可以使用UUID之类的方式命名，这里就没有处理了  
                        String storagePath = rootPath+multipartFiles[i].getOriginalFilename();  
                        System.out.println("上传的文件：" + multipartFiles[i].getName() + "," + multipartFiles[i].getContentType() + "," + multipartFiles[i].getOriginalFilename()  
                                +"，保存的路径为：" + storagePath);  
                        // 3种方法： 第1种  
//                        Streams.copy(multipartFiles[i].getInputStream(), new FileOutputStream(storagePath), true);  
                        // 第2种  
//                        Path path = Paths.get(storagePath);  
//                        Files.write(path,multipartFiles[i].getBytes());  
                        // 第3种  
                        Filedata filedata = new Filedata();  
                        filedata.setFilename(storagePath.substring(16));  
                        filedata.setFilepath(storagePath);  
                        filedata.setCno(cno.substring(8,cno.length()-2));  
                        filedataService.save(filedata);  
                        multipartFiles[i].transferTo(new File(storagePath));  
                    } catch (IOException e) {  
                        e.printStackTrace();  
                    }  
                }  
            }  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        //前端可以通过状态码，判断文件是否上传成功  
        return Result.success("文件上传成功");  
    }  
    @CrossOrigin  
    @GetMapping("/file/getFileList")  
    public Result fileList(String cno) {  
        cno = cno.substring(8,cno.length()-2);  
        Page<Filedata> pageData = (Page<Filedata>) filedataService.page(getPage(),  
                new QueryWrapper<Filedata>()  
                        .eq(StrUtil.isNotBlank(cno), "cno", cno)  
        );  
        return Result.success(pageData);  
    }  
    /\*\*  
     \*  
     \* @param fileName 文件名  
     \* @param response  
     \* @return  
     \*/  
    @CrossOrigin  
    /\*\*  
     \*  
     \* @param fileName 文件名  
     \* @param response  
     \* @return  
     \*/  
    @RequestMapping("/file/download")  
    public Object downloadFile(@RequestParam String fileName, HttpServletResponse response){  
        OutputStream os = null;  
        InputStream is= null;  
        try {  
            // 取得输出流  
            os = response.getOutputStream();  
            // 清空输出流  
            response.reset();  
            response.setContentType("application/x-download;charset=utf-8");  
            //Content-Disposition中指定的类型是文件的扩展名，并且弹出的下载对话框中的文件类型图片是按照文件的扩展名显示的，点保存后，文件以filename的值命名，  
            // 保存类型以Content中设置的为准。注意：在设置Content-Disposition头字段之前，一定要设置Content-Type头字段。  
            //把文件名按UTF-8取出，并按ISO8859-1编码，保证弹出窗口中的文件名中文不乱码，中文不要太多，最多支持17个中文，因为header有150个字节限制。  
            response.setHeader("Content-Disposition", "attachment;filename="+ new String(fileName.getBytes("utf-8"),"ISO8859-1"));  
            //读取流  
            File f = new File(rootPath+fileName);  
            is = new FileInputStream(f);  
            if (is == null) {  
                System.out.println("下载附件失败，请检查文件“" + fileName + "”是否存在");  
                return Result.fail("下载附件失败，请检查文件“" + fileName + "”是否存在");  
            }  
            //复制  
            IOUtils.copy(is, response.getOutputStream());  
            response.getOutputStream().flush();  
        } catch (IOException e) {  
            return Result.fail("下载附件失败,error:"+e.getMessage());  
        }  
        //文件的关闭放在finally中  
        finally  
        {  
            try {  
                if (is != null) {  
                    is.close();  
                }  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
            try {  
                if (os != null) {  
                    os.close();  
                }  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
        //其实，这个返回什么都不重要  
        return Result.success("下载成功");  
    }

### 4.4.3 业务逻辑层

资料上传下载模块的业务逻辑层主要完成对资料上传下载逻辑的判定，同时调用上传下载的业务逻辑接口。

资料上传下载模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 资料上传下载 | TeacherService.java  StudentService.java | 调用  FileDataMapper.java | 返回当前页面 |

在资料上传下载资料模块的业务逻辑层是分别调用了FileDataMapper.java接口。

对应的接口Mybatis plus映射生成实现。

### 4.4.4 数据持久层

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | DAO | 调用数据模型 | 说明 |
| 资料上传下载 | FileDataMapper.java | 调用FileData.java | 调用数据库数据装配对应数据模型 |

对应实现Mybatis Plus映射数据库生成。

### 4.4.5 域模型层

资料上传下载模块用到域模型层中FileData.java，在涉及到资料上传下载等操作时，就会调用到该模型，资料上传下载模块域模型层列表如所示。

资料上传下载模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| FileData.java | FileData实体拥有filename filepath cno三个属性用于装配 |

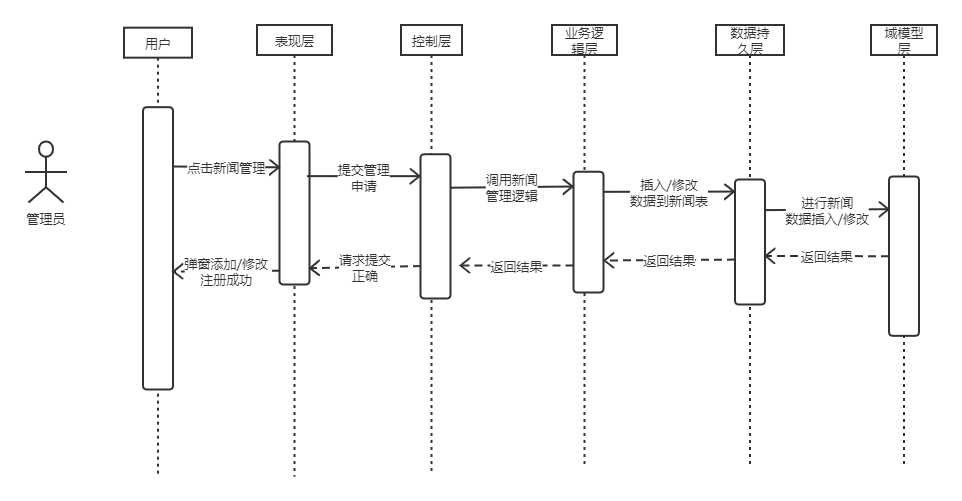
FileData.java属性与方法：

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Filedata implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String filename;  
  
    private String filepath;  
  
    private String cno;  
  
  
}

# 5、管理端模块详细设计

管理端模块主要包括新闻管理、用户管理、课程信息管理模块。

## 5.1、新闻管理模块详细设计

在新闻管理模块时，系统操作示意图如图所示。

### 5.1.1 表现层

新闻管理模块的表现层对新闻列表进行增删查改。

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 新闻管理页面 | NewsEdit.vue  NewsList.vue  NewsDetail.vue | 对新闻列表进行增删查改 |

### 5.1.2 控制层

新闻管理模块的控制层负责接受来自NewsEdit.vue，NewsList.vue，NewsDetail.vue页面输入，调用AdminController中的方法。新闻管理模块控制层列表见表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 创建、编辑、查看新闻 | AdminCotroller.java | SUCCESS | 当前页面，提示对应信息成功 |
| ERROR | 当前页面，提示对应信息失败 |

在控制层中AdminController.java的相关方法描述如下所示：

    @GetMapping("/news")  
    public Result news(Integer currentPage) {  
        if(currentPage == null || currentPage < 1) currentPage = 1;  
        Page page = new Page(currentPage, 5);  
        IPage pageData = newsService.page(page, new QueryWrapper<News>().orderByDesc("created"));  
        return Result.success(pageData);  
    }  
    @GetMapping("/news/{id}")  
    public Result detail(@PathVariable(name = "id") String id) {  
        News news = newsService.getById(id);  
        Assert.notNull(news, "该新闻已删除！");  
        return Result.success(news);  
    }  
  
    @PostMapping("/news/edit")  
    public Result edit(@Validated @RequestBody News news) {  
        System.out.println(news.toString());  
        News temp = null;  
        if(news.getId() != null) {  
            temp = newsService.getById(news.getId());  
        } else {  
            temp = new News();  
            temp.setCreated(LocalDateTime.now());  
        }  
        BeanUtil.copyProperties(news, temp, "id", "created");  
        newsService.saveOrUpdate(temp);  
        return Result.success("操作成功");  
    }

### 5.1.3 业务逻辑层

新闻管理模块的业务逻辑层主要完成对新闻列表进行增删查改逻辑的判定，同时调用新闻管理模块的业务逻辑接口。

新闻管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 新闻列表进行增删查改 | NewsService.java | 分别调用  TeacherMapper.java  StudentMapper.java | 返回当前页面 |

在新闻管理模块的业务逻辑层是分别调用了TeacherMapper.java，StudentMapper.java接口。

对应的接口Mybatis plus映射生成实现。

### 5.1.4 数据持久层

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | DAO | 调用数据模型 | 说明 |
| 新闻查看、添加、修改 | NewsMapper.java | 调用News.java | 调用数据库数据装配对应数据模型 |

对应实现Mybatis Plus映射数据库生成。

### 5.1.5 域模型层

新闻管理模块用到域模型层中News.java，在涉及到新闻查看添加编辑等操作时，就会调用到该模型，新闻管理模块域模型层列表如表所示。

新闻管理模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| News.java | News实体拥有title description content created四个属性用于装配 |

News.java属性与方法：

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class News implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String title;  
  
    private String description;  
  
    private String content;  
  
    private LocalDateTime created;  
  
  
}

## 5.2、课程信息管理模块详细设计

### 5.2.1 表现层

课程信息管理模块的表现层对课程信息进行增删查改。

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 课程信息管理页面 | CourseList.vue | 对课程信息进行增删查改 |

### 5.2.2 控制层

课程信息管理模块的控制层负责接受来自CourseList.vue页面输入，调用CourseController中的方法。课程信息管理模块控制层列表见表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 对课程信息进行增删查改 | CourseCotroller.java | SUCCESS | 当前页面，提示对应信息成功 |
| ERROR | 当前页面，提示对应信息失败 |

在控制层中CourseController.java的相关方法描述如下所示：

@RestController  
@RequestMapping("/course")  
public class CourseController extends BaseController{  
    @Autowired  
    CourseService courseService;  
    @Autowired  
    TeacherService teacherService;  
    @Autowired  
    SCourseService sCourseService;  
    @CrossOrigin  
    @GetMapping("st")  
    public Result list(String cno, Integer current, Integer size) {  
        if (current!=null&size!=null){  
            IPage page= new Page<>(current,size);  
            Page<Course> pageData = (Page<Course>) courseService.page(page,new QueryWrapper<Course>().like(StrUtil.isNotBlank(cno),"cno",cno).or().like(StrUtil.isNotBlank(cno),"cname",cno));  
            return Result.success(pageData);  
        }  
        Page<Course> pageData = (Page<Course>) courseService.page(getPage(),  
                new QueryWrapper<Course>()  
                        .like(StrUtil.isNotBlank(cno), "cno", cno)  
        );  
        return Result.success(pageData);  
    }  
    //课程信息获取  
    @GetMapping("/tnost")  
    public Result getTnoList() {  
        List<String> tnoList = new ArrayList<>();  
        List<Teacher> courseList = teacherService.list();  
        for (Teacher teacher:courseList  
             ) {  
            tnoList.add(teacher.getTno());  
        }  
        return Result.success(tnoList);  
    }  
    //课程信息获取  
    @GetMapping("/Info/{id}")  
    public Result getCourseInfo(@PathVariable String id) {  
        Course course = courseService.getById(id);  
        Assert.notNull(course, "找不到该课程！");  
        return Result.success(course);  
    }  
    //课程信息更新  
    @CrossOrigin  
    @PostMapping("/update")  
    public Result update(@Validated @RequestBody Course course) {  
        if (StrUtil.isNotBlank(course.getTno())) {  
            courseService.updateById(course);  
        }  
        return Result.success(course);  
    }  
    //课程信息保存  
    @CrossOrigin  
    @PostMapping("IMG_258ve")  
    public Result save(@Validated @RequestBody Course course) {  
        if (StrUtil.isNotBlank(course.getTno())) {  
           courseService.save(course);  
        }  
        return Result.success(course);  
    }  
  
    //课程删除  
    @PostMapping("/delete")  
    public Result delete(@RequestBody Integer[] ids){  
        courseService.removeByIds(Arrays.asList(ids));  
        courseService.remove(new QueryWrapper<Course>().in("id", ids));  
        return Result.success("操作成功");  
    }  
    //教师获取课程  
    @GetMapping("/getCourse/{id}")  
    public Result getCourse(@PathVariable String id){  
        List<Course> list= courseService.list(new QueryWrapper<Course>().eq(StrUtil.isNotBlank(id),"tno",id));  
        List<CourseDto> courseList = new ArrayList<>();  
        for (Course course:list  
             ) {  
            CourseDto courseDto = new CourseDto();  
            courseDto.setCname(course.getCname());  
            courseDto.setCno(course.getCno());  
           courseList.add(courseDto);  
        }  
        return Result.success(courseList);  
    }  
    //学生获取课程  
    @GetMapping("/getStuCourse/{id}")  
    public Result getStuCourse(@PathVariable String id){  
        List<SCourse> list= sCourseService.list(new QueryWrapper<SCourse>().eq(StrUtil.isNotBlank(id),"sno",id));  
        List<Course> courseList = new ArrayList<>();  
        for (SCourse scourse:list  
             ) {  
            Course course = courseService.getOne(new QueryWrapper<Course>().eq(StrUtil.isNotBlank(scourse.getCno()),"cno",scourse.getCno()));  
            courseList.add(course);  
        }  
        return Result.success(courseList);  
    }  
}

### 5.2.3 业务逻辑层

课程信息管理模块的业务逻辑层主要完成对课程信息进行增删查改逻辑的判定，同时调用课程信息管理模块的业务逻辑接口。

课程信息管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 对课程信息进行增删查改 | CourseService.java | 分别调用  CourseMapper.java | 返回当前页面 |

在课程信息管理模块的业务逻辑层调用了CourseMapper接口。

对应的接口Mybatis plus映射生成实现。

### 5.2.4 数据持久层

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | DAO | 调用数据模型 | 说明 |
| 对课程信息进行增删查改 | CourseMapper.java | 调用Course.java | 调用数据库数据装配对应数据模型 |

对应实现Mybatis Plus映射数据库生成。

### 5.2.5 域模型层

课程信息管理模块用到域模型层中Course.java，在涉及到课程信息进行增删查改等操作时，就会调用到该模型，课程信息模块域模型层列表如表所示。

课程信息模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Course.java | Course.java实体拥有cno cname tno三个属性用于装配 |

Course.java属性与方法：

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Course implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String cno;  
  
    private String cname;  
  
    private String tno;  
  
  
}

## 5.3、用户管理模块详细设计

### 5.3.1 表现层

用户管理模块的表现层对课程信息进行增删查改。

|  |  |  |
| --- | --- | --- |
| 界面 | VUE | 功能描述 |
| 用户管理页面 | AdminManager.vue  teacherManager.vue | 对学生、教师进行增删查改 |

### 5.3.2 控制层

用户管理模块的控制层负责接受来自AdminManager.vue teacherManager.vue页面输入，调用StudentController和TeacherController中的方法。用户管理模块控制层列表见表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Action | 转移说明 | 出口 |
| 对学生、教师进行增删查改 | StudentController.java  TeacherController.java | SUCCESS | 当前页面，提示对应信息成功 |
| ERROR | 当前页面，提示对应信息失败 |

在控制层中StudentController.java TeacherController.java的相关方法描述如下所示：

  //学生信息获取  
    @GetMapping("/Info/{id}")  
    public Result getStuInfo(@PathVariable String id) {  
        Student student = studentService.getById(id);  
        Assert.notNull(student, "找不到该学生！");  
        return Result.success(student);  
    }  
    //学生信息更新  
    @CrossOrigin  
    @PostMapping("/update")  
    public Result update(@Validated @RequestBody Student student) {  
        if (student.getSex().equals("1")){  
            student.setSex("女");  
        }  
        else {  
            student.setSex("男");  
        }  
        if (StrUtil.isNotBlank(student.getPwd())) {  
            studentService.updateById(student);  
        }  
        return Result.success(student);  
    }  
    //学生信息保存  
    @CrossOrigin  
    @PostMapping("/ve")  
    public Result save(@Validated @RequestBody Student student) {  
        if (student.getSex().equals("1")){  
            student.setSex("女");  
        }  
        else {  
            student.setSex("男");  
        }  
        if (StrUtil.isNotBlank(student.getPwd())) {  
            studentService.save(student);  
        }  
        return Result.success(student);  
    }  
  
    //学生删除  
    @PostMapping("/delete")  
    public Result delete(@RequestBody Integer[] ids){  
        studentService.removeByIds(Arrays.asList(ids));  
        studentService.remove(new QueryWrapper<Student>().in("id", ids));  
        return Result.success("操作成功");  
    }

### 5.3.3 业务逻辑层

用户管理模块的业务逻辑层主要完成对用户信息进行增删查改逻辑的判定，同时调用用户管理模块的业务逻辑接口。

用户管理模块业务逻辑层列表

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | Service | 调用说明 | 出口 |
| 对用户信息进行增删查改 | StudentService.java  TeacherService.java | 调用  TeacherMapper.java  StudentMapper.java | 返回当前页面 |

在用户管理模块的业务逻辑层调用TeacherMapper和StudentMapper接口。

对应的接口Mybatis plus映射生成实现。

### 5.1.4 数据持久层

|  |  |  |  |
| --- | --- | --- | --- |
| 事件 | DAO | 调用数据模型 | 说明 |
| 对用户信息进行增删查改 | TeacherMapper.java  StudentMapper.java | 调用Student.java  和Teacher.java | 调用数据库数据装配对应数据模型 |

对应实现Mybatis Plus映射数据库生成。

### 5.1.5 域模型层

用户管理模块用到域模型层中Course.java，在涉及到用户的增删查改等操作时，就会调用到该模型，用户管理模块域模型层列表如表所示。

用户管理模块域模型层列表

|  |  |
| --- | --- |
| 域模型 | 描述 |
| Student.java  Teacher.java | Teacher实体拥有tno tname sex pwd四个属性用于装配  Student实体拥有sno sname sex pwd四个属性用于装配 |

Student.java Teacher.java属性与方法：

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Admin implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String ano;  
  
    private String pwd;  
  
  
}

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Student implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String sno;  
  
    private String sname;  
  
    private String sex;  
  
    private String pwd;  
  
  
}

@Data  
@EqualsAndHashCode(callSuper = false)  
@Accessors(chain = true)  
public class Teacher implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
  
    @TableId(value = "id", type = IdType.AUTO)  
    private Integer id;  
  
    private String tno;  
  
    private String tname;  
  
    private String sex;  
  
    private String pwd;  
  
  
}

# 6、安全模块模块详细设计

**概述：整合shiro+jwt，并将session存入redis实现会话共享。**

**6.1、自定义的jwt**

/\*\*  
 \* jwt工具类  
 \*/  
@Slf4j  
@Data  
@Component  
@ConfigurationProperties(prefix = "myhub.jwt")  
public class JwtUtils {  
  
    private String secret;  
    private long expire;  
    private String header;  
  
    /\*\*  
     \* 生成jwt token  
     \*/  
    public String generateToken(String no,String role) {  
        Date nowDate = new Date();  
        //过期时间  
        Date expireDate = new Date(nowDate.getTime() + expire \* 10000);  
        return Jwts.builder()  
                .setHeaderParam("typ", "JWT")  
                .setSubject(role+","+no) //sub(Subject)：代表这个JWT的主体，即它的所有人，这个是一个json格式的字符串，可以存放什么userid，roldid之类的，作为什么用户的唯一标志。  
                .setIssuedAt(nowDate) //iat: jwt的签发时间  
                .setExpiration(expireDate) //设置过期时间  
                .signWith(SignatureAlgorithm.HS512, secret)  
                .compact(); //就开始压缩为xxxxxxxxxxxxxx.xxxxxxxxxxxxxxx.xxxxxxxxxxxxx这样的jwt  
    }  
  
    public Claims getClaimByToken(String token) {  
        try {  
            return Jwts.parser() //得到DefaultJwtParser  
                    .setSigningKey(secret) //设置签名的密钥  
                    .parseClaimsJws(token)  
                    .getBody(); //设置需要解析的jwt  
        }catch (Exception e){  
            log.debug("validate is token error ", e);  
            return null;  
        }  
    }  
  
    /\*\*  
     \* token是否过期  
     \* @return  true：过期  
     \*/  
    public boolean isTokenExpired(Date expiration) {  
        return expiration.before(new Date());  
    }  
}

**6.2、自定义的jwt注入shrio实现过滤拦截**

@Component  
public class JwtFilter extends AuthenticatingFilter {  
    @Autowired  
    JwtUtils jwtUtils;  
    @Override  
    protected AuthenticationToken createToken(ServletRequest servletRequest, ServletResponse servletResponse) throws Exception {  
        HttpServletRequest request = (HttpServletRequest) servletRequest;  
        String jwt = request.getHeader("Authorization");  
        if (StringUtils.isEmpty(jwt)){  
            return null;  
        }  
        return new com.example.coursesystem.shiro.JwtToken(jwt);  
    }  
  
    @Override  
    protected boolean onAccessDenied(ServletRequest servletRequest, ServletResponse servletResponse) throws Exception {  
        HttpServletRequest request = (HttpServletRequest) servletRequest;  
        String token = request.getHeader("Authorization");  
        if (StringUtils.isEmpty(token)){ //无jwt时交给注解进行拦截  
            return true;  
        }else {  
            //校验jwt  
            Claims claims = jwtUtils.getClaimByToken(token);  
            if (claims==null||jwtUtils.isTokenExpired(claims.getExpiration())){  
                throw new ExpiredCredentialsException("token已失效，请重新登录");  
            }  
            return executeLogin(servletRequest,servletResponse);  
        }  
    }  
  
    @Override  
    protected boolean onLoginFailure(AuthenticationToken token, AuthenticationException e, ServletRequest request, ServletResponse response) {  
        HttpServletResponse httpServletResponse = (HttpServletResponse) response;  
        Throwable throwable = e.getCause()==null?e:e.getCause();  
        Result result = Result.fail(throwable.getMessage());  
        String json = JSONUtil.toJsonStr(result);  
        try {  
            httpServletResponse.getWriter().print(json);  
        } catch (IOException ioException) {  
            ioException.printStackTrace();  
        }  
        return false;  
    }  
  
    /\*\*  
     \* 对跨域提供支持  
     \*/  
    @Override  
    protected boolean preHandle(ServletRequest request, ServletResponse response) throws Exception {  
        HttpServletRequest httpServletRequest = WebUtils.toHttp(request);  
        HttpServletResponse httpServletResponse = WebUtils.toHttp(response);  
        httpServletResponse.setHeader("Access-control-Allow-Origin", httpServletRequest.getHeader("Origin"));  
        httpServletResponse.setHeader("Access-Control-Allow-Methods", "GET,POST,OPTIONS,PUT,DELETE");  
        httpServletResponse.setHeader("Access-Control-Allow-Headers", httpServletRequest.getHeader("Access-Control-Request-Headers"));  
        // 跨域时会首先发送一个OPTIONS请求，这里我们给OPTIONS请求直接返回正常状态  
        if (httpServletRequest.getMethod().equals(IMG_256RequestMethod.OPTIONS.name())) {  
            httpServletResponse.setStatus(org.springframework.http.HttpStatus.OK.value());  
            return false;  
        }  
        return super.preHandle(request, response);  
    }  
}

**6.3、ShiroConfig的配置**

@Configuration  
public class ShiroConfig {  
    @Autowired  
    JwtFilter jwtFilter;  
  
    @Autowired  
    RedisSessionDAO redisSessionDAO;  
  
    @Autowired  
    RedisCacheManager redisCacheManager;  
    @Bean  
    public SessionManager sessionManager(RedisSessionDAO redisSessionDAO) {  
        DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();  
  
        // inject redisSessionDAO  
        sessionManager.setSessionDAO(redisSessionDAO);  
        return sessionManager;  
    }  
  
    @Bean  
    public DefaultWebSecurityManager securityManager(AccountRealm accountRealm,  
                                                     SessionManager sessionManager,  
                                                     RedisCacheManager redisCacheManager) {  
  
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager(accountRealm);  
  
        //inject sessionManager  
        securityManager.setSessionManager(sessionManager);  
  
        // inject redisCacheManager  
        securityManager.setCacheManager(redisCacheManager);  
        return securityManager;  
    }  
  
    @Bean  
    public ShiroFilterChainDefinition shiroFilterChainDefinition() {  
        DefaultShiroFilterChainDefinition chainDefinition = new DefaultShiroFilterChainDefinition();  
  
        Map<String, String> filterMap = new LinkedHashMap<>();  
  
        filterMap.put("/\*\*", "jwt");  
        chainDefinition.addPathDefinitions(filterMap);  
        return chainDefinition;  
    }  
  
    @Bean("shiroFilterFactoryBean")  
    public ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager,  
                                                         ShiroFilterChainDefinition shiroFilterChainDefinition) {  
        ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();  
        shiroFilter.setSecurityManager(securityManager);  
  
        Map<String, Filter> filters = new HashMap<>();  
        filters.put("jwt", jwtFilter);  
        shiroFilter.setFilters(filters);  
  
        Map<String, String> filterMap = shiroFilterChainDefinition.getFilterChainMap();  
  
//        Map<String, String> filterMap = new LinkedHashMap<String, String>();  
  
        filterMap.put("/login", "anon");  
  
        //设置需要过滤的链接  
        filterMap.put("/admin/\*\*","authc");  
  
        shiroFilter.setFilterChainDefinitionMap(filterMap);  
        return shiroFilter;  
    }  
  
  
}

**6.4、Realm实现身份的校验**

@Component  
public class AccountRealm extends AuthorizingRealm {  
    @Autowired  
    JwtUtils jwtUtils;  
    @Autowired  
    TeacherService teacherService;  
    @Autowired  
    StudentService studentService;  
    @Autowired  
    AdminService adminService;  
    @Override  
    public boolean supports(AuthenticationToken token) {  
        return token instanceof JwtToken;  
    }  
  
    @Override  
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {  
        Object principal = principalCollection.getPrimaryPrincipal();  
        Set<String> roles = new HashSet<>();  
        roles.add(principal.toString());  
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(roles);  
        return info;  
    }  
  
    @Override  
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {  
        JwtToken jwtToken = (JwtToken) token;  
        //获取userId  
        String subject = jwtUtils.getClaimByToken((String) jwtToken.getPrincipal()).getSubject();  
        String[] a = subject.split(","); //a[0]为role,a[1]为no  
        if ("Student".equals(a[0])){  
            Student student = studentService.getOne(new QueryWrapper<Student>().eq("sno", a[1]));  
            if (student==null){  
                throw new UnknownAccountException("账户为空");  
            }  
            AccountProfile profile =  new AccountProfile();  
            profile.setRole(a[0]);  
            profile.setId(Long.parseLong(a[1]));  
            SimpleAuthenticationInfo info  = new SimpleAuthenticationInfo(profile,jwtToken.getCredentials(),getName());  
            return info;  
        }  
        else if ("Teacher".equals(a[0])){  
            Teacher teacher = teacherService.getOne(new QueryWrapper<Teacher>().eq("tno", a[1]));  
            if (teacher==null){  
                throw new UnknownAccountException("账户为空");  
            }  
            AccountProfile profile =  new AccountProfile();  
            profile.setRole(a[0]);  
            profile.setId(Long.parseLong(a[1]));  
            return new SimpleAuthenticationInfo(profile,token.getCredentials(),getName());  
        }  
        else if ("Admin".equals(a[0])){  
            Admin admin = adminService.getOne(new QueryWrapper<Admin>().eq("ano", a[1]));  
            if (admin==null){  
                throw new UnknownAccountException("账户为空");  
            }  
            AccountProfile profile =  new AccountProfile();  
            profile.setRole(a[0]);  
            profile.setId(Long.parseLong(a[1]));  
  
            return new SimpleAuthenticationInfo(profile,token.getCredentials(),getName());  
        }  
        throw new AuthenticationException();  
    }

**签字：**

|  |  |  |  |
| --- | --- | --- | --- |
| 完成者 | | 审核者 | |
| 签署人姓名 | 胡龙威 | 签署人姓名 | 胡斌 |
| 签署日期 | 2021.06.22 | 签署日期 | 2021.06.23 |