大连理工大学本科毕业设计（论文）

大连理工大学本科毕业设计（论文）题目

The Subject of Undergraduate Graduation Project (Thesis) of DUT

学　院（系）：软件学院

专　　业：软件工程

学 生 姓 名：刘树东

学　　号：201792409

指 导 教 师：李明楚

评 阅 教 师：

完 成 日 期：May 24, 2021

大连理工大学

Dalian University of Technology

原创性声明

本人郑重声明：本人所呈交的毕业设计（论文），是在指导老师的指导下独立进行研究所取得的成果。毕业设计（论文）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。

本声明的法律责任由本人承担。

作者签名：　　日期：May 24, 2021

关于使用授权的声明

本人在指导老师指导下所完成的毕业设计（论文）及相关的资料（包括图纸、试验记录、原始数据、实物照片、图片、录音带、设计手稿等），知识产权归属大连理工大学。本人完全了解大连理工大学有关保存、使用毕业设计（论文）的规定，本人授权大连理工大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业设计（论文）。如果发表相关成果，一定征得指导教师同意，且第一署名单位为大连理工大学。本人离校后使用毕业毕业设计（论文）或与该论文直接相关的学术论文或成果时，第一署名单位仍然为大连理工大学。

论文作者签名：　　日期：

指导老师签名：　　日期：

摘　要

本系统的建立旨在将传统的使用纸质论文评审的流程变为 Web 系统，使学生教师可以在网络上完成评审工作，免除了论文传递中的纸张浪费，减小了传递过程中论文丢失的可能性，方便教务统计论文评审情况，论文得分情况。

本文针对论文评审系统相关内容对国内外高校进行了广泛的调查，了解学生，教师以及教务的需求。在技术方面，本系统采用较新，稳定成熟且发展前景较好的框架搭建系统，保证系统的可维护性，例如系统的总体架构选择的前后端分离的架构设计，前后端可独自开发部署和升级，系统的持久化方面选择了 ORM 工具中的 Hibernate 框架，该框架可以适应多种数据库，所以校方可以方便的切换数据库类型。在界面的设计上，本系统参考了许多优秀的 UI 设计和交互设计，使系统界面美观且易操作。

论文分为六章，第一章绪论介绍了研究背景和本系统的开发意义；第二章系统相关技术介绍了本系统的使用的技术以及和其他技术之间的抉择；第三章需求分析从不同用户的角度分析了系统的需求；第四章系统设计详细地设计了系统的各个模块的结构和工作流程；第五章实现与测试重点介绍了系统设计中遇到的技术难题，最后一章为设计总结，总结了系统设计和实现过程中值得记录的内容。

关键词：论文评审；自动化办公；毕业设计（论文）;Spring Boot；Spring Security；JWT;Hibernate；Vue.js

The Subject of Undergraduate Graduation Project (Thesis) of DUT

Abstract

The establishment of this system aims to change the traditional process of using paper paper for review into a Web system, so that students and teachers can complete the review work on the network, eliminate the paper waste in the delivery of paper, reduce the possibility of paper lost in the delivery process, and facilitate the educational administration statistics of paper review, paper score.

In this paper, a broad survey of domestic and foreign colleges and universities has been conducted on the content of paper review system to understand the needs of students, teachers and educational administration.In terms of technology, the system adopts a relatively new, stable, mature and promising framework to build the system, so as to ensure the maintainability of the system. For example, the overall architecture of the system is designed to separate the front and back ends, so that the front and back ends can be independently developed, deployed and upgraded. For the persistence of the system, the Hibernate framework in the ORM tool is chosen.The framework can be adapted to a variety of databases, so schools can easily switch database types.In the interface design, the system refers to many excellent UI design and interaction design, so that the system interface is beautiful and easy to operate.

The paper is divided into six chapters. The first chapter introduces the research background and the significance of the development of this system.The second chapter introduces the technology used in this system and the choice between other technologies.The third chapter analyzes the requirements of the system from the perspective of different users.The fourth chapter of the system design detailed design of the system of each module structure and work flow;The fifth chapter mainly introduces the technical problems encountered in the system design. The last chapter is the design summary, which summarizes the content worth recording in the process of system design and implementation.

Key Words: Paper review;Automated office;Graduation Design (Thesis);Spring Boot；Spring Security；JWT;Hibernate；Vue.js

目　　录

1 绪论

1.1 研究背景

本科生毕业论文是本科教育的最后一步，是对学生大学学习的最后一次考验，毕业论文的质量是考验本科教育质量的最后一关[1]，而毕业论文评审则是把控这最后一关的重要环节。在十二届人大会议，李克强总理曾强调要将互联网技术与各行各业紧密相连，从而有效推动我国电子商务以及工业互联网等领域的稳健前行[2]。自国家提出“互联网+”战略，互联网逐渐成为主流思路和公认的“风口”[3]。但是，目前对于毕业生的论文评审还存在一些问题，最大的问题是许多学校主要还是采用纸质文档的方式，不但对于纸张是一种浪费，更是一种人力的浪费，论文的提交都需实物传递，传递时间相对于上传文档要更长，并且评审结果难以记行集中的管理[4]，根据调查每年部分高校都会出现因为论文管理工作出现错误而导致学生毕业受阻的情况[5]，同时这种管理方式也不符合信息化办公不可阻挡的趋势。因此，我们需要高效的论文评审系统从而可以提高毕业生的论文质量，减轻导师和学校管理的负担[6]。

1.2 国内外发展现状

最早的数字化校园的概念是由美国麻省理工学院在 20 世纪 70 年代提出，自此数字化校园平台、信息化教务管理平台在西方各个高校逐渐发展并走向成熟。其中高效的论文评审及管理系统在一定程度上是这些顶级高校优秀学术研究的保证。这些系统一般支持导师和学生的沟通，对学生完成情况的跟踪，作为学生安排写毕业论文的进度的参考。随着我国社会经济以及科学技术的发展，计算机的应用也变得越来越广泛[7]。国内许多高校也陆续向这些国际顶级高效学习开发自己的论文管理系统，但是和国外顶级高校之间还存在着较大差距[4]。虽然在国外已经有了相对成熟的范例，但是由于国情，文化等等差异，我们并不能完全照搬，仍然需要在不断的实践中逐渐摸索适合国内的数字化校园平台。就目前国内高校，从论文评审这个方面来说，一些高校尚未有论文评审系统，评审过程中仍然使用纸质作为论文的媒介，一些高校已经有了自己的在线论文评审系统，但是这些系统，虽然一定程度上实现了自动化，实际上存在着以下两个主要问题：

（1） 有效性问题[6]

部分功能并不完善，检索效率低，查找和维护困难等问题，甚至已经很老旧以至于系统设计不符合现如今的论文管理，对于部分学生，在毕业论文前期，面临着许多不同的挑战，比如应届生春招、研究生复试、公务员考试、以及应聘单位实习等，留给毕业论文的时间实际上很少[8]，它们需要快速地了解整个进

度，安排整个写毕业论文的进度，并且需要方便快速地在线提交论文以及等到反馈，因为他们可能并不在学校，但是这些系统做的并不好。

（2） 公正性问题[6]

公正性问题也就说评审老师的选择是否符合论文题目，如果这个选择并不合适，那么对学生论文的评判可能会有失公正[9]。这两个关键问题处理不好可能会导致答辩人修改论文时间不足、评审人评审质量低下等问题，不能满足培养高素质人才的要求[6]。基于以上讨论，选择使用 B/S 架构实现此系统，好处是可以屏蔽不同操作系统的差异，不必发布各个版本的桌面应用就可以投入使用。前端选择使用 Vue 框架，该框架具有组件化、视图，数据，结构分离、虚拟 DOM、运行速度更快等优点，类似这种的前端开发框架将传统的琐碎的，杂乱无章的开发过程变成像后端一样有条理结构化的开发过程；后端选择使用 Spring Boot 框架，该框架具有简化配置，简化代码量等优点；数据库连接选择 Hibernate 框架，该框架具有轻量，移植性好等优点。

1.3 研究意义及本人工作

本系统旨在实现论文评审评分工作的自动化，帮助学生更好地完成论文，教师更好地评审论文，教务更好得管理论文。本人主要工作为了解业务需求，定义系统功能，做好系统设计，最后将系统实现并测试系统功能。

2 系统相关技术

2.1 Spring Boot 介绍

该介绍基于 Spring Boot 2.4.5。

Spring Boot 可以更快创建独立的、基于 Spring 的生产级应用。该框架解决了 Spring 框架配置非常繁琐的问题，简化了项目开发流程[10]。

Spring Boot 特点

（1） 创建独立的 Spring 应用程序。

（2） 直接嵌入 Tomcat、Jetty 或 Undertow(不需要部署 WAR 文件)。

（3） 提供可选择的“启动器”依赖项来简化构建配置。

（4） 尽可能地自动配置 Spring 和第三方库。

（5） 提供生产就绪特性，如度量、运行状况检查和外部化配置。

（6） 绝对不需要代码生成，也不需要 XML 配置。

Spring Boot 为 Spring 生态系统带来了一种”opinionated” 的方法。首次发布于 2014 年中期。Spring Boot 经过了很多的开发和改进。它的 2.0 版在 2018 年初发布。

Spring 5 和 Spring Boot 2 需要 Java 8，这使得使用核心 Java 8 函数成为可能。这包括接口上的默认方法和一些 abstractxxxconfiguration 类、Nullable-Support 等相关的弃用。

Spring 中使用的所有 Java EE 规范都升级到了 Java EE 7。Spring Boot 1 是支持 Java 7 的最后一个 Spring Boot 版本，而 Spring Boot 2 将是第一个运行在 Java 9 上的版本 (预计也将运行在 Java 10 上)。这还伴随着一些基本依赖项的升级: 例如 Tomcat、Hibernate 和 Gradle。

Spring Boot 2 的一个亮点是几乎自动可用的 HTTP/2 支持：Tomcat、Jetty、Undertow 和 Reactor Netty 的使用版本支持 HTTP/2 开箱即用。配置非常简单：服务器证书必须配置 server.ssl.* 下的属性。那么 HTTP/2 可以通过 server.http2.enabled 激活。HTTP/2 是快速网络的重要构建块。可以通过压缩头、服务器推送、请求管道和多路复用请求来大幅降低延迟。

2.1.1 Spring 5: Spring Boot 2 及其模块的基础

第五代 Spring 框架于 2017 年 9 月底发布。Spring 5 是 Spring Boot 中的大多数框架的基础，并为“响应式编程”主题创建了基础。

Spring 使创建 Java 企业应用程序变得很容易。它提供了在企业环境中使用 Java 语言所需的一切，支持 Groovy 和 Kotlin 作为 JVM 上的替代语言，以及根据应用程序的需

要创建多种体系结构的灵活性。从 Spring Framework 5.1 开始，Spring 需要 JDK 8+ (Java SE 8+)，并提供对 JDK 11 LTS 的开箱即用支持。

Spring 支持广泛的应用程序场景。在大型企业中，应用程序往往存在很长时间，必须运行在 JDK 和应用服务器上，而 JDK 和应用服务器的升级周期超出了开发人员的控制范围。其他的可能作为一个内置服务器的 jar 运行，可能在云环境中。还有一些可能是不需要服务器的独立应用程序 (如批处理或集成工作负载)。

Spring 是开源的。它有一个庞大而活跃的社区，基于各种不同的现实世界用例提供持续的反馈。这帮助 Spring 在很长一段时间内成功地发展。

Spring 出现于 2003 年，是为了应对早期 J2EE 规范的复杂性。虽然有些人认为 Java EE 和 Spring 是竞争对手，但 Spring 实际上是 Java EE 的补充。Spring 编程模型不包含 Java EE 平台规范；相反，它与 EE 中精心挑选的各个规范集成在一起：

（1） Servlet API (JSR 340)
（2） WebSocket API (JSR 356)
（3） Concurrency Utilities (JSR 236)
（4） JSON Binding API (JSR 367)
（5） Bean Validation (JSR 303)
（6） JPA (JSR 338)
（7） JMS (JSR 914)
（8） 以及 JTA/JCA 设置，以便在必要时进行事务协调。

Spring 框架还支持依赖注入 (JSR 330) 和公共注释 (JSR 250) 规范，应用程序开发人员可以选择使用这些规范来代替 Spring 框架提供的特定于 Spring 的机制。

随着时间的推移，Java EE 在应用程序开发中的角色已经演变。在 Java EE 和 Spring 的早期，创建应用程序是为了部署到应用服务器上。今天，在 Spring Boot 的帮助下，应用程序以一种对 devops 和云友好的方式创建，其中嵌入了 Servlet 容器，更改起来很简单。从 Spring Framework 5 开始，WebFlux 应用程序甚至不直接使用 Servlet API，可以运行在不是 Servlet 容器的服务器上 (比如 Netty)。

2.2 Spring Security 介绍

该介绍基于 Spring Security 5.5.0。

Spring Security 是一个功能强大且高度可定制的身份验证和访问控制框架。它是保护基于 spring 的应用程序的实际标准。目前比较流行的框架有两个，一个是 Spring Security，另一个是 Shiro。Shiro 更加简单易用，Spring Security 更容易和 Spring 整合，更适应分布

式应用，但是配置繁琐，概念复杂。近年随着 Spring Boot 的大火，Spring Security 逐渐流行，因为前者在一定程度上简化了后者的配置。

Spring Security 特点：

（1） 对身份验证和授权的全面且可扩展的支持。

（2） 防止例如点击劫持，跨站点请求伪造等攻击。

（3） Servlet API 的集成。

（4） 与 Spring Web MVC 的可选继承。

2.2.1 Spring Security 5.5 新特性

（1） OAuth 2.0 客户端

（1） 添加了对 Jwt 客户端认证中的 private-key-jwt 和 client-secret-jwt 的支持。

（2） 添加了 Jwt Bearer Authorization Grant 的支持。

（3） 增加了 ReactiveOAuth2AuthorizedClientService 的 R2DBC 实现。

（2） OAuth 2.0 资源服务

（1） 增强的 JWT 译码器派生的签名算法

（2） 改进的内容协商

（3） 改进的 mult-tenancy 支持

（3） SAML 2.0 服务提供者

（1） 增加了 OpenSAML 4 的支持

（2） 增强的 SAML 2.0 断言解密

（3） 增加了基于文件的配置，用于断言当事人元数据

（4） 增强的依赖方元数据支持

（5） 增加了对 ap-specified 签名方法的支持

（4） 配置

（1） 引入了 DispatcherType 请求匹配器

（2） 为过滤器安全性引入了 AuthorizationManager

（5） Kotlin DSL

（1） 添加 rememberMe 支持

JSON Web Token(JWT) 是一种开放的、行业标准 RFC 7519 方法，用于在双方之间安全地表示声明。如图2.1是一个编码和解码的例子。

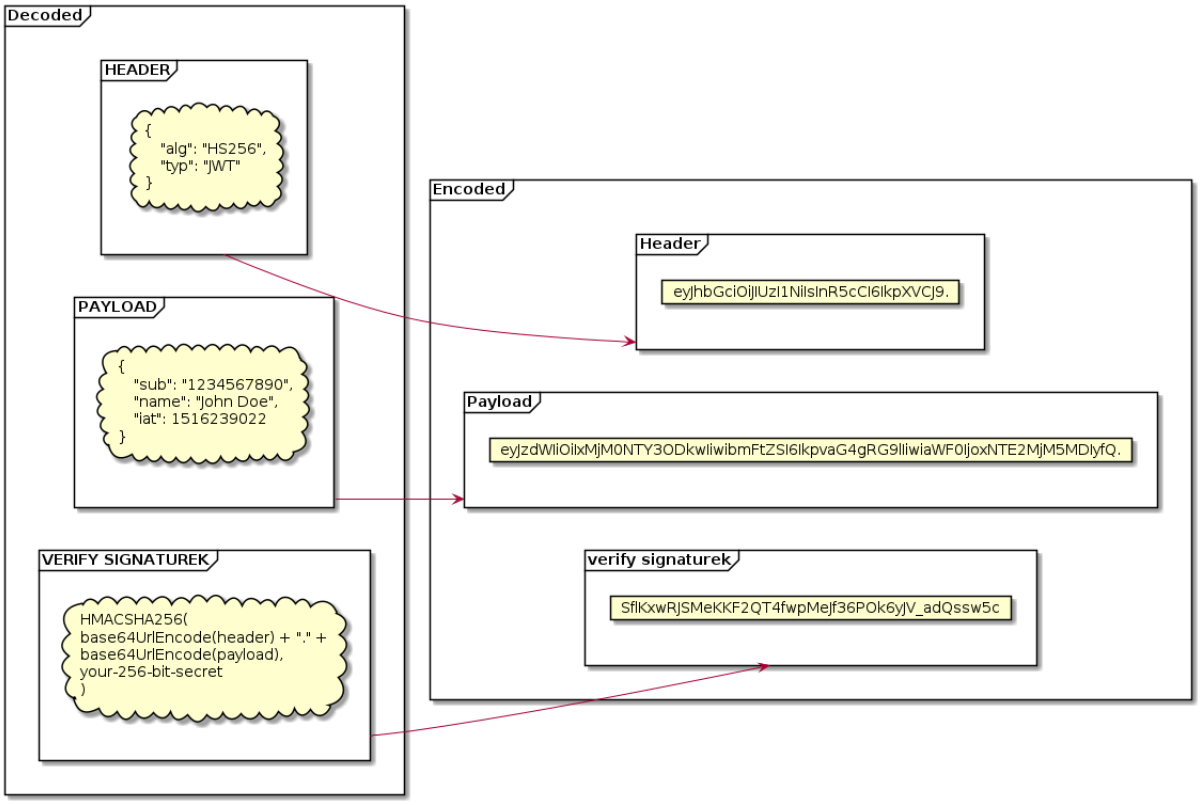

图 2.1　JWT 示例

2.3　Spring Data Jpa & Hibernate 介绍

该介绍基于 Spring Data Jpa 2.5.1，Hibernate 5.4。

目前主流的 ORM(Object/Relation Mapping) 工具有 Hibernate 和 MyBatis。

JPA & Spring Data JPA & Hibernate 三者之间的关系：

（1） JPA(Java Persistence API) 为对象-关系 (ORM) 映射提供了 POJO 持久性模型。JPA 是一种规范，仅定义了接口，但是并没有实现。

（2） Spring Data Jpa 是 Spring Data 家族的一员，使得基于 JPA 的存储库的实现变得更容易。该模块着力于增强对基于 JPA 的数据访问层的支持，简单的说它是一个简化 Jpa 的框架。很长一段时间以来，实现应用程序的数据访问层一直很麻烦，为了执行简单的查询以及分页和审计，必须编写太多的模板代码。Spring Date Jpa 做的事情就是尽量的减少开发人员需要编写的代码量，开发人员要做的事情是编写 Repository 接口，由框架实现复杂的工作。

Spring Data Jpa 特点：

（1） 很好的构建基于 Spring 和 Jpa 的存储库。

（2） 支持 Querydsl 谓词，从而支持类型安全的 JPA 差选。

（3） 对于领域类有选择的映射属性。

（4） 分页支持，动态查询支持，集成自定义数据访问代码。

（3） Hibernate 是实现了 JPA 规范的框架（图2.2）。Hibernate 与 MyBatis 的对比在一

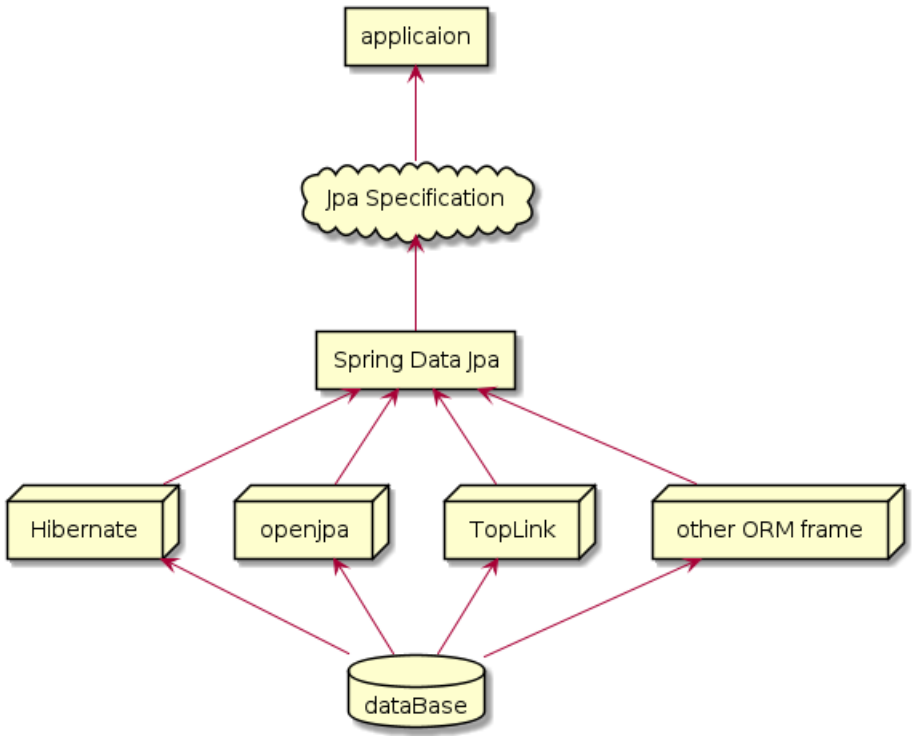

图 2.2　Jpa 和 Hibernate

定程度上和 Spring Security 和 Shiro 的对比很相似，刚上手 Hibernate 的感觉，会觉得非常的好用，几乎不用配置就可以使用了，但是随着功能的复杂，需要深入了解复杂的框架本身，对于数据库模型设计的要求也更高，所以 Hibernate 的学习成本相较于 MyBatis 更高一些，这是它的缺点，但同时 Hibernate 的功能更加强大，这是它的优势。

Hibernate 特点：

（1） 支持自己的“原生”API，同时是 JAP 的实现，这样，它可以轻松地在支持 JPA 的任何环境中使用，包括 Java SE 应用程序，Java EE 应用程序服务器，Enterprise OSGi 容器等。

（2） 允许按照自然的面向对象的方式进行持久化开发。Hibernate 不需要持久化的接口或基类，并且允许任何类或数据持久化。

（3） Hibernate 具有较高的性能，它支持延迟初始化，大量的抓取策略和带有自动版本控制和时间戳的乐观锁。无论是开发人员生产方面还是在运行时性能方面，它始终拥有优于直接使用 JDBC 代码的性能。

（4） Hibernate 被设计为在应用服务器集群中工作，并提供高度可伸缩的体系结构。它在任何环境中都可以很好地扩展：使用它来驱动服务于数百个用户地内部网，或者用于服务于数十万用户的关键任务应用程序。

（5） Hibernate 拥有卓越的稳定性和质量。

总的来说，Hibernate 是一个很强大的 ORM 工具，但是如果想要使用好它，是需要较多的经验的。

2.4 Vue.js 介绍

Vue.js 特点：

（1） 轻量级框架

该框架可以快速下载和安装库。

（2） 虚拟 DOM 渲染及其性能

相关概念：

（1） DOM(The Document Object Model)：Web 页面的 API，允许读取和操作页面的内容，结构和样式。一种对 HTML 基于对象的表示（HTML 文档 ↔ 对象模型）

（2） node tree：DOM 的对象结构。

```
// html
<!doctype html>
<html lang="en">
 <head>
   <title>My first web page</title>
 </head>
 <body>
    <h1>Hello, world!</h1>
    <p>How are you?</p>
 </body>
</html>
```

```
// 对应的 node tree
html
```

```
|-head
|   |-title
|       |-My first web page
|-body
    |-h1
    |   |-hello,world!
    |-p
        |-How are you?
```

（3） Render Tree:（DOM：element 的表示 + CSSOM：element 样式的表示）

虚拟 DOM：如果对象更改了它的状态，浏览器需要更新信息并且重新渲染到屏幕上，这个过程需要更新所有的 DOM，开销是很大的。而 Vue.js 利用虚拟 DOM 解决了这个问题，可以将虚拟 DOM 理解为 DOM 的一个拷贝并且它可以找到需要更新的 element 而不是直接更新所有的 element，这大大的提高了应用程序的性能。

（3） 响应式的双向数据绑定（图2.3）

Vue 改变了前端开发者使用 jQuery 直接对页面上的 DOM 元素进行操作的习惯, 通过数据和模板双向绑定的形式更好地组织和简化了 Web 开发。[11]

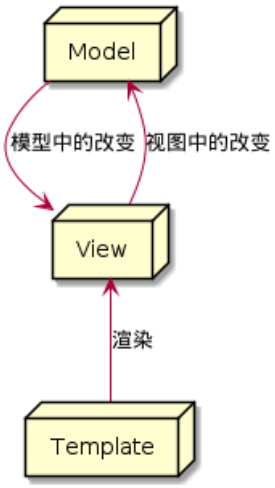

图 2.3　双向数据绑定

（4） CBA

Vue 是基于组件的体系结构（Component Based Architecture(CBA)），这种体系结构的好处主要有：

（1） 代码的可读性：每个组件可以保存在同一个单独的文件，包括结构，

样式和逻辑，可以通过代码看出这个组件的功能。

（2） 组件复用：封装好的组件可以再多个地方使用。

（3） 便于单元测试：每个组件刚好可以作为单元测试的一个单元。

Vue3 变化：

（1） 创建 app

```
// Vue2
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
}).\$mount('#app')
```

```
// Vue3
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'

createApp(App).mount('#app')
```

改变化解决了 Vue2 全局配置在单元测试中产生的问题（测试用户会因全局配置而互相影响）

（2） 多 Root

```
<template>
  <div> // 在Vue3中可以去掉
    <h1>{{ msg }}</h1>
    <button @click="count++">count is: {{ count }}</button>
    <p>Edit
       <code>components/HelloWorld.vue</code>
      to test hot module replacement.</p>
  </div> // 在Vue3中可以去掉
</template>
```

（3） 组合式 API。这应该是 Vue3 最大的变化，该变化主要是为了解决 Vue2 大组件难以阅读和维护的问题以及管理和维护组件之间的逻辑的问题。主要变化有：

（1） Setup

```
<template>
  <!-- your template code -->
</template>
<script>
export default {
  setup() {
    // more code to write
  }
};
</script>
```

（2） Reactive References

ref 即"Reactive References"，它包装了原始数据并且允许我们跟踪变化。（在 Vue2 中是使用 data() 包装原始内部对象。

（3） Methods/Computed/Watch

Vue2 中有一个单独的 methods/Computed/Watch 部分用来编写方法，在 vue3 中则在 setup() 中编写。

（4） Props

在 Vue3 中不需要 this 就可以访问 props

（5） 移除了 Vue filter

在 Vue2 中 filter 主要用于普通文本的格式化，而在 Vue3 中被移除了。移除的主要原因是它的性能和普通的函数没有区别，而写成普通的函数则会有更大的复用空间。

（6） 多个 v-model

一个组件可以处理多个 v-model，这样在多值处理地情况下，Vue3 能更简单地通过多个 event 将值从子组件传递到父组件。

（7） 模块化

可以将 setup() 中的内容分离到另一个 composition function 中并保存在一个 js 文件中以便重用。

（8） 声明周期钩子

（1） (Vue2)beforeCreate()
（2） (vue2) 无 ⇒(Vue3)setup()
（3） (Vue2)created()
（4） (Vue2)beforeMount()
（5） (Vue2)mounted()

（6） (Vue2)beforeUpdate()
（7） (Vue2)updated()
（8） (Vue2)beforeDestroy()⇒(Vue3)beforeUnmount()
（9） (Vue2)destroyed()⇒(Vue3)unmouted()
（10） (Vue3)onRenderTracked()
（11） (Vue3)onRenderTriggered()

并且 beforeCreate() 和 create() 不在需要，它们的工作放在 setup() 中完成。

2.5 Vuex 介绍

Vuex 是 Vue.js 应用程序的状态管理模式库。它作为应用程序中所有组件的集中存储，其规则确保状态只能以可预测的方式变化。它还集成了 Vue 的官方 devtools 扩展，以提供诸如零配置 time-travel 调试和状态快照导出/导入等高级特性。

2.5.1 状态管理模式

一个 Vue app 包括三个部分

（1） 状态（state），真正推动程序运行的源头。
（2） 视图（view），状态的声明性映射。
（3） 动作（actions），状态可能改变的方式，以响应用户从视图的输入。

如图2.4是一个单向数据流的例子。

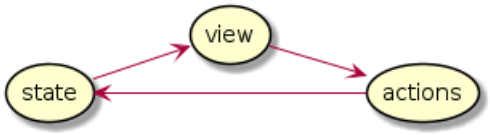

图 2.4 单向数据流

当有多个共享同一个状态的组件时，不同组件之间需要传递装状态去同步状态，这个过程是复杂并且容易出错的。所以将状态从组件中提取出来统一管理，如图2.5所示，任何组件都可以访问状态以及改变状态，从而免除了传递状态的过程。这种模式使代码更容易维护。

五个核心概念：

（1） State

Vuex 使用单一的状态树，这意味着每个应用程序都使用一个单一的状态树，但是这并不会与 Vue 的模块化思想冲突，因为 Vuex 同样可以将 state、mutation 和 actions 分割为各个子模块，使用 namespace 可以保证各个子模块中的 state、mutation 和 actions 可以使用和其他子模块重复的名字。

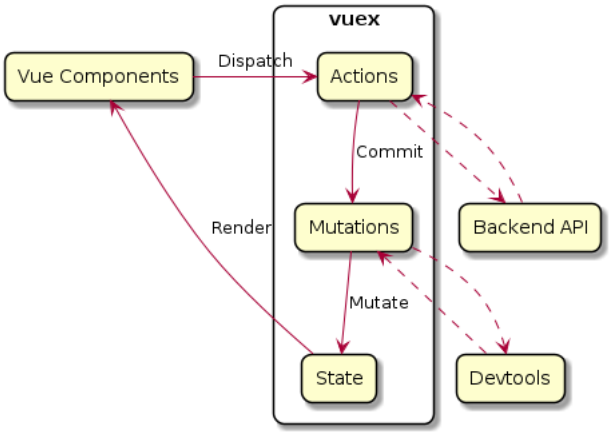

图 2.5　状态管理模式

（2） Getters

Vuex 允许在存储中定义“getter”。可以将它们视为存储的 computed 属性。与 computed 属性一样，getter 的结果是基于它的依赖关系进行缓存的，并且只有在它的一些依赖关系发生变化时才会重新计算。

（3） Mutations

在 Vuex store 中实际更改状态的唯一方法是提交一个 mutation。Vuex mutation 与事件非常相似: 每个 mutation 都有一个字符串类型和一个处理程序。

（4） Actions

Action 和 mutation 相似，不同之处在于 mutation 操作 state，action 操作 mutation，以及 action 可以使用异步操作。

（5） Modules

随着应用程序的增长，如果所有的状态都粗放在一个对象中，会变得非常臃肿，这时可以将其划分为许多子模块。

2.6　Echarts 介绍

Echarts 是快速构建基于 web 可视化的声明性框架。

Echarts 特点：

（1） 灵活的图表类型：Apache Echarts 提供了 20 多种现成的图表类型，以及十几种组件，每个组件都一个任意组合使用。

（2） 优雅的视觉设计：默认设计遵循可视化原则，支持响应式设计。灵活的配置使其易于定制。

（3） 友好的可访问性：自动生成的图表描述帮助有阅读障碍的用户理解图表的内

容和背后的故事。

（4） 强大的渲染引擎：可以轻松地切换画布和 SVG 渲染。渐进式渲染和流加载使实时渲染超大数据成为可能。

（5） 专业的数据分析：通过数据集管理数据，数据集支持数据转换，如过滤，类聚和回归，以帮助实现相同数据的多维分析。

（6） 一个健康的社区：活跃的开源社区确保了项目的健康发展，并为第三方扩展贡献了丰富的资源。

2.7 本章小结

本章主要介绍了系统使用的相关技术，以及和其他系统并未使用的技术，介绍了选择这些技术的原因，在选择上都需要考虑功能是否较为强大，社区是否较为活跃，扩展性是否较强等等，综合多方面进行考虑，为之后顺序实现一个稳定易扩展的系统。

3 需求分析

3.1 可行性分析

3.1.1 技术可行性分析

本系统使用在上一章系统相关技术介绍的全部技术，后端采用 Spring Boot，安全框架采用 Spring Security，Token 采用 JWT，前端使用 Vue.js 框架配置 Vuex 和 Vue-router，以上可实现一个前后端分离的系统。

3.1.2 经济可行性分析

系统的开发需要一名开发人员，一名长期的运行维护人员，以及服务器的费用。从收益的角度来看，系统可以减少多论文评审过程中的人类资源以及传递论文过程中使用的纸张。

3.1.3 操作可行性分析

完成后的系统界面十分简洁易懂且功能完善，由于系统是 Web 应用，用户只需要有一台可以上网的浏览器便可以使用系统，非常方便。

3.2 业务流程分析

（1） 管理员通过教职员学生管理系统与论文评审评分系统联动导入学生，教师信息。

（2） 学生上传论文

在允许论文上传期间，学生可以上传论文，待评审导师进行检查。

（3） 论文审查

教师可以查看由自己管理的所有学生的论文，教师可以下载并审查论文，若论文未达到审批要求，教师可进行留言并贴上缺陷标签，待学生修改论文并重新上传论文。

（4） 论文评分

若论文达到审批要求，教师可对论文进行打分。

（5） 论文评论

教师可以针对论文中出现的问题进行评论，学生也可以回复，探讨问题的解决方式。

（6） 问题标签

教师在评论的时候可以选择问题标签，便于学生快速了解问题所在。

（7） 得分统计

管理员可以查看所有学生的得分统计情况，可以按照不同分数段进行分类统计。

（8） 教师评审任务统计

管理员可以查看所有教师的评审任务完成情况，作为之后任务分配的参考。

（9） 问题标签统计

管理原可以查看问题标签出现过的次数，作为通知学生需要重点注意的问题的参考。

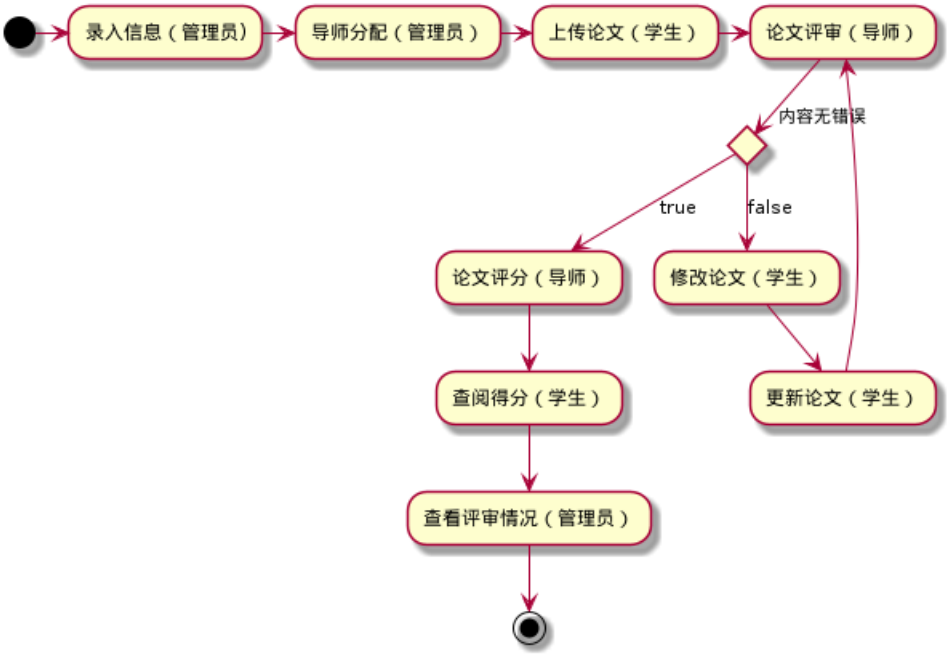

图 3.1　系统总体活动流程图

3.3　系统功能需求分析

3.3.1　系统总体功能

如图3.2所示，系统的主要功能有用户登陆功能（学生，教师，管理员），论文上传功能（学生），论文评审评分功能（教师）, 统计评审情况功能（管理员）。

3.3.2　用户登录

如图3.3所示，用户填入用户名，用户密码，登陆人员类型（学生，教师，管理员），点击提交，后台从数据库验证用户名，用户密码是否正确，如果正确，则为用户授权，不同用户拥有不同的权力，将包含用户信息的 Token 返回给前端，前端将 Token 保存在

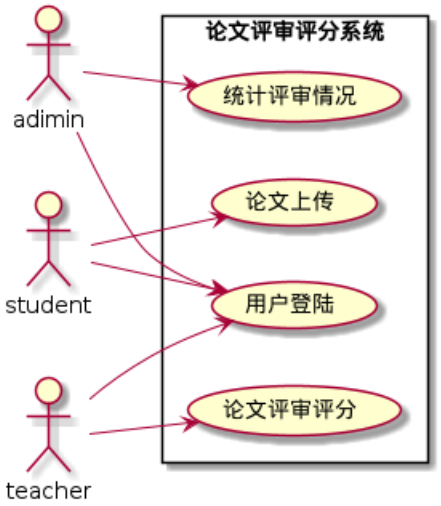

图 3.2　系统总体功能用例图

浏览器，之后的通信都需要携带 Token，后台可以从 Token 中获取用户信息用来验证用户合法性。

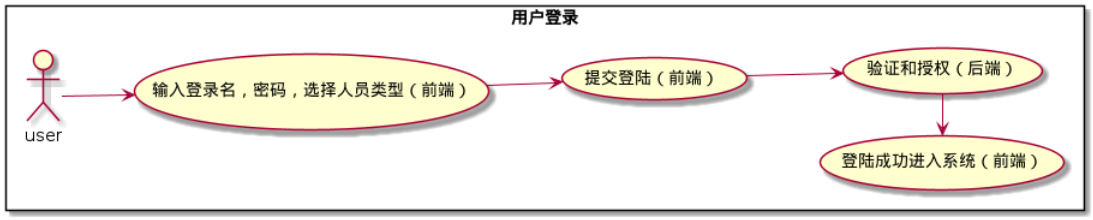

图 3.3　用户登录用例图

3.3.3　论文上传

如图3.4所示，管理员可以选择是否开放论文上传通道，在论文开放通道期间，学生可以使用用户名，用户密码登陆系统，选择自己的论文进行上传。且当自己的论文没有通过教师的评审时，根据老师的评论和问题标签，学生修改自己的论文并更新自己的文档。

3.3.4　论文下载

如图3.5所示，学生，教师，管理员分别可以从各自的页面下载论文的最新版本进行查看。

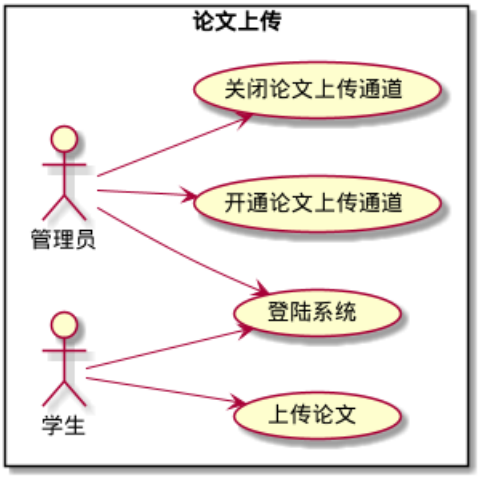

图 3.4　论文上传用例图

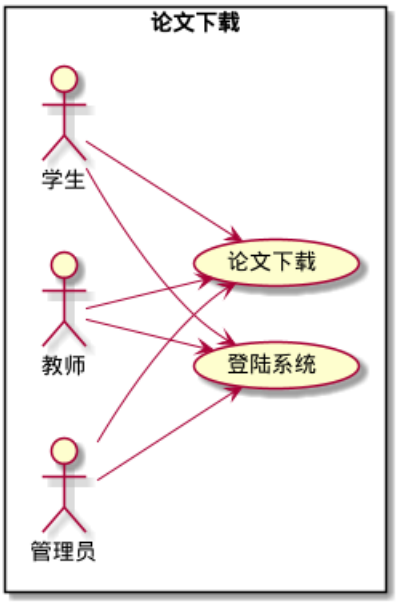

图 3.5　论文下载用例图

3.3.5　论文评审评分

如图3.6所示，教师可以从学生管理页面浏览论文列表，下载之后教师进行阅读和评审，如果评审未通过，教师可以选择论文进行评论，讲解论文出现的问题，同时可以为论文添加问题标签，便于学生快速了解自己的问题所在并进行论文的修改，学生在修改论文之后，重新上传自己的论文，教师重新阅读，达到评审标准之后，教师可以为论文评分，学生可以登陆系统查看论文得分。

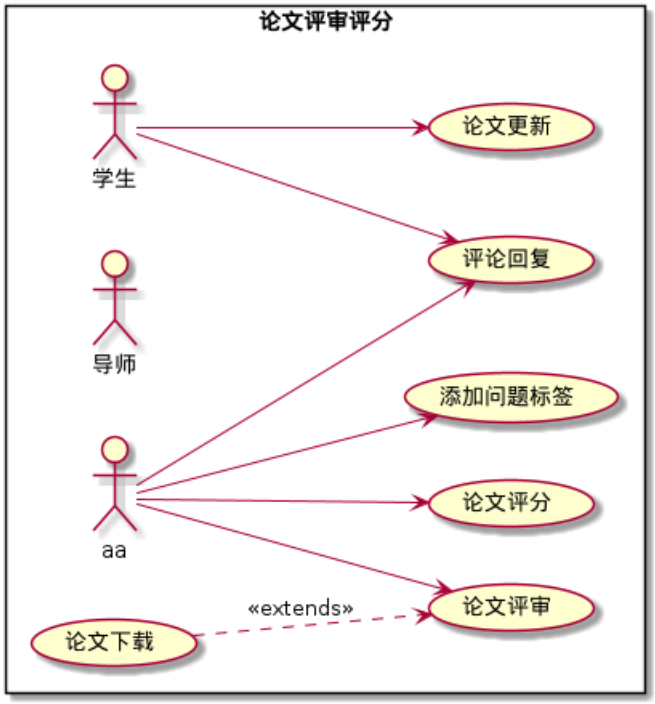

图 3.6　论文评审评分用例图

3.3.6　统计评审情况

如图3.7所示，管理员可以登陆系统查看所有论文的得分统计（分类有 95 以上，90-95 分，80-90 分，70-80 分，60-70 分，60 分以下，待评分），管理员可以通过该比例把控论文评审进程。管理员还可以查看教师的评审工作统计（分类有总分配学生数量，总分类论文数量，已评分论文数量，尚未评审论文数量），管理员可以根据教师的任务完成情况进行之后的任务分配。

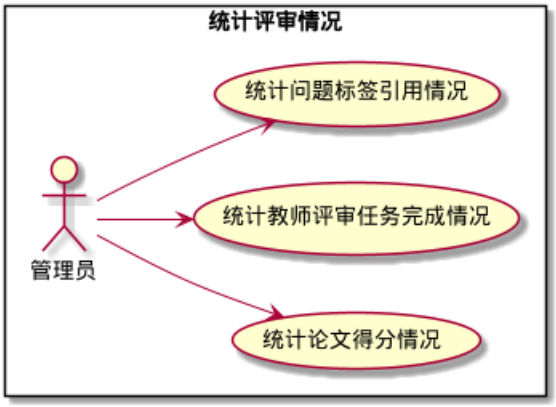

图 3.7　统计评审情况用例图

3.4 本章小结

需求分析是软件开发流程的第一步，同时也是十分关键的一步，如果前期的需求分析出现偏差，可能会导致最后实现的系统并不是想要的系统，需要全部推倒重来，代价十分巨大。所以，一份好的需求分析对于系统开发至关重要。

4 系统设计

4.1 系统架构设计

如图4.1所示，本系统采用的时前后端分离架构，系统前端使用 vue，Element Plus UI，axios，vuex，vue-router，系统后端使用了 Spring Boot，同时配置 Tomcat，Spring Security，Spring Data JPA，系统的持久化选择 Hibernate 和 MySQL。

前后端分离架构优势：

（1） 系统的可伸缩性。

由于将代码分成前端和后端两部分，可以不用考虑对方进行对立优化，只要保证数据接口不改变，那么双方的交互就不会出现问题。

（2） 减少资源传递。

在以往的非前后端分离项目中，服务器要做的工作有解析请求，从数据库中检索相关数据并处理数据，然后结合 HTML 模板并将其发送给浏览器。而在前后端分离的系统中，服务器并不需要处理 HTML，之需要将处理好的数据作为 response（一般是 JSON 对象）返还给前端，而前端根据数据进行展示，这个过程减少了服务器的任务，因此降低了服务器的压力，并且用户体验更好，因为整个页面不会随着每个请求而更新。

（3） 更容易升级。

升级架构是一件痛苦的事情。前后端分类可以更好地进行升级和调试，通过查看接口地返回数据地正确与否就可以判断是前端的问题还是后端的问题，这在系统升级的过程中可以加速问题的排查。

（4） 更容易切换框架。

现如今技术的的变化很快，不仅现在使用的框架更新频率很高，并且新的更好的框架也层出不穷。在使用前后端分离架构的情况下，前后端并不关心对方使用的是什么框架，只需要规定好接口并且保证在切换框架的过程同接口不会改变。比如前端从 Vue 转变为 Angular，这个变化后端并不会有感知。

（5） 更快地部署。

由于前端开发人员和后端开发人员并不需要太多地相互依赖（只有在定义接口地时候需要双方协商），所以前后端可以独立进行测试和部署而无需等待对方任务的完成。

（6） 整合的 API。

当前端需要多平台部署时，例如 Web 端，移动端，在开发应用的过程中，不需

要为各自的编写不同的后台，两者可以使用相同的后台，即使用相同的 API，这无疑减小了不必要的、重复的代码编写。而且在使用相同的 API 的情况下可以保证各个平台的数据的一致性，以及后台升级时，各个平台同时获得更新。

（7） 模块化。

如果您的前端与后端分离，那么在一个模块上工作而不影响另一个模块就变得更容易了。模块化还使两个团队或两个人同时在前端和后端工作变得更容易，而不必担心覆盖或搞乱其他人的工作。他们也可以用不同的节奏工作。

4.2 系统整体模块设计

系统整体模块设计如图4.2所示，总体可以分为学生模块，教师模块，管理员模块。在学生模块中可以进一步将功能细分为论文上传功能，论文下载功能，成绩查看功能，回复评论功能，论文更新功能。教师模块可以进一步分为查看所管理的学生论文信息功能，论文下载功能，论文评论功能，论文添加问题标签功能，论文评分功能。管理模块可以进一步分为系统管理和论文管理，系统管理可以进一步分为用户管理，上传通道管理，论文管理可以进一步分为评审情况统计，教师评审情况统计。

4.2.1 文件上传模块

如图4.3所示，首先用户点击上传文件按钮，选择文件后前端会发起上传请求，请求体中会包含二进制文件以及文件信息，Controller 接收到请求之后，从请求体中取出文件，并获取文件名，文件类型，文件数据，通过 Spring Security 的机制可以获取到用户信息，这样一个文件实体所需要的所有信息都可以获得（文件名，文件类型，文件数据，文件所属学生 id），将创建的文件实体传递给 Service，Service 调用文件 Repository 将文件实体持久化。该流程完毕之后，数据库文件表会增加一条数据。

4.2.2 文件下载模块

如图4.4所示，首先用点击文件下载按钮，之后前端会发起下载请求，url 参数会有想要下载的文件 id，Controller 接收到请求之后，解析出需要获取的文件 id，将 id 传递给 Service，Service 使用文件 Repository 通过 id 查询数据库，并利用查询到的数据生成文件实体，将实体中的 data[] 即二进制文件数据返还给 Controller，Controller 将 response 返还给前端，前端解析出文件并下载。

4.2.3 文件更新模块

如图4.5所示，首先用户点击文件更新按钮，之后前端会发起更新文件请求，请求体中会带有新的二进制文件以及要更新的文件 id，Controller 接收到请求之后，解析出文件和文件 id，将两者传递给 Service，Service 首先会重数据库中取出原来的文件，然后使用

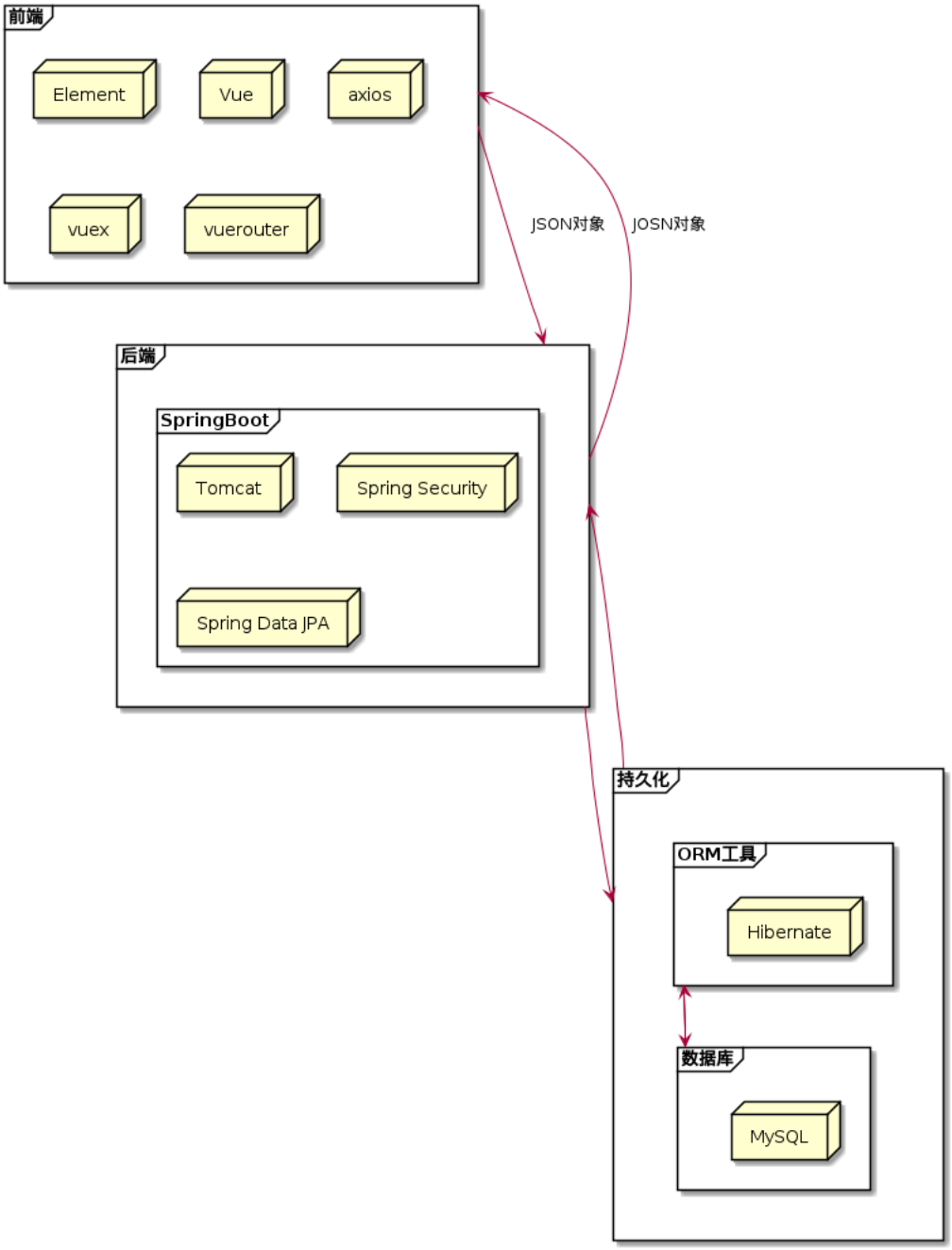

图 4.1　系统架构设计

新传来的文件信息更新就文件，最后再将更新后的文件使用文件 Repository 写回数据库。之后 Controller 返回表示成功的 response 给前端。

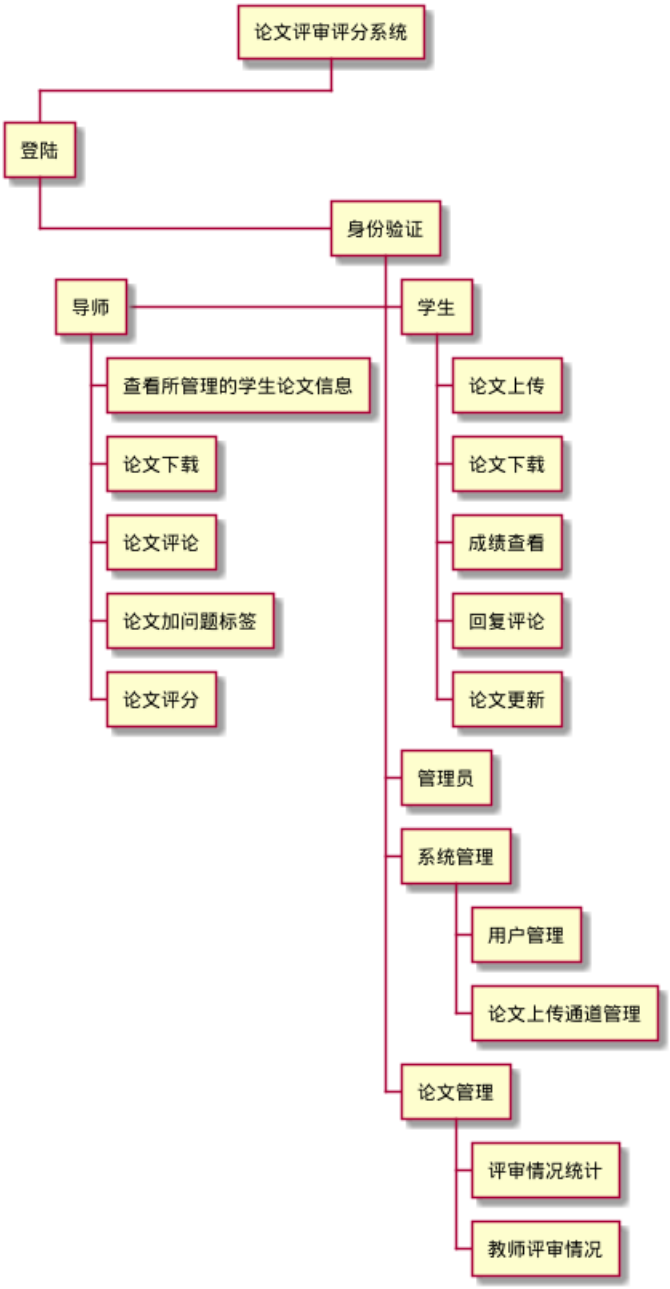

图 4.2　系统整体模块

4.2.4　文件评论模块

学生和教师都可以针对一个文件进行评论，用户双方的沟通，教师可以使用评论功能提出文章的问题，学生可以根据教师的回复进行对文章的更改，更改之后更新文章。如图4.6所示，用户首先选择要评论的文件并填入评论内容，前端发起请求，请求体中带

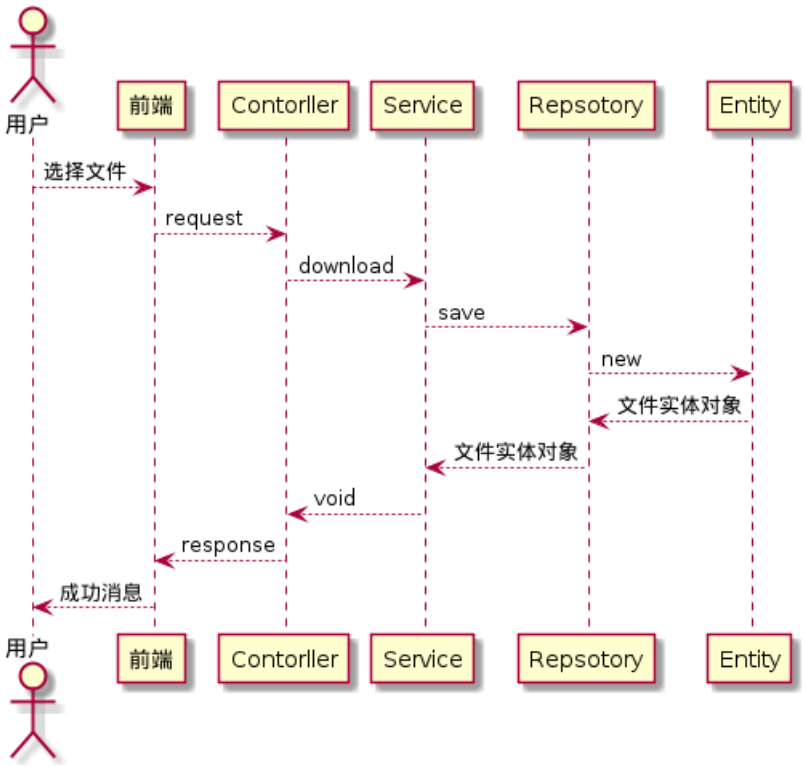

图 4.3　文件上传时序图

有文件 id，父级评论 id 和评论内容，Controller 接收到请求之后，将二者转交给 Service 处理，Service 需要创建一个评论实体，需要填入评论内容，为哪个文件评论（文件 id），由谁评论（从 principal 中可以取出发送请求的用户信息），回复那一条评论（父级评论 id），这些信息便可以确定唯一一条评论，之后使用 CommentRepository 保存，将信息持久化。

4.2.5　问题标签模块

在教师为文章评论时，教师可以为同时为评论添加问题标签，便于学生快速了解到文章的问题所在，标签也可以用于后序的论文问题统计。如图4.7所示，用户可以在评论上点击添加问题标签，当选择标签并且确定之后前端将发起一个请求，Controller 接收到请求之后，从请求体中解析出评论 id 和所有的标签 id 并将其传递给 Service 进行处理，Service 新建一个 TagComment 对象，填充评论属性和标签属性并使用 Repository 将数据保存到数据库。

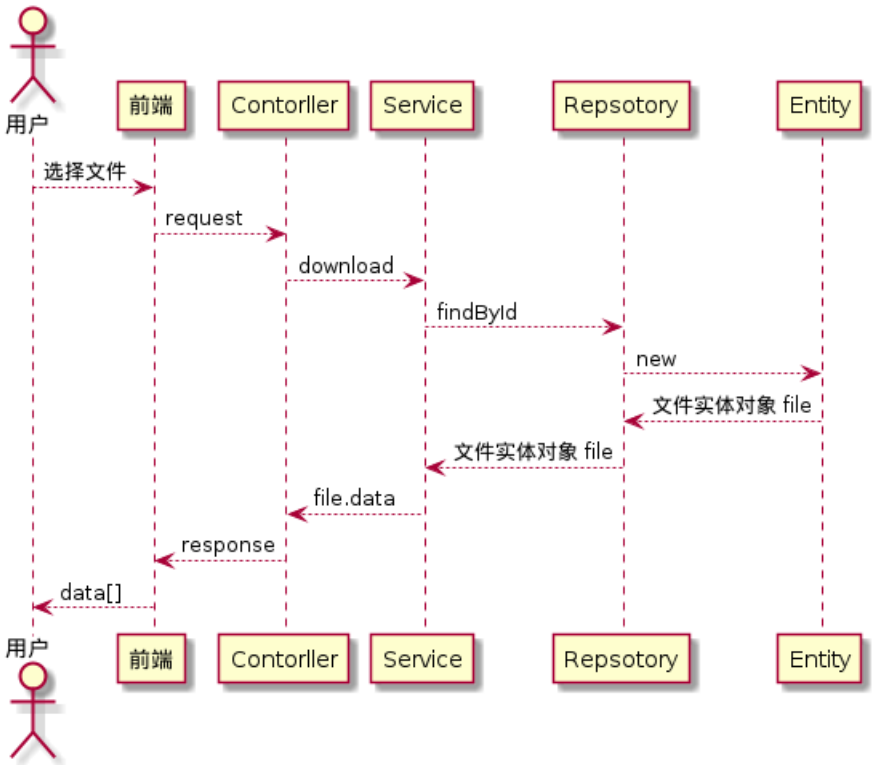

图 4.4　文件下载时序图

4.2.6　论文评分模块

当教师认为论文达到评审标准之后可以对论文进行评分。如图4.8所示，教师点击评分按钮弹出对话框，可以滑动对话框中的滑动条进行分数选择，点击确定按钮，前端将发起评分请求，Controller 接收到请求之后，可以解析出文件的 id 和分数，Controller 将两个参数传递给 Service，Service 调用 Repository 根据文件 id 查询数据库返回文件实体对象，Service 操作实体对象填入分数并调用 Repository 的方法将数据保存回数据库。

4.2.7　论文得分情况统计模块

如图4.9所示，管理员点击得分总览页面，前端向后台发起请求，Controller 接到请求之后请求 Service 服务，Service 从数据库取出所有论文信息并总动填充为文件实体的 list，之后根据文件的得分情况进行分类，分类后的数据从 Service 传回 Controller，之后 Controller 将返回输入放入 response 返还给前端，前端使用 Echarts 可视化展示给用户。

4.2.8　教师任务完成请求统计模块

如图4.10所示，管理员点击学生任务统计页面，前端向后台发起请求，Controller 接到请求 Service 服务，Service 从数据库取出所有的教师信息，其中与教师关联的学生属

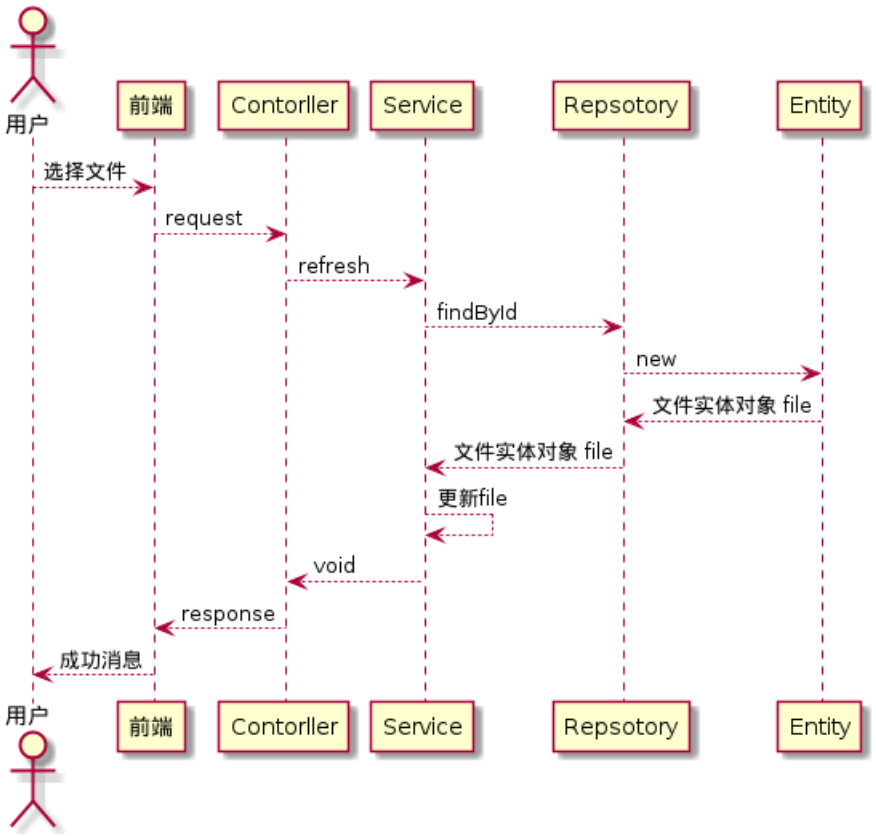

图 4.5　文件更新时序图

性会由 Hibernate 框架填充，同样的，学生的论文属性也会被填充，这样就可以获得所有与教师任务相关的信息，也就可以进行分离，Service 将分类后的数据返还给 Controller，Controller 将其放入 response 中发还给前端，前端使用 Echarts 进行进行可视化处理展示给用户。

4.2.9　学生管理模块

如图4.11所示，当管理员切换到学生管理页面，前端向后端发起请求，Controller 接收到请求之后调用 Service，Service 使用 Repository 查询出所有的学生信息并映射为学生实体，之后返还给 Controller，Controller 将其附加在 response 返还给前端进行展示。用户可以点击学生条目，进行修改，修改后点击确认，前端将向后端发起请求，Controller 接收到请求之后可以解析出学生实体，Controller 调用 Service，Service 将调用 Repository 将实体保存（因为该实体 id 属性不为空，所以会更新 id 相同的那一条学生信息）。

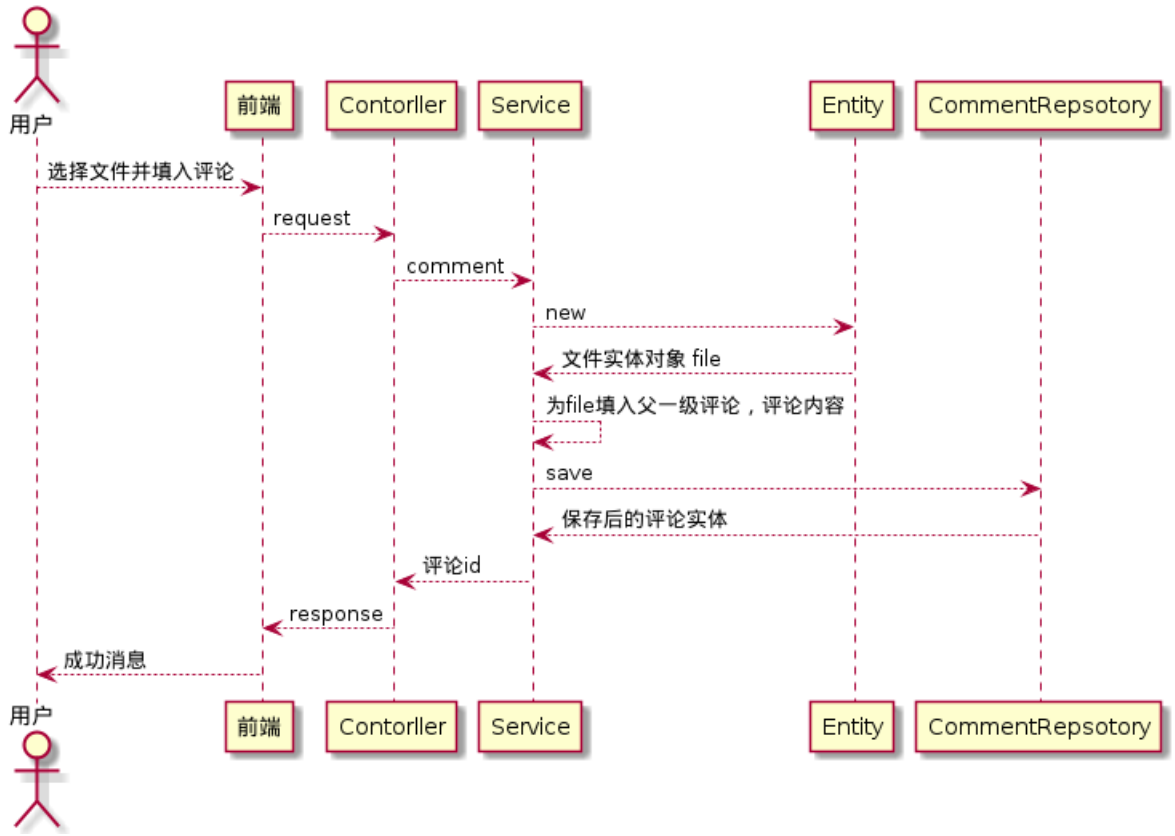

图 4.6　文件评论时序图

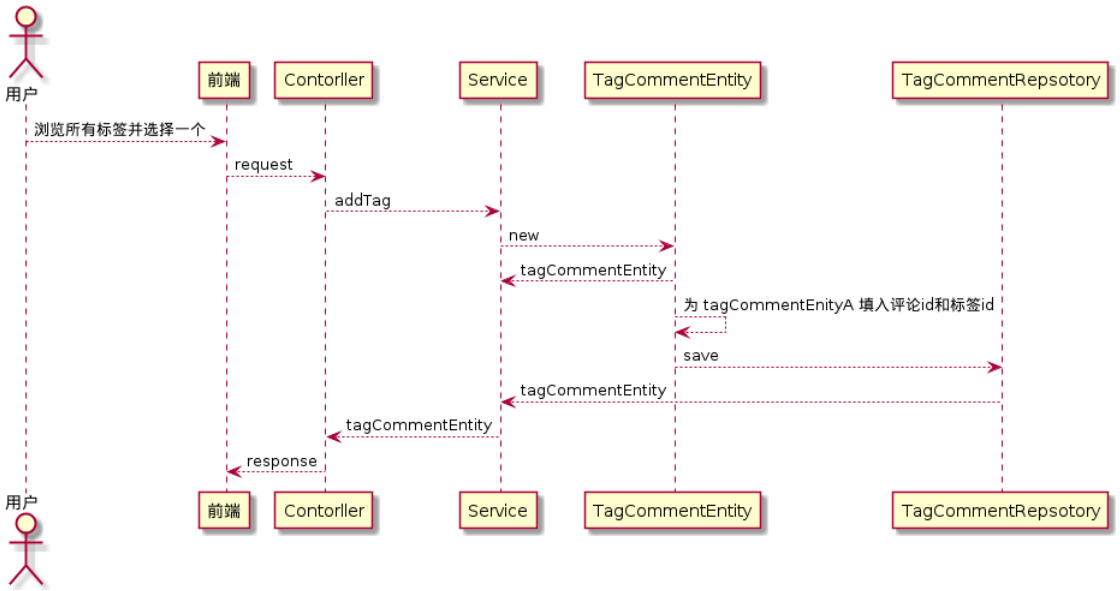

图 4.7　问题标签时序图

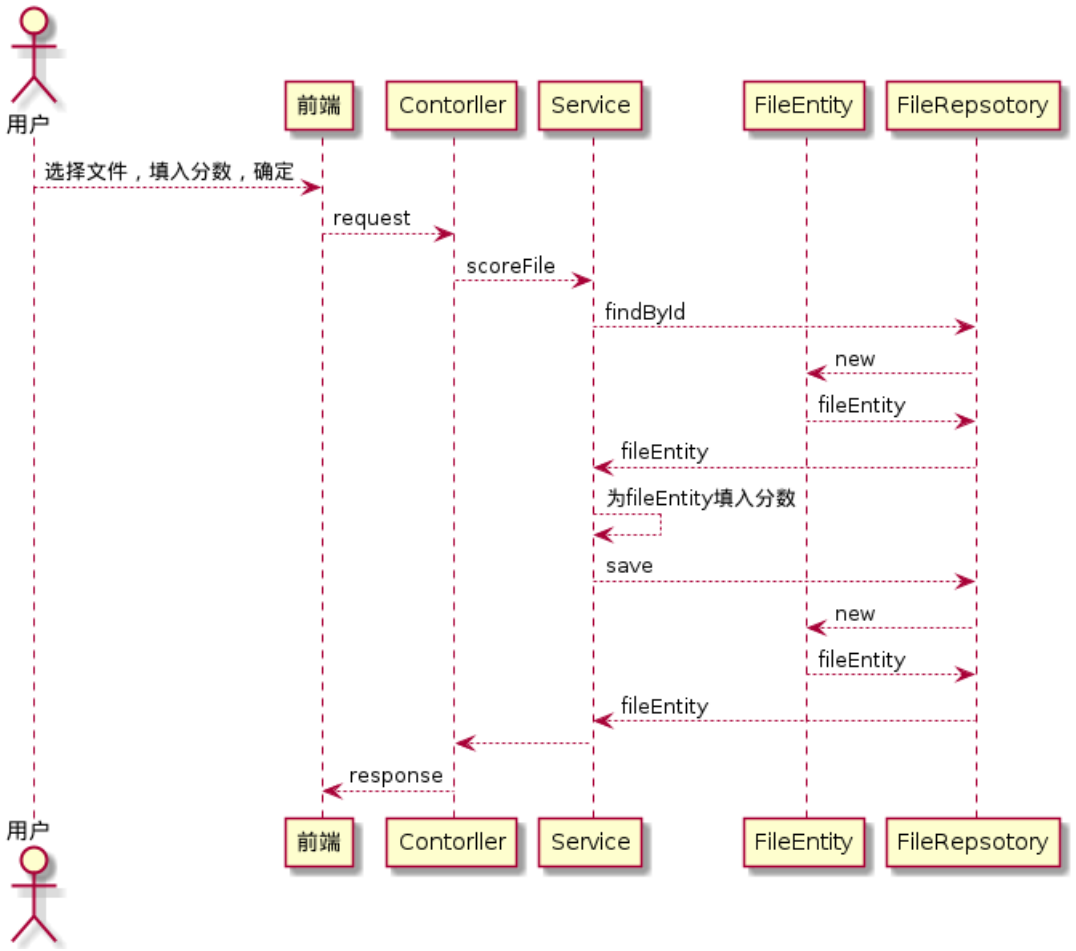

图 4.8　论文评分时序图

4.2.10　教师管理模块

如图4.12所示，当管理员切换到学生管理页面，前端想后端发起请求，Controller 收到请求之后使用 Service 获取所有教师信息，其中 Service 会调用 Repository 查询数据库并将教师信息映射为实体对象，Controller 将获得的所有教师信息放入 response 中返还给前端，用户可以点击教师条目，在对话框中修改教师信息，点击确认之后前端将发起修改请求，Controller 接收到请求之后可以从请求中解析出教师实体，之后使用 Repository 使用这个新的教师实体对象更新旧的教师数据。

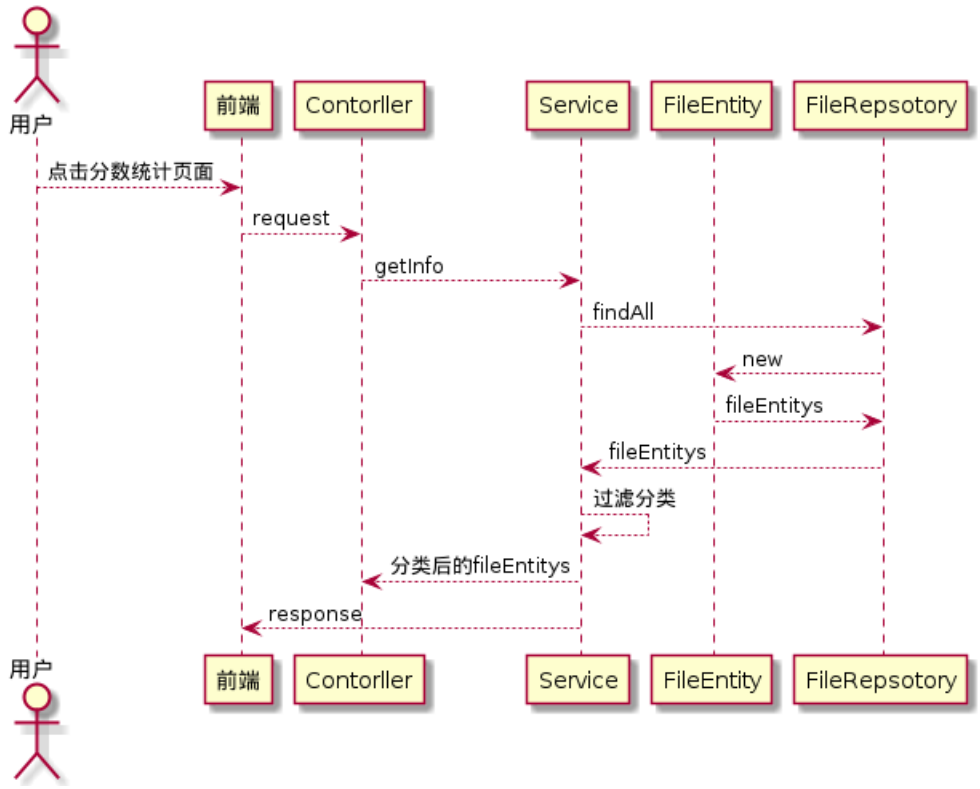

图 4.9　统计分数时序图

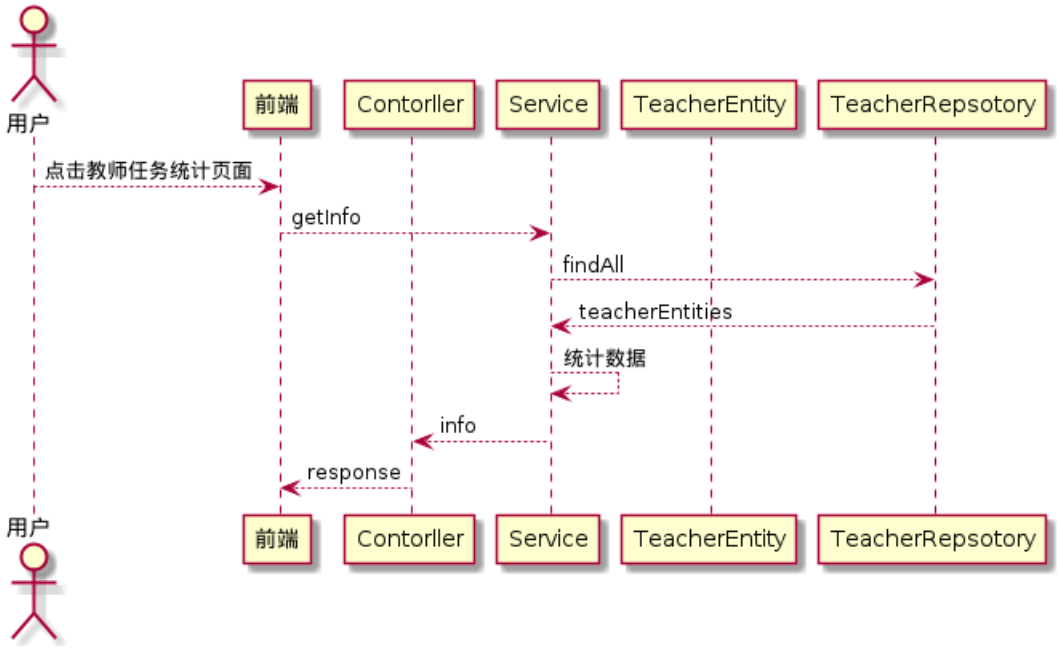

图 4.10　统计教师任务完成情况时序图

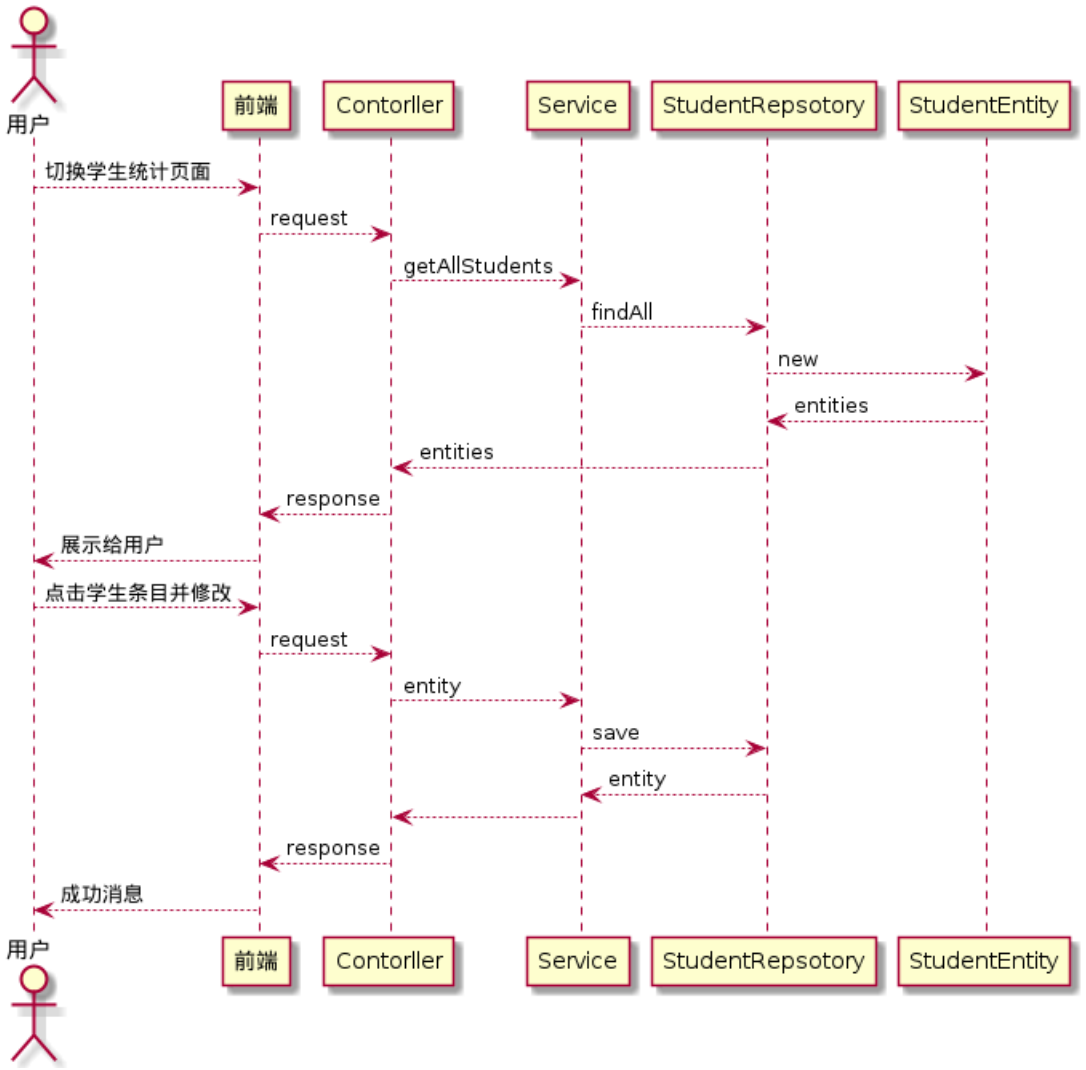

图 4.11　学生管理时序图

4.3　数据库设计

4.3.1　概念结构设计

如图4.13所示，本系统一共有三种用户，学生（student）、教师（teacher）以及管理员（admin），每一个教师拥有多个学生，每个学生分给一个教师。每一名学生拥有多篇论文（student_file），每一篇论文可以有多个评论（comment_for_file），每条评论可以添加多个问题标签（tag）, 同一个标签可以被多条论文引用，所以标签和评论之间时多对多的关系，因此需要一个连接表（tag_comment）。

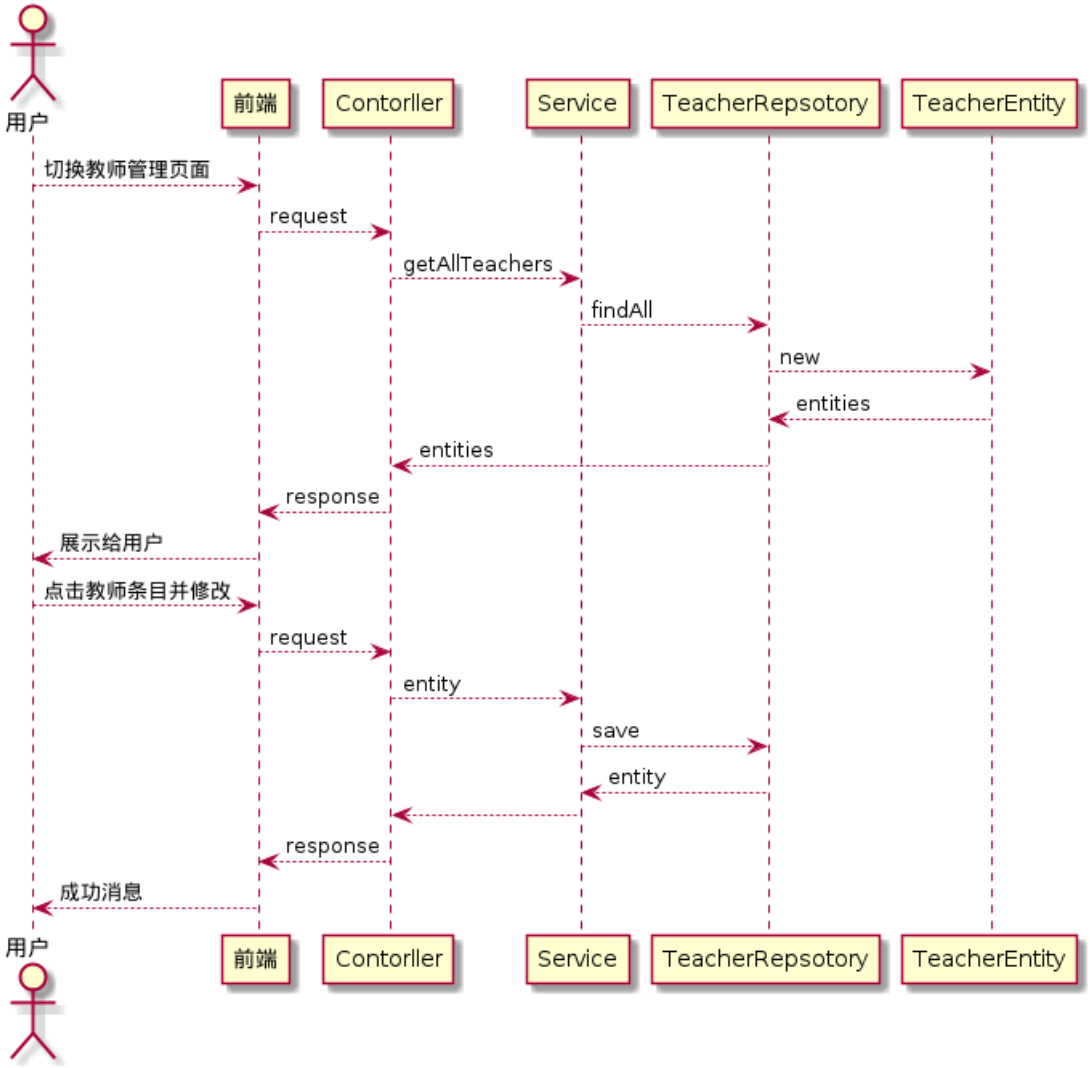

图 4.12　教师管理时序图

4.3.2　逻辑结构设计

如图4.14所示，

（1）　学生表

id 作为学生序号，该字段自增且为主键，不能为空。name 字段为学号，根据学校要求进行编码，又有学号可能有字母例如“student0001”等等用于学生类型分类，所以该字段选择使用 String 类型，password 字段为学生密码，由于密码可以使用字母和数字，所以这里选择 String 类型，teacher_id 字段表示该学生分配给哪一名教师管理，由于教师 id 字段为 Integer 类型，所以 teacher_id 字段也为 Integer 类型。

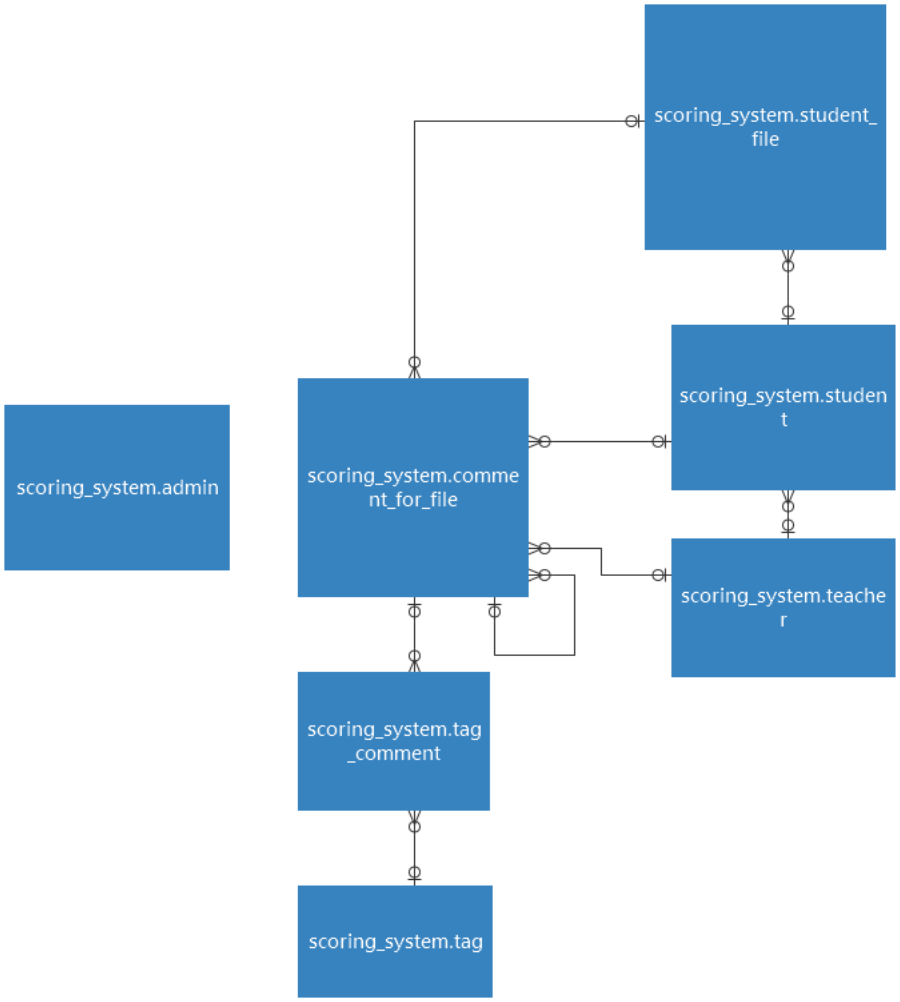

图 4.13　概念模型

（2）　教师表

共有三个字段，id，name 和 password，详细介绍类似学生表这三个字段。

（3）　管理员表

共有三个字段，id，name 和 password，详细介绍类似学生表这三个字段。

（4）　论文表

id 字段作为论文编号，该字段自增且为主键，不能为空，选择 Integer 类型；data 字段为 byte[] 类型，用于存放二进制文件数据；name 字段为文件名，为 String 类型；score 字段存放论文得分，选择 Integer 类型；type 字段用于存放文件类型，例如 pdf，word，选择 String 类型；student_id 字段指示该论文属于哪一名学生，因此该字段为外键，引用学生表的 id 字段；file_abstract 字段为

论文摘要，使用 String 类型。

（5） 评论表

id 字段作为评论编号，该字段自增且为主键，不能为空，选择 Integer 类型；comments 字段为评论的具体内容，选择 String 类型；parent_comemnt_id 字段指示该条评论是回复那一条评论的，该字段为外键应用评论表的 id 字段，所以为 Integer 类型；student_file_id 字段指示该评论是在那一篇论文下回复的，该字段为外键，引用论文表的 id 字段，因此为 Integer 类型；student_id 字段指示该条评论是哪一名学生回复的，该字段为外键，引用学生表的 id 字段，因此为 Integer 类型，如果这条评论是教师回复的，那么该字段的值为 null；teacher_id 字段指示该条评论是哪一名教师回复的，该字段为外键，引用教师表的 id 字段，因此为 Integer 类型，如果这条评论是学生回复的，那么该字段的值为 null。

（6） 标签表

id 字段作为标签编号，该字段自增且为主键，不能为空，选择 Integer 类型；name 字段为标签名字，使用 String 类型。

（7） 评论标签表

该表用于记录某一条评论引用了哪一个标签，为多对多。id 字段作为引用记录的 id，该字段自增且为主键，不能为空，使用 Integer 类型；comment_id 指示为哪一条评论添加标签，该字段为外键，引用评论表的主键 id 字段；tag_id 字段指示为这条评论添加哪一个标签，该字段为外键，引用标签表的主键 id 字段。

4.3.3 物理结构设计

如图4.14所示，

（1） 学生表

如图4.1所示，学生表 id 字段使用 int 类型；学生表 name（编号）字段使用 varchar(255) 类型；学生表 password（密码）字段使用 varchar(255) 类型；学生表 teacher_id（所属教师 id）字段使用 int 类型；

（2） 教师表

如图4.2所示，共有三个字段，id，name 和 password，详细介绍类似学生表这三个字段。

（3） 管理员表

如图4.3所示，共有三个字段，id，name 和 password，详细介绍类似学生表这三个字段。

（4） 论文表

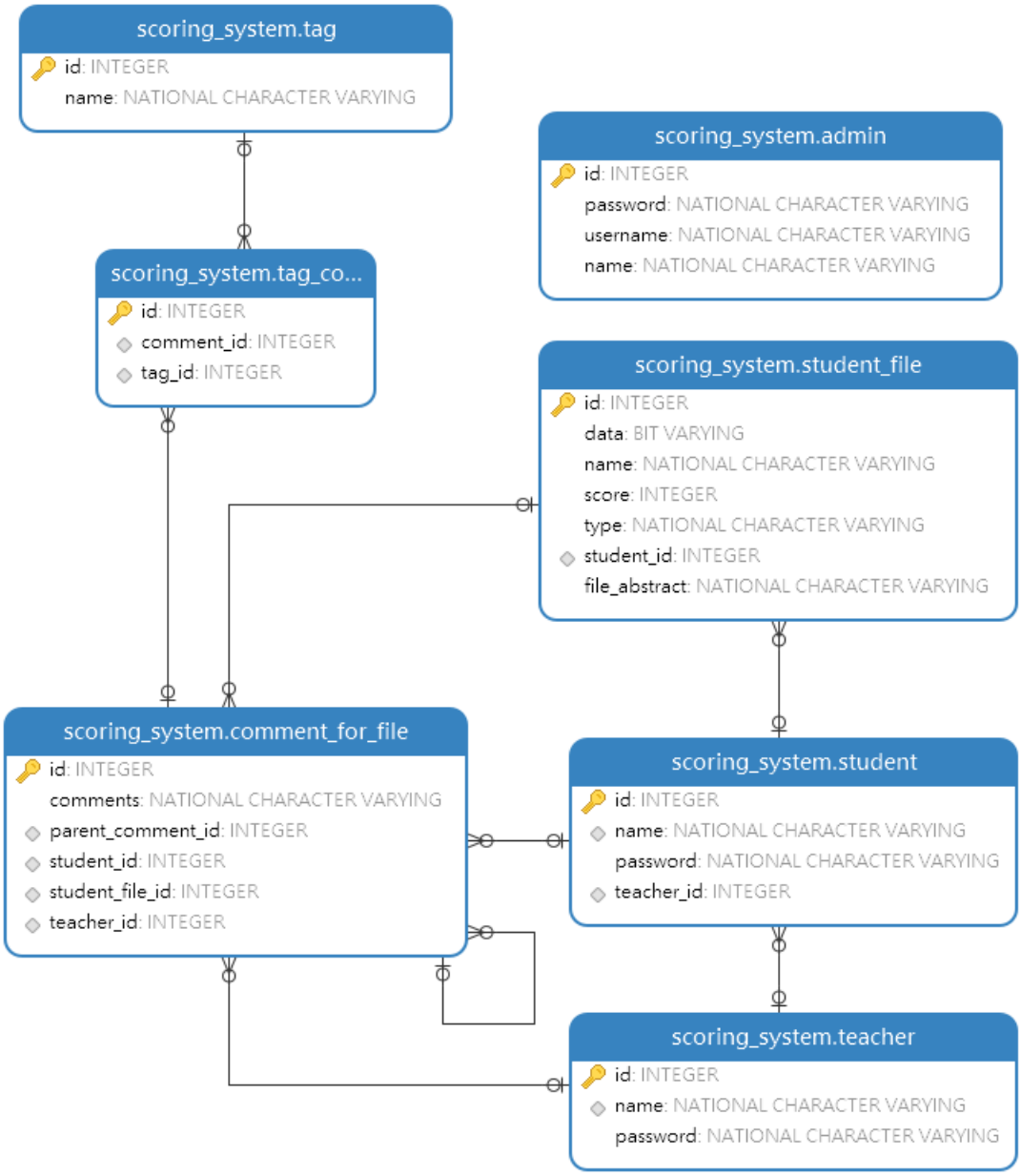

图 4.14　逻辑模型

如图4.4所示，论文表 id（论文编号）字段使用 int 类型；论文表 data（论文二进制数据）字段使用 longblob 类型；论文表 name（论文题目）字段使用 varchar(255) 类型；论文表 score（论文得分）字段使用 int 类型；论文表 type（论文文件的类型）字段使用 varchar(255) 类型；论文表 student_id（论文所属学生编号）字段使用 int 类型；论文表 file_abstract（论文摘要内容）字段使用 varchar(255) 类型。

表 4.1　学生表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
id	int	int	null	NO	null	学生 id
name	varchar(255)	varchar	255	YES	null	学生姓名
password	varchar(255)	varchar	255	YES	null	学生密码
teacher_id	int	int	null	YES	null	教师 id

表 4.2　教师表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
id	int	int	null	NO	null	教师 id
name	varchar(255)	varchar	255	YES	null	教师编号
password	varchar(255)	varchar	255	YES	null	教师密码

（5） 评论表

如图4.5所示，评论表 id（评论编号）字段使用 int 类型；评论表 comments（评论内容字段）使用 varchar(255) 类型；评论表 parent_comment_id（指明该评论是回复哪一条评论的）使用 int 类型；评论表 student_id（指明是哪一名学生写的此条评论）使用 int 类型；评论表 teacher_id（指明这条评论是哪一名教师写的）使用 int 类型；评论表 student_file_id（指明该条评论是回复那一条论文的）。

（6） 标签表

如图4.6所示，标签表 id（标签编号）字段使用 int 类型；标签表 name（标签名字）字段使用 varchar(255) 类型。

（7） 评论标签表

表 4.3　管理员表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
id	int	int	null	NO	null	管理员 id
name	varchar(255)	varchar	255	YES	null	管理员名称
password	varchar(255)	varchar	255	YES	null	管理员密码

表 4.4　论文表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
data	longblob	longblob	4.29E+09	YES	null	文章数据
file_abstract	varchar(255)	varchar	255	YES	null	文章摘要
id	int	int	null	NO	null	文章 id
name	varchar(255)	varchar	255	YES	null	文章名
score	int	int	null	YES	null	文章得分
student_id	int	int	null	YES	null	文章所属的学生 id
type	varchar(255)	varchar	255	YES	null	文件类型

表 4.5　评论表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
comments	varchar(255)	varchar	255	YES	null	评论内容
id	int	int	null	NO	null	评论 id
parent_comment_id	int	int	null	YES	null	父级评论
student_file_id	int	int	null	YES	null	论文 id
student_id	int	int	null	YES	null	学生 id
teacher_id	int	int	null	YES	null	教师 id

如图4.7所示，评论标签表 id（添加标签记录编号）字段使用 int 类型；标签评论表 comment_id（指明将标签添加到哪一条评论上）使用 int 类型；标签评论表 tag_id（指明将哪一个 tag 添加到评论上）使用 int 类型。

表 4.6　标签表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
id	int	int	null	NO	null	问题标签 id
name	varchar(255)	varchar	255	YES	null	问题标签名

表 4.7　标签评论表

列名	数据类型	字段类型	长度	是否为空	默认值	备注
comment_id	int	int	null	YES	null	所属评论 id
id	int	int	null	NO	null	tag 使用记录 id
tag_id	int	int	null	YES	null	使用的问题标签的 id

4.4　本章小结

本章的主要内容是系统设计，首先对系统的整体加架构进行了简单介绍并且谈论了选择该架构的原因，其次介绍了系统各个模块的功能以及工作流程，然后详细介绍了论文评审评分系统的数据库设计。

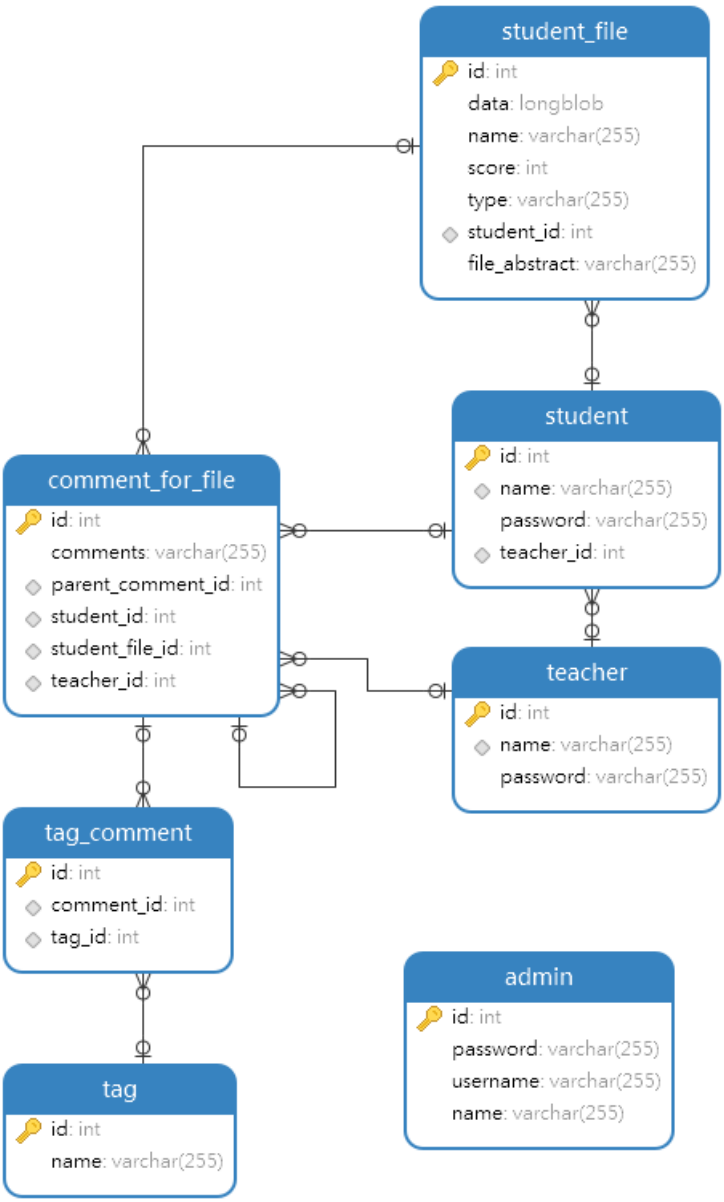
student_file
id: int
data: longblob
name: varchar(255)
score: int
type: varchar(255)
student_id: int
file_abstract: varchar(255)
student
id: int
name: varchar(255)
password: varchar(255)
teacher_id: int
comment_for_file
id: int
comments: varchar(255)
parent_comment_id: int
student_id: int
student_file_id: int
teacher_id: int
teacher
id: int
name: varchar(255)
password: varchar(255)
tag_comment
id: int
comment_id: int
tag_id: int
admin
id: int
password: varchar(255)
username: varchar(255)
name: varchar(255)
tag
id: int
name: varchar(255)

图 4.15　物理模型

5 实现与测试

5.1 系统环境

5.1.1 硬件环境

（1） CPU：Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz
（2） 内存：16.0 GB
（3） 硬盘容量：500GB

5.1.2 软件环境

（1） 操作系统：Windows 10 家庭中文版（20H2）作为编辑环境 + WSL2（Ubuntu 20.04）作为开发环境
（2） 数据库：mysql Ver 8.0.23-0ubuntu0.20.04.1 for Linux on x86_64 (Ubuntu)
（3） 客户端：主流浏览器

5.2 持久化

5.2.1 Spring Date & JPA Hibernate

（1） 使用“逻辑模型工具”（例如 Navicat Data Module 3）建立模型。
（2） 将模型同步至数据库
（3） 使用 IDEA 持久化工具生成实体类
（4） 根据逻辑模型为实体类添加注解（实现级联关系）

关键问题如下：

（1） 多对一，一对多关系实现
这种关系一共有四种实现方式

（1） 单向 `@OneToMany` 绑定
（2） 带有 `@JoinColumn` 的单向 `@OneToMany` 绑定
（3） 双向 `@OneToMany` ，`@ManyToOne`绑定
（4） 不带有 `ManyToOne` 的双向绑定

本系统采用第三种方式

```
public class A { // 一方
    ...
    @OneToMany(...)
    private List<B> bs;
```

```
    ...
}
public class B { // 多方
    @ManyToOne
    @JoinColumn(name = "a_id", referencedColumnName = "id")
    private A a;
}
```

（2） 级联更新

（1） 在“一方”（A）中 `@OneToMany(cascade = CascadeType.ALL)`
（2） ”深拷贝”A 中的 `BCopy = List<B>`
（3） 清空 A 中的 `List<B>`
（4） `BRepository.deleteAll(BCopy)`
（5） `ARepository.save(A)`

（3） 级联查询

（1） Dao 层：继承 JpaSpecificationExecutor

```
public interface ARepository extends
    JpaRepository<A, Integer>,
    JpaSpecificationExecutor<AEntity> // 注意这里
```

（2） Service 层使用：

```
aRepository.findAll(new Specification<A>() {
    @Override
    // Hibernate根据此方法生成过滤语句
    public Predicate toPredicate(
        Root<UnfinishedPlanEntity> root,
        CriteriaQuery<?> query,
        CriteriaBuilder criteriaBuilder) {
      // 实现逻辑
    }
})
```

涉及的重要的类：

（1） `javax.persistence.criteria.Path`：通过实体类属性名获取字段值
（2） `javax.persistence.criteria.Join`：通过实体类属性名级联

（3） `javax.persistence.criteria.CriteriaBuilder`：拼接生成过滤

5.3 统一返回对象

使用统一的返回格式不仅可以让接口更漂亮，也可以让前端更方便地处理数据。为了实现自动格式化，需要：

（1） 接口：`org.springframework.web.servlet.mvc.method.annotation.MappingJackson2HttpMessageConverter`

允许在执行 @ResponseBody 或 ResponseEntity 控制器方法后，但在使用 HttpMessageConverter 编写响应体之前定制响应。

这正是需要处理返回对象的时机。

实现可以直接注册到 RequestMappingHandlerAdapter 和 ExceptionHandlerExceptionResolver，或者更有可能用 @ControllerAdvice 注释，在这种情况下，它们将被两者自动检测到。

本系统的实现选择第二种方式：使用 @ControllerAdvice 注解。

（2） 注解：`org.springframework.web.bind.annotation.RestControllerAdvice`

这是一个方便的注释，它本身由 @ControllerAdvice 和 @ResponseBody 注释组成。携带该注释的类型被视为 controller advice，其中 @ExceptionHandler 方法默认使用 @ResponseBody 语义。

```
@RestControllerAdvice
public class CustomResponseBodyAdvice implements ResponseBodyAdvice<Object> {

    @Override
    public boolean supports(MethodParameter arg0, Class<? extends
                                        HttpMessageConverter<?>> arg1) {
        return true;
    }

    @Override
    public Object beforeBodyWrite(Object arg0, MethodParameter arg1,
            MediaType arg2, Class<? extends HttpMessageConverter<?>> arg3,
            ServerHttpRequest arg4, ServerHttpResponse arg5) {
        if (arg0 == null) {
            return UnifiedResponsor.ofFail(CommonStatusCodeEnum.SUCCESS_COMMON);
```

```
        } else if (arg0 instanceof UnifiedResponsor) {
            return arg0;
        } else if (arg0 instanceof byte[]) {
            return arg0;
        }
        else {
            return UnifiedResponsor.ofSuccess(
                            CommonStatusCodeEnum.SUCCESS_COMMON, arg0);
        }
    }
}
```

@ControllerAdvice 是 Spring 提供的 AOP 特性之一，和普通情况下的不同在于 @ControllerAdvice 是由 Sping MVC 框架织入并提供 Web 特性。

AOP（面向切面编程）可以使我们在一个地方定义通用功能，但是可以通过声明的方式定义这个功能要以何种方式在何处应用，而无需修改受影响的类[12]。AOP 涉及六个概念

（1） 通知（Advice）：即切面的工作，定义了切面的“什么”和“何时”。

（2） 连接点（Join point）：应用程序中可以插入切面的一个个时机。

（3） 切点（Pointcut）：定义了切面在”何处“工作，匹配 advice 要织入的连接点。

（4） 切面（Aspect）：为通知和切点的结合。

（5） 引入（Introduction）：向现有的类添加新方法或属性。

（6） 织入（Weaving）：切面应用到目标对象同时生成代理对象。

需要注意的一点是，如果返回对象为 String 类型时会出现错误：

java.lang.ClassCastException : your response class can not cast to java.lang.String

该问题出现的原因是 String 类型在 ResponseAdvice.beforeBodyWrite() 方法中已经被处理为 UnifiedResponsor 对象，当控制器请求方法的返回类型是 String 时，用于转换返回值的 HttpMessageConverter 实例是 StringHttpMessageConverter。

解决方式如图5.1所示：告诉 spring 框架，不要使用 StringHttpMessageConverter 来处理控制器方法中的 String 返回类型，需要在 StringHttpMessageConverter 前面添加一个 MappingJackson2HttpMessageConverter，即代码的第 16 行。

在运行到 15 行时（即解决问题之前的情况），处理控制器方法的返回值时使用的转换器序列如图5.2所示，第二个即为 StringHttpMessageConverter。

在写入和刷新响应之前，应该计算 ContentLength(当添加 http 头时)。StringHttpMessageConverter.getContentLength 的实现是这样的:

```
// org.springframework.http.converter.StringHttpMessageConverter
@Override
protected Long getContentLength(
                    String str,
                    @Nullable MediaType contentType) {
  Charset charset = getContentTypeCharset(contentType);
  return (long) str.getBytes(charset).length;
}
```

函数的第一个参数为 String 类型，但是返回值已经在 ResponseAdvice.beforeBodyWrite() 方法中已经被处理为 UnifiedResponsor 对象，造成参数类型不匹配，抛出 ClassCastException。

图 5.1　WebMvcConfigurer

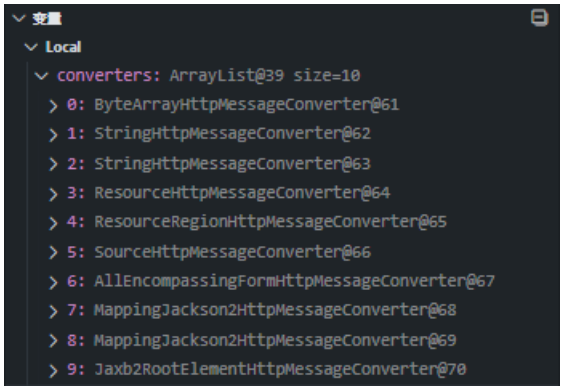

图 5.2　MessageConverter 序列

之后运行到 16 行，添加 MappingJackson2HttpMessageConverter，添加后如图5.3所

示，该类继承 AbstractJackson2HttpMessageConverter，其中的 getContentLength() 方法的实现为：

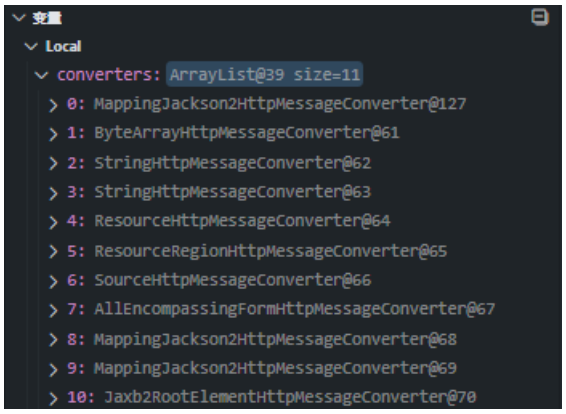

图 5.3　处理后的 MessageConverter 序列

```
//org.springframework.http.converter.json.AbstractJackson2HttpMessageConverter;
 @Override
 protected Long getContentLength(
                 Object object,
                 @Nullable MediaType contentType) throws IOException {
   if (object instanceof MappingJacksonValue) {
     object = ((MappingJacksonValue) object).getValue();
   }
   return super.getContentLength(object, contentType);
 }
```

这里第一个参数类型为 Object，所以并不会出现异常。

5.4　登陆 & 验证授权模块

5.4.1　JWT 优势

随着 Web 应用规模的逐渐扩大，传统的基于 Session 和 cookie 的身份验证技术逐渐显现出它的弊端。随着服务器的不断增加，由于多个请求可以被分发给不同的服务器，那么此时，服务器该如何那些请求来自同一个用户，对此有两种解决方式

（1） 多个服务器之间同步用户状态，这样无论用户的请求分发到哪一个服务器，都可以操作用户状态

（2） 判断来自同一个用户的请求，将同一个用户的请求一直分发到同一个服务器

很显然，无论是上述哪种方式，随着用户量的增长开销都会变得很大。例如第一种方式可以使用会话复制实现，Sesson 复制性能也会随着服务器的增加而急剧下降。[13]

JWT(JSON Web Token) 的优势在于

（1） 将用户认证信息保存在客户端, 减轻服务器的存储压力。

（2） 提供无状态的身份认证——不需要服务器端多端同步用户状态，利于分布式应用开发。[13]

5.4.2 使用 JWT&Spring Security 实现验证和授权

关键问题如下：

（1） 后端配置多用户表使用 Spring Security

验证授权涉及到的关键类（图5.5）：

（1） `Authentication` ：用户认证信息。

（2） `UsernamePasswordAuthenticationToken` ： `Authentication` 的实现类，用于登陆验证，最为常用。

（3） `AuthenticationProvider` :认证器,不同的 `AuthenticationProvider` 处理不同的 `Authentication`

（4） `DaoAuthenticationProvider` ： `AuthenticationProvider` 的实现类，用于处理 `UsernamePasswordAuthenticationToken`

（5） `ProviderManager` ：用于管理 `AuthenticationProvider`

（6） `UserDetailService` ： 为 `DaoAuthenticationProvider` 提供从数据库查询用户的功能，用于做密码对比。

多用户表配置 Spring Security（图5.6）：

（1） 为不同用户建立各自的 `UserDetailService`

```
public class XXXUserDetialsService
          implements UserDetailService
```

，各自的 `UserDetailService` 从各自的数据库表中查询密码，提供给各自的 `DaoAuthenticationProvider` 使用。

（2） 为不同用户建立各自的 `UsernamePasswordAuthenticationToken`

```
public class XXXUPAuthToken
          extends UsernamePasswordAuthenticationToken
```

（3） 为不同用户建立各自的 `DaoAuthenticationProvider`

```
public class XXXDaoAuthProvider
          extends DaoAuthenticationProvider
```

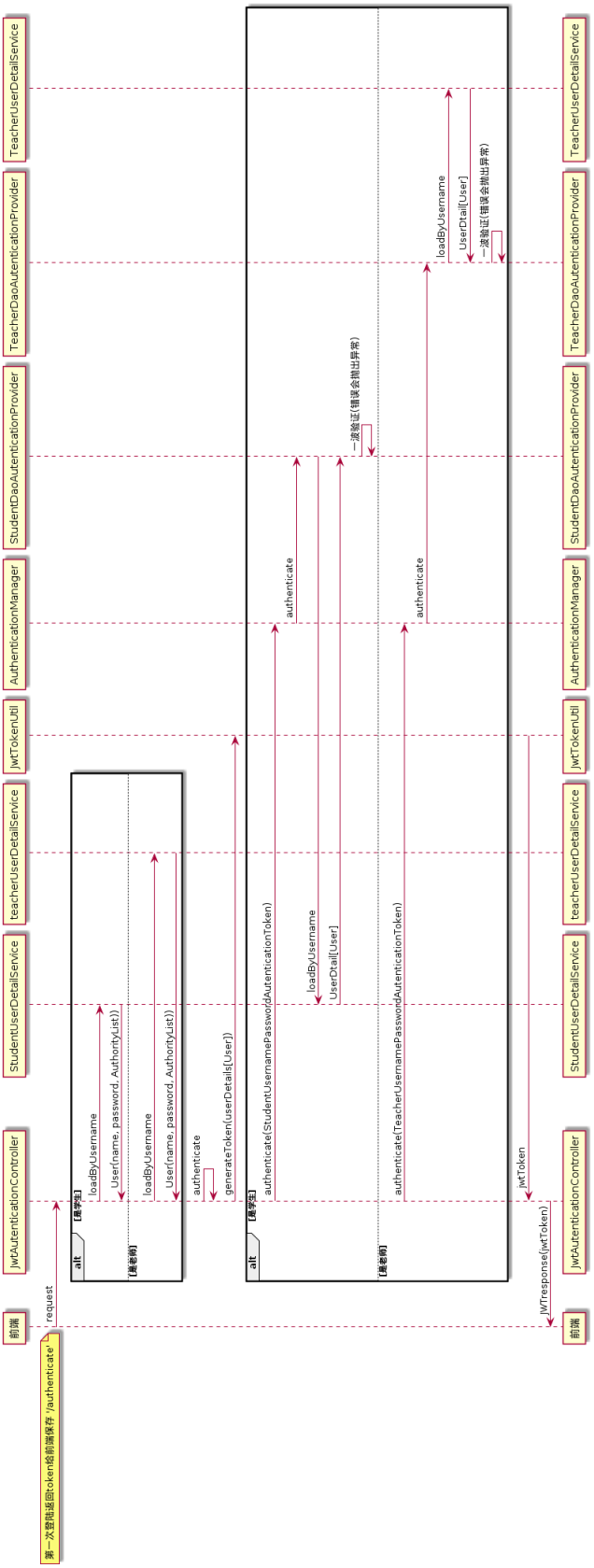

图 5.4　登陆验证授权时序图

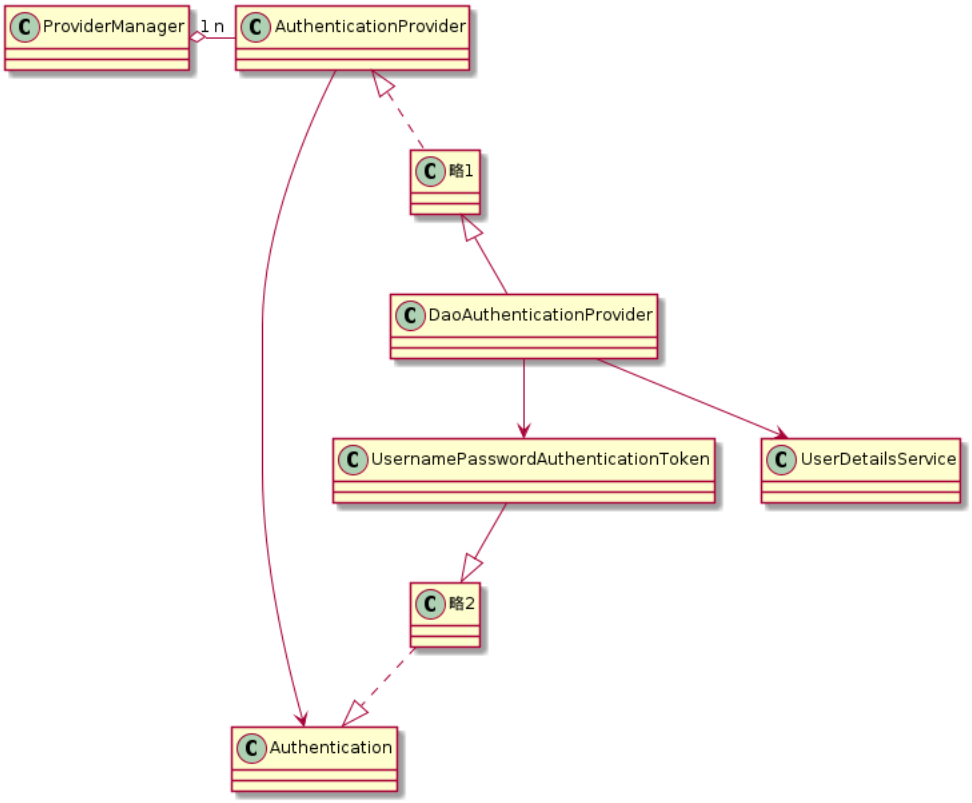

图 5.5　Spring Security 验证授权关键类

（4）　进行 Security 配置

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MySecurityConfig
          extends WebSecurityConfigurerAdapter
```

需要的关键配置：

（1）　注入 XXXAuthProvider
（2）　向 AuthenticationManager 添加 XXXAuthProvider
（3）　注入 AuthenticationManager

总体流程：

（1）　前端发起身份验证请求，由于尚未登陆，所以前端没有 Token 信息，因而无法通过 Spring Security 的验证，这需要在 MySecurityConfig 中配置——放过访问”/authentication” 的请求，以便进行身份验证和授权。

```
@Override
protected void configure(HttpSecurity httpSecurity)
                                throws Exception {
  httpSecurity.authorizeRequests()
      .antMatchers("/authenticate", "/register")
      .permitAll()// 身份验证和注册不需要拦截
      ...
}
```

（2） 请求到达 MyAuthenticationController，此时可以从请求中获取到登陆用户身份类型，如果是学生则需要使用 StudentUserDetailService 获取 UserDetails，如果是教师则需要使用 TeacherDetailsService 获取 UserDetails，同理，如果是管理员则需要使用 AdminDetailsService 获取 UserDetials。

（3） 此时进入到 XXXUserDetialsService 中，这里需要重写方法 loadUserByUsername(String username)，如果是为学生用户服务的，则需要根据传入的 username 从学生表中取出用户，如果是为教师服务的，则需要查询教师表并从中找到用户。如果没有找到用户则抛出异常，如果找的了则建立 org.springframework.security.core.userdetails.User（为 UserDetials 的实现类）其中包含用户名，用户密码，用户权限和用户身份。

（4） 当获得了 XXXUserDetialsService 返还的 UserDetials，则可以开始进行身份验证。该验证需要 AuthenticationProvider 来完成，因此需要传递需要验证的东西（Authentication）给管理 AuthenticationProvider 的 AuthenticationManager 进行身份验证。由于这里有不同种类的用户且不同种类用户需要查询不同的表，所以需要建立不同 AuthenticationProvider（使用不同的 XXXUserDetialsService（从不同的数据库查询用户））去处理不同的 Authentication，而用来处理用户名用户密码的 AuthenticationProvider 是 DaoAuthenticationProvider，用来处理用户名用户密码的 Authentication 是 UsernamePasswordAuthenticationToken，所以需要为不同用户建立不同的 XXXUsernamePasswordAuthenticationToken 和不同的 XXXAuthProvider。为了绑定 XXXAuthProvider 和 XXXUsernamePasswordAuthenticationToken 则需要重写 XXXAuthProvider 的 supports 方法。

```
// XXXDaoAuthProvider
@Override
public boolean supports(Class<?> authentication) {
    return XXXUsernamePasswordAutenticationToken
            .class
            .isAssignableFrom(authentication);
}
```

DaoAuthenticationProvider 是继承 AbstractUserDetailsAuthentication-Provider 的，不同在于 DaoAuthenticationProvider 增加了验证密码的功能，这也正是我们需要的，但是验证密码需要查询数据库也就需要 UserDetailService，所以需要通过 MySecurityConfig 为 XXXAuth-Provider 注入不同的 XXXUserDetialsService。这里选择 setter 注入：

```
// XXXDaoAuthProvider
@Override
public void setUserDetailsService(
    UserDetailsService userDetailsService) {
    super.setUserDetailsService(userDetailsService);
}
```

```
// MySecurityConfig
@Autowired
@Qualifier("XXXUserDetailsService")
private UserDetailsService XXXUserDetailsService;

// 注入
@Bean("XXXDaoAutenticationProvider")
DaoAuthenticationProvider daoXXXDaoAutenticationProvider()
{
  return new XXXDaoAutenticationProvider(
                          passwordEncoder(),
                          XXXUserDetailsService);
}
```

这个过程中涉及的类比较多且关系比较笔复杂，可以参考图5.6（经过简化，略去了与该流程相关程度不大的类）。

（5） 之后流程又回到了 MyAuthenticationController，此时如果通过了之前

的身份验证则可以为用户生成 Token 了，这里需要使用 JwTokenUtil 生成 Token，将之前获得的 userdetails 传递给 JwTokenUtil，需要注意的是要在 claims 中填入用户角色，这样前端拿到 Token 之后可以解析出用户角色（这部分信息对应图2.1Payload 部分）。

```
// 后端生成Token
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    Collection<? extends GrantedAuthority> it =
                    userDetails.getAuthorities();

    List<String> roles = new ArrayList<String>();
    for (GrantedAuthority grantedAuthority : it) {
        System.out.println(grantedAuthority.toString());
        if (grantedAuthority.toString()
                    .startsWith("ROLE_")) {
            roles.add(
                grantedAuthority.toString().substring(5));
        }
    }

    claims.put("userType", roles);
        // 在 JWT payload 中放入 自定义的角色信息

    return doGenerateToken(
            claims, userDetails.getUsername());
}
```

```
// 前端解析Token
export function jwtDecrypt(token) {
  var base64Url = token.split('.')[1];
  var base64 = base64Url.replace(/-/g, '+')
                      .replace(/_/g, '/');
  var jsonPayload = decodeURIComponent(
    atob(base64)
    .split('')
      .map(function (c) {
        return '%' + ('00' +
```

```
            c.charCodeAt(0).toString(16)).slice(-2);
        })
        .join('')
    );
    return JSON.parse(jsonPayload);
}
```

（6） 当前端接收到后端的 response 之后，则可以从返还的 Token 中获取用户信息存储到 localStore。之后的每个请求也都需要在 Header 中放入 Token 用于身份验证。

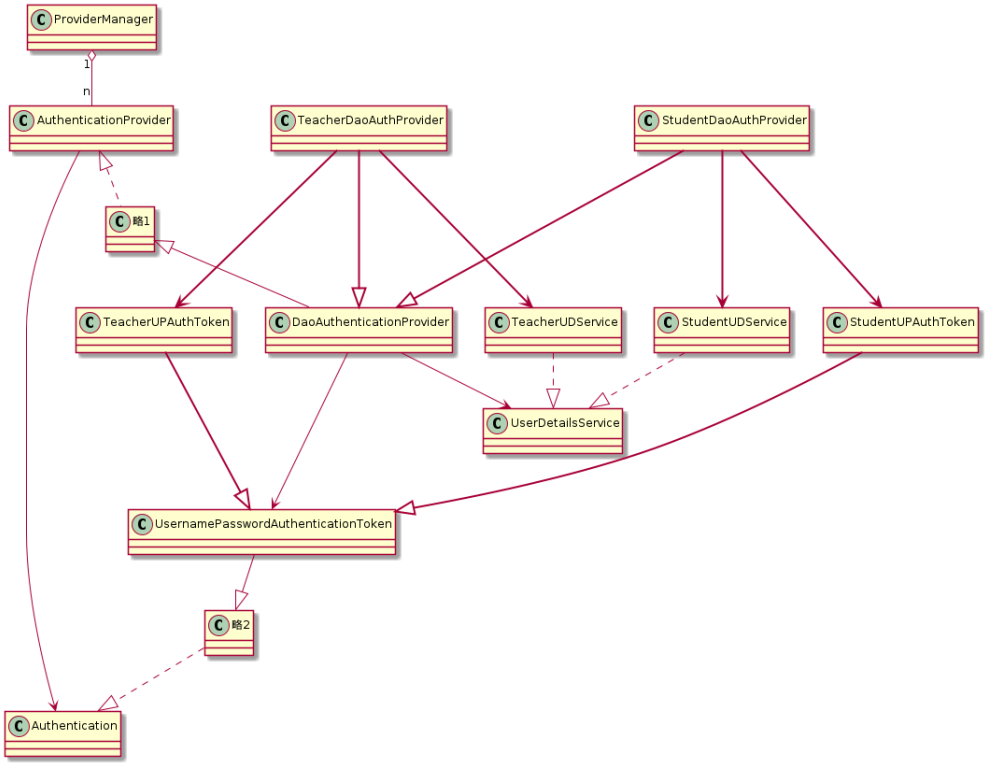

图 5.6　SpringSecurity 多用户表验证

（2） 后端处理 CORS 跨域问题

CORS(Cross-origin resource sharing)——跨域资源共享，一种可以允许访问其他域名下的受限资源的机制。目前 CORS 的现行标准由 WHATWG(Web Hypertext Application Technology Working Group) 在网络浏览器中实现和测试。

CORS-preflight request——预检请求，对于可能会对服务器造成副作用的请求（非简单请求），浏览器会先使用 OPTIONS 方法发送预检请求询问服务器是否允许此次跨域请求。

简单请求需满足：

（1） 请求方法： GET、POST、HEAD

（2） 只使用了如下的安全首部字段，不得人为设置其他首部字段

```
Accept
Accept-Language
Content-Language
Content-Type 仅限以下三种
       text/plain
       multipart/form-data
       application/x-www-form-urlencoded
HTML头部header field字段：
           DPR、Download、Save-Data、Viewport-Width、WIdth
```

（3） 请求中的任意 XMLHttpRequestUpload 对象均没有注册任何事件监听器；XMLHttpRequestUpload 对象可以使用 XMLHttpRequest.upload 属性访问

（4） 请求中没有使用 ReadableStream 对象

总的来说，在 CORS 问题中涉及两种请求：

（1） CORS 请求：CORS 请求是一个包含“Origin”报头的 HTTP 请求。它不能被可靠地识别为参与了 CORS 协议，因为所有的方法既不是’ GET ’ 也不是’ HEAD ’ 的请求都包含了’ Origin ’ 头。

（2） CORS preflight 请求：也就是所谓的预检请求，是检查 CORS 协议是否被理解的 CORS 请求。它使用’ OPTIONS ’ 作为方法，包括以下 header:

（1） `Access-Control-Request-Method` ：指示将来对同一资源的 CORS 请求可能使用的方法。

（2） `Access-Control-Request-Headers` ：指示将来对同一资源的 CORS 请求可能使用的头。

OPTIONS 方法：用于描述对于目标资源的通信选项。客户端可以放松 HTTP OPTIONS 请求来询问 web 服务器支持的 HTTP 方法和其他的选项。如果请求 URL 是一个’*’，那么 HTTP OPTIONS 请求应用于一般的服务器而不是特定的资源。使用 HTTP OPTIONS 方法的请求应该只检索数据 (服务器不能更改其状态)。如果需要更改服务器上的数据，需要使用 POST、PUT、PATCH 或 DELETE 方法。

在 CORS 问题中涉及的响应：

（1） 对于 CORS 请求的响应可以包含下面的 header：

（1） `Access-Control-Allow-Origin` ：通过在相应的 header 中添加被允许的 Origin（可以为 null 也可以为’*’）可以指示那些响应是否可以被分享。

（2） `Access-Control-Allow-Credentials` ：指示当请求的凭据模式为”include” 时响应是否可以被分享。

（3） `Access-Control-Expose-Headers` ：通过列出 header 的名称来指示哪些 header 可以作为响应的一部分公开。

（2） 对于 CORS preflight 请求可以包含下面的 header：

（1） `Access-Control-Allow-Methods` ：指示当请求目的为 CORS 协议时响应的 URL 支持那些请求方法。

（2） `Access-Control-Allow-Headers` ：指示当请求的目的为 CORS 协议时相应的饿 URL 支持那些 header。

（3） `Access-Control-Max-Age` ：表示’ Access-Control-Allow-Methods ’ 和’ Access-Control-Allow-Headers ’ 头提供的信息可以被缓存的秒数 (默认为 5)

CORS 流程（图5.7）：Spring Security 配置配置 CORS：

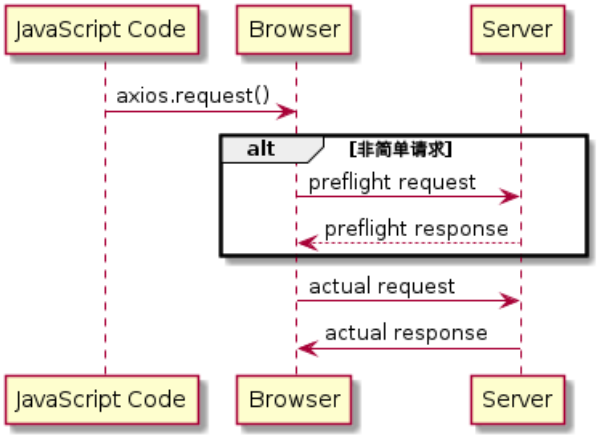

图 5.7　CORS 流程

（1） 打开 CORS

```
// MySecurityConfig.java
  protected void configure(HttpSecurity httpSecurity)
                          throws Exception {
```

```
    httpSecurity..cors()
```

（2） 配置配置 CORS response

```
// MySecurityConfig.java
@Bean
CorsConfigurationSource corsConfigurationSource() {
  CorsConfiguration configuration = new CorsConfiguration();
  configuration.addAllowedOrigin("*");
              // 应设置为前端服务器域名前缀
  configuration.setAllowedMethods(
            Arrays.asList("GET", "POST", "OPTION"));
  configuration.addAllowedHeader("*");
  // 允许的Header，起码需要允许 Access-Control-Allow-Origin
  UrlBasedCorsConfigurationSource source =
                       new UrlBasedCorsConfigurationSource();
  source.registerCorsConfiguration("/**", configuration);
  return source;
}
```

（3） 前端对 Token 的处理

这里使用 Vuex 进行管理，为身份验证单独创建一个 auth 模块。

```
// src/store/index.js
import {
  createStore
} from 'vuex'
import authModule from './modules/auth'

const store = createStore({
  modules: {
    auth: authModule,
  }
})

export default store;
```

（1） 在 auth.js 的 state 中：创建一个 authDate 对象，其中保存用户的 Token，refreshToken，tokenExp，userId，userName，userType，以及一个 loginStatus。

（2） 在 auth.js 的 getter 中：创建 getLoginStatus() 用于获取用户的登陆状态（是否登陆），创建 getAuthDate() 用于获取用户的身份信息，创建 isTokenActive() 用于判断 Token 是否过期。

（3） 在 auth.js 的 mutations 中：创建 saveTokenDate() 用于更新 authDate 以及将 token 保存在浏览器的 localStorage 中。创建 setLoginStatus() 用于更新登陆状态。创建 signout() 用于登出（清除 localStorage 中的 Token，以及将登陆状态更改为未登陆）。

（4） 在 auth.js 的 actions 中：创建异步函数 login() 用于登陆（使用 axios 发送请求，将 paylaod(username, password, usertype) 放在请求体中发送给后端，使用 mutations 中方法处理返回的结果，成功登陆。

（4） 前端配置导航守卫

本系统添加全局的前置守卫：使用 `router.beforeEach` ，该方法接收三个参数，to，from，next（其中第三个参数可选）

（1） to：想要导航到的规范化格式的目标路由位置。

（2） from：其格式为规范化，表示导航时当前的即正要离开的路由位置。

（3） next：经过判断后应该导航到的路由位置。

配置的导航守卫的大体思路是只要未登陆都需要跳转到 login 界面，但是如果访问 login 界面，此时系统尚未登陆，然后又会跳转到 login 界面，无限跳转，无法正常工作。所以这里为每个页面的路由设置一个参数 requiredAuth 用于指示该页面是否需要导航守卫。设置方法如下：

```
// src/router/index.js
import Login from '../components/Login.vue'
const routes = [
{
  path: '/login',
  name: 'Login',
  component: Login,
  meta: {
    requiredAuth: false
  }
},
]
```

导航守卫流程：

（1） 检查 store 中是否保存了 token，如果没有保存则需要从 localStorage 中获取并保存在 store 中。

（2） 调用 auth 模块中的 isTokenActive getter，从而获知该 Token 是否过期。

（3） 如果 Token 尚未过期并且想要导航到的目标路由需要验证 meta.requiredAuth === true）则跳转到系统首页，如果并不满足上述条件则需要跳转到 login 页面。

（5） 为每个请求附带 Token

思路为进行全局配置，在 axios 发送每个请求之前在 Header 中添加 authenticate 字段，其值为登陆时从后端获取的 Token。

使用 axios 的 interceptors（拦截器）实现。interceptors 主要可以分成两种：

（1） 处理 request

```
// 添加一个请求拦截器
axios.interceptors.request.use(function (config) {
    // 在请求发送前的一些处理
    return config;
  }, function (error) {
    // 在请求错误前的一些处理
    return Promise.reject(error);
  });
```

（2） 处理 response

```
// 添加一个响应拦截器
axios.interceptors.response.use(function (response) {
    // 任何处于2xx范围内的状态代码都会触发此函数
    // 处理返回的数据
    return response;
  }, function (error) {
    // 任何超出2xx范围的状态码都会触发该函数
    // 处理响应错误
    return Promise.reject(error);
  });
```

系统需要为请求添加拦截器：

```
// main.js
```

```
axios.interceptors.request.use((config) => {
  const authData = store.getters['auth/getAuthData'];
  config.headers.common.Authorization = 'bearer \${authData.token}';
  return config;
});
```

首先从 store 中获取用户身份信息，之后通过操作 config 对象在 header 中添加 Authorization 字段，值为用户身份信息当中保存的 token。在这样的里处理之后所有的请求都会在 header 中自动加入 token。

5.5 Jackson 序列化时无限递归问题

```
public class A {
  private Integer id;
  private List<B> bs;
}
```

```
public class B {
  private Integer id;
  private A a;
}
```

对应的数据库表表5.1和5.2所示。

例子（省略其他相关性不大代码）：

表 5.1 jackson-A

A 的 id
id

表 5.2 jackson-B

B 的 id	A 的 id
id	A_id

关于 Jackson 需要知道的两个概念：

（1） 序列化：从实体对象映射为 Json 对象

（2） 反序列化：从 Json 对象映射为实体对象

如果现在获取 A，序列化 A 的时候会序列化 bs，而 bs 中的每一个 B 对象都需要序列化 A，此时会造成无限序列化。

解决方式：

（1） 使用 @JsonManagedReference 和 @JsonBackReference

@JsonManagedReference(com.fasterxml.jackson.annotation.JsonManagedReference)

@JsonBackReference(com.fasterxml.jackson.annotation.JsonBackReference)

Managed 用于“一方”，即 A 中的 List<B>，Back 用于“多方”，即 B 中的 A，表示能从 A 放找到 B，但是无法从 B 放找到 A。

（2） 使用 @JsonIgnore

@JsonIgnore(com.fasterxml.jackson.annotation.JsonIgnore)

第一种方法的问题是，序列化时 A 会序列化 List<B>，B 不会序列化 A，如果此时想要 A 不序列化 List<B>，B 序列化 A 就无法做到。可以将 @JsonIgnore 注解加载 List<B> 上从而实现。

（3） 使用 @JsonProperty(access = ...)

@JsonProperty(com.fasterxml.jackson.annotation.JsonIgnore)

前两种方法在序列化和发序列化时的效果都相同，但是如果此时的需求是前端发过来的 json 对象映射为 B 时需要映射 a 属性，但是发送给前端 json 对象时不需要映射 a 属性（防止造成无限递归序列化）时可以设置 JsonProperty.access()，序列化对应 read，反序列化对应 write，共有四种选择：

（1） AUTO
（2） READ_ONLY
（3） READ_WRITE
（4） WRITE_ONLY

选择第四种方式即可实现。

```
public class B {
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private A a;
}
```

5.6 文件上传模块 & 文件更新模块 & 文件下载模块

5.6.1 文件上传模块

（1） 前端配置

前端使用 Elment Plus 的上传组件

```
<template>
  <div class="student_upload_paper_container0 container0">
    <el-upload
      class="upload-demo"
      action="http://localhost:8089/student/studentuploadpaper/"
      :on-success="handleSuccess"
      :on-preview="handlePreview"
      :on-remove="handleRemove"
      :before-remove="beforeRemove"
      multiple
      :on-exceed="handleExceed"
      :file-list="fileList"
      :data="{userName}"
      :auto-upload="true"
      list-type="text"
    >
      <button class="custom-btn btn-12 uploadButton">
        <span>Click!</span>
        <span>UPLOAD</span>
      </button>
      <template #tip>
        <div class="el-upload__tip"></div>
      </template>
    </el-upload>
  </div>
</template>
```

action：上传的地址，即后端定义的接口。

multiple：设置可以多选文件。

on-suscess：文件上传成功时的钩子。

on-preivew：点击文件列表中已上传的文件时的钩子。

on-remove：文件列表移除文件时的钩子。

auto-upload：是否在选取文件后立即进行上传。

data：附带的数据，这里放入文件名字。

file-list：文件列表数据。

（2） 后端配置

（1） Controller：

```
@PostMapping("/student/studentuploadpaper")
public String studentUploadPaper(
    @RequestParam("file") MultipartFile file,
    Principal principal) {
```

MultipartFile file：用来映射文件。

Principal principal：从这里能取得发送请求的用户信息。

（2） Service：

```
public StudentFileEntity store(
    MultipartFile file, String username)
                                throws IOException {
    String filename = StringUtils.cleanPath(
        file.getOriginalFilename());
    StudentFileEntity studentFileEntity =
    new StudentFileEntity(filename, file.getContentType(),
                    file.getBytes());

    studentFileEntity.setStudentByStudentId(
       studentRepository.findByName(username));
    return studentFileRepository.save(studentFileEntity);
}
```

文件名字：从 file 中可以提取出文件名。

拥有该文件的学生姓名：从 principal 中可以获取。

文件数据：file.getBytes()

文件类型：file.getContentType()

至此，存储一个文件的所有信息已经获得，通过这些信息新建一个文件实体对象，使用文件 Repository 存储文件。

5.6.2 文件更新模块

由于页面的结构，该功能并不适合使用 Elment Plus 的组件。

前端功能实现：

（1） 文件更新页面 template 部分：

```
<input
  type="file"
  value=""
  id="refreshFile"
  @change="handleRefreshFile(\$event, file.id)">
<label
  for="refreshFile">
</label>
```

需要一个选择文件的按钮。

（2） 文件更新页面 Script 部分：

```
handleRefreshFile(e, id) {
  const file = e.target.files[0]
  const param = new FormData()
  param.append('file', file)
  param.append('id', id)
  param.append('name', file.name)
  const config = {
    headers: { 'Content-Type': 'multipart/form-data' }
  }
  this.studentRefreshFile({ param: param, config: config })
  .then((res) => {
    console.log('refresh file : ' + id);
  })
},
```

（1） 获得文件 const file
（2） 新建 Form 表单
（3） 为表单填入文件，文件 id，文件名
（4） 新建请求配置，配置 Content-Type 为 multipart/form-data
（5） 调用更新文件函数（传入参数文件和配置）

（3） 更新文件函数：

```
async studentRefreshFile ({
  commit,
}, payload) {
  const response = await axios.post(
```

```
    'http://localhost:8089/student/refreshFile',
                        payload.param, payload.config)
  return response
}
```

后端功能实现：

（1） Controller：

```
public String refreshFile(@RequestParam("file") MultipartFile file
                          , @RequestParam("id") Integer id) {
```

使用 MultipartFile file 映射新上传的文件，使用 Integer id 映射需要被更新的文件的 id。

（2） Service：

```
public void refreshFile(MultipartFile file, Integer id)
                                          throws IOException {
    StudentFileEntity original_file = fileR.findById(id).get();
    original_file.setName(file.getOriginalFilename());
    original_file.setType(file.getContentType());
    original_file.setData(file.getBytes());
    fileR.save(original_file);
}
```

首先从数据库取出原文件，更新文件的名称，文件的类型，文件的数据，之后重新保存回数据库。

5.6.3 文件下载模块

（1） 前端请求函数：

```
async studentDownloadThisFile({
  commit
}, payload) {
  const response = await axios.get(
    'http://localhost:8089/student/downloadThisFile', {
    params: payload,
    responseType: 'arraybuffer'
  });
  return response;
},
```

这里的重点是改变 XMLHttpRequest 对象的 responseType 属性，将其设置为”arraybuffer”，否则返回的数据会被解析为 Json 对象，无法正确的处理文件，导致文件乱码。

（2） 后端处理函数：

```
@GetMapping(value = "/student/downloadThisFile",
                        produces = "application/pdf")
public byte[] getMethodName(@RequestParam Integer fileId) {
    return fileService.downloadThisFile(fileId).getData();
}
```

这里需要注意两点，

第一点是一定要设置 response 的 Content-Type 为”application/pdf”，设置方法为将 produces 参数赋值为”application/pdf”。

第二点是因为本系统配置了统一返回对象，会对返回值在发送前进行封装，这次封装会导致返回的值不再是文件数据本身，所以需要设置不对 byte[] 类型进行重新封装。

```
@RestControllerAdvice
public class CustomResponseBodyAdvice implements
                                ResponseBodyAdvice<Object> {

   @Override
   public Object beforeBodyWrite(Object arg0, MethodParameter arg1,
                                          MediaType arg2,
          Class<? extends HttpMessageConverter<?>> arg3,
          ServerHttpRequest arg4, ServerHttpResponse arg5) {
            ...
      } else if (arg0 instanceof byte[]) {
          return arg0;
      }
      ...
```

5.7 文件评论模块

由于评论模块在学生页面和教师页面都需要使用，所以这里将评论功能抽出为一个组件。评论的结构是树形结构，选择对 Element UI 中的树形组件进行二次封装为一个组件。

新建一条评论需要四个属性，评论 id（回复哪一条评论，如果为空则表示新建一条评论），文件 id（表示在哪篇论文下评论），评论人，评论内容。

在 data() 中建立 fileId，commentId 对象，和组件进行绑定，这样在选择文件和选择评论时就可以记录这两个属性。评论内容只需要将 input 和 commentData 中的 comment 属性绑定就可以获得。

至于最后一个属性——评论人，在后台 Controller 接收到请求后利用 Principal 便可以获得发送请求的用户信息，至此四个属性全部获得便可以存储到数据库。

5.8 统计模块模块

统计数据由后台生成，传递给前台使用 JavaScript 可视化工具——Echarts 展示。

5.9 测试

软件测试是指基于一组有限的测试用例（通常是从无限的执行域中进行适当选择），动态地验证程序的行为，并于预期行为进行对比[14]。常见的测试方法有白盒测试和黑盒测试，白盒测试使基于代码的测试，一般用于单元测试，由程序员进行测试；黑盒测试基于需求描述，通常用于功能测试，一般由测试人员进行。白盒测试在系统开发过程中已经完成，下面记录了功能测试。

5.9.1 系统测试环境

如表5.4所示。

5.9.2 测试用例与测试过程

用户登陆测试如图5.8，需要测试不同用户的登陆能够成功，一点是用户只有输入正确的用户名和密码才能正确登陆，另一点是用户只有选择正确的用户类型才能正确登陆。每类用户都需要进行多次测试，使用正确用户名，错误用户名，正确密码，错误密码，正确类型，错误类型，各自组合测试。

论文上传测试如图5.9所示，需要测试学生上传的论文能否正确的使用后台保存到后台的数据库，论文的所属学生是否是登陆用户。

论文下载测试如图5.10和图5.11所示，需要在学生页面和教师页面都测试，需要测试点击下载按钮能否下载文件，下载的文件能否正常打开，文件的内容是否正确。

论文更新模块测试如图5.12所示，学生点击更新论文按钮，选择新的文件，点击确定，文件上传。需要测试文件是否更新成功，数据库中的文件附加信息（文件名，文件类型，所属学生等）是否正确，更新的文件能否下载，下载的文件及其附加信息是否正确。

论文评论模块测试如图5.13，图5.14所示，学生或教师可以新建评论（不点击回复按

表 5.3　系统配置表

硬件配置		
服务器端	CPU	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
	内存	6GB
	硬盘	1TB
	网卡	1000M 以太网卡
客户端	CPU	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
	内存	10GB
	硬盘	1TB
	网卡	1000M 以太网卡
软件配置		
服务器端	操作系统	Ubuntu 20.04.2 LTS
	数据库服务器	8.0.23-0ubuntu0.20.04.1 for Linux on x86_64 ((Ubuntu))
客户端	操作系统	Windows 10 家庭中文版 (版本号 20H2)
	浏览器	Microsoft Edge(版本 90.0.818.62)

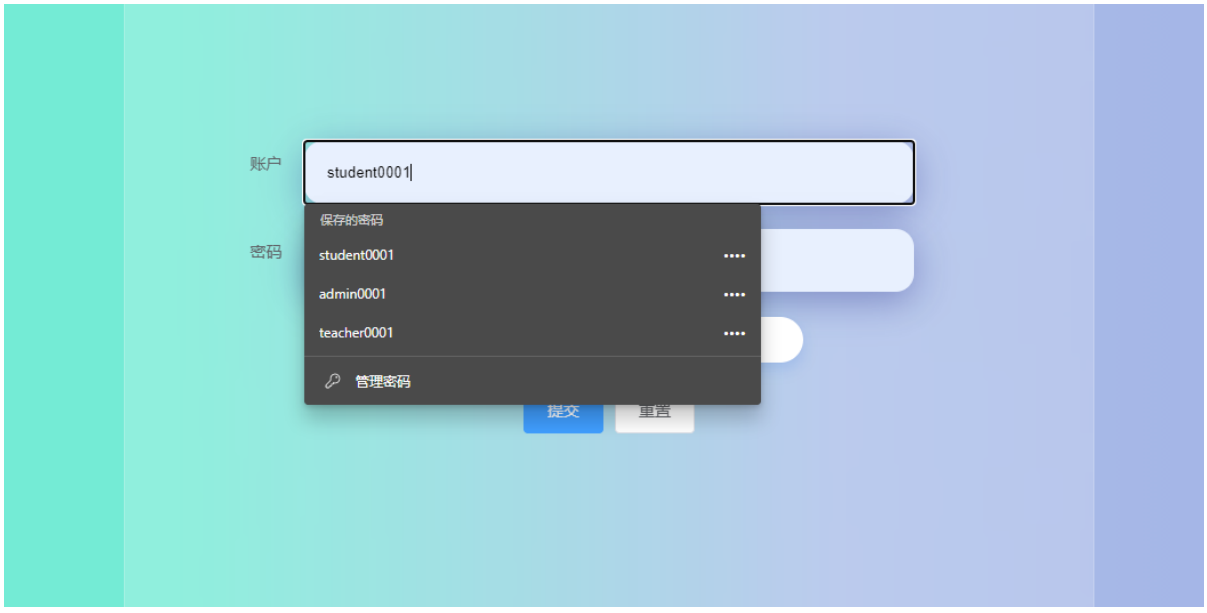

图 5.8　登陆测试图

表 5.4　测试用例表

测试项目	测试用例	测试结果
用户登陆	验证不同用户是否能正确登陆	功能准确实现
论文上传	验证能否正确上传 pdf 论文	功能准确实现
论文下载	验证能否正确下载 pdf 论文	功能准确实现
论文更新	验证能否正确更新选定的论文	功能准确实现
论文评论	验证能否正确新建评论，回复评论	功能准确实现
问题标签	验证能否正确为评论添加标签	功能准确实现
论文评分	验证能否正确为所选论文评分	功能准确实现
论文得分统计	验证能否正确统计论文的得分情况	功能准确实现
教师任务统计	验证能否正确统计教师的评审评分任务完成情况	功能准确实现
学生管理	验证能否正确修改学生信息	功能准确实现
教师管理	验证能否正确修改教师信息	功能准确实现

图 5.9　论文上传测试图

钮或者点击回复按钮之后点击取消），测试新建评论是否是顶级评论，点击评论条目右下角的回复（replay) 按钮将选择回复这一条评论，需要测试回复的评论是否为选择的评论的子级，评论人是否为登录用户，刷新后评论条目结构是否正确，使用其他用户登陆查看此论文评论是否一致。

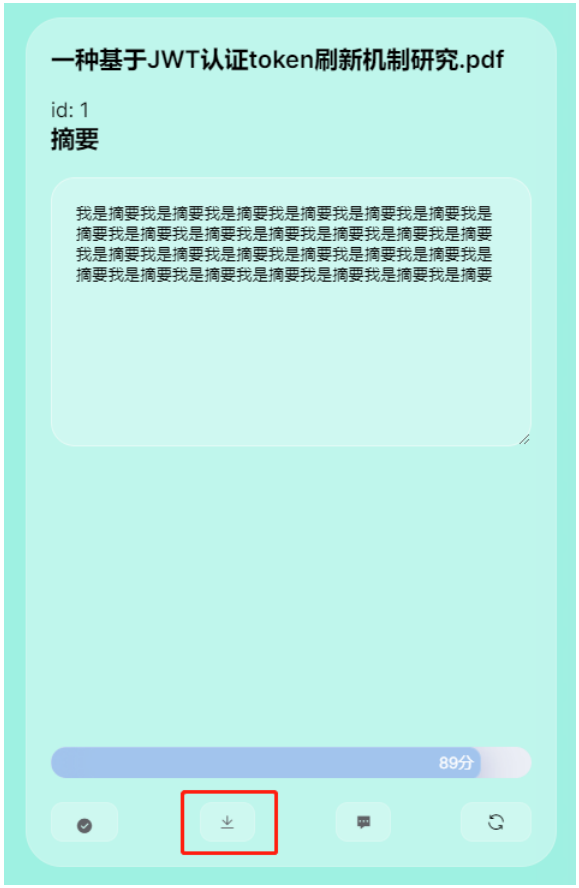

图 5.10　学生论文下载测试图

图 5.11　教师论文下载测试图

论文评分模块测试如图5.15和图5.16所示，教师可以点击评分按钮，弹出评分对话框，滑动分数滑动条可以选择分数，点解确定为学生评分。该部分需要测试是否可以评分，是否为正确的论文评分，分数是否正确，评分人是否正确记录，学生端是否正确显示分数。

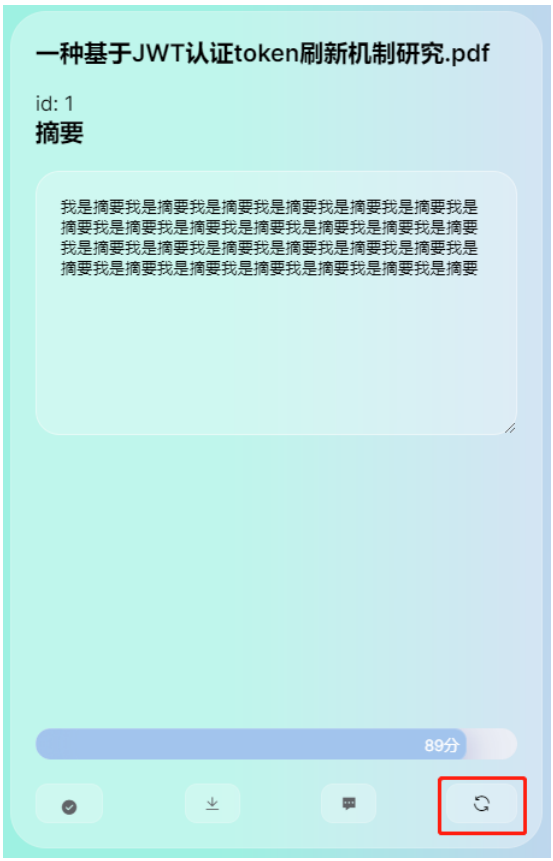

图 5.12　更新文件测试图

论文得分统计模块测试如图5.17所示，管理员可以切换论文得分统计页面，页面将展示学生的得分情况，需要测试各个分数段的人数是否正确，多次增删论文并评分，测试得分统计是否一致。

教师评审任务完成情况统计模块测试如图5.18所示，管理员可以切换到教师评审任务统计完成情况统计页面，需要测试各个教师管理的学生数是否正确，需要管理的所有论文数量是否正确，已经评分的论文数量是否正确，尚未评分的论文数量是否正确。

人员管理模块测试如图5.19和图5.20所示，在查看方面，需要测试所有学生信息是否正确显示；在修改信息方面需要测试各个字段的更改是否有效，密码不更改是否会清空密码，修改密码后是否能使用新密码登陆。

5.10　本章小结

本章主要介绍了在系统实现过程中的主要模块的实现，关键问题的解决，以及系统测试的过程。其中关键问题的的解决对系统的稳定性，可维护性都有很大的提升。

图 5.13　新建评论测试图

图 5.14　回复评论测试图

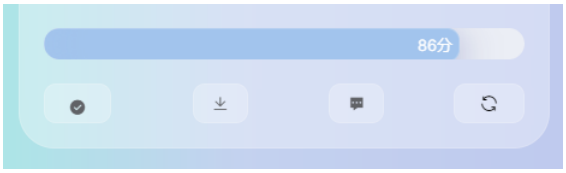

图 5.15　学生查看得分测试图

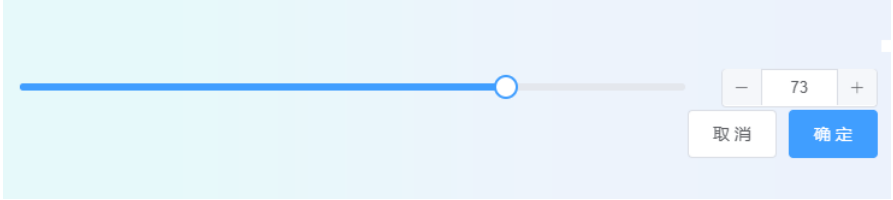

图 5.16　教师评分测试图

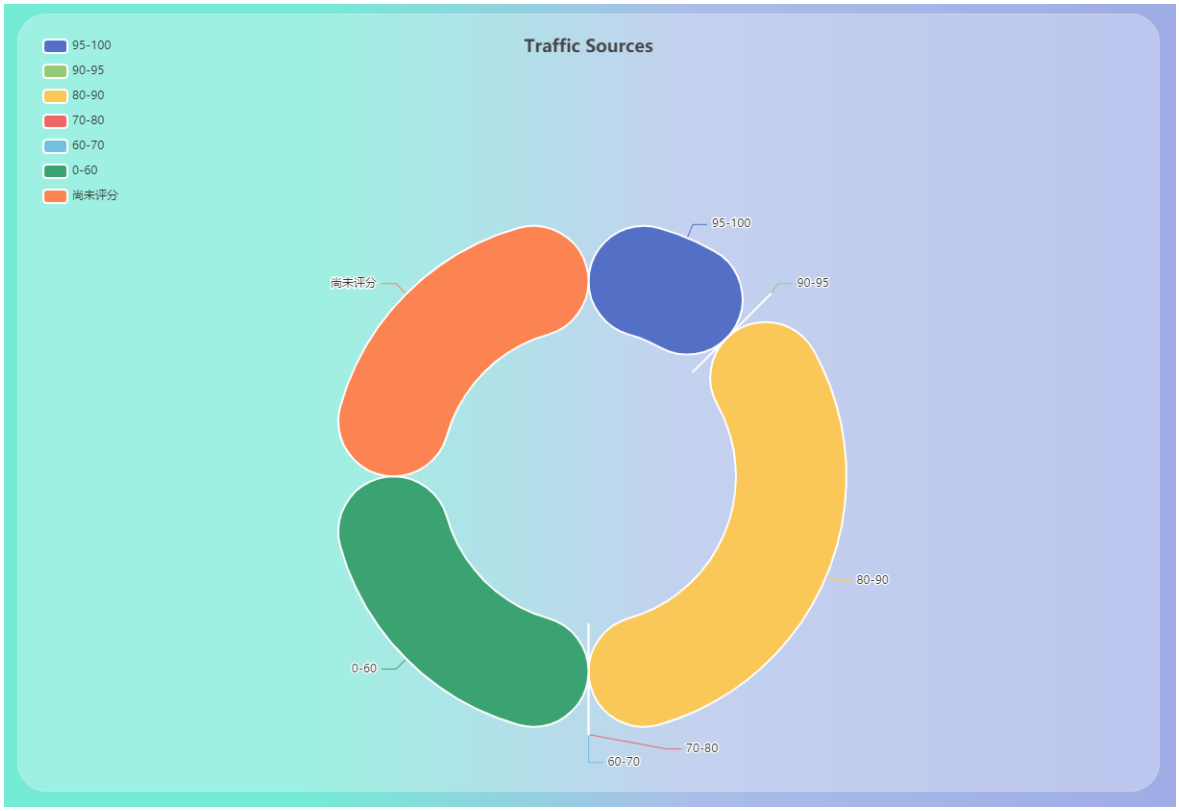

图 5.17　论文得分统计测试图

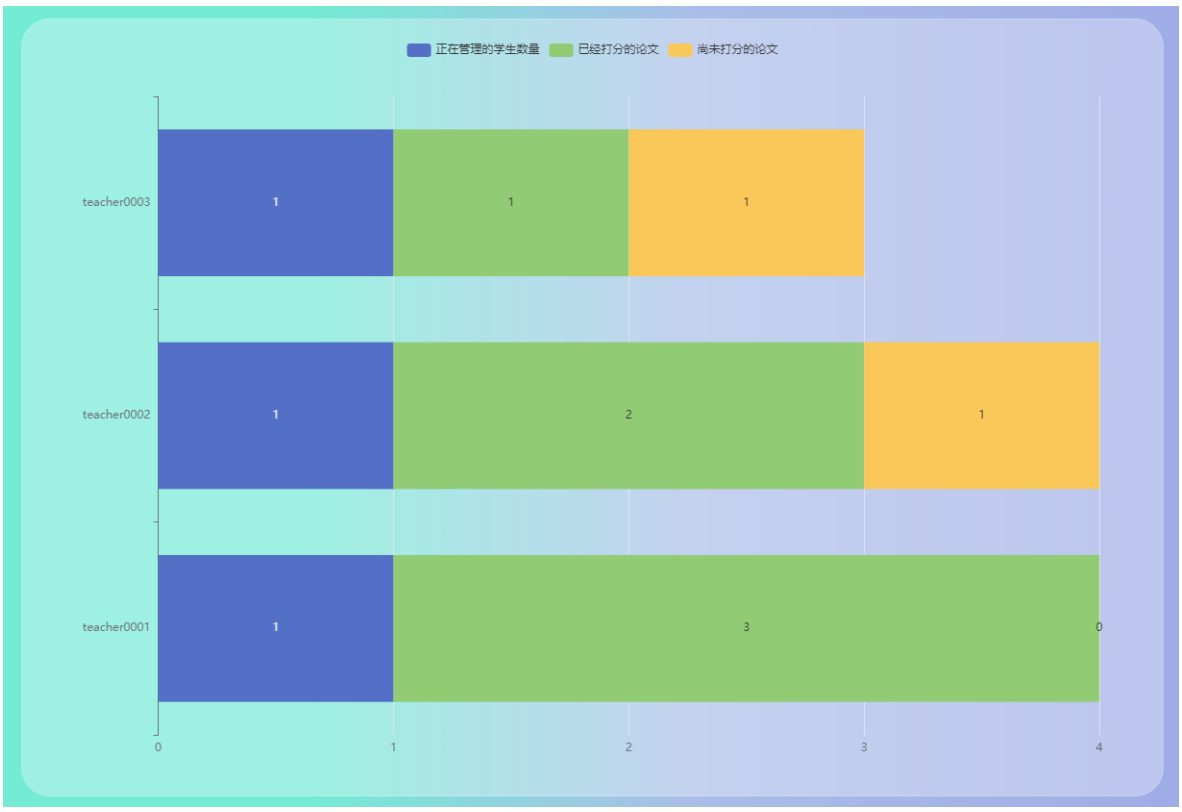

图 5.18　教师任务统计测试图

图 5.19　学生管理功能测试

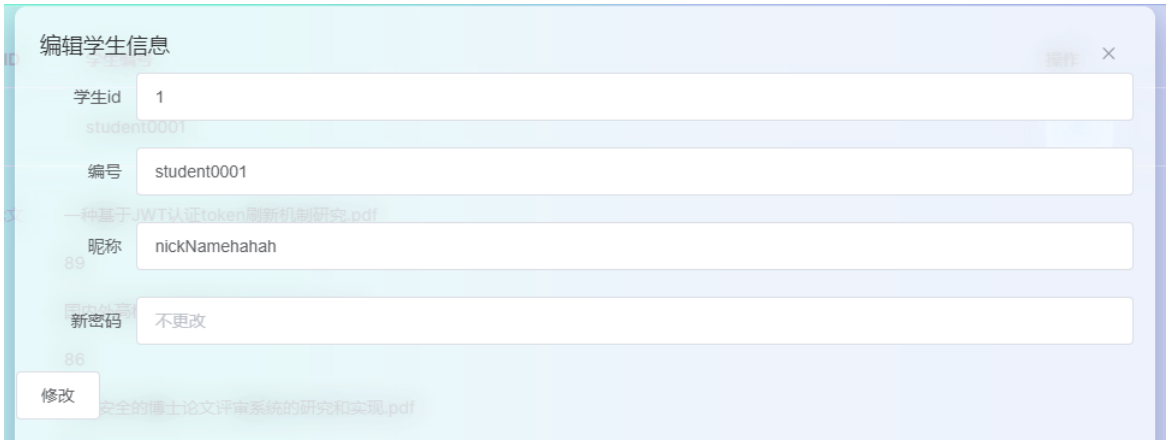

图 5.20　编辑学生功能测试

结　　论（设计类为设计总结）

本课题的主要内容是论文评审评分系统的研究与实现，在研究过程中分析了项目的研究背景和国内外发展情况，了解了学生，教师和教务的实际需求，以此为基础进行系统设计，详细的需求分析和系统设计保证了之后的系统实现不会出现较大的偏差。

在系统的实现过程中，系统选择了前后端分类技术体系，前端使用 Vue.js 框架配合 vuex，vue-router，Echarts 等插件和技术编写了友好的用户界面，后端使用 Spring Boot 框架配合安全框架 Spring Security 和 Json Web Token 实现了无状态的身份验证和授权，数据持久化方面使用 Spring Data Jpa 配合 ORM 工具 Hibernate 框架实现了面向对象的持久化操作方式，最后完成的系统整体安全可靠，易扩展，跨平台且易迁移。

在系统的测试过程发现了一些不足和新的想法，希望在后期的改进中实现：

（1） 文档的在线批阅：教师可以直接在文档上标注需要改进的地方并以评论的形式告知学生，学生同样可以在线查看文档查看问题所在。

（2） 论文感想收集：完成论文的学生可以发表感想和对晚辈的建议。

（3） 数据库优化：随着文档的增多，以往文档的管理和存储需要新的设计以保证系统的快速响应。

当然，应该将本系统和其他系统结合起来成为整个教务系统的一部分，帮助学生更好地完成学业，帮助学校更好地管理。

参 考 文 献

[1] 陈新梅, 赵元, 高雅, 等. 一流学科背景下本科毕业论文质量的提升——以山东中医药大学药学院 2017 届毕业生为例[J]. 山东化工, 2019, 48(1):148-149.

[2] 龚旭. 试析 “互联网 +” 时代下计算机科学技术发展趋势[J]. 轻纺工业与技术, 2020, 49(1): 147-148.

[3] 基于 Java+EE 的项目评审文档管理信息系统的研究与实现[J]. 2018 年 3 月.

[4] 方辉. 基于 ASP.NET 的在线学位论文评审系统设计与实现[D]. [出版地不详]: 广西师范大学, 2017.

[5] 周逸雅. 硕士毕业论文管理系统的设计与实现[D]. [出版地不详]: 电子科技大学, 2018.

[6] 刘辉, 于程名, 吴海平. ” 双一流” 建设下的高校研究生学位论文评审办法改革探讨[J]. 高教论坛, 2019(11):83-85.

[7] 王晓杰, 王欢欢. ” 互联网 +” 背景下计算机科学技术发展趋势[J]. 科技风, 2020(6):123-124.

[8] 郭煜, 刘文胜. 高校本科毕业设计 (论文) 的现状分析和改进措施[J]. 科教导刊, 2019(31): 42-43.

[9] SETIYANI L, SYAMSUDIN A, GINTINGS A, et al. The analysis of functional needs on undergraduate thesis information system management[J]. International Journal of Advances in Data and Information Systems, 2020, 1(2).

[10] 王悦, 张雷, 钱英军. 基于 SpringBoot 微服务的 Spring Security 身份认证机制研究[J]. 电脑编程技巧与维护, 2019(8):64-65,68.

[11] 沈佳棋, 倪珊, 王杰, 等. 基于 Vue+SpringBoot 的分类学科竞赛管理系统设计[J]. 无线互联科技, 2020, 17(17):74-77.

[12] WALLS. C, ZHANG. Spring 实战[M]. 北京: 人民邮电出版社, 2016.

[13] 周虎. 一种基于 JWT 认证 token 刷新机制研究[J]. 软件工程, 2019, 22(12):18-20.

[14] TSUI F, KARAM O, BERNAL B. 软件工程导论[M]. [出版地不详]: 机械工业出版社, 2018.

致　　谢

时光荏苒，四年的大学生活即将结束，最后的毕业论文为我的学业生活画上了句号。

感谢指导老师和老师的研究生的指导，给了本人许多启发。

感谢开发了开源前端框架 Vue.js 的尤雨溪先生的和其他数百名开源贡献者，感谢设计了让 Spring 更易使用的 Spring Boot 框架的 Pivotal 团队的，感谢开发了 Latex 的计算机领域的艺术大师——唐纳德·克努特，感谢开发了优秀的可视化工具的 Echarts 团队，感谢实现了优秀 ORM 工具的 Hibernate 框架团队，感谢发布了优秀文章的无数博主，感谢开发论坛上回答的问题无数优秀开发者。

感谢我的同学在前端方面的帮助，帮助我迈出了从 Vue2 切换到 Vue3 的第一步。

感谢我父母支持我完成了学业。

最后感谢我自己不懈坚持解决了开发过程中遇到问题。