山西工程技术学院

《Java 程序设计》课程设计报告

（ 2024 - 2025 学年第 一 学期）

系　　部：大数据与智能工程系

课程名称：Java 程序设计课程设计

专业班级：22 计算机科学与技术一班

学　　号：2210708130

学生姓名：郝泓毅

任课教师：段新娥

2024 年 12 月

目　录

摘　要

本程序旨在开发一个基于 Spring Boot 框架的学生在线聊天软件，提供即时消息通信、用户管理和聊天室功能。该系统的主要架构采用 Spring Boot 框架，并结合 Java 语言作为后端的核心开发语言，同时利用 MySQL 数据库进行用户信息管理和聊天记录的存储。通过这种设计，实现一个高效、可靠的即时通信平台，用户可以方便地进行实时聊天，并且能够管理个人账号信息。

前端部分使用了 HTML5、CSS、JavaScript 以及 Vue.js 等技术栈，以构建响应式、交互友好的用户界面。前端开发主要包括“登录界面”、“注册界面”和“聊天室主界面”三个模块。登录界面实现了用户账号和密码的验证功能，用户在输入正确的用户名和密码后可以成功进入系统。注册界面则提供了用户信息的注册功能，当用户填写完必要的个人信息并提交后，系统会将注册信息传送至后端进行存储。

前端与后端的交互通过 JavaScript 和 Vue.js 的结合得以实现，当用户在登录界面或注册界面填写并提交信息时，前端会通过 POST 请求将数据发送给后端进行处理。后端接收到数据后，会进行相应的逻辑处理，如验证用户输入的账号密码是否正确、检查注册信息是否合法等。处理完毕后，后端通过 GET 请求将处理结果返回给前端，前端根据返回的数据展示相应的提示信息。

后端开发部分采用了 Spring Boot 框架，这一框架的使用大大简化了项目的开发和配置。通过 Spring Boot 提供的各种组件和工具，可以高效地构建和管理应用的各个部分。后端利用 JDBC 技术连接 MySQL 数据库，并通过数据访问层将用户信息和聊天记录持久化存储到数据库中。用户的注册信息会被存储到数据库的用户表中，而每一条用户发送的聊天消息则会被保存在消息表中，确保聊天记录可以长期存储并随时访问。

关键词：Java；MySQL；SpringBoot；Web；HTML；VUE；Javascript；

第 1 章　项目介绍

1.1 设计校园在线聊天室的背景

设计校园在线聊天室的背景时，我们需要考虑到校园特有的文化、需求和用户群体（学生、教师、校友等)。建立一个校园内部的在线聊天室平台，旨在促进学生与教师之间、同学与同学之间的交流，帮助新生适应校园生活、进行学术讨论、组织社团活动、分享校园新闻等。目前，国内高校大多使用的 APP 是“学习通”“微信”“QQ”“钉钉”等社交媒体软件，对于学生的日常使用有许多不便，例如：“文件保存时间短”“文件大小受限”等问题。为解决以上问题，我开发了校园在线聊天室；

此软件可以实现以下功能：**实时聊天**：支持学生和教师进行即时沟通。**话题讨论**：用户可自行定义不同的聊天室对不同的话题进行讨论。**文件共享**：有相关后端技术，可以支持用户在聊天室内共享文件、图片等资料。**用户管理**：可以通过 JDBG 连接数据库进行注册、登录、修改资料等。

1.2 设计校园在线聊天室的意义

设计校园在线聊天室的意义可以从多个角度来探讨，主要包括促进校园交流、提高学习效率、加强学生社交以及支持校园文化建设等方面。首先促进学生之间的交流与合作，其次打破时间与空间的限制，校园在线聊天室可以帮助学生在不同时间和地点进行即时沟。促进学习与合作：通过在线聊天室，学生可以方便地讨论学术问题、共享学习资源或协作完成项目任务，尤其是对于跨学科的合作学习来说，提供了一个有效的沟通平台。提高教学与学习的效率教师可以通过在线聊天室及时解答学生的问题，在线聊天室可以成为师生进行日常沟通的桥梁。学生和教师可以在聊天室中分享学习资料、课程讲义、参考文献等资源，进一步提高学习效率。

对于作业、考试或项目，教师可以通过聊天室提供快速反馈，帮助学生改进学习方法。增进学生归属感：通过在线聊天室，学生可以更容易地结识来自不同背景的同学，增强与学校的联系感和归属感。

校园在线聊天室不仅是一个简单的沟通工具，它在促进学生学术交流、社交互动、校园文化建设以及学校管理等多个方面具有重要意义。

1.3 设计内容

1.用户登录：使用 VUE、HTML5、CSS、JavaScript 等技术制作了登录界面，同时结合了 Java 后端与 MySQL 数据库链接，通过 SQL 语言存储了相关的“user”（用户），“room”（聊天室）以及“comment”（评论）等等的相关表和信息等。

2.用户注册：基于 Vue.js 和 Element UI 构建的注册页面的 HTML 代码。用户可以通过该页面注册一个账户，包括上传头像、输入账号、密码和个人简介等信息。

3.聊天室：使用 Vue.js 和 Element UI 开发的校园聊天平台，用户信息展示、聊天室功能、以及聊天室内讨论区的动态交互可以实现以下功能：

- 头像上传： 用户可以上传新的头像
- 房间切换和显示： 通过点击不同的房间名来切换讨论区。
- 聊天室评论： 用户可以看到每个聊天室的评论，评论可以是文本、图片或音频。
- 编辑个人信息： 用户可以修改个人信息，包括账号、密码和简介。

4.数据库：生成了 3 个表：“comment”：存储评论数据“room”：存储聊天室的内容和房间数据“user”：用于存储用户数据，密码，用户名，头像等；

第 2 章　开发环境

2.1 集成开发环境 IDE

IntelliJ IDEA 是由 JetBrains 开发的一款广泛使用的集成开发环境（IDE），专为提高开发者的生产力和代码质量而设计。作为一个强大的多语言 IDE，IntelliJ IDEA 以其智能代码补全、代码分析和重构功能而闻名。IntelliJ IDEA 提供了强大的调试工具、版本控制集成（如 Git、SVN 和 Mercurial），以及丰富的插件生态系统，使得开发者能够根据需要扩展和定制其功能。此外，IntelliJ IDEA 拥有直观的用户界面和高度可配置的工作环境，支持快速导航和高效的代码编辑，适用于从小型项目到大型企业级应用的开发。这些特性使得 IntelliJ IDEA 成为了众多开发者的首选 IDE。

2.2 数据库

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL 是最好的 RDBMS (Relational Database Management System，关系数据库管理系统)应用软件之一。

2.3 框架

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化 Spring 应用的创建、运行、调试、部署等。使用 Spring Boot 可以做到专注于 Spring 应用的开发，而无需过多关注 XML 的配置。Spring Boot 使用“习惯优于配置”的理念，简单来说，它提供了一堆依赖打包，并已经按照使用习惯解决了依赖问题。

第 3 章　总体设计

3.1 前端设计

3.1.1 注册界面

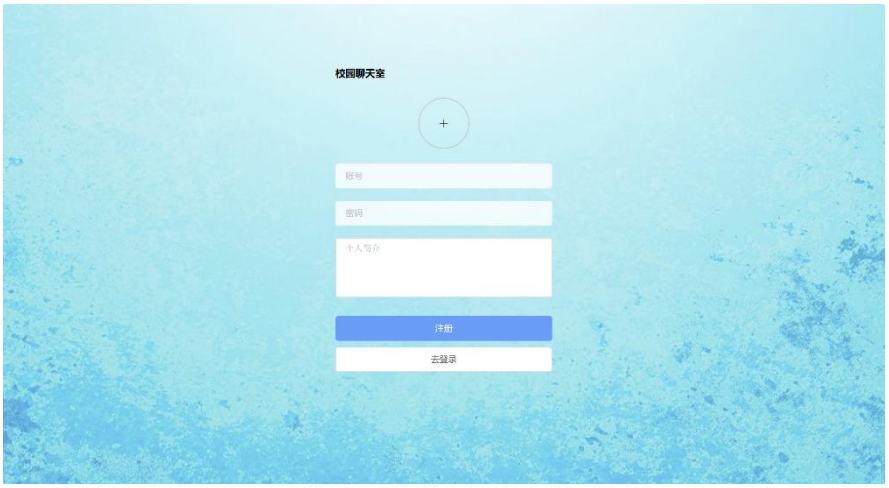

图 3-1 注册界面

结合了 Vue.js 和 Element UI（基于 Vue.js 的 UI 组件库）来实现一个登录页面。页面使用了 CSS 样式来构建视觉效果，并且通过 Axios 来进行表单数据的提交和处理。

VUE 功能实现部分：

```
new Vue({
        el: '#app',
        data() {
            return {
                icon: "",
                account: "",
                password: "",
                remark: "",
            }
        },
        methods: {
            handleUpload(res) {
                let data = res.data;
                if (res.code !== 200) {
                    this.$message.warning("文件上传失败");
                    return;
                }
                this.$set(this, 'icon', data);
            },
```

```
        loginHandler() {
            window.location.href = "login.html";
        },

        registerHandler() {
            if (!this.account) {
                this.$message.warning("请输入账号");
                return;
            }
            if (!this.password) {
                this.$message.warning("请输入密码");
                return;
            }

              let params = {account: this.account, password: this.password, icon: this.icon, remark: this.remark};

                                  axios.post("/user/register", params, {headers: {"Content-Type": "application/json"}}).then((res) => {
                let data = res.data;
                if (data.code !== 200) {
                    this.$message.warning(data.msg);
                    return;
                }
                this.$message.success(data.msg);
                setTimeout(() => {
                    window.location.href = "/login.html";
                }, 800);
            });
        }
    }
})
```

使用 **data()**返回一个对象，包含了当前组件的状态数据：**icon**: 用户上传的头像，初始为空字符串。**account:** 用户输入的账号，初始为空字符串。**password:** 用户输入的密码，初始为空字符串。**remark:** 用户输入的备注信息，初始为空字符串。

使用 methods: 定义了两个方法 **handleUpload** 和 **registerHandler**：

handleUpload(res)：

res 是文件上传的响应对象，包含上传结果信息。如果 res.code 不等于 200，表示文件上传失败，调用 this.$message.warning() 弹出警告信息提示“文件上传失败”。如果上传成功，则通过 this.$set(this, 'icon', data) 将返回的数据存储在 icon 数据属性中。确保 Vue 能够响应性地更新 icon 数据。

registerHandler() 方法：

检查 account 和 password 是否为空。如果为空，则弹出警告提示并停止执行。如果输入合法，则创建一个 params 对象，包含 account, password, icon, remark 这几个字段。然后通过使

用 axios.post() 向注册界面发送一个 POST 请求，传递 params 作为请求体，且设置请求头的 Content-Type 为 application/json，表示请求体是 JSON 格式。执行逻辑后，如果响应的 data.code 不等于 200，表示注册失败，弹出警告信息。如果响应成功（即 data.code === 200），弹出成功提示，并在 800 毫秒后跳转到登录页面。

3.1.2 登录界面

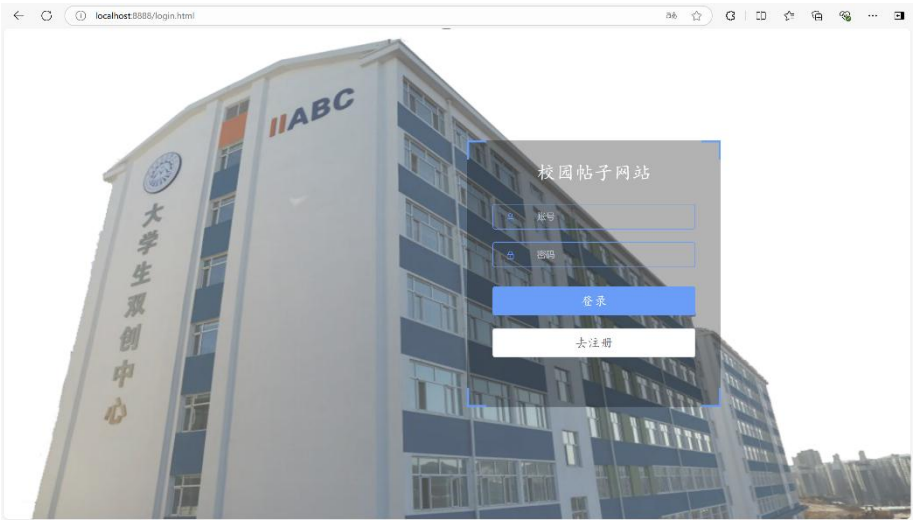

图 3-2 登陆界面

使用了 Vue.js 和 Element UI 组件库，结合了 Axios 用于发起 HTTP 请求

VUE 部分：

```
new Vue({
        el: '#app',
        data() {
            return {
                account: "",
                password: "",

            }
        },
        methods: {
            handlerRegister() {
                window.location.href = "register.html";
            },

            handlerLogin() {
                if (!this.account) {
                    this.$message.warning("请输入账号");
                    return;
                }
```

```
                if (!this.password) {
                    this.$message.warning("请输入密码");
                    return;
                }

                let params = {account: this.account, password: this.password};

                                axios.post("/user/login", params, {headers: {"Content-Type":
"application/json"}}).then((res) => {
                    let data = res.data;
                    if (data.code !== 200) {
                        this.$message.warning(data.msg);
                        return;
                    }
                    this.$message.success(data.msg);
                    setTimeout(() => {
                        window.location.href = "/";
                    }, 800);
                });
            }
        }

    })
```

data 里包含了 account 和 password，这两个字段分别用来存储用户输入的账号和密码
handlerRegister:该方法会通过 window.location.href 将页面跳转到 register.html，从而实现用户注册页面的跳转。
handlerLogin):在登录时，首先会检查 account 和 password 是否为空，如果为空则会弹出提示信息并返回，不继续执行。使用 Axios 向 /user/login 发送一个 POST 请求，传递 account 和 password 作为请求体。请求的 headers 指定了请求的 Content-Type 为 application/json，表示发送 JSON 格式的数据。根据响应中的 code 字段来判断登录是否成功。如果 code 不为 200，说明登录失败，此时会显示错误信息。否则，显示成功消息，并在 800 毫秒后跳转到首页。
catch 方法用来捕获请求中的错误（如网络问题、服务器错误等），并在出现错误时给出提示信息。

3.1.3 聊天室界面

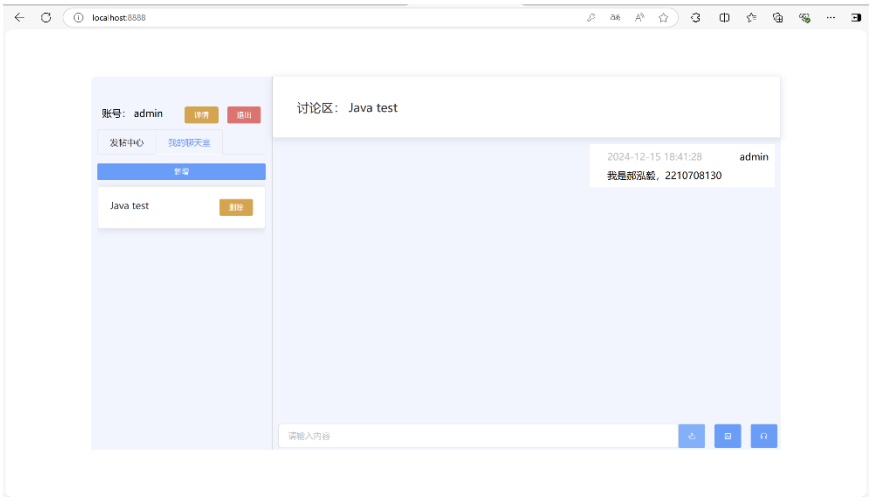

图 3-3 聊天室

```
<div class="main-box">
    <div class="main-item-box">
        <div style="width: 300px; border-right: 1px solid #c0c0c0; display: flex; flex-flow: column;">
            <!-- 左侧用户信息及聊天室导航栏部分 -->
            <div style="display: flex; justify-content: center; margin-top: 30px;">
                <img v-if="user.icon" :src="user.icon" class="avatar icon">
            </div>
            <div style="text-align: center; margin-top: 20px; display: flex; justify-content: space-around;">
                <div>账号： {{user.account}}</div>
                <div>
                    <el-button @click="toDetail" size="mini" type="warning">详 情</el-button>
                    <el-button @click="loginOut" size="mini" type="danger">退出</el-button>
                </div>
            </div>
            <div style="padding: 10px; box-sizing: border-box; flex: 1">
                <el-tabs type="card">
                    <el-tab-pane label="聊天室中心">
                        <div class="userList">
                            <el-card @click.native="showRoom(item)" style="margin-bottom: 10px; cursor: pointer;"
                                     v-for="item in roomData" :key="item.id">
                                <div style="display: flex;">
                                    {{item.name}}
```

```
                                    </div>
                                </el-card>
                                <el-empty v-if="roomData.length === 0"/>
                            </div>
                        </el-tab-pane>
                        <el-tab-pane label="我的聊天室">
                            <div class="userList">
                                <el-button @click="handlerShowEditRoom" size="mini"
type="primary" style="width: 100%">
                                    新增
                                </el-button>
                                <el-card @click.native="showRoom(item)" style="margin-
top: 10px; cursor: pointer;"
                                         v-for="item in selfRoom" :key="item.id">
                                    <div style="display: flex; justify-content: space-
between;">
                                        <div>{{item.name}}</div>
                                        <el-button @click.stop="handlerDelete(item)"
size="mini" type="warning">删除
                                        </el-button>
                                    </div>
                                </el-card>
                                <el-empty v-if="selfRoom.length === 0"/>
                            </div>
                        </el-tab-pane>
                    </el-tabs>
                </div>
            </div>
            <div v-if="showDetail">
                <!-- 用户详情展示及编辑部分，根据 showEdit 等条件控制显示不同内容 -
->
                <div v-if="showEdit" style="width: 500px; margin: 0 auto;">
                    <div style="display: flex; justify-content: center; margin: 30px auto;
position:relative;">
                        <div style="position: absolute; top: -10px; right: 160px; cursor:
pointer" @click="removeIcon"><i class="el-icon-error"></i></div>
                        <el-upload
                                class="avatar-uploader"
                                action="/file/upload"
                                :show-file-list="false"
                                :on-success="handleUploadIcon">
                            <img v-if="updateUser.icon" :src="updateUser.icon"
class="avatar icon">
                            <i v-else class="el-icon-plus avatar-uploader-icon"></i>
```

```
                    </el-upload>
                </div>
                <div style="margin-top: 20px;">
                    <el-input placeholder="账号" disabled v-model="updateUser.account"></el-input>
                </div>
                <div style="margin-top: 20px;">
                    <el-input placeholder="密码" type="password" v-model="updateUser.password"></el-input>
                </div>
                <div style="margin-top: 20px;">
                    <el-input placeholder="个人简介" type="textarea" :rows="4" resize="none" v-model="updateUser.remark"></el-input>
                </div>
            </div>
            <div v-else>
                <div style="display: flex; justify-content: center; margin-top: 100px;">
                    <img v-if="user.icon" :src="user.icon" class="avatar icon">
                </div>
                <div style="text-align: center; margin-top: 20px; display: flex; justify-content: space-around;">
                    <div>账号： {{user.account}}</div>
                </div>
                <div style="text-align: center; margin-top: 20px; display: flex; justify-content: space-around;">
                    <div>个人简介：
                        <span v-if="user.remark">{{user.remark}}</span>
                        <span v-else>暂无个人简介</span>
                    </div>
                </div>
            </div>
            <div style="display: flex; justify-content: center; margin-top: 100px;">
                <el-button @click="handlerEdit" type="primary">{{showEdit? '编辑' : '去编辑'}}</el-button>
                <el-button @click="toHandlerClose" type="danger">{{showEdit? '返回' : '关闭'}}</el-button>
            </div>
        </div>
        <div v-else style="flex: 1">
            <!-- 聊天消息展示及发送部分，根据 selectRoom 等条件控制显示不同内容 -->
            <div v-if="!selectRoom.name"
                 style="height: 100%; display: flex; justify-content: center; flex-flow: column;">
```

```
                    <el-empty/>
                </div>
                <div v-if="selectRoom.name" style="height: 100%;">
                    <el-card>
                        <div style="display: flex;">
                            <div style="margin-left: 20px; font-size: 20px; height: 60px; line-height: 60px;">聊天室：
                                {{selectRoom.name}}
                            </div>
                        </div>
                    </el-card>
                    <div style="height: calc(100% - 145px); overflow:auto; padding: 0 10px; box-sizing: border-box;">
                        <div v-if="commentData.length === 0"
                             style="display: flex; justify-content: center; flex-flow: column; height: 100%">
                            <el-empty></el-empty>
                        </div>
                        <div v-for="(item, index) in commentData" :key="index" style="display: flex; margin-top: 10px;">
                            <div v-if="user.account === item.account"
                                 style="display: flex; justify-content: end; width: 100%">
                                <div style="display: flex; justify-content: end; background: #FFFFFF; padding: 10px; padding-left: 30px;">
                                    <div style="margin-right: 30px;">
                                        <div style="color: #c0c0c0">{{item.createTime}}</div>
                                        <div v-if="item.type === 'txt'" style="margin-top: 10px;">{{item.content}}</div>
                                        <div v-if="item.type === 'img'" style="margin-top: 10px;">
                                            <img :src="item.content" style="height: 100px"/>
                                        </div>
                                        <div v-if="item.type === 'music'" style="margin-top: 10px;">
                                            <audio ref="music" :id="'music' + index" controls preload="auto">
                                                <source :src="item.content" type="audio/mpeg">
                                            </audio>
                                        </div>
                                    </div>
                                </div>
                                <div>
```

```
                                        <span v-if="!item.icon">{{item.account}}</span>
                                        <img v-else class="friendUser" :src="item.icon"/>
                                    </div>
                                </div>
                            </div>
                            <div v-else>
                                <div style="display: flex; justify-content: start; background:
#FFFFFF; padding: 10px; padding-right: 30px;">
                                    <div>
                                        <span v-if="!item.icon">{{item.account}}</span>
                                        <img v-else class="friendUser" :src="item.icon"/>
                                    </div>
                                    <div style="margin-left: 30px;">
                                        <div                                    style="color:
#c0c0c0">{{item.createTime}}</div>
                                        <div                v-if="item.type                ===
'txt'">{{item.content}}</div>
                                        <div v-if="item.type === 'img'">
                                            <img    :src="item.content"    style="height:
100px"/>
                                        </div>
                                        <div v-if="item.type === 'music'" style="margin-
top: 10px;">
                                            <audio  ref="music"  :id="'music' + index"
controls preload="auto">
                                                <source                :src="item.content"
type="audio/mpeg">
                                            </audio>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div style="height: 100px;"></div>
                    </div>
                    <div style="height: 60px; display: flex; width: 100%; margin-left: 10px;">
                        <div style="flex: 1;">
                            <el-input placeholder="请输入内容" v-model="content"
                                      class="input-with-select"></el-input>
                        </div>
                        <div style="width: 60px;">
                            <el-button @click="sendHandler" size="mini" type="primary"
style="height: 40px"
                                       icon="el-icon-thumb"></el-button>
```

```
                        </div>
                        <div style="width: 60px;">
                            <el-upload
                                    class="avatar-uploader"
                                    action="/file/upload"
                                    :show-file-list="false"
                                    :on-success="handleUpload">
                                <el-button size="mini" type="primary" style="height:
40px"
                                           icon="el-icon-picture-outline"></el-button>
                            </el-upload>
                        </div>
                        <div style="width: 60px;">
                            <el-button @click="sendMusicHandler" size="mini"
type="primary" style="height: 40px"
                                       icon="el-icon-headset"></el-button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

左侧用户信息及聊天室导航栏部分：通过 flex 布局将这部分划分为包含用户头像展示、账号显示以及操作按钮（查看详情、退出登录）等区域，同时有 el-tabs 组件用于切换不同的聊天室列表视图（“聊天室中心” 展示所有可加入的聊天室，“我的聊天室” 展示用户自己创建的聊天室），每个聊天室以 el-card 组件展示，点击可进入对应的聊天室，并且在列表为空时通过 el-empty 组件显示相应提示。

用户详情展示及编辑部分：根据 showEdit 和 showDetail 等 Vue 实例中的数据属性值来控制显示内容，可展示用户的基本信息（头像、账号、个人简介等），并且在编辑状态下提供修改密码、个人简介等信息的输入框，还支持上传头像等操作，通过相关的按钮（编辑、返回、关闭等）实现不同视图的切换以及保存修改等功能。

聊天消息展示及发送部分：根据是否选择了具体的聊天室（通过 selectRoom 属性判断）来显示不同内容，若未选择则显示空提示（el-empty 组件），选择了聊天室后，展示聊天室名称以及聊天消息列表，消息列表根据消息类型（文本 txt、图片 img、音频 music）分别进行不同的展示（如图片消息展示图片、音频消息展示音频播放控件等），同时提供输入框用于输入文本消息以及按钮用于发送文本、图片、音频消息等操作。

3.1.4 对话弹出

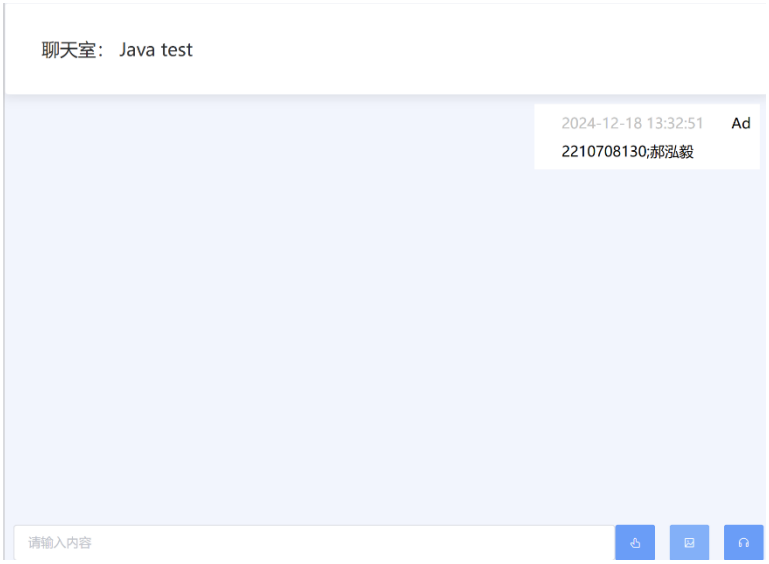

图 3-4 对话弹出

弹出对话部分：

```
<el-dialog
            title="sound recording"
            :close-on-click-modal="false"
            :visible.sync="sendMusic"
            width="600px"
            :before-close="cancelMusic">
        <div style="text-align: center; font-size: 30px;">
            录音中。。。
        </div>
        <span slot="footer" class="dialog-footer">
              <el-button @click="cancelMusic">关闭录音</el-button>
              <el-button type="primary" @click="handlerSubmitMusic">发送录音</el-button>
            </span>
</el-dialog>

<el-dialog
            title="新增聊天室"
            :visible.sync="showRoomFlag"
            width="600px"
            :close-on-click-modal="false"
            :before-close="() => {showRoomFlag = false}">
        <div>
            <el-form :model="form" :rules="rules" ref="form" label-width="12 0px"
class="demo-form">
                  <el-form-item label="聊天室名称" prop="name">
```

```
                    <el-input v-model="form.name"/>
                </el-form-item>
            </el-form>
        </div>
        <span slot="footer" class="dialog-footer">
                <el-button @click="() => {showRoomFlag = false}">关闭</el-button>
                <el-button type="primary" @click="handlerSubmit">确认</el-button>
            </span>
</el-dialog>
```

定义了两个 **el-dialog** 弹出对话框组件，分别用于录音操作（显示录音状态以及提供关闭录音、发送录音的按钮）和新增聊天室操作（包含一个输入框用于输入聊天室名称以及确认、关闭按钮用于提交或取消新增操作），通过相关的 Vue 数据属性（sendMusic、showRoomFlag 等）来控制对话框的显示与隐藏，并且在对话框的操作按钮上绑定对应的方法来实现具体功能逻辑。

3.2 数据库设计

SQL 语句：

```
SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;


-- ----------------------------
-- Table structure for comment
-- ----------------------------
DROP TABLE IF EXISTS `comment`;
CREATE TABLE `comment`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `content` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `account` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `room_id` int(0) NULL DEFAULT NULL,
  `type` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `create_time` datetime(0) NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 45 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
ROW_FORMAT = Dynamic;


-- ----------------------------
-- Table structure for room
-- ----------------------------
```

```
DROP TABLE IF EXISTS `room`;
CREATE TABLE `room`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `account` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 13 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for user
-- ----------------------------
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user`  (
  `id` int(0) NOT NULL AUTO_INCREMENT,
  `account` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `password` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `icon` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL DEFAULT NULL,
  `remark` text CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 12 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

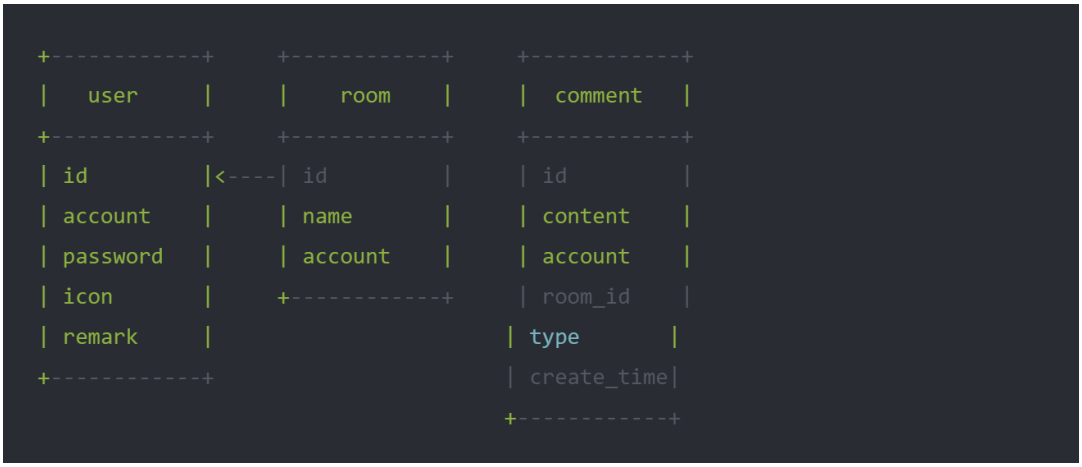

图 3-5 数据库 ER 图

3.3 后端设计

3.3.1 数据库链接

Application.yml：

```
spring:
  datasource:
      url:  jdbc:mysql://127.0.0.1:3306/manager?useSSL=false&useUnicode=true&characterEncoding=utf-8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&serverTimezone=GMT%2B8
    username: root
    password: 123456
    driver-class-name: com.mysql.cj.jdbc.Driver
  servlet:
    multipart:
      max-request-size: 1GB
      max-file-size: 1GB

server:
  port: 8888
```

jdbc:mysql://127.0.0.1:3306/manager: 表示使用 MySQL 数据库，连接本地的 127.0.0.1（也可以用 localhost）的 3306 端口上的 manager 数据库。
spring.datasource.username: 数据库用户名，当前配置为 root。
spring.datasource.password: 数据库密码，当前配置为 123456。
spring.servlet.multipart.max-request-size: 设置上传文件的最大请求大小。这里配置为 1GB，表示整个请求（包括文件和其他表单数据）不能超过 1GB。
spring.servlet.multipart.max-file-size: 设置上传的单个文件的最大大小。这里配置为 1GB，表示每个上传的文件不能超过 1GB。
server.port: 这是 Spring Boot 应用程序的 HTTP 服务器端口。这里配置为 8888，表示应用会监听在 http://localhost:8888 上。

3.3.2 comment 表和实体

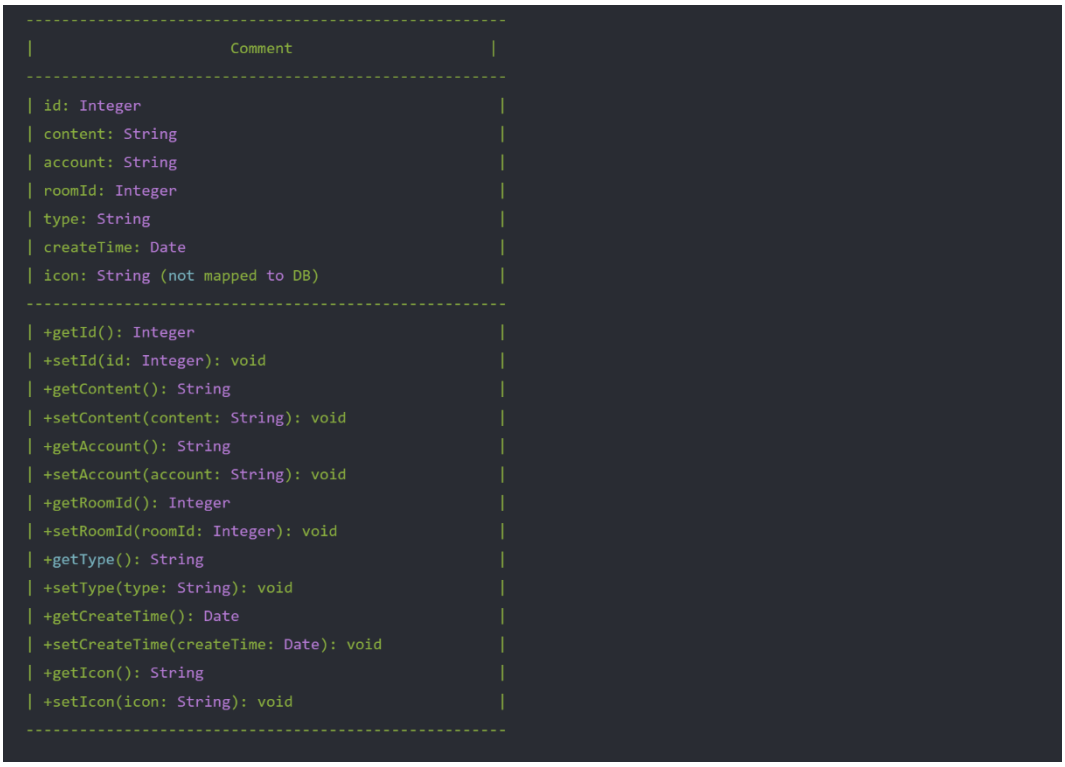

图 3-6 comment 表和方法结构示意图

```
package com.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.fasterxml.jackson.annotation.JsonFormat;

import java.util.Date;

public class Comment {
    @TableId(type = IdType.AUTO)
    private Integer id;

    private String content;

    private String account;

    private Integer roomId;

    private String type;
```

```
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")
    private Date createTime;

    @TableField(exist = false)
    private String icon;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public String getAccount() {
        return account;
    }

    public void setAccount(String account) {
        this.account = account;
    }

    public Integer getRoomId() {
        return roomId;
    }

    public void setRoomId(Integer roomId) {
        this.roomId = roomId;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
```

```
        this.type = type;
    }

    public Date getCreateTime() {
        return createTime;
    }

    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }

    public String getIcon() {
        return icon;
    }

    public void setIcon(String icon) {
        this.icon = icon;
    }
}
```

3.3.3 Room 表和实例

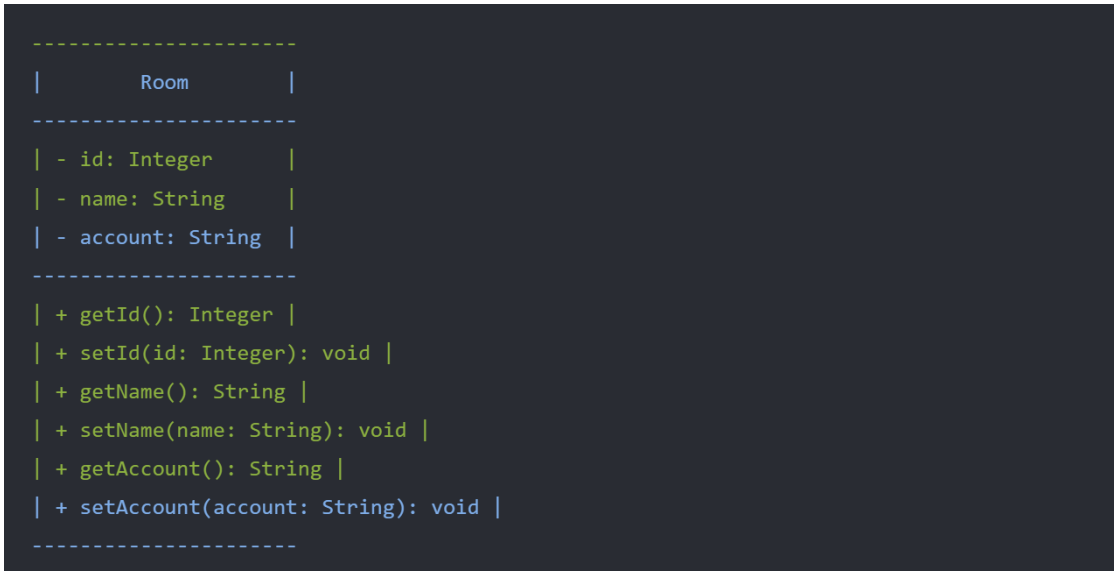

图 3-7 Room 表和方法结构示意图

```
package com.entity;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
```

```
public class Room {
    @TableId(type = IdType.AUTO)
    private Integer id;

    private String name;

    private String account;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAccount() {
        return account;
    }

    public void setAccount(String account) {
        this.account = account;
    }
}
```

3.3.4 User 表和实例

图 3-8 User 表和方法的结构图

```
package com.entity;

import com.baomidou.mybatisplus.annotation.IdType;

import com.baomidou.mybatisplus.annotation.TableId;

public class User{

    @TableId(type = IdType.AUTO)

    private Integer id;

    private String account;

    private String password;

    private String icon;

    private String remark;

    public String getRemark() {

        return remark;

    }

    public void setRemark(String remark) {

        this.remark = remark;
```

```
}
public Integer getId() {
    return id;
}
public void setId(Integer id) {
    this.id = id;
}
public String getIcon() {
    return icon;
}
public void setIcon(String icon) {
    this.icon = icon;
}
public String getAccount() {
    return account;
}
public void setAccount(String account) {
    this.account = account;
}
public String getPassword() {
    return password;
}
```

```
    public void setPassword(String password) {
        this.password = password;
    }
}
```

3.3.5 实体与数据库的映射

对象 - 关系映射（ORM）: 这是数据映射在软件开发中最常见的应用场景之一。在面向对象编程中，对象代表了应用程序中的实体（如用户、订单、产品等），而数据库使用关系模型（表、列、行等）来存储数据，本程序中，User 类，它有 id、aaccount、ppasswords、icon、remark 等属性。通过 spring boot 框架，可以将 User 类的对象映射到数据库中的 users 表。User 对象的属性对应 users 表中的列，这样就可以方便地将对象的状态保存到数据库中，或者从数据库中加载数据并构建对象。
使用 Spring boot 框架的

```
public interface UserMapper extends BaseMapper<User> {
}
```

定义了一个名为 UserMapper 的接口，它继承自 BaseMapper<User>，用于简化对 User 实体类对应的数据库表进行操作的代码编写，将通用的数据操作方法（如增删改查等）进行封装并继承使用，同时也方便在此接口上扩展特定于 User 表的自定义数据库操作方法。

```
public interface RoomMapper extends BaseMapper<Room> {
}
```

定义了一个名为 RoomMapper 的接口，它继承自 BaseMapper<Room>。旨在为对 Room 实体类对应的数据库表进行各类操作提供便捷的接口，方便地在此基础上扩展特定于 Room 表的自定义数据库操作逻辑，有助于构建清晰且高效的数据库访问层代码结构。

```
public interface CommentMapper extends BaseMapper<Comment> {

}
```

定义了一个名为 CommentMapper 的接口，它继承自 BaseMapper<Comment>。主要目的是为了方便对 Comment 实体类所对应的数据库表开展各种数据库操作，可以按需扩展特定于 Comment 表的个性化数据库操作逻辑。

3.3.6 控制器

3.3.6.1User 控制器

```
public class UserController {

    @Autowired
    private UserMapper userMapper;

    @GetMapping("/loginOut")
    public R loginOut() {
        AuthUtil.clear();
        return R.success("操作成功");
    }

    @GetMapping("/current")
    public R current() {
        User user = AuthUtil.getUser();
        return R.data(user);
    }

    @PostMapping("/register")
    public R register(@RequestBody User user) {
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(User::getAccount, user.getAccount());
        Long count = userMapper.selectCount(wrapper);

        if (count > 0) {
            throw new ServiceException("账号已存在");
        }

        user.setPassword(AuthUtil.codePassword(user.getPassword()));
        userMapper.insert(user);
        return R.success("注册成功");
    }

    @PostMapping("/login")
    public R login(@RequestBody User user) {
        AuthUtil.clear();

        user.setPassword(AuthUtil.codePassword(user.getPassword()));

        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
```

```
        wrapper.eq(User::getAccount, user.getAccount());
        wrapper.eq(User::getPassword, user.getPassword());
        User exist = userMapper.selectOne(wrapper);

        if (exist == null) {
            return R.fail("用户名或密码错误");
        }

        AuthUtil.setToken(exist);
        return R.success("登录成功");
    }

    @PostMapping("/update")
    public R update(@RequestBody User user) {
        userMapper.updateById(user);
        AuthUtil.setToken(user);
        return R.success("");
    }
}
```

UserController 类是一个 Spring MVC 中的控制器，用于处理与用户相关的多种操作，主要功能通过@Autowired 注入 UserMapper，以便借助其进行数据库操作。**/loginOut** 处理 GET 请求的/loginOut 路径，调用 AuthUtil.clear()清除相关认证信息后，返回操作成功的响应。获取当前用户**/current** 响应 GET 请求的/current 路径，调用 AuthUtil.getUser()获取当前用户对象，并将其封装在响应中返回。用户注册**/register** 处理 POST 请求的/register 路径，先利用 LambdaQueryWrapper 查询数据库中是否已存在相同账号，若账号已存在则抛异常提示；不存在则对密码加密后，通过 userMapper.insert 将用户信息插入数据库，最后返回注册成功的响应。用户登录**/login** 针对 POST 请求的/login 路径，先清除旧认证信息并加密密码，接着用 LambdaQueryWrapper 按账号和密码查询用户，若没查到返回登录失败响应；查到则设置令牌并返回登录成功响应。用户信息更新**/update** 响应 POST 请求的/update 路径，通过 userMapper.updateById 更新用户信息到数据库，再设置令牌，最后返回成功响应。

3.3.6.2room 控制器

```
@RestController
@RequestMapping("room")
public class RoomController {

    @Autowired
    private RoomMapper mapper;

    @RequestMapping("/self/list")
    @ResponseBody
    public R selfList(Room room) {
        LambdaQueryWrapper<Room> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(Room::getAccount, AuthUtil.getUserAccount());
```

```
        List<Room> list = mapper.selectList(wrapper);
        return R.data(list);
    }

    @RequestMapping("/list")
    @ResponseBody
    public R list(Room room) {
        LambdaQueryWrapper<Room> wrapper = new LambdaQueryWrapper<>();
        List<Room> list = mapper.selectList(wrapper);
        return R.data(list);
    }

    @RequestMapping("/delete")
    public R userDelete(@RequestParam("id") Integer id) {
        mapper.deleteById(id);
        return R.success("操作成功");
    }

    @RequestMapping("/save")
    public R save(@RequestBody Room room) {
        room.setId(null);
        room.setAccount(AuthUtil.getUserAccount());
        mapper.insert(room);
        return R.success("操作成功");
    }
}
```

RoomController 是一个带有@RestController 注解的 Spring MVC 控制类，映射路径为 room，主要用于处理与房间相关的操作。通过@Autowired 注入 RoomMapper，用于后续和数据库交互操作。获取用户自有房间列表（**/self/list**）：响应/self/list 路径的请求，创建 LambdaQueryWrapper 并设置条件，按照当前登录用户账号（通过 AuthUtil.getUserAccount()获取）查询对应的房间列表，再将查询结果封装在响应中返回。获取所有房间列表（/list）：针对/list 路径请求，创建 LambdaQueryWrapper 后直接查询所有房间列表，并将结果封装返回。删除房间**/delete** 处理/delete 路径请求，接收房间的 id 作为参数，调用 RoomMapper 的 deleteById 方法根据 id 删除对应房间，然后返回操作成功的响应。保存房间**/save** 响应/save 路径请求，接收 Room 对象作为请求体，先设置其 id 为 null 并设置账号为当前登录用户账号，接着通过 RoomMapper 的 insert 方法将房间信息插入数据库，最后返回操作成功的响应。

3.3.6.2comment 控制器

```
@RequestMapping("/comment")
public class CommentController {

    @Autowired
    private CommentMapper commentMapper;
```

```
    @Autowired
    private UserMapper userMapper;

    @PostMapping("/save")
    public R save(@RequestBody Comment comment) {
        comment.setAccount(AuthUtil.getUserAccount());
        comment.setCreateTime(new Date());
        commentMapper.insert(comment);
        return R.success("");
    }

    @GetMapping("/list")
    public R list(Comment comment) {
        LambdaQueryWrapper<Comment> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(Comment::getRoomId, comment.getRoomId()).orderByAsc(Comment::getCreateTime);

        List<Comment> comments = commentMapper.selectList(wrapper);

        for (Comment item : comments) {
            LambdaQueryWrapper<User> userWrapper = new LambdaQueryWrapper<>();
            userWrapper.eq(User::getAccount, item.getAccount());

            User user = userMapper.selectOne(userWrapper);
            item.setIcon(user.getIcon());
        }

        return R.data(comments);
    }
}
```

CommentController 是 Spring MVC 控制器类，映射路径为 /comment，负责处理评论相关操作，通过 @Autowired 注入 CommentMapper 与 UserMapper 实现与数据库交互，完成评论保存及按条件查询评论列表功能，并以 R 类型封装返回响应结果。

其中，CommentMapper 用于评论表相关数据库操作，UserMapper 辅助查询用户信息完善评论处理，二者配合完成评论业务逻辑的数据库交互。@PostMapping ("/save") 将 save 方法映射为处理 POST 请求（路径 /save）的接口用于接收客户端评论信息，@GetMapping ("/list") 把 list 方法映射为处理 GET 请求（路径 /list）的接口用于获取评论列表数据。

第 4 章　校园在线聊天软件结构图和流程图

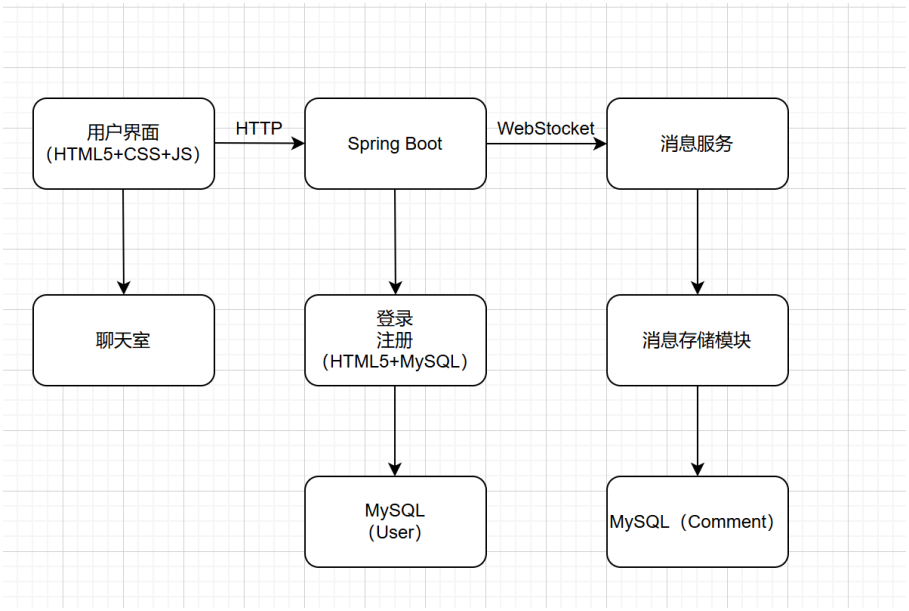

图 4-1 结构图

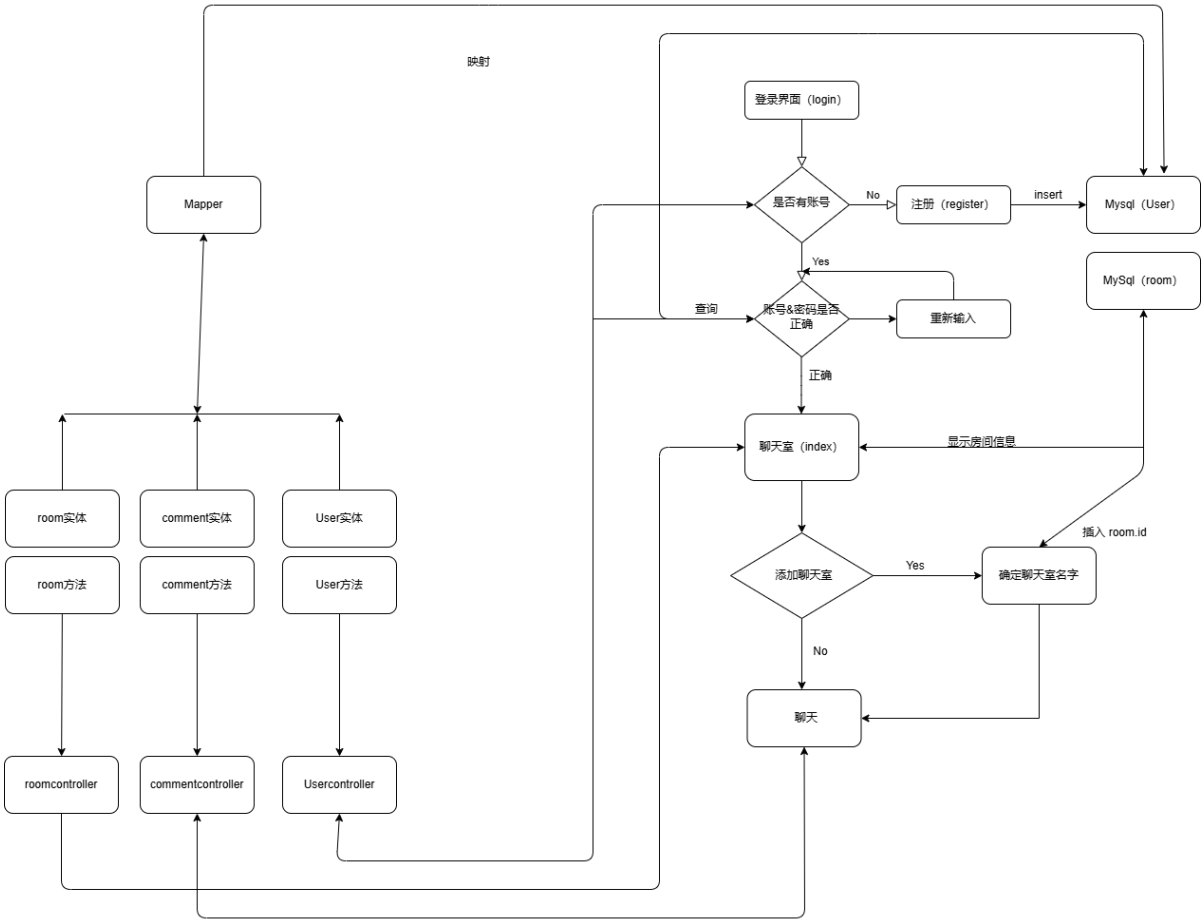

图 4-2 流程图

第 5 章　调试结果

5.1 错误调试

图 5-1 错误密码的测试

5.2 个人信息修改

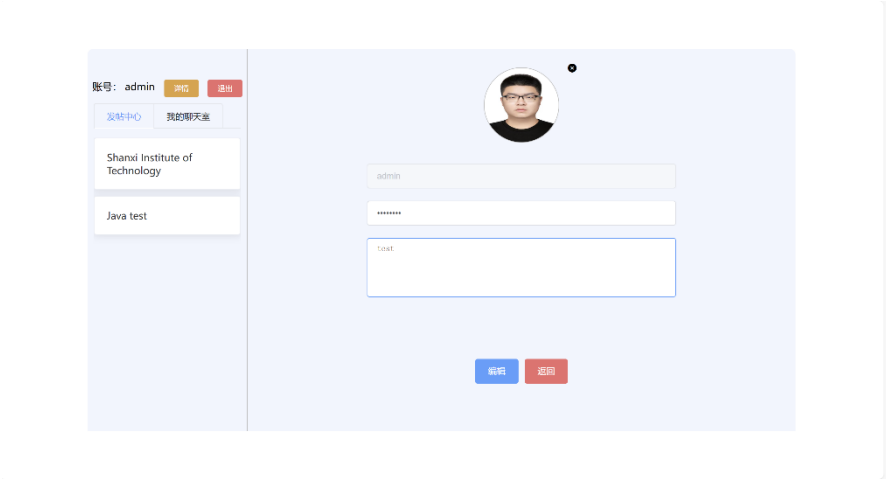

图 5-2 个人信息修改界面

确保界面能够正确加载当前用户的个人信息，用户名、邮箱、密码，个人简介等，并在用户修改信息后实时显示更新。在用户提交修改请求时，调试服务器端接收到的数据是否正确，确保信息正确传递给后端进行更新操作。检查数据库中相应字段是否成功更新，调试数据库查询和更新操作，确认没有 SQL 错误或数据丢失。

5.3 报错设计

```
package com.common.api;

import java.io.Serializable;

public class R<T> implements Serializable {
    private static final long serialVersionUID = 1L;

    private int code;

    private boolean success;

    private T data;

    private String msg;

    private R(IResultCode resultCode) {
        this(resultCode, (T)null, resultCode.getMessage());
    }

    private R(IResultCode resultCode, String msg) {
        this(resultCode, (T)null, msg);
    }

    private R(IResultCode resultCode, T data, String msg) {
        this(resultCode.getCode(), data, msg);
    }

    private R(int code, T data, String msg) {
        this.code = code;
        this.data = data;
        this.msg = msg;
        this.success = ResultCode.SUCCESS.code == code;
    }

    public static <T> R<T> data(T data) {
        return data(data, "操作成功");
    }

    public static <T> R<T> data(T data, String msg) {
        return data(200, data, msg);
    }
```

```
public static <T> R<T> data(int code, T data, String msg) {
    return new R(code, data, data == null ? "暂无承载数据" : msg);
}

public static <T> R<T> success(String msg) {
    return new R(ResultCode.SUCCESS, msg);
}

public static <T> R<T> fail(String msg) {
    return new R(ResultCode.FAILURE, msg);
}

public int getCode() {
    return this.code;
}

public boolean isSuccess() {
    return this.success;
}

public T getData() {
    return this.data;
}

public String getMsg() {
    return this.msg;
}

public void setCode(final int code) {
    this.code = code;
}

public void setSuccess(final boolean success) {
    this.success = success;
}

public void setData(final T data) {
    this.data = data;
}

public void setMsg(final String msg) {
    this.msg = msg;
}
```

```
    public String toString() {
        return "R(code=" + this.getCode() + ", success=" + this.isSuccess() + ", data=" + this.getData() + ",
msg=" + this.getMsg() + ")";
    }

    public R() {
    }
}
```

R<T>类位于 com.common.api 包下，实现了 Serializable 接口，主要用于封装操作结果并返回统一格式的响应信息，定义了如 code（状态码）、success（操作是否成功的标识）、data（承载的数据）、msg（提示消息）等属性。有多个私有构造方法，可根据传入的 IResultCode（结果码相关接口，文中未展示其完整定义）、状态码、数据以及消息等不同参数组合来初始化对象。**data(T data)**等一系列 data 开头的静态方法，用于便捷地创建包含数据的 R<T>对象，可指定状态码、数据以及消息内容，默认成功状态码为 200，无数据时消息有相应默认值。**success(String msg)**方法用于创建表示操作成功的 R<T>对象，使用默认成功对应的结果码等信息。**fail(String msg)**方法则用于创建表示操作失败的 R<T>对象，使用默认失败对应的结果码等信息。**属性访问方法方面**：提供了 getCode、isSuccess、getData、getMsg 等获取属性值的方法，以及 setCode、setSuccess、setData、setMsg 等设置属性值的方法，还有重写的 toString 方法用于方便地展示对象信息，同时有一个默认的无参构造方法。

public enum ResultCode implements IResultCode {

5.4 报错代码返回

```
 SUCCESS(200, "操作成功"),
    FAILURE(400, "操作失败")
    final int code;
    final String message;
    public int getCode() {
        return this.code;
    }
    public String getMessage() {
        return this.message;
    }
    private ResultCode(final int code, final String message) {
        this.code = code;
        this.message = message;
    }
}
```

表示操作结果的状态码及对应的消息，其中预定义了 SUCCESS 和 FAILURE 两种常见的结果情况，并且提供了获取状态码和消息的方法，以及对应的私有构造方法用于初始化对象。初步构建了一个用于管理操作结果状态码和消息的类结构，通过定义固定的结果表示、提供访问方法以及控制对象创建等方式

第 6 章　设计总结

使用 Spring Boot 框架搭建校园在线多人聊天室系统的设计，可以结合多种技术来实现其高效性、可扩展性和用户体验。常见的技术栈包括：Spring Boot：作为主框架，简化了项目配置和开发流程，自动化配置及快速构建 Web 应用。WebSocket：为了实现实时双向通信，WebSocket 是聊天室应用中不可或缺的技术，它允许服务器和客户端在打开连接后进行持续的数据交换。前端技术：前端使用 Vue.js 框架来构建用户界面，实现即时聊天的交互。数据库：MySQL 数据库，用于存储用户信息、聊天记录等数据。Redis：用于消息队列管理、缓存以及实时消息广播。Security：Spring Security 用于身份验证与授权，确保用户登录后的安全性。

系统采用了 前后端分离 的架构，前端通过 WebSocket 实现与后端的实时通信，后端通过 Spring Boot 提供 RESTful API 服务。整体架构分为以下几个层次：

1. 前端层：负责与用户交互，展示聊天界面，收发消息。
2. 后端层：管理连接，推送消息。业务层：实现具体的聊天业务逻辑，包括用户登录、聊天信息存储、聊天室管理等。数据访问层：与数据库交互，存储用户信息、聊天记录等数据。
3. 数据库：用于存储用户信息、聊天记录、群组信息等。

同时，本程序也可以实现消息历史记录与查询：消息存储：所有聊天记录会保存到数据库中（如 MySQL），包括时间戳、消息内容、发送者、接收者等信息。消息查询：用户可以查看自己的聊天历史记录，支持按时间、关键字等查询。可以对消息进行分页查询，避免一次性加载大量数据。

基于 Spring Boot 的校园在线多人聊天室设计，结合了现代 Web 开发技术，具备实时性、扩展性和高可用性。通过 WebSocket 实现高效的实时通信，配合 MySQL 和 Redis 提供稳定的后端支持，实现了基本的消息发送、群聊、私聊等功能。系统具有较好的扩展性，可以通过增加微服务、负载均衡等手段，进一步提升性能和可用性。

通过该设计，学生和教职工可以方便地进行实时的在线沟通交流，极大地提升了校园内的信息流通效率。

参考文献

[1]《Introduction to Algorithm》Thomas H. Cormen .Charles E. Leiserson . Ronald L. Rivest

[2] JDBC allows Java applications to connect to databases like MySQL, enabling seamless interaction with the underlying data."

— Java SE Documentation.

[3] Spring Boot simplifies database interaction by providing auto-configuration for JDBC and JPA, enabling you to quickly integrate MySQL in your applications.

[4]管纪文，刘大有。数据结构，高等教育出版社，1985

[5]沈泽刚.伞晓丽——Java 基础入门，清华大学出版社，2023

附录：源程序代码

```
package com.controller;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.common.api.R;

import com.common.utils.AuthUtil;

import com.entity.Comment;

import com.entity.User;

import com.mapper.CommentMapper;

import com.mapper.UserMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.*;

import java.util.Date;

import java.util.List;

@RestController

@RequestMapping("/comment")

public class CommentController {

    @Autowired
```

```
    private CommentMapper commentMapper;

    @Autowired

    private UserMapper userMapper;

    @PostMapping("/save")

    public R save(@RequestBody Comment comment) {

        comment.setAccount(AuthUtil.getUserAccount());

        comment.setCreateTime(new Date());

        commentMapper.insert(comment);

        return R.success("");

    }

    @GetMapping("/list")

    public R list(Comment comment) {

        LambdaQueryWrapper<Comment> wrapper = new LambdaQueryWrapper<>();

        wrapper.eq(Comment::getRoomId, comment.getRoomId()).orderByAsc(Comment::getCreateTime);

        List<Comment> comments = commentMapper.selectList(wrapper);

        for (Comment item : comments) {

            LambdaQueryWrapper<User> userWrapper = new LambdaQueryWrapper<>();

            userWrapper.eq(User::getAccount, item.getAccount());
```

```
            User user = userMapper.selectOne(userWrapper);
            item.setIcon(user.getIcon());
        }
        return R.data(comments);
    }
}
```

```
package com.controller;
import com.common.api.R;
import com.common.exception.ServiceException;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import java.io.*;
@RestController
@RequestMapping("file")
public class FileController {
```

```
@PostMapping("/upload")
public R upload(MultipartFile file) {
    InputStream in = null;
    OutputStream out = null;
    try {
        in = file.getInputStream();
        byte[] buffer = new byte[in.available()];
        in.read(buffer);
        String upload = System.getProperty("user.dir") + File.separator
                + "target" + File.separator
                + "classes" + File.separator
                + "static" + File.separator
                + "upload";
        upload += File.separator + file.getOriginalFilename();
        out = new FileOutputStream(upload);
        out.write(buffer);
        return R.data("/upload/" + file.getOriginalFilename());
    } catch (IOException e) {
```

```
            e.printStackTrace();
        } finally {
            if (in != null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (out != null) {
                try {
                    out.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
```

```
            throw new ServiceException("");
        }
    }
    package com.controller;
    import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
    import com.common.api.R;
    import com.common.utils.AuthUtil;
    import com.entity.Room;
    import com.mapper.RoomMapper;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.web.bind.annotation.*;
    import java.util.List;
    @RestController
    @RequestMapping("room")
    public class RoomController {
        @Autowired
        private RoomMapper mapper;
        @RequestMapping("/self/list")
```

```
@ResponseBody
public R selfList(Room room) {
    LambdaQueryWrapper<Room> wrapper = new LambdaQueryWrapper<>();
    wrapper.eq(Room::getAccount, AuthUtil.getUserAccount());
    List<Room> list = mapper.selectList(wrapper);
    return R.data(list);
}
@RequestMapping("/list")
@ResponseBody
public R list(Room room) {
    LambdaQueryWrapper<Room> wrapper = new LambdaQueryWrapper<>();
    List<Room> list = mapper.selectList(wrapper);
    return R.data(list);
}
@RequestMapping("/delete")
public R userDelete(@RequestParam("id") Integer id) {
    mapper.deleteById(id);
    return R.success("操作成功");
```

```
    }

    @RequestMapping("/save")

    public R save(@RequestBody Room room) {

        room.setId(null);

        room.setAccount(AuthUtil.getUserAccount());

        mapper.insert(room);

        return R.success("操作成功");

    }

}

package com.controller;

import com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;

import com.common.api.R;

import com.common.exception.ServiceException;

import com.common.utils.AuthUtil;

import com.entity.User;

import com.mapper.UserMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.*;
```

```
@RestController
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserMapper userMapper;
    @GetMapping("/loginOut")
    public R loginOut() {
        AuthUtil.clear();
        return R.success("操作成功");
    }
    @GetMapping("/current")
    public R current() {
        User user = AuthUtil.getUser();
        return R.data(user);
    }
    @PostMapping("/register")
    public R register(@RequestBody User user) {
```

```
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(User::getAccount, user.getAccount());
        Long count = userMapper.selectCount(wrapper);
        if (count > 0) {
            throw new ServiceException("账号已存在");
        }
        user.setPassword(AuthUtil.codePassword(user.getPassword()));
        userMapper.insert(user);
        return R.success("注册成功");
    }
    @PostMapping("/login")
    public R login(@RequestBody User user) {
        AuthUtil.clear();
        user.setPassword(AuthUtil.codePassword(user.getPassword()));
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(User::getAccount, user.getAccount());
        wrapper.eq(User::getPassword, user.getPassword());
        User exist = userMapper.selectOne(wrapper);
```

```
        if (exist == null) {
            return R.fail("用户名或密码错误");
        }
        AuthUtil.setToken(exist);
        return R.success("登录成功");
    }
    @PostMapping("/update")
    public R update(@RequestBody User user) {
        userMapper.updateById(user);
        AuthUtil.setToken(user);
        return R.success("");
    }
}
spring:
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/manager?useSSL=false&useUnicode=true&characterEncoding=utf-
8&zeroDateTimeBehavior=convertToNull&transformedBitIsBoolean=true&serverTimezone=GMT%2B8
    username: root
```

```
    password: 123456
    driver-class-name: com.mysql.cj.jdbc.Driver
  servlet:
    multipart:
      max-request-size: 1GB
      max-file-size: 1GB
server:
  port: 8888
```

评阅意见

评分标准 1、按时完成规定任务。（10 分） 2、格式规范，页面排版严格按照规范标准设计。（20 分） 3、观点正确，结构合理，包含项目规定的内容。（20 分） 4、掌握知识程度。（20 分） 5、综合应用知识能力，综合分析处理问题能力。（30 分）
评阅意见 评分
指导教师：　　日期：