2024--2025 学年第 1 学期 计算机科学与工程 学院

期末考试卷《计算机网络课程设计》

学号: 202231607206 姓名: 鱼洲 班级: 计算机科学与技术卓越班

题目：基于 WebScoket 的网络聊天室

评语：

装

成绩：

订

（考试题目及要求）

自选题目，应用计算机网络的基本原理、理论和相关技术，在企业级网络设计和搭建、网络嗅探器开发及协议分析、Socket 编程、简单网络管理工具的开发、TCP 可靠传输分析和演示工具、局域网文件传输工具实现等题目中自选一类题目，完成设计、实现和报告撰写。要求：课程报告撰写中，引用内容及成果要有标注，要综合应用计算机网络的相关理论、方法和技术；撰写规范，叙述清楚。

线

西北師範大學

计算机网络 课程设计报告

题　　目：基于WebScoket的网络聊天室

学　　院：计算机科学与工程学院

专　　业：计算机科学与技术

班　　级：卓越班

姓　　名：鱼洲

学　　号：202231607206

教师姓名：颉满刚

摘　要

随着互联网应用的快速发展，实时通信的需求日益增加，传统的 HTTP 协议无法满足高频实时消息的传递，而 WebSocket 技术提供了一种持久连接、低延迟的双向通信方式。本文设计并实现了一个基于 WebSocket 的多人在线聊天系统，旨在实现实时消息传递和在线用户管理。系统通过 WebSocket 实现用户之间的实时双向通信，支持广播消息和点对点私聊，且能够实时展示在线用户列表。当用户进入或退出时，系统会自动广播更新信息给其他用户。此外，系统能够有效管理连接状态、处理错误和重连机制，确保高可用性。本项目使用 Java 语言结合 Spring、Spring MVC 和 MyBatis 等框架进行后端开发，前端采用 Amaze UI 提供简洁直观的界面。通过本项目的实现，展示了 WebSocket 在即时通讯系统中的广泛应用潜力，尤其在多人聊天室和在线协作等场景中表现出其高效、低延迟的优势。

关键词：WebSocket；实时通信；在线用户管理；消息广播；Java

目　　录

第一章 绪论

1.1 开发背景

在这个信息化飞速发展的时代，互联网技术已经成为人类社会的基础设施，实时通讯更是成为了现代网络应用的核心需求之一。传统的基于 HTTP 协议的客户端与服务器之间的通信方式，虽然在许多场景下已经足够满足需求，但它的局限性也愈发显现。特别是在需要实时互动的应用场景中，如在线聊天、实时协作、股市交易等，HTTP 由于是单向请求-响应模式，无法及时处理大量频繁的数据交互。于是，WebSocket 技术作为一种新兴的、基于全双工通信的协议，逐渐成为解决这一问题的利器。

WebSocket 作为一种协议，通过在客户端与服务器之间建立持久连接，使得双方能够进行实时、低延迟的数据交换，不再依赖传统的请求-响应模式。自从其在 HTML5 中标准化以来，WebSocket 已被广泛应用于多种实时应用场景中，成为了即时通讯、在线游戏、在线客服、社交平台等领域不可或缺的技术。

与此同时，网络聊天系统作为最为常见的即时通讯工具之一，始终是 WebSocket 技术应用的热点。尤其是在用户数量日益增加、信息交互频率不断提高的今天，传统的即时通讯系统面临着越来越多的挑战：如何处理海量在线用户的请求，如何实现高效、低延迟的消息传递，如何在保证系统稳定性的前提下，提升用户体验，这些问题亟需解决。因此，本项目的设计目的在于基于 WebSocket 技术，构建一个高效、可靠的实时多人在线聊天系统。

1.2 开发意义

随着社会各行业数字化转型的深入发展，实时信息的需求愈加迫切。即时通讯技术不仅仅是人们日常交流的工具，它在企业协作、教育培训、客户服务等领域的应用，也日渐成为推动生产力和创新力的重要力量。而 WebSocket 技术的应用正好满足了这种需求，它通过保持客户端与服务器的持续连接，实现了低延迟、双向通讯，为用户带来更流畅的实时体验。

从技术的角度来看，WebSocket 技术的使用能够有效突破传统 HTTP 模型的限制，减少了服务器和客户端之间反复建立和断开连接的开销，提高了数据传输的效率。同时，得益于其全双工通信特性，WebSocket 能够在客户端和服务器之间实现即时信息的同步，使得实时消息推送成为可能。这对于提升系统性能，尤其是在高并发场景下，具有重要的实际意义。

对于开发者而言，基于 WebSocket 的开发不仅能加深对现代 Web 开发技术的理解，还能够锻炼其在高效通信和用户管理方面的编程能力。项目中的实时双向通信、用户在线状态管理、消息广播机制等功能，都是开发者常见的技术难点，对于

提升其解决复杂系统设计问题的能力具有显著作用。此外，项目的实现也为实际应用中的多用户即时通讯系统提供了一个可参考的解决方案，具有较强的实践意义。

在实际应用中，本项目所构建的聊天系统，能够同时支持数百甚至更多用户在线，实时传递信息并处理大量并发消息，从而确保在日常运营中的高效性与稳定性。无论是在小型团队协作，还是大规模的社交平台中，WebSocket 技术都能够为用户提供无缝的通讯体验，提升信息流转的效率和用户满意度。因此，基于 WebSocket 技术开发一个高效、稳定的即时通讯系统，不仅符合当前技术发展的趋势，也具备了广泛的社会需求和应用前景。

第二章 开发环境及关键技术

2.1 开发平台

本项目的开发平台主要基于 IntelliJ IDEA，这一强大的 IDE 为开发过程提供了极大的便利。作为一款专为 Java 开发设计的集成开发环境，IntelliJ IDEA 提供了强大的智能提示、代码补全、调试工具等功能，使得开发过程更加高效。对于后端技术的构建，它支持 Spring、Spring MVC 以及 MyBatis 等框架的整合，能够使开发者更专注于业务逻辑的实现。而前端的开发则通过浏览器直接实现，无需额外的编辑器或工具，简化了开发流程。

2.2 后端技术栈

后端部分，本项目采用了 Java 编程语言和广泛使用的 Spring 框架。Spring 框架以其灵活性、模块化和轻量级的特性，帮助开发者快速搭建复杂的企业级应用。在此基础上，Spring MVC 模块用于实现控制器层的逻辑处理，通过其精简的请求-响应模型，有效提高了 Web 应用的开发效率。

为了更好地处理数据存储和访问，本项目还使用了 MyBatis 作为持久化层框架。MyBatis 使得开发者能够以更精细的控制操作数据库，同时也能享受 SQL 映射和 ORM 框架带来的便利。在实现业务逻辑时，MyBatis 提供了高效且灵活的数据库访问能力，能够快速处理大量数据的增、删、改、查等操作。

2.3 数据库技术

在数据库技术方面，本项目采用了 MySQL 数据库，它作为广泛使用的关系型数据库系统，具有高效、可靠、易于扩展的特点。MySQL 的使用保证了数据存储和检索的高效性，并且通过合理的表结构设计，保证了数据库的灵活性和扩展性。

为了实现数据的持久化，本项目通过 MyBatis 框架与 MySQL 数据库进行集成，简化了数据操作流程。MyBatis 通过提供映射文件，让开发者能够精细控制 SQL 语句，避免了传统 JDBC 编程的冗长和复杂。此外，MySQL 数据库的事务管理和索引优化也进一步提升了系统在高并发条件下的稳定性和响应速度。

2.4 前端技术栈

前端技术的恰当选择，对于确保用户交互的流畅性和界面响应速度至关重要。在本项目中，前端采用了HTML5、CSS3和JavaScript作为基础技术栈，辅以Amaze UI 框架进行页面布局和响应式设计，确保了不同场景下的良好显示效果。

HTML5和CSS3提供了强大的置标语言和样式表功能，尤其是CSS3的动画效果、响应式布局和媒体查询特性，使得系统的用户界面既现代又富有层次感。同时，HTML5 的新增标签和 API，增强了前端页面的交互性和用户体验。

在JavaScript的实现上，项目使用了JS和AJAX技术，以确保页面无需刷新即可获取数据，从而提高了用户的操作流畅度。通过Fetch API，页面与后端服务器的通信变得高效且稳定，数据交互过程没有冗余延迟，极大地提升了用户体验。

此外，为了提高前端的开发效率和界面的美观性，项目中还使用了Amaze UI，这是一款基于HTML5和CSS3的前端框架，提供了丰富的UI组件和响应式设计，使得用户界面在不同设备上都能够良好适配。Amaze UI的使用有效缩短了开发周期，提升了系统的用户体验。

这些前端技术的巧妙融合，使得项目能够为用户呈现出一个既简洁直观又高度交互的操作界面，让顾客在享受餐饮服务的同时，深刻感受到数字化技术所带来的便捷与舒适体验。

2.5 WebScoket

WebSocket 是一种在单个 TCP 连接上进行全双工通信的协议，常用于实现实时 Web 应用。与传统的 HTTP 协议不同，WebSocket 能够在客户端和服务器之间建立持久的连接，使得双方可以在任意时刻进行数据交换，而无需每次都重新建立连接。这一特性使得 WebSocket 成为实时聊天、在线游戏、股票交易等应用的理想选择。

在本项目中，WebSocket 被广泛应用于聊天功能的实现。每当用户连接到服务器时，WebSocket 连接就被建立起来，服务器和客户端可以相互发送消息，无需频繁的 HTTP 请求响应。这种双向通信模式，极大地降低了延迟，提高了用户体验。尤其是在聊天应用中，信息传递的实时性至关重要，WebSocket 能够确保消息几乎即时到达目标用户，而不像传统的轮询方式那样浪费带宽和服务器资源。

WebSocket 连接的建立过程相对简单，客户端发起一个带有“Upgrade”请求头的 HTTP 请求，向服务器申请建立 WebSocket 连接。当服务器接收到请求后，会返回一个“101 Switching Protocols”响应，表明协议切换成功。此时，HTTP 连接会被升级为 WebSocket 连接，双方可以通过该连接发送和接收数据。此时，客户端与服务器之间的通信不再依赖于 HTTP 请求和响应，而是通过 WebSocket 协议进行数据交换，直到连接被主动关闭。

WebSocket 的优势不仅仅体现在低延迟和高效性，它的设计还考虑了实时性和可扩展性。例如，WebSocket 支持广播功能，能够在多个客户端之间广播消息而无需重复请求。为了确保系统的可靠性和性能，WebSocket 的实现通常需要配合线程池管理和消息队列等机制，优化并发处理能力，减少服务器压力。在本项目中，每当用户连接、断开连接或发送消息时，系统会进行动态管理，确保每一条消息都能实时到达指定用户，且在线用户的列表始终保持更新。

简而言之，WebSocket 在本项目中的引入，为系统提供了一个高效、实时的通信通道。通过这种技术，用户能够在不离开页面的情况下，体验到流畅的实时互动，系统的性能和扩展性也得到了显著提升。无论是系统的响应速度，还是对多用户并发的支持，WebSocket 都起到了至关重要的作用。

图 2.1 WebScoket 生命周期图

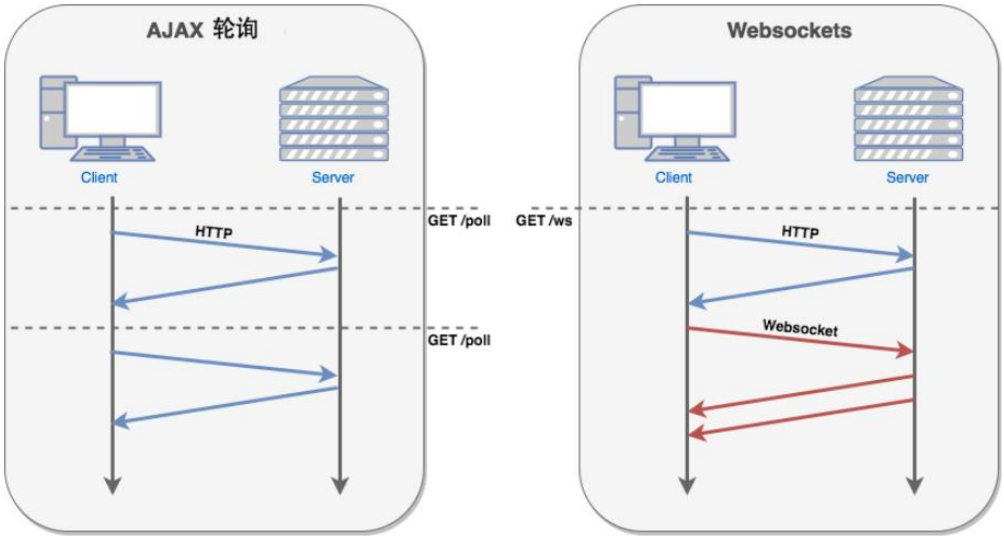

图 2.2 HTML5 定义的 WebSocket 协议

第三章 系统需求分析

3.1 登录注册

登录和注册功能是系统的基础，也是用户使用本聊天平台的入口。为了确保用户身份的唯一性和安全性，系统需要提供两种主要操作：注册和登录。在注册阶段，用户需要提供有效的邮箱和密码，系统会进行基本的输入验证，确保邮箱格式正确，密码符合安全要求（如长度、字符类型等）。一旦注册成功，用户便可以通过自己的账号和密码进行登录。登录时，系统将验证用户身份，如果验证通过，将允许用户访问个人主页及聊天功能。同时，为了提升用户体验，系统还将支持忘记密码功能，用户可以通过绑定邮箱来重置密码。考虑到安全性，密码将采用加密存储，以防止数据泄露和恶意攻击。

3.2 用户私聊

私聊功能是本聊天系统的核心之一，允许用户与其他单一用户进行一对一的实时通信。用户可以通过输入对方的用户名或者从在线用户列表中选择目标用户，发起私聊。系统会即时创建一个独立的对话窗口，用户可以在该窗口内进行文字、表情和其他附件的交流。每条消息将实时送达对方，并且系统会支持消息的发送顺序和接收顺序的准确显示。此外，系统也将提供消息已读功能，用户可以清楚地知道对方是否已阅读自己的信息。

为了确保私聊信息的保密性，系统将采用加密传输协议（如 TLS/SSL）来保障信息在传输过程中的安全，避免中间人攻击或者数据泄露。同时，考虑到系统的可扩展性，将允许未来在私聊功能中增加音视频通话、文件共享等扩展功能。

3.3 用户群聊

群聊功能是为多个用户提供协同交流的场所，适用于讨论、工作合作、兴趣小组等场景。用户可以根据自己的需求创建群聊，设置群聊名称、群主等基本信息，并邀请其他用户加入。群聊将支持多种互动方式，如文字聊天、图片分享、文件传输等。每个群聊中的消息都会广播给群内的所有成员，确保信息的即时传播。

3.4 用户信息修改

用户信息修改功能为用户提供个性化定制和自我管理的空间。在个人主页中，用户可以自由修改自己的基本信息，包括用户名、头像、个人资料等内容。为了提升用户体验，系统将允许用户上传自定义的头像，并提供多种预设的头像选择，避免用户在上传时遇到格式或大小的限制。同时，个人资料字段将支持用户填写更多的信息，如个人签名、兴趣爱好等。所有修改过的个人信息将在后台进行实时更新，并且立即反映到用户的个人页面中。

此外，用户信息的修改过程将受到严格的安全性保护。在用户修改敏感信息（如密码或绑定邮箱）时，系统会要求进行身份验证，确保修改请求来自于合法的用户。所有敏感信息的存储将采用加密算法，防止数据泄露或非法篡改。

3.5 其他需求

3.5.1 性能需求

系统的性能要求是确保在高并发情况下仍然能稳定运行。为了应对大量用户同时在线的情况，系统将采用高效的 WebSocket 连接管理机制，每个 WebSocket 连接都会分配一个独立的线程进行管理，并通过线程池来控制资源的消耗。消息的发送和接收过程将经过精心设计，确保消息的高效传输和处理，避免因流量过大导致的延迟或阻塞。此外，服务器会对每个连接进行负载均衡，确保系统在高负载状态下也能提供流畅的服务。

3.5.2 安全性需求

安全性是本系统设计中的重要考虑因素。所有的敏感数据（如用户密码、聊天记录等）都将采用加密技术进行保护。为了防止用户信息泄露，系统将严格控制用户数据的存储和访问权限，并采用 HTTPS 协议保障数据传输的安全。在 WebSocket 连接的建立过程中，系统将采用验证机制，确保只有经过认证的用户才能接入聊天服务，防止恶意用户进行攻击或干扰。同时，系统还将定期进行安全漏洞扫描和修复，保障平台的长期安全运行。

3.5.3 可用性需求

为了提高系统的可用性，系统设计中将考虑高可用架构，支持多节点分布式部署，确保在某一节点发生故障时，其他节点能够接管流量，保证服务不间断。系统的故障恢复能力将通过定期备份、日志记录和自动化运维等方式得到增强。除了硬件上的可用性保障，系统还会为用户提供友好的错误提示，确保用户在操作过程中能够快速理解问题并进行有效反馈。此外，系统还会在遇到意外情况时提供相应的故障恢复和重试机制，避免因小故障造成大规模的服务中断。

第四章 数据库设计

4.1 系统概念结构设计

在设计本系统的数据库时，我们首先要清晰地了解系统的核心功能及其数据交互方式。系统的设计要充分考虑用户信息的存储、聊天记录的管理以及消息的实时传输等需求。数据库概念结构设计将围绕以下几个关键实体进行：用户表、消息表、群组表、用户群组关联表、以及会话记录表。每个表都将承载着与用户行为密切相关的数据，并与其他表通过外键关系进行联接，从而保障数据的完整性和一致性。

通过建立用户表，系统能够有效地管理用户的基本信息，如用户名、密码、昵称、性别、年龄等字段。为了提升用户体验，还设计了个人头像和简介字段，以便用户能够更自由地展示个人资料。聊天功能则依赖于消息表和会话记录表，消息表负责存储用户发送的所有聊天信息，而会话记录表将记录用户与特定群体或其他用户之间的聊天会话，确保信息能快速而准确地找到。

4.2 系统逻辑结构设计

逻辑结构设计不仅仅是数据库表的布局，更是表与表之间如何通过关系纽带连接的设计。我们在本系统中采用了关系型数据库设计原则，确保所有的实体关系都能够高效地映射到数据库表中。用户表作为系统中最基础的部分，连接着消息、群聊、私聊等各个模块。每个用户都可以拥有多个消息记录，而每条消息都将关联到发送者和接收者的用户 ID。同时，为了支持多对多的群聊功能，我们引入了群组表和用户群组关联表，通过外键将用户与群组关联，确保群聊信息的统一管理。

4.3 数据库实施

数据库实施是将设计阶段的概念结构和逻辑结构转化为具体可执行的 SQL 语句和数据库对象。在本项目的数据库实施阶段，选择 MySql 作为数据库系统。下面将详细说明数据库实施的五个主要步骤：建库、建表、建索引、创建存储过程、创建触发器。

4.3.1 创建数据库

在数据库实施的第一步，我们需要创建数据库本身，以便在其上构建相关的表和其他数据库对象。通过执行创建数据库的 SQL 语句，系统能够在数据库管理系统中分配一个独立的空间，供所有数据表使用。

```
CREATE DATABASE chat_system;
```

图 4.1 数据库创建

4.3.2 创建数据库表

接下来，我们根据之前的设计，创建具体的数据表。以用户表为例，用户表将存储所有与用户相关的信息，包括用户名、密码、昵称、性别、头像等。在此表中，userid 是主键，确保每个用户都有一个唯一标识。

```
CREATE TABLE `user` (
  `userid` varchar(255) NOT NULL COMMENT '用户名',
  `password` varchar(255) NOT NULL COMMENT '密码',
  `nickname` varchar(255) DEFAULT NULL COMMENT '昵称',
  `sex` int(1) DEFAULT NULL COMMENT '性别',
  `age` int(5) DEFAULT NULL COMMENT '年龄',
  `profilehead` varchar(255) DEFAULT NULL COMMENT '头像',
  `profile` varchar(255) DEFAULT NULL COMMENT '简介',
  `firsttime` varchar(255) DEFAULT NULL COMMENT '注册时间',
  `lasttime` varchar(255) DEFAULT NULL COMMENT '最后登录时间',
  `status` int(1) DEFAULT NULL COMMENT '账号状态(1正常 0禁用)',
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

图 4.2 数据表创建

4.3.3 创建索引

索引的创建旨在提升数据库查询的效率，尤其是在数据量庞大的情况下，查询的速度和性能至关重要。在本系统中，我们为 userid 字段创建了索引，以提高在用户表中查询特定用户的效率。此外，针对其他查询频繁的字段，如消息表中的 sender_id 和 receiver_id，我们也会相应地建立索引，确保消息传输和检索更加高效。

```
CREATE INDEX idx_userid ON `user`(`userid`);
CREATE INDEX idx_sender_receiver ON `messages`(`sender_id`, `receiver_id`);
```

图 4.3 索引的创建

4.3.4 创建触发器

触发器是数据库中在特定事件发生时自动执行的程序，用于监控和响应数据的变化。为了保证用户信息的安全性和一致性，我们可以在用户表中创建触发器，在每次用户密码修改时，自动记录修改日志，或者在用户删除时，清理相关的历史消息数据。

```
CREATE TRIGGER before_user_delete
BEFORE DELETE ON `user`
FOR EACH ROW
BEGIN
    DELETE FROM messages WHERE sender_id = OLD.userid OR receiver_id = OLD.userid;
END;
```

图 4.4 触发器的创建

第五章 前台设计及功能实现

5.1 前台设计

前端设计是本系统实现中的核心部分之一，直接影响到用户的使用体验。为了提升用户交互体验，采用了前沿的前端技术，利用 HTML、CSS 及 JavaScript 构建页面，同时借助 Amaze UI 框架进行美化，确保页面能在各类设备上完美呈现。

5.1.1 登录注册页面

登录和注册页面是用户进入系统的第一步，界面设计简洁且直观。用户只需输入账号和密码即可登录系统，如图 5.1 所示。注册页面则允许用户创建新账号，除了输入必要的用户名、密码，还可以选择设置昵称、性别等信息，如图 5.2 所示。为增强用户体验，页面内加入了表单验证功能，确保用户输入的合法性。

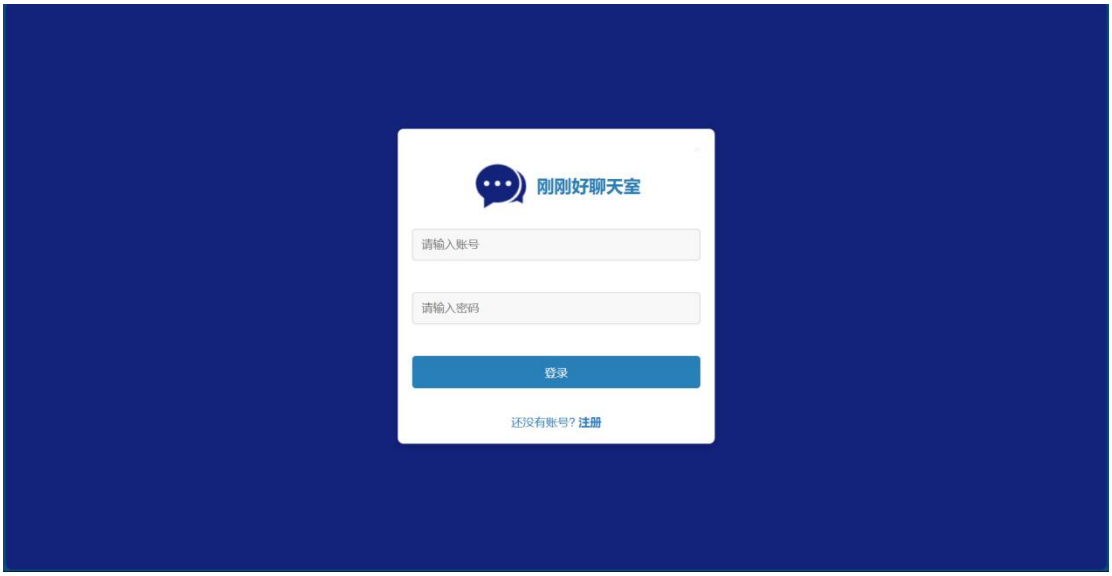

图 5.1 登录页面

图 5.2 注册页面

5.1.2 聊天界面

聊天界面是系统的核心，所有的即时通信都围绕这个页面展开。聊天界面采用了分栏布局，上部是消息展示区，下部是用户输入区。消息区中，用户的聊天信息以气泡形式显示，区分发送者与接收者。输入区则包含了文本框、清屏、发送等功能，极大丰富了用户的聊天体验。聊天信息即时展示，支持滚动查看历史记录。

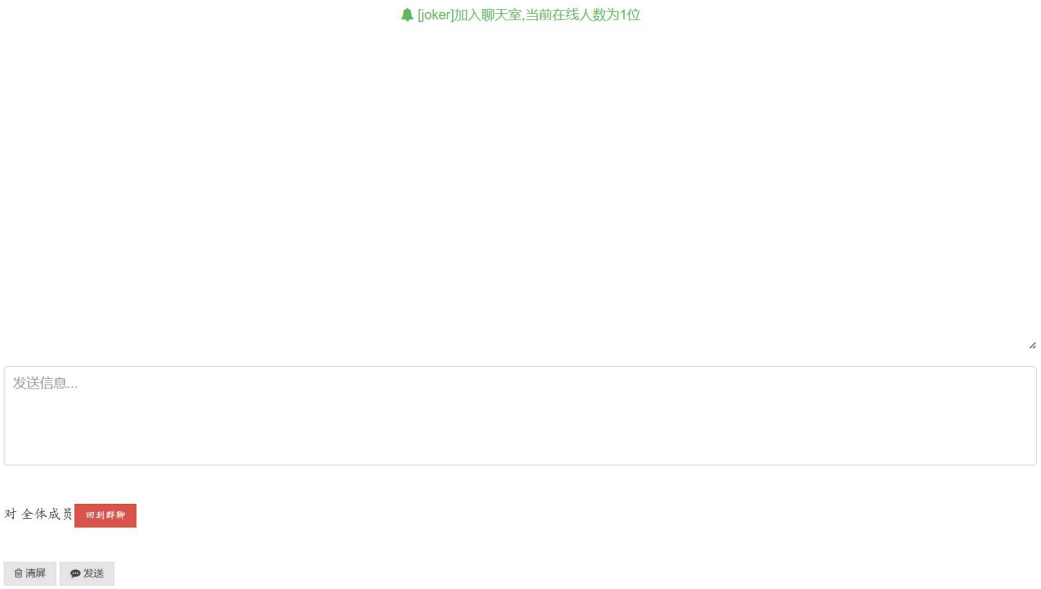

图 5.3 聊天页面

图 5.4 聊天页面

5.1.3 侧边栏菜单

为了提高操作的便捷性，侧边栏菜单提供了简洁且清晰的导航，帮助用户快速跳转至所需功能。侧边栏分为三个主要选项：聊天、个人设置和个人信息，并且在下方展示了当前在线列表，可以展示当前在线的用户昵称。每个选项点击后会展开相应的功能模块。侧边栏的设计采用了现代的图标和简洁的布局，使得页面操作不至于繁琐且更加直观。动态的侧边栏，增强了交互性，用户能够一键跳转至各个功能模块。每个功能模块均配有图标，直观易懂，符合用户习惯。并且实时更新在线人数和当前聊天状态，极大提升了信息的即时性。

图 5.5 侧边栏

5.1.4 个人设置

在个人设置页面，用户可以修改自己的基本信息，如昵称、头像、性别和简介以及密码等。所有更改的内容都会实时更新到数据库，并且在用户的个人主页上立即生效。此页面的设计不仅简洁，而且用户友好，配有表单验证，避免用户输入错误。头像上传功能，支持多种图片格式和尺寸，自由裁剪。设置页面设计简洁清晰，操作直观。

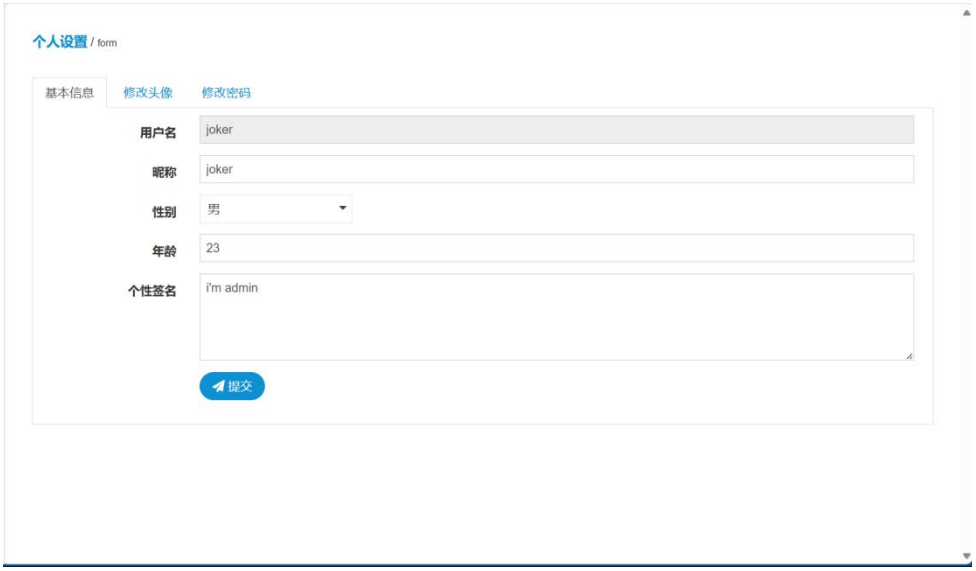

图 5.6 基本信息修改

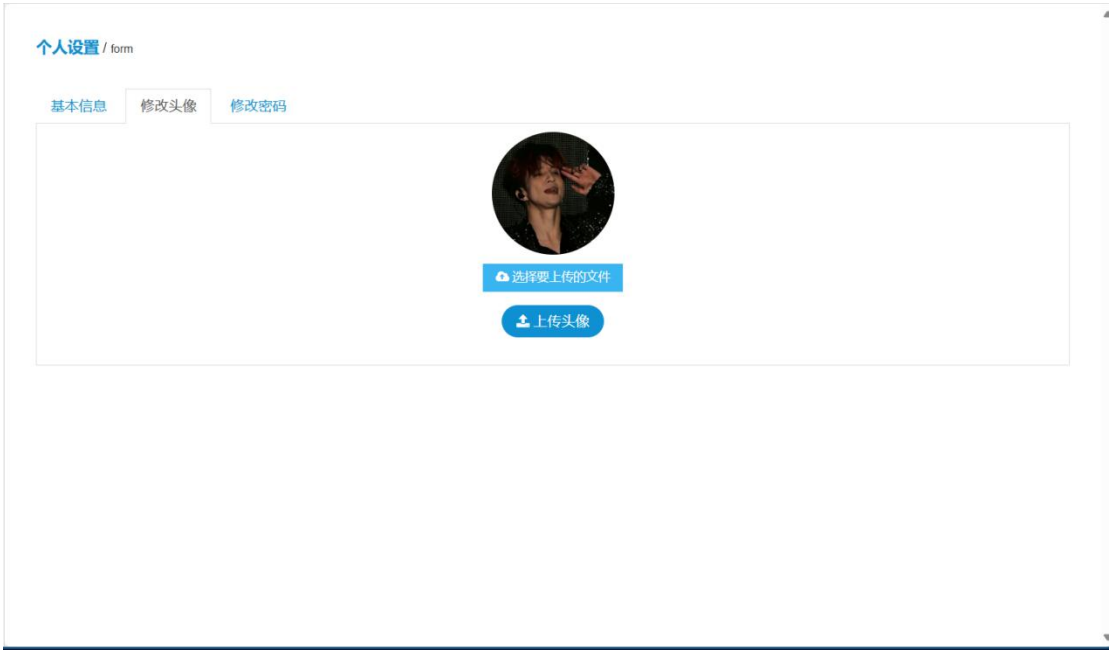

图 5.7 个人头像修改

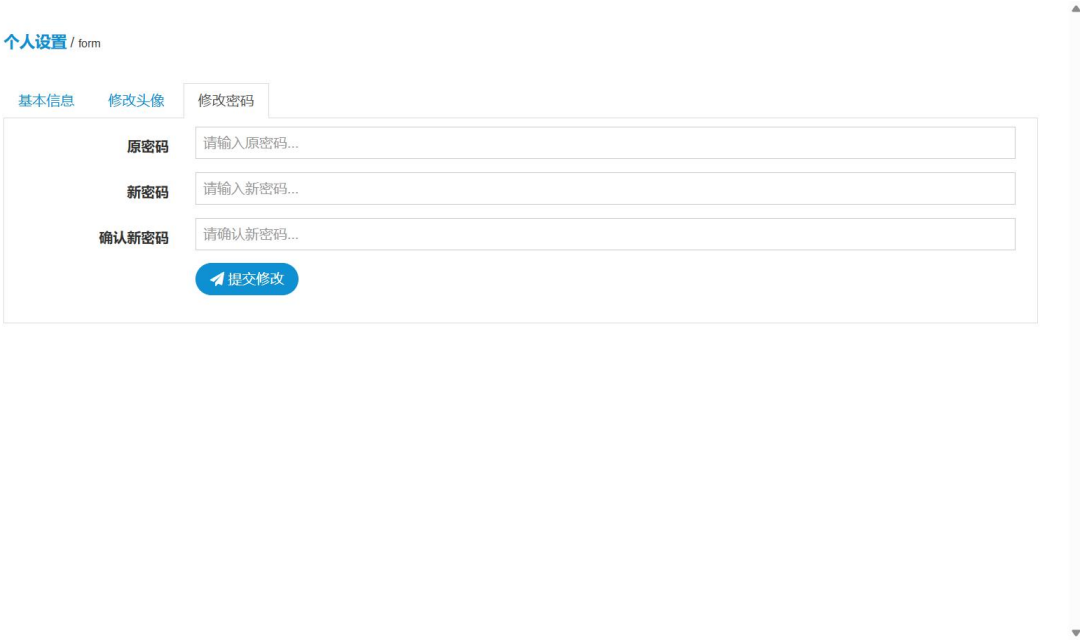

图 5.8 个人密码修改

5.1.5 个人信息

个人信息页面是展示用户所有基本信息的地方，包括用户名、昵称、性别、年龄、头像以及个人简介等。用户可以在此查看自己的资料，也可以通过“编辑”按钮跳转到个人设置页面进行修改。页面布局简洁、信息一目了然，避免了繁琐的操作步骤。

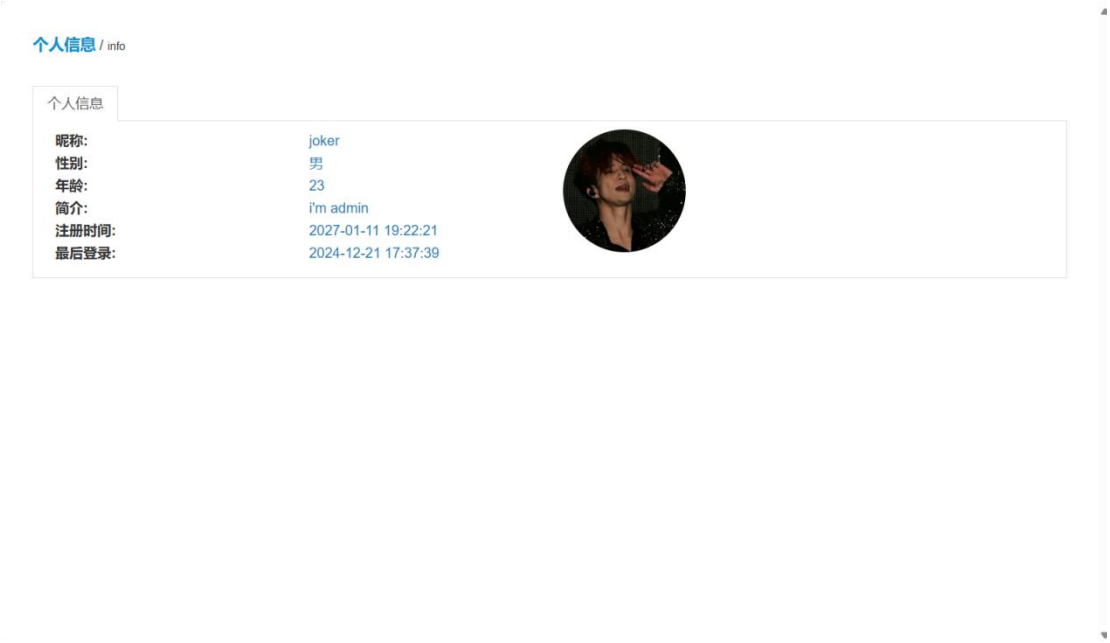

图 5.9 个人信息展示

5.2 前后台连接

前后台连接的核心目的是确保前端界面与后端数据的实时交互。在本系统中，我们通过 WebSocket 实现了与服务器的双向通信，前端页面通过 WebSocket 与后端建立持久连接，实现了即时消息传输。后端使用 Spring 框架配合 MyBatis 实现业务逻辑处理，数据操作和持久化，所有的数据交互都通过 RESTful API 与前端连接，确保前端获取最新的用户状态、消息内容和群组信息。

在前端与后端的连接中，最为关键的技术是 WebSocket，它不仅实现了即时的消息推送，还能够保持长连接，减少了频繁建立连接的开销，提高了系统的响应速度和用户体验。如图 5.10 所示，用户在前端界面上输入登录信息、消息内容或执行其他操作（如发送私聊消息、群聊消息等）。前端根据需要选择发送 WebSocket 或 HTTP 请求。如果使用 WebSocket，前端会建立一个长连接与服务器进行实时双向通信。如果是 HTTP 请求，前端通过 POST、GET 等方式与后端交互，适用于不需要实时通信的场景。

当请求到达后端时，WebSocket 服务器会接收到来自前端的消息或请求。对于 HTTP 请求，Spring MVC 控制器会处理请求，并将其传递给后端业务逻辑处理层。后端根据请求的类型做相应的处理，若是 WebSocket 消息，后端会解析并处理消息内容，例如发送私聊消息或群聊消息。若是 HTTP 请求，后端会验证登录信息、处理注册等请求，并返回相关数据（如用户信息、群聊信息等）。

后端处理完成后，会根据请求类型进行响应。对于 WebSocket，后端会通过 WebSocket 将响应发送给前端，内容包括聊天消息、系统提示或用户信息等。对于 HTTP 请求，后端会将处理结果通过 HTTP 响应返回给前端，通常以 JSON 格式传递数据，包含操作结果或用户信息。

前端接收到后端响应后，会解析数据并更新用户界面。例如，在聊天界面中展示消息内容、更新用户状态或展示群聊成员等。在 WebSocket 的场景下，

前端会实时接收新的消息或在线状态变化，并立即更新界面，确保用户看到其他用户的动态。与此不同，HTTP 请求通常是一次性的数据交互，处理完成后就不需要保持连接。

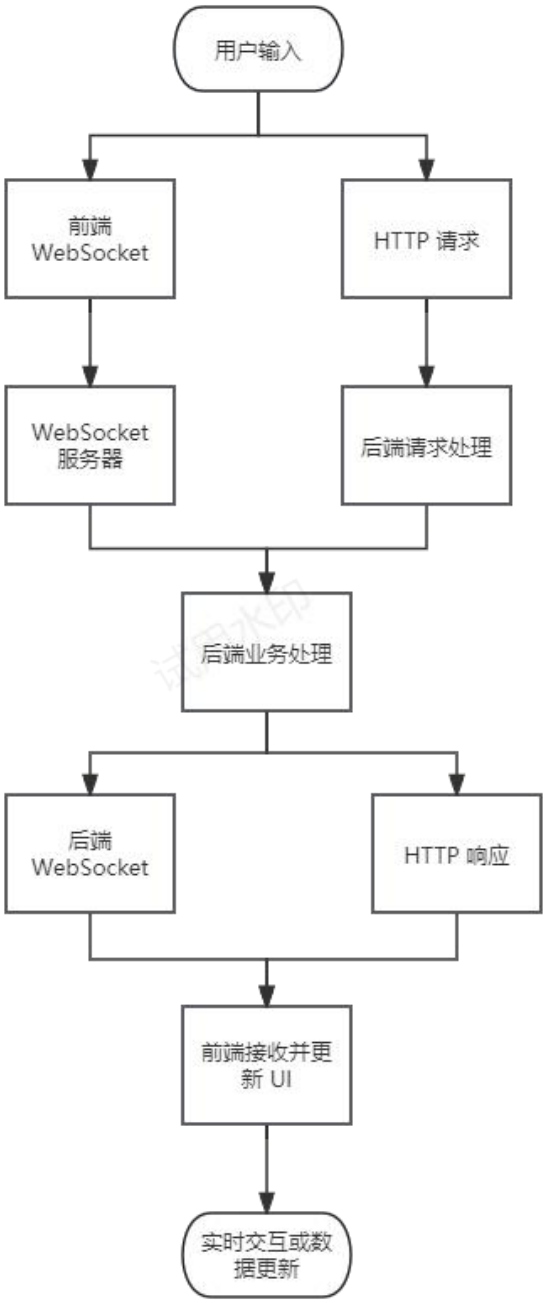

图 5.10 个人信息展示

5.3 功能实现及截图展示

5.3.1 登录注册功能

登录功能通过检查用户输入的用户名和密码，在后端与数据库进行匹配验证，若验证通过则允许用户进入系统。注册功能则会收集用户提供的基本信息，保存至数据库并完成注册流程。界面友好，操作简单直观。

首先，用户访问登录页面时，系统通过 `@RequestMapping` 注解映射到 `/user/login` 路径，采用 GET 请求方式显示登录页面。此时，前端呈现一个简单的登录表单，等待用户输入用户名和密码。当用户填写表单并提交后，前端通过 POST 请求将数据发送到后端。

登录处理逻辑复杂且层层把关。系统首先根据传入的 `userid` 查询数据库，验证用户是否存在。如果未找到对应的用户，则返回错误信息，提示用户输入的用户名有误。如果找到了该用户，系统会继续验证用户输入的密码是否正确。若密码不匹配，同样会返回一个错误提示，要求重新输入。在这里，系统的安全性体现在密码比对过程中——密码是从数据库中提取的加密值，而不是明文密码。

如果用户名和密码均验证通过，后端接着检查用户的状态是否为激活（状态码为 1）。如果用户被禁用，系统会及时反馈并中止登录流程。若所有条件都满足，系统进一步判断该用户是否已在线。此时，后端通过 `ServletContext` 获取全局应用对象来管理在线用户。若用户已在线，系统将会提示“已在线”并阻止再次登录，确保同一账户无法同时在多个地方登录。

通过 WebSocket 或 HTTP 进行的登录操作最终会被记录到日志中，详细记录每次登录操作的时间、IP 地址和具体的操作类型，这对于后期审计和排查异常至关重要。最后，用户的登录状态被保存在 `HttpSession` 中，并且数据库中的最后登录时间和 IP 也会被更新。

在退出登录时，用户可以通过访问 `/user/logout` 进行注销。系统通过 `session.removeAttribute` 方法清除会话中的登录信息，并将用户重定向回登录页面。用户登出时，系统会反馈一条“注销成功”的信息，提示用户操作完成。

注册流程则更为细致。在访问注册页面时，系统会返回注册表单。用户提交表单后，后端首先检查用户名是否已被注册，如果已存在，系统会返回错误，提示用户更换用户名。紧接着，系统检查用户输入的密码是否一致。如果两次密码不一致，注册请求也会被拒绝。

若一切顺利，后端会构造一个新的 `User` 对象，填充用户信息，如用户名、密码、IP 地址、激活状态等。重要的是，虽然代码中暂时注释了保存用户数据的操作，正常情况下系统会将新用户保存到数据库中，并返回一个提示，告知用户注册成功。用户在成功注册后，可以直接跳转到登录页面，进行登录操作。

整个流程不仅严谨，还考虑了多种边界情况，如用户名重复、密码错误、账号被禁用等，确保了系统的健壮性和用户体验的流畅性。

图 5.11 登录功能

图 5.12 注册功能

5.3.2 用户群聊功能

群聊功能是本系统的核心功能之一，用户可以加入群组，与群成员进行即时交流。群聊支持文字、图片、表情等多种信息传输方式，并支持群消息的回溯查看。每个群组内的消息都由服务器推送给群成员，确保信息的及时性。

群聊的核心在于如何管理用户连接、处理消息的发送与接收，以及广播消息的机制。当一个新的用户连接到聊天室时，onOpen 方法会被调用，这时系统会为该用户分配一个 WebSocket 会话，并将其加入到一个名为 webSocketSet 的集合中。这个集合用于追踪所有连接的 WebSocket 客户端，确保系统能够识别每个用户。用户的用户名会从 HTTP 会话中获取，并加入到在线用户列表中，形成一个完整的在线用户数据结构。

接下来，onClose 方法的触发确保当某个用户离开时，系统能够及时清理资源。用户的连接会被从集合中移除，在线用户列表会随之更新，并通过广播通知所有用户当前在线人数发生变化。这种处理方式确保了聊天室的实时性，始终保持每个用户的在线状态准确。

在处理消息时，onMessage 方法会根据用户发送的消息来判断其目标。如果消息没有指定接收人（即目标为空），系统会进行广播，将信息发送给所有在线的用户。反之，如果消息指定了接收人，系统会根据用户的列表，将消息发送给每个目标用户。此时，系统使用一个名为 routetab 的路由表来定位每个用户对应的 WebSocket 会话，确保消息能够准确到达目标用户。

消息的发送是通过 broadcast 和 singleSend 方法实现的。broadcast 方法用于广播消息，向所有连接的客户端发送信息，而 singleSend 则是为特定用户发送消息，无论是群聊中的一个目标用户，还是需要发送给自己的私聊信息。这些操作都依赖于 WebSocket 的 session.getBasicRemote().sendText(message) 方法来进行数据的传输。

在构建消息时，getMessage 方法负责将交互信息、消息类型和当前在线的用户列表封装成一个 JSON 格式的字符串，供前端进行处理。这样，聊天室不仅能够实现消息的即时传递，还能够实时更新在线用户列表，增强用户体验。

我们通过 WebSocket 实现了一个高效且灵活的群聊功能，能够实时处理用户的连接、消息广播和私聊信息的发送。系统通过共享的 webSocketSet 集合和 routetab 路由表来管理所有用户的 WebSocket 会话，确保消息能够准确地传达给每个用户，且每个用户的在线状态实时更新。

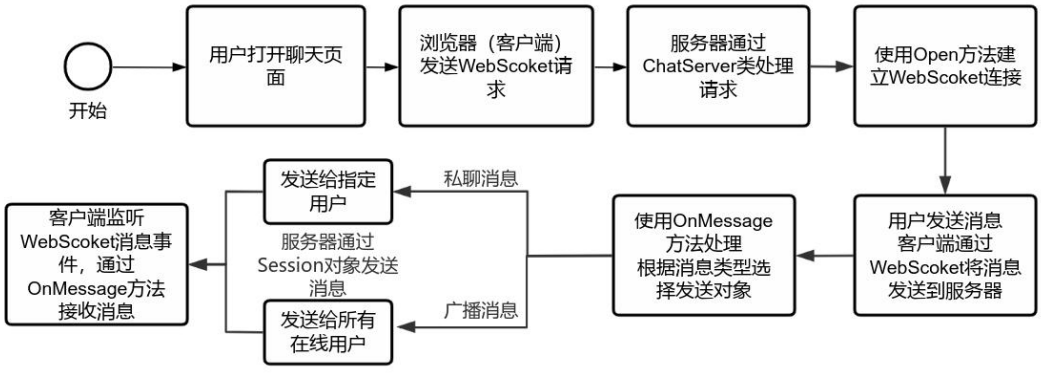

图 5.13 WebScoket 在聊天功能中的工作流程

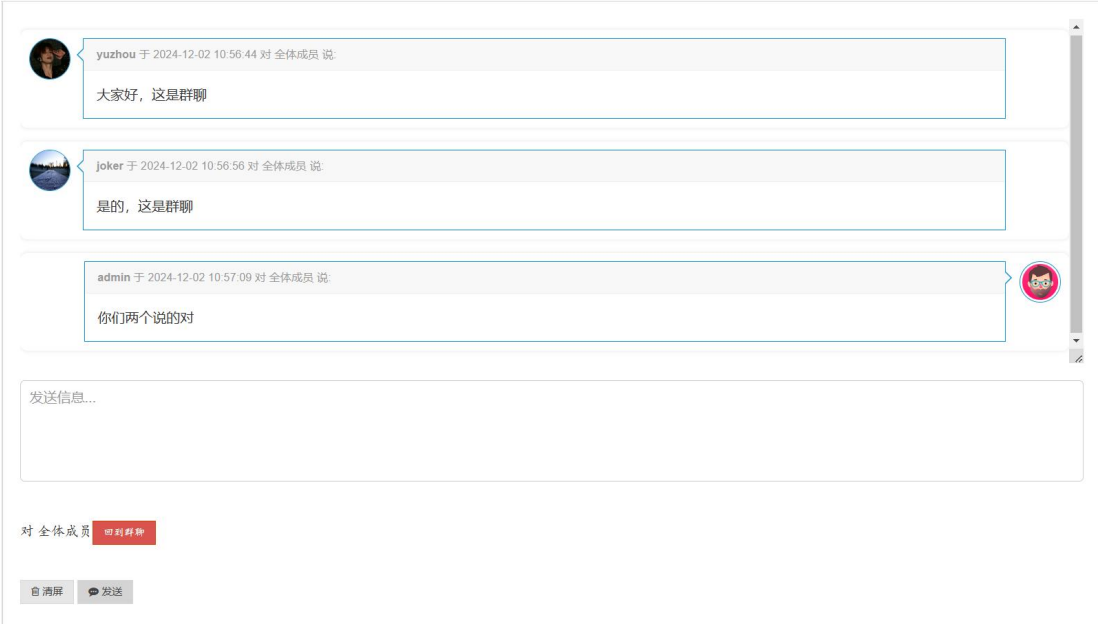

图 5.14 群聊功能

5.3.3 用户私聊功能

用户私聊功能允许两个用户通过点对点的消息传输进行交流。私聊窗口与群聊类似，但仅支持两人之间的消息传递。用户可以轻松查看对话记录，并发送文本、图片等消息。

用户的私聊功能通过 onMessage 方法中的逻辑来实现。该方法接收来自前端客户端的消息，并根据消息的内容判断是否是私聊消息，进而决定是进行群聊广播还是发送私聊消息。让我们从代码中解读私聊的实现细节。

首先，当客户端发送过来一条消息时，onMessage 方法会解析这条消息。消息体是一个 JSON 格式的数据，其中包含了消息的内容、消息类型、目标用户等信息。通过解析该消息，系统可以判断是否存在 to 字段，即是否指定了私聊的目标用户。

如果消息的 to 字段为空，说明用户希望发送的是群聊消息，系统会调用 broadcast 方法，将消息广播给所有在线用户。这个操作确保了群聊功能的实现。

然而，如果 to 字段不为空，表示这条消息是针对某个或某些用户的私聊消息。此时，系统会将目标用户列表中的每个用户名（通过逗号分隔）取出，并通过 routetab 路由表查找每个目标用户的 WebSocket 会话。routetab 是一个存储用户与其 WebSocket 会话映射关系的哈希表，它确保系统可以精确地找到每个目标用户的会话并将消息发送给他/她。

首先，消息会发送给发送者自己，确保发送者能够看到自己发送的消息（这对于前端的用户体验至关重要）。接着，系统会遍历 userlist 中的每个目标用户（从消息中的 to 字段获取），如果目标用户与发送者不同，系统便会通过 singleSend 方法将消息单独发送给这些目标用户。

singleSend 方法会根据每个目标用户的 WebSocket 会话（通过 routetab.get(user) 获取）来发送消息。此时，消息会通过 session.getBasicRemote().sendText(message) 这一 WebSocket API 被发送给指定用户。

整个过程中，WebSocket 连接的状态、用户的身份信息和消息的接收方都被巧妙地绑定在一起，确保私聊功能的顺利进行。通过这种方式，即便在群聊和私聊混合的情况下，系统也能准确无误地处理每个用户的消息。

私聊功能的实现依赖于以下几个关键点：首先是消息解析，确定消息的接收对象；然后是路由表 routetab，它存储了用户与其 WebSocket 会话的映射，确保每个消息能够精确送达目标用户；最后是 singleSend 方法，负责将消息发送给特定的用户。通过这些机制，系统能够高效地实现用户之间的私聊功能，同时保持聊天室的整体性能和实时性。

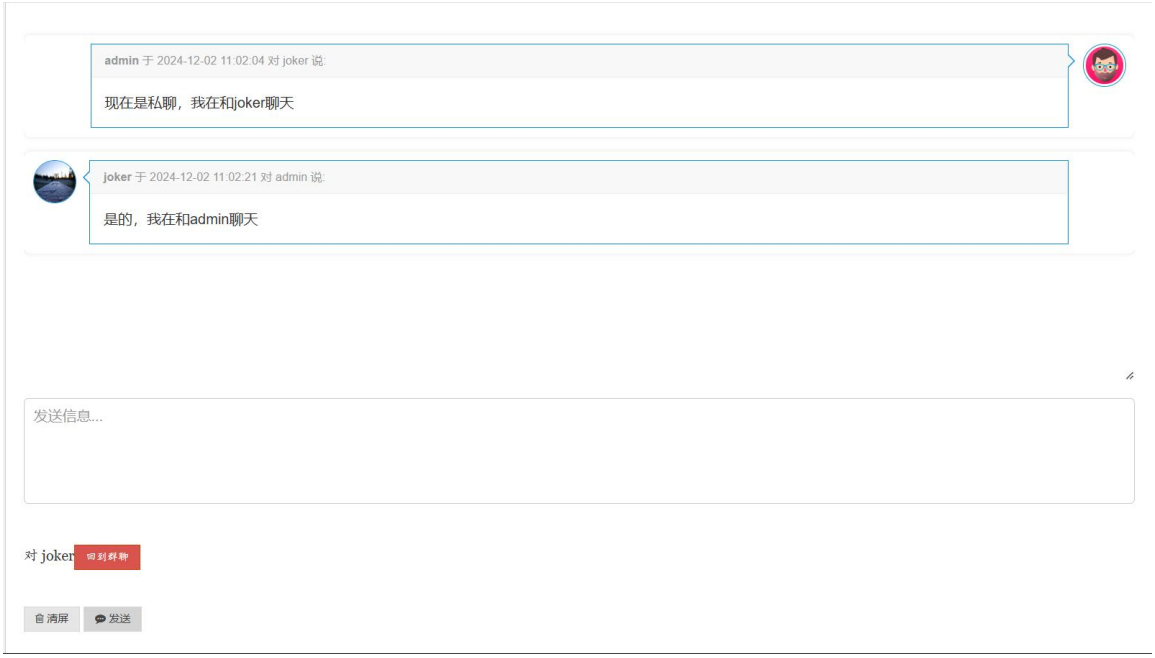

图 5.15 用户私聊功能

5.3.4 在线列表功能

在线列表功能可以实时展示当前在线的用户。用户进入系统时，所有在线的用户将自动添加至在线列表中，并能够看到每个用户的昵称和状态。系统会实时更新用户的在线状态，一旦某个用户下线，列表也会即时更新。

在用户成功连接到聊天室时，onOpen 方法会被触发。这时，系统会为新连接的用户分配一个 WebSocket 会话，并将该用户的用户名添加到 list 在线列表中，同时将其 WebSocket 会话与用户名进行绑定，存入 routetab 路由表。此时，所有在线用户的列表就会更新，并广播给所有连接的客户端，通知大家当前的在线人数和在线用户列表。

当一个用户连接时，系统将该用户的用户名添加到 list 中，同时更新在线人数，并广播一条消息给所有在线用户，内容包括新用户的加入和当前在线人数。这一操作确保了聊天室的在线列表始终是最新的，所有用户能够实时看到当前在线的用户及人数。

当一个用户离开聊天室时，onClose 方法会被调用。系统会从 list 在线列表中移除该用户，并从 routetab 路由表中删除用户和其 WebSocket 会话的绑定。随后，更新的在线人数和用户列表会再次通过广播通知所有在线用户。

用户离开时，系统会更新在线用户列表，并通过广播将这一变化通知所有用户，确保每个用户都能看到实时更新的在线状态。

此外，getMessage 方法用于构造和返回需要广播的消息内容，这条消息不仅包含聊天内容，还包括在线用户列表以及当前在线人数的变化。这样，每当有用户加入或离开时，系统就会自动生成并广播一条消息，通知其他用户。

broadcast 方法负责将消息发送给所有在线的客户端，它会遍历所有连接的 WebSocket 会话，并通过 session.getBasicRemote().sendText(message) 发送消息。所有在线用户都能及时接收到最新的在线列表和在线人数的变化。

在线列表功能的核心在于用户连接和断开时，实时更新在线用户列表，并将这一变化广播给所有在线的用户。通过 onOpen 和 onClose 方法，系统能够精确地管理每个用户的连接状态；通过 broadcast 方法，系统确保所有用户能够实时看到在线列表的变化。借助 routetab 和 list 数据结构，系统能够高效地管理在线用户的状态，实现聊天室中在线人数和用户列表的动态更新。

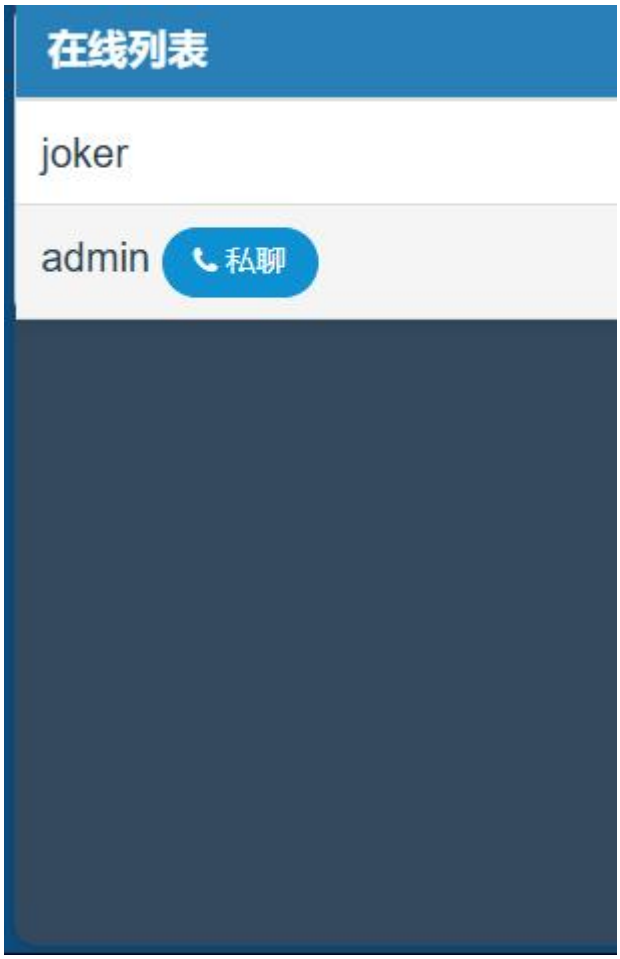

图 5.16 在线列表功能

5.3.5 用户信息修改功能

用户可以在个人信息页面中修改自己的基本资料，如昵称、头像、性别等。修改后的信息会及时更新，并在系统中同步显示。个人资料修改功能不仅提升了用户体验，也增强了系统的灵活性和可定制化。

首先，用户更新个人资料的功能通过 update 方法实现。用户可以修改自己的基本资料，这一过程首先需要从前端接收用户输入的更新信息，之后通过 userService.update(user) 方法将修改内容保存至数据库。如果更新成功，系统会通过 logService.insert 记录更新操作的日志，确保系统的操作可追溯性。与此同时，成功或失败的结果会通过 RedirectAttributes 传递给前端，并显示相应的消息，给用户反馈。成功时，提示信息中会表明“资料更新成功”，而失败时则会显示“资料更新失败”。

接下来是修改密码的功能。这一操作首先需要验证用户输入的旧密码是否正确。如果旧密码和数据库中存储的密码一致，则用户输入的新密码将被保存到数据库，并记录这次密码更新的操作日志。若旧密码输入错误，系统会向用户返回“密码错误”的提示。整个过程同样通过 RedirectAttributes 将结果反馈给用户。如果修改成功，系统会给予“密码更新成功”的提示，若失败，则反馈“密码更新失败”。

头像的上传功能则更加依赖于文件操作。用户上传头像文件时，系统首先通过 uploadUtil.upload 方法将文件存储到指定路径。然后，系统会更新用户的头像路径，并将新的路径信息存入数据库。如果头像上传和保存成功，系统将记录这一操作并给予成功提示。如果出现任何异常（如上传失败或文件存储错误），则会反馈错误信息给用户。头像的获取功能则通过 head 方法实现。该方法根据用户 ID 从数据库获取头像路径，然后通过 ServletOutputStream 读取图片文件，将其传送给客户端，完成头像的展示。

在整个过程中，系统的用户操作并非单纯的数据库交互，更多的是对用户行为的细致追踪与反馈。每一项操作，成功与否，都通过日志记录下其详细的执行情况。无论是资料更新、密码修改，还是头像上传，都结合了日志管理、文件处理和数据更新等多个模块。并且，所有的操作都在用户进行修改后，通过重定向的方式反馈给用户，确保操作的透明性和实时性。

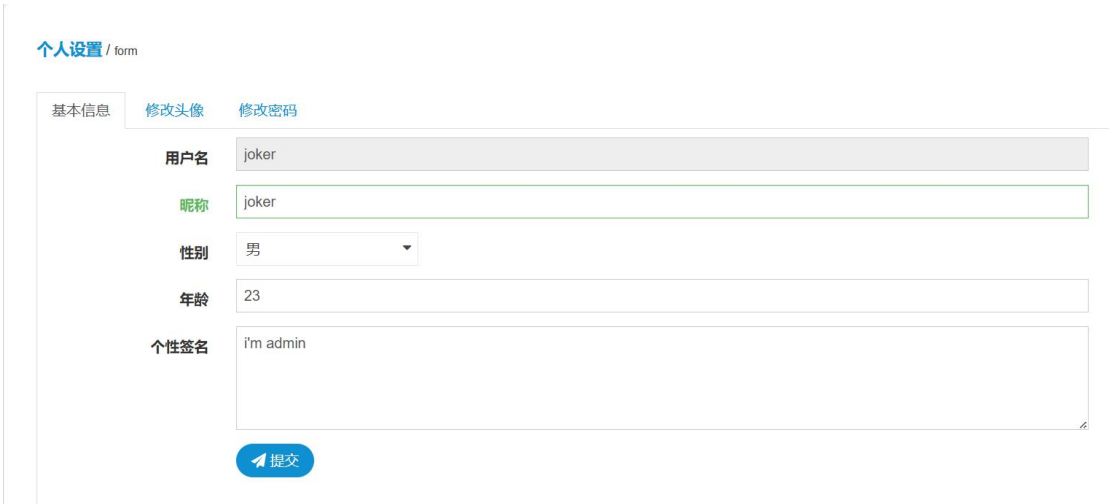

图 5.17 个人信息修改功能

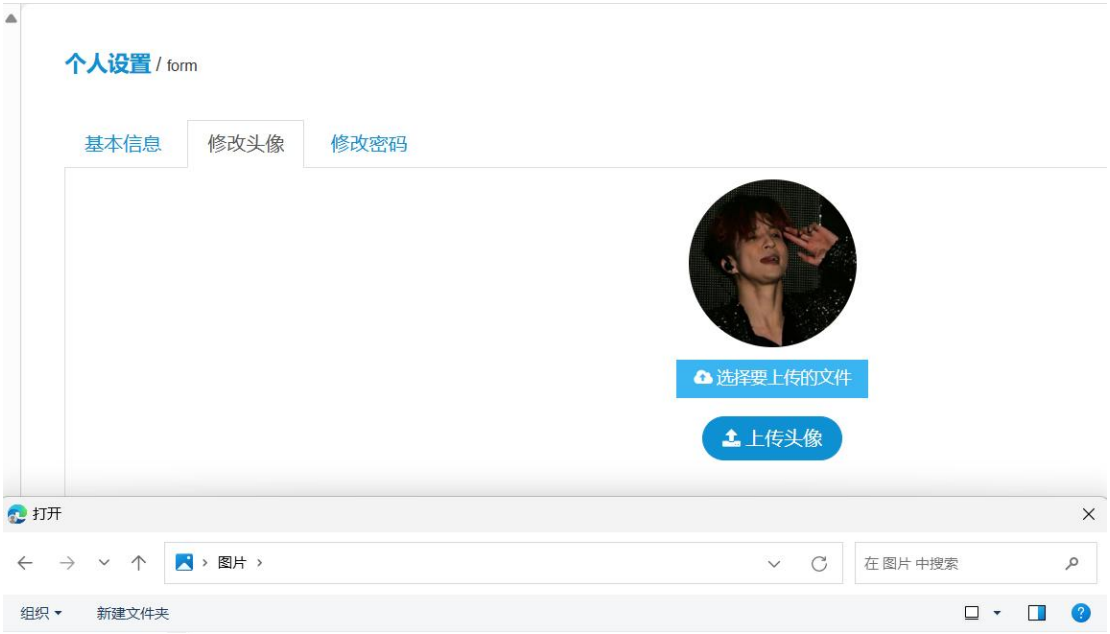

图 5.18 个人头像修改功能

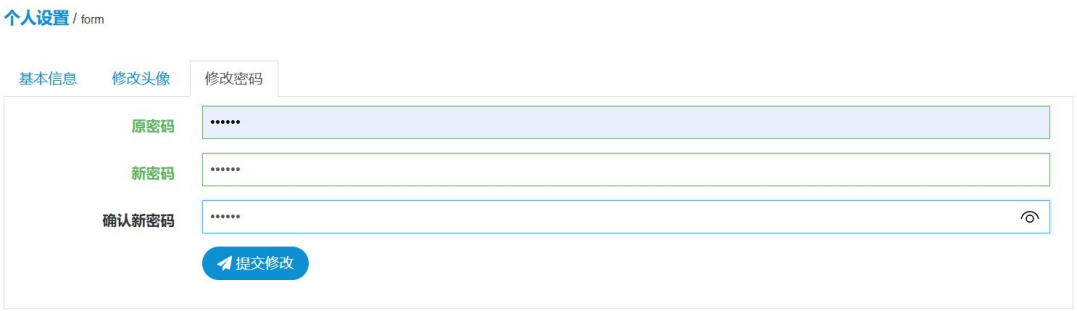

图 5.19 个人密码修改功能

第六章 总结

6.1 系统优点

经过一段时间的开发与测试，本系统无论在功能的完备性，还是在性能的优化方面，都展现出了显著的优势。首先，系统实现了高度的实时性，借助于 WebSocket 技术，用户可以进行即时的双向通信，信息的传递几乎无延迟，使得用户体验得到了显著提升。其次，系统支持用户群聊和私聊两种形式，满足了多样化的沟通需求。每当新消息到达时，客户端会立即通过 WebSocket 推送更新，大大降低了用户等待的时间。此外，用户信息修改和个人设置功能也体现了系统的灵活性，用户可以根据自身需求自定义个人资料，增强了互动性和个性化。

在设计层面，系统采用了简洁且直观的界面布局，不仅提升了用户的操作效率，也优化了视觉效果。无论是登录注册页面、聊天界面，还是个人信息修改页面，都经过了精心设计，力求让用户在最短的时间内找到所需功能并完成操作。另一方面，系统还考虑到了可用性与安全性，用户密码经过加密存储，账户安全得到了有效保障。

本系统最大的优点在于其高效的实时通信能力、灵活的用户管理和直观易用的界面设计，充分满足了用户即时沟通和个性化设置的需求，极大地提升了整体用户体验。

6.2 系统缺点

尽管本系统在许多方面表现优异，但也存在一定的不足之处，主要体现在以下几个方面：

首先，系统的扩展性不足。当前，系统在支持大量并发用户时可能会遇到性能瓶颈，尤其是在高并发的环境下，WebSocket 长连接的维护可能会导致服务器资源的消耗过大。为了保证系统能够顺畅运行，当前版本对并发的支持还显得有些薄弱。其次，虽然系统实现了基本的用户私聊和群聊功能，但在处理复杂的消息队列和消息持久化方面还略显不足。例如，当用户离线时，消息未能及时存储和恢复，造成了一定程度的信息丢失。最后，系统在安全性方面还有改进的空间。尽管用户密码使用了加密存储，但目前没有实现更高级别的认证机制，比如双因素认证或者验证码机制，这可能导致系统受到恶意攻击的风险。

此外，界面设计方面虽然简洁直观，但缺乏一些高级功能，比如聊天记录的导出、消息搜索等。这些功能的缺失在一定程度上影响了系统的便利性和可操作性。

6.3 改进方向

首先，性能优化与扩展性提升是当前版本的一个重要改进方向。目前，系统的性能主要受限于 WebSocket 连接的并发能力，尤其是在高并发场景下，WebSocket 长连接的管理可能会占用过多服务器资源。为了提高系统的承载能力和性能，可以引入消息队列中间件（如 Kafka 或 RabbitMQ）来分担消息传输的压力。此外，集群部署和分布式架构的引入也能有效提升系统的扩展性，使得系统能够在用户数量激增的情况下保持稳定运行。

其次，系统在消息持久化和恢复方面的功能尚显不足。目前，当用户离线时，系统无法保存未送达的消息，这容易导致信息丢失。为了解决这一问题，系统应加强消息的持久化机制，将离线期间的消息存储至数据库中，待用户上线后及时恢复。同时，增加历史聊天记录的查询功能，使用户能够方便地查找和管理过去的聊天内容，提升系统的实用性。

在安全性方面，虽然系统采用了基本的密码加密存储，但随着系统规模的扩大，安全性显得尤为重要。未来可以引入更强的身份认证机制，如双因素认证（2FA）和短信验证码，增强系统对恶意攻击的防护能力。除此之外，所有通信过程中的数据应使用 HTTPS 或 WSS 协议加密，确保用户数据在传输过程中不被窃取或篡改。

界面和功能的优化也是一个重要的方向。虽然当前的界面设计简洁直观，但随着用户需求的多样化，系统需要提供更多互动性和便捷性的功能。例如，聊天记录搜索功能的引入，将大大提升用户查找消息的效率。群聊和私聊功能的进一步优化，比如支持消息置顶、会话分类和自定义提醒等，能够满足不同用户的个性化需求，提升用户体验。

此外，考虑到当前版本主要是基于网页端设计，未来可以扩展到移动端平台，开发相应的移动应用，进一步满足用户随时随地在线交流的需求。跨平台的即时通讯功能将使用户无论身处何地，都能保持良好的沟通体验。

最后，随着聊天记录量的不断增加，如何有效管理大量消息成为了新的挑战。系统需要加入更强大的消息分类和筛选功能，让用户能够方便地按照不同标准（如关键词、时间段等）进行消息检索。同时，可以考虑为用户提供更丰富的消息管理选项，如批量删除、归档等，提高整体的操作便捷性。

尽管系统在现有的功能和性能上已经达到了较高的水平，但为了满足更高的用户需求和适应不断变化的技术环境，仍需要在多个方面进行优化和改进。这些改进不仅能够提升系统的稳定性和性能，还能极大丰富用户体验，使系统能够长久地适应用户日益复杂的需求。

参考文献

[1]郭毅,涂婧璐.基于 Web 页面网络实时消息推送的可靠性研究[J].电脑编程技巧与维护,2019,(04):158-160.DOI:10.16184/j.cnki.comprg.2019.04.057.

[2]丁立国,熊伟,周斌.Html5 WebScoket 对 Web Map 实时通信的影响[J].测绘与空间地理信息,2017,40(07):23-26.

[3]任琴,孔令慧,秦冰.基于 JAVA 的多人聊天室的系统设计[J].时代农机,2018,45(01):142.

[4]董克基.基于 SOCKET 的网络聊天系统分析设计[J].电子世界,2014,(16):189.