目录

[1. 绪论 1](#_Toc13460)

[1.1 课题背景 5](#_Toc26603)

[1.2 课题意义 5](#_Toc26819)

[1.3 相关技术概述 5](#_Toc30226)

[1.4 课题研究现状与发展趋势 6](#_Toc15694)

[2. 需求分析 6](#_Toc32274)

[2.1 功能需求 6](#_Toc3216)

[2.2 非功能需求 6](#_Toc29224)

[2.3 用户需求与系统需求 6](#_Toc17371)

[2.4 系统目标与约束 7](#_Toc7105)

[3. 总体设计 7](#_Toc6072)

[3.1 系统架构设计 7](#_Toc6735)

[3.2 系统模块划分 8](#_Toc12258)

[3.3 技术选型 8](#_Toc12434)

[3.4 系统设计原则与规范 9](#_Toc24055)

[4. 详细设计 9](#_Toc11356)

[4.1 系统功能模块设计 9](#_Toc20133)

[4.1.1 用户登录模块 9](#_Toc19407)

[4.1.2 消息发送模块 9](#_Toc17132)

[4.1.3 消息接收模块 9](#_Toc436)

[4.1.4 在线用户管理模块 10](#_Toc18739)

[4.2 类设计 10](#_Toc22182)

[4.2.1 服务器类设计 10](#_Toc29738)

[4.2.2 客户端类设计 10](#_Toc13557)

[4.2.3 数据传输设计 10](#_Toc9664)

[4.3 数据库设计 10](#_Toc31815)

[4.3.1 数据库设计原则 10](#_Toc30382)

[4.3.2 数据表结构设计 10](#_Toc25461)

[4.4 接口设计 10](#_Toc18274)

[5. 系统实现 11](#_Toc11609)

[5.1 服务端 11](#_Toc27102)

[5.1.1 服务器启动与监听 11](#_Toc18467)

[5.1.2 多线程处理客户端请求 11](#_Toc10826)

[5.1.3 在线用户管理 12](#_Toc9124)

[5.2 客户端 12](#_Toc21531)

[5.2.1 用户登录 12](#_Toc14677)

[5.2.2 消息发送与接收 13](#_Toc312)

[5.2.3 在线用户列表 13](#_Toc28815)

[5.3 图形用户界面（GUI）设计与实现 14](#_Toc1144)

[5.4 数据传输与通信协议 14](#_Toc15909)

[5.5 多线程机制实现 14](#_Toc8019)

[6. 系统测试 15](#_Toc17882)

[6.1 测试策略与方法 15](#_Toc22020)

[6.2 功能测试 15](#_Toc13427)

[6.3 性能测试 15](#_Toc24014)

[6.4 异常处理测试 15](#_Toc15930)

[6.5 测试结果与分析 16](#_Toc24545)

[7. 总结与展望 16](#_Toc17810)

[7.1 系统总结 16](#_Toc18241)

[7.2 存在的不足与挑战 16](#_Toc12769)

[7.3 未来扩展与改进方向 16](#_Toc19242)

8.参考文献

## 

# 1. 绪论

## 1.1 课题背景

随着互联网和局域网（LAN）技术的普及，越来越多的公司、学校和家庭开始依赖局域网内的即时通讯工具进行信息交流。在没有外部网络连接的情况下，局域网聊天室成为了一个重要的解决方案。对于不希望依赖互联网的环境，局域网内的实时通讯提供了可靠、安全的通信渠道。

目前，许多现有的聊天工具主要依赖互联网进行通信，但这些工具需要外部网络支持，并且可能存在一定的延迟和带宽限制。在局域网环境下，低延迟、无须外部网络连接的聊天系统越来越受到青睐。因此，本课题旨在设计并实现一款**基于局域网的即时通讯系统**，支持基本的群聊和私聊功能，适用于不依赖互联网的局域网环境。

## 1.2 课题意义

本项目的开发具有以下几方面的意义：

**局域网内的高效通讯**：用户可以在没有互联网的情况下通过局域网进行实时消息交流。该系统无需外部网络支持，避免了互联网延迟带来的不便。

**提升工作与学习效率**：在企业或学校环境中，局域网聊天室能够有效促进团队协作、信息交流，并帮助用户提高工作和学习效率。

**解决传统通讯工具的局限**：许多传统的即时通讯工具依赖外部网络，而本系统通过局域网实现更快的消息传递，且不受外部网络问题的干扰。

## 1.3 相关技术概述

本系统的实现依赖于以下关键技术：

**Java Scket编程**：Java的Scket编程提供了客户端与服务器之间的双向通信机制。本系统使用Java的Scket API，确保客户端与服务器之间的稳定连接。

**多线程编程**：多线程用于并发处理多个客户端的请求，每个客户端连接由一个独立的线程来处理，保证系统的高并发性能。

**Swing图形界面设计**：客户端采用Java Swing框架实现用户界面，提供直观易用的图形界面来支持用户登录、聊天等功能。

**自定义协议与数据传输**：系统采用简单的自定义协议进行消息传输和接收，以确保数据的稳定传输与处理。

## 1.4 课题研究现状与发展趋势

目前，局域网即时通讯工具多用于团队、学校或家庭等小型组织，广泛应用于办公、学习以及娱乐等领域。大多数现有的聊天软件，如企业内网聊天工具，依赖互联网或云服务，而局域网聊天系统仍有较大的市场需求，特别是在没有稳定外网环境的情况下。随着多媒体功能的增多，未来的局域网聊天工具将不仅支持文本消息，还会集成语音、视频和文件传输等多种功能。

# 2. 需求分析

## 2.1 功能需求

**用户登录与注册**：用户需要输入用户名和密码登录系统，支持新用户注册功能。系统能够对用户名进行验证，确保同一用户名不会重复。

**群聊功能**：用户可以在群聊中与所有在线用户进行信息交流。群聊功能支持消息广播，所有用户接收到相同的消息。

**私聊功能**：用户可以选择与特定在线用户进行私聊，私聊消息仅限于发送者和接收者。

**在线用户管理**：系统显示当前在线用户列表，用户可以查看所有在线用户并选择私聊对象。

**消息推送与接收**：系统支持消息的实时推送，确保群聊和私聊的即时性。服务器端负责消息的转发和推送。

**异常处理**：系统能够处理网络中断、客户端断开等异常情况，并保持系统的稳定性。

## 2.2 非功能需求

**性能要求**：系统应支持在同一局域网内多客户端并发连接，要求响应时间小于1秒，确保聊天过程的实时性。

**可靠性要求**：系统应在客户端断开连接或网络中断时，能够进行适当的处理和恢复，保证系统的稳定性。

**可扩展性要求**：系统设计应支持未来功能的扩展，支持更多的聊天功能，如文件传输、视频聊天等。

## 2.3 用户需求与系统需求

### 2.3.1用户需求：

简洁的登录界面，快速进入系统。

实时聊天功能，快速发送和接收消息。

在线用户列表，便于选择私聊对象。

### 2.3.2系统需求：

提供多用户支持，能够高效处理多个客户端连接。

支持群聊和私聊两种通信方式。

提供稳定的消息传递机制，确保系统的高可用性。

## 2.4 系统目标与约束

### 2.4.1系统目标：

构建一个简洁易用的局域网即时通讯系统。

实现高效的消息传递机制，保证群聊和私聊的实时性。

提供一个友好的图形用户界面，支持用户注册、登录及聊天功能。

### 2.4.2系统约束：

系统支持在局域网或外网运行，能跨越不同网络环境进行通信。

所有消息传递和通信都在开启服务器的主机网络内部完成。

# 3.总体设计

## 3.1 系统架构设计

系统采用**客户端-服务器（C/S）架构**，其中：

**服务器端**负责监听客户端的连接请求，处理消息的转发（包括群聊和私聊消息），管理员管理在线用户列表。

**客户端**提供图形用户界面供用户登录、发送消息、查看在线用户等操作。

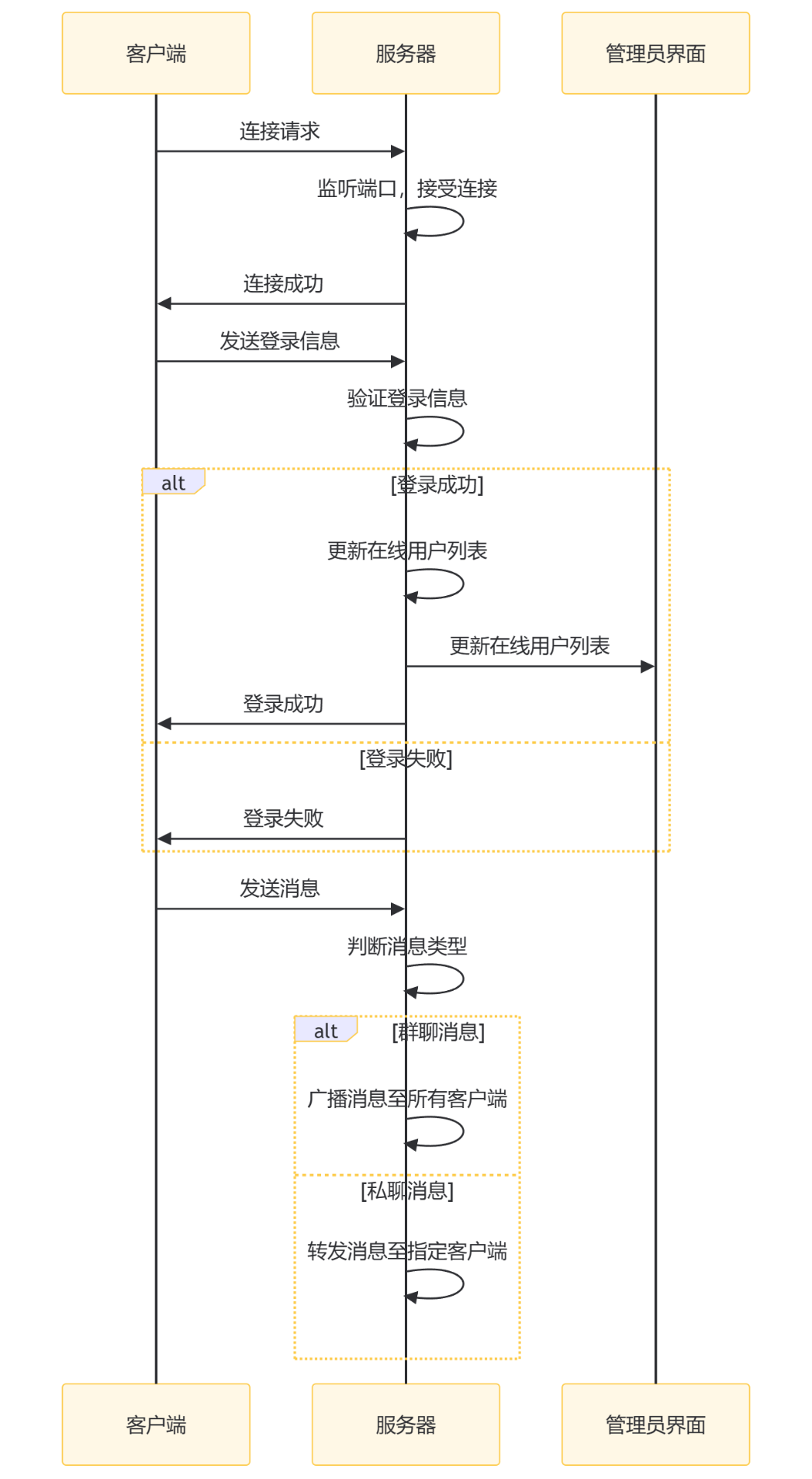
## 3.2 系统模块划分

**3.2.1服务器端模块**：

**连接管理模块**：处理客户端连接，维护在线用户列表。

**消息转发模块**：接收并转发消息（群聊和私聊）。

**在线用户管理模块**：跟踪用户登录状态，更新在线用户列表。

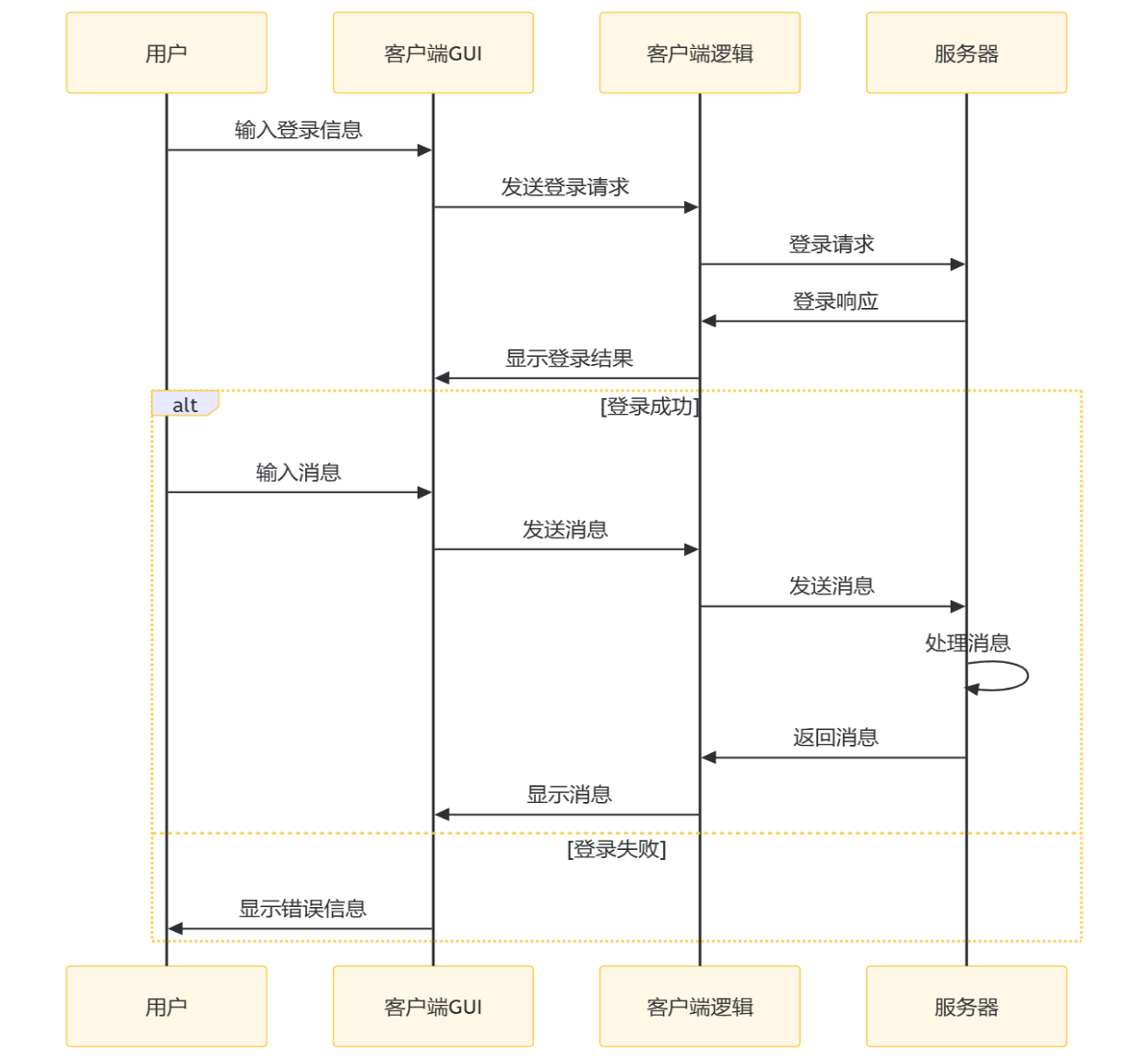


3.2.1服务端框架图

**3.2.1客户端模块**：

**登录与注册模块**：用户通过用户名和密码进行登录，支持新用户

注册。**聊天模块**：支持群聊和私聊消息的发送与接收。**在线用户管理模块**：显示所有在线用户，支持私聊选择。



3.2.1客户端架构图

## 3.3 技术选型

**开发语言**：Java，用于客户端和服务器端开发，具备良好的跨平台性。

**网络通信**：使用Java的Scket编程来实现客户端与服务器之间的通信。

**图形界面**：客户端使用Java Swing库进行GUI设计，提供直观易用的用户界面。

**数据库**：服务器使用txt格式文件存储账户密码信息。

**多线程**：使用多线程技术，确保服务器能够同时处理多个客户端的请求。

## 3.4 系统设计原则与规范

1.**模块化设计**：系统功能进行模块化划分，确保每个模块职责单一、易于维护。

2.**高可用性**：设计容错机制和异常处理机制，确保系统在异常情况下能够恢复正常工作。

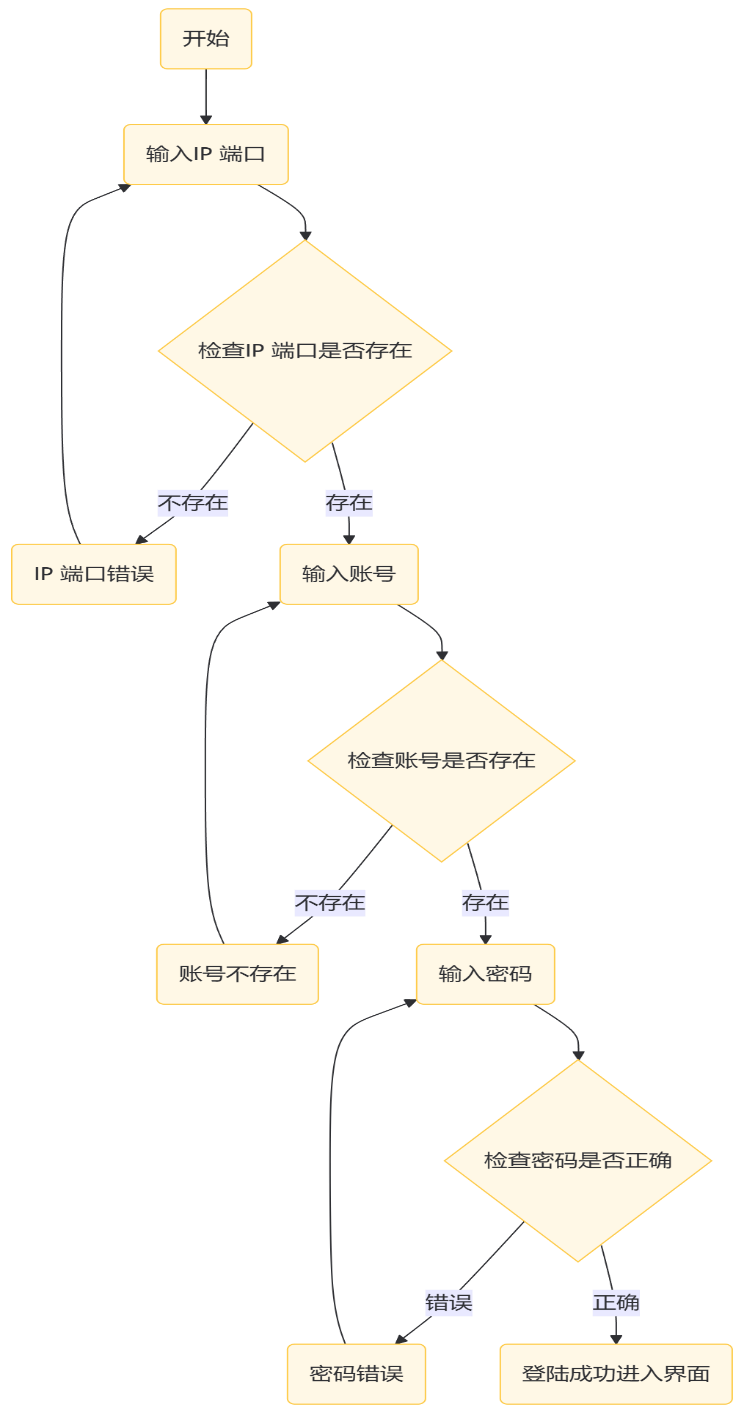
3.**易用性**：用户界面简洁，操作直观，尽量避免复杂的设置与操作。

# 4. 详细设计

## 4.1 系统功能流程设计

### 4.1.1 用户登录

该模块负责用户的身份验证。用户通过输入用户名和密码登录，如果用户未注册，系统会提供注册功能。



4.1.1用户登录流程图

### 4.1.2 用户注册

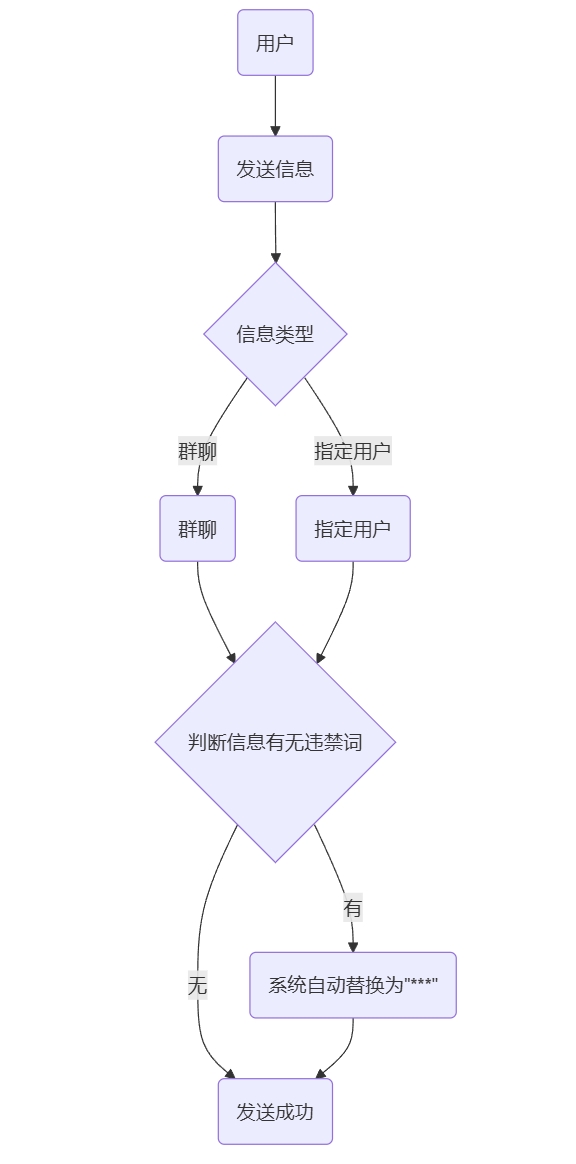
### 该模块的设计提供了一个简洁的界面，让用户能够输入用户名、密码以及确认密码，系统随后会进行验证，确保密码一致性，并用txt文件存储账号密码，同时检查用户名的唯一性，以保障账户安全和数据的准确性。

### 

4.1.2用户注册流程图

### 4.1.3 消息发送

用户在聊天界面输入消息后，消息将通过网络发送到服务器。服务器根据消息类型（群聊或私聊）进行消息转发。



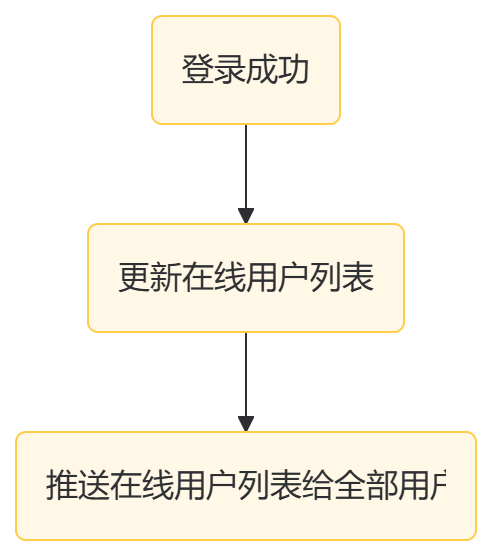
4.1.3消息发送流程图

### 4.1.4 消息接收模块

客户端通过多线程机制，监听服务器发送的消息。接收到消息后，系统根据消息类型显示在相应的聊天界面中。

### 4.1.5 在线用户管理

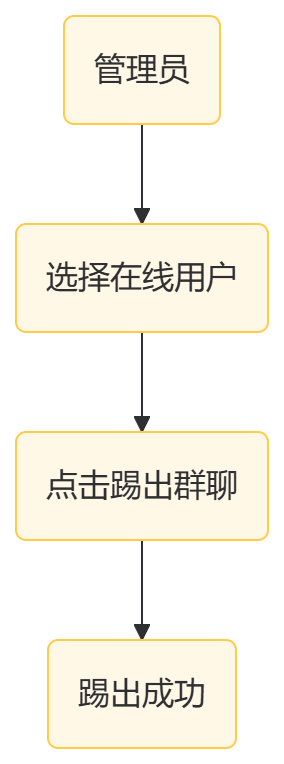
每当有用户登录或退出时，系统会更新在线用户列表，并通过广播的方式将在线用户信息传递给所有客户端。



4.1.5更新在线用户列表流程图

### 4.1.6 管理员管理用户

管理员管理用户模块的设计目标是构建一个高效、直观的管理界面，让管理员能够实时监控在线用户列表，快速执行踢出操作，并维护聊天室秩序，同时确保整个系统的安全性和用户的良好体验。



4.1.6管理员踢人流程图

## 4.2 类设计

## 

4.2类设计图

### 4.2.1 服务器类设计

### 服务器类作为系统架构的中心，承担着核心的网络管理职责。它不仅负责监听和接纳来自客户端的连接请求，还涉及对消息的中转处理以及维护一个实时更新的在线用户列表。该类设计注重于提供一个稳定且高效的通信枢纽，确保数据在服务器与客户端之间的流畅传递，并支持高并发的网络交互。

### 4.2.2 客户端类设计

### 客户端类专注于提供用户与系统之间的交互界面，它集成了图形用户界面（GUI），使用户能够方便地发送和接收消息。该类设计的核心在于确保用户界面的友好性与易用性，同时通过Socket与服务器建立稳定连接，实现消息的及时收发，为用户提供流畅的聊天体验。

### 4.2.3 数据传输设计

数据传输设计采用了DataInputStream和DataOutputStream作为主要的数据传输工具，以二进制形式确保数据在客户端与服务器间的高效、稳定传输。这一设计考虑了数据的完整性和传输的可靠性，通过这些流类提供的一系列方法，实现了数据的精确读写，为系统的通信提供了坚实的基础。

## 4.3 数据库设计

### 4.3.2 数据表结构设计

**用户表**：存储用户名、密码。

**违禁词：**存储违禁词汇，用文档存储。

## 4.4 接口设计

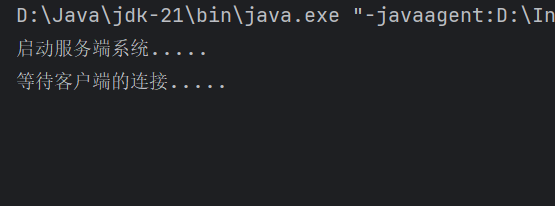
接口主要定义了客户端和服务器之间的通信协议，确保双方能够按照约定的格式进行数据交换。

# 5. 系统实现

## 5.1 服务端

### 5.1.1 服务器启动与监听

服务器端通过ServerScket类启动，并监听指定的端口，等待客户端的连接请求。每当接收到客户端的连接请求，服务器会创建一个新的线程（ServerReaderThread），用来处理该客户端的消息交换。



5.1.1服务端启动图

实现代码如下：

启动服务器并监听端口

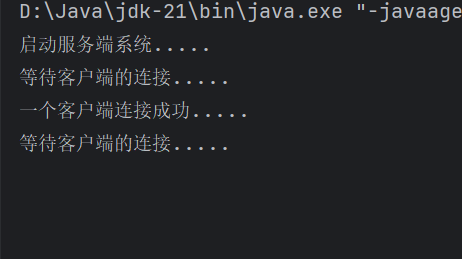
ServerSocket serverSocket = new ServerSocket(Constant.*PORT*);

循环等待客户端的连接请求

while (true) {  
 // 3、调用accept方法，获取到客户端的Socket对象  
 System.*out*.println("等待客户端的连接.....");  
  
 Socket socket = serverSocket.accept();  
 // 把这个管道交给一个独立的线程来处理：以便支持很多客户端可以同时进来通信。  
 new ServerReaderThread(socket).start();  
  
 System.*out*.println("一个客户端连接成功.....");  
}

### 5.1.2 多线程处理客户端请求

为了支持多个客户端并发连接，服务器采用多线程技术，每个客户端连接都由一个独立的线程来处理。ServerReaderThread负责接收和转发消息，处理登录、群聊、私聊等功能。



5.1.2服务端处理客户端请求图

实现代码如下：

@Override  
public void run() {  
 try {  
 // 接收的消息可能有很多种类型：1、登录消息（包含昵称） 2、群聊消息 3、私聊消息 4、管理员踢出命令  
 DataInputStream dis = new DataInputStream(socket.getInputStream());  
 while (true) {  
 int type = dis.readInt(); // 1、2、3、4  
 switch (type) {  
 case 1:  
 // 处理登录请求  
 handleLogin(dis);  
 // 更新管理员界面上的在线用户列表  
 ServerAdminFrame.*getInstance*().updateUserList();  
 break;  
 case 2:  
 // 处理群聊消息  
 handleGroupMessage(dis);  
 break;  
 case 3:  
 // 处理私聊消息  
 handlePrivateMessage(dis);  
 break;  
 case 4:  
 // 处理管理员踢出命令  
 handleKickUserCommand(dis);  
 break;  
 default:  
 break;  
 }  
 }  
 } catch (Exception e) {  
 System.*out*.println("客户端下线了：" + socket.getInetAddress().getHostAddress());  
 Server.*onLineSockets*.remove(socket); // 把下线的客户端socket从在线集合中移除  
 ServerAdminFrame.*getInstance*().updateUserList();  
 updateClientOnLineUserList(); // 下线了用户也需要更新全部客户端的在线人数列表。  
 }  
}

5.1.3 在线用户管理

服务器端维护一个Map<Scket, String>，该数据结构用于存储在线用户的连接Scket及对应的用户名。在用户登录、退出时，服务器会更新在线用户列表，并将其推送到所有客户端。

实现代码如下：

Server.onLineSckets.put(scket, nickname); // 添加新用户到在线列表

updateClientnLineUserList(); // 更新所有客户端的在线用户列表

### 5.1.3 管理员对用户进行操作

##### 管理员对用户进行操作，允许管理员执行包括查看在线用户列表、踢出不当行为用户等管理任务。

### 

5.1.3管理员界面图

### 

5.1.3用户被踢出服务器图

实现代码如下：

private void kickUser(String username) {  
 for (Map.Entry<Socket, String> entry : Server.*onLineSockets*.entrySet()) {  
 if (entry.getValue().equals(username)) {  
 Socket targetSocket = entry.getKey();  
 try {  
 DataOutputStream dos = new DataOutputStream(targetSocket.getOutputStream());  
 dos.writeInt(4); // 踢出消息类型  
 dos.writeUTF("你已被管理员踢出服务器！");  
 dos.flush();  
  
 targetSocket.close(); // 断开连接  
 Server.*onLineSockets*.remove(targetSocket); // 从在线用户列表中移除  
 updateUserList(); // 更新在线用户列表  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 break;  
 }  
 }  
}

## 5.2 客户端实现

### 5.2.1 用户登录

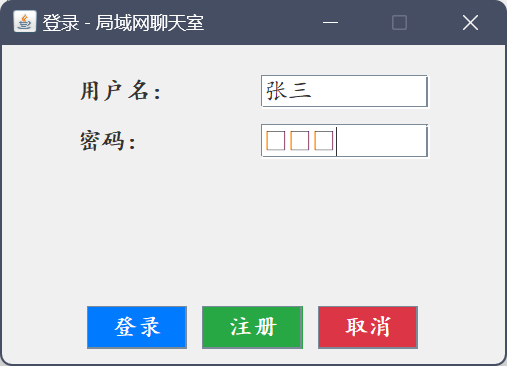
用户登录过程开始于客户端的EntryIpandPORT界面，用户首先需要输入服务器的IP地址和端口号以建立网络连接。随后，用户将被引导至ChatEntryFrame登录界面，在此他们需填写用户名和密码。一旦输入完毕，客户端通过Socket连接向服务器发送这些凭据以请求登录。服务器端随后对提交的登录信息进行验证：若验证成功，用户将被授权进入聊天室的主界面，开始他们的聊天会话。



5.2.1用户输入IP和端口图

实现代码如下：

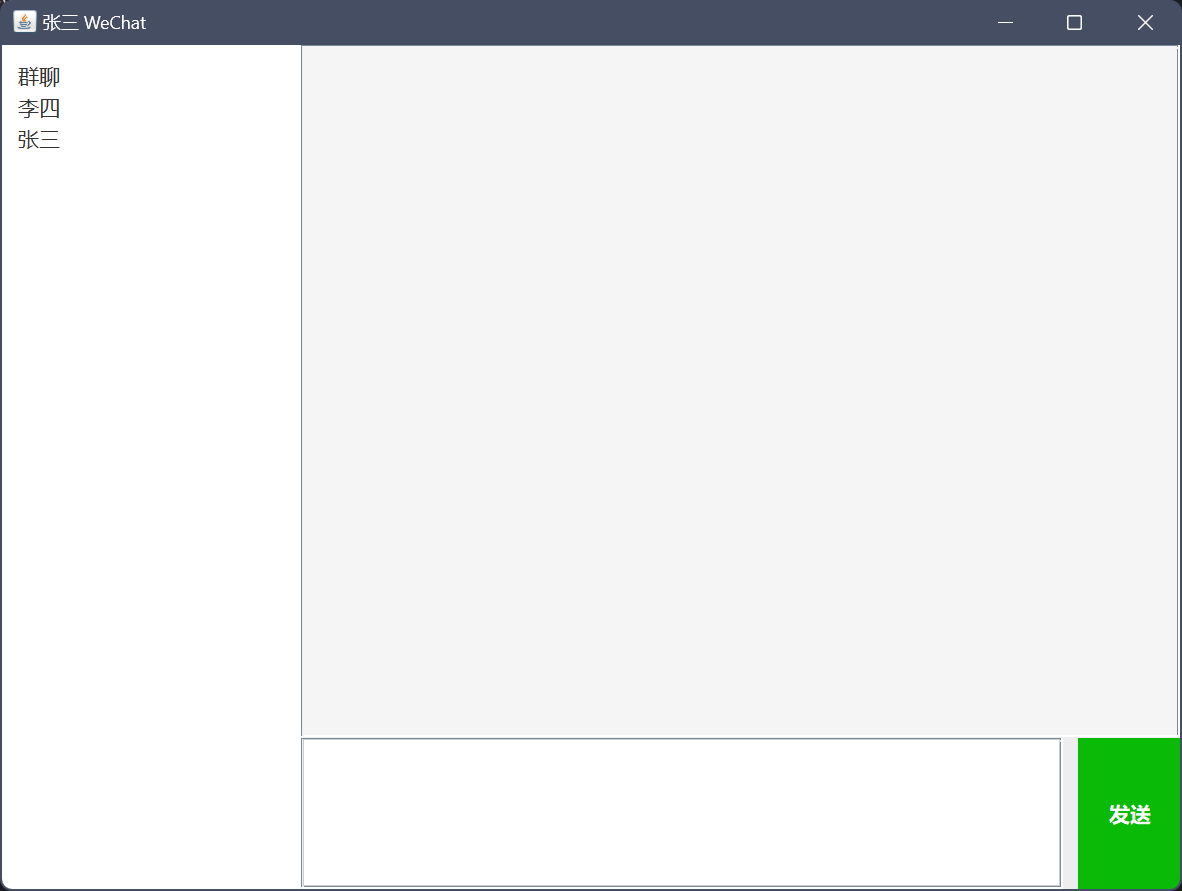
confirmButton.addActionListener(e -> {  
 String serverIP = ipField.getText().trim();  
 String serverPort = portField.getText().trim();  
 try {  
 // 创建 Constant 对象来保存 IP 和端口  
 new Constant(serverIP, serverPort);  
 // 启动 ChatEntryFrame  
 new ChatEntryFrame();  
 dispose(); // 关闭当前窗口  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(this, "端口号格式错误！请重新输入。", "错误", JOptionPane.*ERROR\_MESSAGE*);  
 }  
});



5.2.1用户输入用户名和密码图

实现代码如下：

loginButton.addActionListener(e -> {  
 String username = nicknameField.getText();  
 String password = new String(passwordField.getPassword());  
  
 if (username.isEmpty() || password.isEmpty()) {  
 JOptionPane.*showMessageDialog*(this, "请输入用户名和密码!");  
 } else {  
 if (AccountManager.*login*(username, password)) {  
 try {  
 socket = new Socket(Constant.*getServerIP*(), Integer.*parseInt*(Constant.*getServerPort*()));  
 DataOutputStream dos = new DataOutputStream(socket.getOutputStream());  
 dos.writeInt(1); // 登录消息类型  
 dos.writeUTF(username);  
 dos.flush();  
 new ClientChatFrame(username, socket);  
 this.dispose(); // 关闭登录窗口  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 } else {  
 JOptionPane.*showMessageDialog*(this, "登录失败: 用户名或密码错误");  
 }  
 }  
});



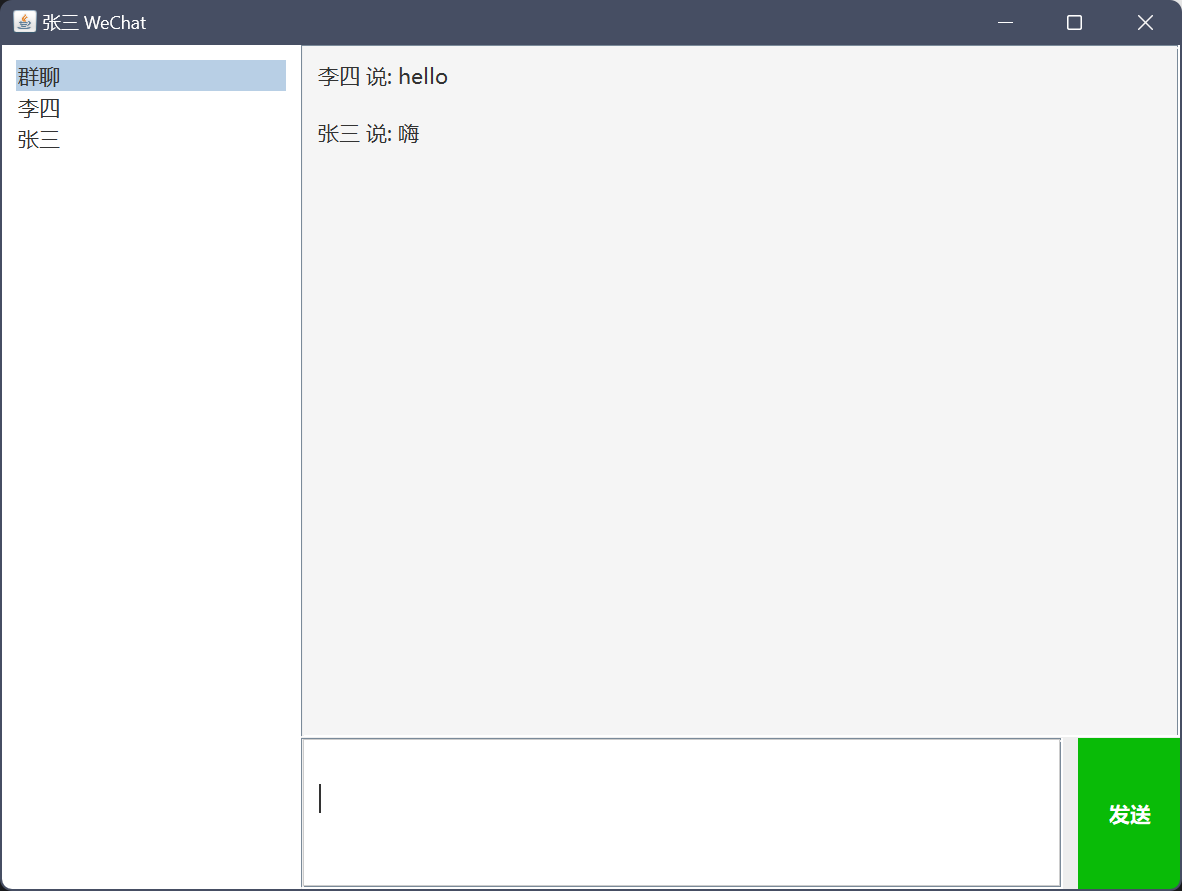
5.2.1用户进入聊天框图

实现代码如下：

JTextArea messageDisplayArea = new JTextArea();  
messageDisplayArea.setEditable(false);  
messageDisplayArea.setLineWrap(true);  
messageDisplayArea.setWrapStyleWord(true);  
messageDisplayArea.setFont(new Font("微软雅黑", Font.*PLAIN*, 14));  
messageDisplayArea.setBorder(BorderFactory.*createEmptyBorder*(10, 10, 10, 10));  
messageDisplayArea.setBackground(Color.*decode*("#F5F5F5"));  
JScrollPane messageScrollPane = new JScrollPane(messageDisplayArea);

### 5.2.2 消息发送与接收

客户端实现了群聊和私聊的功能。消息发送通过DatautputStream向服务器传输，消息接收通过DataInputStream从服务器读取。系统通过多线程处理接收的消息，并将消息显示在对应的聊天窗口。



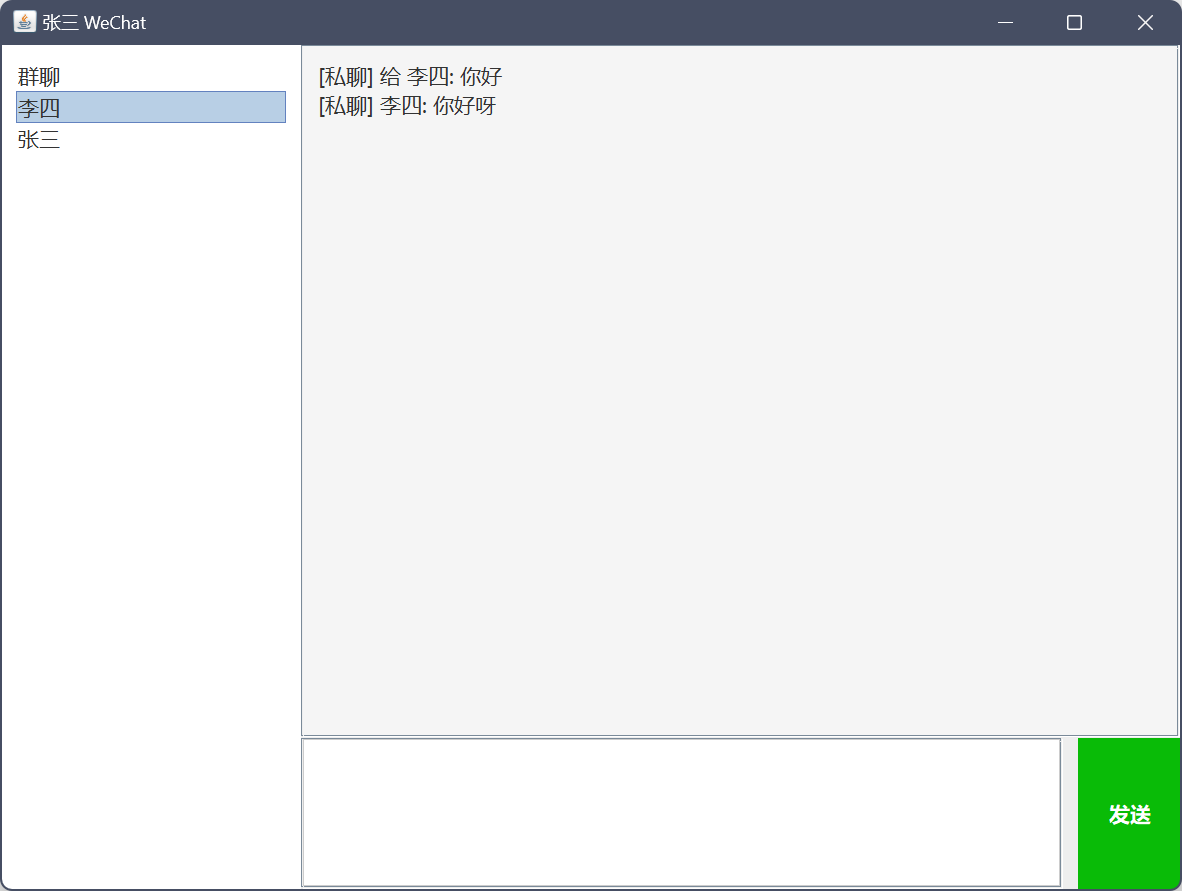
5.2.2消息发送和接收图

实现代码如下：

@Override  
public void run() {  
 try {  
 dis = new DataInputStream(socket.getInputStream());  
 while (!socket.isClosed()) { // 检查连接是否关闭  
 try {  
 int type = dis.readInt(); // 读取消息类型  
  
 switch (type) {  
 case 1:  
 // 更新在线用户列表  
 updateClientOnLineUserList();  
 break;  
 case 2:  
 // 群聊  
 getGroupMessage();  
 break;  
 case 3:  
 // 私聊  
 getPrivateMessage();  
 break;  
 case 4:  
 // 处理踢出命令  
 handleKickOut();  
 break;  
 default:  
 break;  
 }  
 } catch (SocketException e) {  
 // 连接已关闭，退出线程  
 System.*out*.println("连接已关闭，停止接收数据");  
 break; // 退出循环  
 } catch (IOException e) {  
 e.printStackTrace();  
 break; // 发生 I/O 异常时退出循环  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 if (dis != null) dis.close(); // 关闭 DataInputStream  
 if (socket != null && !socket.isClosed()) socket.close(); // 关闭 Socket  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

### 5.2.3 在线用户列表

客户端维护一个在线用户列表，显示所有当前在线的用户。用户可以选择某个用户进行私聊，点击用户后，系统会切换到该用户的聊天窗口。



5.2.3在线用户列表

实现代码如下：

private void updateClientOnLineUserList() {  
 for (Socket socket : Server.*onLineSockets*.keySet()) {  
 try {  
 DataOutputStream dos = new DataOutputStream(socket.getOutputStream());  
 dos.writeInt(1); // 消息类型：在线用户列表更新  
 dos.writeInt(Server.*onLineSockets*.size()); // 在线用户数量  
 for (String nickname : Server.*onLineSockets*.values()) {  
 dos.writeUTF(nickname); // 发送每个在线用户的昵称  
 }  
 dos.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

## 5.3 图形用户界面（GUI）设计与实现

客户端的图形用户界面使用Java Swing库进行开发，界面包含以下主要部分：

**登录界面**：用户输入用户名和密码，点击登录按钮进行身份验证。

**聊天界面**：显示群聊和私聊窗口，提供发送消息、显示消息历史等功能。

**在线用户列表**：左侧显示所有在线用户，用户可以选择与某个用户进行私聊。

**（注；图 5.2.3）**

**实现代码如下：**

public class ClientChatFrame extends JFrame {  
 private String nickname; // 当前用户名  
 private Socket socket;  
 private JList<String> onlineUsersList; // 在线用户列表  
 private JPanel chatPanelsContainer; // 用于存放不同聊天界面的容器  
 private CardLayout cardLayout; // 用于切换不同聊天界面

## 5.4 数据传输与通信协议

客户端与服务器端通过自定义的消息协议进行数据传输。每条消息包含消息类型、发送者、目标用户（如果是私聊）、消息内容等信息。通过DataInputStream和DatautputStream进行高效的二进制数据传输。

例：

@Override  
public void run() {  
 try {  
 // 接收的消息可能有很多种类型：1、登录消息（包含昵称） 2、群聊消息 3、私聊消息 4、管理员踢出命令  
 DataInputStream dis = new DataInputStream(socket.getInputStream());  
 while (true) {  
 int type = dis.readInt();  
 switch (type) {  
 case 1:  
 // 处理登录请求  
 handleLogin(dis);  
 // 更新管理员界面上的在线用户列表  
 ServerAdminFrame.*getInstance*().updateUserList();  
 break;  
 case 2:  
 // 处理群聊消息  
 handleGroupMessage(dis);  
 break;  
 case 3:  
 // 处理私聊消息  
 handlePrivateMessage(dis);  
 break;  
 case 4:  
 // 处理管理员踢出命令  
 handleKickUserCommand(dis);  
 break;  
 default:  
 break;  
 }  
 }  
 } catch (Exception e) {  
 System.*out*.println("客户端下线了：" + socket.getInetAddress().getHostAddress());  
 Server.*onLineSockets*.remove(socket); // 把下线的客户端socket从在线集合中移除  
 ServerAdminFrame.*getInstance*().updateUserList();  
 updateClientOnLineUserList(); // 下线了用户也需要更新全部客户端的在线人数列表。  
 }  
}

## 5.5 多线程机制实现

为了保证服务器在多客户端同时连接时仍能保持高效运行，服务器端使用了多线程处理每个客户端的请求。每个客户端的连接都会启动一个新的ServerReaderThread，独立处理该客户端的消息。

实现代码如下：

try {  
 // 注册端口。  
 ServerSocket serverSocket = new ServerSocket(Constant.*PORT*);  
 // 主线程负责接受客户端的连接请求  
 while (true) {  
 // 调用accept方法，获取到客户端的Socket对象  
 System.*out*.println("等待客户端的连接.....");  
  
 // 阻塞监听，等待客户端的连接，然后返回一个Socket对象  
 Socket socket = serverSocket.accept();  
  
 // 把这个管道交给一个独立的线程来处理：以便支持很多客户端可以同时进来通信。  
 new ServerReaderThread(socket).start();  
  
 System.*out*.println("一个客户端连接成功.....");  
 }  
} catch (Exception e) {  
 e.printStackTrace();  
}

# 6. 系统测试

## 6.1 测试策略与方法

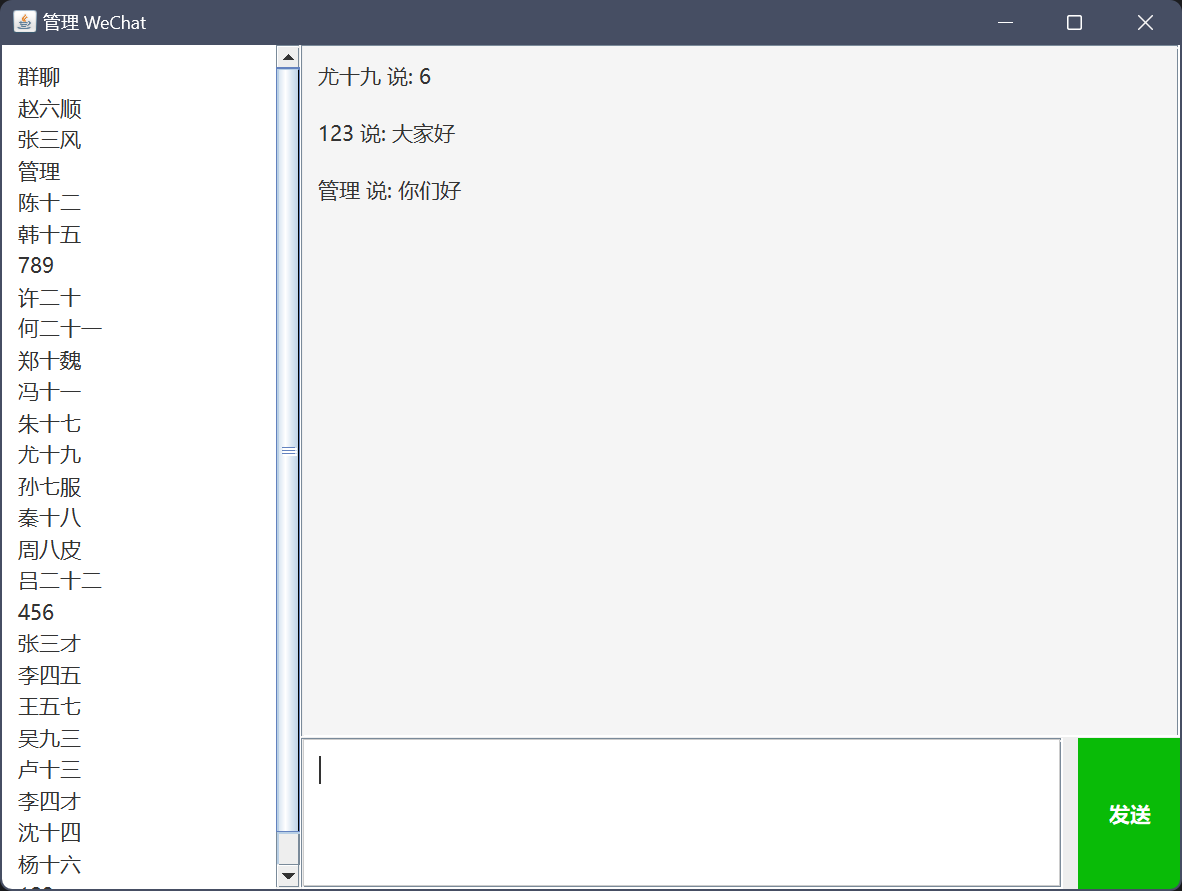
本系统的测试分为功能测试、性能测试、安全性测试和异常处理测试等多个方面。通过单元测试、集成测试和系统测试等手段，确保系统的稳定性、可用性和安全性。

## 6.2 功能测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试编号** | **测试功能** | **测试步骤** | **预期结果** | **实际结果** | **备注** |
| FT-01 | 用户登录 | 1. 输入正确的用户名和密码 2. 点击登录按钮 | 用户成功登录，进入聊天室界面 | 用户成功登录，进入聊天室界面 |  |
| FT-02 | 用户注册 | 1. 输入用户名和密码 2. 点击注册按钮 | 用户注册成功，能够使用新用户名登录 | 用户注册成功，能够使用新用户名登录 | 用户名应为 3 到 15 个字符，密码长度应为 6 到 20 字符 |
| FT-03 | 重复用户名处理 | 1. 注册已存在的用户名2. 尝试登录 | 系统提示用户名已存在，注册失败 | 用户名已存在，注册失败 |  |
| FT-04 | 群聊消息发送与接收 | 1. 用户A发送消息2. 用户B和C检查接收到的消息 | 所有在线用户及时且准确地接收到消息 | 所有在线用户及时准确接收到消息 |  |
| FT-05 | 私聊消息发送与接收 | 1. 用户A发送私聊消息给用户B2. 用户B检查接收到的消息 | 只有发送者和指定接收者能够看到消息 | 只有发送和指定接收者能看到消息 |  |
| FT-06 | 消息完整性 | 1. 用户发送包含特殊字符消息2. 其他用户接收消息 | 消息格式和内容保持完整，无丢失或损坏 | 内容保持完整，无丢失或损坏 | 若存在违规词被替换为“\*”系正常 |
| FT-07 | 群聊功能 | 1. 多个用户发送群聊消息2. 检查消息显示 | 所有参与者的消息被正确显示 | 所有参与者的消息被正确显示 |  |
| FT-08 | 私聊功能 | 1. 用户之间发送私聊消息2. 检查消息私密性 | 消息私密性得到保证，不被其他用户查看 | 消息私密性得到保证，不被其他用户查看 |  |
| FT-09 | 聊天模式切换 | 1. 用户在群聊和私聊之间切换2. 检查聊天记录 | 用户界面平滑过渡，聊天记录完整 | 用户界面平滑过渡，聊天记录完整 |  |

## 6.3 性能测试

测试系统的响应时间和并发处理能力。通过模拟多个客户端同时连接，测试系统在高并发下的性能表现，确保系统能够稳定运行。



6.3性能测试

## 6.4 异常处理测试

模拟不同的异常场景（如网络断开、客户端异常退出等），测试系统是否能够正确处理并恢复正常工作。

测试均通过



6.4异常测试

## 6.5 测试结果与分析

所有功能模块通过了单元测试和集成测试，系统在高并发环境下能够稳定运行。经过安全性测试后，发现系统能够有效防止数据泄露。

# 7. 总结与展望

## 7.1 系统总结

本系统实现了一个基于局域网的即时通讯工具，具备群聊、私聊和在线用户管理等功能。通过多线程机制，系统能够高效处理多个客户端的请求，保证了系统的响应速度。客户端和服务器之间的通信采用了简单的自定义协议，确保了数据传输的高效性与稳定性。

## 7.2 存在的不足与挑战

尽管系统在基本功能上已能稳定运行，但仍存在以下不足：

**扩展性不足**：目前仅支持文字消息，未来可以考虑加入表情、图片等多媒体消息的支持。

## 7.3 未来扩展与改进方向

未来可以考虑在以下几个方面进行改进：

**增加更多消息类型**：如表情、文件传输、视频通话等。

**引入分布式架构**：当前系统只支持单机部署，未来可以支持分布式部署，提高系统的可扩展性。

**增强安全性**：增加对消息的加密传输和用户身份验证，提升系统的安全性。

**增加消息存储**：实现聊天记录的持久化存储，支持用户查看历史消息。

# 8. 参考文献

Java SE 8 官方文档

《Java网络编程（第四版）》 - Elliotte Rusty Harold

Swing 开发指南

实验室项目参考资料