

浙江大学计算机科学与技术学院

Java 程序设计课程报告

2017—2018 学年秋冬学期

题目	多人聊天室开发
学号	3160104875
学生姓名	杨樾人
所在专业	软件工程
所在班级	软工 1602

目录

一、引言

本次作业是使用 Java 的 swing 框架开发一个多人聊天的网络应用，支持多人使用，同时使用数据库，记录用户的消息。通过此次编程，加深自己对 Java 的理解，提高自己的编程水平。

1.1 设计目的

- 1. 使用 GUI 编程。用户界面用 GUI 实现
- 2. 使用网络编程。支持多用户同时使用
- 3. 使用数据库(例如记录用户行为)、并发编程

此次的多人聊天应用，我使用到了 socket 编程，多线程，数据库，以及用户的注册登陆，查看历史消息等功能，在一定程度上满足老师布置的实验要求，并且编程过程中，也对照老师的代码要求，对代码进行了规整。

1.2 设计说明

本程序采用 Java 程序设计语言，在 IntelliJ Idea 平台下编辑、编译与调试。具体程序由我个人开发而成。使用了数据库，socket，多线程，swing 框架设计 gui，工作时间轴如表 1 所示:

表 1 工作时间轴

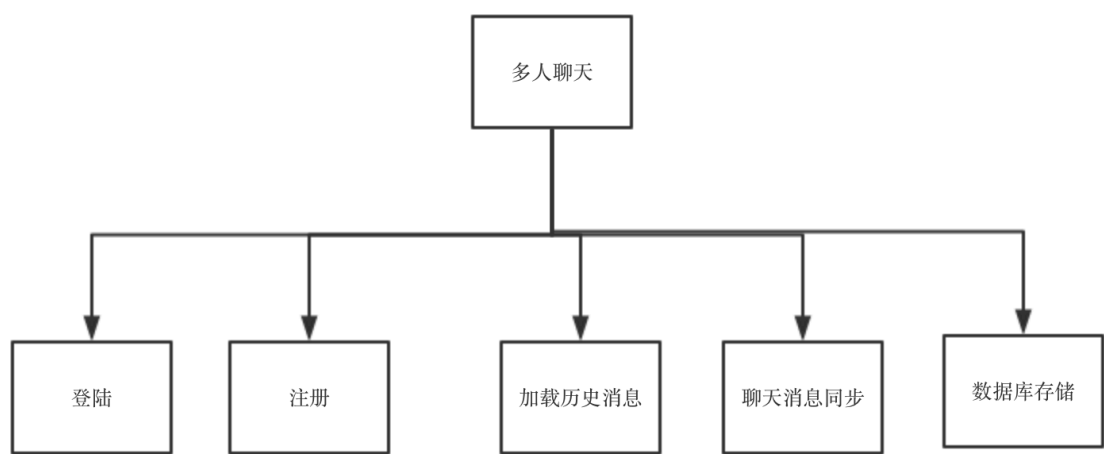
时间	完成的工作
1 月 4 日	学习 swing 框架
1 月 10 日	数据库
1 月 15 日	客户端编程
1 月 18 日	服务器编程
1 月 20 日	文档撰写

二、总体设计

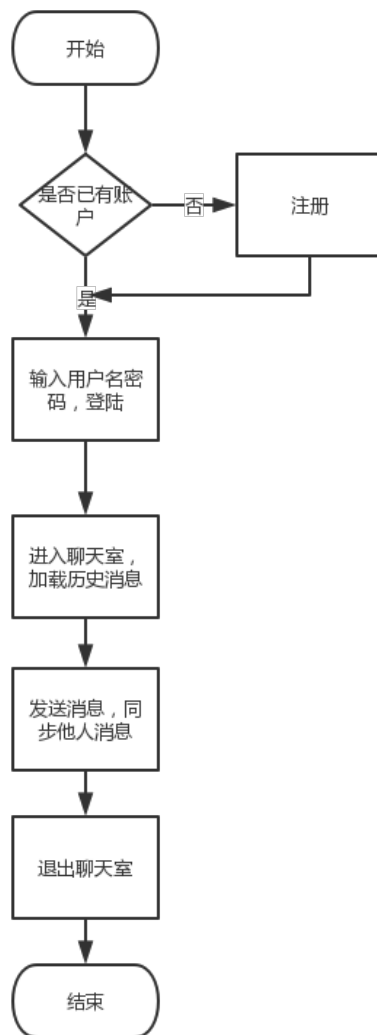
2.1 功能模块设计

本程序主要完成了以下功能：

- 1. 用户登陆
- 2. 用户注册
- 3. 用户退出
- 4. 多人聊天
- 5. 数据库存储历史消息
- 6. 加载历史消息

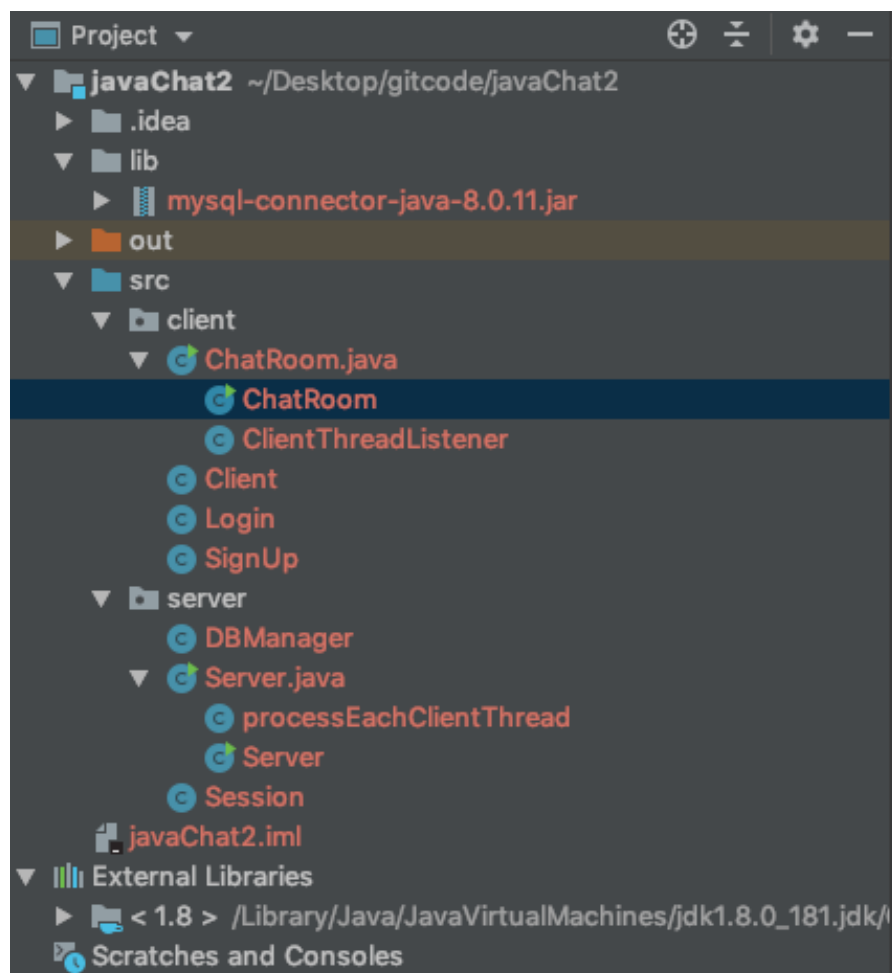


2.2 流程图设计



三、详细设计






















在设计过程中，主要分为三部分，客户端设计，服务器设计，数据库管理。以下将逐一进行说明。



3.1 客户端设计

当开启客户端后，客户端首先会连接 127.0.0.1 端口为 12345 的服务器，连接成功后，跳出登陆界面。

登陆功能中的类如下：

	Login	
	client	Client
	chatRoom	ChatRoom
	loggedIn	boolean
	exitButton	JButton
	jLabel1	JLabel
	jLabel2	JLabel
	jLabel3	JLabel
	loginButton	JButton
	passwordField	JPasswordField
	signupButton	JButton
	usernameField	JTextField
	Login(Frame, boolean, Client)	
	initFrame()	void
	attemptLogin()	void
	loginButtonActionPerformed(ActionEvent)	void
	signupButtonActionPerformed(ActionEvent)	void
	exitButtonActionPerformed(ActionEvent)	void
	passwordFieldKeyPressed(KeyEvent)	void
	formWindowClosed(WindowEvent)	void
	getPassword()	char[]

initFrame 是初始化界面，其余的 label 和 button 是让用户输入账号密码。signupButton 是跳转注册页面。

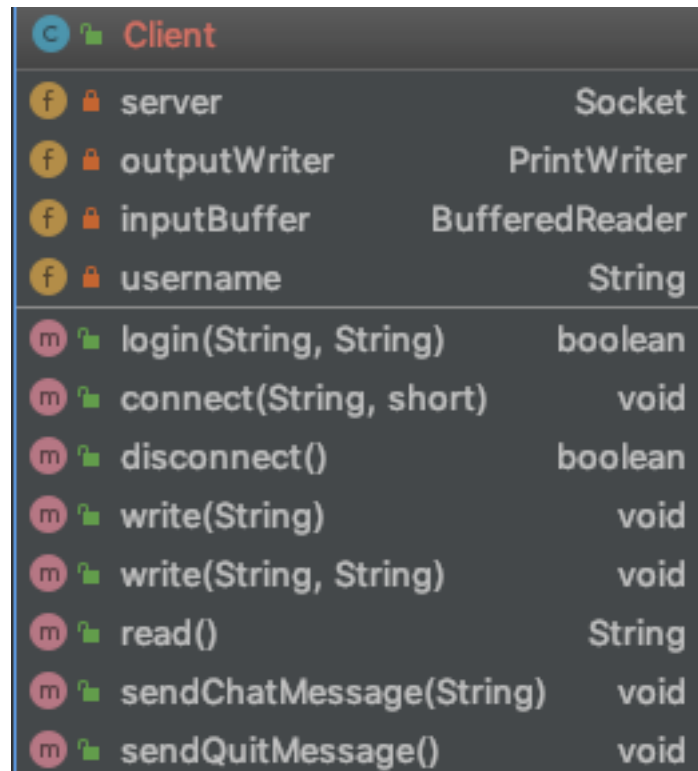
用户输入账户密码，Login 类中的 attemptLogin 方法会查询数据库中的用户名和密码，与用户输入的用户名密码进行匹配，如果成功，则创建一个监听线程，用来与服务器进行通信，如果密码错误，则让用户重新输入。

函数如下：

```
private void attemptLogin() {
    if (usernameField.getText().length() < 4) {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Username must be at least 4 characters long.", title: "Warning", messageType: JOptionPane.WARNING_MESSAGE);
    } else if ((new String(passwordField.getPassword())).length() < 8) {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Password must be at least 8 characters long.", title: "Warning", messageType: JOptionPane.WARNING_MESSAGE);
    } else {
        if (client.login(usernameField.getText(), new String(passwordField.getPassword()))) {
            chatRoom.setVisible(true);
            chatRoom.startChatListener();
            loggedIn = true;
            dispose();
        } else {
            JOptionPane.showMessageDialog( parentComponent: this, message: "Incorrect username or password.", title: "Warning", messageType: JOptionPane.WARNING_MESSAGE);
        }
    }
}
```


当客户端连接服务器成功后，客户端会收到服务器发送过来的该聊天室的历史消息，这些历史消息存储在数据库之中。

client 类中有发送和接收消息的功能：



Client		
f	server	Socket
f	outputWriter	PrintWriter
f	inputBuffer	BufferedReader
f	username	String
<hr/>		
m	login(String, String)	boolean
m	connect(String, short)	void
m	disconnect()	boolean
m	write(String)	void
m	write(String, String)	void
m	read()	String
m	sendChatMessage(String)	void
m	sendQuitMessage()	void

在监听线程中，调用 client 类中的读取消息来对聊天内容和聊天在线用户进行更新。

```

/** 通过socket写消息 */
public void write(String msg) {
    outputWriter.println(msg);
    outputWriter.flush();
}

/**重载，处理含有时间的消息*/
public void write(String time, String msg) {
    outputWriter.println(time);
    outputWriter.println(msg);
    outputWriter.flush();
}

/**通过socket读消息*/
public String read() {
    String line = null;
    try {
        line = inputBuffer.readLine();
    } catch (IOException e) {
        System.err.println(e);
        e.printStackTrace();
    }
    return line;
}

/**发送消息*/
public void sendChatMessage(String msg) {
    write(msg: username + ": " + msg);
}

/**退出登陆 关闭连接*/
public void sendQuitMessage() { write(msg: "QUIT"); }

```

Throwable argument 'e' to 'System.err.println()' call [more...](#) (%F1)

其中，从服务器读取消息并更新客户端的消息列表以及在线用户列表的实现方法如下：

```

public void run() {
    while (true) {
        String line;
        if ((line = client.read()) != null) {
            /*更新在线用户，通过 USERLIST 字符串来说明是在线用户列表*/
            if (line.startsWith("USERLIST: ")) {
                String[] usernames = line.substring(line.indexOf(' ')).split(" ");
                usernameList.setListData(usernames);
            } else {
                chatBox.setText(chatBox.getText() + line + "\n"); /*更新聊天框的内容*/
            }
        }
    }
}

```

用户注册类如下：

SignUp		
f	client	Client
f	cancelButton	JButton
f	confirmField	JPasswordField
f	jLabel1	JLabel
f	jLabel2	JLabel
f	jLabel3	JLabel
f	jLabel4	JLabel
f	jLabel5	JLabel
f	jPanel1	JPanel
f	passwordField	JPasswordField
f	submitButton	JButton
f	usernameField	JTextField
m	SignUp(Frame, boolean, Client)	

其中的 jLabel 和 jPanel1 是界面里的内容，让用户填写注册需要用的用户名和密码。

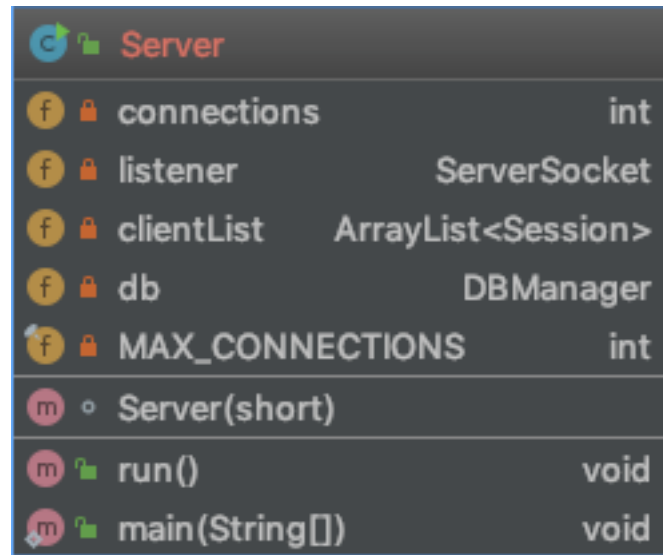
提交用户名和密码之后，在客户端首先会对用户名和密码进行验证，密码长度需要大于 8 位，用户名需要大于 4 位，通过验证后，将用户名和密码的消息通过 socket 发送到服务器。

```
/**提交*/
private void submitButton(java.awt.event.ActionEvent evt) { // GEN-FIRST:event_submitButton
    String pass1 = new String(passwordField.getPassword());
    String pass2 = new String(confirmField.getPassword());
    /**密码是否匹配*/
    if (!pass1.equals(pass2)) {
        JOptionPane.showMessageDialog( parentComponent: this, message: "Password fields do not match.", title: "Warning", messageType: 0);
    } else if (pass1.length() < 8) {
        /**密码长度大于8位*/
        JOptionPane.showMessageDialog( parentComponent: this, message: "Password must be at least 8 characters long.", title: "Warning", message
    } else if (usernameField.getText().length() < 4) {
        /**用户名大于三位*/
        JOptionPane.showMessageDialog( parentComponent: this, message: "Username must be at least 4 characters long.", title: "Warning", messa
    } else {
        client.write( msg: "NEWUSER: " + usernameField.getText() + "," + new String(passwordField.getPassword()));
        String response = client.read();
        /**服务器返回消息。 TAKEN代表已经存在用户 USERCREATED代表成功*/
        if (response.equals("TAKEN")) {
            JOptionPane.showMessageDialog( parentComponent: this, message: "Username is already taken.", title: "Warning", messageType: 0);
        } else if (response.equals("USERCREATED")) {
            JOptionPane.showMessageDialog( parentComponent: this, message: "Your account has been created.", title: "Warning", messageType: 1);
            dispose();
        }
    }
}
```

3.2 服务器设计

服务器设计如下：

server 类：



The image shows a UML class diagram for a class named 'Server'. It lists several attributes and methods. Attributes include 'connections' (int), 'listener' (ServerSocket), 'clientList' (ArrayList<Session>), 'db' (DBManager), and 'MAX_CONNECTIONS' (int). Methods include 'Server(short)', 'run()', and 'main(String[])'. Each attribute and method is preceded by a small icon: a yellow circle with 'f' for fields, a pink circle with 'm' for methods, and a blue circle with 'c' for constructors.

Server		
f	connections	int
f	listener	ServerSocket
f	clientList	ArrayList<Session>
f	db	DBManager
f	MAX_CONNECTIONS	int
m	Server(short)	
m	run()	void
m	main(String[])	void

主要变量如下：

```
/** T 连接的客户端的数量 */
private int connections;
/**数据库设置*/
private String dbUser1 = "root";
private String dbPass1 = "mysqllyr";

/**
 * 服务器的 socket
 */
private ServerSocket listener;

/** 连接的客户端 */
private ArrayList<Session> clientList;

/**数据库
 */
private DBManager db;
```

server 类中最主要的方法是 run，run 中对 12345 的端口进行监听，当有一个客户端连接到服务器时，服务器就新开一个线程进行处理。

```

/** Main method to start up the server on a port */
public static void main(String args[]) {

    final short PORT = 12345;
    Server server = new Server(PORT);

    try {
        server.run();
    } catch (IOException e) {
        System.err.print(e);
        e.printStackTrace();
    }

}

```

```

/**
 * 服务器运行
 */
public void run() throws IOException {

    String dbUser = dbUser1;
    String dbPass = dbPass1;

    String dbAddress = "jdbc:mysql://localhost:3306/chatdb?autoReconnect=true&useSSL=false";
    db = new DBManager(dbAddress, dbUser, dbPass);

    /** 监听连接 */
    while (true) {

        if (++connections < MAX_CONNECTIONS || MAX_CONNECTIONS == 0) {

            /** 阻塞 */
            Socket client = listener.accept();
            /** 使用线程处理客户端消息 */
            new Thread(new processEachClientThread(client, clientList, db)).start();

        }

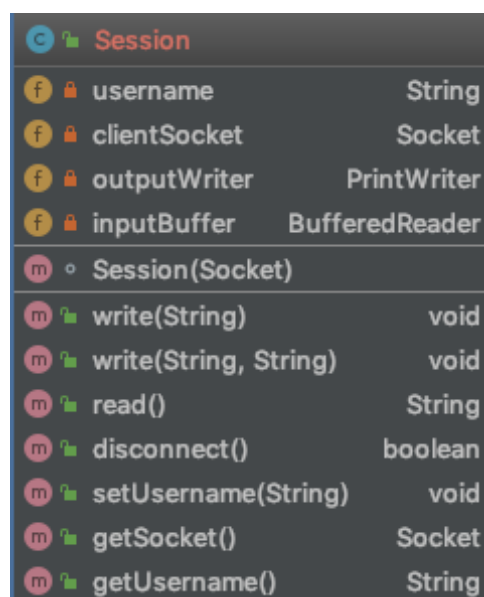
    }

}
}

```

session 类:

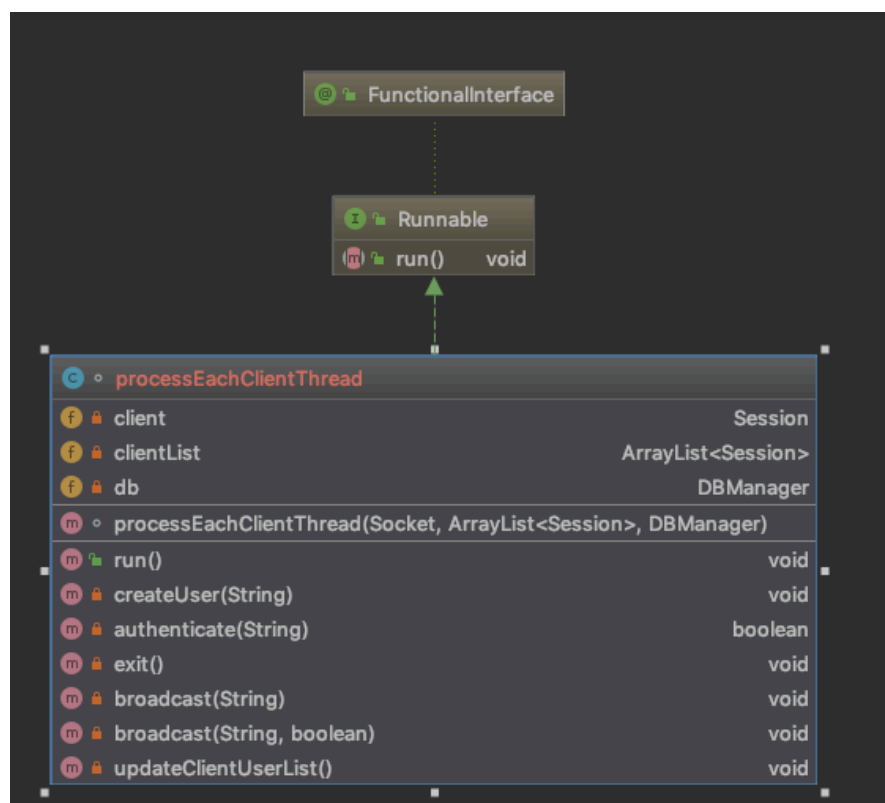
对每一个连接到客户端，都有一个 session 类来存储其信息。



clientSocket 变量存储着该 socket 的信息，方便与该客户端进行数据交换。

wwrite 和 read 方法是与该客户端进行数据的发送和读取。disconnect 方法是断开连接。

processEachClientThread 类是对客户端消息处理。



broadcast 方法是广播消息，updateClientUserList 方法是更新在线用户列表，

clientList 是 server 类中传进来的引用，所有的线程共享。

当一个用户连接到服务器时，会读取历史消息，发送给客户端。

```
/**history*/
ArrayList<String> history = null;
try {
    history = db.queryHistory();
} catch (SQLException e) {
    e.printStackTrace();
}
String mess = null;
for(int i = 0; i < history.size(); i++){
    mess = history.get(i);
    System.out.println(history.get(i));

    String time = mess.split( regex: "::")[0];
    String msg = mess.substring(mess.indexOf("::")+2);
    client.write( time: "\n"+time, msg);
}
```

当该用户发送了一个消息时，服务器也会将该消息广播，并且存储到数据库之中。

```
/**
 *从客户端读取消息，然后广播
 */
while (true) {
    String line = client.read();
    if (line == null)
        break;
    else {
        Date now = new Date();//可以精确到时分秒
        try {
            db.insertMessage(line.split( regex: ":")[0], line.split( regex: ":")[1], now.toString());
            System.out.println("hhhhh" + line);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        broadcast( mess: now.toString() + "::"+line, ismsg: true);
    }
}
```

当一个客户端断开连接时，服务器会更新客户端列表。

```
private synchronized void exit() {
    String exitMsg = "ChatServer: User " + client.getUsername();
    exitMsg += " has left the chat.";

    /** 广播客户端退出通知 */
    broadcast(exitMsg);

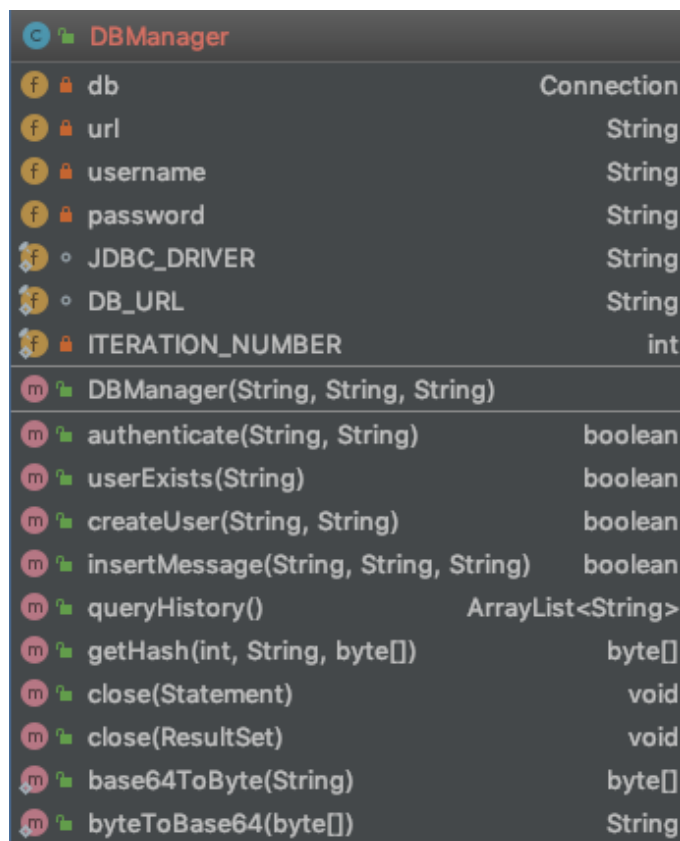
    /** 客户端列表删掉该客户端 */
    client.disconnect();
    clientList.remove(client);

    /** 更新每个客户端的在线用户list */
    updateClientUserList();

    System.out.println("Log: Client socket closed, removed from client list");
}
```

3.3 数据库设计

数据库类如下，主要完成用户名密码的存储查询、消息的存储读取功能。



DBManager		
f	db	Connection
f	url	String
f	username	String
f	password	String
f	JDBC_DRIVER	String
f	DB_URL	String
f	ITERATION_NUMBER	int
DBManager(String, String, String)		
m	authenticate(String, String)	boolean
m	userExists(String)	boolean
m	createUser(String, String)	boolean
m	insertMessage(String, String, String)	boolean
m	queryHistory()	ArrayList<String>
m	getHash(int, String, byte[])	byte[]
m	close(Statement)	void
m	close(ResultSet)	void
m	base64ToByte(String)	byte[]
m	byteToBase64(byte[])	String

数据库使用的是 mysql，有两张表，MESSAGES 和 CREDENTIAL，messages 表中存放发送消息的用户名，消息内容和时间，CREDENTIAL 存放用户名、hash 之后的密码和 salt。

DBManager 中 createUser 方法是将用户名和密码插入到数据库中，userExists 方法是判定当前用户名是否在数据库中已经存在，主要用户注册账户时的判断，authenticate 是验证登陆时用户名和密码是否与数据库中的一致。

insertMessage 方法是插入一条消息到数据库之中，queryHistory 是读取数据库中的历史消息。


```
mysql> show tables;
+-----+
| Tables_in_chatdb |
+-----+
| CREDENTIAL        |
| MESSAGES           |
+-----+
2 rows in set (0.00 sec)

mysql> select * from credential;
+-----+-----+-----+
| LOGIN | PASSWORD | SALT |
+-----+-----+-----+
| aabaab | 7E+8M6qhZY9cA/6jCQfkWz7htVI= | Y/g5e2Z/bZ8= |
| cqycqy | IspXuJrfZm/1fJ75GiuG74FJ8pY= | ubp17MaIT9I= |
| llqllq | Ml+kJ5vU0MPqUk8XawUHJmAJqLk= | QneFVhrdyc8= |
| yyryyr | TiIBATXAz3nua6sF/GscRxRCISM= | ZB4rKvFkF+c= |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select *
-> from messages;
+-----+-----+-----+
| USERNAME | CONTENT | THETIME |
+-----+-----+-----+
| yyryyr   | hello   | Mon Jan 21 21:39:06 CST 2019 |
| cqycqy   | 你好    | Mon Jan 21 21:39:26 CST 2019 |
| cqycqy   | 你喜欢最近上映的哪部电影呀 | Mon Jan 21 21:40:08 CST 2019 |
| yyryyr   | 我对喜剧之王很感兴趣呢 | Mon Jan 21 21:40:19 CST 2019 |
| llqllq   | 我也喜欢 | Mon Jan 21 21:41:24 CST 2019 |
| cqycqy   | 我看过第一部后，一直期待第二部！听说贺岁档要上映了，好开心。 | Mon Jan 21 21:43:11 CST 2019 |
| yyryyr   | 我想去看第二部 | Mon Jan 21 23:04:12 CST 2019 |
| llqllq   | 我们可以一起去看 | Mon Jan 21 23:04:24 CST 2019 |
+-----+-----+-----+
8 rows in set (0.00 sec)
```

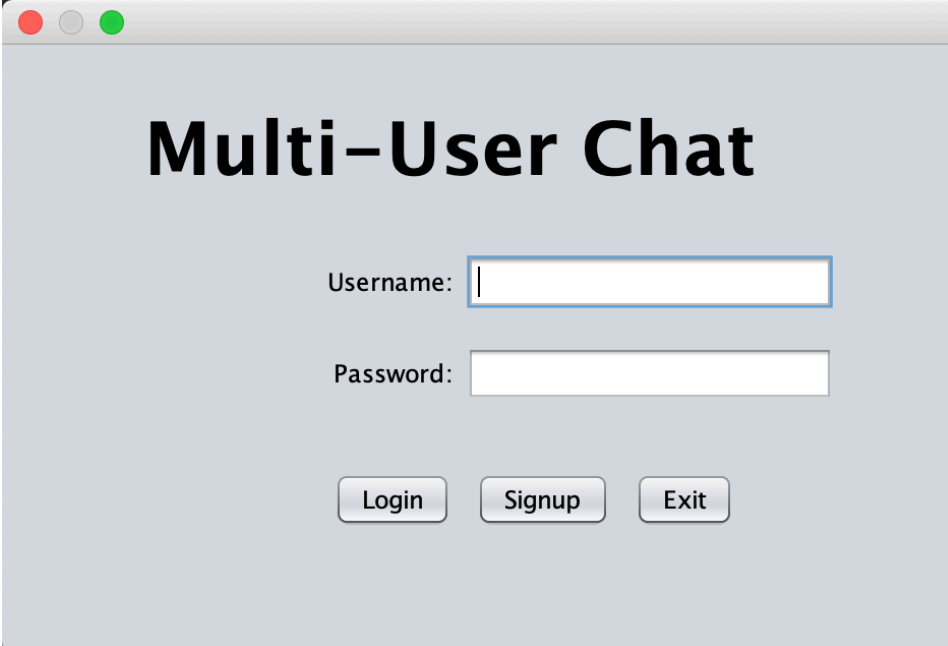
四、测试与运行

4.1 程序测试

在程序代码基本完成后，经过不断的调试与修改，最后测试本次所设计的多人聊天可以正常运行，没有出现明显的错误和漏洞，但是在一些方面仍然需要完善，比如用户交互方面，可以进行优化，可以扩展私聊的功能，读取历史消息时应该加上判断，如果消息规模越来越大，历史消息将过于庞杂。

4.2 程序运行

4.2.1 登陆界面：

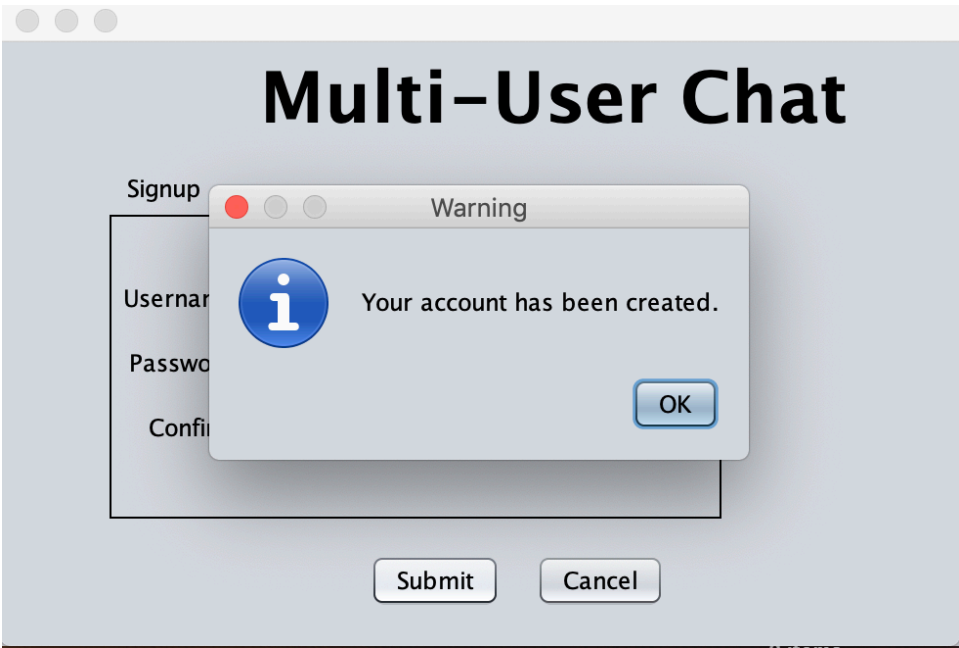
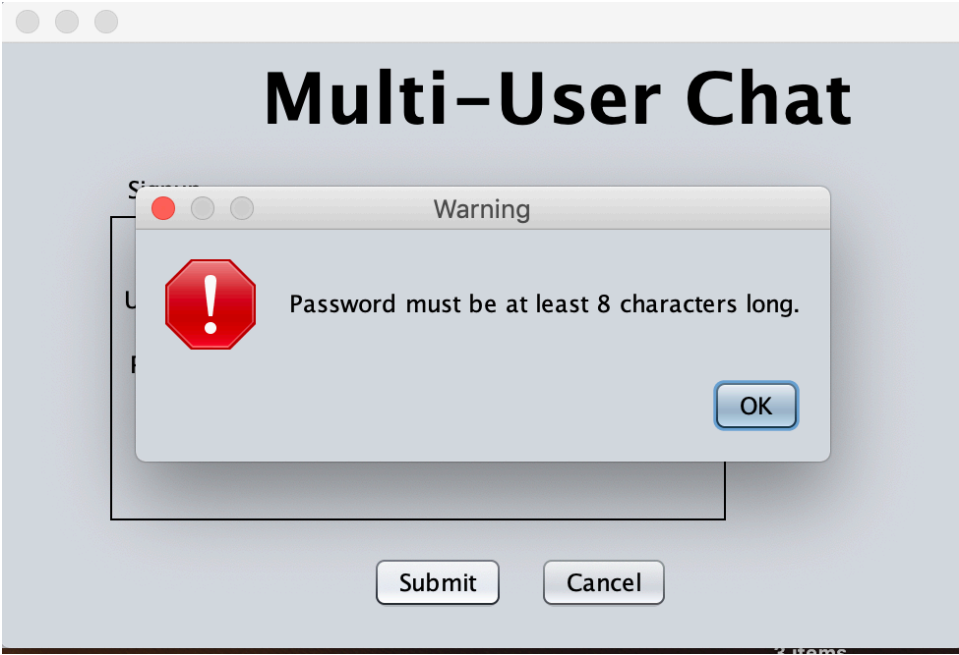
A screenshot of a web application window titled "Multi-User Chat". The window has a light blue background and a standard macOS-style title bar with red, yellow, and green buttons. The title "Multi-User Chat" is centered at the top in a large, bold, black font. Below the title, there are two input fields: "Username:" followed by a text box with a blue border, and "Password:" followed by a text box. At the bottom, there are three buttons: "Login", "Signup", and "Exit", each with a light blue background and a thin border.

Multi-User Chat

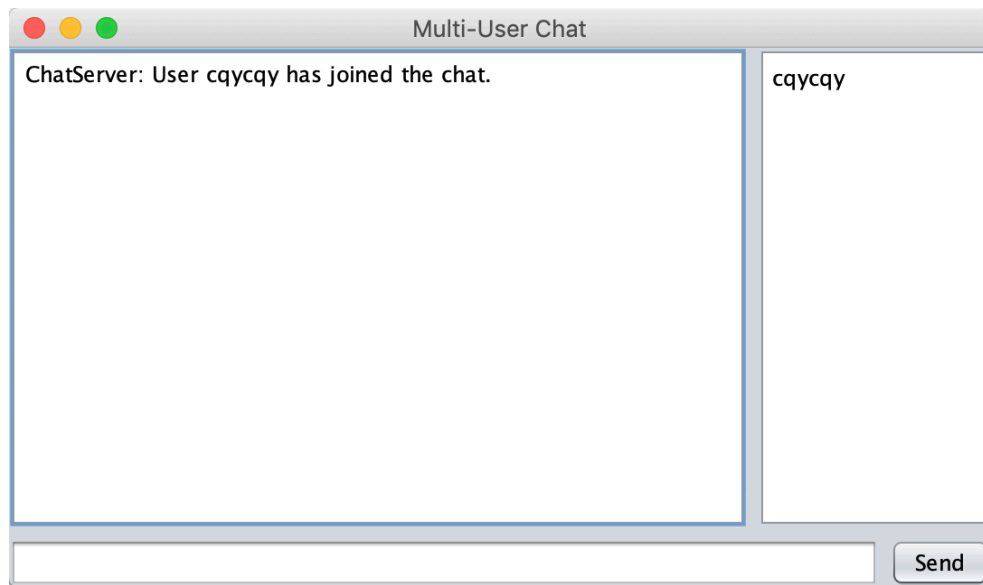
Username:

Password:

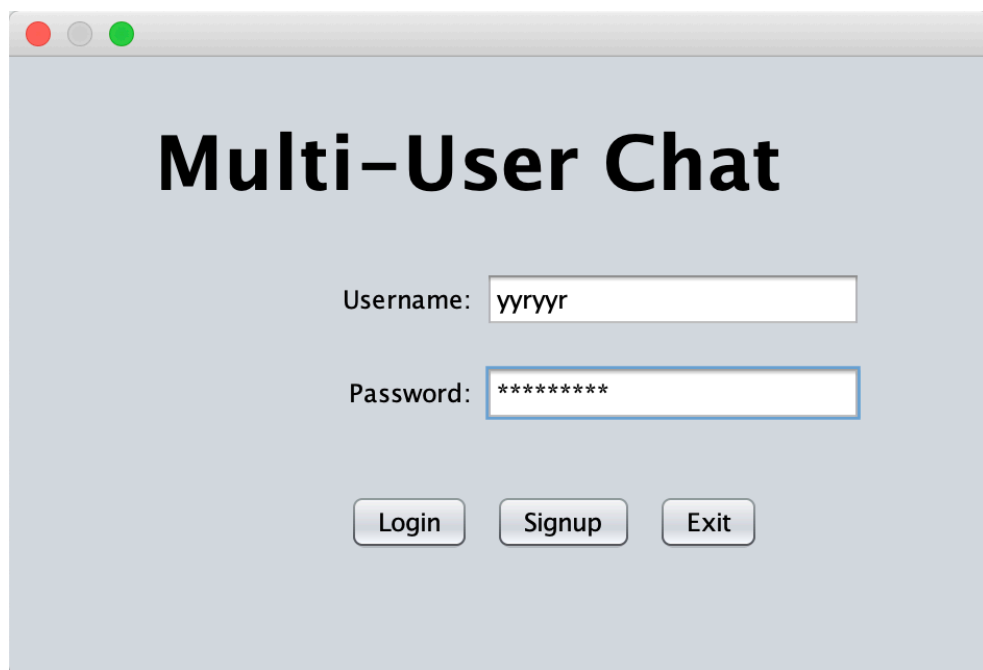
4.2.2 注册界面：

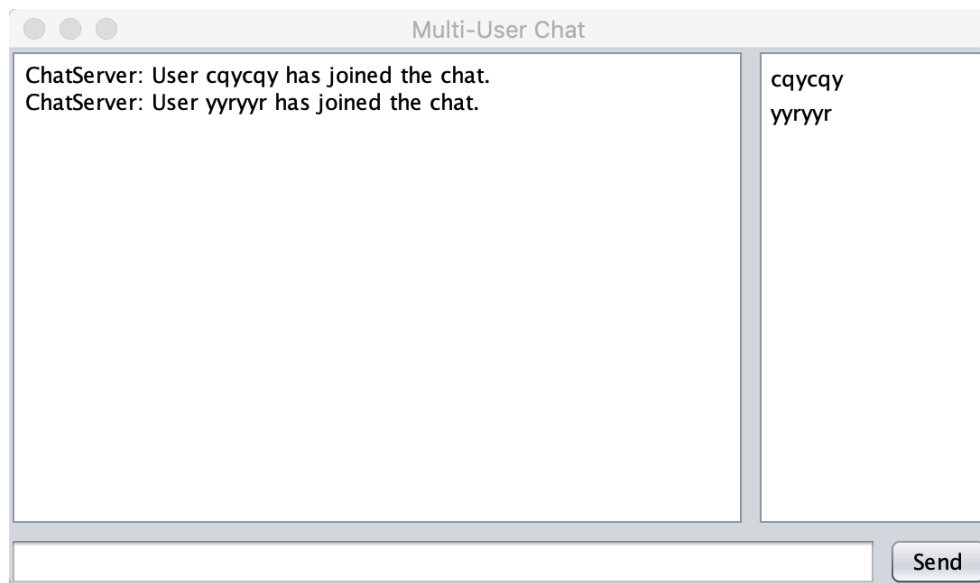


4.2.3 聊天界面:

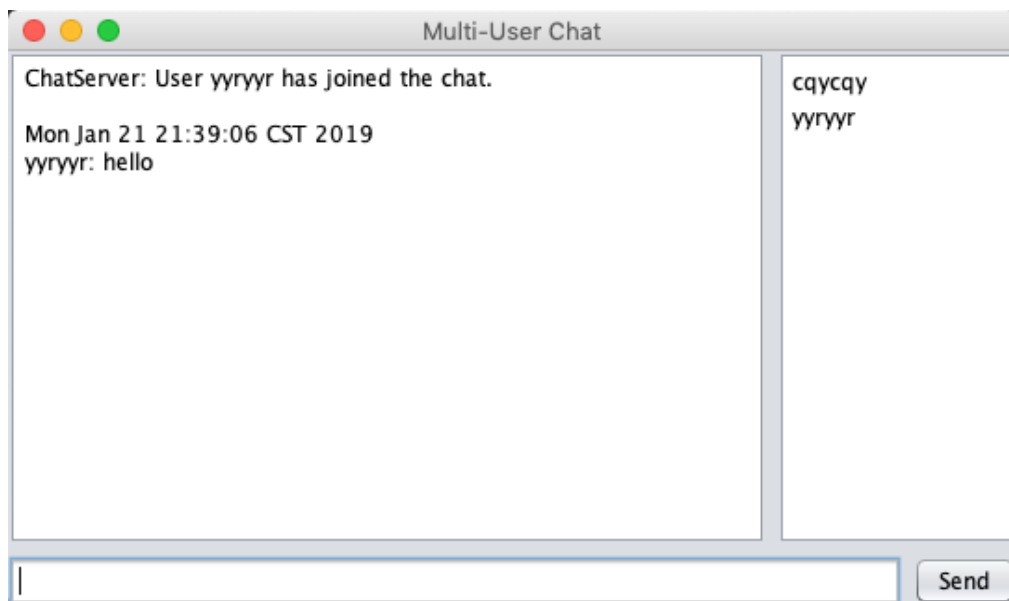


4.2.4 第二个用户登陆

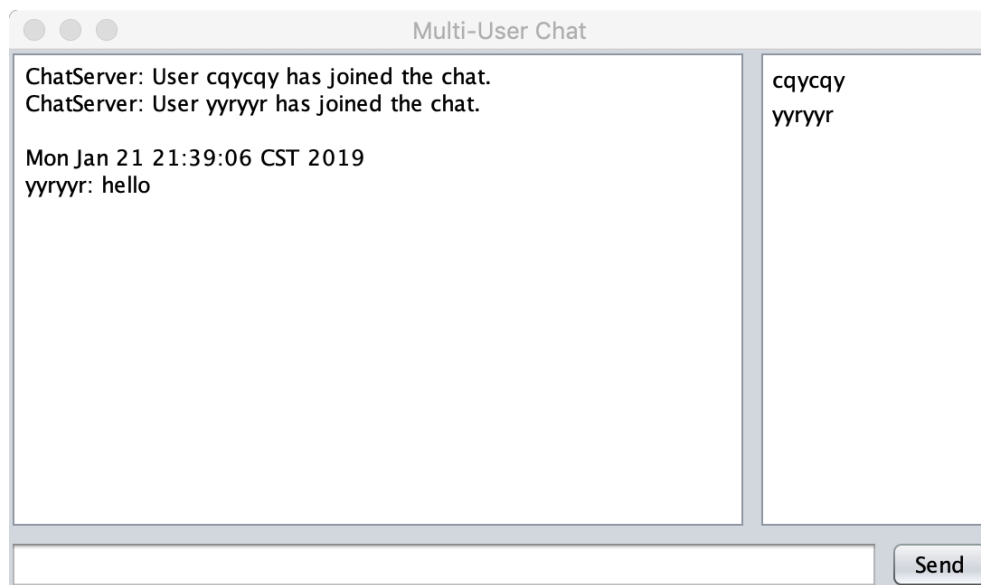




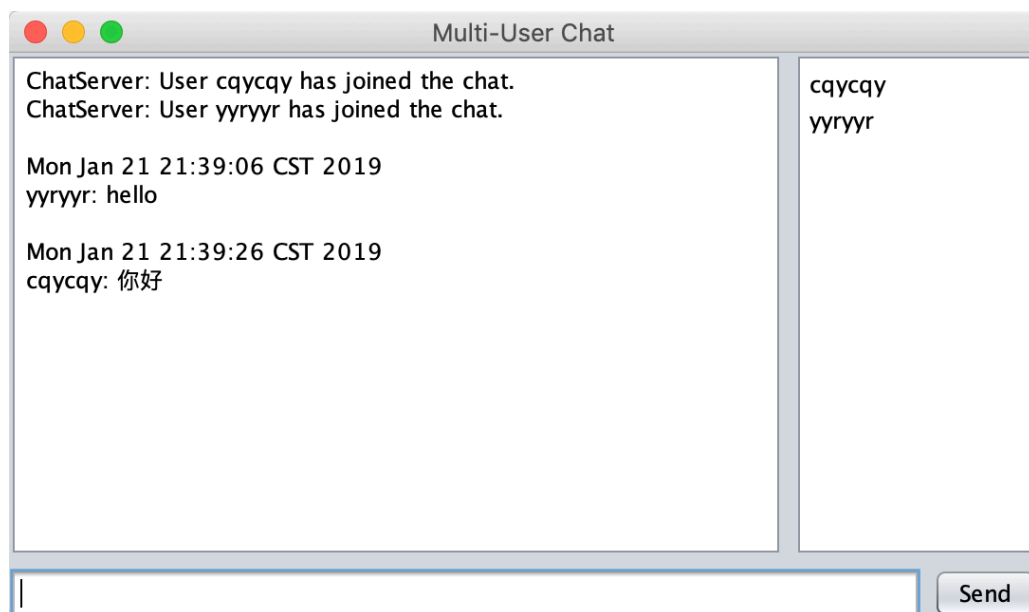
4.2.5 yyryyr 发送消息:



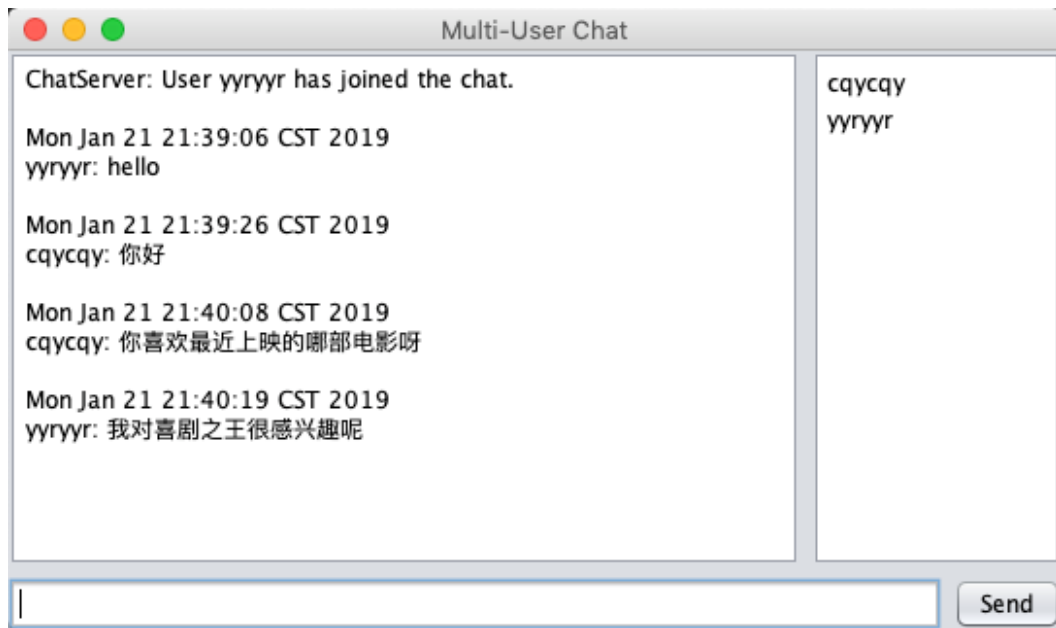
4.2.6 另外的客户端 cqycqy 收到消息:



4.2.7 cqycqy 发送消息

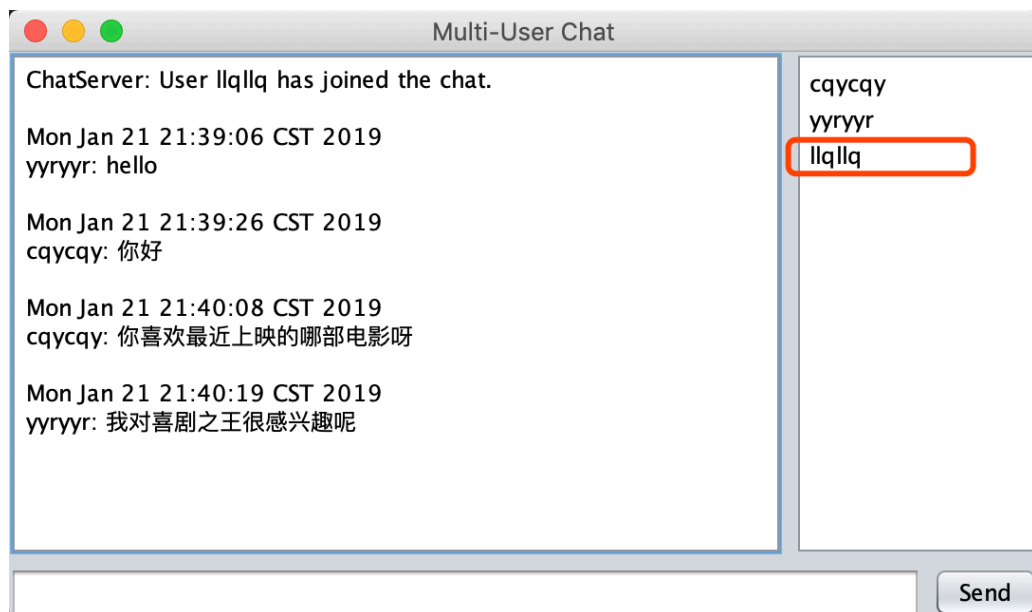


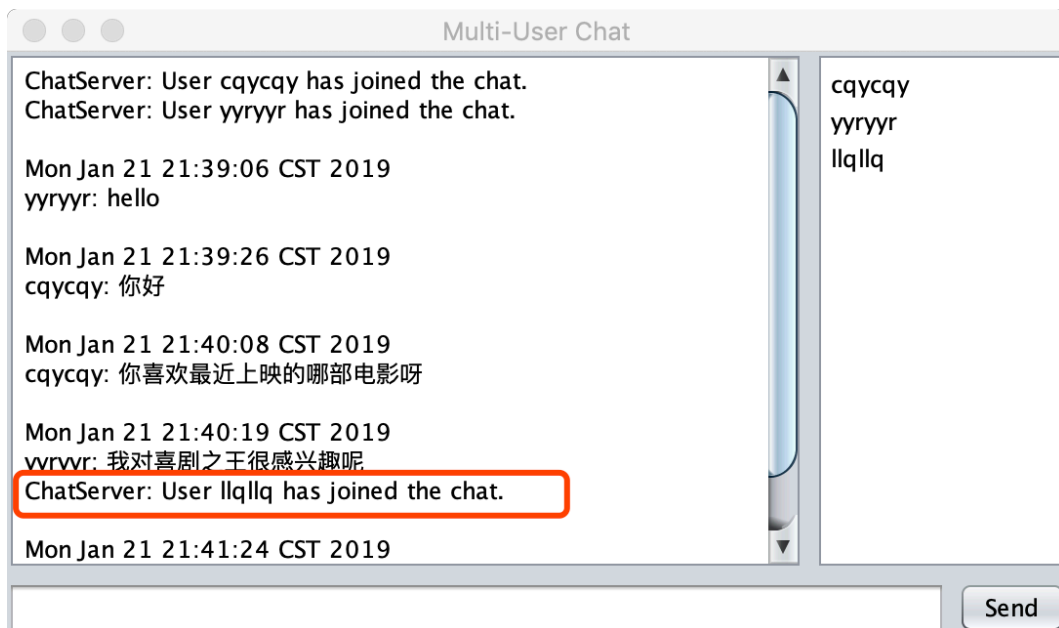
4.2.8 yyryyr 收到消息:



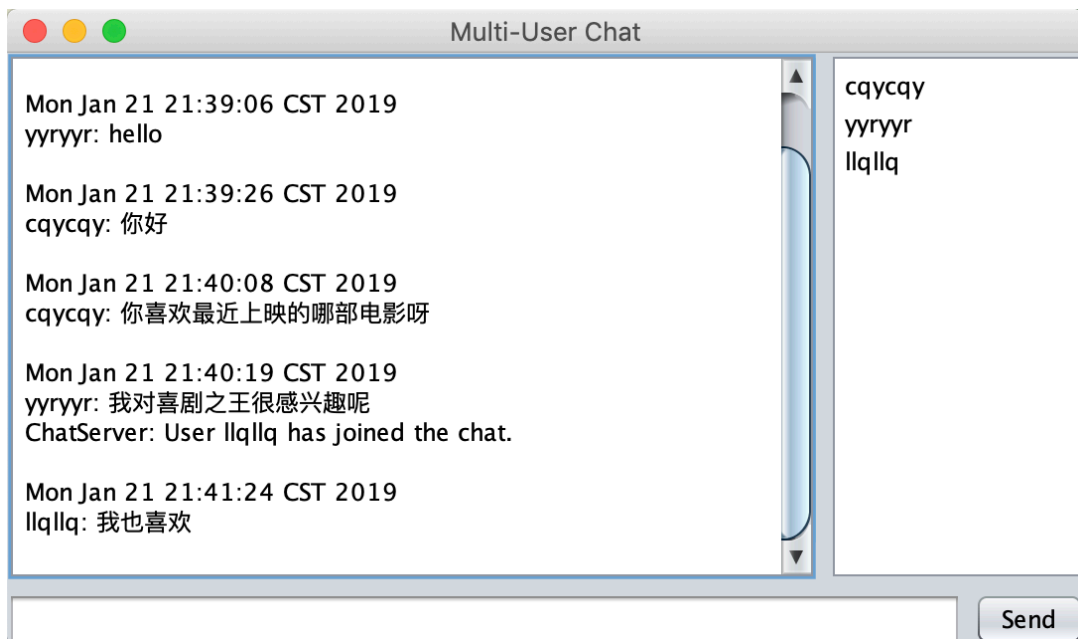
4.2.9 第三个用户 llqllq 登陆:

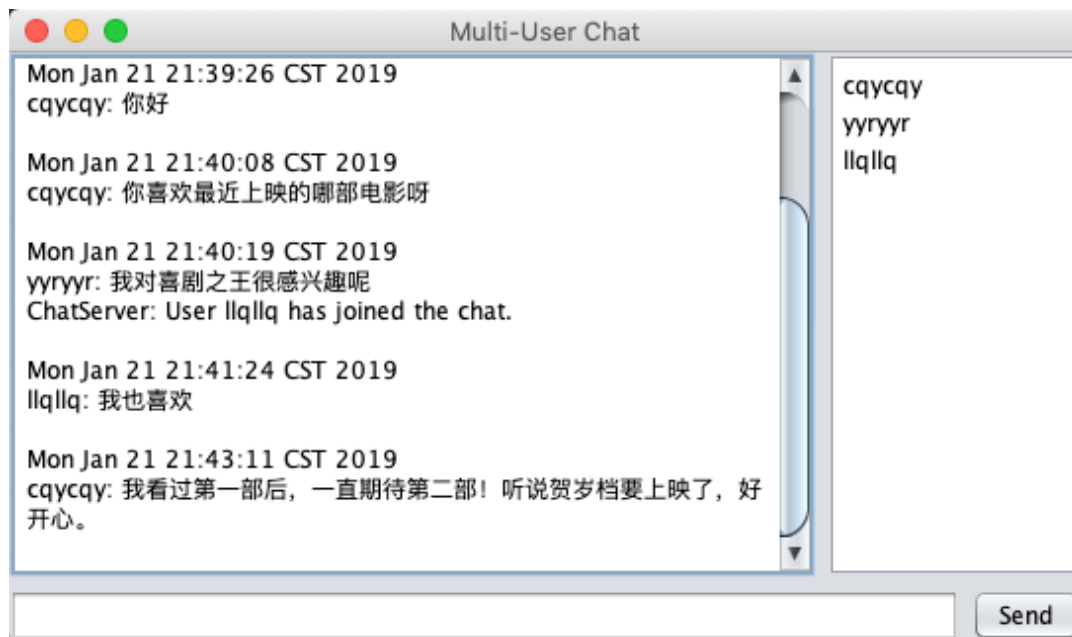
加载了历史消息。



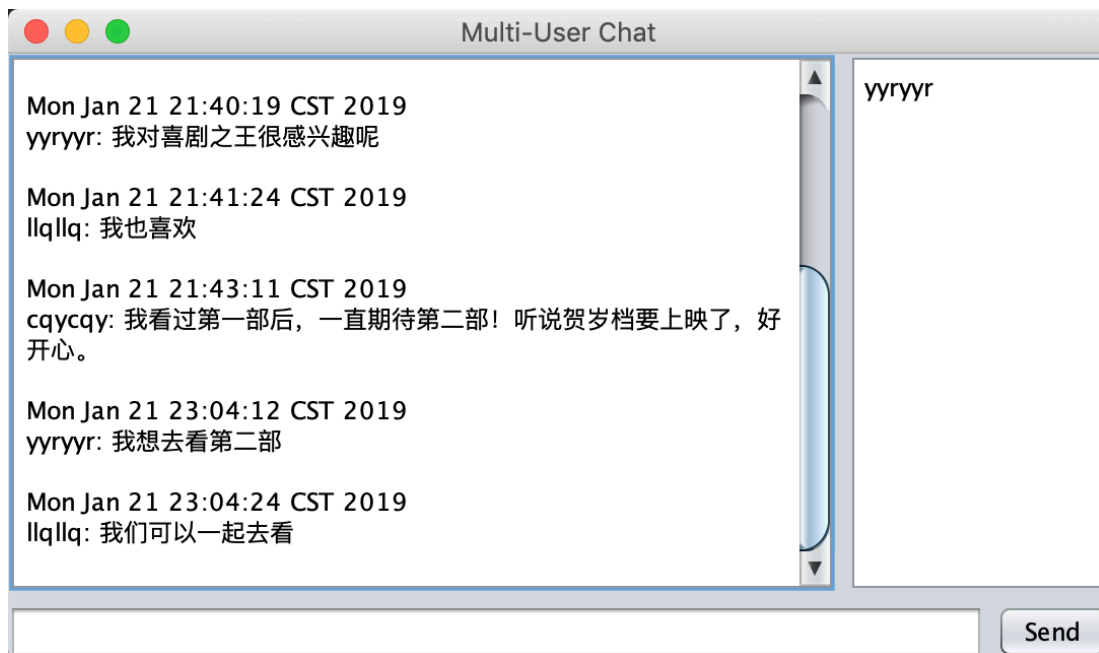


4.2.10 三个人消息同步:





4.2.11 三人退出后，yyryyr 退出后重新登陆，加载历史消息：



五、心得感想

此次的实验，我对于 Java 的 gui 编程有了一定的认识，最重要的是 socket 编程和多线程处理机制，我在一开始对多线程是有些畏惧心理的，但是 Java 的多线程处理起来，比 c 要容易太多。比起操作系统的多线程编程，Java 实现多线程的方式就简单很多，Java 已经帮我做好了同步互斥这些东西，感叹一句封装的好处。