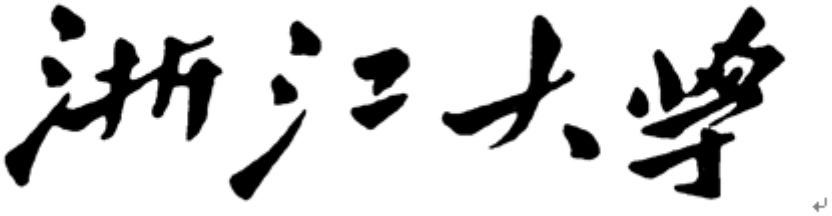


Java Homework4

学号：3220104519

姓名：蔡佳伟

一、引言

本次作业，我开发的是一个聊天室项目。

该项目支持多人在线聊天，可以开启多个界面，多个用户同时进入聊天室，每个用户可以在界面编辑文字，进行发言。每个用户的发言会被当前聊天室内所有用户看到，每个用户的进入和退出、当前聊天室内所有用户也会被所有用户看到。用户登录时可以自定义用户名，如果想要改变端口，服务器同时改变监听端口的前提下也可以正常工作。

服务器界面显示用户的登录和登出，也显示当前监听的端口。服务器支持用户搜索功能。

数据库使用MySQL，用户登录成功后用户名被存入数据库中，退出和异常关闭时，用户信息会被删除。

1.1 设计说明

本程序只采用了Java语言，在IntelliJ IDEA下进行编辑、编译与调试，全部由本人独立开发完成。

1.2 设计目的

聊天室是一个经典而有用的场景，主要功能为用户可以登录聊天室进行聊天。开发聊天室项目不仅符合本次java作业的要求，编写了一个网络程序；使用了GUI编程，用户界面用GUI实现；使用了网络编程，支持多组用户同时实现；同时使用了数据库记录用户注册的用户名，满足了Bonus之一。

本次程序设计的主要目的如下：

- 提升设计能力与使用Java语言开发较大型项目的能力；
- 熟悉Java语言中的GUI编程和Swing框架的使用；
- 熟悉Java语言中网络编程相关的类和方法，能够利用Java语言实现多台计算机之间的网络通信；
- 熟悉Java语言与MySQL数据库的交互，使用Connector/J，使用JDBC驱动，实现程序与数据库的交互。

二、总体设计

2.1 功能模块设计

2.1.1 客户端（Client）模块

功能如下：

- 通过指定的IP地址和服务器连接到指定的服务器；
- 发送消息给其他连接到同一服务器的客户端；
- 查看当前在线用户信息，包括在线用户数，用户名，上线时间；
- 在其他客户端连接服务器或断开连接时，展示提示信息。
- 与数据库进行交互，添加用户和断开用户时，改变数据库用户信息。

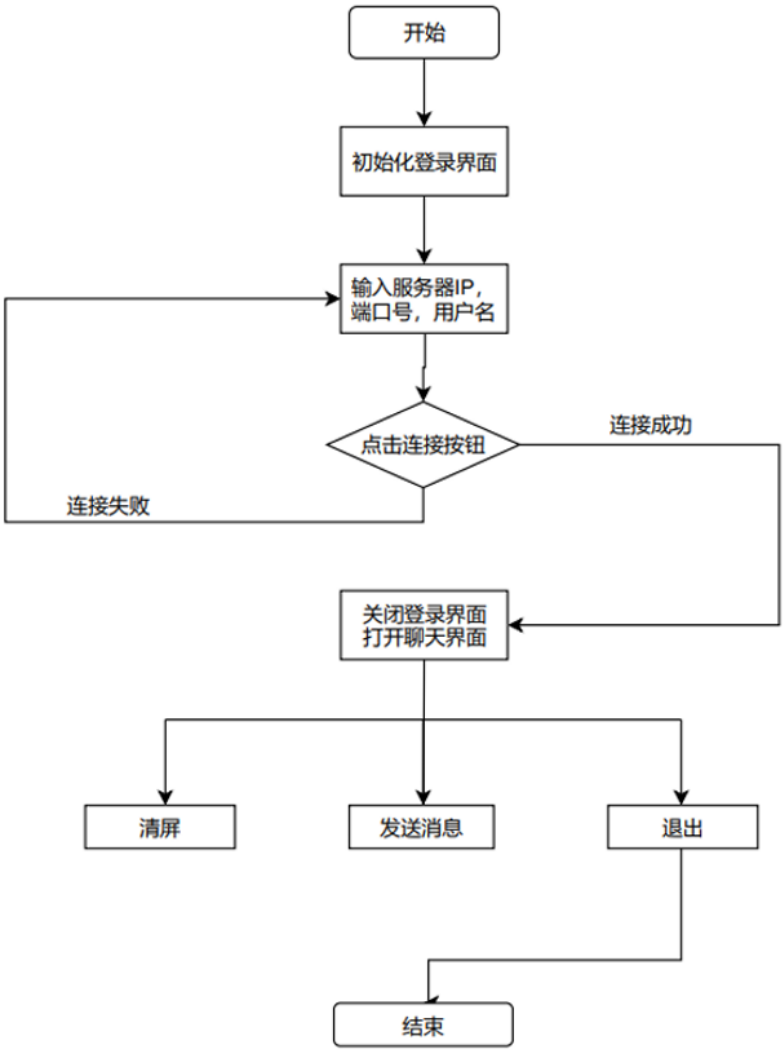
2.1.2 服务器（Server）模块

功能如下：

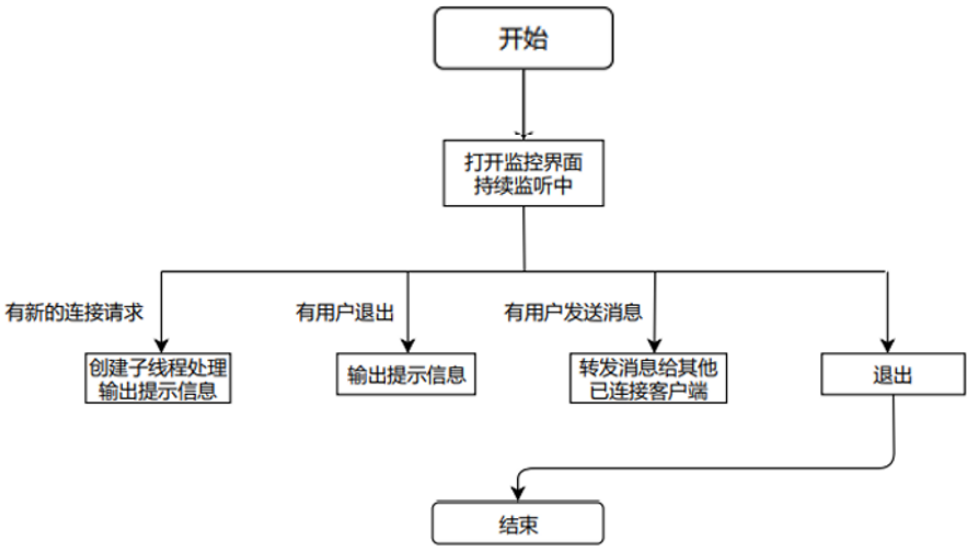
- 可以监听指定的端口，并与发出连接请求的客户端建立TCP连接；
- 接受到连接请求并建立连接之后，通过创建一个子线程来实现与该客户端的后续通信，主线程依然要持续监听，以实现支持多组用户同时使用的目的；
- 维护一个当前以连接的客户端的列表，记录当前在线用户数；
- 转发一个客户端的消息到所有已连接的客户端。
- 与数据库进行交互，关闭服务器时，删除数据库所有用户信息。

2.2 流程图设计

服务器流程图如下：

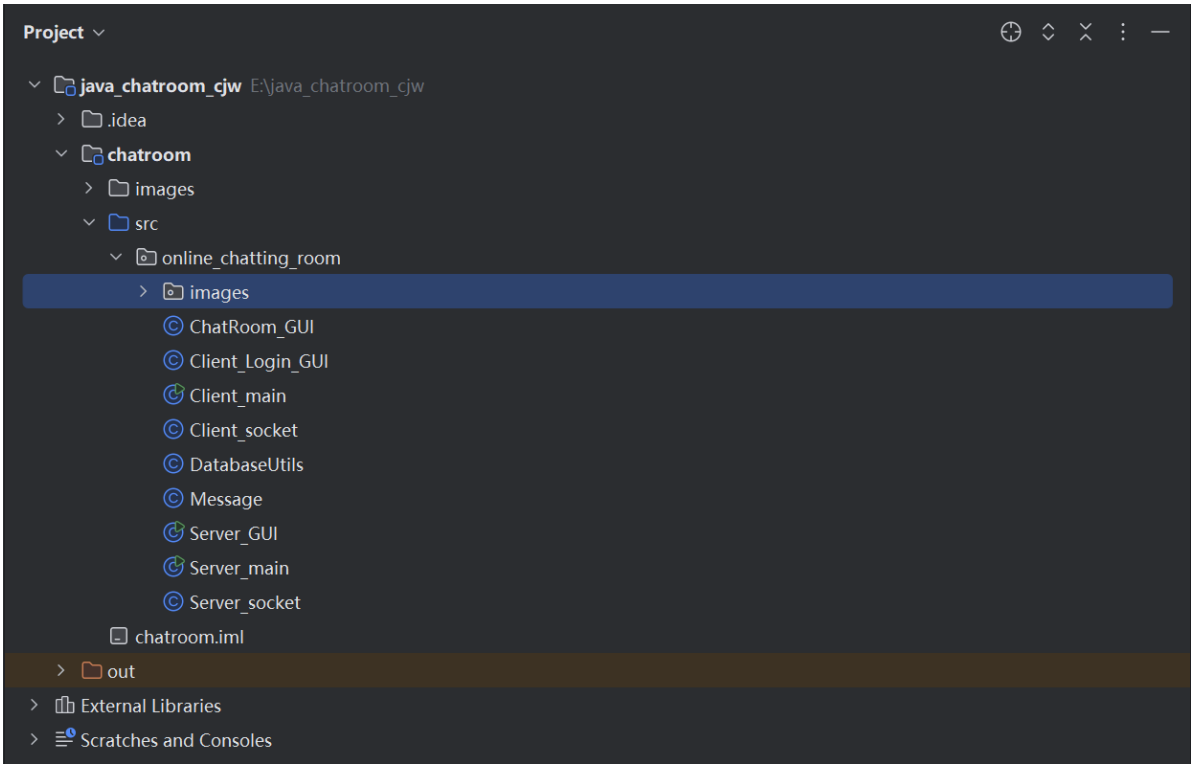


客户端流程图如下：



2.3 整体项目设计

整体项目结构如下：



其中两个images文件夹最后没有使用，可以忽略。src是工程文件夹，里面online_chatting_room是我的聊天室项目，里面有以下的这些java文件，chatroom.iml是生成产物。详细的java文件说明见第三部分。

三、详细设计

3.1 DatabaseUtils.java

这个类是数据库交互的类，首先配置了连接数据库的所需变量。

```
private static final String DB_URL = "jdbc:mysql://localhost:3306/chat_app?
useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";  // 用户名
    private static final String PASS = "040517cc";  // 密码
```

创建了连接数据库的方法，在insert和delete中方法调用

```
public static Connection getConnection() throws SQLException
```

然后实现了插入用户名到数据库和删除数据库中指定用户名的两个方法：

```
 public static boolean insertUsername(String username) {
     String query = "INSERT INTO users (username) VALUES (?)";

        try (Connection connection = getConnection();
             PreparedStatement stmt = connection.prepareStatement(query)) {

            stmt.setString(1, username);
```

```
            int rowsAffected = stmt.executeUpdate();

            return rowsAffected > 0;  // 如果插入成功，返回 true
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
}

public static boolean deleteUsername(String username) {
    String query = "DELETE FROM users WHERE username = ?";

        try (Connection connection = getConnection();
             PreparedStatement stmt = connection.prepareStatement(query)) {

            stmt.setString(1, username);
            int rowsAffected = stmt.executeUpdate();

            return rowsAffected > 0;  // 返回 true 如果删除成功
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
}
```

在具体实现中，先尝试连接数据库，连接成功以后，使用`PreparedStatement`和数据库的语句完成数据库的更新。

3.2 ChatRoom_GUI.java

这几个GUI类主要使用了Java的Swing框架进行设计，该类继承自`javax.swing`包中的`JFrame`类，还实现了`ActionListener`接口，以相应用户按下按钮后的事件。整体元素在JPanel上的布局使用了自由布局的方式。

类中的变量和方法主要如下：

```
private JTextArea chatTa;         // Chat display area
private JTextArea userlistTa;     // Online users list
private JTextField inputTf;       // Input area for typing messages
private JButton sendBtn;          // Send message button
private JButton clearBtn;         // Clear chat button
```

```
public ChatRoom_GUI() {
        init();
    }
private void init() {
    }
private void sendMessage() {
        String message = inputTf.getText();
        if (message.isEmpty()) return;

        // 创建一个message实例
        Message msg = new Message(Message.MESSAGE, message);
        Client.sendMessage(msg);
```

```
        inputTf.setText("");  // 发送信息之后清空发送框
    }
private void clearChat() {
        chatTa.setText("");
    }
```

在`init`方法中，初始化了聊天框大小，按钮位置等基本信息，然后把所有components添加到了container中，完成初始化。

3.3 Server_GUI.java

该GUI的设计思路和3.2部分类似：

类中的变量和方法主要如下：

```
private JTextArea MonitorTa; // 显示服务器信息
private JTextField searchField; // 搜索框
private JTextArea userListTa; // 显示在线用户列表

private Server_socket server;
```

```
public Server_GUI(int port) {
        this.server = new Server_socket(port, this); // 传入端口和Server_GUI实例
        init();
        startServer();
    }

private void init() {
    }

private void filterUserList() {
        String searchText = searchField.getText().toLowerCase();
        String userListContent = server.getUserListContent(); // 获取用户列表内容
        String[] users = userListContent.split("\n");

        StringBuilder filteredUsers = new StringBuilder();
        for (String user : users) {
            if (user.toLowerCase().contains(searchText)) {
                filteredUsers.append(user).append("\n");
            }
        }

        userListTa.setText(filteredUsers.toString());
    }

// 更新监控信息区域
    public void MonitorTaDisplay(String message) {
        MonitorTa.append(message);
        MonitorTa.setCaretPosition(MonitorTa.getText().length()); // 自动滚动到底部
    }

// 启动服务器
    private void startServer() {
        new Thread(() -> {
```

```
            server.listen(); // 服务器开始监听
        }).start();
    }

public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new Server_GUI(1235)); // 启动服务器并监听
端口
    }
```

Server_GUI直接指定了需要监听的端口号，`startServer`使得服务器开始监听，而`init`方法又调用了`startServer`方法，使得可以顺利监听。

3.4 Server_socket.java

该类主要处理与客户端的连接，主要运用了Java中Socket编程的思想和多线程编程的思想，主要变量和方法如下：

```
private int port;
private ArrayList<InteractWithClient> clientlist;
private int clientid;
private Server_GUI server;

private SimpleDateFormat currenttime;
```

```
// 构造函数，时间取用当前时间
    public Server_socket(int port, Server_GUI server) {
        this.port = port;
        this.clientlist = new ArrayList<>();
        this.clientid = 0;
        this.currenttime = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        this.server = server;
    }

// 监听端口
    public void listen() {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            server.MonitorTaDisplay("正在监听端口 " + port + "\n");
            while (true) {
                Socket socket = serverSocket.accept();
                InteractWithClient client = new InteractWithClient(socket);
                addClient(client);
                startClientThread(client);
                enterRoomMessage(client);
                updateClientList();
            }
        } catch (IOException e) {
            server.MonitorTaDisplay("创建ServerSocket过程出错: " + e + "\n");
        }
    }

private String ListUserInfo() {
    }

private void addClient(InteractWithClient client) {
```

```
        clientlist.add(client);
    }

    private void startClientThread(InteractWithClient client) {
        client.start();
    }

    private void enterRoomMessage(InteractWithClient client) {
    }

    private synchronized void notifyClients(String message) {
    }

    private synchronized void updateClientList() {
    }

    private void sendToAllClients(Message msg) {
    }

    public synchronized void remove(int id) {
        clientlist.removeIf(client -> client.id == id);
    }

    public String getUserListContent() {
    }

    class InteractWithClient extends Thread {

        InteractWithClient(Socket socket) {
        }

        private void initializeStreams() {
            try {
                socketOutput = new ObjectOutputStream(socket.getOutputStream());
                socketInput = new ObjectInputStream(socket.getInputStream());
                userName = (String) socketInput.readObject();
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }

        public void run() {
        }

        private void processClientMessage(Message clientMessage) {
        }

        private void handleClientDisconnect() {
        }

        // 发送信息
        public boolean sendMessage(Message msg) {
            if (!socket.isConnected()) {
                return false;
            }
```

```
            try {
                socketOutput.writeObject(msg);
            } catch (IOException e) {
                return false;
            }
            return true;
        }
    }
```

`listen()`方法用于监听端口并等待客户端连接，是该类设计的核心。主要流程如下：

- 根据指定的端口号实例化一个serverSocket对象
- 调用serverSocket类的accept()方法，该方法将一直等待，直到客户端连接到服务器上的给定端口；
- 如果通信建立成功，accpet()方法返回服务器上的一个新的socket引用，该socket连接到客户端的socket
- 创建一个子线程处理后续与该客户端的通信
- 将新创建的子线程加入保存与客户端通信的子线程的ArrayList中
- 发送提示消息给其他已连接的客户端，在服务器也输出相关提示信息，循环，继续调用accept()监听端口

可以看到，在该类中，我们还实现了一个InteractWithClient类，这个类继承自Thread类，该类的实例就是与客户端通信的子线程。

该类的run()方法中定义如何处理从客户端中收到的消息，首先通过套接字输入流读取一个Message对象，然后直接调用notifyClients()方法，将收到的消息发送给当前所有连接到服务器的客户端，如果接收消息的过程中捕捉到异常，说明与指定客户端的连接已经断开。此时将与客户端交互的子进程移出ArrayList，并将该客户端下线的消息转发给当前服务器的所有客户端，并在服务器显示。

```
class InteractWithClient extends Thread {
        int id;
        Socket socket;
        ObjectInputStream socketInput;
        ObjectOutputStream socketOutput;
        String userName;
        String linktime;

    public void run() {
            while (true) {
                try {
                    Message clientMessage = (Message) socketInput.readObject();
                    processClientMessage(clientMessage);
                } catch (IOException | ClassNotFoundException e) {
                    break;
                }
            }
            handleClientDisconnect();
        }
        }
```

另外，notifyClients(String message)方法主要是给当前所有连接到服务器的客户端发送消息，实现过程为：给需要转发的消息加上一个时间戳，然后组装一个类型为MESSAGE的Message对象，通过遍历clientlist中的子线程对象，一一调用他们的sendMessage()方法来将消息发送给所有客户端。

```
private synchronized void notifyClients(String message) {
        String notice = currenttime.format(new Date()) + "\n" + message +
"\n\n";
        sendToAllClients(new Message(Message.MESSAGE, notice));
    }
```

InteractWithClient类还实现了sendMessage方法，该方法的主要作业是发送消息给客户端，通过向套接字输出流中写入Message类的对象msg来实现。

3.5 Server_main.java

这个main类只包含一个主方法main()，在main()中，Client_main类实例化了一个Client_Login_GUI()类的对象来显示登录界面。

```
package online_chatting_room;

public class Server_main {
    public static Server_GUI server;

    public static void main(String[] args) {
        server = new Server_GUI(1235);
    }
}
```

3.6 Client_Login_GUI.java

该类同样主要使用了Java的Swing框架进行设计，该类继承自javax.swing包中的JFrame类，还实现了ActionListener接口来相应用户按下登录按钮的事件。主要变量和方法如下：

```
private JLabel serverLabel = new JLabel("服务器IP地址:");
    private JLabel portLabel = new JLabel("端口号:");
    private JLabel userLabel = new JLabel("用户名:");
    private JLabel titleLabel = new JLabel("登录界面");

    // TextField信息
    private JTextField serverTf = new JTextField("127.0.0.1", 15);
    private JTextField portTf = new JTextField("1235", 15);
    private JTextField userTf = new JTextField(15);

    // Button信息
    private JButton loginBtn = new JButton("登录(Enter)");

    // Label信息
    private JLabel bgLabel;

    // 创建GUI和Socket实例
    public static ChatRoom_GUI chatRoom = new ChatRoom_GUI();
    public static Client_socket Client;
```

```
public Client_Login_GUI() {
        init();
```

```
        addListeners();

        // 窗口关闭时删除数据库中的用户名信息
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                String username = userTf.getText().trim();
                if (!username.isEmpty()) {
                    boolean isDeleted = DatabaseUtils.deleteUsername(username);
                    if (isDeleted) {
                        System.out.println("用户名 " + username + " 已删除");
                    } else {
                        System.out.println("删除用户名 " + username + " 失败");
                    }
                }
                System.exit(0);  // 关闭程序
            }
        });
    }

    // init函数
    private void init() {
    }

    private void addListeners() {
        // 登录按钮点击事件
        loginBtn.addActionListener(e -> forLogin(serverTf.getText().trim(),
portTf.getText().trim(), userTf.getText().trim()));

        // 设置Enter作为快捷键
        KeyListener enterKeyListener = new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                    loginBtn.doClick();
                }
            }
        };

        serverTf.addKeyListener(enterKeyListener);
        portTf.addKeyListener(enterKeyListener);
        userTf.addKeyListener(enterKeyListener);
        loginBtn.addKeyListener(enterKeyListener);

        // Esc 键关闭窗口
        getRootPane().registerKeyboardAction(e -> System.exit(0),
                KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0),
                JComponent.WHEN_IN_FOCUSED_WINDOW);
    }

    // 登录函数
    private void forLogin(String serverIP, String port, String username) {
    }
```

其中按下登录按钮的相应函数，传入的三个参数分别是用户输入的服务器IP地址，端口号和用户名，在检查完输入的合法性之后，该函数创建一个新的Client_socket类对象，并调用connectServer()方法来连接服务器，如果连接成功，则设置当前登录页面不可见，设置聊天室界面为可见，这样就实现了界面的跳转。

```
private void forLogin(String serverIP, String port, String username) {
        int portnum;
        if (serverIP.isEmpty()) {
            JOptionPane.showMessageDialog(this, "服务器IP地址不能为空", "输入错误",
JOptionPane.ERROR_MESSAGE);
            return;
        }
        if (port.isEmpty()) {
            JOptionPane.showMessageDialog(this, "端口号不能为空", "输入错误",
JOptionPane.ERROR_MESSAGE);
            return;
        }
        if (username.isEmpty()) {
            JOptionPane.showMessageDialog(this, "用户名不能为空", "输入错误",
JOptionPane.ERROR_MESSAGE);
            return;
        }
        try {
            portnum = Integer.parseInt(port);
            if (portnum < 1 || portnum > 65535) {
                throw new NumberFormatException();
            }
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "端口号必须是 1-65535 之间的整数",
"输入错误", JOptionPane.ERROR_MESSAGE);
            return;
        }

        boolean isRegistered = DatabaseUtils.insertUsername(username);
        if (!isRegistered) {
            JOptionPane.showMessageDialog(this, "用户名注册失败，请稍后重试");
        }

        // 创建客户端连接
        Client = new Client_socket(serverIP, portnum, username, chatRoom);

        if (!Client.connectServer()) {
            JOptionPane.showMessageDialog(this, "无法连接到服务器！请检查IP和端口。",
"连接失败", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // 登录成功
        setVisible(false);
        chatRoom.setVisible(true);
    }
```

3.7 Client_socket.java

该类主要处理与服务器的连接，需要实现与服务器连接的有关的一些方法，主要运用到Java中Socket编程的思想以及多线程编程的思想，主要变量和方法如下：

```
private String serverIP;
private int port;
private Socket cliSocket;
private ObjectInputStream socketIn;
private ObjectOutputStream socketOut;
private String username;
private ChatRoom_GUI chatRoom;
```

```
    // 构造Client_socket类
    public Client_socket(String serverIP, int port, String username,
ChatRoom_GUI chatRoom) {
        this.serverIP = serverIP;
        this.port = port;
        this.username = username;
        this.chatRoom = chatRoom;
    }

    public boolean connectServer() {
    }

    // 建立套接字连接
    private boolean connectSocket() {
        try {
            cliSocket = new Socket(serverIP, port);
        } catch (Exception e) {
            // showErrorDialog("连接失败！服务器可能关闭，或者输入的服务器IP地址或端口有
误");
            return false;
        }
        showSuccessDialog("连接服务器成功");
        return true;
    }

    // 初始化输入输出流
    private boolean initializeStreams() {
        try {
            socketIn = new ObjectInputStream(cliSocket.getInputStream());
            socketOut = new ObjectOutputStream(cliSocket.getOutputStream());
        } catch (Exception e) {
            showErrorDialog("创建套接字的I/O流出错 " + e);
            return false;
        }
        return true;
    }

    // 向服务器发送用户名
    private boolean sendUsernameToServer() {
    }
```

```
    // 启动一个线程监听来自服务器的消息
    private void startListeningThread() {
        new InteractWithServer().start();
    }

    private void showErrorDialog(String message) {
        JOptionPane.showMessageDialog(null, message, "错误",
JOptionPane.ERROR_MESSAGE);
    }

    // 监听成功
    private void showSuccessDialog(String message) {
        JOptionPane.showMessageDialog(null, message);
    }

    // 发送消息
    public void sendMessage(Message msg) {
        try {
            socketOut.writeObject(msg);  // 向服务器发送消息
        } catch (Exception e) {
            showErrorDialog("消息发送过程出错！");
        }
    }

    // 线程：接收并打印服务器发送的消息
    class InteractWithServer extends Thread {
        @Override
        public void run() {
            }
        }

        private void handleServerMessage(Message servermsg) {

        }

        private void handleDisconnection() {

        }
    }
```

其中connectServer()方法用于连接服务器，是该类设计的核心，主要流程为：

- 实例化一个Socket对象，指定服务器名称和端口号来请求连接
- 如果连接成功，接着创建套接字的输入输出流
- 如果创建也成功，则启动一个子线程来完成后续与服务器的通信

```
public boolean connectServer() {
        if (!connectSocket()) return false;  // 连接套接字
        if (!initializeStreams()) return false;  // 初始化输入输出流
        if (!sendUsernameToServer()) return false;  // 向服务器发送用户名

        startListeningThread();  // 启动监听线程
        return true;
    }
```

可以看到在CLient_Socket中，我们还实现了一个InteractWithServer类，这个类继承自Thread类，需要实现其run()方法。InteractWithServer类的实例也即与服务器通信的子线程，该类的run()方法中定义了该如何处理从服务器收到的消息：

首先通过套接字输入流读取一个Message对象，然后取出实际消息类型和内容，如果类型为MESSAGE，则在聊天框中显示该消息；如果是USERLIST，则在在线用户列表框中显示该消息。如果接收消息的过程中捕获到异常，说明与服务器的连接已经断开，弹出提示对话框并关闭用户端。

```
// 线程：接收并打印服务器发送的消息
    class InteractWithServer extends Thread {
        @Override
        public void run() {
            while (true) {
                try {
                    // 接收消息
                    Message servermsg = (Message) socketIn.readObject();
                    handleServerMessage(servermsg);  // 处理服务器的消息
                } catch (Exception e) {
                    handleDisconnection();  // 处理断开连接的情况
                    break;
                }
            }
        }

        // 处理服务器发送的消息
        private void handleServerMessage(Message servermsg) {
            switch (servermsg.getType()) {
                case Message.MESSAGE:
                    chatRoom.ChatTaDisplay(servermsg.getContent());  // 显示聊天消
息
                    break;
                case Message.USERLIST:
                    chatRoom.UserlistTaClear();  // 清空用户列表
                    chatRoom.UserlistTaDisplay(servermsg.getContent());  // 更新用
户列表
                    break;
            }
        }

        // 处理与服务器断开连接的情况
        private void handleDisconnection() {
            showErrorDialog("与服务器的连接已断开！");
            deleteUsername(username);
            chatRoom.setVisible(false);  // 隐藏聊天室界面
        }
    }
```

3.8 Client_main.java

这个main类同样只包含一个主方法main()，在main()中，Server_main类实例化了一个Server_GUI()类的对象来显示服务器界面。

```
package online_chatting_room;

public class Client_main {

    public static Client_Login_GUI client;

    public static void main(String[] args) {
        client = new Client_Login_GUI();
    }
}
```

3.9 Message.java

Message类定义了客户端和服务器之间传递消息的规范格式，它需要实现Serializable接口，这样才能被ObjectOutputStream转换为字节流。

Message中定义了MESSAGE和USERLIST两种消息的类型，用来表示该消息是正常消息还是服务器发送的维护在线列表的消息。Message类又两个变量，分别是String类的content和int型的type，分别保存消息内容和类型，两个方法分别返回这两个成员变量的内容。

```
import java.io.Serializable;

public class Message implements Serializable{
    // Message类
    static final int MESSAGE = 0, USERLIST = 1;
    private int type;
    private String content;

    Message(int type, String content) {
        this.type = type;
        this.content = content;
    }

    /* Funtion to get type of the Message*/
    int getType() {
        return type;
    }

    /* Funtion to get content of the Message*/
    String getContent() {
        return content;
    }
}
```

四、测试与运行

4.1 项目运行

4.1.1 数据库配置

我是在IntelliJ IDEA中运行的，助教gg或者鲁老师在解压后，可以用IDEA打开。首先配置数据库，sql语言的配置如下，我没有在java文件中写配置表格的脚本，所以需要手动添加。

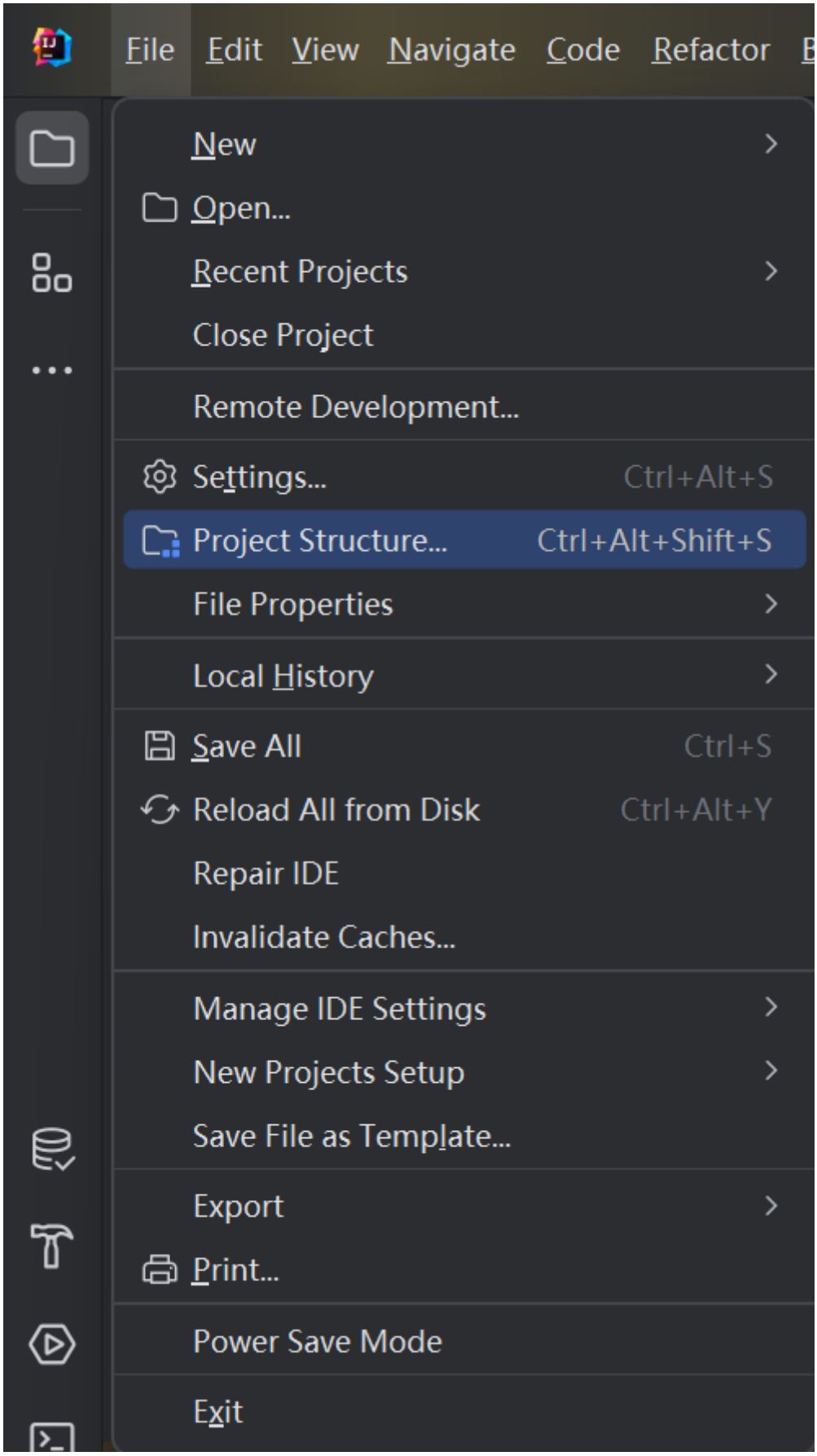
登录进自己的数据库后：

```
CREATE DATABASE chat_app;

USE chat_app;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL
);
```

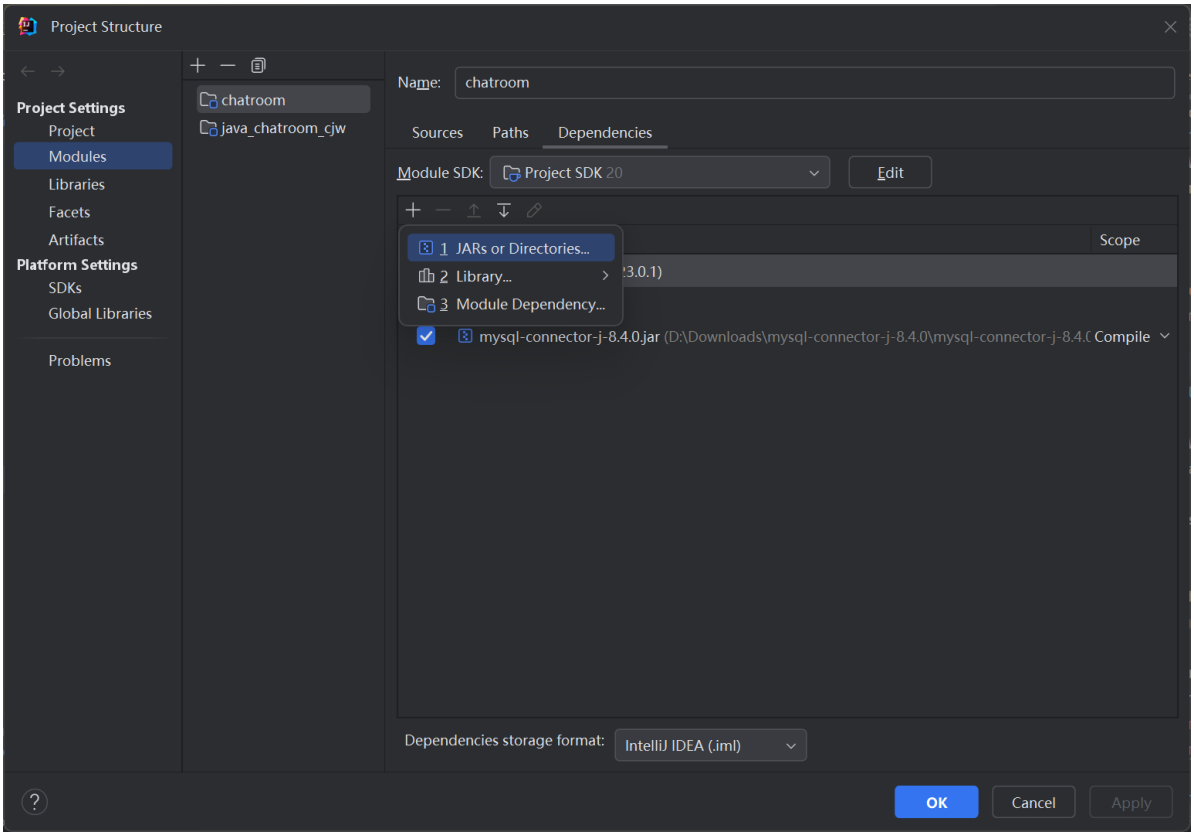
在`DatabaseUtils`类中，修改PASS为自己的数据库密码。

（以下操作不知道是否必须，还是展示一下）

选择`File`一栏中，`Project Structure`。



选择`Project Settings`中的`Modules`，选择`chatroom`项目，点击加号，选择`1 JARS or Directories...`，在自己的本地目录中，选择`mysql-connector-j-8.4.0.jar`，点击后选择`OK`确认。



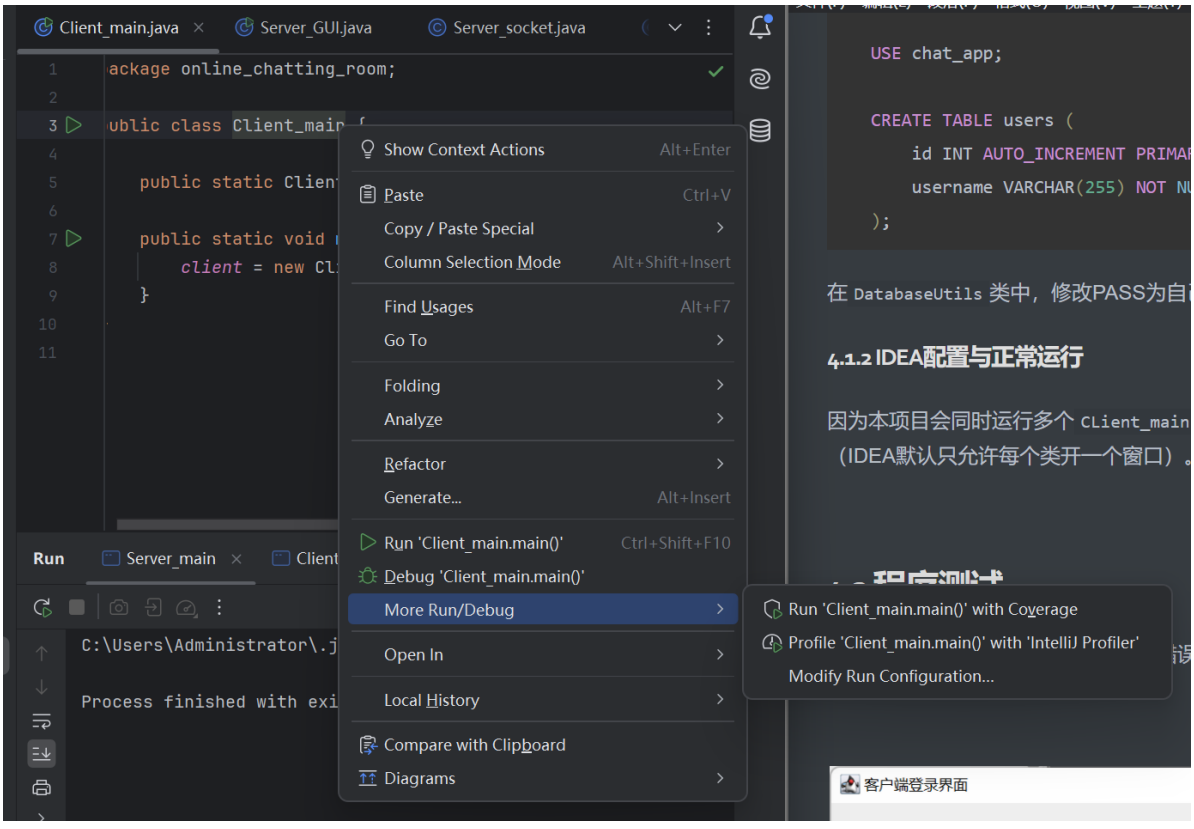
如果没有这个文件，可以在[这个网站]点击下载，选择8.4.0版本，`Platform Independent`，下载`zip`后缀的文件，解压后，进入文件夹，即可获得这个文件。

https://downloads.mysql.com/archives/c-j/

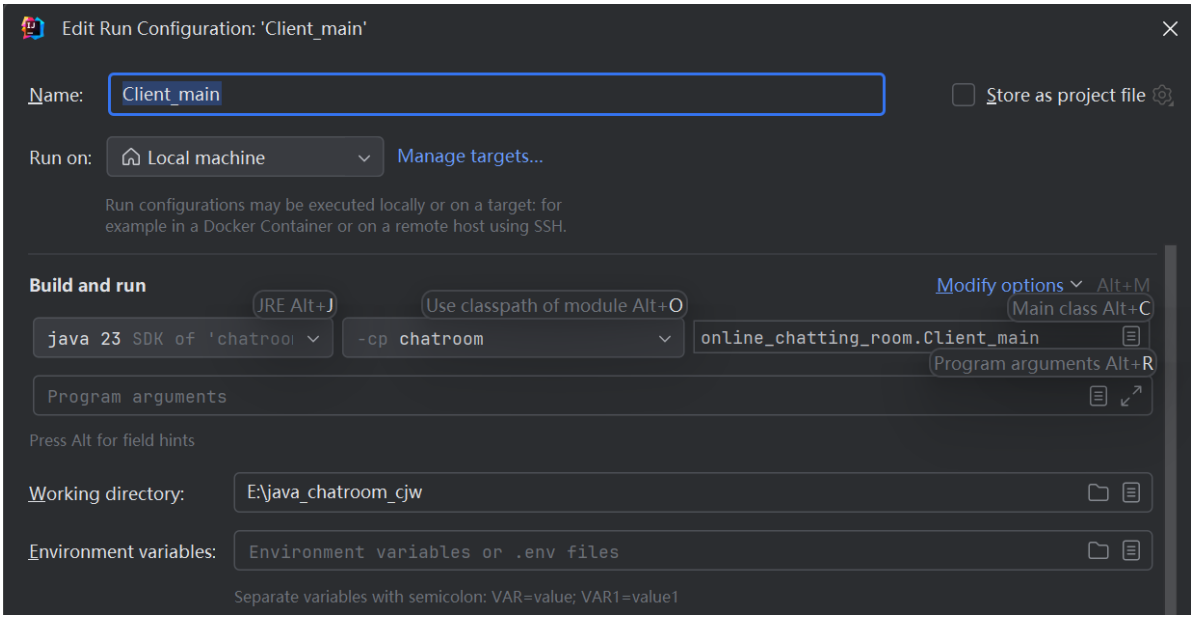
4.1.2 IDEA配置与正常运行

因为本项目会同时运行多个`CLient_main`类，所以还需要修改IDEA的默认设置（IDEA默认只允许每个类开一个窗口）。

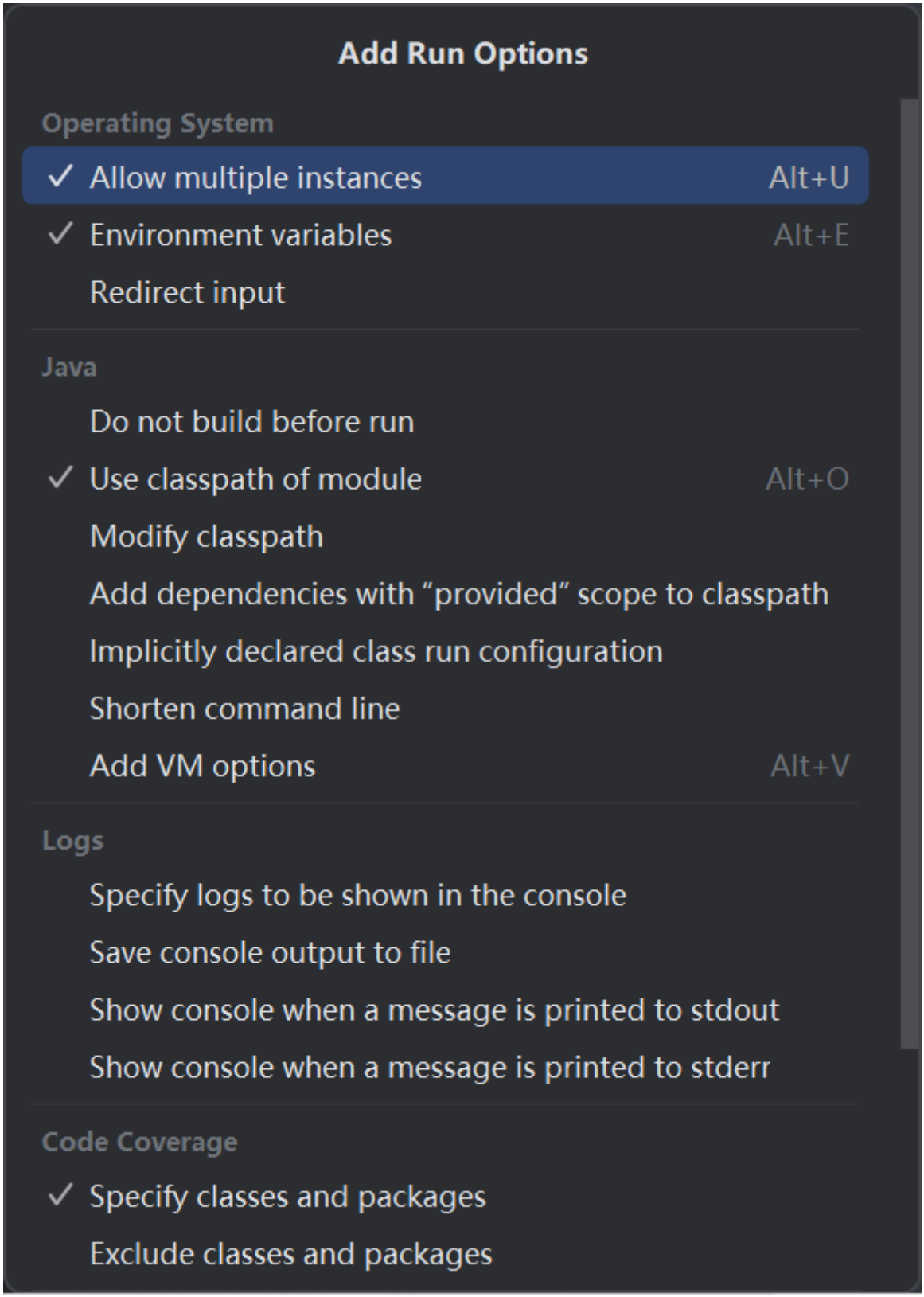
如图，右键点击`Client_main`类，选择`More Run/Debug`中的`Modify Run Configuration...`一栏



选择`Modify options`选项



勾选`Allow multiple instances`，然后在Edit界面点击OK，完成设置。



之后可以正常运行，在Client窗口点击登录前，需要运行Server窗口，否则会发生错误，错误在下方展示。需要几个用户，即开启几个Client窗口。

4.2 程序测试

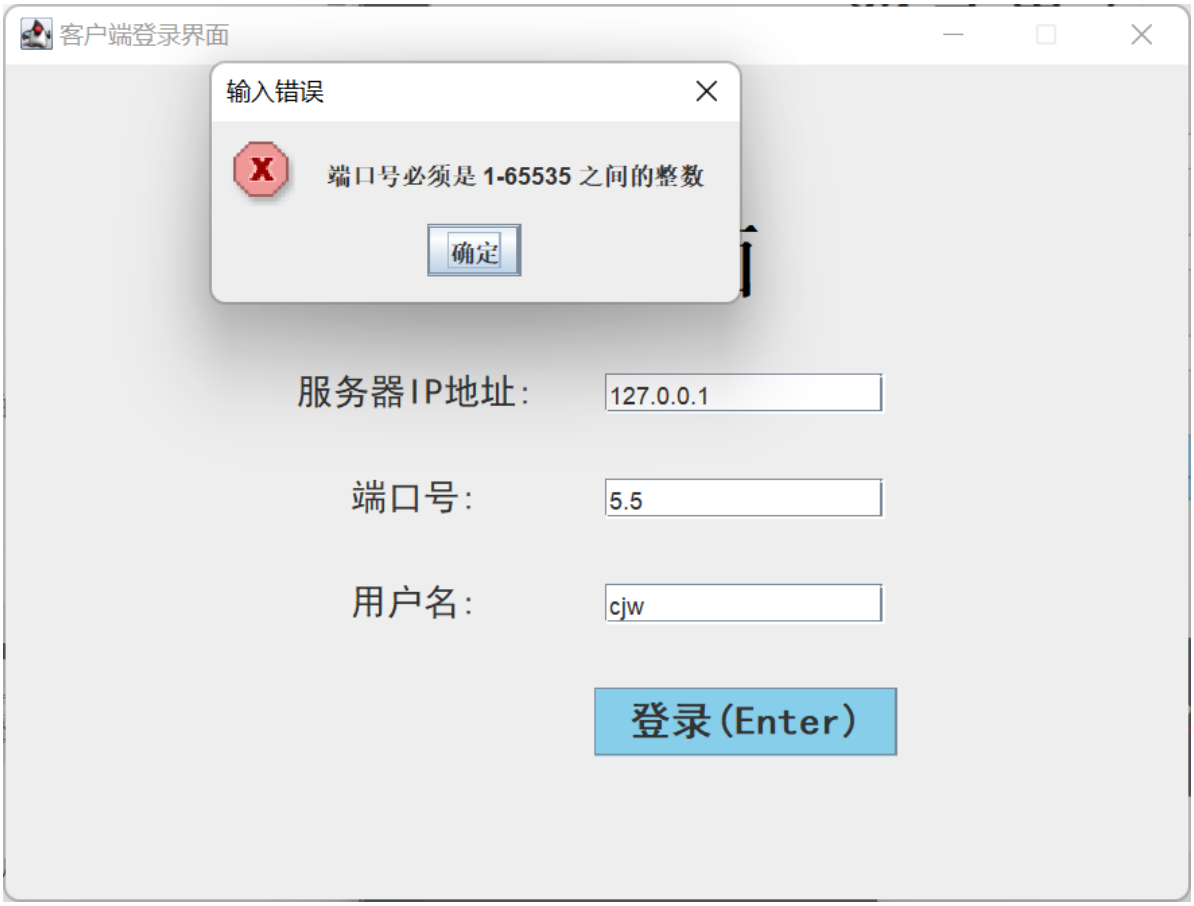
该部分完整展现了程序的运行和一些错误情况的处理。

这是Client的主界面，服务器IP和端口都默认有输入，不需要更改（当然也可以改），在下方输入自己的用户名，用户名不能重复。输入成功后，点击登录或Enter即可登录。

这是IP地址为空的情况，会显示IP地址不能为空。

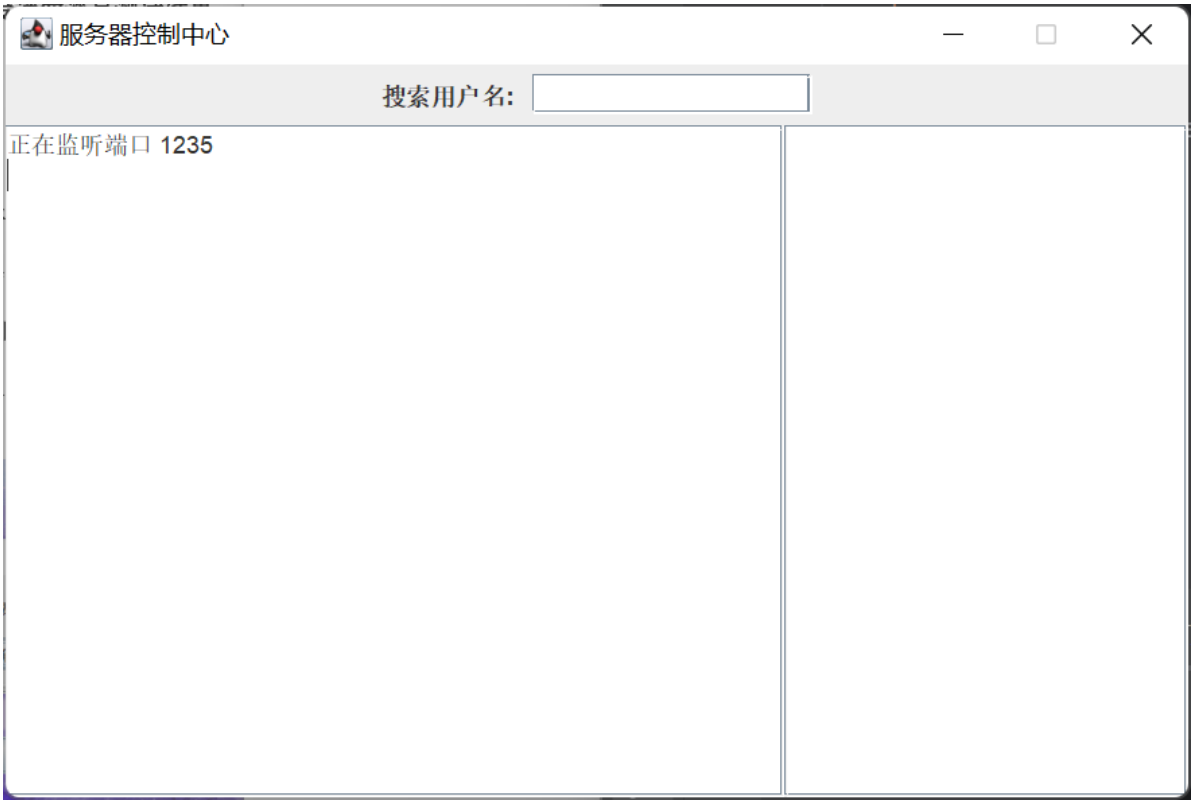
这是IP地址不合法的情况，会显示无法连接到服务器。

这是端口号不合法的情况，会提示端口号是1-65535的整数。



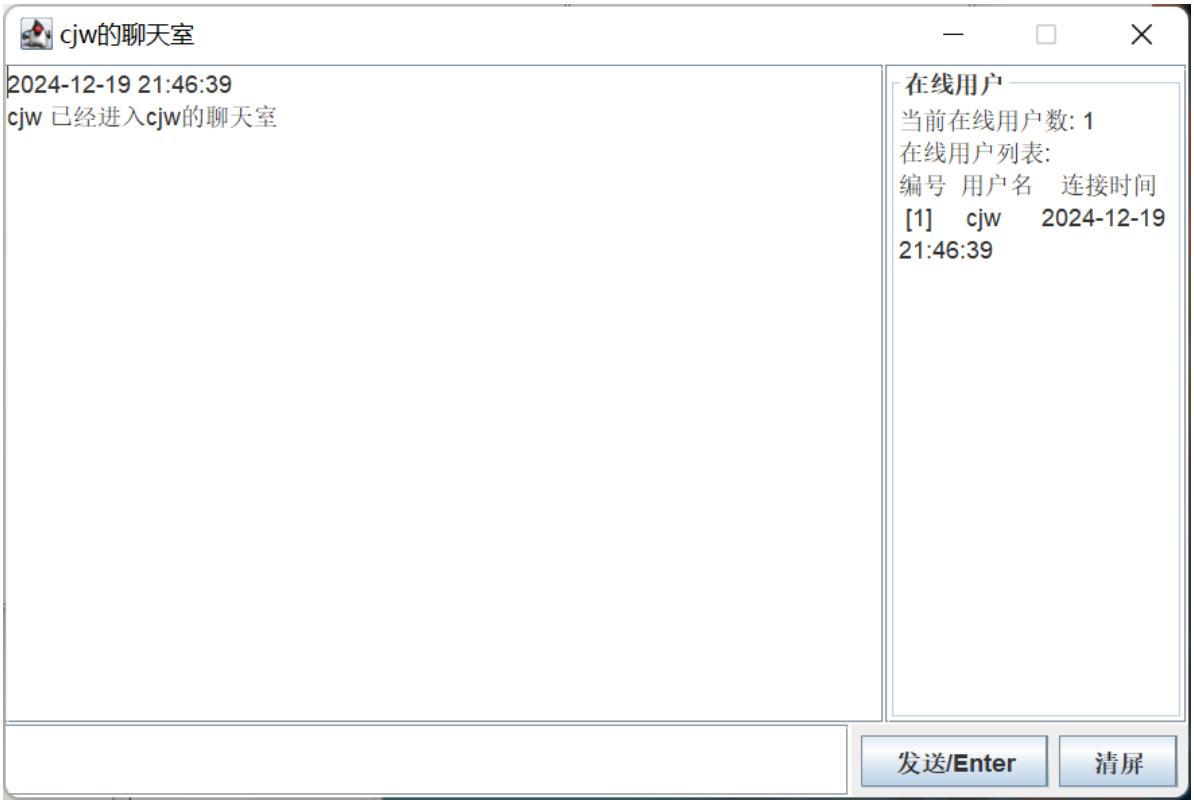
如果还没有打开服务器监听端口，会提示无法连接到服务器。

开始运行`Server_main`，这是初始页面，想要监听的端口可以在`Server_main.java`文件和`Server_GUI.java`文件中更改，这里是监听1235端口。

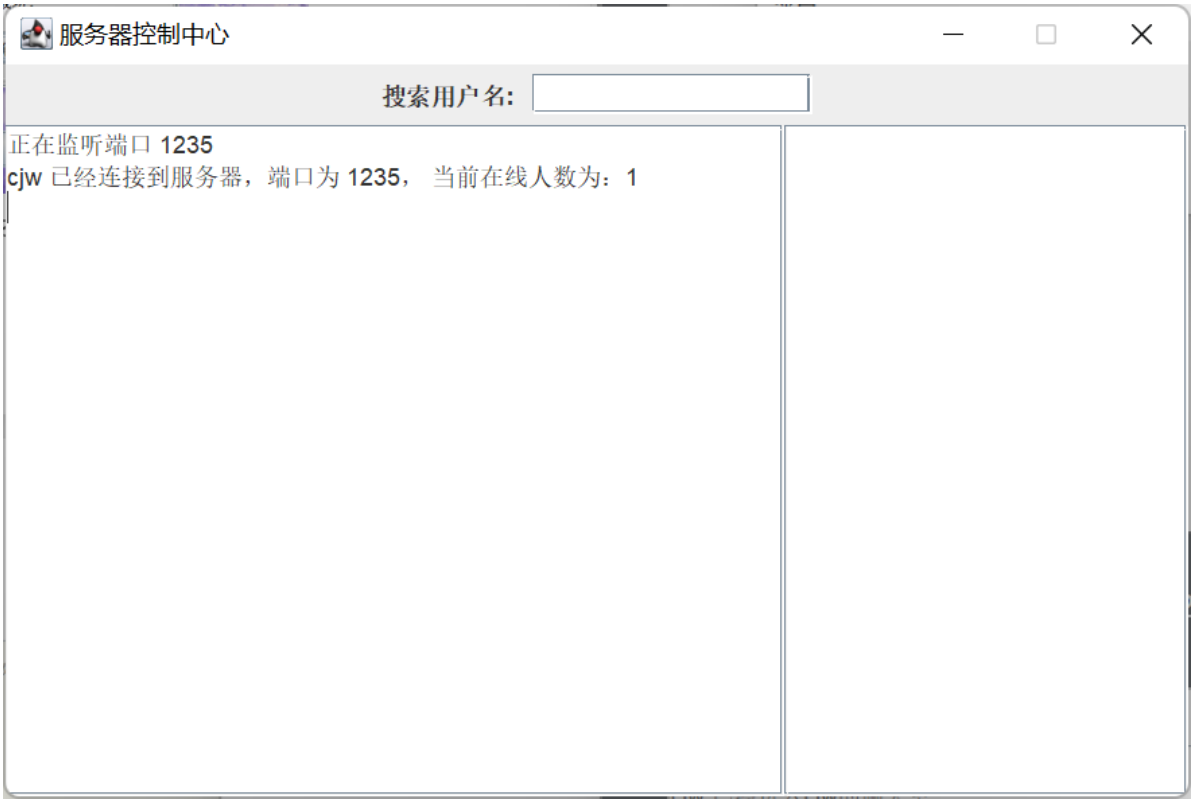


在打开服务器后，连接成功。

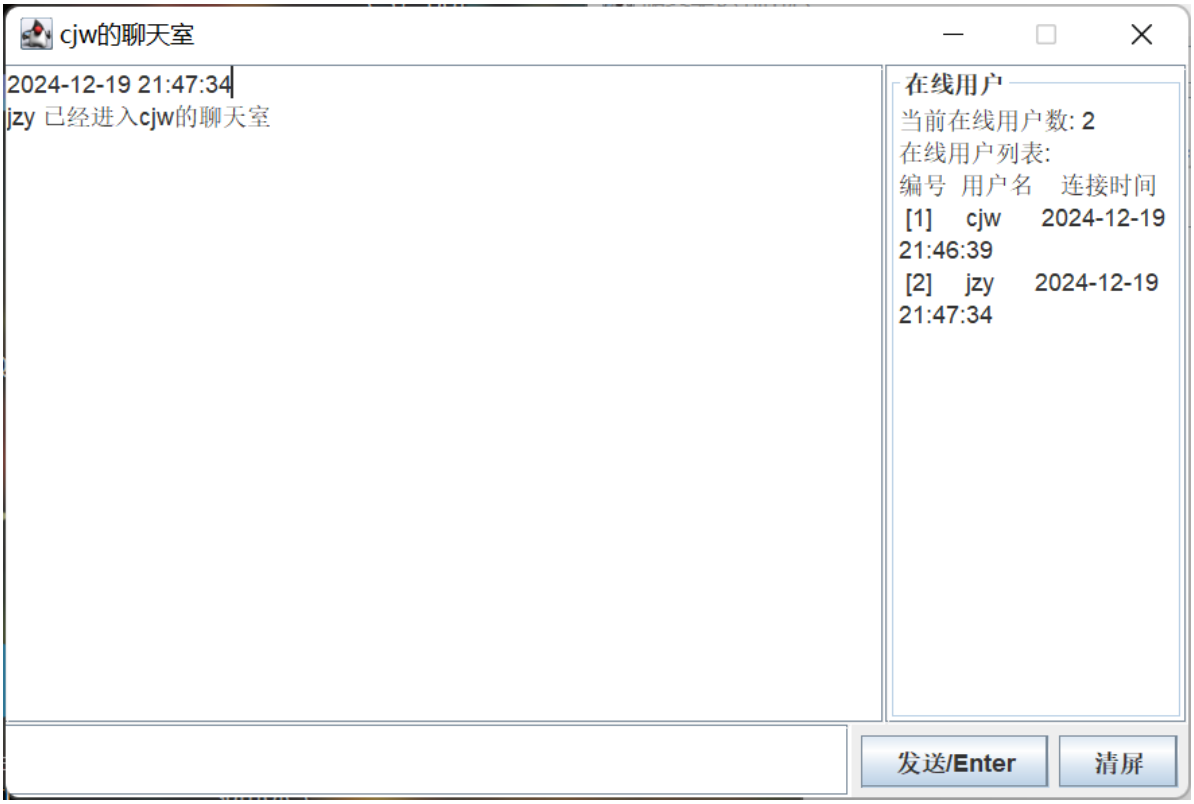
用户进入聊天室页面（cjw的聊天室是我写的默认提示字符，无论用户名是啥，标题"cjw的聊天室"和'已经进入cjw的聊天室'都不会改变）



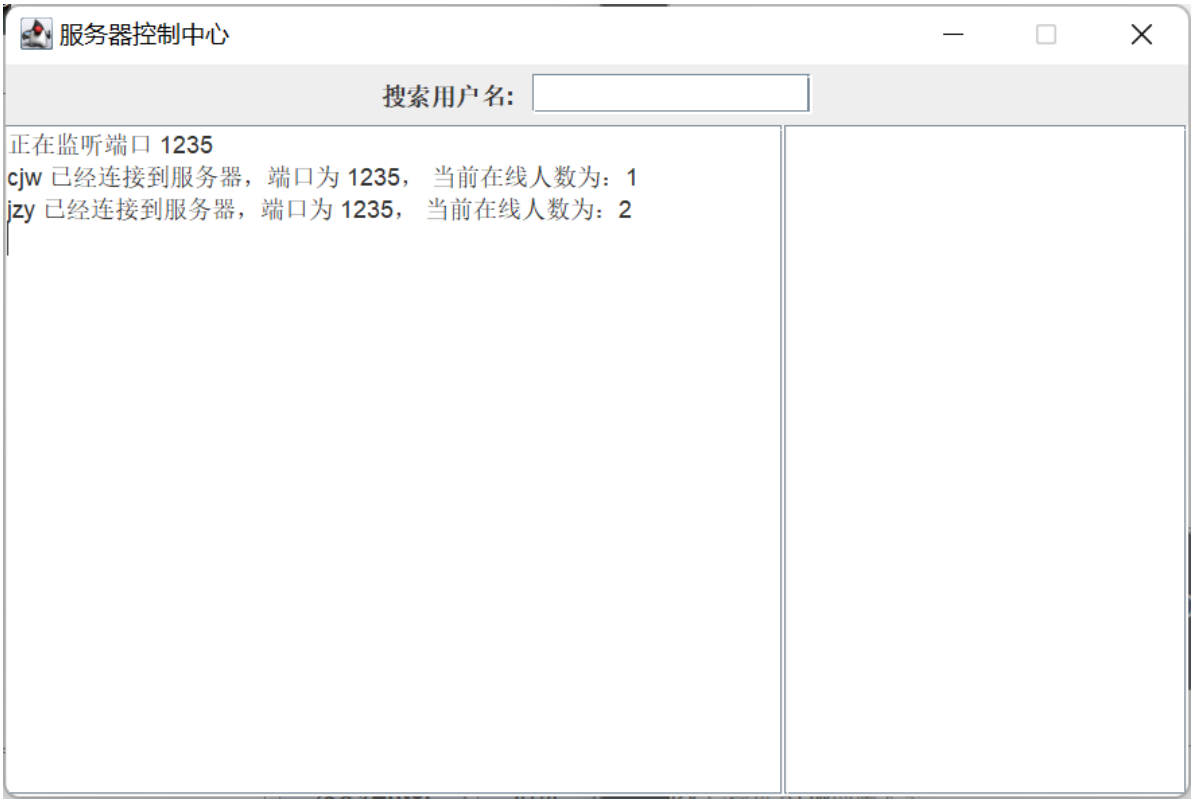
服务器界面显示cjw用户已经进入和在线人数



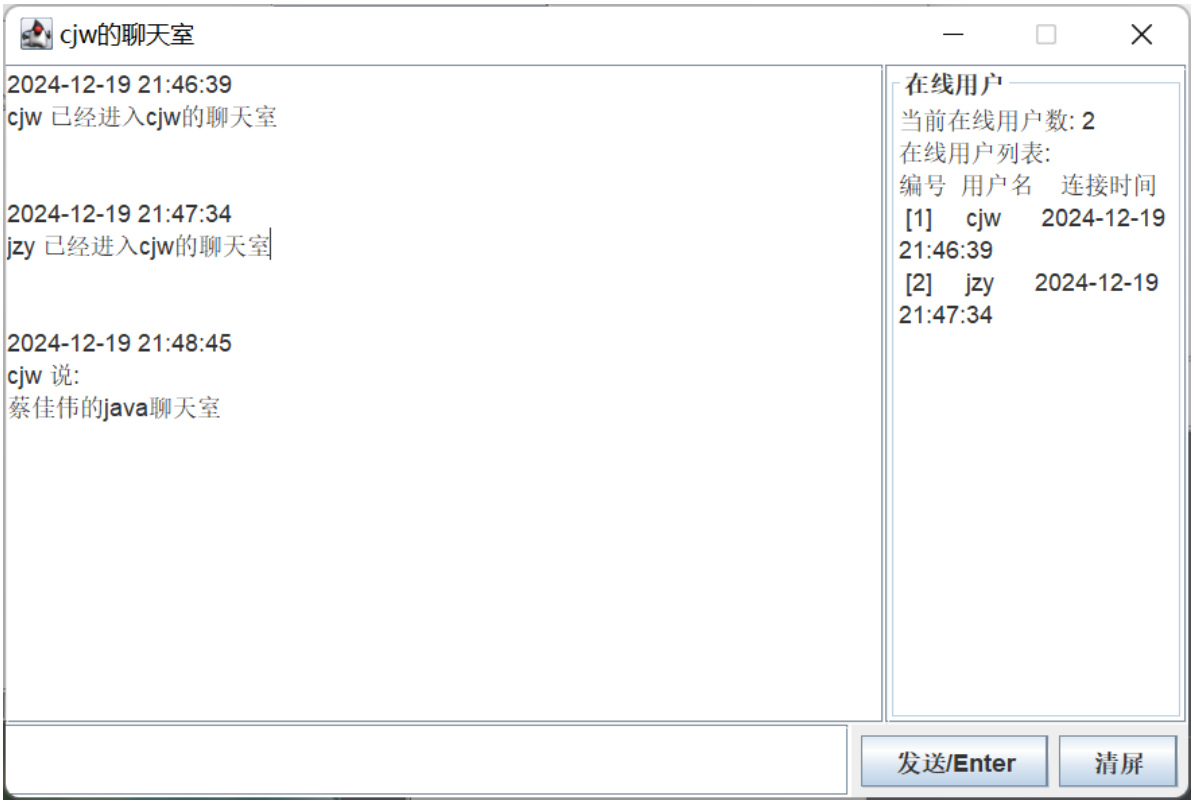
现在我让第二个用户jzy进入，这是jzy用户的Client界面。可以看到右边的用户界面显示的当前在线用户增加了。



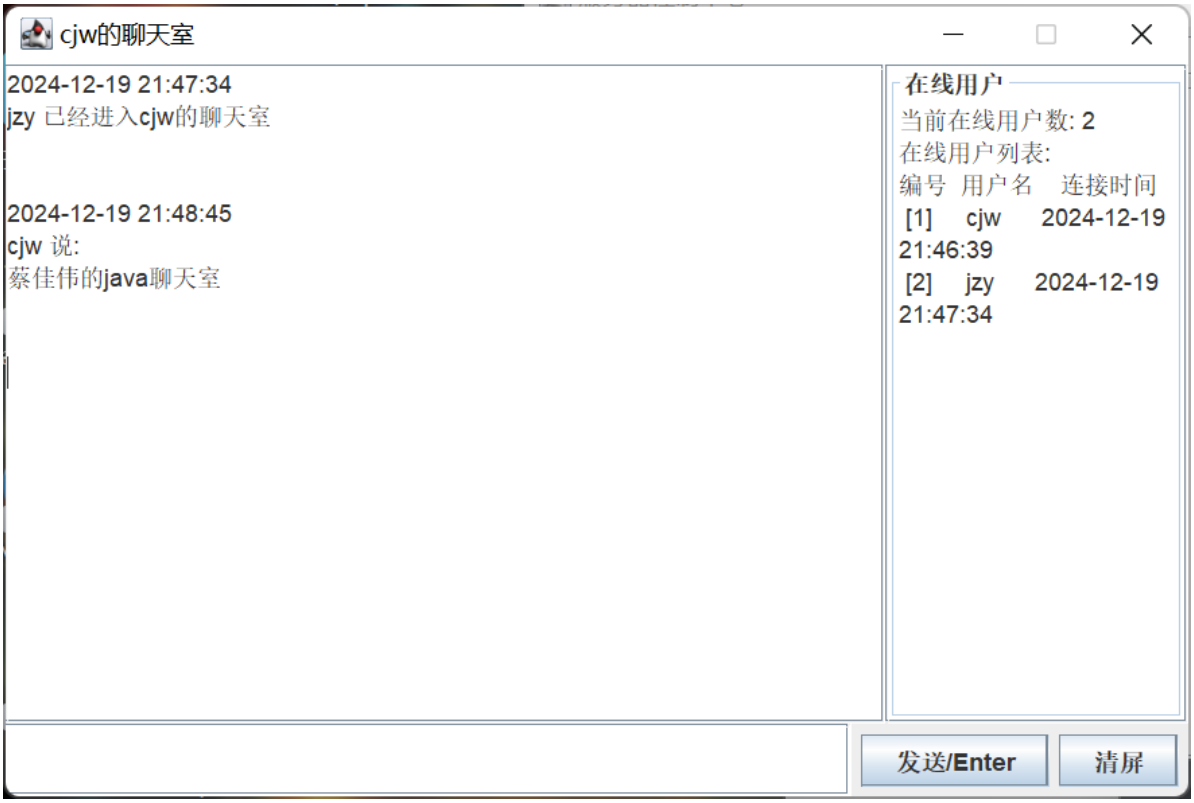
服务器也监听到jzy的接入。



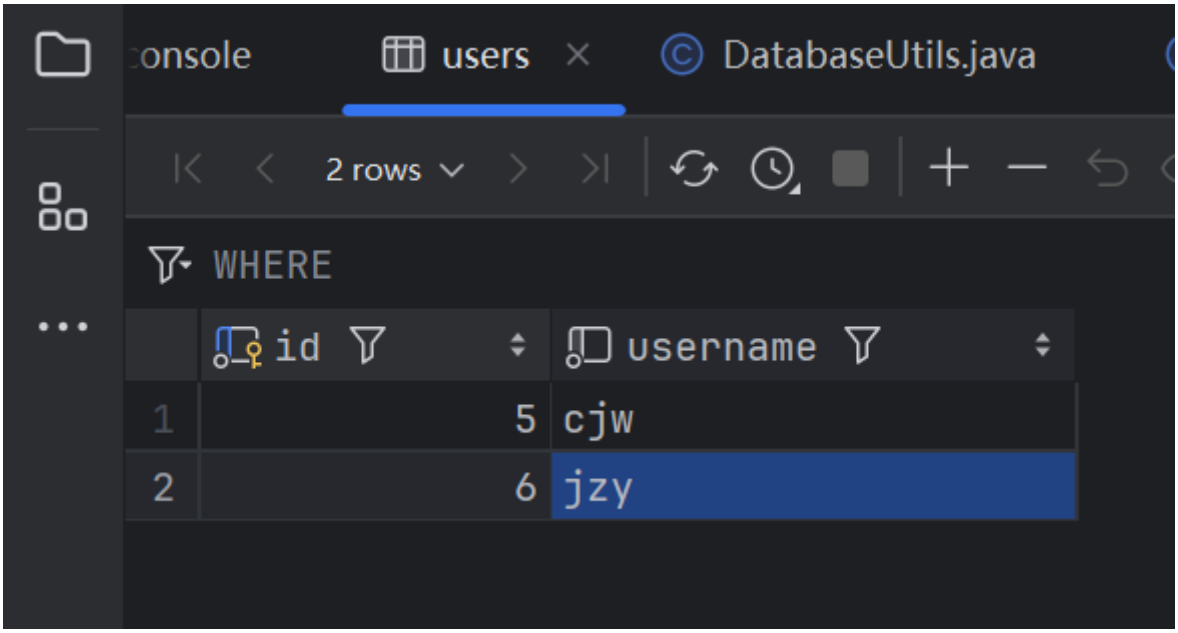
cjw用户发送一句话，这是cjw用户的界面。



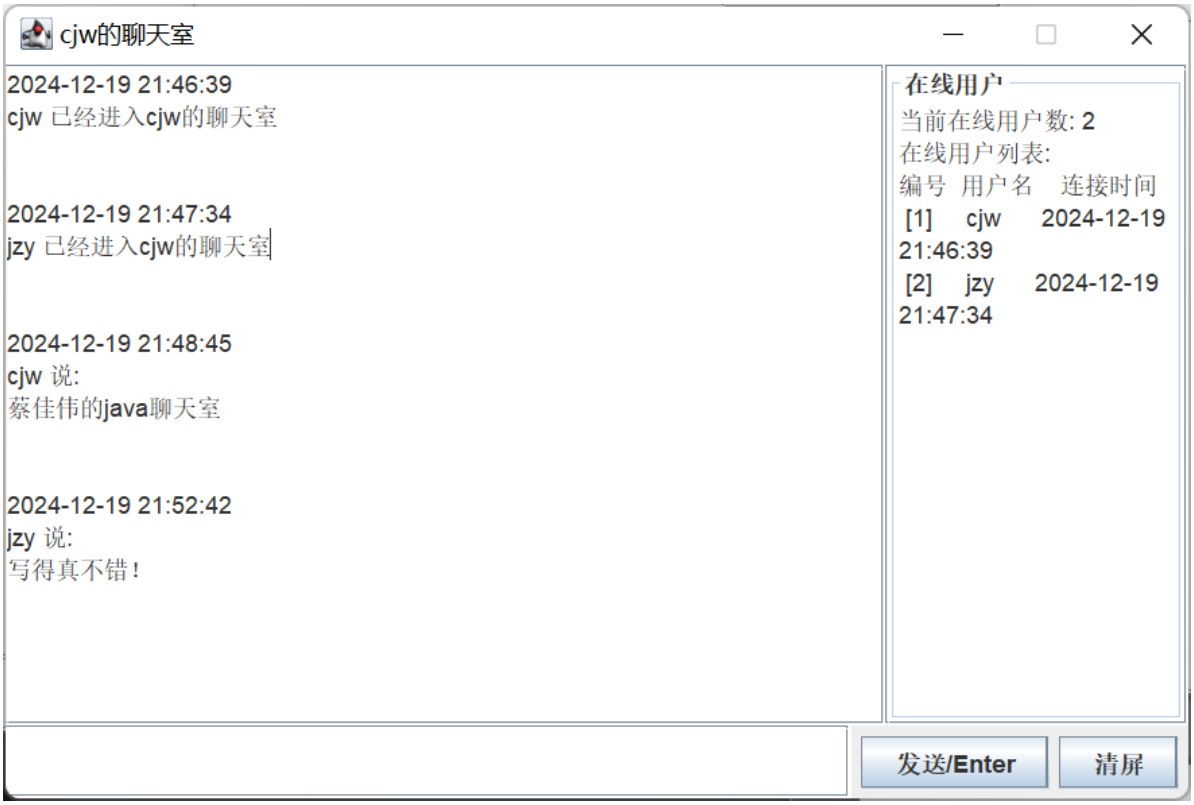
这是jzy用户的界面，同样正确显示了聊天。

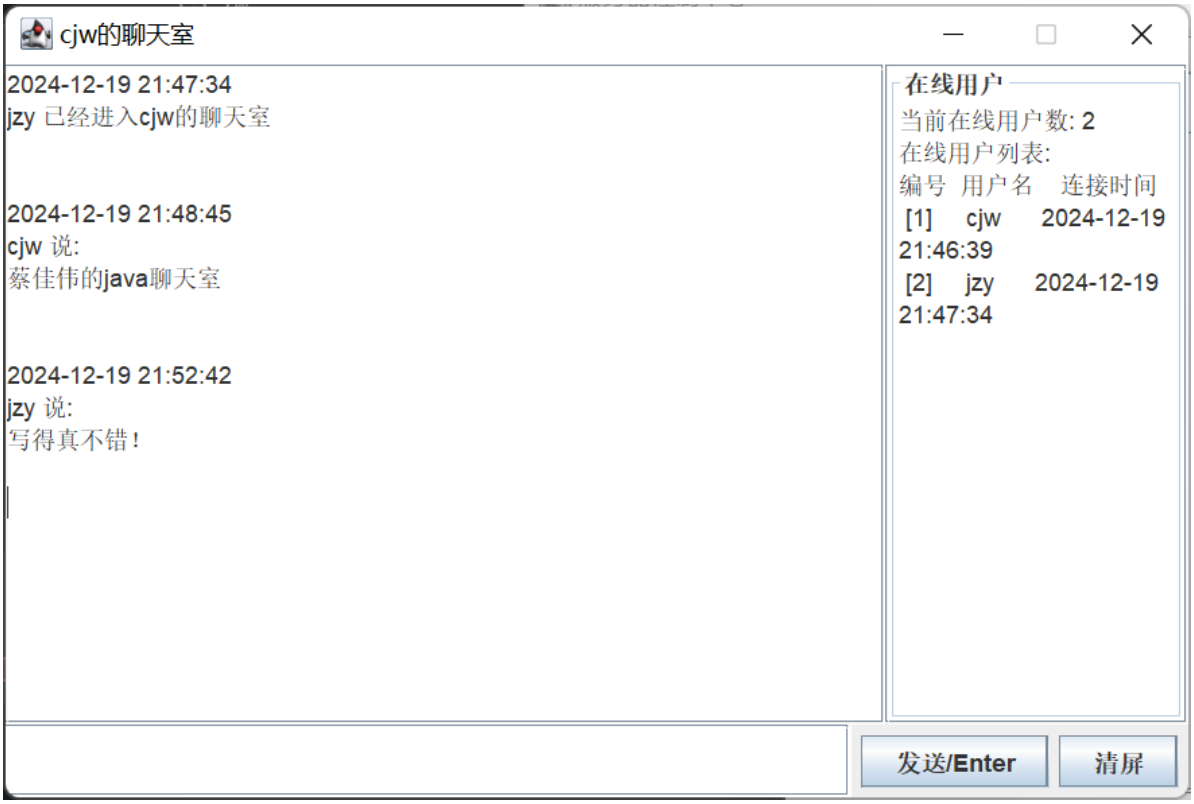


这是我的数据库，展示了用户名已经被添加入表格中。（id是自增属性，它一直有记录，所以不是从1开始的，不会影响使用）

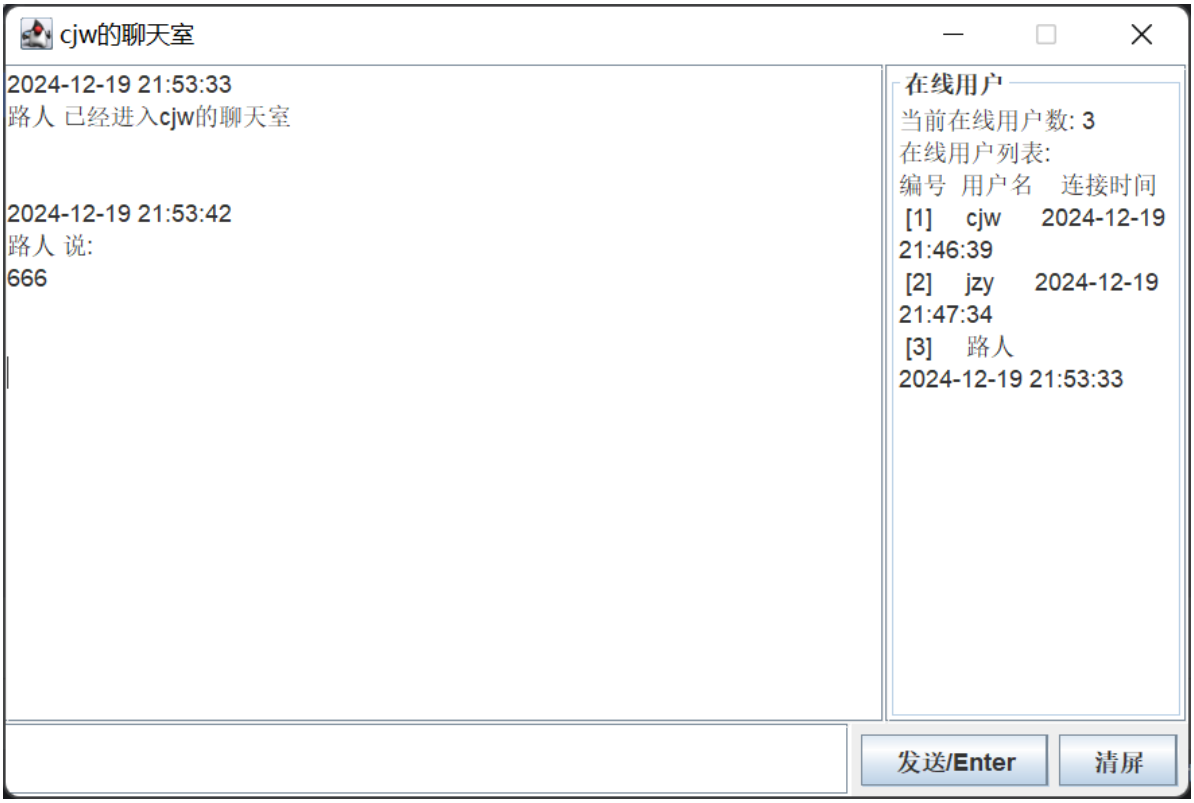


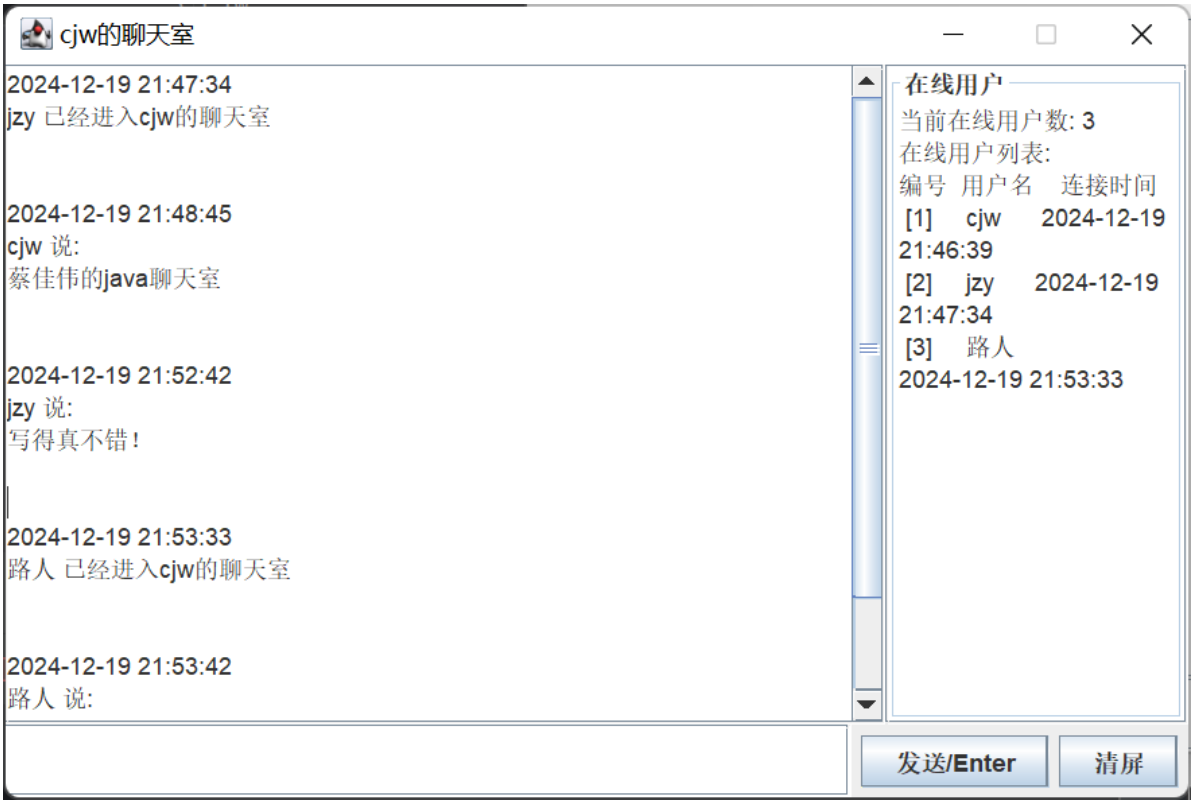
可以看到每个用户都可以在聊天室发言，所有用户都可以看到任何用户的发言。并且显示的发言时间也统一。



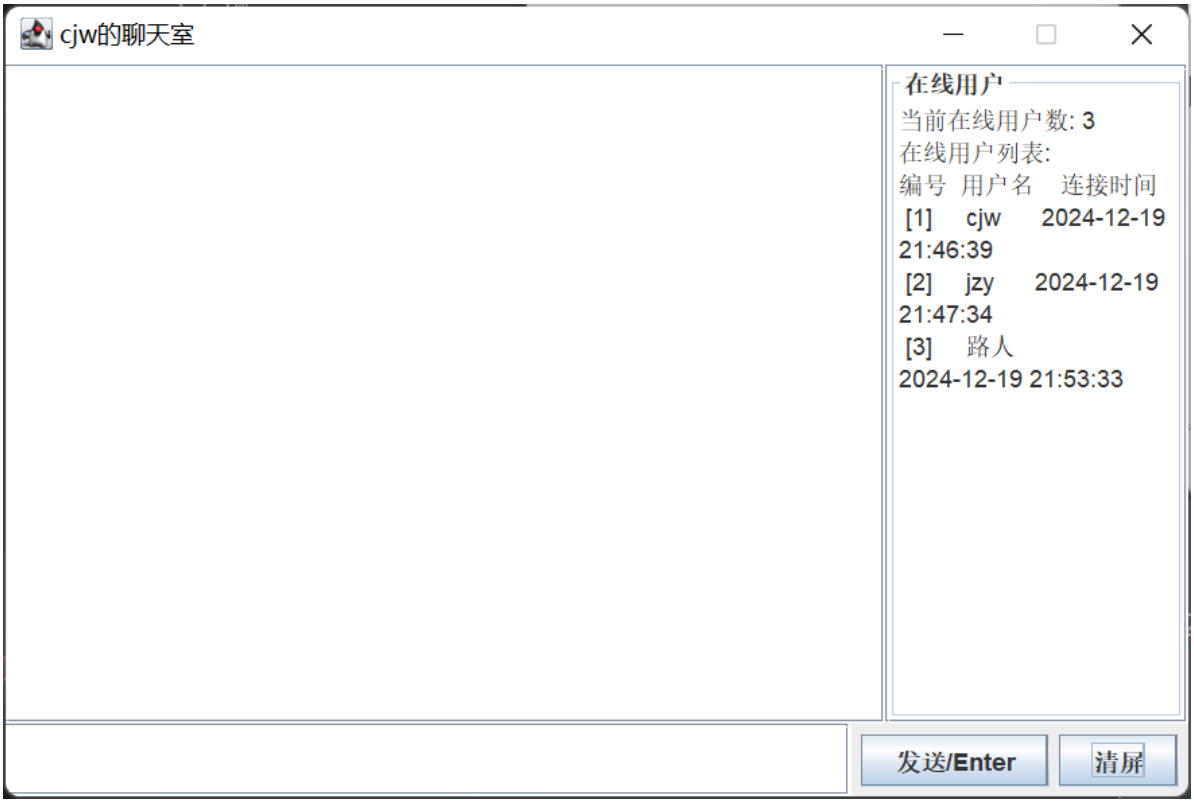


用户随时加入聊天室，看不到之前的消息，但是其他用户之前的消息不会被影响。



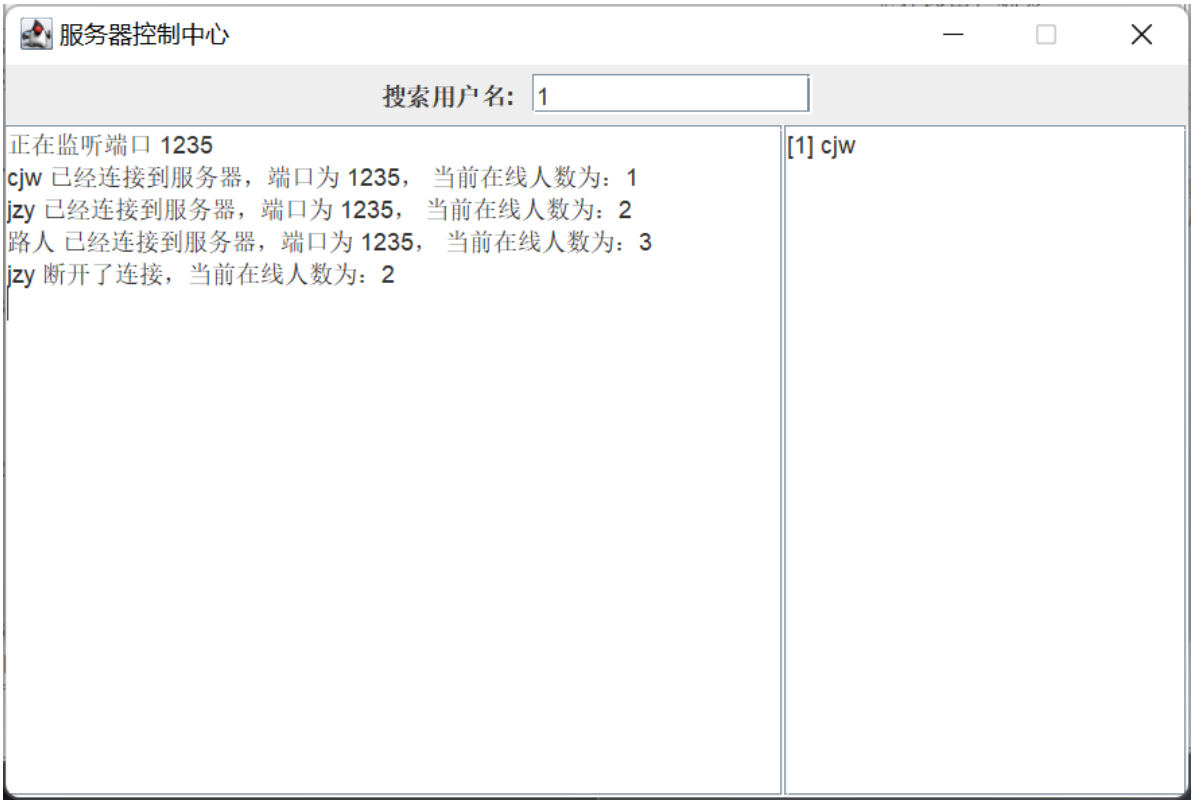


点击清屏按键，会把当前用户聊天界面所有聊天清空。

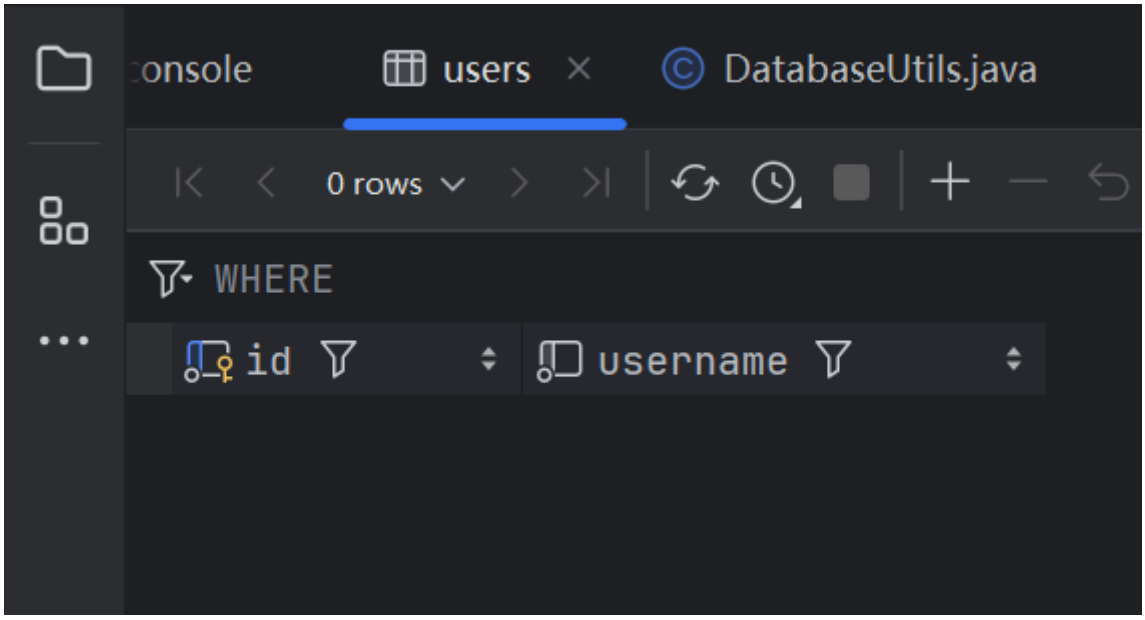


当用户断开连接后，服务器显示某用户断开连接和当前在线人数。

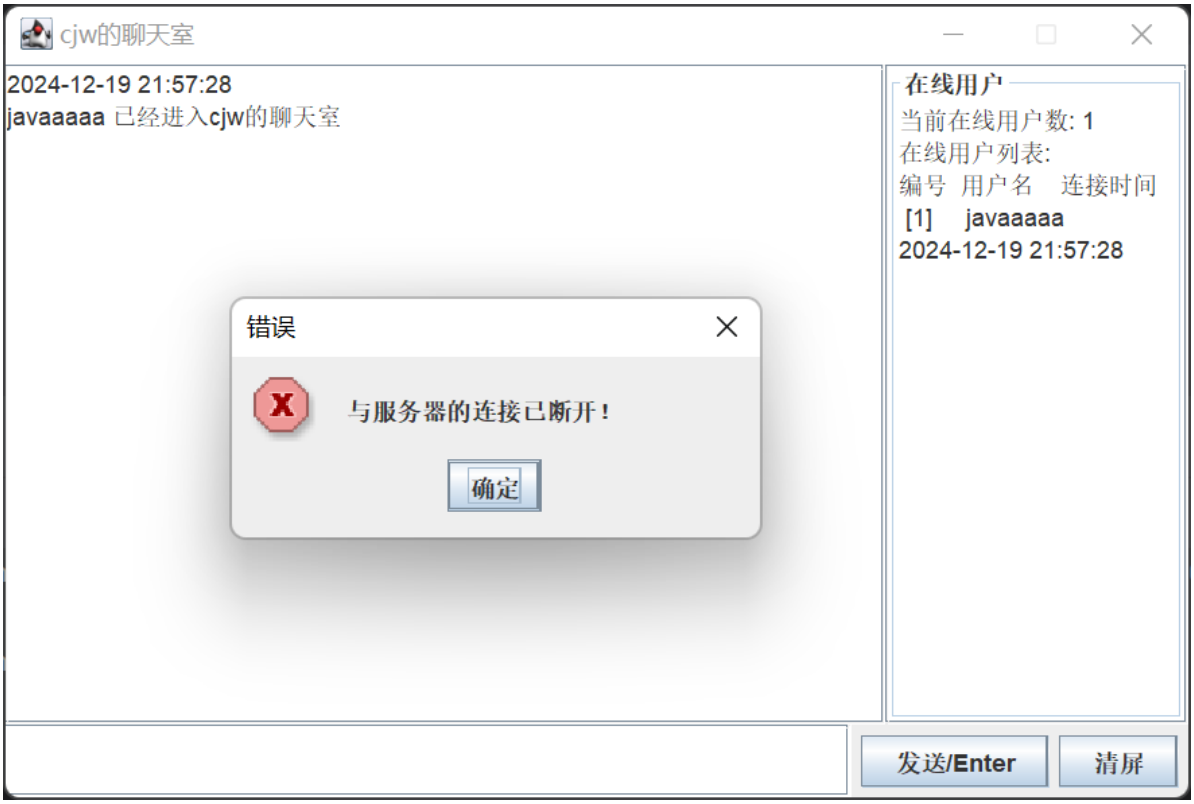
服务器界面还有搜索功能，搜索框内输入字符串，用户名或编号中含有对应的信息的用户名都会显示在右边的框内，这个功能主要方便用户很多的情况下查找关心的用户。



这是用户全部断开后，数据库截图，可见用户已全部删除，不影响聊天室下次使用。



如果先断开服务器，那么未关闭的Client会显示与服务器的连接已断开，然后点击确定后，会自动关闭Client。



通过以上的测试，可见聊天室功能均正确实现，且遇到的错误情况能正常处理，与数据库的交互也是符合预期的。

五、总结

本次的开发大概花费了我一周的时间，通过这样一个综合性的聊天室大项目设计与开发，我对于鲁老师课堂上学习到的Java的GUI编程、网络编程和多线程编程的理论知识有了进一步的理解。

本次编写中，遇到的最大问题是网络相关的。Java中有封装好的Socket接口，但我一开始对使用不了解，也不知道如何实现通信。在经过学习后，并结合一些其他项目的例子，我基本了解了如何使用Socket实现网络通信，端口监听等。

关于GUI部分的Swing框架，还是比较容易理解的，个人感觉似乎比较像Qt，在设计好功能后，前端界面设计上没有遇到太多困难。数据库的使用上，由于有上学期数据库系统课程使用Java完成图书管理系统的经验，也没有遇到太大困难。

在主要实现思路和变量设计完毕后，我还花费了大量的时间满足面向对象的设计要求，例如类的嵌套设计，类的继承，私有变量的保护，文件系统的设计，这些反而是花费更多时间的部分。

Debug中，主要遇到的问题还是不能运行或者系统运行不符合预期，第一个问题主要是文件配置错误造成，第二个问题主要是参数传递和函数调用的问题，我主要通过打断点和print中间变量来进行调试。

总而言之本次作业还是很有收获的，项目也成功实现了需求的功能。