# 浙江大学计算机科学与技术学院

Java 程序设计课程报告

2020—2021 学年　秋冬学期

| | |
|---|---|
| 题目 | 多人在线聊天软件设计与开发 |
| 学号 | 3180106148 |
| 学生姓名 | 李向 |
| 所在专业 | 计算机科学与技术 |
| 所在班级 | 计科 1802 |

# 目　　录

# 1 引言

我本次开发的是一个多人在线聊天软件，需要运用到 Java 语言中的 GUI 编程知识，网络编程知识，并发编程知识以及其他 Java 语言的有关知识。通过这样一个综合性的项目设计与开发，可以对课堂上学习到的理论知识进行进一步的理解吸收，更是可以在实践过程中提高自己使用 Java 语言解决实际问题的能力。

## 1．1 设计目的

本次课程设计的主要目的如下：

(1) 锻炼使用 Java 语言开发大型项目的能力；

(2) 熟悉 Java 语言中的 GUI 编程，熟悉 Swing 框架的使用；

(3) 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；

(4) 熟悉 Java 语言中的并发编程，通过并发编程技术实现聊天软件支持多组用户同时使用的功能。

## 1．2 设计说明

本程序采用 Java 程序设计语言，在 Eclipse 平台下编辑、编译与调试。具体程序分为客户端和服务器两部分，均由本人独立开发完成。

# 2 总体设计

## 2．1 功能模块设计

本程序分为客户端和服务器两部分分别进行实现，服务器要实现的主要功能有：

(1) 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；

(2) 接收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，主线程仍然要继续监听，以实现支持多组用户同时使用的目的；

(3) 维护一个当前已连接的客户端的列表，记录当前在线用户数；

(4) 转发一个客户端的消息到所有已连接的客户端；

客户端要实现的主要功能有：

(1) 通过指定 IP 地址和端口号，连接到指定的服务器；

(2) 发送消息给其他连接到同一服务器的客户端；

(3) 查看当前在线用户信息，包括在线用户数，用户名与上线时间等；

(4) 在其他客户端连接了服务器或者断开连接时会显示提示信息；

## 2. 2 流程图设计

程序总体流程分为客户端和服务器两个部分展示，客户端的流程设计如图 1 所示，服务器的流程设计如图 2 所示：

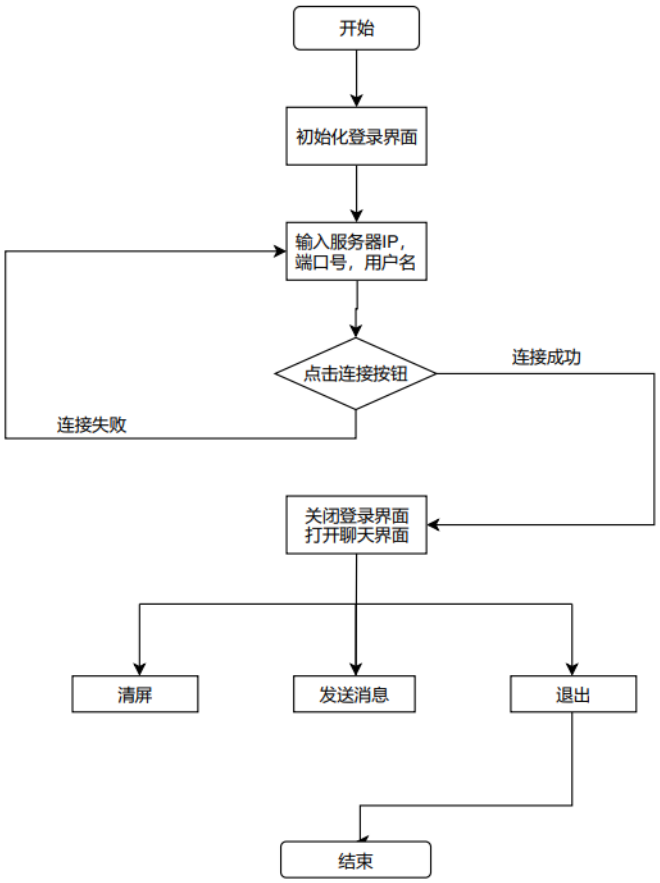


**图 1 客户端流程图**

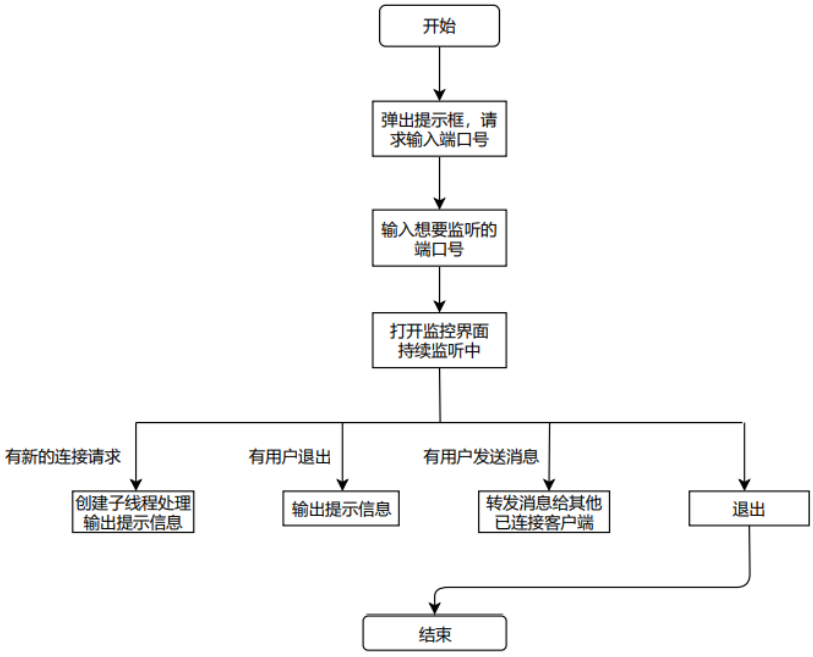


**图 2 服务器流程图**

# 3　详细设计

## 3．1　总体描述

系统的整体框架如图 3 所示，主要分为 Client_main，Client_Login_GUI，ChattingRoom_GUI，Client_socket，Server_main，Server_GUI，Server_socket 和 Message 这 8 个类，各个类的具体功能和实现将在接下来的部分一一介绍。

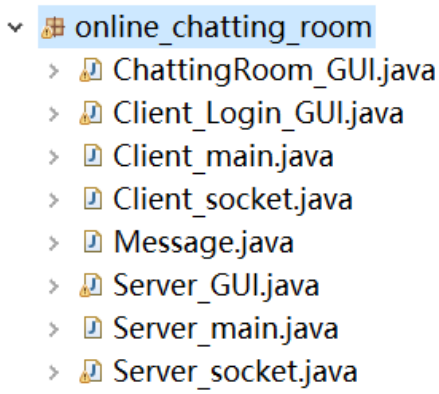


**图 3　系统整体框架图**

## 3．2　Client_Login_GUI 类的设计

Client_Login_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 接口来响应用户按下登录按钮的事件。整体元素在 JPanel 上的布局使用了自由布局的方式，通过对各个组件调用 setBounds()函数来自由地设置其位置。

Client_Login_GUI 类是 Swing 框架下的一个具体实现，设计流程较为固定，依次为：

整体窗口初始化 -> 创建各个组件 -> 组件内容初始化 -> 设置组件在窗口中的位置 -> 添加组件到窗口 -> 实现 ActionListener 接口来响应事件。

该类的整体结构如图 4 所示，布局展示如图 5 所示：

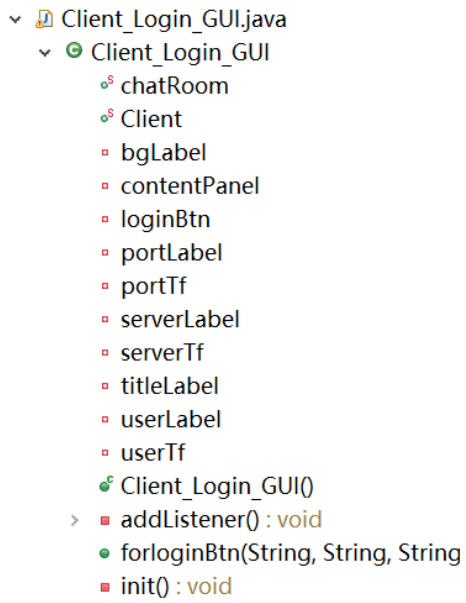


**图 4　Client_Login_GUI 类的整体结构**

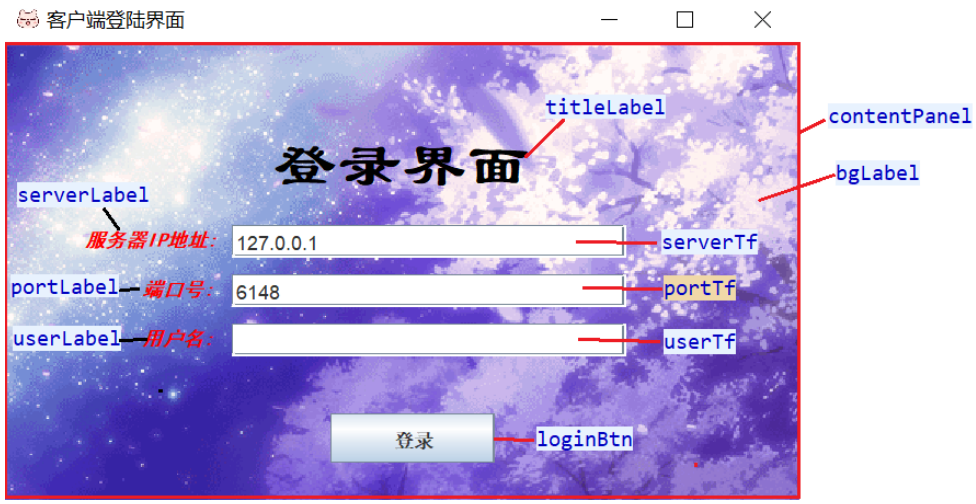


**图 5　Client_Login_GUI 类的布局展示**

其中按下登录按钮的响应函数，它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，并调用其 connectServer()方法来连接服务器，如果连接成功，则设置当前的登录界面为不可见，并设置在线聊天室界面为可见，这样就实现了界面的跳转。

## 3．3　ChattingRoom_GUI 类的设计

ChattingRoom_GUI 类的设计思路与 3.2 类似，该类的整体结构如图 6 所示，布局展示如图 7 所示：

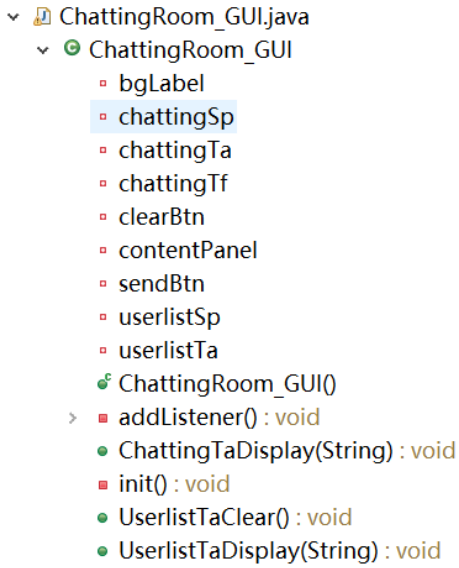


**图 6　ChattingRoom_GUI 类的整体结构**

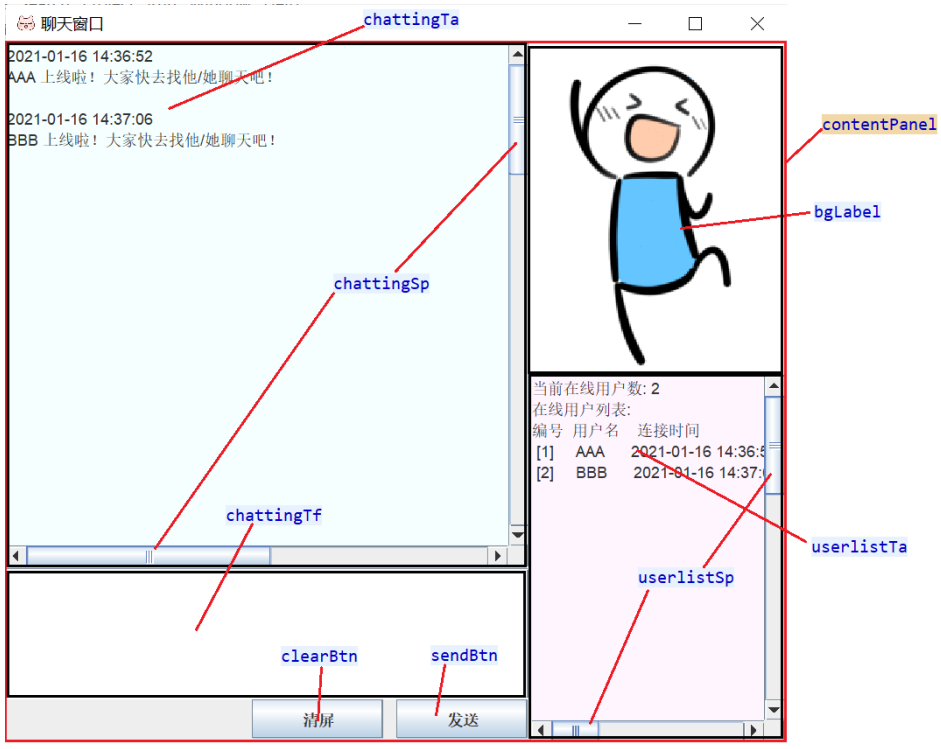


**图 7　ChattingRoom_GUI 类的布局展示**

其中处理按下发送按钮事件的响应函数的功能为：取出 chattingTf 中用户输入的要发送的消息，先判断其合法性，然后将其封装为一个 Message 类的实例，通过调用 Client_socket()类的 sendMessage()方法来将消息发送给服务器，最后通过调用 chattingTf.setText("")来清空 chattingTf 的内容。

处理按下清屏按钮事件的响应函数的功能为：调用 chattingTa.setText("")清空 chattingTa 的内容，也即清除对话框中已有的内容。

ChattingTaDisplay(String msg)方法的功能为将传入的字符串 msg 显示到 chattingTa，也即对话框中，通过调用 JTextArea 类的 append()方法实现。

UserlistTaDisplay(String msg)方法的功能为将传入的字符串 msg 显示到 userlistTa，也即在线用户列表框中，也是通过调用 JTextArea 类的 append()方法实现。

UserlistTaClear()方法的功能为清空 userlistTa 的内容，通过调用 userlistTa.setText("")即可实现。

### 3．4　Client_socket 类的设计

Client_socket 类主要处理与服务器的连接，需要实现与服务器连接有关的一些方法。主要运用到了 Java 中 Socket 编程的思想以及多线程编程的思想，该类的整体结构如图 8 所示：

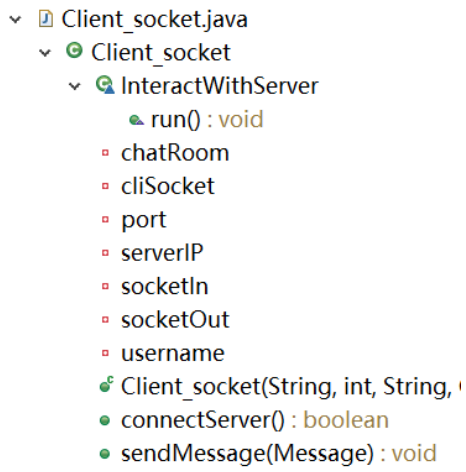


**图 8　Client_socket 类的整体结构**

Client_socket 类主要成员变量及其含义如下图所示：

```
private Socket cliSocket;                // Client socket
private ObjectInputStream socketIn;      // InputStream of socket
private ObjectOutputStream socketOut;    // OutputStream of socket
private String serverIP;                 // IP address of the server
private int port;                        // Port of the server
private String username;                 // Username of the client
private ChattingRoom_GUI chatRoom;       // existing GUI of chatRoom
```

**图 9　Client_socket 类主要成员变量**

Client_socket 类的构造函数较为简单，就是将传入的参数赋值给对应的成员变量。

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

(1) 实例化一个 Socket 对象，指定服务器名称和端口号来请求连接；

(2) 如果连接成功，接着创建套接字的输入输出流。

(3) 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。

对于这些过程中出现的 exception 也需要进行妥善处理，相关的具体代码如下图所示：

```
/* Function to connect to server */
public boolean connectServer() {
    try {
        // Create a socket
        cliSocket = new Socket(serverIP,port);
    }catch (Exception e) {
        JOptionPane.showMessageDialog(null, "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", "Error",JOptionPane.ERROR_M
        return false;
    }
    JOptionPane.showMessageDialog(null, "连接服务器成功！");
    String errorMsg;

    // Initialize two stream
    try {
        socketIn = new ObjectInputStream(cliSocket.getInputStream());
        socketOut = new ObjectOutputStream(cliSocket.getOutputStream());
    }catch (Exception e) {
        errorMsg = "创建套接字的I/O流时出错！ " + e;
        JOptionPane.showMessageDialog(null, errorMsg, "Error",JOptionPane.ERROR_MESSAGE);
        return false;
    }

    // Try to send message to server
    try {
        socketOut.writeObject(username);
    }catch (IOException e) {
        errorMsg = "向服务器发送消息时出错！ " + e;
        JOptionPane.showMessageDialog(null, errorMsg, "Error",JOptionPane.ERROR_MESSAGE);
        return false;
    }

    // Create a thread to listen the message from Server
    new InteractWithServer().start();
    return true;
}
```

**图 10　Client_socket 类的 connectServer()方法**

可以看到在 Client_socket 类中，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，然后取出其实际消息内容和类型，如果类型为 MESSAGE，则在聊天框中显示该消息；如果类型为 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捉到了异常，则说明与服务器的连接已经断开，弹出提示对话框并关闭聊天界面。具体实现如下图所示：

```
// Thread: Print message sent by server
class InteractWithServer extends Thread{
    @Override
    public void run() {
        while(true) {
            try {
                // Receive message
                Message servermsg = (Message) socketIn.readObject();
                String content = servermsg.getContent();
                switch(servermsg.getType()) {
                // Type of message is MESSAGE, then display it on ChattingTa
                case Message.MESSAGE:
                    chatRoom.ChattingTaDisplay(content);
                    break;
                 // Type of message is MESSAGE, then display it on UserlistTa
                case Message.USERLIST:
                    chatRoom.UserlistTaClear();
                    chatRoom.UserlistTaDisplay(content);
                    break;
            }

            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "与服务器的连接已断开！");
                chatRoom.setVisible(false);
                break;
            }
        }
    }
}
```

**图 11　InteractWithServer 类中 run()方法的实现**

最后是 Client_socket 类的 sendMessage(Message msg)方法，该方法的主要作用是给服务器发送消息，通过向套接字输出流中写入 Message 类的对象 msg 即可实现。

### 3．5　Message 类的设计

Message 类定义了在客户端和服务器之间传递的消息的一个规范格式，它需要实现 Serializable 接口，这样才能够被 ObjectOutputStream 转换为字节流，同时也可以通过 ObjectInputStream 再将其解析为对象。Message 类中定义了两种消息的类型，分别为 MESSAGE 和 USERLIST，用来标识该消息是正常消息还是服务器发来的维护在线用户列表的消息。Message 类有两个成员变量，分别为 String 类的 content 和 int 型的 type，分别保存消息的实际内容和类型。两个方法 getType()和 getContent()分别返回这两个成员变量的内容。

### 3．6　Server_GUI 类的设计

Server_GUI 类的设计思路仍然与 3.2，3.3 类似，该类的整体结构如图 12 所示，布局展示如图 13 所示：

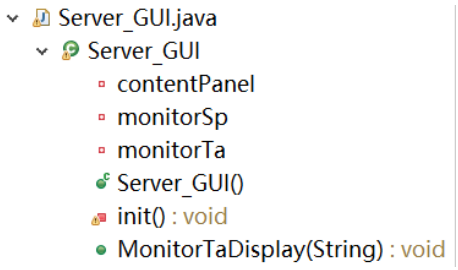


**图 12　Server_GUI 类的整体结构**

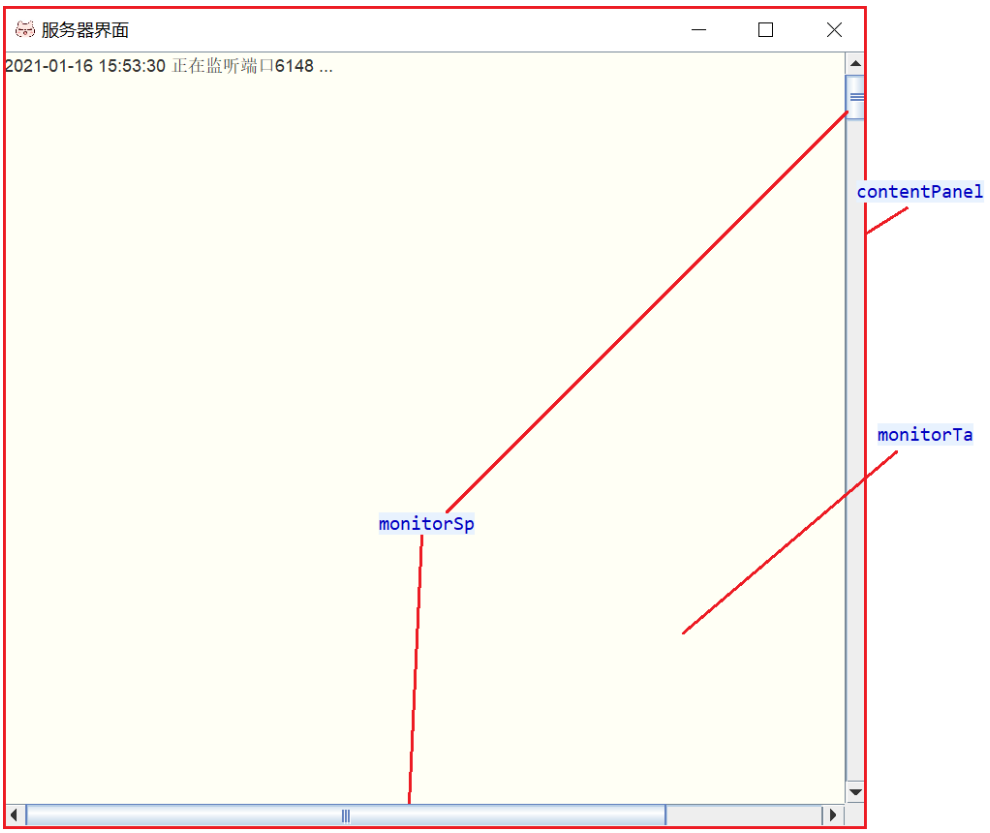


**图 13　Server_GUI 类的布局展示**

Server_GUI 类除了显示服务器界面，在最开始还通过一个可输入的对话框，获取了要监听的端口号。在最后创建了一个 Server_socket 类的对象，并调用其 listen()方法来监听指定的端口。

## 3．7　Server_socket 类的设计

Server_socket 类主要处理与客户端的连接，需要实现与客户端连接有关的一些方法。主要运用到了 Java 中 Socket 编程的思想以及多线程编程的思想，该类的整体结构如图 14 所示：

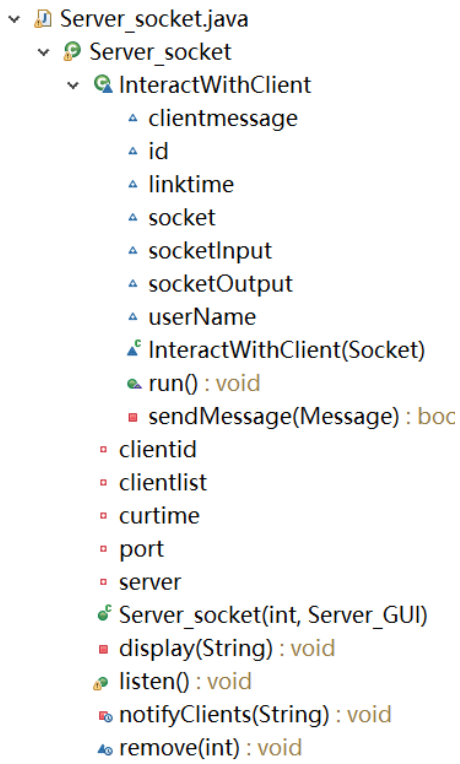


**图 14　Server_socket 类的整体结构**

Server_socket 类主要成员变量及其含义如下图所示：

```
private int port;                                    // Port to listen
private ArrayList<InteractWithClient> clientlist;    // ArrayList of all connected client
private int clientid;                                // Unique id for each client
private SimpleDateFormat curtime;                    // current time
private Server_GUI server;                           // The instance of Server_GUI
```

**图 15　Server_socket 类主要成员变量**

Server_socket 类的构造函数较为简单，首先将传入的参数赋值给对应的成员变量，然后创建了元素为子线程的 ArrayList 以及一个 SimpleDateFormat 的对象，并赋给了对应的成员变量。

Server_socket 类的 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。其主要流程为：

(1) 根据指定的端口号实例化一个 serverSocket 对象；

(2) 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给定的端口；

(3) 如果通信建立成功，accept()方法返回服务器上一个新的 socket 引用，该 socket 连接到客户端的 socket；

(4) 创建一个子线程处理后续与该客户端的通信；

(5) 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

(6) 发送提示消息给其他已连接的客户端，在服务端也输出相关提示信息，然后循环，继续调用 accept()监听端口。

对于这些过程中出现的 exception 也需要进行妥善处理，相关的具体代码如下图所示：

```
/* Function to listen and wait connect */
public void listen() {
    try
    {
        ServerSocket serverSocket = new ServerSocket(port);
        display("正在监听端口" + port + " ...\n");
        // Keep listening
        while(true)
        {
            Socket socket = serverSocket.accept();
            InteractWithClient tempclient = new InteractWithClient(socket);
            // Add connected client to clientlist
            clientlist.add(tempclient);
            tempclient.start();
            notifyClients(tempclient.userName+" 上线啦！大家快去找他/她聊天吧！");
            display(tempclient.userName+" 连接到服务器，当前在线人数为：" + clientlist.size() + "\n");
        }
    }
    catch (IOException e) {
        String msg = curtime.format(new Date()) + " 创建ServerSocket过程出错！： " + e + "\n";
        display(msg);
    }
}
```

**图 16　Server_socket 类的 listen()方法**

可以看到在 Server_socket 类中，我们还实现了一个 InteractWithClient 类，这个类继承自 Thread 类，需要实现其 run()方法。InteractWithClient 类的实例也即与客户端通信的子线程。该类的 run()方法中定义了该如何处理从客户端收到的消息：首先通过套接字输入流读取一个 Message 对象，然后直接调用 notifyClients()函数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了异常，则说明与指定客户端的连接已经断开，需要将与该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所有连接到服务器的客户端，并在服务器显示，具体实现如下图所示：

```
public void run() {
    // Loop until disconnect
    while(true) {
        // Read message from client
        try {
            clientmessage = (Message) socketInput.readObject();
        }
        catch (IOException e1) {
            break;
        }
        catch(ClassNotFoundException e2) {
            break;
        }
        notifyClients(userName + " 说: \n" + clientmessage.getContent());
    }
    // If client is shutdown, then remove it from the clientlist
    remove(id);
    notifyClients(userName+" 下线了，下次再找他/她聊天吧~\n");
    display(userName+" 断开了连接，当前在线人数为：" + clientlist.size() + "\n");
}
```

**图 17　InteractWithClient 类中 run()方法的实现**

InteractWithClient 类中还实现了 sendMessage(Message msg)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 即可实现。

最后再来看看 Server_socket 类中 notifyClients(String message)方法的实现，该方法的主要作用是给当前所有连接到服务器的客户端发送消息。其实现过程为：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，一一调用它们的 sendMessage()方法来将消息发送给所有客户端。除此之外还需要组装一个类型为 USERLIST 的 Message 对象，这个对象的 content 包含了当前在线用户的相关信息，也需要用同样的方法转发给当前每一个连接到服务器的客户端。其具体实现如下图所示：

```
/* Function to notify all the connected clients */
private synchronized void notifyClients(String message) {

    String notice = curtime.format(new Date()) + "\n" + message + "\n\n";
    Message chatmsg = new Message(Message.MESSAGE, notice);
    /* Construct the chatmessage and notify this to all clients*/
    for(int i = 0; i < clientlist.size(); ++i) {
        InteractWithClient nowclient = clientlist.get(i);
        nowclient.sendMessage(chatmsg);
    }
    /* Construct the message of online client list and notify this to all clients*/
    String tempstr = "当前在线用户数: " + clientlist.size() + "\n";
    tempstr += "在线用户列表:\n";
    tempstr += "编号   用户名     连接时间\n";
    for(int i = 0; i < clientlist.size(); ++i) {
        InteractWithClient nowclient = clientlist.get(i);
        tempstr += " ["+(i+1) + "]     " + nowclient.userName + "      " + nowclient.linktime +  "\n";
    }
    Message listmsg = new Message(Message.USERLIST, tempstr);

    for(int i = 0; i < clientlist.size(); ++i) {
        InteractWithClient nowclient = clientlist.get(i);
        nowclient.sendMessage(listmsg);
    }
}
```

**图 18　Server_socket 类中 notifyClients()方法的实现**

## 3．8　Client_main 类和 Server_main 类的设计

Client_main 类和 Server_main 类都只包含一个主方法 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，而 Server_main 类实例化了一个 Server_GUI()类的对象来显示服务器界面

# 4 测试与运行

## 4．1 程序测试

该部分主要测试了程序中的一些错误或者异常处理模块的功能是否正确。

测试是否可以判断服务器监听端口的合法性：

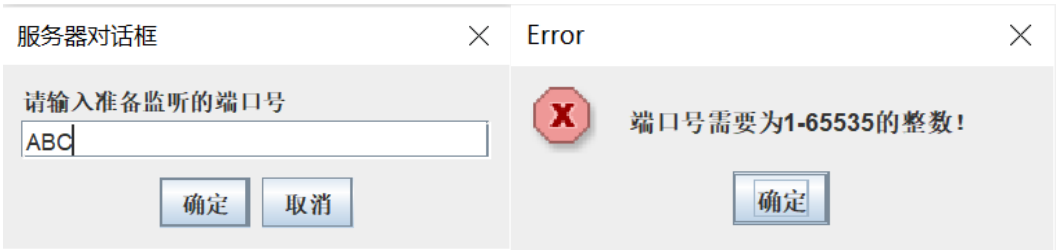


**图 19 服务器监听端口的合法性测试结果**

测试是否可以判断客户端登录界面输入 IP，端口号，用户名的合法性：



**图 20　客户端登录界面合法性测试结果**

测试连接不上服务器时的结果：


**图 21　客户端连接不上服务器时的测试结果**

测试发送消息为空时的结果：

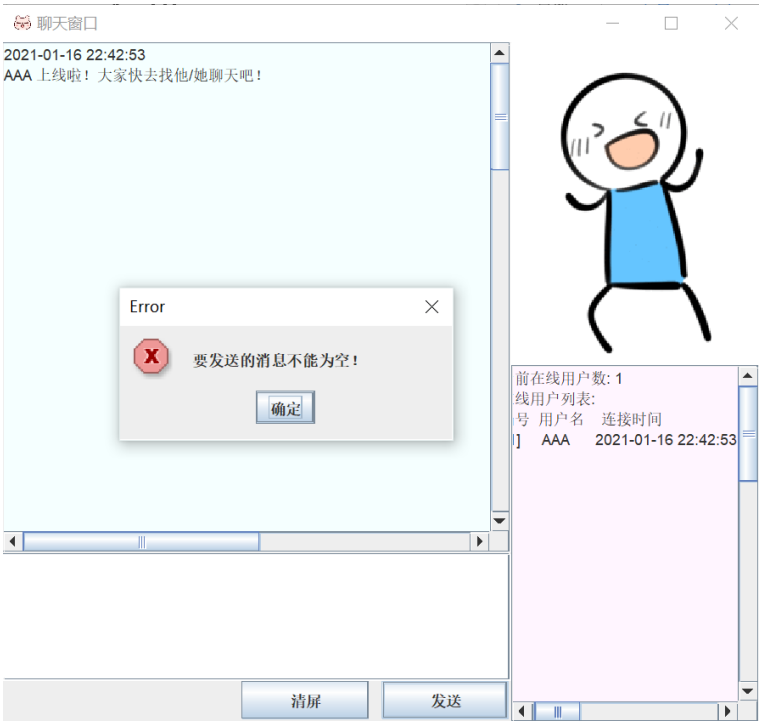


**图 22　聊天界面发送空消息测试结果**

## 4．2　程序运行

完整执行一遍程序的功能，首先打开服务器程序，输入监听端口 6148，之后进入服务器界面，接着打开一个客户端程序，输入服务器 IP 地址为 127.0.0.1（回环测试地址，表示本主机的 IP），端口号为 6148，用户名为 AAA，点击登录按钮，弹出对话框提示登录成功，进入聊天界面。 截图如下所示：


**图 23　弹出对话框提示连接服务器成功**

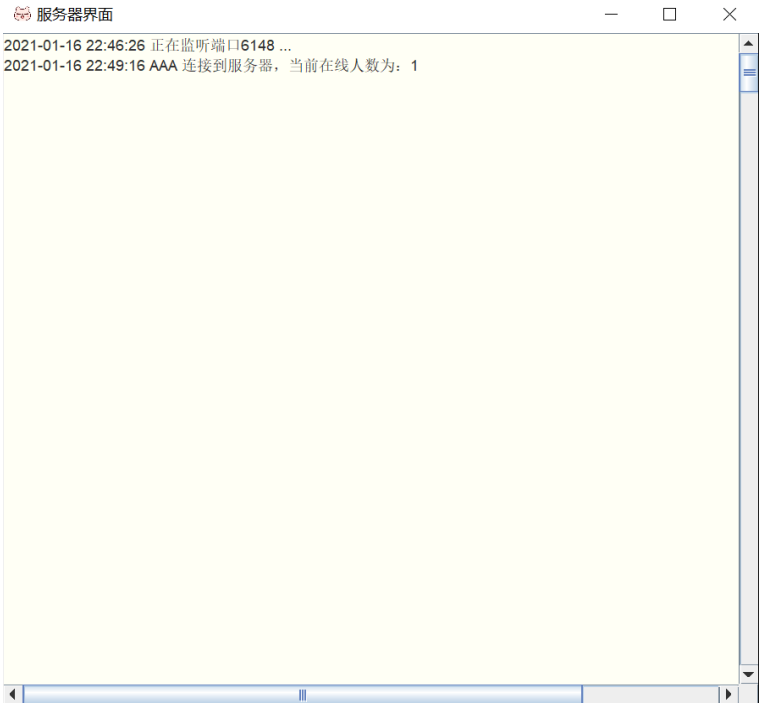


**图 24　服务器界面**

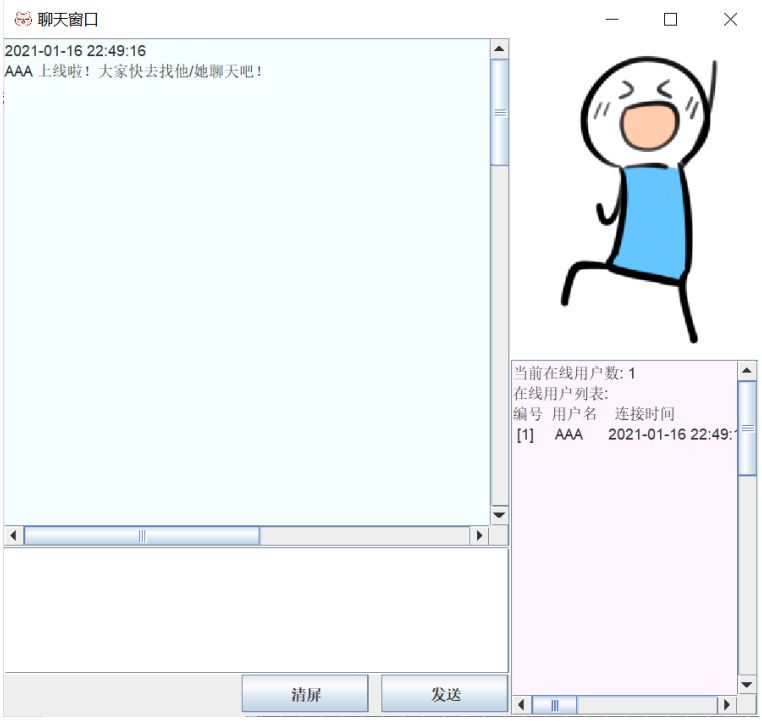


**图 25　聊天窗口界面**

接着再打开两个客户端程序，按同样的方式连接到服务器，用户名分别为 BBB 和 CCC，观察各个界面的显示结果：

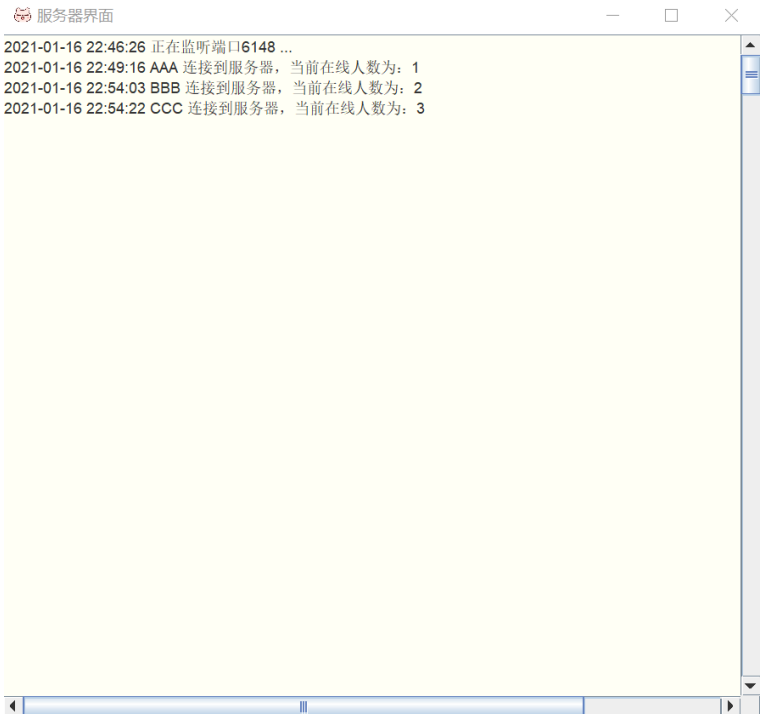


**图 25　服务器界面显示结果**

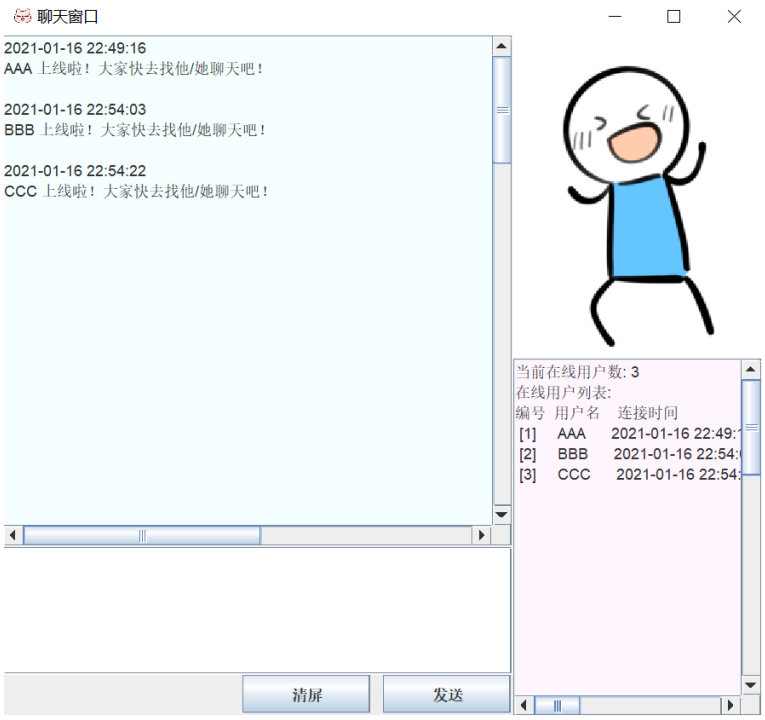


**图 26　AAA 聊天界面显示结果**

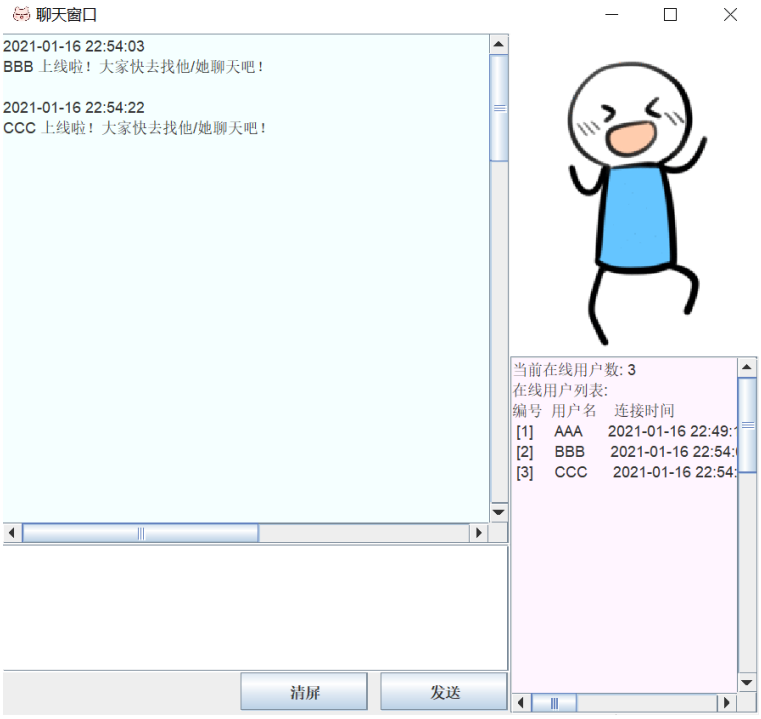


**图 27　BBB 聊天界面显示结果**

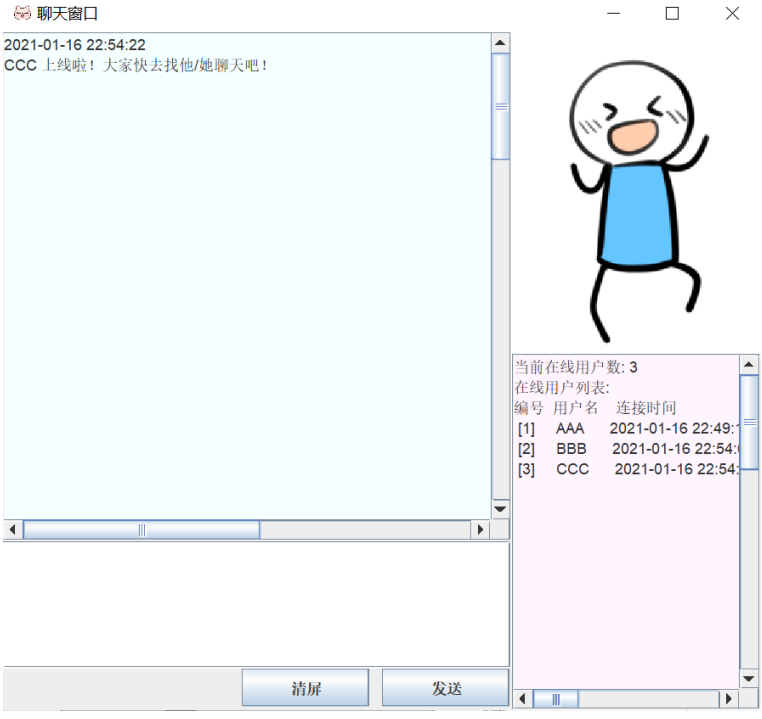


**图 28　CCC 聊天界面显示结果**

分别使用三个客户端发送一条消息，观察各个界面的显示结果：

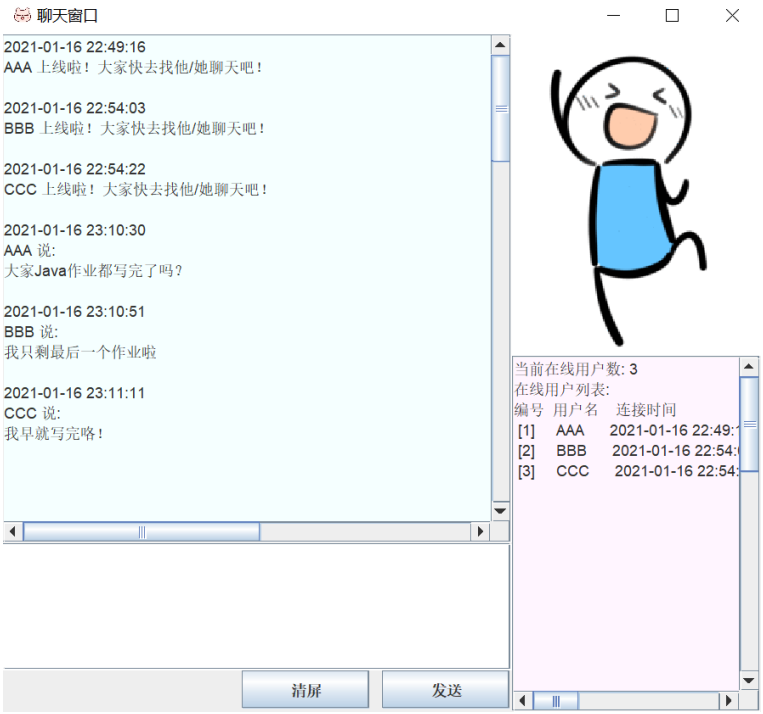


**图 29　AAA 聊天界面显示结果**

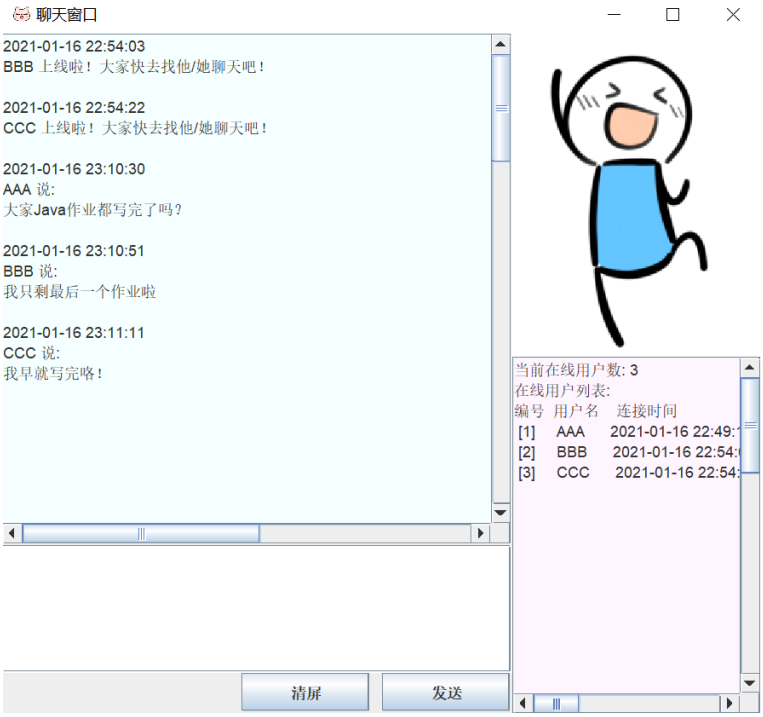


**图 30　BBB 聊天界面显示结果**

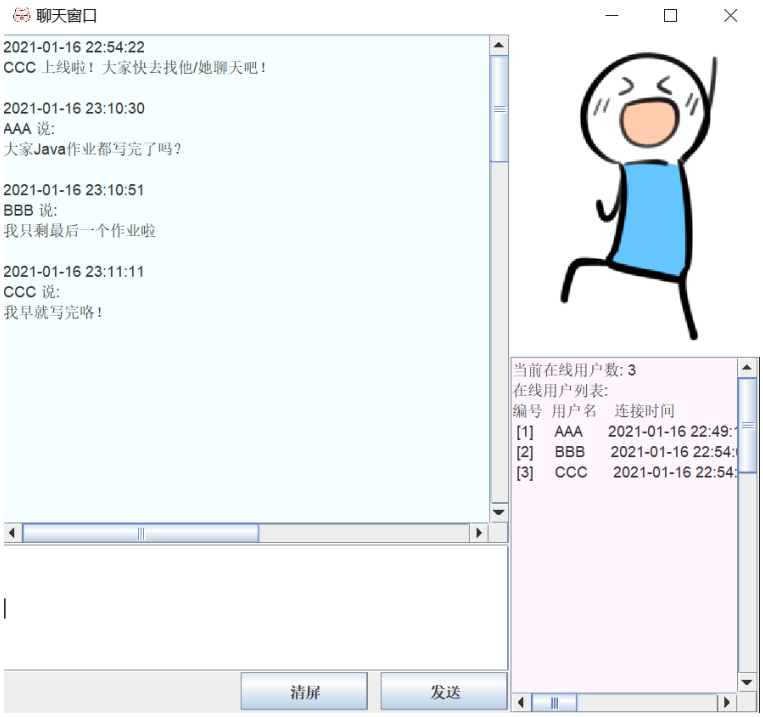


**图 31　CCC 聊天界面显示结果**

退出用户名为 AAA 和 BBB 的客户端观察各个界面的显示结果：

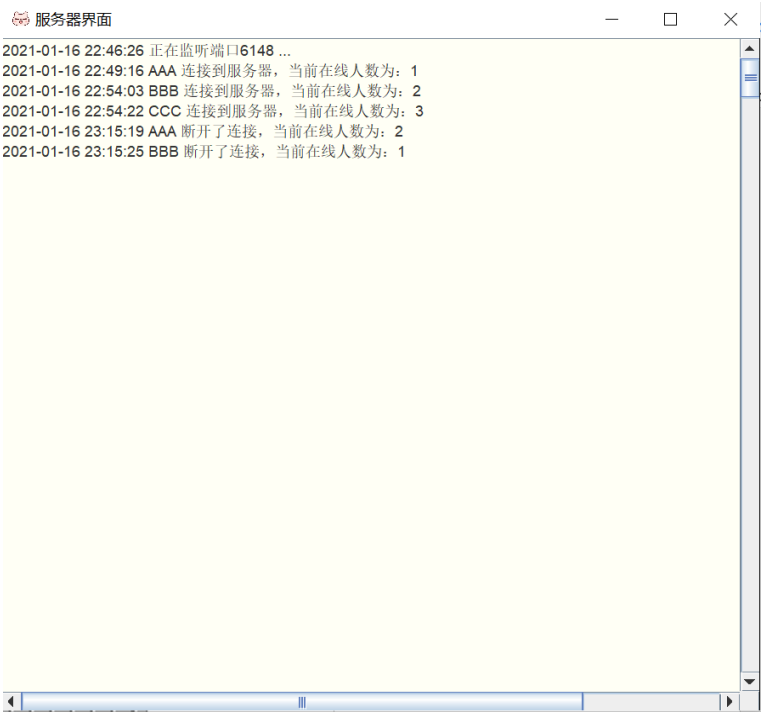


**图 32　服务器界面显示结果**

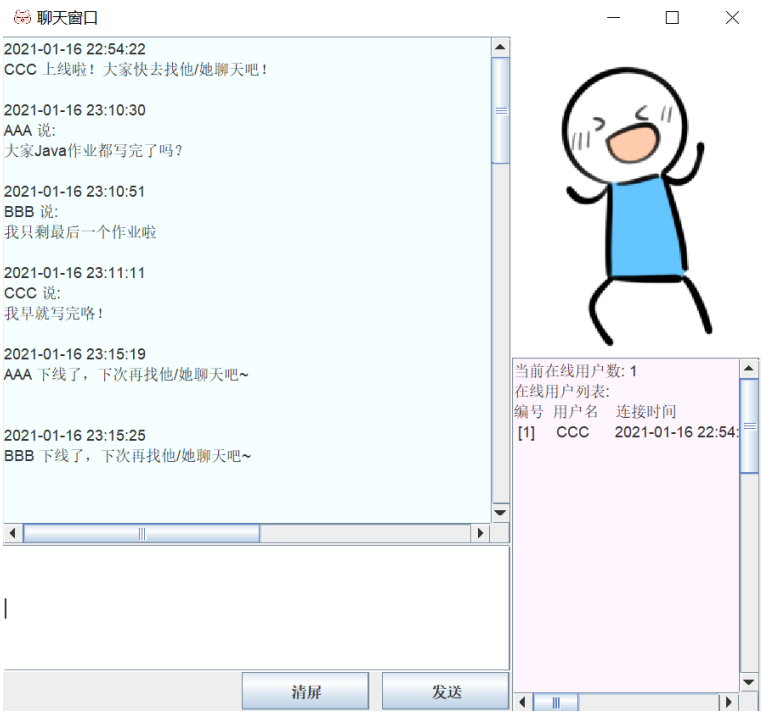


**图 33　CCC 聊天界面显示结果**

最后退出服务器，客户端将会出现弹窗：

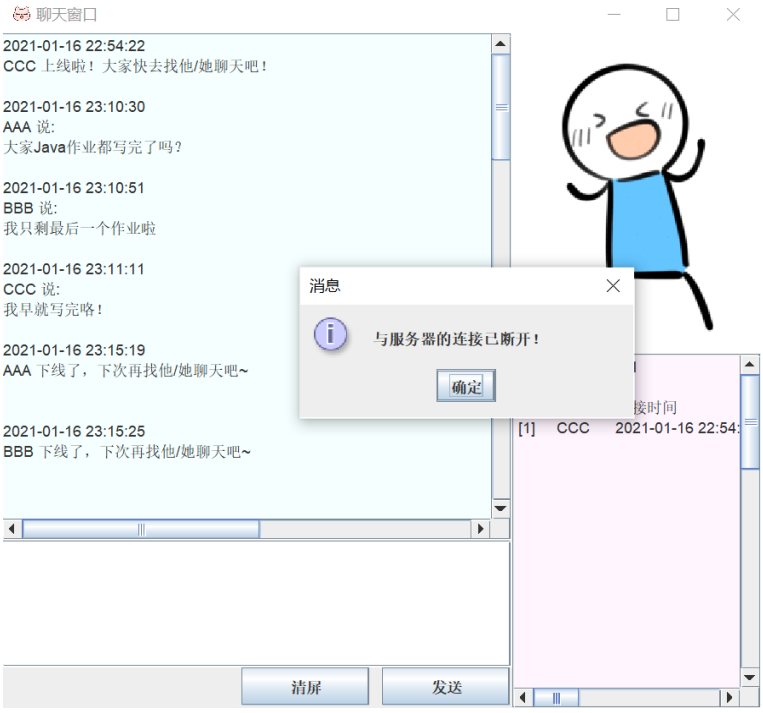


**图 34　弹窗提示与服务器的连接已断开**

## 5．总结

我本次开发的是一个多人在线聊天软件，通过这样一个综合性的项目设计与开发，我对于课堂上学习到的有关 Java 的 GUI 编程，多线程编程以及网络编程的理论知识有了进一步的理解。

在整个程序的编写过程中，由于在计算机网络课程里曾经使用过 C 语言中最底层的 socket 接口实现过一个服务器和客户端的程序，所以本次项目设计的过程中对于网络编程以及多线程编程的部分还是比较熟悉的。而且 Java 的 Socket 编程以及多线程编程相对于 C 语言来说要简单很多，因为 Java 对于相关的类都封装的很好，使用起来非常方便。

本次项目设计的过程中我最不熟悉的就是 GUI 编程的部分了，但经过实际操作我发现 Swing 框架使用起来还是非常方便的，整个页面的布局也十分灵活，页面之间的跳转通过 setVisible()方法就可以轻松实现，还有 JScrollPane 类可以直接实现滚动条的效果，以及通过弹出对话框的方式与用户交互，都让我体验到了 Swing 框架的易用性。