TCP/IP 网络编程实验

项目名：网络文字聊天系统的设计与实现

目录

一、 实验目的

1.掌握 C/S 结构软件的设计与开发方法。

2.掌握基于 Socket 的网络通信程序的设计与开发方法。

3.掌握多线程/多进程编程的基本概念与实现方法。

二、 内容要求

在 UNIX/Linux 或 Windows 环境下编写一个基于 TCP 协议的多线程/多进程网络文字聊天系统,要求聊天双方都能够从键盘读取输入信息发送给对方，同时要求聊天双方都能够同时和多个人进行基于 TCP 协议的文字聊天。

本项实验可以分组完成，每个人必须熟悉了解本项目的所有实验内容，并在此基础上完成实验报告，编程语言可以使用 C 语言或 JAVA 等其他语言，操作系统可选用 UNIX/Linux 或 Windows。

三、 程序设计原理与流程图

1. 程序设计原理

1.1 客户端向服务器发起第一次握手

类似 QQ 那样的，当客户要求跟某人说话的时候，重新开启一个客户模式，实现点对点的通信。如果成功进入服务器界面和客户端界面,表明服务器和客户端连接正常。如果用户名、服务器地址或者端口号码输入错误，将不能连接到服务器，系

统显示不能连接并且询问是否重新登陆。此时须重新设定以上三项内容然后登陆，虽然这个程序中对发送的信息判断比较多，信息的格式定义也比较多，但是难度不大，都是一些字符串的操作。

1.2 客户端发信息给服务器

客户选择聊天的对象，将信息发送给服务器，第一种处理方式就是：服务器对这个信息进行判断，找到对应的套接字发送，而不发给其他用户；另一种处理方式就是：服务器不管这个消息，只认为是普通的信息，逐次发给每一个与其相连接客户，由客户程序来判断，如果是自己的消息就显示出来，如果不是，则不加理睬（这是考虑到建立 TCP 连接过程中要完成‘三次握手’，不可以对于所有信息进行广播发出）。

1.3 服务器发信息给客户端

服务器在收到建立握手请求或是在之后连接过程中进行聊天消息的收发过程中，相当于一个中转站的作用，所有想要进入这个聊天室的客户端都需要完成与服务器端正确无误的‘三次握手’。这样进入聊天室后，服务端通过接收缓存所有用户的聊天信息以及动态更新目前在线用户信息，再通过广播及时发送给所有用户聊天界面进行显示。

1.4 有关在线用户处理

在线用户处理过程有一个问题就是在填写用户名连接服务器时，为了保证聊天室内没有两个相同用户名，处理方法就

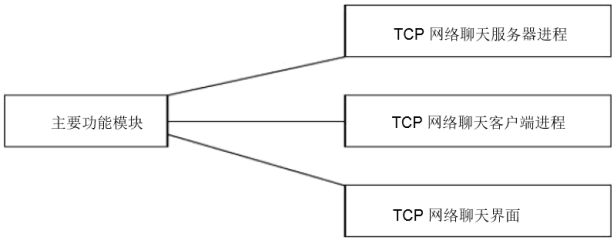
是：在出现相同用户名后，在用户名后添加线程 ID 号，以区分不同用户。同时，在线用户要显示在每个客户的聊天界面上(当然,要除去自己),也要考虑到用户退出/加入聊天室动态更新。

1.5 聊天信息与实现握手报文信息的区分

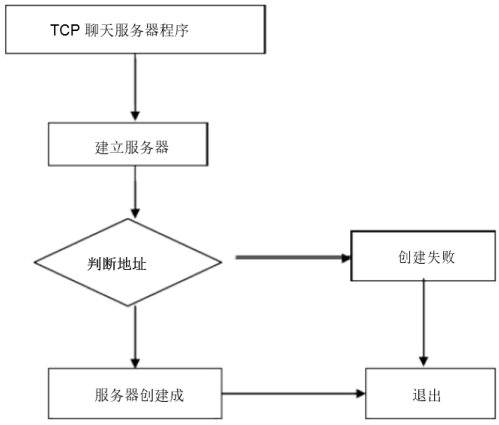
属于聊天信息就需要广播发送给每一个客户的聊天界面,属于握手过程中的报文信息就只能发送给唯一的客户端以实现 TCP 连接。区分方法有很多种，这里采用的是对信息进行包装处理（添加一些特殊信息字段来区分信息类型，这与后来区分界面不同按钮的功能处理机制是一样的方法）。

2. 程序设计流程图

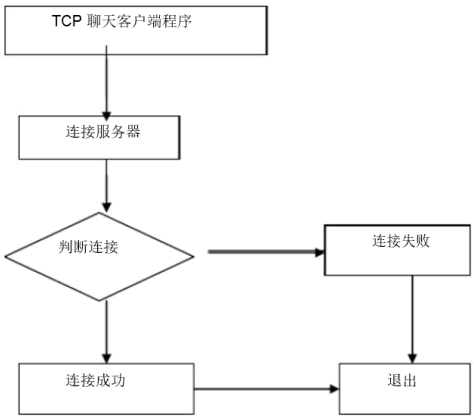
2.1 主要的几个功能模块



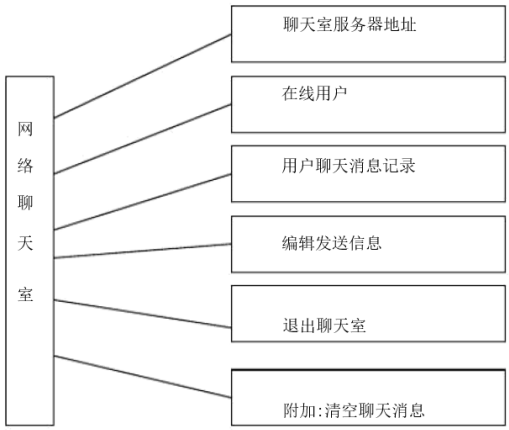
2.2 TCP 网络聊天服务器流程图



2.3 TCP 网络聊天客户端流程图

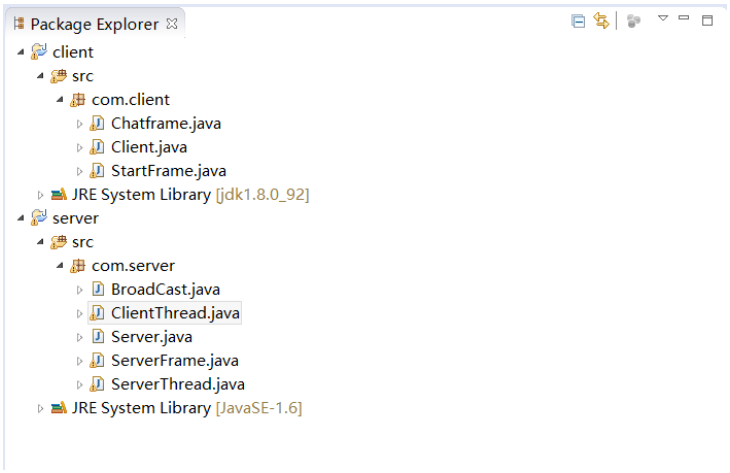


2.4 网络聊天室聊天界面程序



四、程序主要代码

1.总体分析代码结构



	类名	大致介绍
客户端	Client	客户端主程序由此执行
	StartFrame	客户端通过客户端启动界面提交用户名、IP 地址、端口号向服务器发起建立第一次握手
	Chatframe	客户端的聊天界面，显示服务端广播发送过来的聊天记录数据、在线用户数据，并向服务端发送聊天消息
服务器端	Server	服务器端主程序由此执行
	BroadCast	该类用来不停向客户端广播数据
	ClientThread	该类继承了线程类，其中的 run()方法会不断接收数据,并保存在容器中
	ServerThread	服务器线程类，用来和每个客户建立连接
	ServerFrame	服务器端界面，显示服务端缓存的聊天记录数据（含用户名、时间）、所有在线用户数据

2. 详细说明

2.1 第一次握手验证

Client.java

```
1  package com.client;
2
3  import java.io.DataInputStream;
4  import java.io.DataOutput;
5  import java.io.DataOutputStream;
6  import java.io.IOException;
7  import java.net.Socket;
8  import java.net.UnknownHostException;
9
10 public class Client extends Thread{
11
12     public String username;
13     public Socket socket;
14     public Chatframe chatframe;
15     private DataInputStream dis;
16     private DataOutputStream dos;
17     public boolean flag_exit;
18     private String chat_re;
19     public int threadId;
20     public static void main(String[] args) {
21
22         Client client = new Client();
23         StartFrame enterframe = new StartFrame(client);
24         enterframe.setVisible(true);
25     }
```

通过继承 Frame 类，设计的这个 StartFrame 类提供了客户端进行更加自主选择与某个服务端 IP 地址以及通过某个端口号进行 TCP 连接，当然少不了输入数据的检测以及进入聊天室验证（实际上这是课堂知识中的 socket()函数具体化）。

StartFrame.java

```
64          this.setTitle("QACQ聊天室");
65          this.setIconImage(Toolkit.getDefaultToolkit().createImage(StartFrame.class.getResource("start.png")));
66          this.setLayout(null);
67          this.setResizable(false);
68          this.addWindowListener(new WindowAdapter() {
69              @Override
70              public void windowClosing(WindowEvent e) {
71                  jbt_exit.doClick();
72              }
73          });
74
75          jbl_userName = new JLabel("用户名");
76          jbl_userName.setFont(new Font("宋体",Font.PLAIN,14));
77          jbl_userName.setBounds(23, 30, 81, 34);
78          this.add(jbl_userName);
79
80          jtf_userName = new JTextField();
81          jtf_userName.setBounds(114,30,143,30);
82          this.add(jtf_userName);
83
84          jbl_hostIp = new JLabel("服务器地址");
85          jbl_hostIp.setFont(new Font("宋体",Font.PLAIN,14));
86          jbl_hostIp.setBounds(23, 74, 81, 34);
87          this.add(jbl_hostIp);
88
89          jtf_hostIp = new JTextField();
90          jtf_hostIp.setBounds(114,74,143,30);
91          this.add(jtf_hostIp);
92          try {
93              String ip = Inet4Address.getLocalHost().getHostAddress();
94              jtf_hostIp.setText(ip);
95          } catch (UnknownHostException e1) {
96              e1.printStackTrace();
97          }
98
99          jbl_hostPost = new JLabel("端口号");
100         jbl_hostPost.setFont(new Font("宋体",Font.PLAIN,14));
101         jbl_hostPost.setBounds(23, 118, 81, 34);
102         this.add(jbl_hostPost);
```

StartFrame.java

```
122     @Override
123     public void actionPerformed(ActionEvent e) {
124         if (e.getSource() == jbt_exit) {
125             this.setVisible(false); //关闭这个启动界面
126             client.exitLogin();
127         }
128         if (e.getSource() == jbt_enter) {
129             String username = jtf_userName.getText().trim();
130             String hostIp = jtf_hostIp.getText().trim();
131             String hostPost = jtf_hostPost.getText().trim();
132             if (!username.equals("")) {
133                 if (!hostIp.equals("")) {
134                     if (!hostPost.equals("")) {
135                         String result = null;
136                         result = client.login(username,hostIp,hostPost);
137                         if (result.equals("true")) {//验证通过
138                             client.showChatframe();
139                             this.setVisible(false);
140                         }else {
141                             JOptionPane.showMessageDialog(this, result);
142                         }
143                     }else {
144                         JOptionPane.showMessageDialog(this, "端口号不能为空");
145                     }
146                 }else {
147                     JOptionPane.showMessageDialog(this, "服务器地址ip不能为空");
148                 }
149             }else {
150                 JOptionPane.showMessageDialog(this, "用户名不能为空");
151             }
152         }
153     }
```

Client.java

```
32      //连接进入聊天室验证
33      public String login(String username, String hostIp, String hostPost) {
34          this.username = username;
35          String message = null;
36          try {
37              socket = new Socket(hostIp,Integer.parseInt(hostPost));
38          } catch (NumberFormatException e) {
39              message = "端口号异常 只能是 1024<post<65535";
40              return message;
41          } catch (UnknownHostException e) {
42              message = "服务器地址错误";
43              return message;
44          } catch (IOException e) {
45              message = "与服务器连接异常";
46              return message;
47          }
48          return "true";
49      }
```

2.2 第二、三次握手

2.2.1 客户端在建立第一次握手成功后，由服务端给客户端（注意这里不是广播给客户端发出响应）发出响应报文,即识别出信息中包含@clientThread，说明是第二次握手。第三次握手时，由客户端将用户名信息发给服务端。至此就实现了‘三次握手’。

Client.java

```
72      @Override
73      public void run() {
74          //客户读取数据
75          while (flag_exit) {
76              try {
77                  chat_re = dis.readUTF();
78
79              } catch (IOException e) {
80                  flag_exit = false;
81                  if (!chat_re.contains("serverexit")) {
82                      chat_re = null;
83                  }
84              }
85              if (chat_re!=null) {
86                  if (chat_re.contains("@clientThread")) {//如果信息中包含@clientThread 说明是第二次握手
87                      //切割消息内容
88                      int local = chat_re.indexOf("@clientThread");
89                      threadId = Integer.parseInt(chat_re.substring(0, local));
90                      try {
91                          dos.writeUTF(username + "@login" + threadId + "@login");//第三次握手时，将用户名信息发给服务器
92                      } catch (IOException e) {
93                          // TODO Auto-generated catch block
94                          e.printStackTrace();
95                      }
96                  }else {
97                      if (chat_re.contains("@userlist")) {
98                          chatframe.setDisUser(chat_re);
99                      }else {
100                         if (chat_re.contains("@chat")) {
101                             chatframe.setDisMess(chat_re);
102                         }else {
103                             if (chat_re.contains("@serverexit")) {
104                                 chatframe.closeClient();
105                             }
106                         }
107                     }
108                 }
109             }
110         }
111     }
```

2.2.2不断接收数据保存容器中(ClientThread类),注意有

一点特殊情况，对于出现相同用户名的情况干扰聊天，采取在用户名后加上线程ID号以区分开来。

```
@Override
public void run() {
    while(flag_exit){
        try {
            String message = dis.readUTF();
            if (message.contains("@login")) {
                String[] userInfo = message.split("@login");//进行字符串切割成
            字符数组
                int userID = Integer.parseInt(userInfo[1]); //获取线程ID
                serverThread.users.remove(userID);
                //判断集合中是否存在该用户名
                if (serverThread.users.containsValue(userInfo[0])) {
                    //遍历线程集合得到每个线程ID
            for (int i = 0; i < serverThread.clients.size(); i++) {
                int threadID = (int) serverThread.clients.get(i).getId();
                //判断哪个线程ID对应的用户名和userInfo[0] 一样
                if (serverThread.users.get(threadID).equals(userInfo[0])) {
                //删除该线程的信息
                serverThread.users.remove(threadID);
            //重新添加用户信息
            serverThread.users.put(threadID, userInfo[0] + "_" + threadID);
            break;
                }
            }
            //添加新的用户信息
            serverThread.users.put(userID, userInfo[0] + "_" + userID);
        }else {
            serverThread.users.put(userID, userInfo[0]);
        }
        //将在线用户显示在服务器界面右边
        message = null;
        StringBuffer sb = new StringBuffer();
        synchronized (serverThread.clients) {
        //遍历集合得到每一个线程ID ,将线程ID 对应的用户名添加到StringBuffer 中
            for (int i = 0; i < serverThread.clients.size(); i++) {
                int id = (int) serverThread.clients.elementAt(i).getId();
                //得到线程对应的用户名
                String username = serverThread.users.get(new Integer(id));
                sb.append(username + "@userlist"+id + "@userlist");//这是
                }
```

```
                }
            message = new String(sb);
            serverThread.serverFrame.setDisUser(message);//将用户显示在服务器界面
        }else {
            if (message.contains("@exit")) {//表示客户端推出聊天室
                String[] userInfo = message.split("@exit");
                int userID = Integer.parseInt(userInfo[1]);
                serverThread.users.remove(userID);
                message = null;
                StringBuffer sb = new StringBuffer();
                synchronized (serverThread.clients) {
                for (int i = 0; i < serverThread.clients.size(); i++) {
                    //获取每个线程id
                   int threadID=(int) serverThread.clients.elementAt(i).getId();
                    if (userID == threadID) {
                          serverThread.clients.removeElementAt(i);
                          i--;
                     }else {
        sb.append(serverThread.users.get(new Integer(threadID)) + "@userlist");
        sb.append(threadID + "@userlist");
                     }
                }
            }
        message = new String(sb);
        if (message.equals("")) {
            serverThread.serverFrame.setDisUser("@userlist");
        }else {
            serverThread.serverFrame.setDisUser(message);
        }
        this.clientSocket.close();
        }else {
            //将聊天信息显示在服务器界面左边
            if (message.contains("@chat")) {
            String[] chat = message.split("@chat");
            StringBuffer sb = new StringBuffer();
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            String date = sdf.format(new Date());
            sb.append(chat[0] + "  " + date + "\n");
            sb.append(chat[2] + "@chat");
            //将消息显示到服务器界面
            message = new String(sb);
            serverThread.serverFrame.setDisMess(message);
        }
    }
```

```
}
    synchronized (serverThread.message) {//同一时刻，只容许一个用户进行保存到容器
        if (message != null) {//将信息保存到容器中,如果信息为空，不用保存!
            serverThread.message.add(message);
        }
    }
} catch (Exception e) {
        }
    }
}
```

2.2.3不同过程中信息的广播流向(Broadcast类)

BroadCast.java

```
@Override
public void run() {
    boolean flag = true;
    while (flag_exit) {
        synchronized (serverThread.message) {//同步代码块，一个线程在操作，不容许另外的线程来操作
            if (serverThread.message.isEmpty()) {//先要判断容器中是否有数据
                continue;
            }else {
                str = serverThread.message.firstElement();//获取第一条信息
                serverThread.message.removeElement(str);//取出以后，就要删掉这条信息
                if (str.contains("@clientThread")) {//第二次握手 需要发的数据不是针对所有客户端，而是只发给那个与他建立了第一次握手的客户端
                    flag = false;
                }
            }
        }
        synchronized (serverThread.clients) {
            for (int i = 0; i < serverThread.clients.size(); i++) {
                clientThread = serverThread.clients.elementAt(i);//遍历线程集合，得到每个用户对应的线程
                if (flag) {
                    try {
                        if (str.contains("@chat") || str.contains("@userlist") || str.contains("@serverexit")) {
                            clientThread.dos.writeUTF(str);
                        }
                    } catch (Exception e) {
                        // TODO: handle exception
                    }
                }else {
                    String value = serverThread.users.get((int) clientThread.getId());
                    System.out.println(value);
                    if (value.contains("@login@")) {
                        flag =true;
                        try {
                            clientThread.dos.writeUTF(str);
                        } catch (IOException e) {

                            e.printStackTrace();
                        }
                    }
                }
            }
        }
```

2.3 服务器界面

带有@chat标记的信息会被识别为聊天消息用来识别在聊天界面以及服务端后台中，还有弄清退出/停止服务端的区别。

ServerFrame.java

```
    //将聊天信息显示到聊天界面
    public void setDisMess(String mess) {
        if (mess.contains("@exit")) {
            jta_mess.setText("");
        }
        if (mess.contains("@chat")) {
            int local = mess.indexOf("@chat");
            String mess_re = mess.substring(0,local);
            jta_mess.append(mess_re+"\n");
            jta_mess.setCaretPosition(jta_mess.getText().length());
        }
    }
```

```java
    //将用户显示在服务器界面右侧
    public void setDisUser(String mess) {
        if (mess.equals("@userlist")) {
            jlt_user.removeAll();
            jlt_user.setListData(new String[]{});
        }else {
            if (mess.contains("@userlist")) {
                String[] dis = mess.split("@userlist");
                String[] disUsers = new String[dis.length/2];
                int j = 0;
                for (int i = 0; i < dis.length; i++) {
                    disUsers[j++] = dis[i++];
                }
                jlt_user.removeAll();
                jlt_user.setListData(disUsers);//保存所有用户名
            }
            if (mess.equals("@exit")) {
                jlt_user.setListData(new String[]{});
            }
        }
    }
```

ServerThread.java

```java
    public ServerThread(ServerFrame serverFrame) {
        this.serverFrame = serverFrame;
        message = new Vector<String>();
        clients = new Vector<ClientThread>();
        users = new HashMap<Integer,String>();
        try {
            serverSocket = new ServerSocket(5000);
        } catch (IOException e) {
            this.serverFrame.setStartAndStopUnable();//不能同时开启两个服务器
            System.exit(0);
        }
        //开启广播线程(服务端向客户端发送信息)
        broadCast = new BroadCast(this);
        broadCast.flag_exit = true;
        broadCast.start();
    }
    @Override
    public void run() {
        Socket socket;
        while(flag_exit){
            if (serverSocket.isClosed()) {
                flag_exit = false;
            }else {
                try {
                    socket = serverSocket.accept();
                } catch (IOException e) {
                    socket = null;
                    flag_exit = false;
                }
                if (socket != null) {
                    //从客户端接受信息
                    ClientThread clientThread = new ClientThread(this,socket);
                    clientThread.flag_exit = true;
                    clientThread.start();   //开启线程

                    synchronized (clients) {    //保存每个用户对应的clientThread对象
                        clients.addElement(clientThread);
                    }

                    synchronized (message) {    //为了完成第二次握手，服务端向客户端发送的数据信息
                        users.put((int)clientThread.getId(), "@login@");    //获取线程id
```

2.4 附加的一些小功能模块

2.4.1在窗口加入自定义图标

```
this.setIconImage(Toolkit.getDefaultToolkit().createImage(Ser
ver.class.getResource("图标存放的相对路径")));
//需要导包 import java.awt.Toolkit;
```

2.4.2实现聊天信息发送按钮等效于回车键

```
public void keyPressed(KeyEvent e) {
```

```
    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
        jbt_tran.doClick();//如果事件源是回车的话
    }
}
```

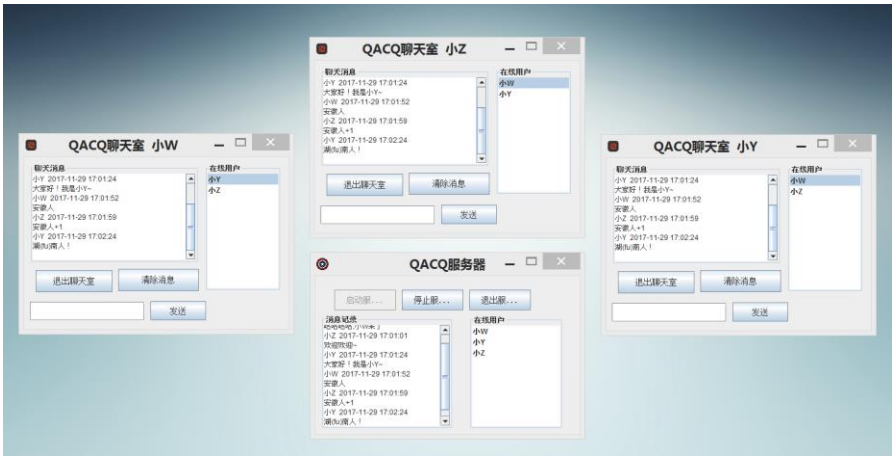
2.4.3实现清除聊天记录

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jbt_clear) {
        jta_mess.setText("");
    }
}
```

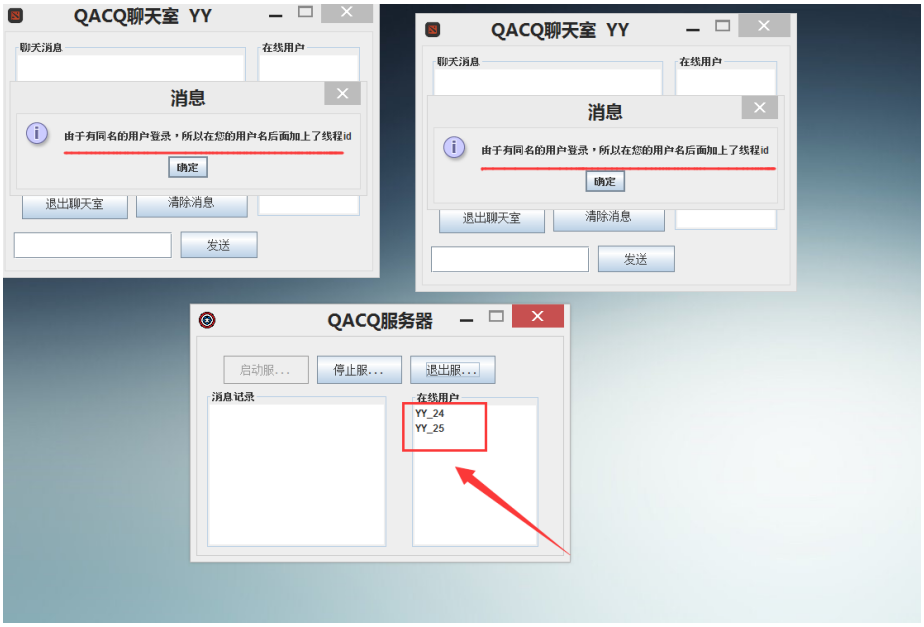
五、程序功能截图

1.连接服务器的一些异常情况列举

2.正常聊天室情况



3. 当已有用户 YY 连入服务器端时，面对同用户名 YY 连接



4. 服务端出现异常关闭

5. 关闭聊天室界面，观察在线用户信息的更新

六、 设计心得与体会

1. 周末两天的网络编程课程设计到今天为止完满结束，相信一起努力的我们多多少少都有相应的磨练与收获。通过本次网络编程的 TCP/IP 网络聊天室设计，我学会了程序设计与实现的一般方法，了解了 JAVA 编程技术的网络应用，领悟到用 JAVA 实现动态更新的方法要领，同时也是对这段时间所学内容的一个综合复习。

2. 不过，在系统的开发过程中，确实遇到了不少问题，特别是在之后的程序的调试中。刚开始，由于对一些代码的不熟悉、不够熟练理解与运用，就是一个小小的错误也花了不少的时间和精力，时常要翻看 JAVA 语言的 API，看看需要用到的知识点，一点一滴的了解掌握，敲代码的时候虽然错误每次都是

一大版,但一个个仔细改了之后,程序能够运行就会油然而生一种成就感，当然有的错误自己都不知道错在哪，特别是在连接的时候，每次都快崩溃了，不断地查阅信息资料，在网上搜索相关素材，同时也请教讲师和组内外的同学，才把它搞好。

3.随着对一些相关知识的不断学习，慢慢也了解了网络编程的过程，后来也慢慢就缩短了课程设计所用的时间，在整个网络聊天室设计和完成的过程中，我觉得，功夫是不会不负有心人的，最后终于成功地完成了此次课程设计的所有要求，同时除了上面具体说的那些我还拓宽了自己的眼界，将书上的理论与实践有效的结合转换我自己的知识，也有了一点点的设计经验，又学到了不少的知识，我相信以后不论是在工作中还是学习中对我都有很大的帮助。