

**计算机网络**

实验报告

|  |  |
| --- | --- |
| 题目 | B1-套接字编程-聊天程序 |
| 学生姓名 | 谭哲文 |
| 学 号 | 8202191123 |
| 专业班级 | 计科2105 |
| 指导教师 | 王建新,王伟平 |
| 学 院 | 计算机学院 |
| 完成时间 | 2023-05-22 |

目录

[一、实验目的 2](#_Toc104128490)

[二、实验要求 2](#_Toc104128491)

[三、实验内容 2](#_Toc104128492)

[四、实验方案设计与实施 3](#_Toc104128493)

[1. Socket编程接口 3](#_Toc104128494)

[2. 服务端设计与实施 4](#_Toc104128495)

[Server.java 5](#_Toc104128496)

[ParseMsg.java 8](#_Toc104128497)

[3、客户端详细设计 9](#_Toc104128498)

[UserLogin.java 10](#_Toc104128499)

[Client.java 11](#_Toc104128500)

[五、实验结果展示与分析 14](#_Toc104128501)

[1、用户登陆界面 15](#_Toc104128502)

[2、聊天界面 16](#_Toc104128503)

[3、实现广播、组播和点对点 16](#_Toc104128504)

[4、实现用户离开提醒 17](#_Toc104128505)

[六、实验心得体会 19](#_Toc104128506)

# 一、实验目的

1、掌握C++、JAVA或Python等集成开发环境编写网络程序的方法；

2、掌握客户/服务器（C/S）应用的工作方式；

3、学习网络中进程之间通信的原理和实现方法；

# 二、实验要求

1、编写一个基于socket的简易聊天程序。所编写的程序应具有如下功能：

2、具有点对点通信功能，任意客户端之间能够发送消息；

3、具有群组通信功能，客户端能够向组内成员同时发送消息，其他组成员不能收到；

4、具有广播功能，客户端能够向所有其他成员广播消息；

5、要求本机既是客户端又是服务器端；

# 三、实验内容

1、初始化服务端套接字(ServerSocket)，为TCP连接的建立做准备,服务器首先启动，根据定义的端口号与套接字进行绑定。

2、启动监听，不断接受客户端请求建立TCP连接。

3、接受客户端请求，建立相应套接字，启动用户线程。

4、根据套接字获取输入输出流，从而实现接收报文。

5、解析传输信息

其中包括：

（1）用户登录信息

（2）某用户离开信息

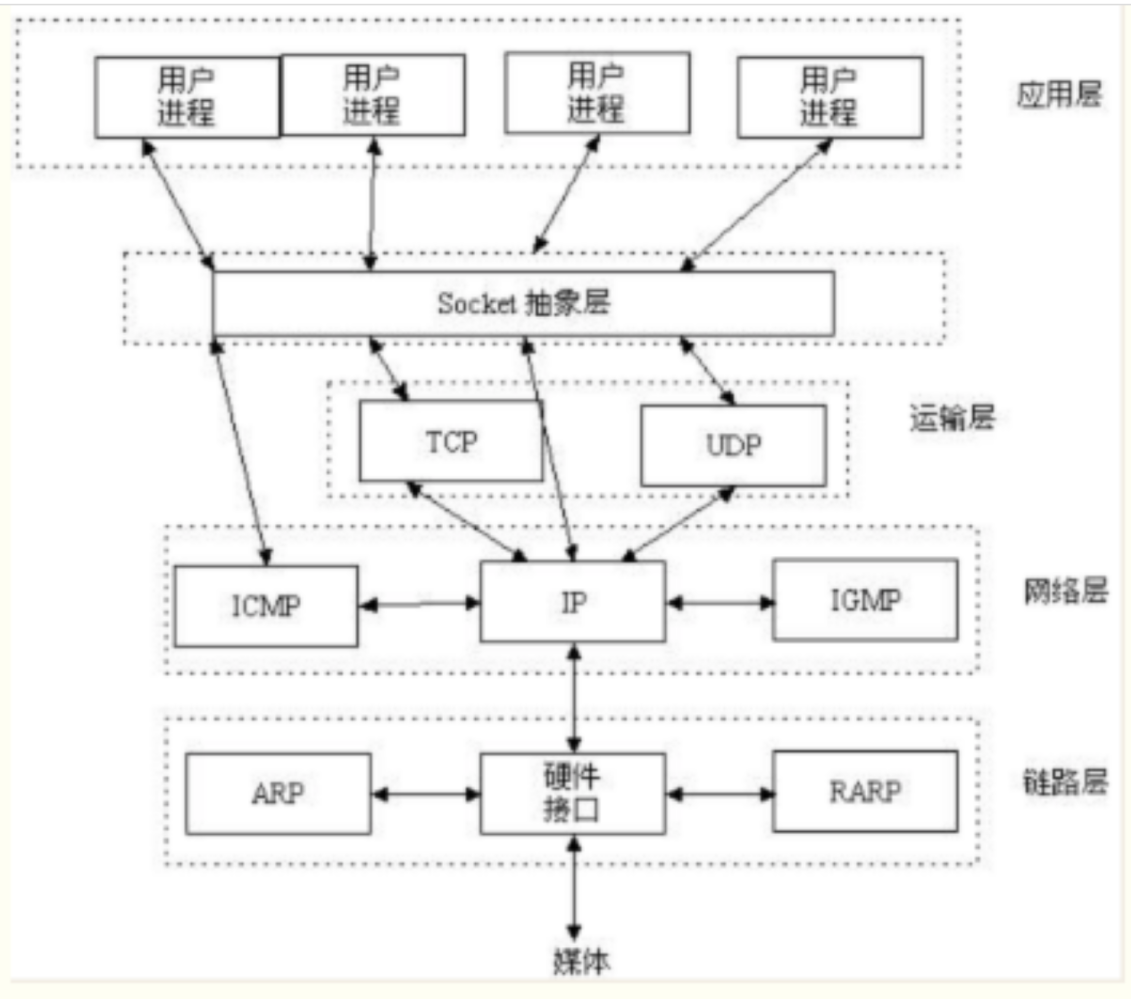
（3）用户广播、组播、点对点方式传输的信息

6、若用户离开，关闭用户线程，关闭输入流、输出流、套接字，蒋用户线程移出监听线程列表。

# 四、实验方案设计与实施

## 1. Socket编程接口

要实现 Web 服务器，需使用套接字 Socket编程接口来使用操作系统提供的网络通信功能。 Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，是一组编程接口。它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。使用 Socket 后，无需深入理解 TCP/UDP 协议细节（因为Socket 已经为我们封装好了），只需要遵循 Socket 的规定去编程，写出的程序自然就是遵循 TCP/UDP 标准的。Socket 的地位如下图所示：



从某种意义上说，Socket 由地址IP和端口Port构成。IP 是用来标识互联网中的一台主机的位置，而 Port 是用来标识这台机器上的一个应用程序，IP 地址是配置到网卡上的，而 Port 是应用程序开启的，IP 与 Port 的绑定就标识了互联网中独一无二的一个应用程序。

套接字类型 流式套接字（SOCK\_STREAM）：用于提供面向连接、可靠的数据传输服务。 数据报套接字（SOCK\_DGRAM）：提供了一种无连接的服务。该服务并不能保证数据传输的可靠性，数据有可能在传输过程中丢失或出现数据重复，且无法保证顺序地接收到数据。 原始套接字（SOCK\_RAW）：主要用于实现自定义协议或底层网络协议。

## 2. 服务端设计与实施

服务端需要依次实现以下功能：

（1）建立ServerSocket，启动线程循环等待连接

（2）根据连接创建套接字，并根据套接字建立输入输出流，建立用户线程并存储相关信息

（3）用户线程不断接受客户端发来的信息，并对信息进行解析

（4）服务器根据用户线程接收并解析的信息，执行相应动作，其中包括：通知上线、下线、并实现广播、组播、点对点的逻辑。

代码如下：

### Server.java

|  |
| --- |
| import java.awt.BorderLayout;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.io.PrintStream;  import java.net.ServerSocket;  import java.net.Socket;  import java.util.ArrayList;  import javax.swing.JFrame;  import javax.swing.JTextArea;  // 服务器类  public class Server extends JFrame implements Runnable {      private ServerSocket ss = null;      private JTextArea jta = new JTextArea();      // 存储正在运行的用户线程      private ArrayList<ChatThread> chatThread = new ArrayList<ChatThread>();      public Server() throws Exception {          this.setTitle("服务器");          this.setSize(300, 400);          this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);          this.setVisible(true);          this.add(jta, BorderLayout.NORTH);          ss = new ServerSocket(8080);          new Thread(this).start();      }      // 线程运行内容      public void run() {          while (true) {              try {                  Socket socket = this.ss.accept();                  // 解析用户登录信息                  BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));                  String msg;                  while ((msg = bufferedReader.readLine()) == null)                      ;                  jta.append(msg + "\n");                  sendMsg(msg);                  ParseMsg parseMsg = new ParseMsg(msg);                  // 根据用户名、聊天室名称、套接字创建用户聊天线程                  ChatThread tmp = new ChatThread(parseMsg.getSourse(), parseMsg.getTarget(), socket);                  // 线程开启                  tmp.start();                  // 向新用户发送服务器用户情况                  PrintStream printStream = new PrintStream(socket.getOutputStream());                  msg = ("UserName/");                  for (ChatThread i : chatThread) {                      msg = msg + i.userName + "/";                  }                  printStream.println(msg);                  chatThread.add(tmp);              } catch (IOException ioe) {              }          }      }      // 根据消息解析并按不同方式发送      public void sendMsg(String msg) {          ParseMsg parseMsg = new ParseMsg(msg);          // 广播          if (parseMsg.getMethod().equals("BROADCAST")) {              broadcast(parseMsg.getMsg());              // 群组          } else if (parseMsg.getMethod().equals("GROUP")) {              group(parseMsg.getTarget(), parseMsg.getMsg());              // 点对点          } else if (parseMsg.getMethod().equals("PTP")) {              ptp(parseMsg.getTarget(), parseMsg.getMsg());              // 用户退出          } else if (parseMsg.getMethod().equals("LEAVE")) {              for (ChatThread i : chatThread) {                  if (parseMsg.getSourse().equals(i.userName)) {                      // 中断线程                      i.interrupt();                      // 从运行列表中移出                      chatThread.remove(i);                      try {                          // 关闭输入流、输出流、套接字                          i.bufferedReader.close();                          i.printStream.close();                          i.socket.close();                          break;                      } catch (IOException ioe) {                          ioe.printStackTrace();                      }                  }              }              // 广播用户离开信息              broadcast(parseMsg.getMsg());          }      }      // 广播      public void broadcast(String msg) {          for (ChatThread i : chatThread) {              i.printStream.println(msg);          }      }      // 群组      public void group(String target, String msg) {          for (ChatThread i : chatThread) {              if (i.chatRoom.equals(target))                  i.printStream.println(msg);          }      }      // 点对点      public void ptp(String target, String msg) {          for (ChatThread i : chatThread) {              if (i.userName.equals(target)) {                  i.printStream.println(msg);                  break;              }          }      }      // 聊天线程类      class ChatThread extends Thread {          String userName = null;          String chatRoom = null;          Socket socket = null;          BufferedReader bufferedReader = null;          PrintStream printStream = null;          public ChatThread(String userName, String chatRoom, Socket socket) {              this.userName = userName;              this.chatRoom = chatRoom;              this.socket = socket;              try {                  this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));                  this.printStream = new PrintStream(socket.getOutputStream());              } catch (IOException ioe) {                  ioe.printStackTrace();              }          }          // 线程运行内容          public void run() {              // 不断从输入流中获取信息              while (true) {                  try {                      String msg = bufferedReader.readLine();                      if (msg != null && !msg.equals("")) {                          jta.append(msg + "\n");                          sendMsg(msg);                      }                  } catch (Exception e) {                  }              }          }      }      public static void main(String[] args) throws Exception {          new Server();      }  } |

### ParseMsg.java

|  |
| --- |
| // 解析信息类  public class ParseMsg  {      private String method;      private String target=null;      private String sourse=null;      private String msg=null;      public ParseMsg(String msg)      {          this.method=msg.split("/")[0];          this.sourse=msg.split("/")[1];          this.target=msg.split("/")[2];          this.msg=msg.split(msg.split("/")[2]+"/")[1];          System.out.println(msg+"   "+this.getMsg());      }      // 获取信息类型      public String getMethod()      {          return this.method;      }      // 获取信息发送方      public String getSourse()      {          return this.sourse;      }      // 获取信息目标方      public String getTarget()      {          return this.target;      }      // 获取信息内容      public String getMsg()      {          return this.msg;      }  } |

## 3、客户端详细设计

客户端需要依次实现以下功能

（1）登录界面，其中包括要求用户输入用户名和加入的聊天室的名称

（2）聊天界面，存在文本框使得用户能够打字发送信息，存在选择框使用户能够选择发送对象。

（3）根据服务端对应的IP地址及端口号创建套接字，从中获取输入输出流实现网络通信。

（4）根据服务器发送来的信息或用户的操作，做出相应的动作。如，在聊天消息框中打印聊天信息，在用户编辑完信息后发送时、上线下线时向服务端发送相应信息。

源代码如下：

### UserLogin.java

|  |
| --- |
| import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.net.Socket;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JTextField;  // 用户登录界面类  public class UserLogin extends JFrame implements ActionListener {      private String UserName = null;      private String chatRoom = null;      private JLabel jl1 = new JLabel("请输入用户名:");      private JTextField jtf1 = new JTextField();      private JLabel jl2 = new JLabel("请输入聊天室:");      private JTextField jtf2 = new JTextField();      private Socket socket;      private JButton jb = new JButton("确定");      private JLabel jle = new JLabel();      public UserLogin() {          this.setTitle("登录");          this.setSize(420, 400);          this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);          this.setVisible(true);          this.setLayout(null);          jl1.setBounds(80, 100, 120, 20);          this.add(jl1);          jl2.setBounds(80, 150, 120, 20);          this.add(jl2);          jtf1.setBounds(200, 100, 120, 20);          jtf1.addActionListener(this);          this.add(jtf1);          jtf2.setBounds(200, 150, 120, 20);          jtf2.addActionListener(this);          this.add(jtf2);          jb.setBounds(160, 180, 80, 20);          jb.addActionListener(this);          this.add(jb);          jle.setBounds(180, 200, 120, 20);          this.add(jle);      }      public void actionPerformed(ActionEvent e) {          // 获取用户名          this.UserName = jtf1.getText();          // 获取聊天室名          this.chatRoom = jtf2.getText();          if (!this.UserName.equals("") && !this.chatRoom.equals("")) {              try {                  // 创建套接字                  socket = new Socket("127.0.0.1", 8080);                  // 登录界面关闭窗口                  this.dispose();                  // 实例化用户类                  new Client(this.UserName, this.chatRoom, this.socket);              } catch (Exception ex) {                  jle.setText("连接失败，请重试!");              }          } else {              jle.setText("请重新输入!");          }      }      public static void main(String[] args) {          new UserLogin();      }  } |

### Client.java

|  |
| --- |
| import java.awt.BorderLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.awt.event.WindowListener;  import java.awt.event.WindowEvent;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.io.PrintStream;  import java.net.Socket;  import javax.swing.JComboBox;  import javax.swing.JFrame;  import javax.swing.JTextArea;  import javax.swing.JTextField;  // 用户类  public class Client extends JFrame implements Runnable, ActionListener, WindowListener {      private String userName;      private String chatRoom;      private Socket socket;      private PrintStream ps = null;      private BufferedReader bf = null;      private JTextField jtf = new JTextField();      private JTextArea jta = new JTextArea();      private JComboBox<String> jcb = new JComboBox<String>(new String[] { "Server" });      public Client(String userName, String chatRoom, Socket socket) throws Exception {          this.userName = userName;          this.chatRoom = chatRoom;          jcb.addItem(this.chatRoom);          this.setTitle(userName + "-已连接");          this.setSize(400, 400);          this.setDefaultCloseOperation(HIDE\_ON\_CLOSE);          this.setVisible(true);          this.add(jtf, BorderLayout.SOUTH);          this.add(jta, BorderLayout.CENTER);          this.add(jcb, BorderLayout.NORTH);          bf = new BufferedReader(new InputStreamReader(socket.getInputStream()));          ps = new PrintStream(socket.getOutputStream());          ps.println("BROADCAST/" + userName + "/" + chatRoom + "/" + userName + "已进入" + chatRoom + "\n");          jtf.addActionListener(this);          this.addWindowListener(this);          new Thread(this).start();      }      // 根据发送方法发送信息      public void sendmsg(String msg) {          if (jcb.getSelectedItem().equals("Server")) {              ps.println("BROADCAST/" + userName + "/Server/" + msg);          } else if (jcb.getSelectedItem().equals(this.chatRoom)) {              ps.println("GROUP/" + userName + "/" + this.chatRoom + "/" + msg);          } else {              ps.println("PTP/" + userName + "/" + jcb.getSelectedItem() + "/" + msg);          }      }      // 线程运行内容      public void run() {          while (true) {              try {                  String msg = bf.readLine();                  // 新用户进入聊天室                  if (msg.indexOf("已进入") != -1) {                      jcb.addItem(msg.split("已进入")[0]);                      jta.append(msg + "\n");                      // 用户离开聊天室                  } else if (msg.indexOf("已离开") != -1) {                      jcb.removeItem(msg.split("已离开")[0]);                      jta.append(msg + "\n");                      // 接收已经在聊天室的用户信息                  } else if (msg.startsWith("UserName")) {                      String[] parse = msg.split("/");                      for (int i = 1; i < parse.length; i++) {                          jcb.addItem(parse[i]);                      }                  } else {                      jta.append(msg + "\n");                  }              } catch (Exception e) {              }          }      }      // 如果发送信息，根据下拉列表框内容确定发送方式      public void actionPerformed(ActionEvent e) {          String msg = new String(this.userName + "对");          if (jcb.getSelectedItem().equals("Server")) {              msg += "所有人说：";          } else if (jcb.getSelectedItem().equals(this.chatRoom)) {              msg += "聊天室" + this.chatRoom + "中的所有人说：";          } else {              msg += jcb.getSelectedItem() + "说：";          }          if (!jcb.getSelectedItem().equals("Server") && !jcb.getSelectedItem().equals(this.chatRoom)) {              jta.append(msg + ":" + jtf.getText() + "\n");          }          sendmsg(msg + jtf.getText() + "\n");          jtf.setText("");      }      public void windowActivated(WindowEvent e) {      }      // 如果窗口关闭，则发送用户离开信息      public void windowClosing(WindowEvent e) {          ps.println("LEAVE/" + this.userName + "/" + this.chatRoom + "/" + this.userName + "已离开" + this.chatRoom + "\n");          this.ps.close();          try {              this.bf.close();          } catch (IOException ioe) {              ioe.printStackTrace();          }          this.dispose();          System.exit(0);      }      public void windowClosed(WindowEvent e) {      }      public void windowIconified(WindowEvent e) {      }      public void windowOpened(WindowEvent e) {      }      public void windowDeiconified(WindowEvent e) {      }      public void windowDeactivated(WindowEvent e) {      }  } |

# 五、实验结果展示与分析

实验操作系统:Windows 10

实验工具：VSCode

### 1、用户登陆界面

预设用户1、用户2进入同一聊天室，用户3进入另一聊天室

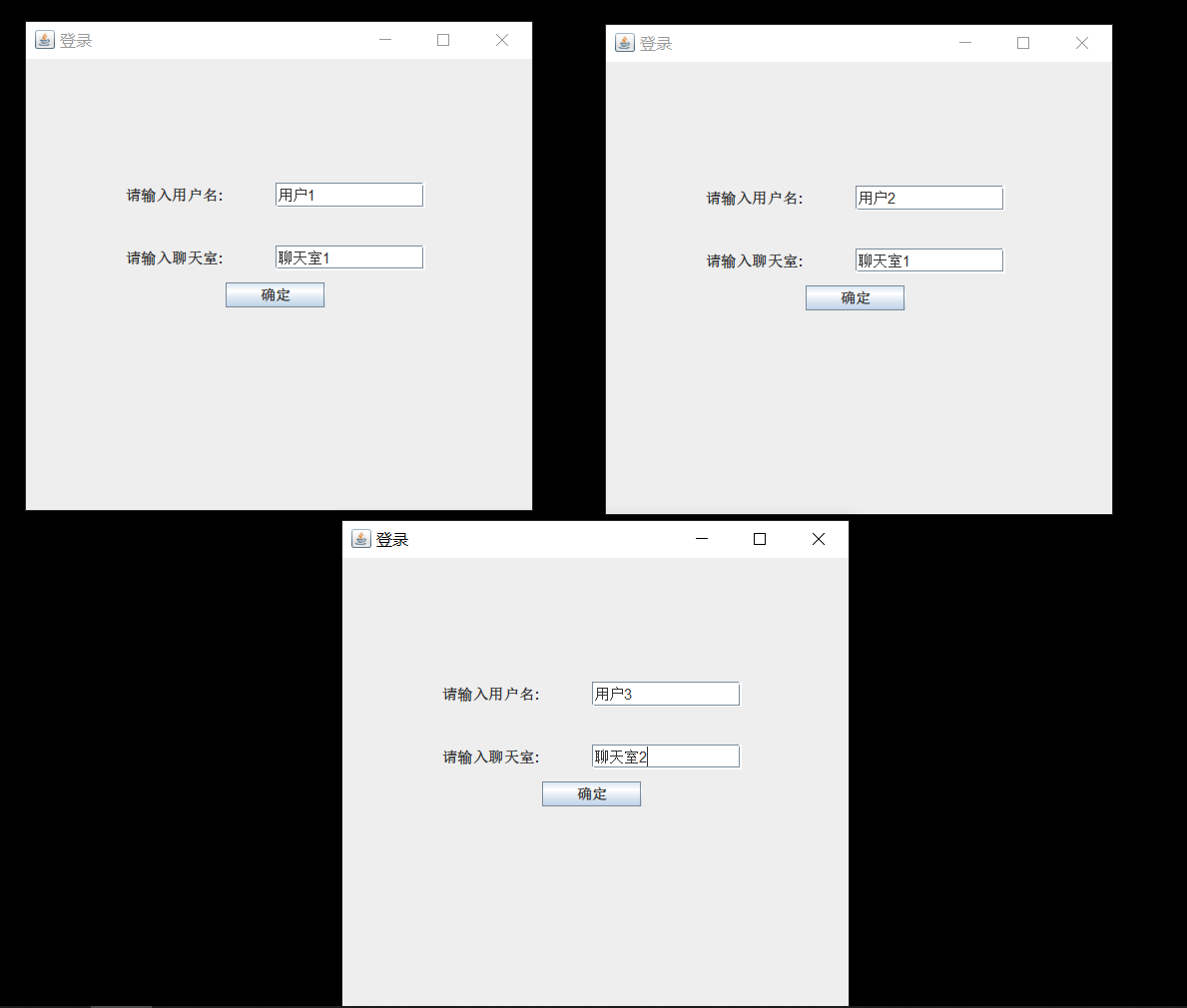


图5.1 用户登录界面

### 2、聊天界面

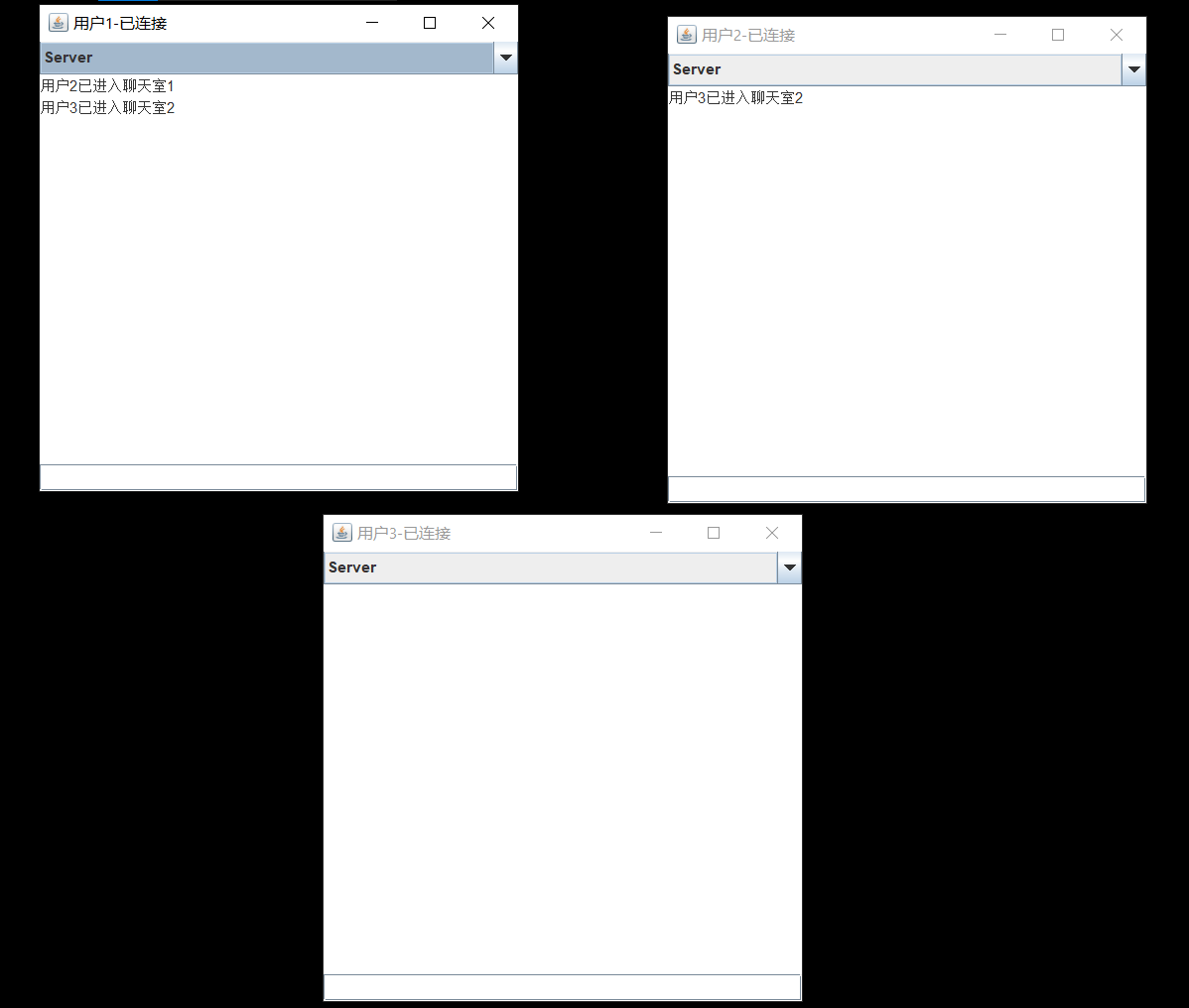


图5.2 聊天界面及聊天内容

### 3、实现广播、组播和点对点

1、看见用户1向Server发出信息实现了广播，聊天内容显示用户1同时向用户2、3发送了信息。

2、用户2实现了组播，只用与用户2同一聊天室内的用户1收到了信息，而在另一聊天室的用户3并未收到信息

3、用户3实现了点对点通信，用户3指定向用户1发送信息，只有用户1收到了信息。

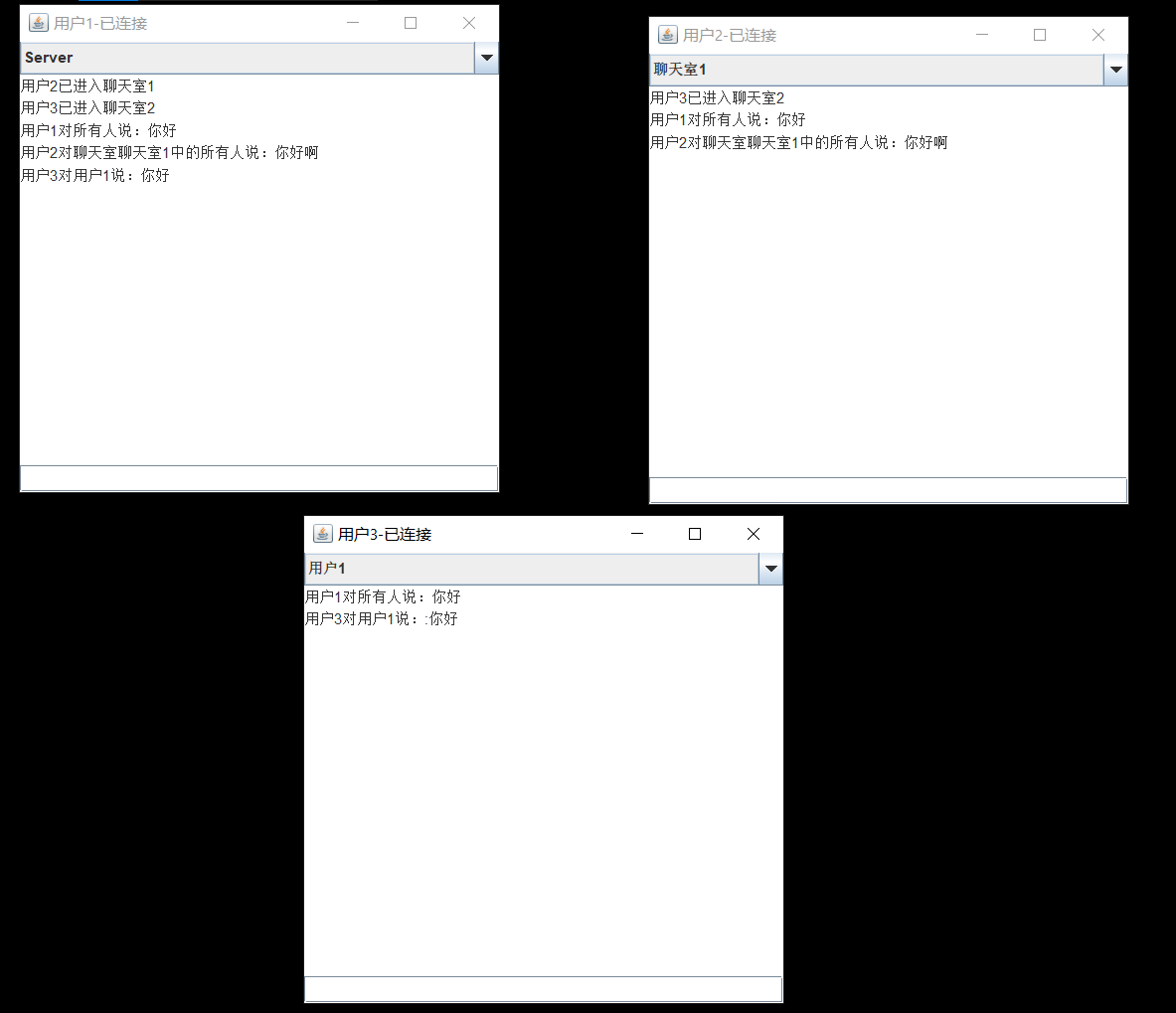


图5.3 聊天内容

### 4、实现用户离开提醒

用户1离开聊天室，其他用户收到了提醒，不能再对其进行点对点通信

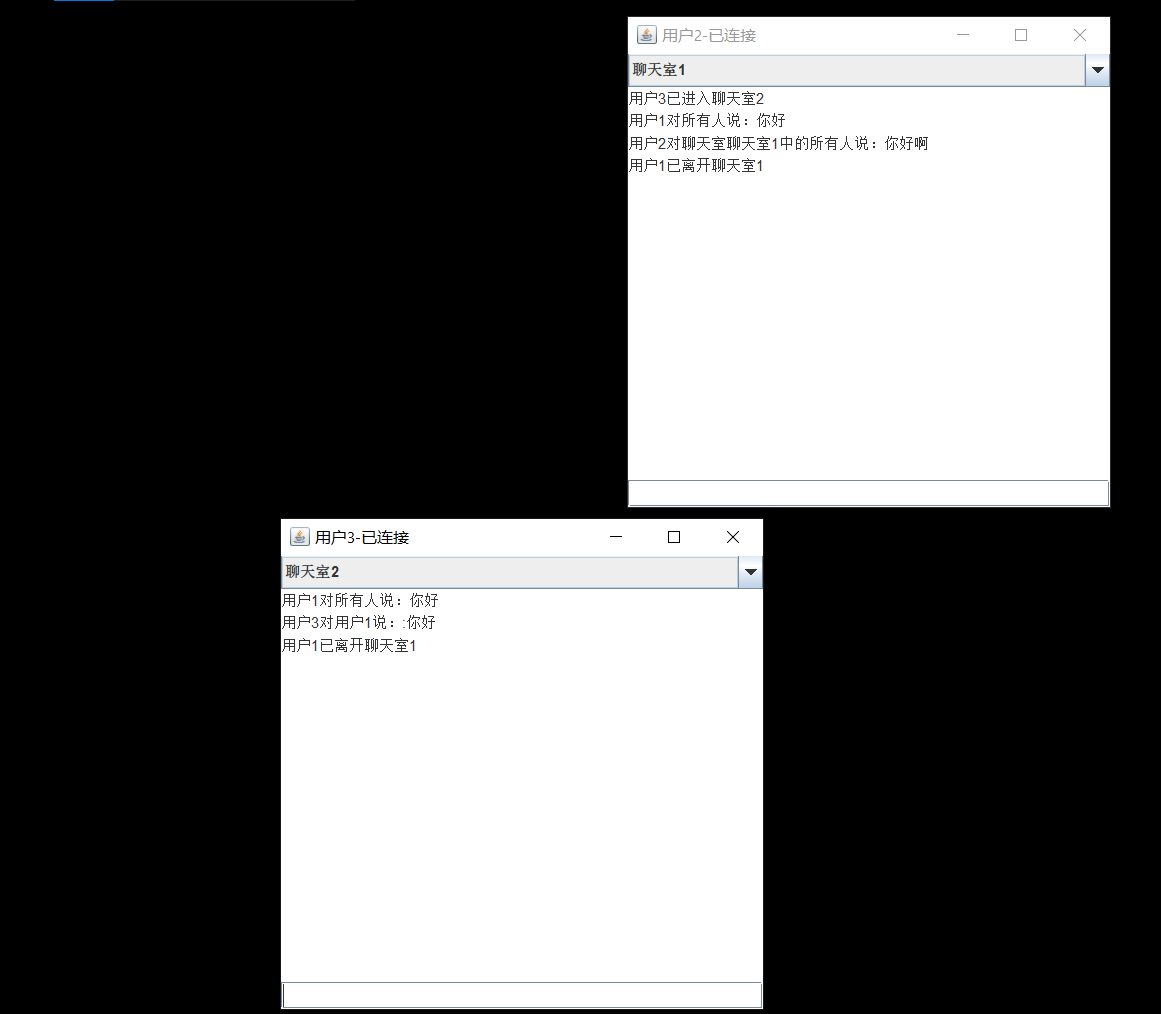


图5.4 用户1离开了聊天室

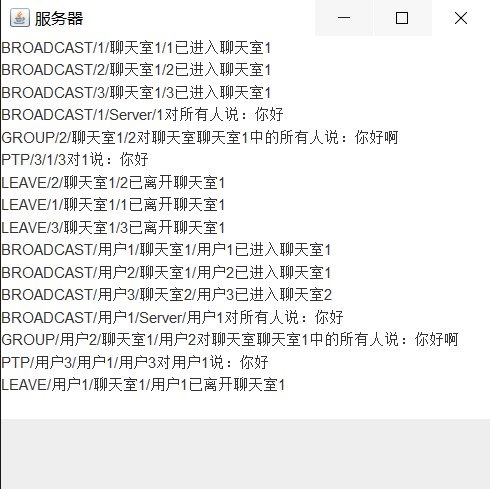


图5.5 聊天过程中服务器收到的信息

# 六、实验心得体会

本次实验我通过TCP协议、和java网络编程的相关知识实现了简单的聊天室程序的模拟，使我对C/S结构有了更深入的理解，也对网络程序的原理的理解更加深刻。当然，本次只是实现了对聊天室程序进行模拟，对于一些现实中可能存在的问题并没有进行解决，如存储用户聊天信息和完成更多更实用的功能等等。因此，再下一步的学习中，我也应该把所学的知识更多地去面向现实，面向实践，因此，下一步，我也应该向着这个方向继续不断改进和努力。