



中南大学  
CENTRAL SOUTH UNIVERSITY

# 计算机网络 实验报告

B1:编写一个基于 socket 的简易聊天程序

学 院: 计算机学院

专业班级: \_\_\_\_\_

学生姓名: \_\_\_\_\_

学 号: \_\_\_\_\_

指导教师: \_\_\_\_\_

年 月 日

# 目录

一、实验目的和要求 .....	1
二、实验内容与实现原理 .....	1
三、实验具体设计实现及结果 .....	3
UML.....	3
关键代码说明.....	7
运行结果.....	17
四、实验设备与实验环境 .....	20
五、实验总结 .....	20
六、附录：源代码 .....	21

# 一、实验目的和要求

## 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；
4. 要求本机既是客户端又是服务器端；

## 要求

所编写的程序应具有如下功能：

- a) 具有点对点通信功能，任意客户端之间能够发送消息；
- b) 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；
- c) 具有广播功能，客户端能够向所有其他成员广播消息。

# 二、实验内容与实现原理

## 实验内容

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：

1. 具有点对点通信功能，任意客户端之间能够发送消息；
2. 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；
3. 具有广播功能，客户端能够向所有其他成员广播消息；

## 实现原理

基于 Socket 的简易聊天程序的实现原理如下：

服务器端：

1. 创建一个 ServerSocket 对象，并指定一个端口号。
2. 使用 ServerSocket 的 accept() 方法监听客户端的连接请求，当有客户端连接时，accept() 方法返回一个 Socket 对象，代表与该客户端的通信套接字。

3. 使用 Socket 对象的输入流和输出流进行数据的读取和写入。
4. 在一个循环中，持续地接收来自客户端的消息，并将消息广播给所有连接的客户端。

#### 客户端：

1. 创建一个 Socket 对象，指定服务器的 IP 地址和端口号。
2. 使用 Socket 对象的输入流和输出流进行数据的读取和写入。
3. 在一个循环中，持续地读取用户在控制台输入的消息，并将消息发送给服务器。
4. 这样，服务器端和多个客户端之间建立了一个基于 Socket 的通信链路。
5. 服务器端通过监听客户端连接请求并创建对应的 Socket 对象，实现了与客户端的通信。客户端通过创建 Socket 对象连接到服务器，并与服务器进行通信。

在具体实现上，可以使用 Java 的 Socket 类和 ServerSocket 类来简化开发。服务器端和客户端分别通过 Socket 的输入流和输出流进行数据的读取和写入，实现消息的发送和接收。服务器端可以维护一个客户端列表，用于保存所有连接的客户端，从而实现消息的广播。

### 三、实验具体设计实现及结果

UML

Server.java

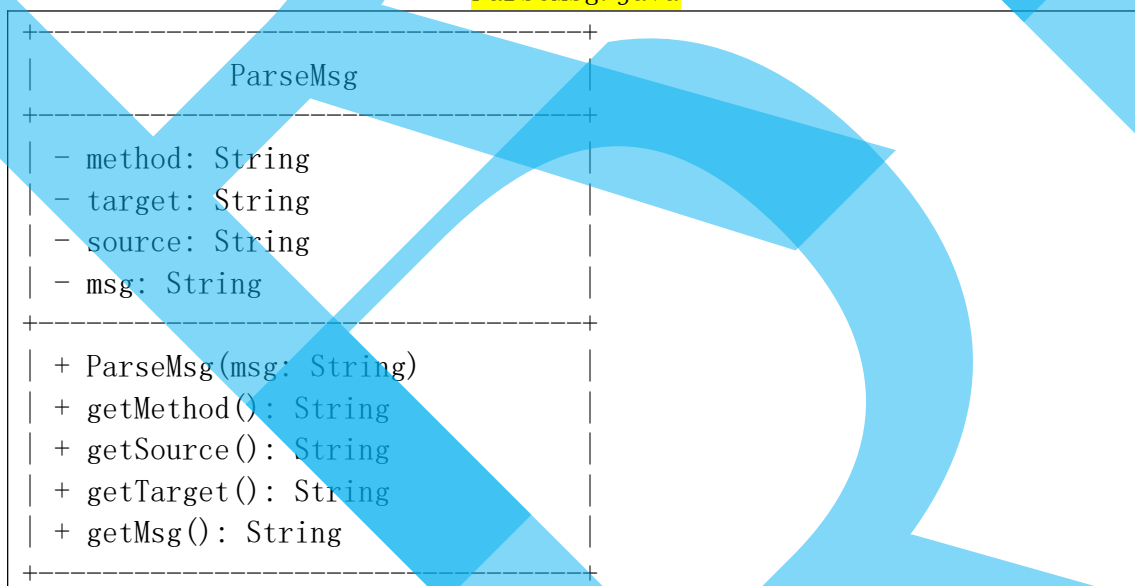


- **Server** 类继承自 `JFrame` 并实现了 `Runnable` 接口，表示服务器端的窗口和运行逻辑。
- **Server** 类的属性包括 `ss` (`ServerSocket` 对象用于监听连接请求)、`jta` (`JTextArea` 对象用于显示服务器日志)、`chatThread` (`ArrayList` 用于存储运行的用户线程)。
- **Server** 类的方法包括构造方法 `Server()`、`run()` (重写自 `Runnable` 接

口，用于多线程运行服务器逻辑）、sendMsg(msg: String)（根据消息解析并按不同方式发送消息）、broadcast(msg: String)（广播消息给所有客户端）、group(target: String, msg: String)（向指定群组发送消息）、p2p(target: String, msg: String)（点对点发送消息）。

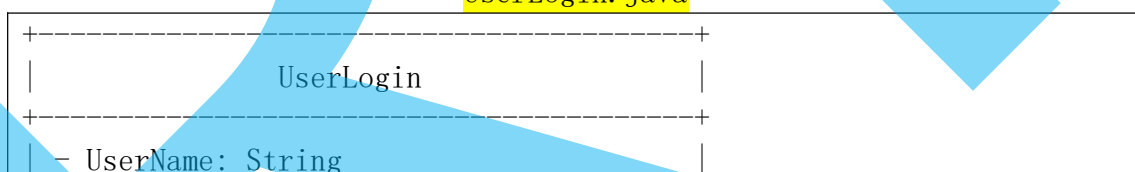
- ChatThread 是 Server 类的内部类，表示与客户端通信的线程。
- ChatThread 类的属性包括 userName（客户端的用户名）、chatRoom（客户端所属的聊天室名称）、socket（与客户端通信的套接字）、bufferedReader（从输入流中读取客户端发送的数据的 BufferedReader 对象）、printStream（向输出流发送数据的 PrintStream 对象）。
- ChatThread 类的方法包括构造方法 ChatThread(userName: String, chatRoom: String, socket: Socket)、run()（重写自 Thread 类，表示线程运行的内容）。

#### ParseMsg. java



- ParseMsg 类表示解析信息的类。
- ParseMsg 类的私有属性包括 method（消息的类型）、target（消息的目标方）、source（消息的发送方）、msg（消息的内容）。
- ParseMsg 类的公有方法包括构造方法 ParseMsg(msg: String)（用于解析消息并初始化属性值）、getMethod(): String（获取消息类型）、getSource(): String（获取消息发送方）、getTarget(): String（获取消息目标方）、getMsg(): String（获取消息内容）。

#### UserLogin. java



```

- chatRoom: String
- j11: JLabel
- jtf1: JTextField
- j12: JLabel
- jtf2: JTextField
- socket: Socket
- jb: JButton
- jle: JLabel
+-----+
+ UserLogin()
+ actionPerformed(e: ActionEvent): void
+ main(args: String[]): void
+-----+

```

- UserLogin 类表示用户登录界面类。
- UserLogin 类的私有属性包括 UserName（用户名）、chatRoom（聊天室名称）、j11（JLabel 对象，用于显示“请输入用户名:”）、jtf1（JTextField 对象，用于用户输入用户名）、j12（JLabel 对象，用于显示“请输入聊天室:”）、jtf2（JTextField 对象，用于用户输入聊天室名称）、socket（Socket 对象，用于连接到服务器）、jb（JButton 对象，用于确定按钮）、jle（JLabel 对象，用于显示错误信息）。
- UserLogin 类的公有方法包括构造方法 UserLogin()（用于创建用户登录界面）、actionPerformed(e: ActionEvent)（实现 ActionListener 接口的方法，处理用户的操作事件）、main(args: String[])（程序的入口点）。

#### Client.java

```

Client
+-----+
- userName: String
- chatRoom: String
- socket: Socket
- ps: PrintStream
- bf: BufferedReader
- jtf: JTextField
- jta: JTextArea
- jcb: JComboBox<String>
+-----+
+ Client(userName: String, chatRoom: String, socket: Socket)
+ run(): void
+ sendmsg(msg: String): void
+ actionPerformed(e: ActionEvent): void
+ windowActivated(e: WindowEvent): void

```

+ windowClosing(e: WindowEvent): void	
+ windowClosed(e: WindowEvent): void	
+ windowIconified(e: WindowEvent): void	
+ windowOpened(e: WindowEvent): void	
+ windowDeiconified(e: WindowEvent): void	
+ windowDeactivated(e: WindowEvent): void	
+-----+	

- Client 类表示用户类。
- Client 类的私有属性包括 userName（客户端的用户名）、chatRoom（客户端所属的聊天室名称）、socket（与服务器建立的套接字）、ps（用于向服务器发送数据的 PrintStream 对象）、bf（从服务器接收数据的 BufferedReader 对象）、jtf（用于输入要发送的消息的 JTextField 对象）、jta（用于显示接收到的消息的 JTextArea 对象）、jcb（用于选择消息目标的 JComboBox<String> 对象）。
- Client 类的公有方法包括构造方法 Client(userName: String, chatRoom: String, socket: Socket)（用于初始化类的成员变量并创建用户界面）、run(): void（线程运行内容）、sendmsg(msg: String): void（根据发送方法发送信息）、actionPerformed(e: ActionEvent): void（处理用户在文本框中输入并按下回车键的操作）、windowActivated(e: WindowEvent): void、windowClosing(e: WindowEvent): void、windowClosed(e: WindowEvent): void、windowIconified(e: WindowEvent): void、windowOpened(e: WindowEvent): void、windowDeiconified(e: WindowEvent): void、windowDeactivated(e: WindowEvent): void（窗口监听器的实现方法）。



## 关键代码说明

Server.java

### 定义线程运行内容

```
// 重写 run 方法实现多线程
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())); //将字节流转化为字符流，
再放入 BufferedReader

            String msg;
            while ((msg = bufferedReader.readLine()) != null)
                ;
            jta.append(msg + "\n"); // 将消息追加到服务器日志文本
区域 jta 中

            sendMsg(msg);
            ParseMsg parseMsg = new ParseMsg(msg); // 解析 msg
            ChatThread tmp = new ChatThread(parseMsg.getSource(),
parseMsg.getTarget(), socket); // 根据用户名、聊天室名称、套接字创建用
户聊天线程

            tmp.start(); // 开启此线程

            PrintStream printStream = new
PrintStream(socket.getOutputStream()); // 创建一个 PrintStream 对象，
使用套接字的输出流作为输出目标。这样，printStream 就可以通过 print()、
println() 等方法将数据发送到客户端。

            msg = ("UserName/");
            for (ChatThread i : chatThread) {
                // 遍历每个 ChatThread 对象，将每个 ChatThread 对象
的用户名 i.userName 追加到 msg 中，并在每个用户名后面添加一个斜杠"/"，用
于分隔不同的用户名

                // 形成一个以 "UserName/" 开头的消息字符串。这个消
息字符串用于向客户端发送服务器的用户情况，以告知客户端当前连接到服务器
的用户名列表。

                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

```

// 根据消息解析并按不同方式发送
public void sendMsg(String msg) {
    ParseMsg parseMsg = new ParseMsg(msg); // 自创 ParseMsg 对象，
    解析 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("p2p")) {
        p2p(parseMsg.getTarget(), parseMsg.getMsg());
    }
    // 用户退出
    } else if (parseMsg.getMethod().equals("LEAVE")) {
        for (ChatThread i : chatThread) {
            if (parseMsg.getSource().equals(i.userName)) {
                i.interrupt(); // 中断线程
                chatThread.remove(i); // 从运行列表中移出
                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();
                }
            }
        }
        broadcast(parseMsg.getMsg()); // 广播用户离开信息
    }
}

// 广播
public void broadcast(String msg) {
    for (ChatThread i : chatThread) {
        // 遍历存储在 chatThread 列表中的每个 ChatThread 对象
    }
}

```

```

        // 对于每个 ChatThread 对象 i，通过其关联的 PrintStream 对象的 println() 方法，向客户端发送消息 msg
        i.printStream.println(msg);
    }
}

// 群组
public void group(String target, String msg) {
    for (ChatThread i : chatThread) {
        // 对于每个 ChatThread 对象 i，检查它的 chatRoom 属性是否与目标群组 target 相匹配
        // 如果 i 的 chatRoom 属性与目标群组 target 相匹配，那么通过 i 对应的 PrintStream 对象的 println() 方法，向客户端发送消息 msg
        if (i.chatRoom.equals(target))
            i.printStream.println(msg);
    }
}

// 点对点
public void p2p(String target, String msg) {
    for (ChatThread i : chatThread) {
        if (i.userName.equals(target)) {
            i.printStream.println(msg);
            break;
        }
    }
}
}

```

### 聊天线程类(与客户端的通信线程)

```

class ChatThread extends Thread {
    String userName = null; // 客户端的用户名
    String chatRoom = null; // 客户端所属的聊天室名称
    Socket socket = null; // 与客户端进行通信的套接字
    BufferedReader bufferedReader = null; // 用于从套接字的输入流中读取客户端发送的数据
    PrintStream printStream = null; // 用于向套接字的输出流发送数据

    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();
        }
    }

    // 重写 run 方法，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(); // 从输入
                流中读取一行消息，并将其存储在变量 msg 中
                if (msg != null && !msg.equals("")) {
                    jta.append(msg + "\n"); // 将读取到的消息追加
                    到服务器的日志文本区域 (jta) 中
                    sendMsg(msg);
                }
            } catch (Exception e) {
            }
        }
    }
}

```

ChatThread 类具有以下主要功能：

1. 在构造函数中，通过接收客户端的用户名、聊天室名称和套接字作为参数，初始化类的成员变量，并创建与客户端通信所需的输入流和输出流。
2. 重写了 run() 方法，定义了线程的运行内容。在线程运行期间，通过不断从输入流中读取客户端发送的消息，并根据读取到的消息进行相应的处理和分发。同时，将读取到的消息追加到服务器的日志文本区域中。
3. 提供了发送消息的能力，通过关联的输出流将消息发送给客户端。

ChatThread 类的实例代表与一个客户端的通信线程。它负责接收客户端发送的消息，处理和分发这些消息，同时也负责向客户端发送服务器的响应。通过创建多个 ChatThread 实例，服务器能够同时与多个客户端进行并发的通信。

用于解析消息：

```
public class ParseMsg
{
    private String method; //消息的类型
    private String target=null; // 消息的目标方
    private String source=null; // 消息的发送方
    private String msg=null; // 消息的内容

    public ParseMsg(String msg/* 要解析的字符串 msg */)
    {
        this.method=msg.split("/")[0]; // 将字符串 msg 使用 / 分割，并
        将分割后的第一个部分（索引为 0）赋值给成员变量 method，表示消息的类型
        或方法。
        this.source=msg.split("/")[1]; // 将字符串 msg 使用 / 分割，并
        将分割后的第二个部分（索引为 1）赋值给成员变量 source，表示消息的发送
        方。
        this.target=msg.split("/")[2]; // 将字符串 msg 使用 / 分割，并
        将分割后的第三个部分（索引为 2）赋值给成员变量 target，表示消息的目标
        方。
        this.msg=msg.split(msg.split("/")[2]+"/")[1]; // 使用 / 分割
        字符串 msg，然后使用分割后的第三个部分（索引为 2）拼接成一个子字符串，
        再将该子字符串使用 / 分割，并取分割后的第二个部分（索引为 1），最后将
        该部分赋值给成员变量 msg，表示消息的内容。
        System.out.println(msg+" "+this.getMsg());
        /* 例
           输入：BROADCAST/User1/Group1/Hello, everyone!
           输出：
           内容：Hello, everyone!
           类型：BROADCAST
           发送方：User1
           目标方：Group1
        */
    }

    // 获取信息类型
    public String getMethod()
    {
        return this.method;
    }

    // 获取信息发送方
    public String getSource()
```

```

    {
        return this.source;
    }
    // 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;
    }
}

```

UserLogin.java

### 构造函数

```

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);
}

```

### 获取输入的用户名和聊天室名称

```

public void actionPerformed(ActionEvent e) { // 实现 ActionListener

```

接口的方法，用于处理用户的操作事件。

```
// 获取用户输入的用户名和聊天室名称
this.UserName = jtf1.getText();
this.chatRoom = jtf2.getText();
// 检查输入是否有效，即用户名和聊天室名称都不为空。如果输入有效，创建套接字并连接到服务器。
if (!this.UserName.equals("") && !this.chatRoom.equals("")) {
    try {
        // 创建套接字并连接到服务器
        socket = new Socket("127.0.0.1", 8080);
        // 登录界面关闭窗口
        this.dispose();
        // 实例化用户类，以启动客户端
        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!");
}
}
```

### Client.java

#### 构造函数

```
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())); // 将套接字的输入流转换为字符
    流，用于从服务器接收数据
    ps = new PrintStream(socket.getOutputStream()); // 将套接字的输出流用
    于向服务器发送数据
    ps.println("BROADCAST/" + userName + "/" + chatRoom + "/" + userName
```

```

+"已进入" + chatRoom + "\n"); // 向服务器发送一条消息，通知服务器当前
用户已进入聊天室
    jtf.addActionListener(this); // 事件监听器
    this.addWindowListener(this); // 窗口监听器
    new Thread(this).start(); // 启动线程
}

```

这段代码的作用是初始化客户端界面，包括设置窗口标题、大小和布局，添加用户界面元素（文本框、文本区域、下拉列表框），创建输入流和输出流用于与服务器通信，并向服务器发送用户进入聊天室的消息。同时，注册事件监听器和窗口监听器，以便处理用户交互和窗口关闭事件。最后，创建一个新的线程来接收从服务器接收的消息。

### 定义线程运行内容

```

public void run() {
    while (true) {
        try {
            String msg = bf.readLine();

            // case1 新用户进入聊天室。检查消息中是否包含“已进入”，
            即检查是否有新用户加入聊天室。
            if (msg.indexOf("已进入") != -1) {
                jcb.addItem(msg.split("已进入")[0]); // 将新加入
                的用户的用户名添加到下拉列表框 jcb 中
                jta.append(msg + "\n"); // 将接收到的消息追加到文
                本区域 jta 中
            }
            // case2 用户离开聊天室。检查消息中是否包含“已离开”，
            即检查是否有用户离开聊天室。
            else if (msg.indexOf("已离开") != -1) {
                jcb.removeItem(msg.split("已离开")[0]);
                jta.append(msg + "\n");
            }
            // case3 接收已经在聊天室的用户信息。检查消息是否以
            "UserName"开头，即接收已经在聊天室的用户信息。
            else if (msg.startsWith("UserName")) {
                String[] parse = msg.split("/"); // 将消息按照 "/"
                进行拆分，并存储到 parse 数组中。
                for (int i = 1; i < parse.length; i++) {
                    jcb.addItem(parse[i]); // 将遍历到的用户名添
                    加到下拉列表框 jcb 中，即添加已经在聊天室的其他用户。
                }
            }
        }
    }
}

```



```

        // case4 如果以上条件都不满足，即接收到的消息为普通的
        聊天消息。
        else {
            jta.append(msg + "\n"); // 将接收到的消息追加到文
            本区域 jta 中
        }
    } catch (Exception e) {
    }
}

```

### 根据发送方法发送信息

```

public void sendmsg(String msg) {
    // 如果是广播。通过输出流 ps 向服务器发送广播消息。消息格式为
    "BROADCAST/用户名/Server/消息内容"
    if (jcb.getSelectedItem().equals("Server")) {
        ps.println("BROADCAST/" + userName + "/Server/" + msg);
    }
    // 如果是组播。通过输出流 ps 向聊天室中的其他成员发送群组消息。
    消息格式为 "GROUP/用户名/聊天室名称/消息内容"。
    else if (jcb.getSelectedItem().equals(this.chatRoom)) {
        ps.println("GROUP/" + userName + "/" + this.chatRoom + "/"
        + msg);
    }
    // 如果其他情况 (p2p)。通过输出流 ps 向选择的目标发送点对点消
    息。消息格式为 "p2p/用户名/目标用户名/消息内容"。
    else {
        ps.println("p2p/" + userName + "/" + jcb.getSelectedItem()
        + "/" + msg);
    }
}

```

### 确定发送方式

```

public void actionPerformed(ActionEvent e) { // actionPerformed
    方法是 ActionListener 接口的实现方法，用于处理用户在文本框中输入并按下
    回车键时的操作

    String msg = new String(this.userName + "对");
    /* (客户名) 对 */

    if (jcb.getSelectedItem().equals("Server")) {
        msg += "所有人说: ";
    }
}

```

```

        /* (客户名) 对所有人说: */
    } else if (jcb.getSelectedItem().equals(this.chatRoom)) {
        msg += "聊天室" + this.chatRoom + "中的所有人说: ";
        /* (客户名) 对聊天室 (聊天室名) 中的所有人说: */
    } else {
        msg += jcb.getSelectedItem() + "说: ";
        /* (客户名) 对 (客户 2 名) 说: */
    }

    if (!jcb.getSelectedItem().equals("Server")
        && !jcb.getSelectedItem().equals(this.chatRoom)) { // p2p
        jta.append(msg + ":" + jtf.getText() + "\n");
    }

    sendmsg(msg + jtf.getText() + "\n"); // 将拼接好的消息字符串和
    文本框中输入的内容发送给选择的目标
    jtf.setText("");

    /*
    actionPerformed 方法根据用户选择的目标和文本框中的输入内容，
    构造相应的消息字符串，
    并将该消息字符串发送给服务器或聊天室中的其他成员。
    然后，将消息内容追加到文本区域 jta 中供用户查看，并清空文本框
    的内容，以便下次输入。
    */
}

```

### 窗口关闭

```

public void windowClosing(WindowEvent e) { // windowClosing 方法是
WindowListener 接口的实现方法，用于处理窗口关闭事件。
    // 通过输出流 ps 向服务器发送用户离开的消息。消息格式为
    "LEAVE/用户名/聊天室名称/用户已离开聊天室"
    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);
}

```

## 运行结果

图 1: 运行 `Server.java`, 生成服务器窗口

图 2: 运行 `UserLogin.java`, 生成登录界面

The background of the page is a light gray with a large, abstract pattern of overlapping, semi-transparent blue shapes. These shapes include thick lines, circles, and polygons, creating a complex, modern geometric design.

图 3: 允许 UserLogin.java 有多个实例

图 4: 再次运行 UserLogin.java 并输入用户名和聊天室名，模拟两个用户进入聊天室

The background of the entire page is a complex, abstract pattern of overlapping, rounded rectangular shapes in various shades of blue. These shapes are arranged in a way that creates a sense of depth and movement, with some shapes appearing to be in front of others. The pattern is dense and covers the entire area of the page.

图 5：以此类推

图 6：聊天室效果

## 四、实验设备与实验环境

### 操作系统

Windows 11

### 开发工具

IntelliJ IDEA Community Edition 2023.1.2

### 编程语言

Java

## 五、实验总结

## 六、附录：源代码

### Server.java

```
import javax.swing.*;
import java.awt.*;
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;

// 继承自 JFrame 的服务器类来实现图形用户界面
public class Server extends JFrame implements Runnable {
    private ServerSocket ss = null; // ServerSocket 对象，用于监听连接请求
    private JTextArea jta = new JTextArea(); // JTextArea 对象，用于在界面上显示服务器日志
    private ArrayList<ChatThread> chatThread = new ArrayList<ChatThread>(); // ArrayList，用于存储正在运行的用户线程

    public Server() throws Exception {
        this.setTitle("服务器窗口");
        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();
        /* 构造函数 Server() 用于初始化服务器对象。
        * 它设置了窗口的标题、大小、默认关闭操作，并将日志文本区域添加到界面上。
        * 然后，创建一个 ServerSocket 对象并开始在 8080 端口上监听连接请求。
        * 最后，启动一个新的线程来运行服务器。 */
    }

    // 重写 run 方法实现多线程
    public void run() {
        while (true) {
            try {

                Socket socket = this.ss.accept();
                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); // ParseMsg 对象，解析 msg
                ChatThread tmp = new ChatThread(parseMsg.getSource(),
parseMsg.getTarget(), socket);
                tmp.start();

                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); // ParseMsg 对象，解析 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("p2p")) {
            p2p(parseMsg.getTarget(), parseMsg.getMsg());
            // 用户退出
        } else if (parseMsg.getMethod().equals("LEAVE")) {
            for (ChatThread i : chatThread) {
                if (parseMsg.getSource().equals(i.userName)) {
                    i.interrupt(); // 中断线程
                    chatThread.remove(i); // 从运行列表中移出
                    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();
                    }
                }
            }
            broadcast(parseMsg.getMsg()); // 广播用户离开信息
        }
    }

    // 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 p2p(String target, String msg) {
    for (ChatThread i : chatThread) {
        if (i.userName.equals(target)) {
            i.printStream.println(msg);
            break;
        }
    }
}

// 聊天线程类(与客户端的通信线程)
class ChatThread extends Thread {
    String userName = null; // 客户端的用户名
    String chatRoom = null; // 客户端所属的聊天室名称
    Socket socket = null; // 与客户端进行通信的套接字
    BufferedReader bufferedReader = null; // 用于从套接字的输入流中读取客户端发送的数据
    PrintStream printStream = null; // 用于向套接字的输出流发送数据

    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();
        }
    }

    // 重写 run 方法, 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) {
            }
        }
    }
}

```

```

    }
}

// main
public static void main(String[] args) throws Exception {
    new Server();
}
}

```

### ParseMsg. java

```

// 解析信息类
public class ParseMsg
{
    private String method; //消息的类型
    private String target=null; // 消息的目标方
    private String source=null; // 消息的发送方
    private String msg=null; // 消息的内容

    public ParseMsg(String msg/* 要解析的字符串 msg */)
    {
        this.method=msg.split("/") [0]; // 将字符串 msg 使用 / 分割, 并将分割后的第一个部分 (索引为 0) 赋值给成员变量 method, 表示消息的类型或方法。
        this.source=msg.split("/") [1]; // 将字符串 msg 使用 / 分割, 并将分割后的第二个部分 (索引为 1) 赋值给成员变量 source, 表示消息的发送方。
        this.target=msg.split("/") [2]; // 将字符串 msg 使用 / 分割, 并将分割后的第三个部分 (索引为 2) 赋值给成员变量 target, 表示消息的目标方。
        this.msg=msg.split(msg.split("/") [2] + "/" ) [1]; // 使用 / 分割字符串 msg, 然后使用分割后的第三个部分 (索引为 2) 拼接成一个子字符串, 再将该子字符串使用 / 分割, 并取分割后的第二个部分 (索引为 1), 最后将该部分赋值给成员变量 msg, 表示消息的内容。
        System.out.println(msg+" "+this.getMsg());
    }
    /* 例:

        输入: BROADCAST/User1/Group1/Hello, everyone!

        输出:
        内容: Hello, everyone!
        类型: BROADCAST
        发送方: User1
        目标方: Group1
    */
}

```

```

    }

    // 获取信息类型
    public String getMethod()
    {
        return this.method;
    }

    // 获取信息发送方
    public String getSource()
    {
        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;
    }
}

```

### UserLogin. java

```

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.Socket;

// 用户登录界面类 UserLogin, 用于用户输入用户名和聊天室名称进行登录。
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)` { // 实现 *ActionListener* 接口的方法，用于处理用户的操作事件。

// 获取用户输入的用户名和聊天室名称

`this.UserName = jtf1.getText();`

`this.chatRoom = jtf2.getText();`

// 检查输入是否有效，即用户名和聊天室名称都不为空。如果输入有效，创建套接字并连接到服务器。

`if (!this.UserName.equals("") && !this.chatRoom.equals("")) {`

`try {`

// 创建套接字并连接到服务器

`socket = new Socket("127.0.0.1", 8080);`

// 登录界面关闭窗口

`this.dispose();`

// 实例化用户类，以启动客户端

`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 javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;

// 用户类
public class Client extends JFrame implements Runnable, ActionListener, WindowListener
{
    private String userName; // 客户端的用户名
    private String chatRoom; // 客户端所属的聊天室名称
    private Socket socket; // 用于与服务器建立连接
    private PrintStream ps = null; // 用于向服务器发送数据
    private BufferedReader bf = null; // 用于从服务器接收数据
    private JTextField jtf = new JTextField(); // 声明一个文本框 jtf, 用于输入要发送的消息
    private JTextArea jta = new JTextArea(); // 声明一个文本区域 jta, 用于显示接收到的消息
    private JComboBox<String> jcb = new JComboBox<String>(new String[] { "Server" });
    // 声明一个下拉列表框 jcb, 用于选择消息的目标 (发送给服务器还是聊天室中的其他成员)

    // 构造函数用于初始化类的成员变量, 并创建用户界面
    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())); //
        将套接字的输入流转换为字符流，用于从服务器接收数据
        ps = new PrintStream(socket.getOutputStream()); // 将套接字的输出流用于向服
        务器发送数据
        ps.println("BROADCAST/" + userName + "/" + chatRoom + "/" + userName + "已进
        入" + chatRoom + "\n"); // 向服务器发送一条消息，通知服务器当前用户已进入聊天室
        jtf.addActionListener(this); // 事件监听器
        this.addWindowListener(this); // 窗口监听器
        new Thread(this).start();
    }

    // 线程运行内容
    public void run() {
        while (true) {
            try {
                String msg = bf.readLine();

                // case1 新用户进入聊天室。检查消息中是否包含“已进入”，即检查是否有
                新用户加入聊天室。
                if (msg.indexOf("已进入") != -1) {
                    jcb.addItem(msg.split("已进入")[0]); // 将新加入的用户的用户名
                    添加到下拉列表框 jcb 中
                    jta.append(msg + "\n"); // 将接收到的消息追加到文本区域 jta 中
                }
                // case2 用户离开聊天室。检查消息中是否包含“已离开”，即检查是否有用
                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");
                }
                // case3 接收已经在聊天室的用户信息。检查消息是否以 "UserName" 开头，
                即接收已经在聊天室的用户信息。
                else if (msg.startsWith("UserName")) {
                    String[] parse = msg.split("/"); // 将消息按照 "/" 进行拆分，并
                    存储到 parse 数组中。
                    for (int i = 1; i < parse.length; i++) {
                        jcb.addItem(parse[i]); // 将遍历到的用户名添加到下拉列表框
                        jcb 中，即添加已经在聊天室的其他用户。
                    }
                }
            }
        }
    }

```

```

    }
    // case4 如果以上条件都不满足，即接收到的消息为普通的聊天消息。
    else {
        jta.append(msg + "\n"); // 将接收到的消息追加到文本区域 jta 中
    }

} catch (Exception e) {
}

}

// 根据发送方法发送信息（根据下拉列表框的内容确定发送方式：广播消息、群组消息还是点对点消息。）
public void sendmsg(String msg) {
    // 如果是广播。通过输出流 ps 向服务器发送广播消息。消息格式为 "BROADCAST/用户名/Server/消息内容"
    if (jcb.getSelectedItem().equals("Server")) {
        ps.println("BROADCAST/" + userName + "/Server/" + msg);
    }
    // 如果是组播。通过输出流 ps 向聊天室中的其他成员发送群组消息。消息格式为 "GROUP/用户名/聊天室名称/消息内容"。
    else if (jcb.getSelectedItem().equals(this.chatRoom)) {
        ps.println("GROUP/" + userName + "/" + this.chatRoom + "/" + msg);
    }
    // 如果其他情况（p2p）。通过输出流 ps 向选择的目标发送点对点消息。消息格式为 "p2p/用户名/目标用户名/消息内容"。
    else {
        ps.println("p2p/" + userName + "/" + jcb.getSelectedItem() + "/" + msg);
    }
}

// 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) { // actionPerformed 方法是 ActionListener 接口的实现方法，用于处理用户在文本框中输入并按下回车键时的操作

    String msg = new String(this.userName + "对");
    /* （客户名）对 */

    if (jcb.getSelectedItem().equals("Server")) {
        msg += "所有人说： ";
        /* （客户名）对所有人说： */
    } else if (jcb.getSelectedItem().equals(this.chatRoom)) {
        msg += "聊天室" + this.chatRoom + "中的所有人说： ";
        /* （客户名）对聊天室（聊天室名）中的所有人说： */
    }
}

```



```

    } else {
        msg += jcb.getSelectedItemAt() + "说: ";
        /* (客户名) 对 (客户 2 名) 说: */
    }

    if (!jcb.getSelectedItemAt().equals("Server")
    && !jcb.getSelectedItemAt().equals(this.chatRoom)) { // p2p
        jta.append(msg + ":" + jtf.getText() + "\n");
    }

    sendmsg(msg + jtf.getText() + "\n"); // 将拼接好的消息字符串和文本框中输入的
    内容发送给选择的目标
    jtf.setText("");

    /*
    actionPerformed 方法根据用户选择的目标和文本框中的输入内容，构造相应的消息
    字符串，
    并将该消息字符串发送给服务器或聊天室中的其他成员。
    然后，将消息内容追加到文本区域 jta 中供用户查看，并清空文本框的内容，以便下
    次输入。
    */
}

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) { // windowClosing 方法是 WindowListener
    接口的实现方法，用于处理窗口关闭事件。
    // 通过输出流 ps 向服务器发送用户离开的消息。消息格式为 "LEAVE/用户名/聊天室
    名称/用户已离开聊天室"
    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) {  
  
}  
  
}
```