**重庆大学Project报告**

Project题目： 基于socket的多人聊天室

学 院： 计算机学院

专业班级： 计算机科学与技术6班

计算机科学与技术4班

信息安全2班

年 级： 2020级

姓 名： 蒲佳嘉、陈沐阳、杨佳慧

学 号： 20204263、20204345、20204358

完成时间： 2022 年 12 月 11 日

成 绩：

指导教师： 陈自郁

重庆大学教务处制

Project教师评定成绩表

A. 课程设计的成绩构成

|  |  |  |  |
| --- | --- | --- | --- |
| 课程设计项目 | 该项目标准分值 | 评定分值（100分） | |
| 项目完成质量（含检查） | 项目报告 |
| **多人聊天系统** | 10 | 80% | 20% |

B. 课程设计项目的成绩评定标准

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | 得分情况 | | | | |
| 优秀  (90～100) | 良好  (80～89) | 中等  (70～79 | 及格  60～69 | 不及格  0～59 |
| 参考标准 | 参考标准 | 参考标准 | 参考标准 | 参考标准 |
| 项目完成质量 | 1.分工明确，独立完成课程设计项目，有源码  2.设计合理，功能完善，满足课程设计全部要求，且有拓展  3.界面友好，Bug极少，针对异常情况有处理  4.汇报检查时，讲解清晰，演示流畅，能正确回答问题 | 1.分工明确，独立完成课程设计项目，有源码  2.设计合理，功能完善，满足课程设计全部要求  3.界面良好，Bug较少，针对异常情况有处理  4.汇报检查时，讲解清晰，演示流畅，能正确回答问题 | 1.独立完成课程设计项目，有源码  2.功能较完善，基本满足课程设计要求  3.界面一般，Bug较多，无异常情况有处理  4.汇报检查时，讲解较清晰，完成演示，基本能正确回答问题 | 1.独立完成课程设计项目，有源码  2.功能基本完成，基本满足课程设计要求  3.界面一般，Bug很多，无异常情况有处理  4.汇报检查时，能基本讲清楚，主要功能能演示，基本能正确回答问题 | 1.抄袭或被抄袭以0分计，没有参与课程设计检查以0分计；  2.未能独立完成课程设计项目；  3.不能通过检查，不能进行课程设计汇报，主要功能无法演示，不能正确回答问题。 |
| 项目报告 | 1.结构严谨，逻辑清晰  2.文档规范，文字、图表表达清楚，用词专业，完全符合规范化要求  3.计算机打印  4.有完整的项目设计方案，采用的理论技术合理正确，结果及分析正确，功能有拓展  5.报告内容与课程设计检查结果相符 | 1.结构合理，符合逻辑；  2. 文档规范，文字、图表表达较清楚，用词专业，符合规范化要求  3.计算机打印  4.有较完整的项目设计方案，采用的理论与技术正确，结果及分析正确  5.报告内容与课程设计检查结果相符 | 1.结构合理，符合逻辑  2. 文档较规范，文字、图表表达较清楚，基本达到规范化要求  3.计算机打印  4.项目设计方案基本完整，采用的理论与技术基本正确，结果及分析基本正确。  5.报告内容与课程设计检查结果相符 | 1.结构基本合理,逻辑基本清楚  2. 文档较规范，文字、图表表达较清楚，勉强达到规范要求  3.计算机打印  4.项目设计方案有欠缺，采用的理论与技术基本可行，没有测试结果与结果分析  5.报告内容与课程设计检查结果相符 | 1.抄袭与被抄袭以0分计；  2.结构混乱，内容空泛，文字表达不清，错别字较多，达不到规范化要求  3.人工书写或计算机打印  4.报告内容与课程设计检查结果不相符 |

说明：评定分值要折算成项目标准分值。课程设计项目得分 = 课程设计项目的评定分值\*10%

指导教师评定成绩：

指导教师签名： 年 月 日

课题任务分工说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 角色 | 承担的任务 |
| 1 | 刘静婷 |  | 组长 | （1）实现共享功能  （2）实现私聊功能  （3）撰写课程报告  （4）协调团队工作 |
| 2 | 皮悦颖 |  | 组员 | （1）实现文字传输功能  （2）实现群聊功能  （3）撰写课程报告 |
| 3 | 覃倩瑶 |  | 组员 | （1）实现登录退出  （2）GUI图形界面  （3）撰写课程报告 |
| 4 | 邓茜 |  | 组员 | （1）实现文件传输功能  （2）实现图片传输功能  （3）实现用户列表功能  （4）撰写课程报告 |

目录

[1 需求分析 6](#_Toc17071)

[1.1 功能需求 6](#_Toc27600)

[1.2 技术需求 6](#_Toc14224)

[2 结构设计 6](#_Toc19160)

[2.1 结构设计分析 6](#_Toc2453)

[2.1.1 socket通信的主要步骤 6](#_Toc3856)

[2.1.2 Socket连接 7](#_Toc30517)

[2.1.3TCP三次握手协议 8](#_Toc23544)

[2.1.4GUI部分 8](#_Toc331)

[2.1.5多线程 9](#_Toc21739)

[2.1.6核心框架 9](#_Toc5105)

[2.1.7重要类： 9](#_Toc23332)

[2.2 工作流程和类间关系 11](#_Toc8939)

[2.2.1 工作流程 11](#_Toc13536)

[2.2.2 类间关系 12](#_Toc16081)

[3 关键代码（技术）描述 13](#_Toc11905)

[3.1 源文件文件树 13](#_Toc4116)

[3.2 Login 15](#_Toc3555)

[3.2.1 Login类的重要属性： 15](#_Toc20364)

[3.2.2 Login类的重要方法： 15](#_Toc12611)

[3.3 ChatRoom 16](#_Toc20541)

[3.3.1 ChatRoom类的重要属性： 16](#_Toc15969)

[3.3.1 ChatRoom类的重要方法： 16](#_Toc3182)

[3.4 ClientListener 20](#_Toc11213)

[3.4.1 ClientListener类重要属性： 20](#_Toc30793)

[3.4.2 ClientListener类重要方法： 20](#_Toc32126)

[3.5 MultiServer 22](#_Toc11926)

[3.5.1 MultiServer类重要属性： 22](#_Toc5913)

[3.5.2 MultiServer类重要方法： 23](#_Toc28763)

[3.6 ServerThread 23](#_Toc22945)

[3.6.1 ServerThread类重要属性： 23](#_Toc30305)

[3.6.2 ServerThread类重要方法： 23](#_Toc12123)

[3.7 ServerFileThread 24](#_Toc19435)

[3.7.1 ServerFileThread类重要属性： 24](#_Toc50)

[3.7.2 ServerFileThread类重要方法： 25](#_Toc25638)

[3.8 ClientFile 26](#_Toc6625)

[3.8.1 ClientFile类重要属性： 26](#_Toc26650)

[3.8.2 ClientFile类重要方法： 26](#_Toc12962)

[3.9 User 28](#_Toc26916)

[3.9.1 User类重要属性： 28](#_Toc8766)

[3.9.2 User类重要方法： 28](#_Toc17474)

[4测试运行 31](#_Toc3244)

[4.1 运行过程 31](#_Toc9400)

[4.1.1 登录 31](#_Toc21159)

[4.1.2 私聊 32](#_Toc18642)

[4.1.3 群聊 34](#_Toc32731)

[4.1.4 发送文件 34](#_Toc468)

[4.1.5 发送图片 35](#_Toc27549)

[4.1.6 退出 37](#_Toc28761)

[4.1.7 服务器端记录 37](#_Toc10317)

[4.1.8 数据库 38](#_Toc18604)

[5 总结 38](#_Toc32461)

# 1 需求分析

## 功能需求

1. 实现基于服务器转发的多个客户间的数据共享与交换。
2. 共享和交换的数据包括文字、图片和文件。
3. 服务器端记录当前在线客户列表，客户端能够获取该客户列表，并实时刷新。
4. 任一个客户可以和任意其它的客户进行交互。即支持一对一通信、组通信。所谓一对一通信是指，通信初始发起者可从在线客户列表中选择一个人进行一对一通信；所谓组通信，是指初始发起者可以从在线列表中选择多人（至少2人），实现彼此间的通信。

## 技术需求

1. 基于SOCKET编程实现。
2. 传输层可采用TCP或UDP协议。
3. 必须提供详细的应用协议设计。
4. 清晰地展示系统功能和关系。
5. 具有良好的可视化界面。

# 2 结构设计

## 2.1 结构设计分析

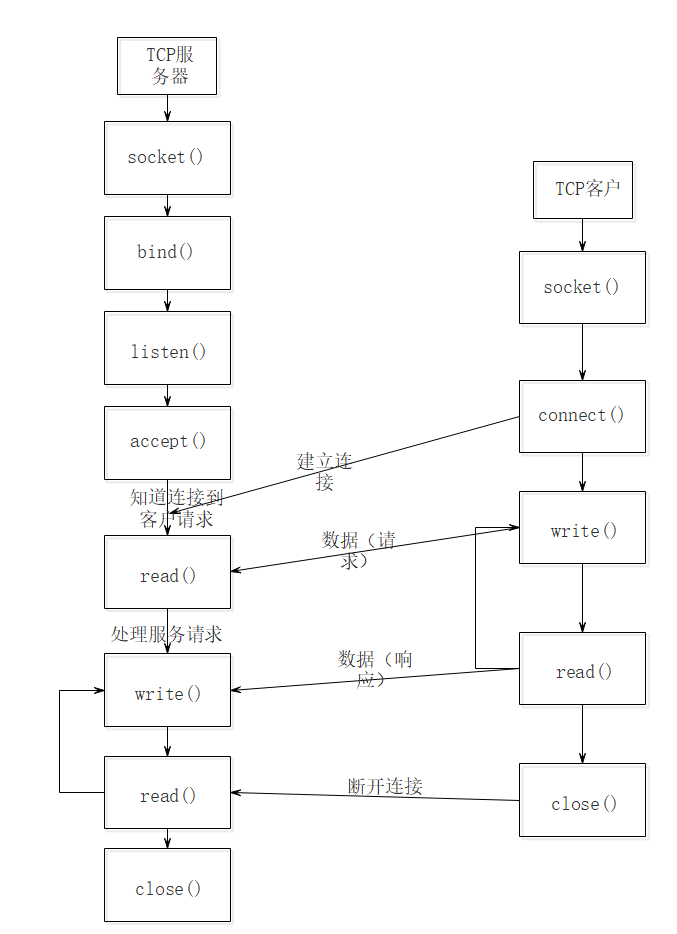
我们的project的三大核心部分分别为socket、GUI和多线程。

### 2.1.1 socket通信的主要步骤

* 1. 服务器根据地址类型（ipv4,ipv6）、socket类型、协议创建socket
  2. 服务器为socket绑定ip地址和端口号
  3. 服务器socket监听端口号请求，随时准备接收客户端发来的连接，这时候服务器的socket并没有被打开
  4. 客户端创建socket
  5. 客户端打开socket，根据服务器ip地址和端口号试图连接服务器socket
  6. 服务器socket接收到客户端socket请求，被动打开，开始接收客户端请求，直到客户端返回连接信息。这时候socket进入阻塞状态，所谓阻塞即accept()方法一直到客户端返回连接信息后才返回，开始接收下一个客户端谅解请求
  7. 客户端连接成功，向服务器发送连接状态信息
  8. 服务器accept方法返回，连接成功
  9. 客户端向socket写入信息
  10. 服务器读取信息
  11. 客户端关闭
  12. 服务器端关闭

### 2.1.2 Socket连接

在建立聊天通讯时，服务器端与客户端的连接是采用TCP套接字进行的，建立Socket连接至少需要一对套接字，其中一个运行于客户端，另一个运行于服务器端。



利用Socket建立网络连接的分为三个步骤：服务器监听，客户端请求，连接确认。

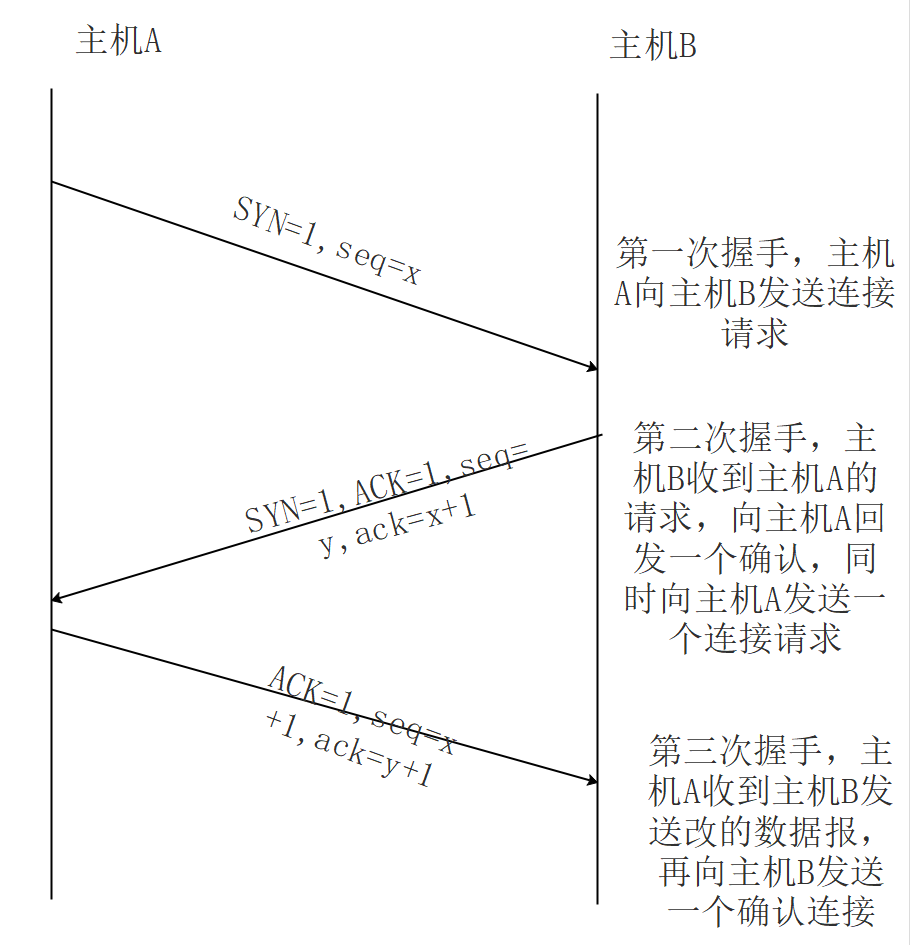
1. 服务器监听：服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。
2. 客户端请求：指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。

为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

1. 连接确认：当服务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，双方就正式建立连接。
2. 而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

在我们的project中，用户会先进行登录，成功登录被监听到后，才会在ClientListener类中的actionPerforme方法中创建socket.

### 2.1.3TCP三次握手协议



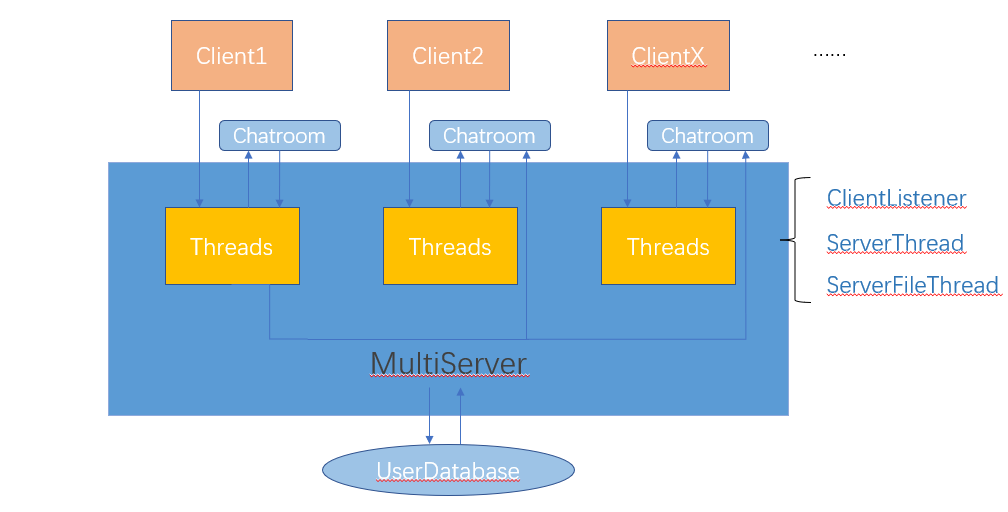
### 2.1.4GUI部分

主要用于绘制图形窗口，后续我们会给出核心代码，也会展示效果截图。

### 2.1.5多线程

我们采用多线程的方式。在服务端，每接收到一个连接时，就启动一个线程对象，并把当前客户端加入到定义好的列表List<socket>中以便后续使用，然后去执行相应的任务。我们把这个代码块可以写成Thread类。值得注意的是，因为在socket=serverSocket.accept(); 等待连接的时候是阻塞的状态，所以不会一直无限循环下去，会导致资源的浪费，所以要在服务端的accept() 的方法下改成while循环，不断的接收新连接的客户端。

### 2.1.6核心框架



### 2.1.7重要类：

**Login**：登录界面设计，输入用户名字，按下登录按钮即登录；设有监听器，通过ClientListener中的LoginListen实现Socket的初始化和连接服务器。

**ChatRoom**：客户端聊天界面，在文本框中输入聊天内容，点击发送即同步发送到各个客户端；点击文件按钮，可以选择发送文件/图片到各个客户端；还设有用户列表，当前用户可以点击选择列表中的用户名作为聊天对象，与之开启通信。

**ClientListener**：对各个界面设置监听类，以便实现多个界面信息的互通。登录界面监听：建立与服务器的联系，获取用户名字，将用户的登录信息发送给服务器，同时实现当登录按键按下时对登录界面进行隐藏。聊天界面监听：实现消息发送，当文本框内容为空的时候，弹出当输入为空的提示，并提醒重新输入；当输入内容不为空时，则将输出的发送消息的用户名称、文本内容、内容发送时间同时显示到聊天界面中；当聊天界面被关闭时，关闭对应客户端并在主程序中显示用户已退出的信息。

**MultiServer**：运用一个List对客户端进行储存，等待客户端的请求，如果建立了新的客户端就将客户端存放到List中，并对每次请求都建立一个ServerThread线程进行处理。

**ServerThread**：消息传输线程，在ServerThread的构造函数中接受一个socket，并启动线程。run()中获取客户端的信息，遍历List中的客户端，并用输入流和输出流将信息发送到每个客户端，同时输出到服务器主程序(服务器主程序直接使用命令行进行展示)。当线程关闭则移除相应的socket。

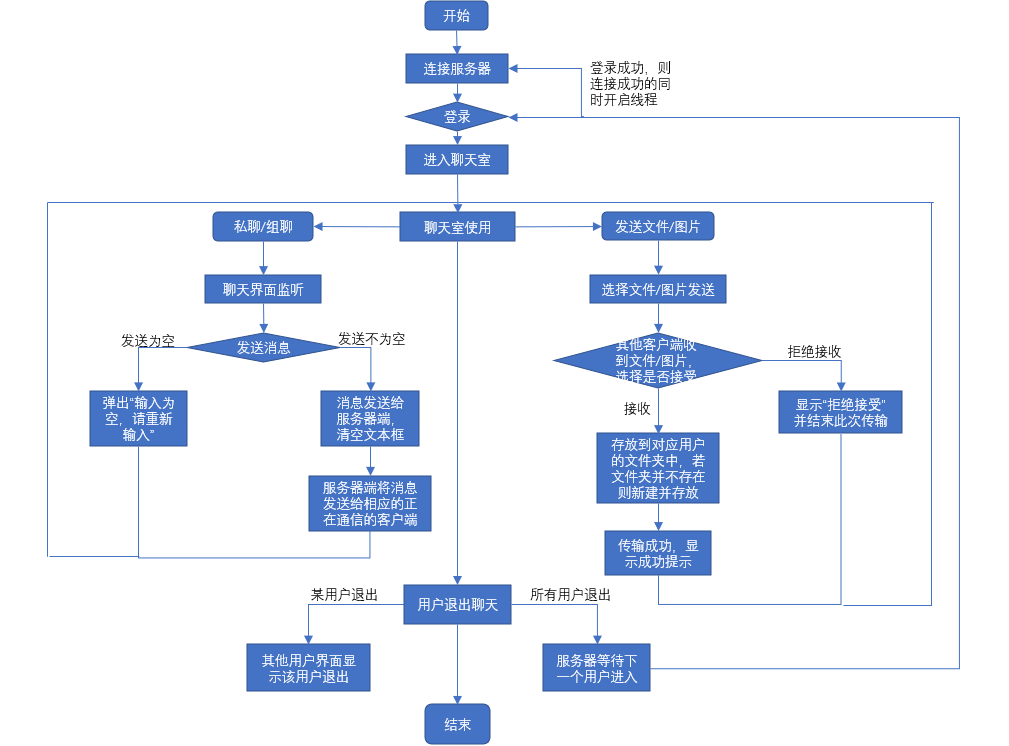
**ServerFileThread**：运用一个List对客户端进行储存，等待客户端的请求，如果建立了新的客户端就将客户端存放到List中，并对每次请求都建立一个FileThread线程进行处理。1）、文件传输线程：用输入流获取文件名和文件长度，遍历List中的客户端，用输出流将文件发送给其他客户端；2）、FileReadAndWrite线程：用输入流和输出流将文件名称、文件长度和内容发送给每一个客户端。

**ClientFile**：负责客户端文件（包括图片）的发送和接收。当一个用户发送文件，则其他用户窗口弹出是否接受文件的弹窗，此时提示框的选择结果0为接收，1为拒绝接收。如果确定接收则输出用户确认接收的消息，并将接收到的文件存放到对应用户的文件夹下面，如果文件夹不存在则新建该文件夹，文件传输成功后输出“文件接收成功”的信息；如果拒绝接收，则输出提示“拒绝接收文件”。

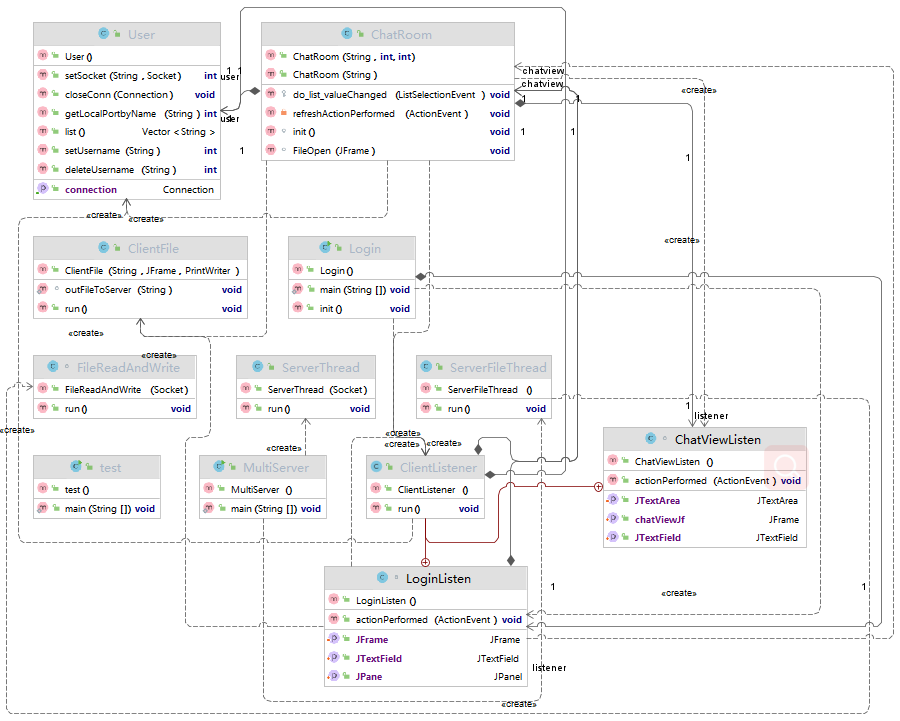
**User**：保存了客户端的登录用户名与端口号，用于私聊时聊天对象的选择。

## 2.2 工作流程和类间关系

### 2.2.1 工作流程

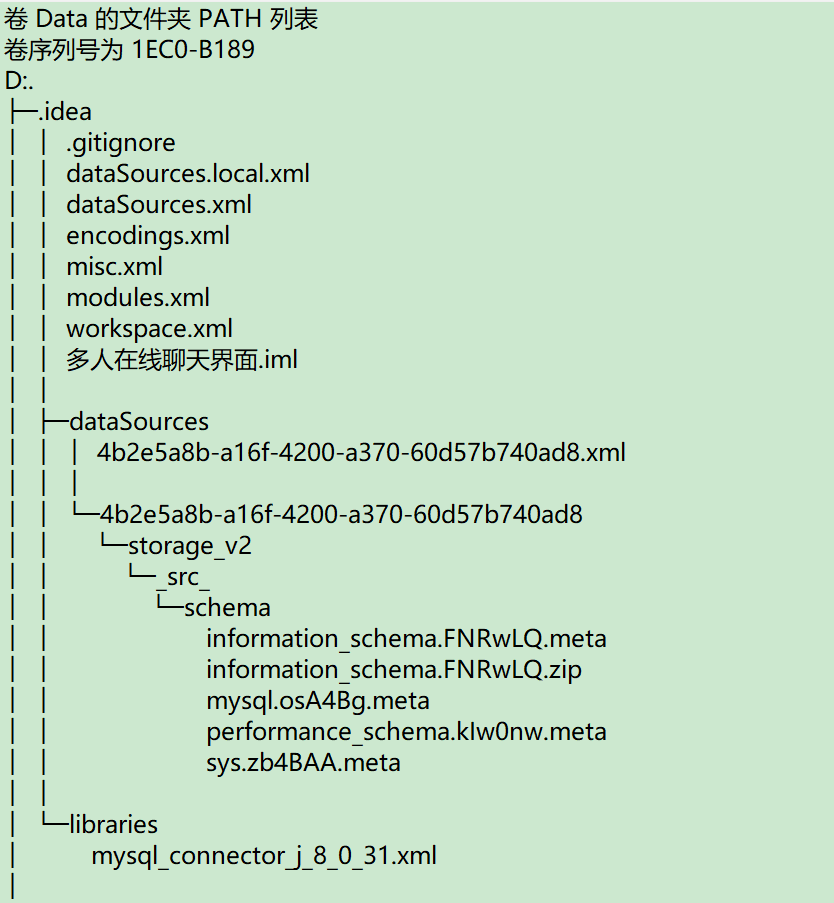


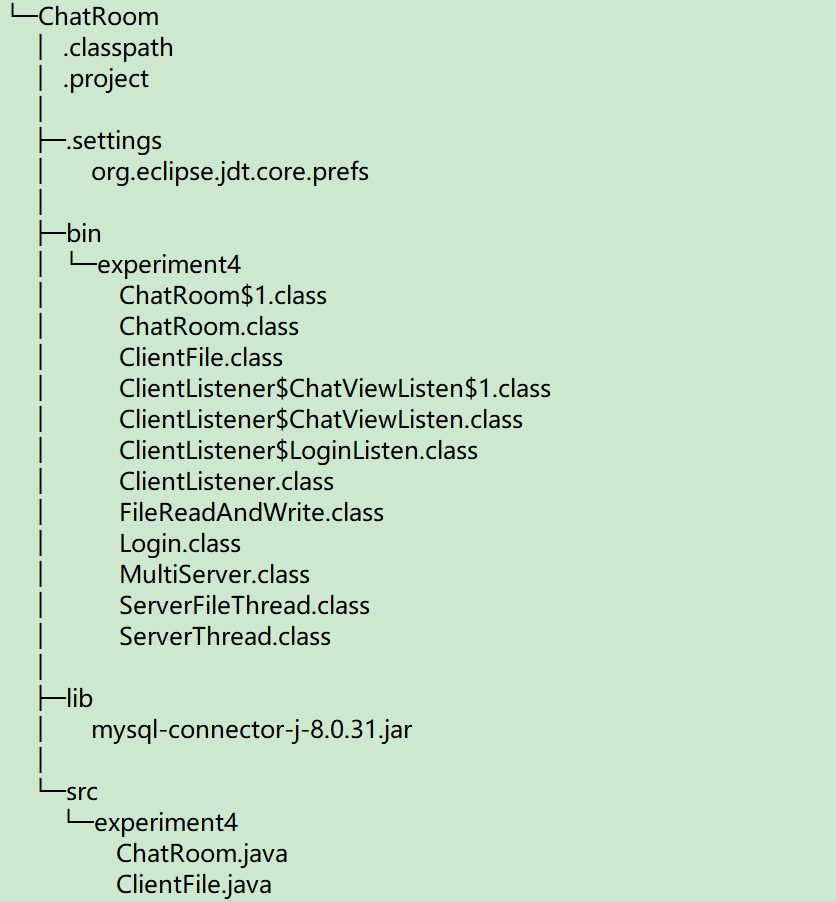
### 2.2.2 类间关系

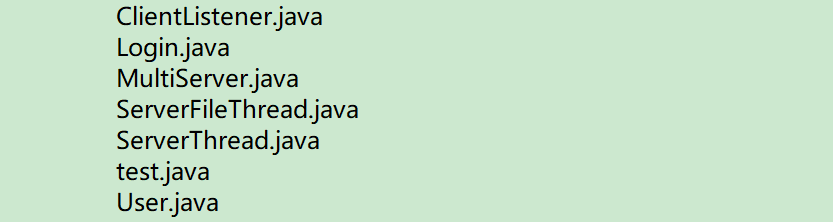


# 3 关键代码（技术）描述

## 3.1 源文件文件树







## 3.2 Login

### 3.2.1 Login类的重要属性：

JPanel **contentPane**;  
JTextField **textField** = **null**;  
ClientListener.LoginListen **listener**;

### 3.2.2 Login类的重要方法：

**public** Login() {  
 init();

//创建Frame:**public void** init() {  
 User user;  
 setTitle(**"\u767B\u5F55"**);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 setBounds(100, 100, 300, 150);  
 **contentPane** = **new** JPanel();  
 **contentPane**.setToolTipText(**""**);  
 **contentPane**.setBorder(**new** EmptyBorder(5, 5, 5, 5));  
 setContentPane(**contentPane**);  
 **contentPane**.setLayout(**null**);  
   
 **textField** = **new** JTextField();  
   
 **textField**.setBounds(76, 30, 166, 21);  
 **contentPane**.add(**textField**);  
 **textField**.setColumns(10);  
   
 JLabel lblNewLabel = **new** JLabel(**"\u7528\u6237\u540D"**);  
 lblNewLabel.setBounds(23, 33, 43, 15);  
 **contentPane**.add(lblNewLabel);  
   
 JButton btnNewButton = **new** JButton(**"\u767B\u5F55"**);*//登录按钮* btnNewButton.setBounds(102, 65, 97, 23);  
 **contentPane**.add(btnNewButton);  
   
 *//设置监控* **listener** = **new** ClientListener().**new** LoginListen();  
 **textField**.addActionListener(**listener**);  
 **listener**.setJTextField(**textField**);  
 **listener**.setJPane(**contentPane**);  
 **listener**.setJFrame(**this**);  
 btnNewButton.addActionListener(**listener**);  
   
 **this**.setVisible(**true**);  
}

## 3.3 ChatRoom

### 3.3.1 ChatRoom类的重要属性：

**static** ClientListener.ChatViewListen *listener*;  
*// static ClientListener.chatAloneListen listener1;***static** PrintWriter *out* = **null**;  
**static** String *userName*;  
JPanel **view**;  
**static** Socket *mySocket*;  
**static** JFrame *chatViewJFrame*;  
DefaultListModel **listModel**;  
JTextArea **textArea**;  
JTextField **text**;  
JButton **button**;  
JButton **refreshButton**;  
JList **jList**;  
String **chatWith**;  
  
JScrollPane **scrollPane**;  
User **user**=**new** User();  
**static int** *type*=0;*//默认0是群聊，1是私聊***static int** *otherport*=0;*//与你私聊的人的port号*

### 3.3.1 ChatRoom类的重要方法：

**public** ChatRoom(String name) {  
 *userName* = name;  
 init();  
  
}  
**public** ChatRoom(String name,**int** type,**int** otherport) {  
 *userName* = name;  
 **this**.*type*=type;  
 **this**.*otherport*=otherport;  
  
 init();  
}  
  
**void** init() {  
 setTitle(**"\u804A\u5929\u5BA4"**);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 setBounds(500, 200, 500,390);  
 **view** = **new** JPanel();  
 **view**.setBorder(**new** EmptyBorder(0, 0, 0, 0));  
 setContentPane(**view**);  
 JLabel Label = **new** JLabel(**"用户："**+*userName*);  
 Label.setBounds(186, 0, 94, 32);  
 **text** = **new** JTextField();  
 **text**.setBounds(6, 310, 328, 27);  
 **text**.setColumns(10);  
 **button** = **new** JButton(**"\u53D1\u9001"**);  
 **button**.setBounds(344, 310, 65, 27);  
 **refreshButton**=**new** JButton(**"刷新"**);  
 **refreshButton**.setBounds(412,15,60,25);  
 **view**.add(**refreshButton**);  
 **refreshButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 refreshActionPerformed(e);  
 }  
 });  
 *//String[] set = {"aa","bb","cc","dd","ee","ff","gg"};* **listModel**=**new** DefaultListModel();  
 Vector<String> set=**user**.list();  
  
 **for** (String str : set ) {  
 **listModel**.addElement(str);  
 }  
 **jList**=**new** JList(**listModel**);  
 **jList**.setBounds(400,50,100,100);  
 **jList**.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);*//设置单选* JScrollPane jsp=**new** JScrollPane(**jList**);  
 jsp.setBounds(400, 50, 80, 100);  
 **view**.add(jsp);  
 *//点击后发生响应事件* **jList**.addListSelectionListener(**new** ListSelectionListener() {  
 @Override  
 **public void** valueChanged(ListSelectionEvent e) {  
 do\_list\_valueChanged(e);  
 }  
 });  
 *//打字区域* **scrollPane** = **new** JScrollPane(**textArea**);  
 **scrollPane**.setLocation(6, 42);  
 **scrollPane**.setHorizontalScrollBarPolicy(JScrollPane.***HORIZONTAL\_SCROLLBAR\_AS\_NEEDED***);  
 **scrollPane**.setSize(370, 258); *//要用setSize设定固定大小* **scrollPane**.setVerticalScrollBarPolicy(JScrollPane.***VERTICAL\_SCROLLBAR\_ALWAYS***);  
 **view**.setLayout(**null**);  
 **view**.add(**text**);  
 **view**.add(**button**);  
 **view**.add(**scrollPane**);  
   
 **textArea** = **new** JTextArea(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*欢迎来到聊天室!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);  
 **scrollPane**.setViewportView(**textArea**);  
 **view**.add(Label);  
 *//添加监听  
 listener* = **new** ClientListener().**new** ChatViewListen();  
 *listener*.setJTextField(**text**);  
 *listener*.setJTextArea(**textArea**);  
   
 JButton fileButton = **new** JButton(**"\u6587\u4EF6"**);  
 JFrame jf = **this**;  
 fileButton.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 FileOpen(jf);  
 }  
 });  
 fileButton.setBounds(411, 310, 65, 27);  
 **view**.add(fileButton);  
 *// view.add(fileButton2);  
 listener*.setChatViewJf(**this**);  
 **text**.addActionListener(*listener*);  
 **button**.addActionListener(*listener*);  
   
 **this**.setVisible(**true**);  
}

*// 创建文件选择器***void** FileOpen(JFrame jf) {  
   
 JFileChooser fileChooser = **new** JFileChooser();  
 fileChooser.setCurrentDirectory(**new** File(**"C:/Users/Samven/Desktop"**));*// 设置默认显示的文件夹* fileChooser.setFileFilter(**new** FileNameExtensionFilter(**"(txt)"**, **"txt"**));  
 **int** result = fileChooser.showOpenDialog(jf); *// 对话框将会尽量显示在靠近 parent 的中心* **if**(result == JFileChooser.***APPROVE\_OPTION***) {*//如果选择了文件* File file = fileChooser.getSelectedFile();  
 String path = file.getAbsolutePath();  
 ClientFile.*outFileToServer*(path);  
 }  
}  
**protected void** do\_list\_valueChanged(ListSelectionEvent e) {  
 JOptionPane.*showMessageDialog*(**this**, **"我要与"** +**jList**.getSelectedValue()+**"私聊"**, **"选择私聊对象"**, JOptionPane.***INFORMATION\_MESSAGE***);  
  
 String chatwithName=**jList**.getSelectedValue().toString();  
 **try** {  
 User user=**new** User();  
 InetAddress addr = InetAddress.*getByName*(**null**); *// 获取主机地址* MultiServer.*mapServer*=**new** HashMap<Socket, Integer>();  
 System.***out***.println(**"私聊时聊天对象:"**+chatwithName);  
 *//Socket ChatWithSocket=user.getSocketbyName(chatwithName);//得到对方的socket* **int** localport=user.getLocalPortbyName(chatwithName);  
 System.***out***.println(**"私聊时localport:"**+localport);  
 ChatRoom chatview=**new** ChatRoom(*userName*+**"与"**+**jList**.getSelectedValue()+**"私聊"**,1,localport);  
 *mySocket* = **new** Socket(addr,5500);  
 MultiServer.*mapServer*.put(*mySocket*,localport);  
 *out* = **new** PrintWriter(*mySocket*.getOutputStream()); *// 输出流  
 out*.println(**"开启私聊s"**); *// 发送用户名给服务器  
 out*.flush(); *// 清空缓冲区out中的数据* } **catch** (IOException a) {  
 a.printStackTrace();  
 }  
 ClientListener readAndPrint = **new** ClientListener();  
 readAndPrint.start();  
 *// 新建文件读写线程并启动* ClientFile fileThread = **new** ClientFile(*userName*, *chatViewJFrame*, *out*);  
 fileThread.start();  
 }

**private void** refreshActionPerformed(ActionEvent evt) {  
 Vector<String> set=**user**.list();  
 **listModel**.clear();  
 **for** (String str : set ) {  
 **listModel**.addElement(str);  
 }  
  
}

### 

## 3.4 ClientListener

### 3.4.1 ClientListener类重要属性：

**static** Socket *mySocket*;  
**static** JTextField *textInput*;  
**static** JTextArea *textShow*;  
**static** JFrame *chatViewJFrame*;  
**static** BufferedReader *in* = **null**;  
**static** PrintWriter *out* = **null**;  
**static** String *userName*; *//用户名*

### 3.4.2 ClientListener类重要方法：

*// 用于接收从服务端发送来的消息*

**public void** run() {  
 **try** {  
 *in* = **new** BufferedReader(**new** InputStreamReader(*mySocket*.getInputStream())); *// 输入流* **while** (**true**) {  
 String str = *in*.readLine(); *// 获取服务端发送的信息  
 textShow*.append(str + **'\n'**); *// 添加进聊天客户端的文本区域  
 textShow*.setCaretPosition(*textShow*.getDocument().getLength());  
 }  
 } **catch** (Exception e) {}  
}

登录界面监听执行函数：

*//点击登录之后到这里来了*@Override  
**public void** actionPerformed(ActionEvent e) {  
 *//* ***TODO Auto-generated method stub*** *userName* = **textField**.getText();  
 *// user.setUsername(userName);* **chatview** = **new** ChatRoom(*userName*);  
 **try** {  
 InetAddress addr = InetAddress.*getByName*(**null**); *// 获取主机地址  
 mySocket* = **new** Socket(addr,5500);  
  
 System.***out***.println(**"登录时mysocket:"**+*mySocket*);  
 System.***out***.println(**"登录时userName:"**+*userName*);  
 **user**.setSocket(*userName*,*mySocket*);  
 **loginJFrame**.setVisible(**false**); *// 隐藏登录窗口  
 out* = **new** PrintWriter(*mySocket*.getOutputStream()); *// 输出流  
 out*.println(**"用户["** + *userName* + **"]进入聊天室！"**); *// 发送用户名给服务器  
 out*.flush(); *// 清空缓冲区out中的数据* } **catch** (IOException a) {  
 a.printStackTrace();  
 }  
 ClientListener readAndPrint = **new** ClientListener();  
 readAndPrint.start();  
 *// 新建文件读写线程并启动* ClientFile fileThread = **new** ClientFile(*userName*, *chatViewJFrame*, *out*);  
 fileThread.start();  
}

聊天界面监听执行函数：

**public void** actionPerformed(ActionEvent event) {  
 **try** {  
 **int** type=ChatRoom.*type*;  
 **int** port=ChatRoom.*otherport*;  
 String str = *textInput*.getText();  
 System.***out***.println(**"我在监听这里"**+type);  
 *// 文本框内容为空* **if**(**""**.equals(str)) {  
 *textInput*.grabFocus();  
 JOptionPane.*showMessageDialog*(*chatViewJFrame*, **"输入为空，请重新输入！"**, **"提示"**, JOptionPane.***WARNING\_MESSAGE***);*// 弹出消息对话框（警告消息）* **return**;  
 }  
 LocalDateTime t = LocalDateTime.*now*(); *//获取本地时间并格式化* DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*(**"YYYY/MM/dd HH:mm:ss"**);  
 String time = formatter.format(t);  
 String s;  
 **if**(type==1){  
 System.***out***.println(**"type是"**+type);  
 s = *userName* + **": "** + str + **" "** + time+port+**"s"**;  
 System.***out***.println(**"发送："**+s);  
 }  
 **else**{  
 s = *userName* + **": "** + str + **" "** + time;}  
 System.***out***.println(**"要发给服务器了"**);  
 *out*.println(s); *// 输出给服务端  
 out*.flush(); *// 清空缓冲区out中的数据  
 textInput*.setText(**""**); *// 清空文本框  
 textInput*.grabFocus();  
 } **catch** (Exception e) {}  
 }  
}

私聊监听：

**public** chatAloneListen(String chatWith,String userName1){  
 **this**.**name**=chatWith;  
 **this**.**userName1**=userName1;  
 }  
 **public void** setJPane(JPanel pane) {  
 **this**.**contentPane** = pane;  
 }  
 @Override  
 **public void** valueChanged(ListSelectionEvent e) {  
 ChatRoom chatview = **new** ChatRoom(**name**);  
 **try** {  
 InetAddress addr = InetAddress.*getByName*(**null**); *// 获取主机地址  
 mySocket* = **new** Socket(addr,5500);  
 **loginJFrame**.setVisible(**false**); *// 隐藏登录窗口  
 out* = **new** PrintWriter(*mySocket*.getOutputStream()); *// 输出流  
 out*.println(**"用户["** + **userName1** + **"]进入聊天室！"**); *// 发送用户名给服务器  
 out*.flush(); *// 清空缓冲区out中的数据* } **catch** (IOException a) {  
 a.printStackTrace();  
 }  
 ClientListener readAndPrint = **new** ClientListener();  
 readAndPrint.start();  
 *// 新建文件读写线程并启动* ClientFile fileThread = **new** ClientFile(**userName1**, *chatViewJFrame*, *out*);  
 fileThread.start();  
 }  
}

## 3.5 MultiServer

### 3.5.1 MultiServer类重要属性：

**static** ServerSocket *server* = **null**;  
**static** Socket *socket* = **null**;  
**static** List<Socket> *list* = **new** ArrayList<Socket>(); *// 存储客户端***static** Map<Socket, Integer> *mapServer*=**new** HashMap<>();

### 3.5.2 MultiServer类重要方法：

**public static void** main(String[] args) {  
 **try** {  
 System.***out***.println(**"等待连接"**);  
 *// 在服务器端对客户端开启文件传输的线程* ServerFileThread serverFileThread = **new** ServerFileThread();  
 serverFileThread.start();  
 *server* = **new** ServerSocket(5500);  
 *// 等待连接并开启相应线程* **while** (**true**) {  
 *socket* = *server*.accept(); *// 等待连接  
 list*.add(*socket*); *// 添加当前客户端到列表  
 // 在服务器端对客户端开启相应的线程* ServerThread s = **new** ServerThread(*socket*);  
 s.start();  
 System.***out***.println(**"连接成功"**);  
 }  
 } **catch** (IOException e1) {  
 e1.printStackTrace(); *// 出现异常则打印出异常的位置* }  
}

## 3.6 ServerThread

### 3.6.1 ServerThread类重要属性：

Socket **nowSocket** = **null**;  
BufferedReader **in** =**null**;  
PrintWriter **out** = **null**;

### 3.6.2 ServerThread类重要方法：

**public void** run(){  
 **try** {  
 *// 输入流* **in** = **new** BufferedReader(**new** InputStreamReader(**nowSocket**.getInputStream()));   
 **while** (**true**) {*// 获取客户端信息并把信息发送给所有客户端* String str = **in**.readLine();  
 **char** c = str.charAt(str.length() - 1);  
 System.***out***.println(**"run"**+str);  
 **if**(c==**'s'**){  
 System.***out***.println(**"run1"**+str);  
 String str2=str.substring(str.length() - 6,str.length() - 1);  
 System.***out***.println(**"run2"**+str2);  
 **int** inum = Integer.*parseInt*(str2);  
 System.***out***.println(MultiServer.*list*);  
 **for**(Socket socket: MultiServer.*list*) {*// 发送给所有客户端* **out** = **new** PrintWriter(socket.getOutputStream());  
 System.***out***.println(**"这里遍历："**);  
 System.***out***.println(socket.getLocalPort());  
 **if**(socket == **nowSocket**) { *// 发送给当前客户端* **out**.println(**"(你)"** + str.substring(0,str.length() - 6));  
 }  
 **else if**(socket.getPort()==inum){ *// 发送给其它客户端* System.***out***.println(**"私聊发出来"**);  
 **out**.println(str.substring(0,str.length() - 6));  
 }  
 **out**.flush();  
 }  
 }  
 **else**{  
 **for**(Socket socket: MultiServer.*list*) {*// 发送给所有客户端* **out** = **new** PrintWriter(socket.getOutputStream());  
 **if**(socket == **nowSocket**) { *// 发送给当前客户端* **out**.println(**"(你)"** + str);  
 }  
 **else** { *// 发送给其它客户端* **out**.println(str);  
 }  
 **out**.flush();  
 }  
 System.***out***.println(str)  
 }}  
  
 } **catch** (Exception e) {  
 MultiServer.*list*.remove(**nowSocket**); *// 线程关闭，移除相应套接字* }  
}

## 3.7 ServerFileThread

### 3.7.1 ServerFileThread类重要属性：

文件传输：

ServerSocket **server** = **null**;  
Socket **socket** = **null**;  
**static** List<Socket> *list* = **new** ArrayList<Socket>(); *// 存储客户端*

文件读写：

**private** Socket **nowSocket** = **null**;  
**private** DataInputStream **input** = **null**;  
**private** DataOutputStream **output** = **null**;

### 3.7.2 ServerFileThread类重要方法：

文件传输：

**public void** run() {  
 **try** {  
 **server** = **new** ServerSocket(8090);  
 **while**(**true**) {  
 **socket** = **server**.accept();  
 *list*.add(**socket**);  
 *// 开启文件传输线程* FileReadAndWrite fileReadAndWrite = **new** FileReadAndWrite(**socket**);  
 fileReadAndWrite.start();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
}

文件读写：

**public** FileReadAndWrite(Socket socket) {  
 **this**.**nowSocket** = socket;  
}  
**public void** run() {  
 **try** {  
 **input** = **new** DataInputStream(**nowSocket**.getInputStream());  
 **while** (**true**) {  
 String textName = **input**.readUTF();  
 **long** textLength = **input**.readLong();  
 **for**(Socket socket: ServerFileThread.*list*) {*//发送文件名字和文件长度给所有客户端* **output** = **new** DataOutputStream(socket.getOutputStream());  
 **if**(socket != **nowSocket**) {  
 **output**.writeUTF(textName);  
 **output**.flush();  
 **output**.writeLong(textLength);  
 **output**.flush();  
 }  
 }  
 *// 发送文件内容* **int** length = -1;  
 **long** curLength = 0;  
 **byte**[] buff = **new byte**[1024];  
 **while** ((length = **input**.read(buff)) > 0) {  
 curLength += length;  
 **for**(Socket socket: ServerFileThread.*list*) {  
 **output** = **new** DataOutputStream(socket.getOutputStream());  
 **if**(socket != **nowSocket**) {  
 **output**.write(buff, 0, length);  
 **output**.flush();  
 }  
 }  
 **if**(curLength == textLength) {  
 **break**;  
 }  
 }  
 }  
 } **catch** (Exception e) {  
 ServerFileThread.*list*.remove(**nowSocket**); *// 线程关闭* }  
}

## 3.8 ClientFile

### 3.8.1 ClientFile类重要属性：

**private** Socket **socket** = **null**;  
**private** JFrame **chatViewJFrame** = **null**;  
**static** String *userName* = **null**;  
**static** PrintWriter *out* = **null**;  
**static** DataInputStream *fileIn* = **null**;  
**static** DataOutputStream *fileOut* = **null**;  
**static** DataInputStream *fileReader* = **null**;  
**static** DataOutputStream *fileWriter* = **null**;

### 3.8.2 ClientFile类重要方法：

*// 客户端接收文件***public void** run() {  
 **try** {  
 InetAddress addr = InetAddress.*getByName*(**null**);  
 **socket** = **new** Socket(addr, 8090);   
 *fileIn* = **new** DataInputStream(**socket**.getInputStream());   
 *fileOut* = **new** DataOutputStream(**socket**.getOutputStream());  
 *//接收文件* **while**(**true**) {  
 String textName = *fileIn*.readUTF();  
 **long** totleLength = *fileIn*.readLong();  
 **int** result = JOptionPane.*showConfirmDialog*(**chatViewJFrame**, **"是否接受？"**, **"提示"**,  
 JOptionPane.***YES\_NO\_OPTION***);  
 **int** length = -1;  
 **byte**[] buff = **new byte**[1024];  
 **long** curLength = 0;  
 *//提示框选择结果，0为确定接收，1为拒绝接收* **if**(result == 0){  
 *out*.println(**"【"** + *userName* + **"选择接收文件！】"**);  
 *out*.flush();  
 File userFile = **new** File(**"D:/test/"** + *userName*);  
 **if**(!userFile.exists()) { *//如果文件夹不存在则新建当前用户的文件夹* userFile.mkdir();  
 }  
 File file = **new** File(**"D:/test/"** + *userName* + **"/"** + textName );  
 *fileWriter* = **new** DataOutputStream(**new** FileOutputStream(file));  
 **while**((length = *fileIn*.read(buff)) > 0) {  
 *fileWriter*.write(buff, 0, length);  
 *fileWriter*.flush();  
 curLength += length;  
 **if**(curLength == totleLength) { *//结束* **break**;  
 }  
 }  
 *out*.println(**"【"** + *userName* + **"文件接收成功！】"**);  
 *out*.flush();  
 }  
 **else** { *// 不接受文件  
 out*.println(**"【"** + *userName* + **"拒绝接收文件！】"**);  
 **while**((length = *fileIn*.read(buff)) > 0) {  
 curLength += length;  
 **if**(curLength == totleLength) { *//结束* **break**;  
 }  
 }  
 }  
 *fileWriter*.close();  
 }  
 } **catch** (Exception e) {}  
}  
  
*// 客户端发送文件***static void** outFileToServer(String path) {  
 **try** {  
 File file = **new** File(path);  
 *fileReader* = **new** DataInputStream(**new** FileInputStream(file));  
 *fileOut*.writeUTF(file.getName()); *// 发送文件名字  
 fileOut*.flush();  
 *fileOut*.writeLong(file.length()); *// 发送文件长度  
 fileOut*.flush();  
 **int** length = -1;  
 **byte**[] buff = **new byte**[1024];  
 **while** ((length = *fileReader*.read(buff)) > 0) { *// 发送内容  
   
 fileOut*.write(buff, 0, length);  
 *fileOut*.flush();  
 }  
 *out*.println(**"【"** + *userName* + **"请求发送文件！】"**);  
 *out*.flush();  
 } **catch** (Exception e) {}  
}

## 3.9 User

### 3.9.1 User类重要属性：

**private** String **username**;  
**private** Socket **socket**;

### 3.9.2 User类重要方法：

*//连接数据库*

**public** Connection getConnection(){  
 **try**{  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 **return** DriverManager.*getConnection*(**"jdbc:mysql://localhost:3306/my\_database"**,**"root"**,**"123456AB"**);  
 }**catch**(ClassNotFoundException e){  
 e.printStackTrace();  
 }**catch**(SQLException e){  
 e.printStackTrace();  
 }  
 **return null**;  
}

*//断开连接***public void** closeConn(Connection conn){  
 **try** {  
 conn.close();  
 } **catch** (SQLException e) {  
 *//* ***TODO Auto-generated catch block*** System.***out***.println(**"数据库关闭异常"**);  
 e.printStackTrace();  
 }  
}  
**public int** deleteUsername(String name) {  
  
 Connection conn=getConnection();  
 **int** i = 0;  
 String sql = **"delete from my\_db where name='"** + name + **"'"**;  
 PreparedStatement pstmt;  
 **try** {  
 pstmt = (PreparedStatement) conn.prepareStatement(sql);  
 i = pstmt.executeUpdate();  
 System.***out***.println(**"resutl: "** + i);  
 pstmt.close();  
 conn.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** i;  
}

**public int** getLocalPortbyName(String Name) **throws** IOException {  
 Connection conn=getConnection();  
 String sql=**"select \* from my\_db where name=Name"**;  
 **int** SocketPort = 0;  
 String Socketlocal;  
 String Addr = **null**;  
 InetAddress addr1 = **null**;  
 **int** SocketLocal = 0;  
 **try**{Statement statement = conn.createStatement();  
 ResultSet resultSet = statement.executeQuery(sql);  
 **while** (resultSet.next()){  
  
 SocketPort=resultSet.getInt(**"port"**);  
 SocketLocal=resultSet.getInt(**"localport"**);  
 Addr=resultSet.getString(**"addr"**);  
*// addr1=InetAddress.getByName(Addr);* }  
  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 } **finally**{  
 closeConn(conn);}  
*// Socketlocal=String.valueOf(SocketLocal);  
// Socket socket = new Socket(addr1,SocketPort);* **return** SocketLocal;  
 }

**public int** setSocket(String name,Socket socket) {  
 **this**.**socket** = socket;  
 **int** localport=socket.getLocalPort();  
 System.***out***.println(localport);  
 **int** port=socket.getPort();  
 System.***out***.println(port);  
 String addr= String.*valueOf*(socket.getInetAddress());  
 System.***out***.println(addr);  
 Connection conn=getConnection();  
 String sql = **"insert into my\_db (name,localport,port,addr) values(?,?,?,?)"**;*//定义sql语句* PreparedStatement ps;  
 **int** i=0;  
 **try** {ps=conn.prepareStatement(sql);  
 ps.setString(1,name);  
 ps.setInt(2,localport);  
 ps.setInt(3,port);  
 ps.setString(4,addr);  
 i = ps.executeUpdate();}  
 **catch**(SQLException e) {  
 e.printStackTrace();  
 **try**{  
 conn.rollback();  
 }**catch**(SQLException el){  
 el.printStackTrace();  
 }  
 }**finally**{  
 closeConn(conn);  
 }  
 **return** i;  
}  
  
**public** Vector<String> list() {  
 Connection conn = getConnection();  
 String sql = **"select \* from my\_db"**;  
 PreparedStatement pstmt;  
 Vector<String> vector =**new** Vector<>();  
 Set<String> set=**new** HashSet<>();  
 **try** {  
 pstmt = (PreparedStatement)conn.prepareStatement(sql);  
 ResultSet rs = pstmt.executeQuery();  
 **int** col = rs.getMetaData().getColumnCount();*//列数* **while** (rs.next()) {*//一行一行输出* set.add(rs.getString(1));  
 vector.add(rs.getString(1));  
  
 }  
  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **return** vector;  
}

# 4测试运行

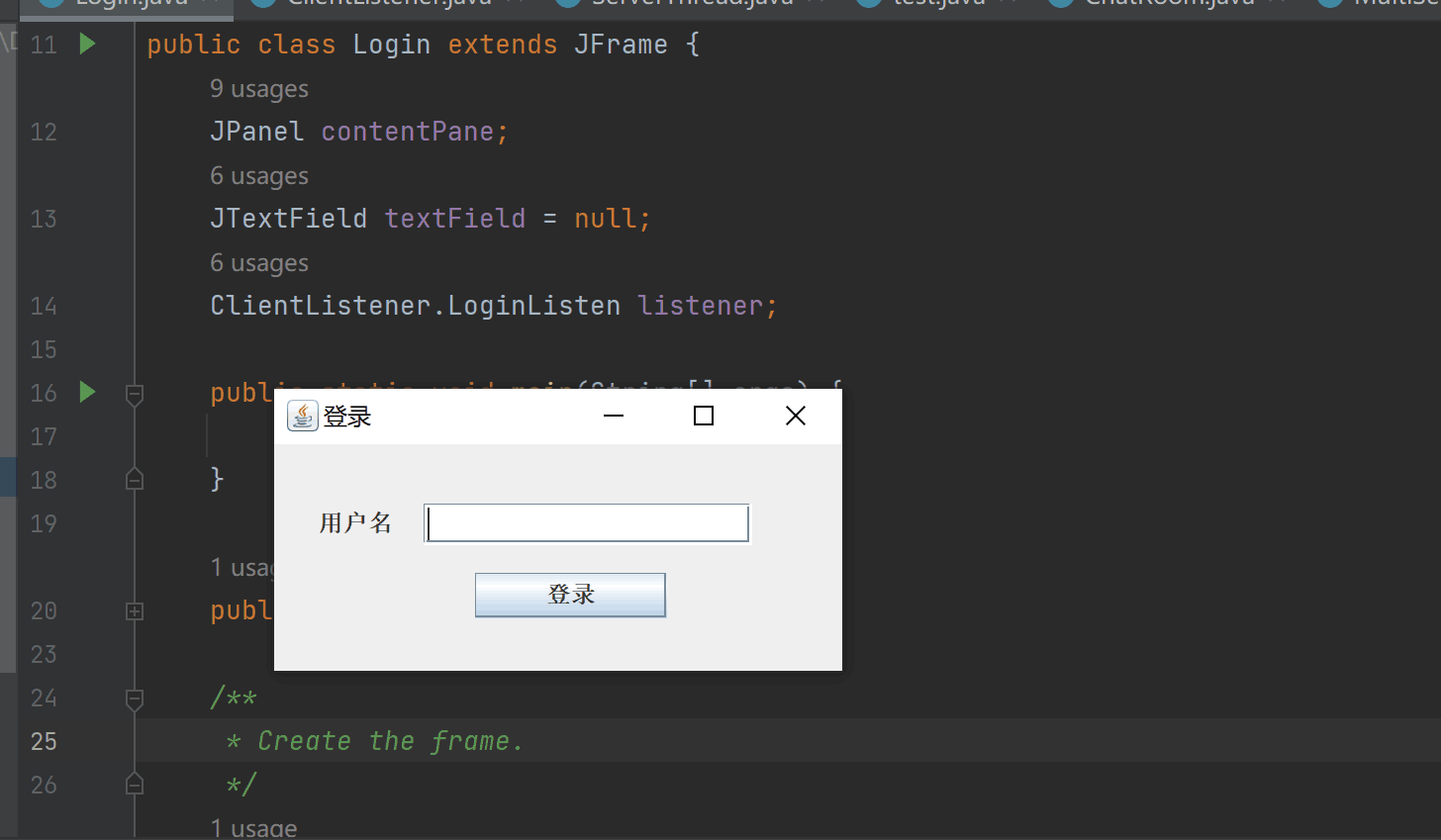
## 4.1 运行过程

### 4.1.1 登录

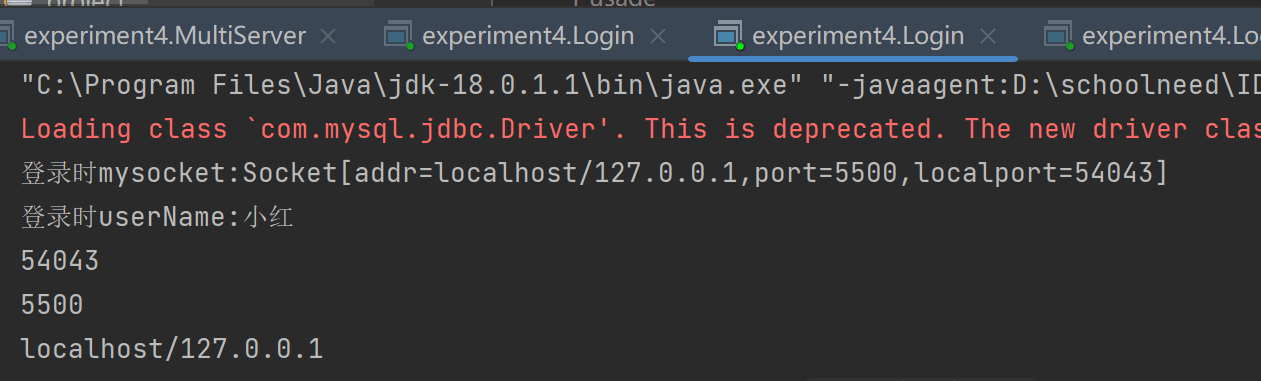
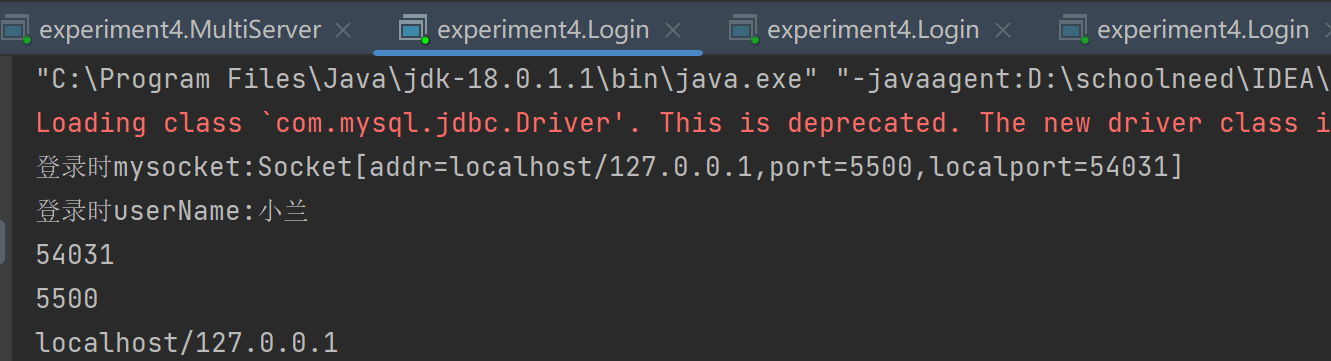
首先运行MultiServer.java文件

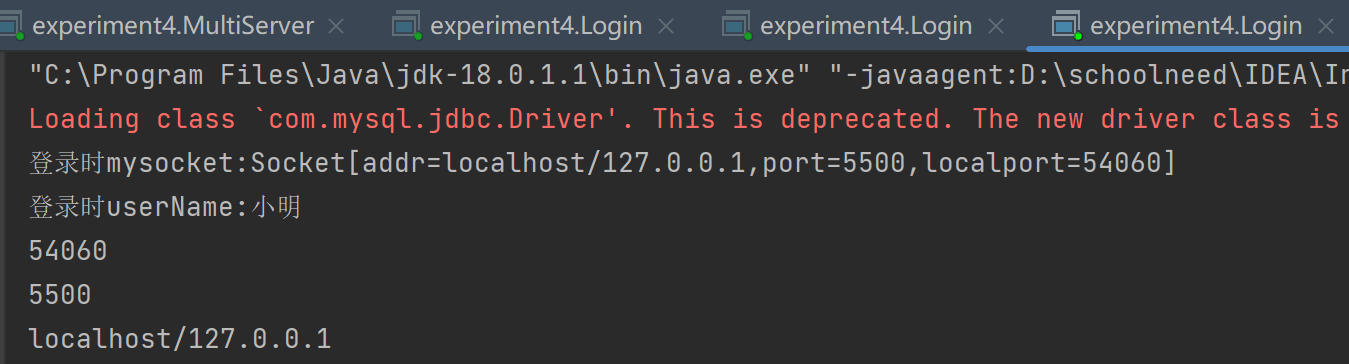
再运行Login.java文件

客户端登录界面：

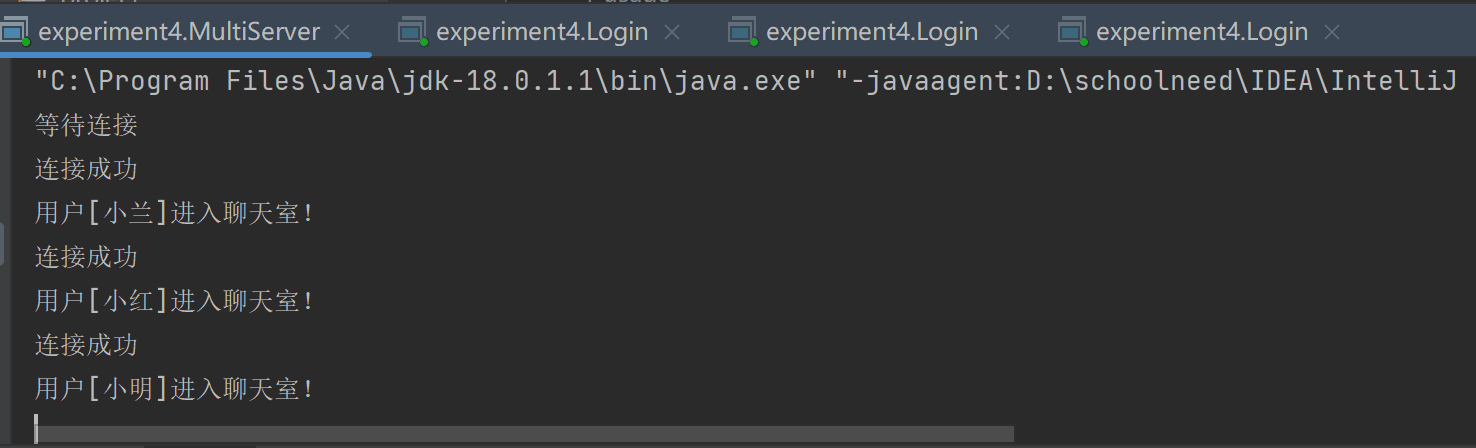


用户小兰、小红、小明成功登录后，各个客户端显示：



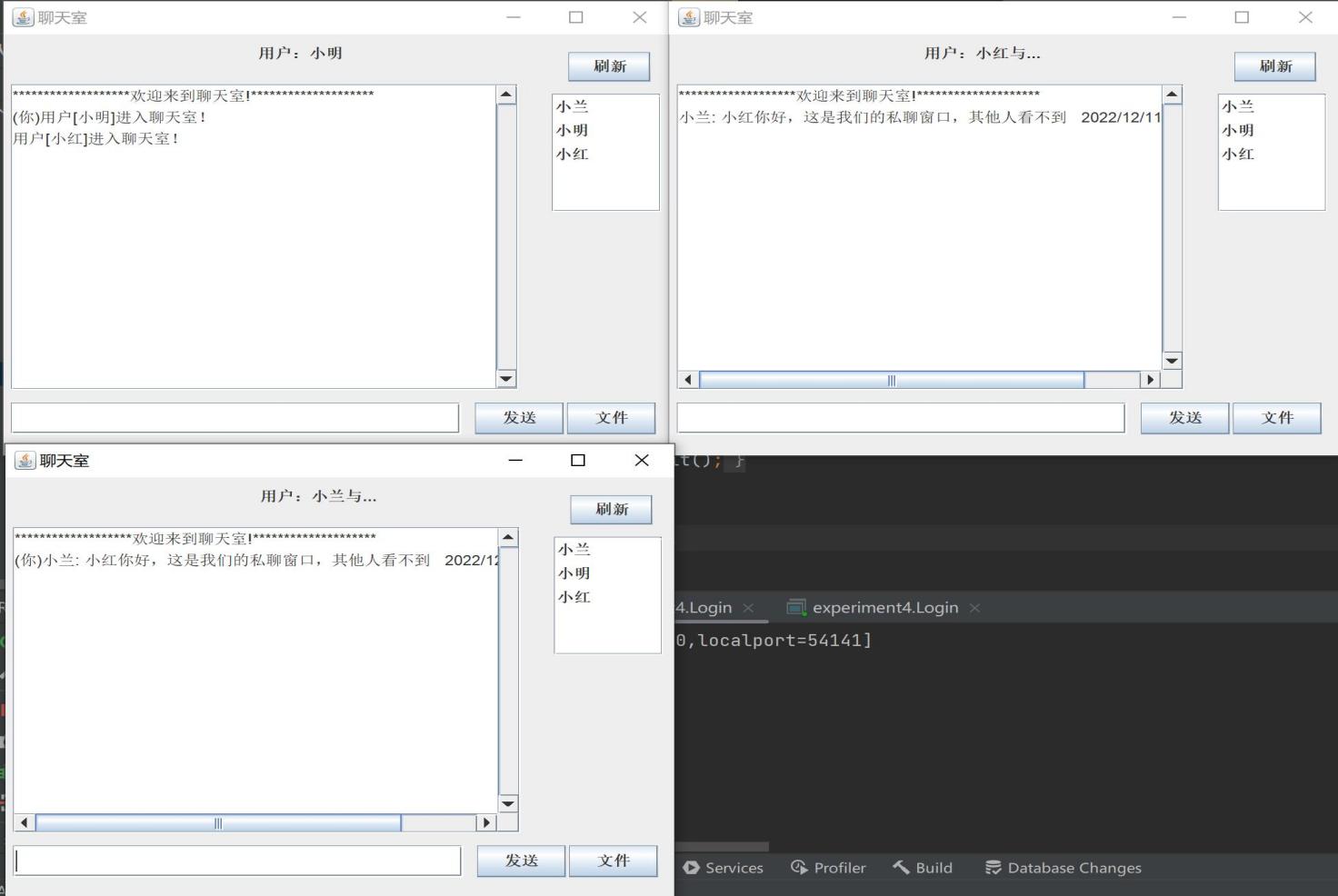


服务器端终端显示：



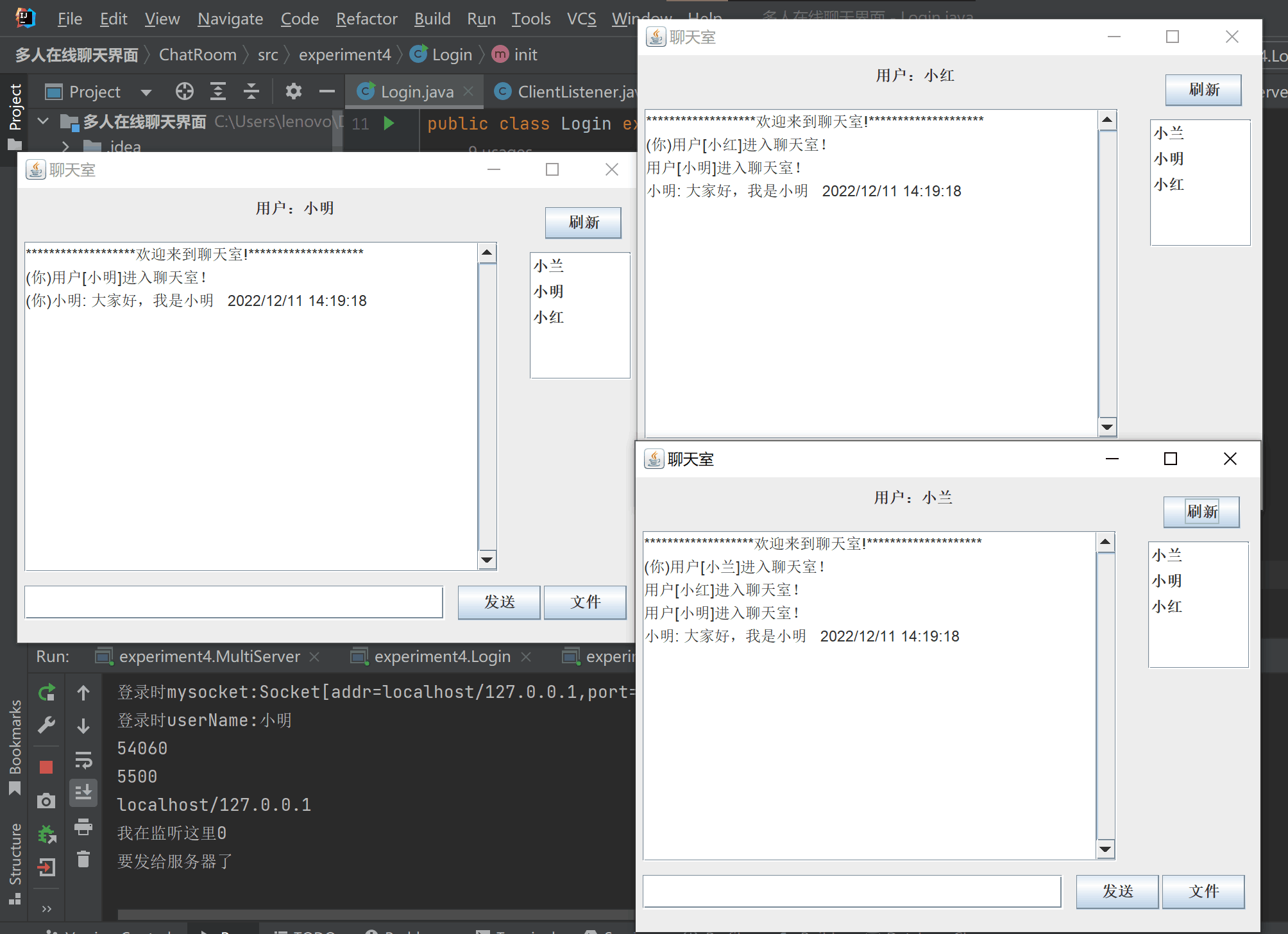
### 4.1.2 私聊

小兰在聊天室的列表中选择向小红发起私聊



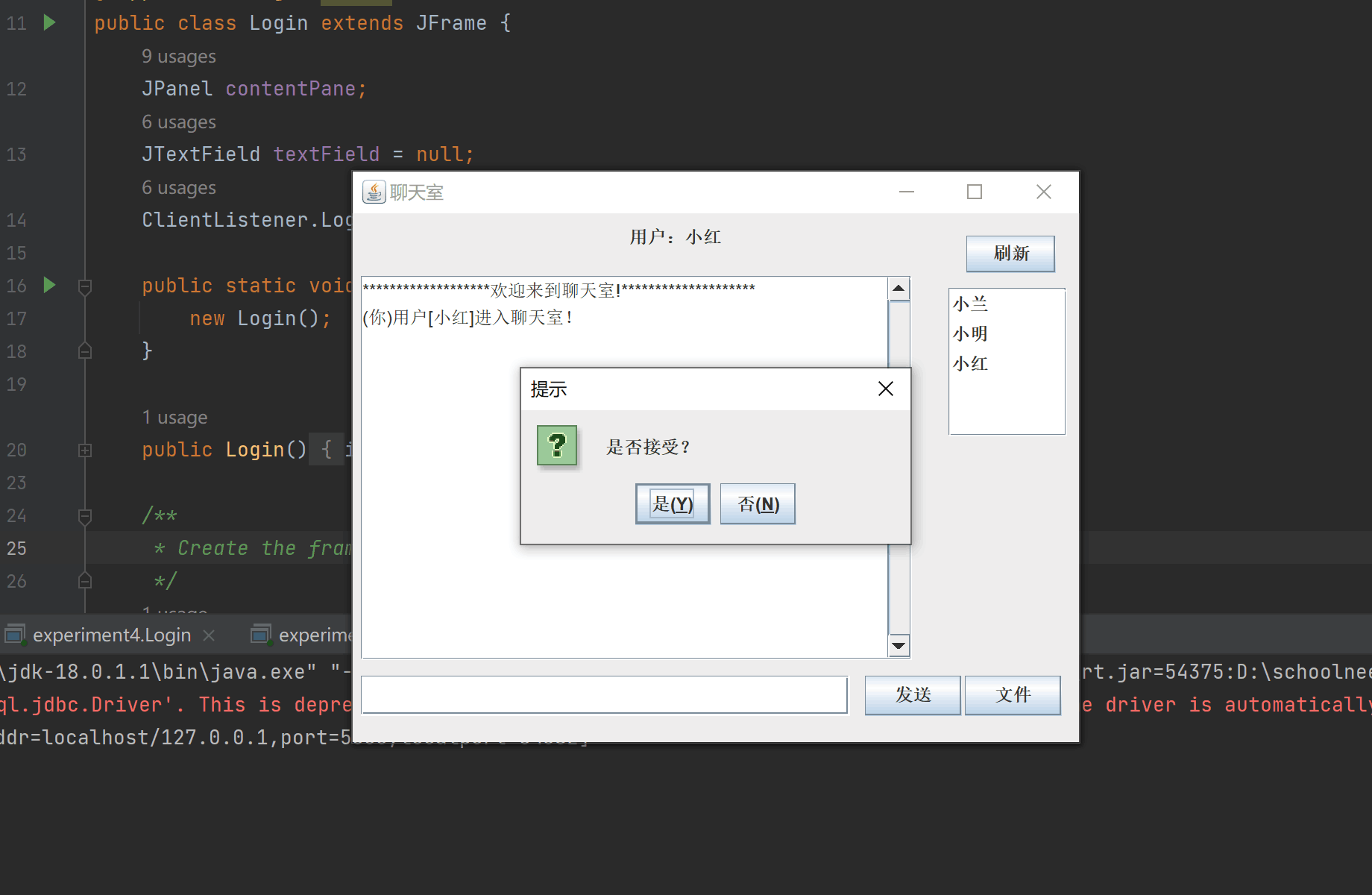
可以看到，小明是收不到小兰和小红间发送的消息的。

### 4.1.3 群聊

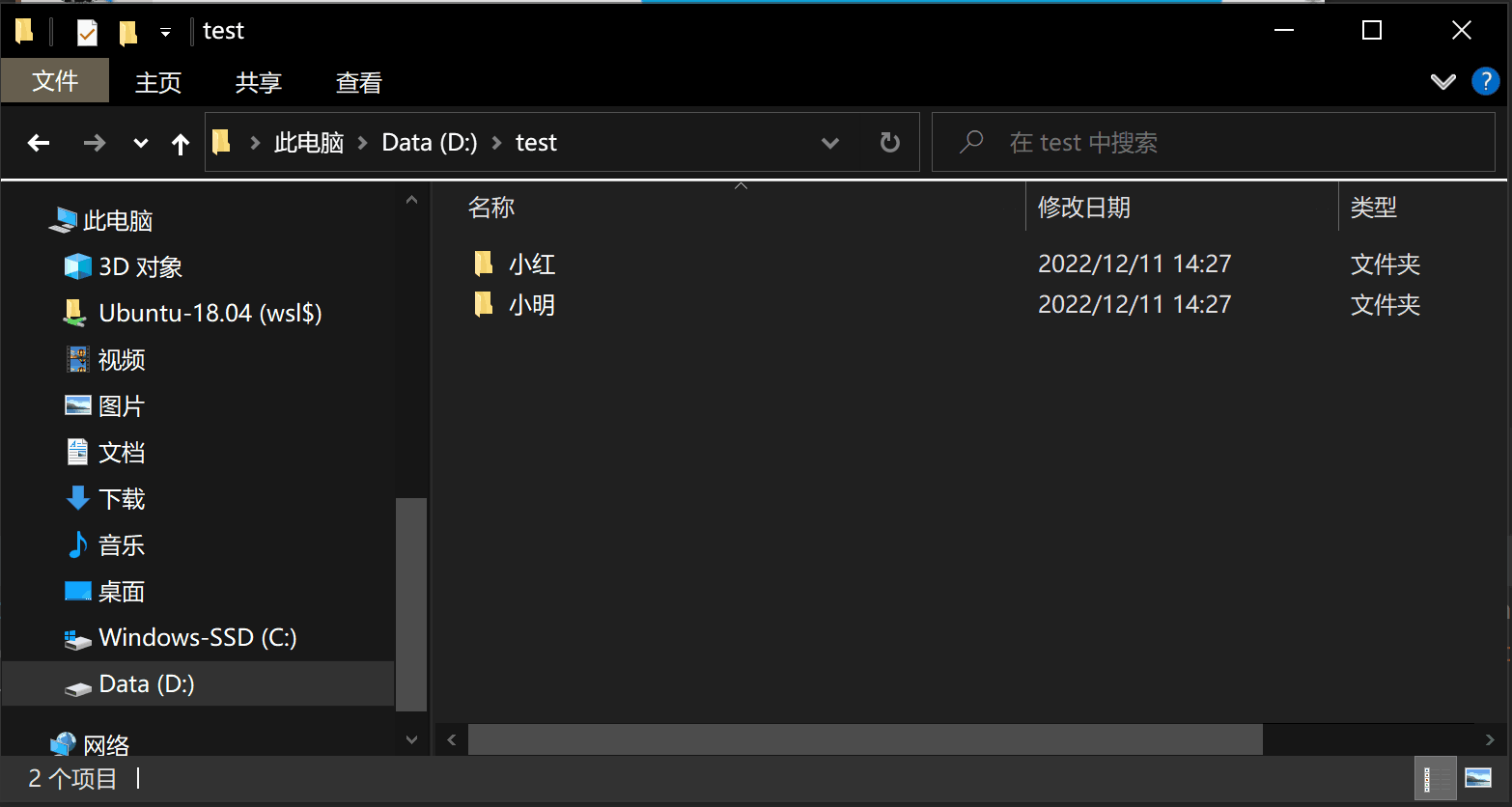


### 4.1.4 发送文件

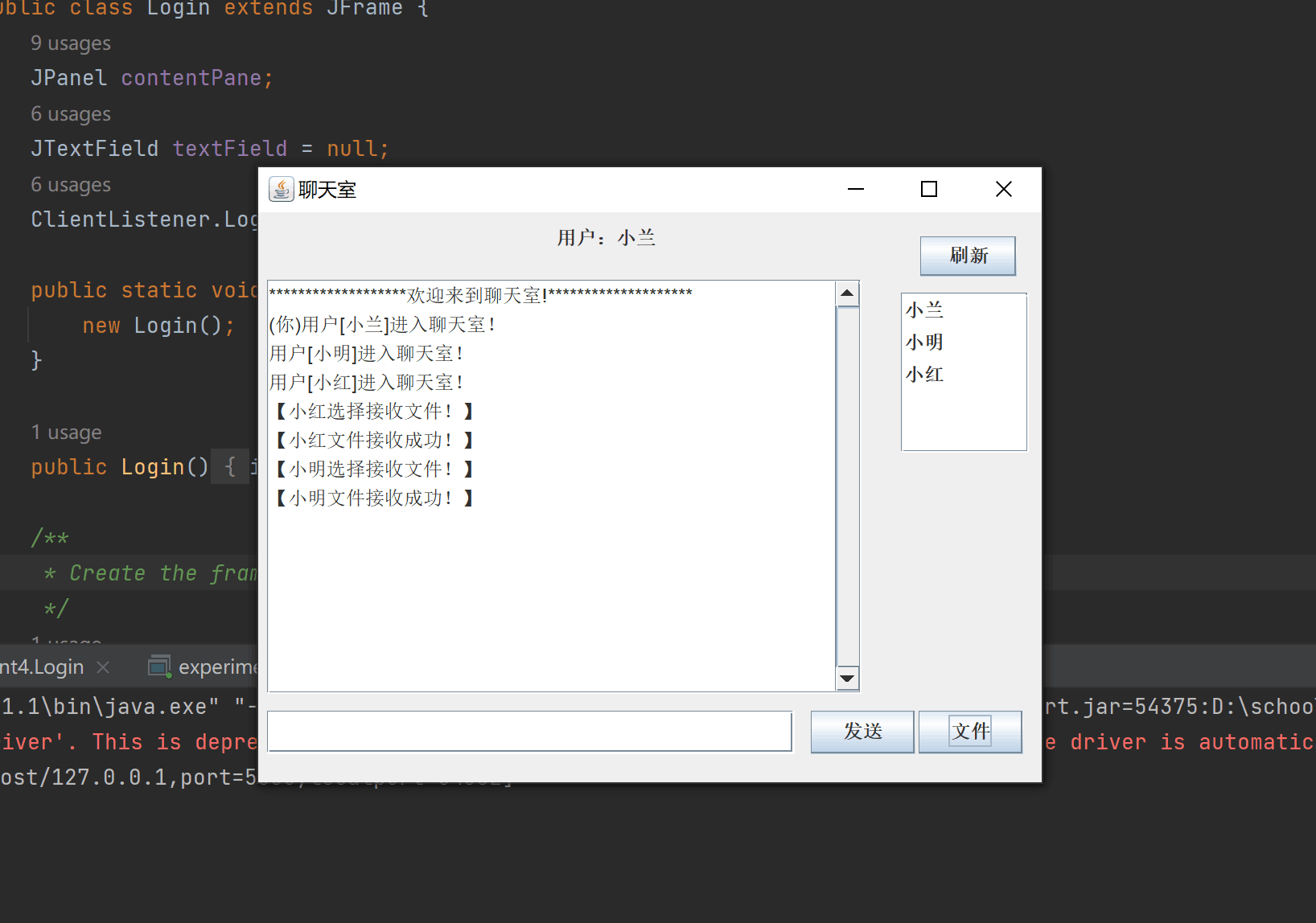
小兰发送文件，小红和小明都会收到提示：



小红和小明接收文件后，文件会保存在各自的文件夹中，若是没有则会新建文件夹：

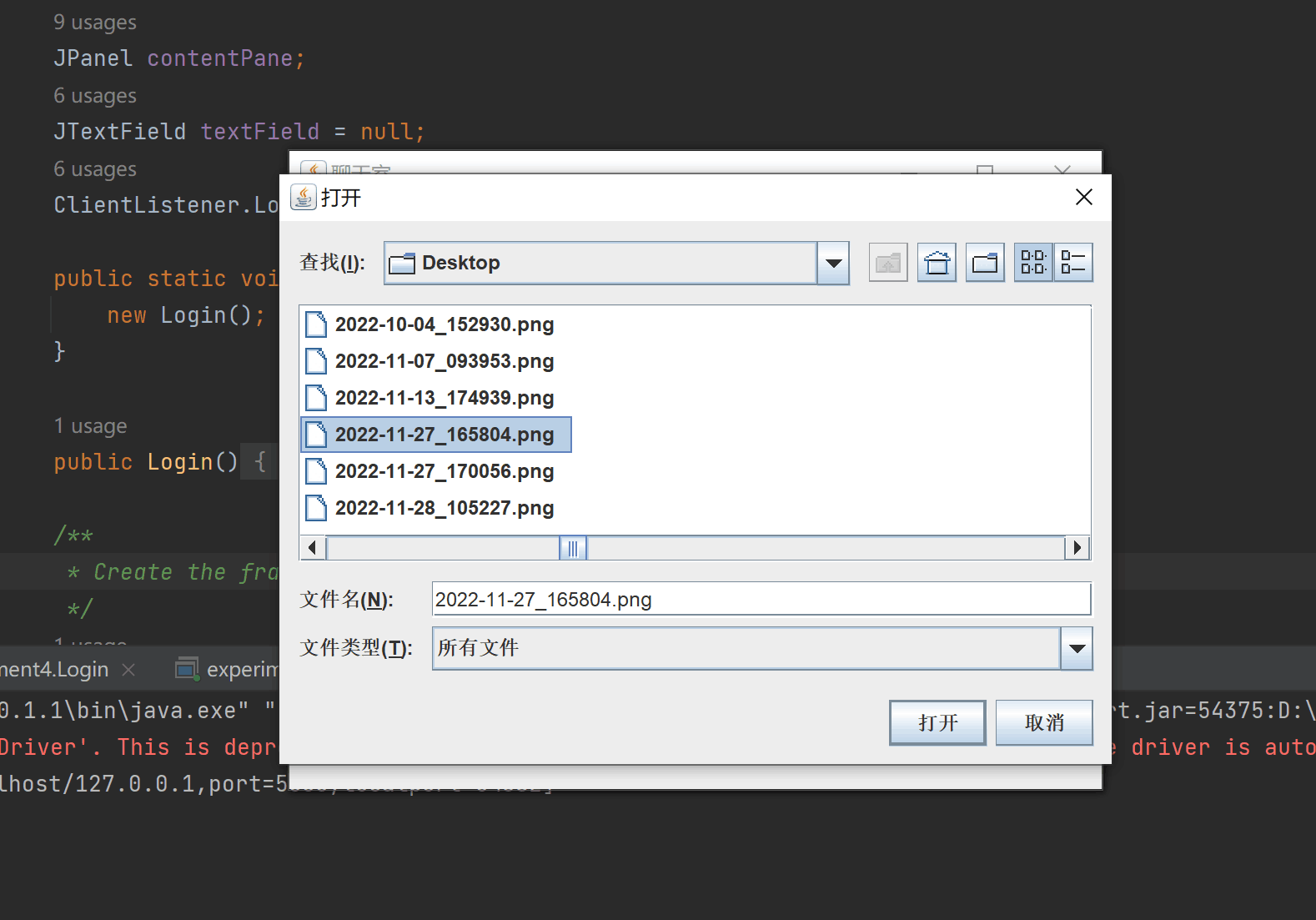


小兰会收到消息：

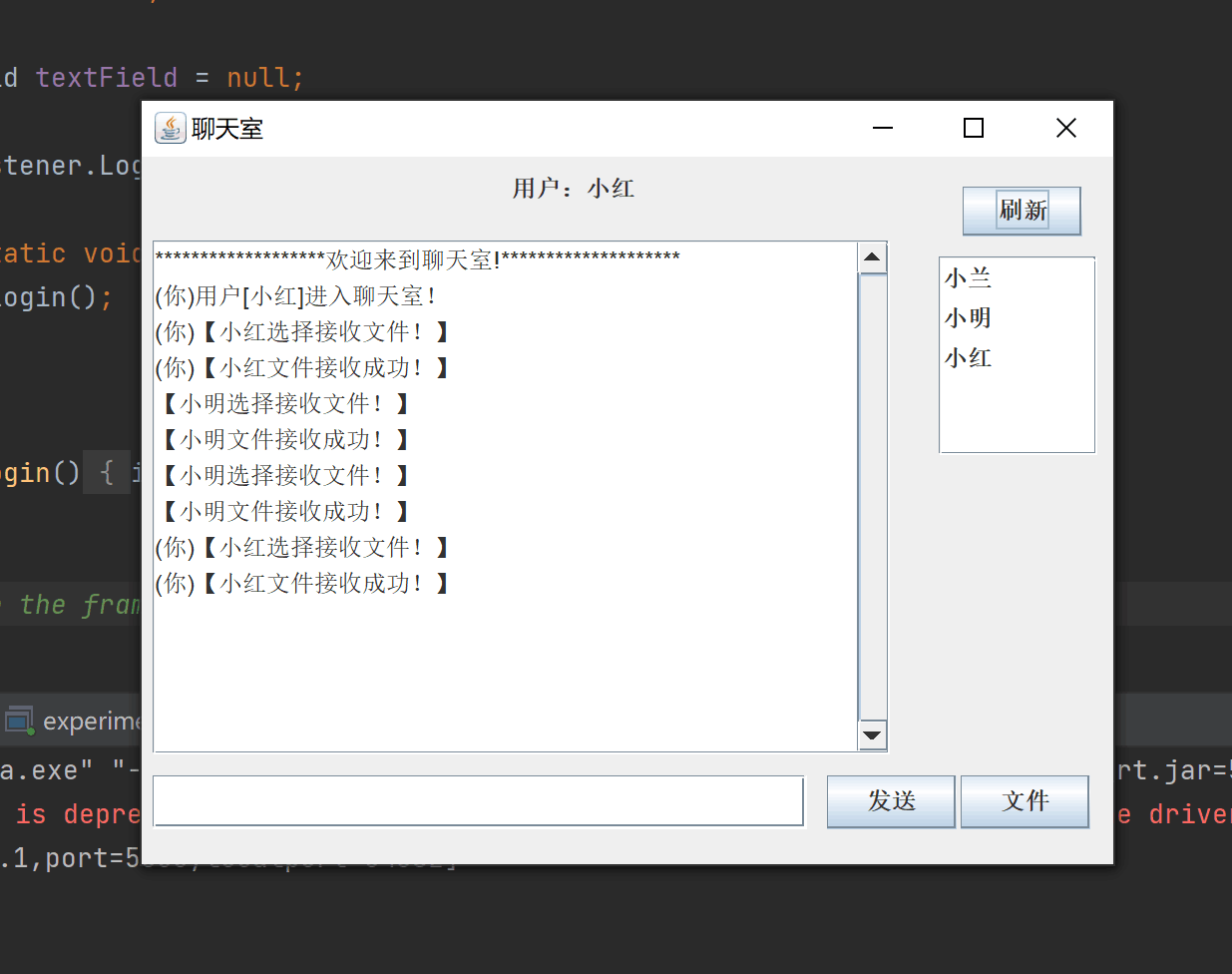


### 4.1.5 发送图片

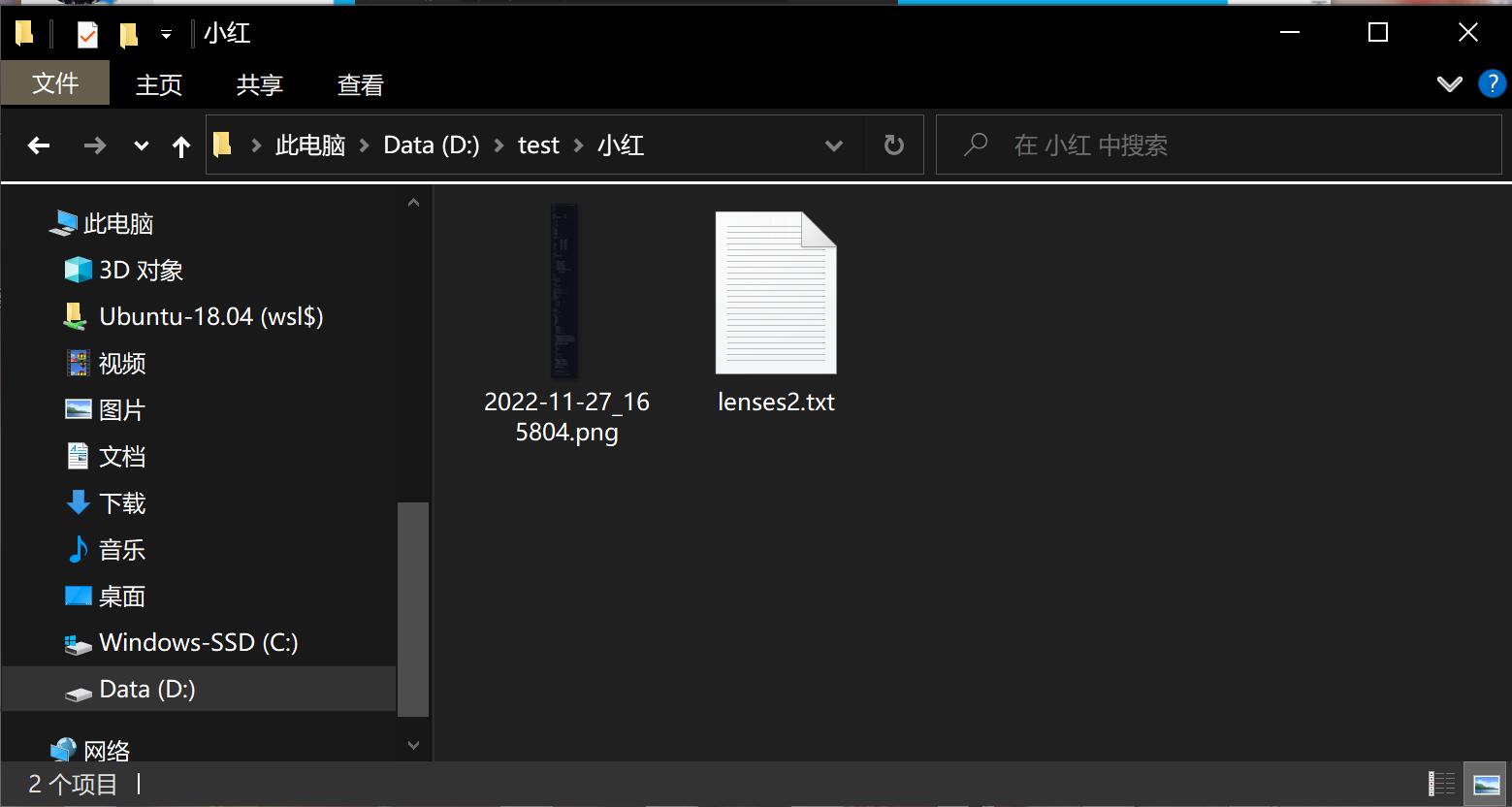
与选择文件同理，选择图片进行发送：



小红接收图片：



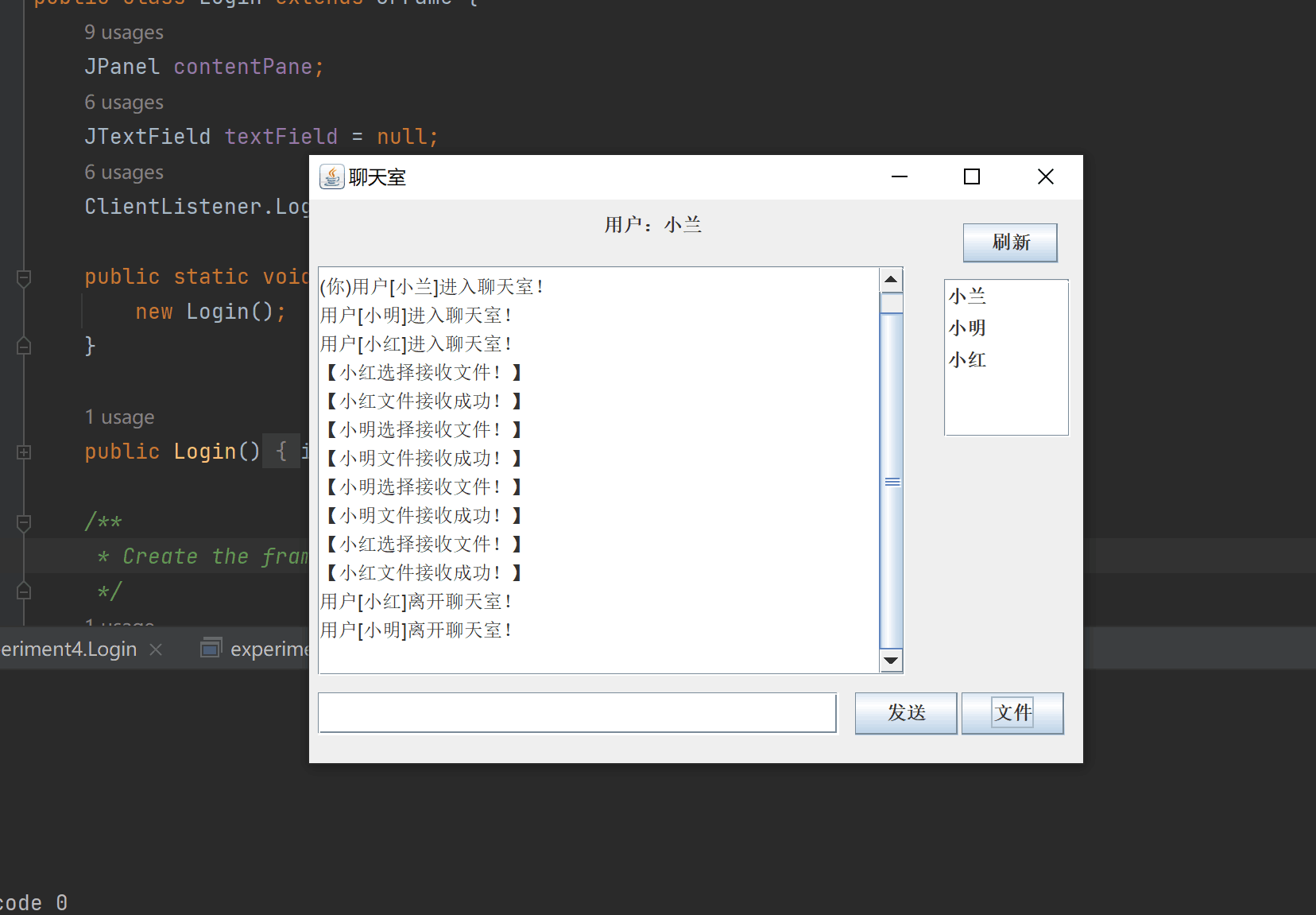
图片保存在相应的文件夹下：





### 4.1.6 退出

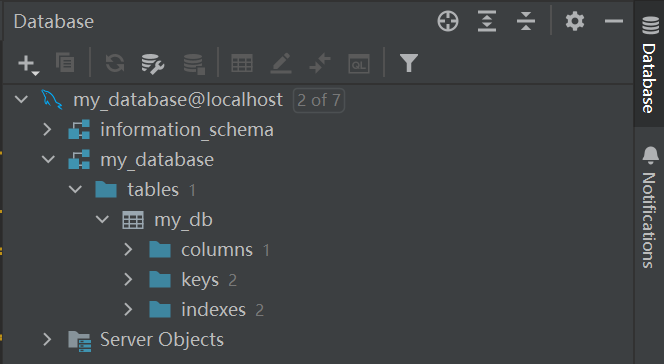
当有用户退出后，聊天室中会有提示：



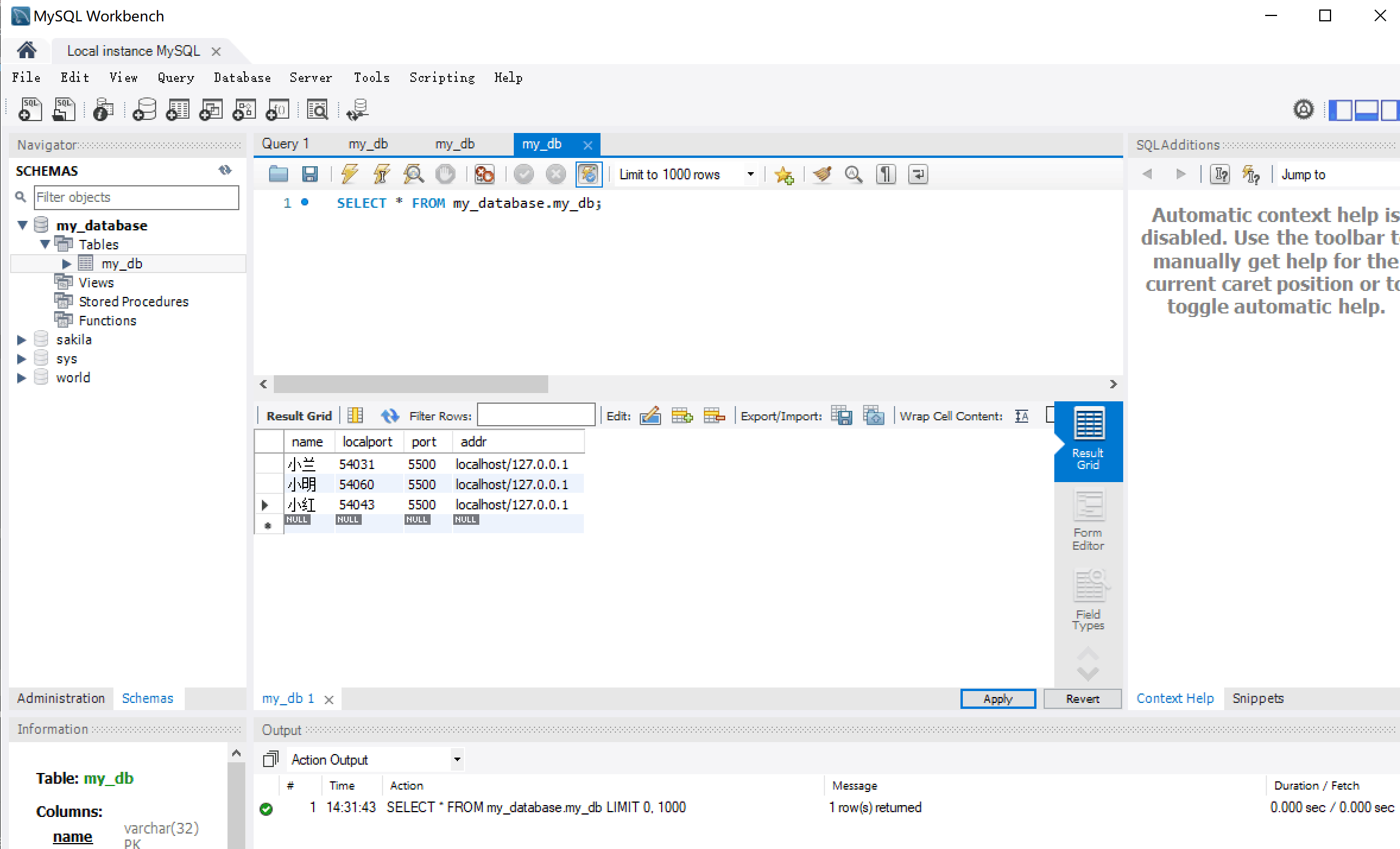
### 4.1.7 服务器端记录



### 4.1.8 数据库



数据库中保存了用户的name、localport、port和address:



# 5 总结

* 实现了所有基本需求：

1. 实现基于服务器转发的多个客户间的数据共享与交换。
2. 共享和交换的数据包括文字、图片和文件。
3. 服务器端记录当前在线客户列表，客户端能够获取该客户列表，并实时刷新。
4. 任一个客户可以和任意其它的客户进行交互。即支持一对一通信、组通信。

* 未来的工作

1. 在聊天界面实现更多功能，如用户分组、搜索用户、编辑用户和删除用户等。
2. 增强数据的共享和交换功能，实现选定用户批发送信息、文件、图片。