实验报告

目录

1 实验准备

1.1 实验环境

项目环境: Windows 10;
exe 运行环境: Windows 10, Oracle - JDK 1.8

```
C:\Users\Administrator>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

1.2 编程语言

编程语言: Java
语言版本: Oracle - JDK 1.8

1.3 技术准备

服务端: Socket 网络编程; 多线程处理; SQLite 数据库 CRUD; 文件读写
客户端: Socket 网络编程; 多线程处理; Java - swing 界面设计; 文件读写

2 实验设计

2.1 服务端设计

2.1.1 数据库设计

为了实现实验要求的好友管理,离线留言与记录和消息记录,服务端使用轻量级数据库 SQLite 来存储数据,选择该数据库的主要原因在于 SQLite 不需要依赖第三方软件,零配置情况下就能使用,方便在不同的运行环境下运行服务端.下面是数据库表的设计:

friend 表: 字段: user1,user1;
该表用来存储好友关系,好友 A 和 B 会被存为:
user1,user2
A, B
B, A

user1	user2
root	test
test	root
1234	root
root	1234
12345	1234
1234	12345

friendApplication 表,字段: sendUserName, getUserName, applicationId, time
该表用来存储好友申请，字段分别表示:
好友申请发送者，好友申请接收者，用来排序的 id 字段，申请时间

sendUserName	getUserName	applicationId	time
(Null)	(Null)	(Null)	(Null)

history 表,字段: sendUserName, getUserName, message, time, msg_id
该表用来存储历史消息,字段分别表示:
消息发送者，消息接收者，消息，发送时间, id 字段

sendUserName	getUserName	message	time	msg_id
1234	12345	123456789	08/06/2024:04:00:57	16
1234	1234	为什么删我好友	08/06/2024:22:11:29	17
1234	12345	为什么删我好友	08/06/2024:22:11:41	18
12345	1234	1234	09/06/2024:16:17:35	19
1234	12345	你好	09/06/2024:16:41:45	20

user 表，字段: userName
该表用来存储用户信息, userName 字段表示：用户名

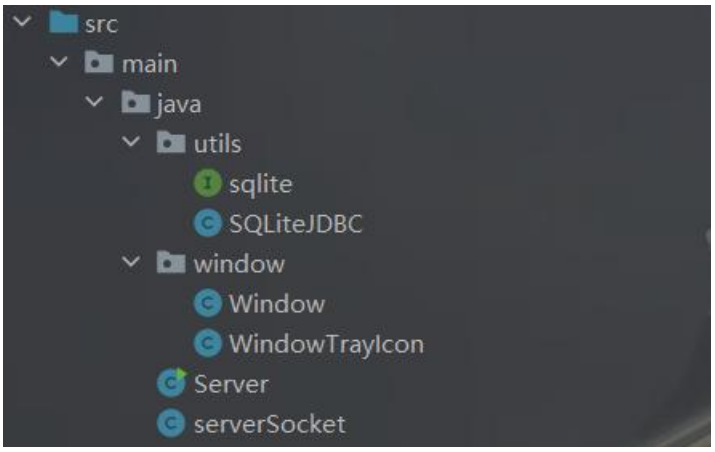
userName
root
test
1234
12345

waitToSend 表，字段: sendUserName, getUserName, message, time
该表用来存储待发送的离线消息,字段分别表示:
消息发送人，消息接收人，消息，发送时间

sendUserName	getUserName	message	time
(Null)	(Null)	(Null)	(Null)

2.1.2 框架设计

2.1.2.1 主要组件



Server 类

功能: 初始化服务器，处理客户端连接，管理线程池。

关键点:

使用 Swing 的 GUI 来获取服务器端口并启动服务器。

使用 ExecutorService 线程池来管理客户端连接，最多支持 20 个并发连接。

在接收到客户端连接后，创建 serverSocket 实例来处理每个客户端的通信。

serverSocket 类

功能: 处理客户端通信和业务逻辑。

关键点:

实现 Runnable 接口，负责读取客户端消息并执行相应的操作。

用户注册、登录处理，并通知好友用户的上线状态。

处理各种客户端请求，如私聊、群聊、查询历史消息、添加/删除好友等。

使用 ConcurrentHashMap 来管理在线用户的 Socket 连接。

与 SQLite 数据库交互以存储和检索用户数据和消息记录。

Window 类

功能: 提供图形用户界面来启动服务器并显示服务器运行状态。

关键点:

使用 Swing 组件构建界面，包括端口输入框、启动按钮、和显示服务器信息的文本区域。

设置窗口关闭事件，确保应用程序可以正确退出。

SQLiteJDBC 类

功能: 与 SQLite 数据库交互，提供数据持久化支持。

关键点:

包含各种方法用于用户和消息的增删查改操作。

sqlite 接口

功能: 定义了数据库操作的方法规范。

关键点:

包括用户注册、查询、好友管理、消息存储和检索等方法。

2.1.2.2 系统工作流程

服务器启动

用户在 GUI 中输入端口号并点击启动按钮。
Server 类创建一个新的线程并调用 execute()方法。
execute()方法创建 ServerSocket 并启动线程池来处理客户端连接。

客户端连接和通信

客户端连接到服务器，服务器接收连接并创建一个新的 serverSocket 实例来处理通信。
serverSocket 实例在单独的线程中运行，不断读取客户端消息并执行相应的操作。

业务逻辑处理

用户登录：验证用户身份并通知好友用户上线。
私聊：将消息发送给指定用户，如果用户不在线则存储离线消息。
群聊：将消息广播给所有在线用户。
查询历史消息：从数据库中检索并返回历史聊天记录。
添加/删除好友：更新好友关系并通知相关用户。

数据持久化

所有用户和消息数据存储在 SQLite 数据库中，通过 SQLiteJDBC 类进行操作。

关键技术点

多线程处理: 使用线程池来处理多个客户端连接，确保服务器高效运行。
Swing GUI: 提供图形用户界面来启动服务器和显示服务器信息。
Socket 通信: 使用 Socket 进行网络通信，实现客户端和服务器之间的数据传输。
SQLite 数据库: 使用 SQLite 数据库来存储用户数据和聊天记录，确保数据持久化。
并发管理: 使用 ConcurrentHashMap 来管理在线用户，保证线程安全。

2.1.3 功能设计

主要功能实现类:serverSocket

字段

SQLiteJDBC sqLiteJDBC：用于与 SQLite 数据库进行交互的工具类。

Map<String, Socket> map：用于存储用户名与其对应的 Socket 对象，使用 ConcurrentHashMap 保证线程安全。

Socket socket：当前处理的客户端 Socket。

JTextArea output：用于显示服务器端信息的 Swing 组件。

String userName：当前连接的用户名。

PrintStream printStream：用于向客户端发送信息的输出流。

PrintStream printStream2：用于发送系统消息的输出流。

SimpleDateFormat strFormat：用于格式化日期时间的工具。

构造方法

serverSocket(Socket serverSocket, JTextArea output)：初始化 Socket 和 JTextArea。

方法

run()：处理客户端的所有请求。包括用户登录、私聊、群聊、查询历史消息、好友管理等。

读取并解析客户端发送的消息，根据消息的类型执行相应的操作。

用户登录、用户私聊、查询历史消息、添加/删除好友、查询好友请求等操作。

userRegist(String userName, Socket socket)：注册用户信息，更新用户在线状态并通知在线好友。

将用户名和 Socket 存入 map 中。

向服务器输出组件记录用户上线信息。

groupChat(Socket socket, String msg)：处理群聊消息，将消息发送给所有在线用户（除了发送者）。

遍历 map，将消息发送给每一个在线用户。

privateChat(String userName, String msg)：处理私聊消息，将消息发送给指定用户。

如果目标用户在线，将消息直接发送。

如果目标用户离线，保存离线消息。

historyChat(Socket socket, String userName)：查询与某个用户的历史消息。

从数据库中查询最近的历史消息，并发送给客户端。

userExit(Socket socket)：处理用户退出操作。

将用户从 map 中移除。

通知用户的在线好友。

更新服务器输出组件显示用户下线信息。

功能点详解

用户登录：

消息格式：Login:username

注册用户信息，将用户加入在线用户列表，并通知在线好友用户上线。

从数据库中查询并发送用户的未读消息。

私聊：

消息格式：@username-消息内容

将私聊消息发送给指定用户，如果用户离线则保存为离线消息。

群聊：

消息格式：#@nobody-消息内容

将群聊消息发送给所有在线用户。

查询历史消息：

消息格式：#history-username
从数据库中查询与指定用户的历史消息，并发送给客户端。

好友管理：
添加好友：#addFriend-username
发送好友请求，如果目标用户在线则实时通知，否则保存为离线消息。
删除好友：#delete-username
从好友列表中删除指定用户。
查询好友请求：#queryFriendApplication
查询并显示所有未处理的好友请求。
同意好友请求：#passFirstFriendApplication
同意第一个好友请求，并将请求人加入好友列表。
拒绝好友请求：#refuseFirstFriendApplication
拒绝第一个好友请求，并通知请求人。

查询在线/离线好友：
查询在线好友：#queryOnlineFriend
显示所有在线的好友列表。
查询离线好友：#queryDeadFriend
显示所有离线的好友列表。

用户退出：
消息格式：#exit
将用户从在线列表中移除，并通知好友用户下线。

2.2 客户端设计

2.2.1 框架设计

2.2.1.1 主要组件

Client 类

功能：作为客户端程序的入口，负责初始化客户端并尝试连接服务器。

关键点： 使用无限循环确保登录界面可以重新显示，处理可能的异常。 创建 LoginFrame 实例来展示登录界面。

LoginFrame 类

功能：提供用户登录界面，允许用户输入连接信息并尝试连接服务器。

关键点： 集成 Swing 组件，创建输入 IP、端口和用户名的文本框以及登录按钮。 通过 addActionListener 为登录按钮添加事件监听，处理用户登录请求。

test 类

功能：作为客户端主界面，集成了私聊、群聊、文件传输和系统命令功能。

关键点： 实现 Runnable 接口，用于处理接收到的服务器消息和用户输入的命令。 使用 Swing 组件构建复杂的聊天界面，包括私聊和群聊的消息展示与输入。

私聊和群聊模块

功能：提供私聊和群聊的消息发送和接收功能。

关键点： 私聊模块允许用户选择聊天对象并发送消息，群聊模块允许用户向所有在线用户发送消息。 为发送按钮添加事件监听，实现消息的发送逻辑。

系统命令模块

功能：提供用户执行系统命令的界面，如查询好友、添加/删除好友等。

关键点： 为系统命令按钮添加事件监听，根据用户选择发送相应的命令到服务器。

文件传输功能

功能：为用户打开服务端/客户端的文件传输窗口,用户自行连接后便可传输文件,传输的文件存放于客户端的对应文件夹下。

关键点： 提供服务端和客户端文件传输的按钮，通过事件监听触发文件传输操作。

线程管理

功能：使用线程来处理网络通信，保证用户界面的响应性。

关键点： test 类在用户成功登录后，通过新线程启动用户与服务器的通信。

错误处理

功能：处理可能发生的异常，如 IOException。

关键点： 使用 try-catch 块捕获并处理异常，确保程序稳定运行。

用户体验

功能：通过界面反馈提升用户体验。

关键点： 在聊天区域显示系统消息，提示用户操作结果。

窗口事件

功能：监听窗口关闭事件，执行清理操作。

关键点： 在窗口关闭时，发送退出信号给服务器并退出程序。

2.1.2.2 系统工作流程

客户端启动

用户执行 Client 类 main 方法，展示登录界面。

用户登录

用户在 LoginFrame 中输入连接信息，点击登录后，通过 SIGN 方法尝试连接服务器。

通信建立

用户成功登录后，test 类的实例在新线程中启动，开始与服务器的通信。

消息处理

客户端接收服务器消息，在聊天界面展示。用户通过聊天输入框发送消息，通过事件监听处理消息发送。

文件传输

用户通过文件传输按钮触发文件传输操作，创建对应的服务端/客户端文件传输窗口。

关键技术点

Swing GUI：使用 Swing 组件构建图形用户界面，提供良好的用户体验。

Socket 通信：使用 Socket 实现客户端与服务器之间的网络通信。

多线程处理：使用线程确保网络通信不阻塞用户界面操作。

事件驱动：通过事件监听来响应用户操作，实现交互逻辑。

2.2.2 功能设计

主要功能实现类：test

字段

Socket socket：用于与服务器建立的通信 Socket。

String userName：标识当前客户端用户的用户名。

Box up、bottom、map 等：Swing 组件，用于构建窗体的布局。

JLabel、JTextArea、JTextField、JButton 等：Swing 组件，用于实现私聊、群聊和系统命令的用户界面。

构造方法

test(String userName, Socket socket)：通过用户名和 Socket 初始化客户端窗体。

方法

initJ()：初始化窗体组件，设置 GUI 布局和属性。

initBottom()：初始化窗体底部的系统命令按钮，并为它们添加事件监听器。

功能点详解

私聊发送： 消息格式："@" + 对方用户名 + "-" + 消息内容 用户输入对方用户名和消息后，点击发送，消息通过 sendDataToServerPrivate 方法发送给服务器。

群聊发送： 消息格式："#@nobody-" + 消息内容 用户在群聊输入框中输入消息并发送，消息通过 sendDataToServerPublic 方法广播给所有在线用户。

系统命令： 包括查询在线好友、查询离线好友、添加好友、删除好友、查看用户历史消息、查看好友申请、同意和拒绝好友申请等。 每个命令都通过 sendDataToServer 方法发送到服务器，执行相应的功能。

文件传输： 按钮"服务端传输文件"和"客户端传输文件"触发文件传输功能，创建 serverUserInputPanel 或 clientUserInputPanel 窗口并运行。

窗口关闭事件： 当用户关闭客户端窗口时，通过 addWindowListener 添加的事件监听器将调用 sendDataToServer 方法发送"#exit"命令给服务器，然后安全退出程序。

消息接收： run()方法在新线程中运行，不断监听服务器的响应。收到消息后，根据消息类型（私聊或群聊）更新聊天界面。

错误处理： 在发送数据和接收数据的过程中，使用 try-catch 块捕获并处理可能发生的 IOException 异常。

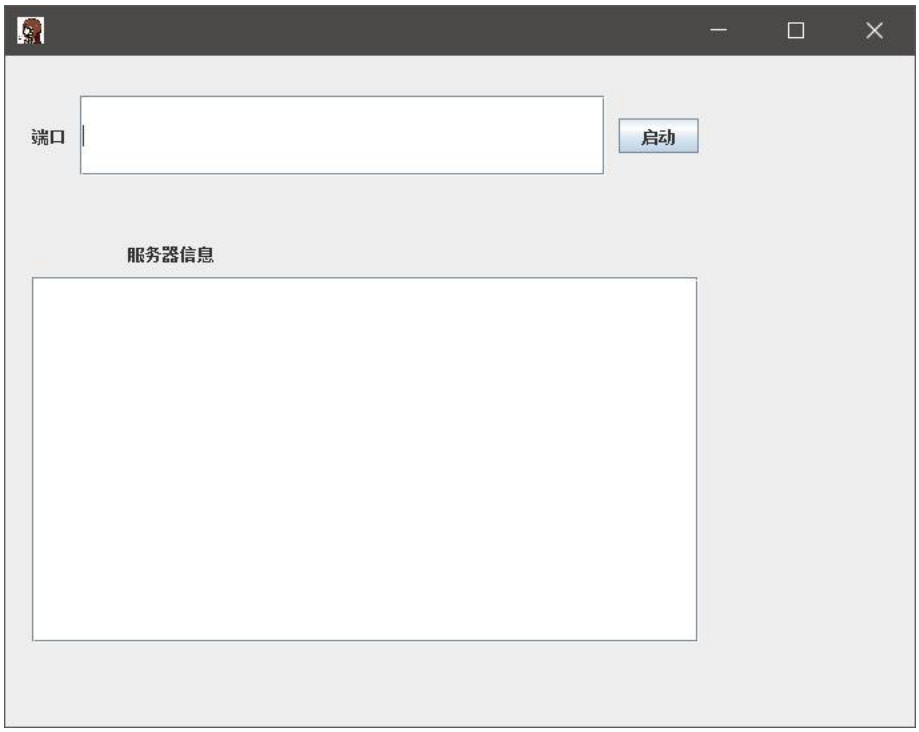
用户退出： 用户通过点击关闭窗口或发送"#exit"命令来退出客户端，此时将断开与服务器的连接，并在服务器端更新用户状态。

3 实现与测试

3.1 服务端实现与测试

3.1.1 服务端启动

```
private Window window = new Window();

Server(){

    window.portButton.addActionListener(new ActionListener(){

        @Override

        public void actionPerformed(ActionEvent a){

            Thread thread = new Thread(()->{

                execute();

            });window.portInput.setEditable(false);window.portButton.setEnabled(false);thread.
start();

        }

    });

    window.run();

}
```

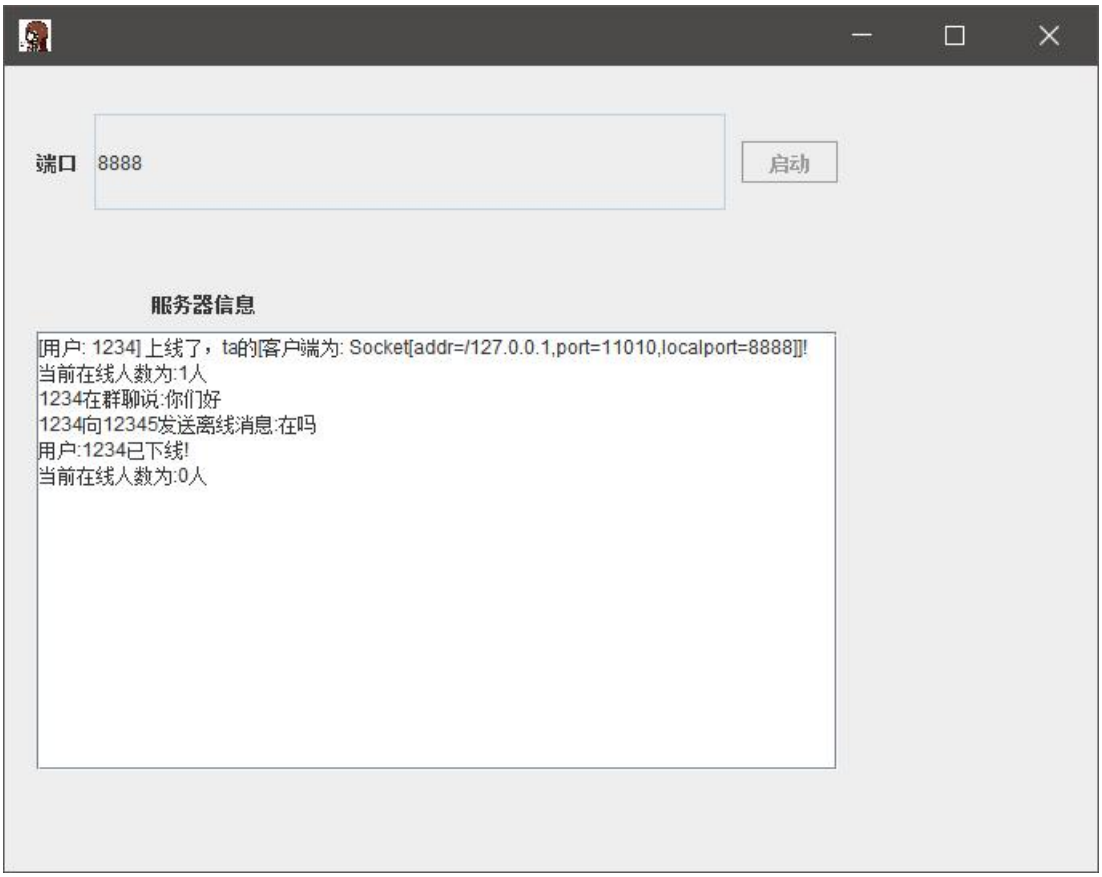


3.1.2 服务端在端口部署服务

```
1 个用法
public void execute(){
    ServerSocket serverSocket = null;     // 创建服务端套接字
    try {
        serverSocket = new ServerSocket(Integer.parseInt(window.portInput.getText()));
        // 创建线程池,从而可以处理多个客户端
        ExecutorService executorService= Executors.newFixedThreadPool( nThreads: 20);//最大20个
        int i = 0;
        while (true) {
            Socket socket = serverSocket.accept();     // 监听客户端的情况，等待客户端连接
            executorService.execute(new serverSocket(socket,window.output));     // 对每一个客户端，创建一个套接字处理器线程
        }
    } catch(IOException e) {
        e.printStackTrace();
    } finally {
        if (serverSocket != null) {
            try {
                serverSocket.close();     // 关闭serverSocket通道
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

3.1.3 服务端消息接收

```
@Override
public void run() {
    try {
        Scanner scanner = new Scanner(socket.getInputStream());     // 获取客户端的输入流
        String msg = null;     // 接收用户的信息
        while(true){
            if(scanner.hasNextLine()){
                msg = scanner.nextLine();     // 读取客户端传来的数据信息
                // 用户登录
                if(msg.startsWith("Login:")){...}
                else if(msg.startsWith("@") && msg.contains("-")){...}
                else if (msg.startsWith("#history-")) {...}
                else if (msg.startsWith("#delete-")) {...}
                else if (msg.startsWith("#addFriend-")) {...}
                else if (msg.equals("#queryFriendApplication")){...}
                else if (msg.equals("#passFirstFriendApplication")){...}
                else if(msg.equals("#refuseFirstFriendApplication")){...}
                else if (msg.equals("#queryOnlineFriend")) {...}
                else if (msg.equals("#queryDeadFriend")) {...}
                else if(msg.equals("#exit")){...}
                else if(msg.startsWith("#@nobody-")){...}
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ParseException e) {
        throw new RuntimeException(e);
    }
}
```



3.1.4 服务端数据库读写操作

```
2 个用法  1 个实现
public interface sqlite {
    1 个用法  1 个实现
    public int insertUser(String userName);
    1 个用法  1 个实现
    public boolean queryUserIsExist(String userName);
    1 个用法  1 个实现
    public List<List<String>> getHistoryChat(String getUserName,String sendUserName);//返回: 时间, 用户名, 消息
    1 个用法  1 个实现
    public void deleteFriend(String userName1,String userName2);
    1 个用法  1 个实现
    public void addFriend(String userName1,String userName2);
    4 个用法  1 个实现
    public List<String> queryFriend(String userName);
    1 个用法  1 个实现
    public int queryFriend(String userName1,String userName2);
    1 个用法  1 个实现
    public List<List<String>> getWaitChat(String getUserName);//返回: 时间, 用户名, 消息
    1 个用法  1 个实现
    public int insertHistoryMessage(String sendUserName,String getUserName,String message,String time);
    2 个用法  1 个实现
    public int insertWaitToSendMessage(String sendUserName,String getUserName,String message,String time);
    2 个用法  1 个实现
    void deleteFriendApplication(String userName1, String userName2);
    1 个用法  1 个实现
    List<List<String>> getFriendApplication(String getUserName);
    1 个用法  1 个实现
    int insertFriendApplication(String sendUserName, String getUserName, String time);
    2 个用法  1 个实现
    String getFriendApplicationOne(String getUserName);
}
```

sendUserName	getUserName	message	time
(Null)	(Null)	(Null)	(Null)

sendUserName	getUserName	message	time
1234	12345	在吗	09/06/2024:23:48:21

3.2 客户端实现与测试

3.2.1 客户端启动

```
public class Client {
    public static void main(String[] args) throws IOException, InterruptedException {
        //1.客户端连接服务器端,返回套接字Socket对象
        while (true) {
            try {
                LoginFrame loginFrame = new LoginFrame();
                break;
            }catch (Exception ignored){}
        }
    }
}
```

3.2.2 客户端登录及获取离线留言

服务端注册在线用户并读取数据库获取未读消息并发送

```
// 将用户名保存在userName中
String userName = msg.split( regex: "\\:")[1];    // 获取用户名
// 注册该用户
userRegist(userName,socket);
if(!sqLiteJDBC.queryUserIsExist(userName)){
    sqLiteJDBC.insertUser(userName);
}
//上线通知好友
List<String> friends = sqLiteJDBC.queryFriend(userName);
for(String i : friends){
    if(map.containsKey(i)){//在线才通知
        Socket client = map.get(i);    // 取得私聊用户名对应的客户端
        printStream = new PrintStream(client.getOutputStream());    // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println(userName + "上线啦!!!!!\n");
        printStream.println("--------------------------------\n");
        printStream.flush();
    }
}
//获取未读消息
List<List<String>> waitToGet = sqLiteJDBC.getWaitChat(userName);
printStream = new PrintStream(socket.getOutputStream());    // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
printStream.println("未读消息:\n");
for(List<String> i : waitToGet){
    printStream.println("----------------------------\n");
    printStream.printf("时间:%s\n%n", i.get(0));
    printStream.println("发信人:"+i.get(1)+"\n");
    printStream.println("消息:"+i.get(2)+"\n");
    printStream.println("----------------------------\n");
}printStream.println("----------------------------\n");
printStream.flush();
```

客户端创建用户界面

```
public void SIGN() throws IOException {
    PrintStream printStream = null;
    userName = name.getText();    // 获取用户输入的名称
    String strPort = port.getText();
    String strIP = IP.getText();
    if (userName.equals("")) {
        name.setText("用户名不能为空");    // 提示用户名不能为空
        name.setForeground(Color.gray);     // 提示文字颜色
    }else if (strIP.equals("")||strIP.equals("连接超时,请稍后再试!")) {
        IP.setText("IP地址不能为空");    // 提示用户名不能为空
        IP.setForeground(Color.gray);     // 提示文字颜色
    }else if (strPort.equals("")||strPort.equals("连接超时,请稍后再试!")) {
        port.setText("端口不能为空");    // 提示用户名不能为空
        port.setForeground(Color.gray);     // 提示文字颜色
    } else {
        Socket socket = new Socket(strIP, Integer.parseInt(strPort));
        String temp = "Login:" + userName;
        try {
            //1.获取服务器端的输出流
            printStream = new PrintStream(socket.getOutputStream());
            if (!userName.equals("")) {
                printStream.println(temp);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        this.setVisible(false);    // 登录窗口设为不可见
        // 启动客户端
        Thread client = new Thread(new test(userName, socket));
        client.start();
    }
}
```

3.2.3 客户端好友上/下线通知

客户端下线发送通知给服务端:

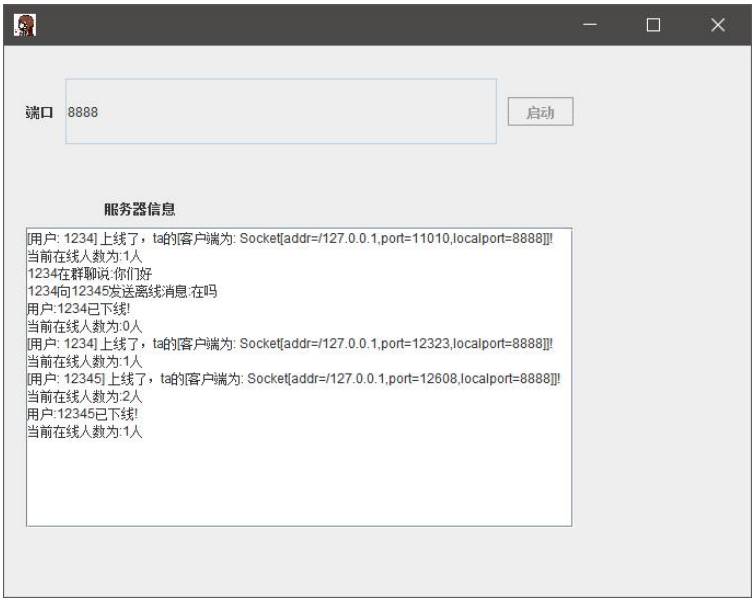
```
addWindowListener(new WindowAdapter() {

    public void windowClosing(WindowEvent e) {System.out.println("程序关

闭!");sendDataToServer("#exit");System.exit(0);}

});
```

服务端转发上下线通知给在线的好友:

```
//上线通知好友
List<String> friends = sqLiteJDBC.queryFriend(userName);
for(String i : friends){
    if(map.containsKey(i)){//在线才通知
        Socket client = map.get(i);     // 取得私聊用户名对应的客户端
        printStream = new PrintStream(client.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println(userName + "上线啦!!!!!\n");
        printStream.println("--------------------------------\n");
        printStream.flush();
    }
}
```

```java
private void userExit(Socket socket) throws IOException, ParseException {
    socket.close();
    map.remove(userName,socket);     // 将userName,Socket元素从map集合中删除

    //下线通知好友
    List<String> friends = sqLiteJDBC.queryFriend(userName);
    for(String i : friends){
        if(map.containsKey(i)){//在线才通知
            Socket client = map.get(i);     // 取得私聊用户名对应的客户端
            printStream = new PrintStream(client.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
            printStream.println(strFormat.format(new Date())+"\n");
            printStream.println(userName + "下线啦!!!!!\n");
            printStream.println("-------------------------------\n");
            printStream.flush();
        }
    }

    // 提醒服务器该客户端已下线
    output.append("用户:"+ userName +"已下线!\n");
    output.append("当前在线人数为:" + map.size() + "人\n");
}
```

测试效果



3.2.4 客户端即时文字交流(一对一及一对多聊天)

3.2.4.1 私聊

客户端发送私聊对象和消息到服务端

```
privateChatInputSendMsg.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if(privateChatInputUserNameInput.getText()!=""&&privateChatInputMsgInput.getText()!=""){
            String msg = "@" + privateChatInputUserNameInput.getText() + "-" + privateChatInputMsgInput.getText();
            privateChatInputMsgInput.setText("");
            sendDataToServerPrivate(msg);
        }
        else{
            privateChatOutputArea.append(new Date().toString() + "\n");   // 在自己的聊天界面中显示
            privateChatOutputArea.append("#系统消息\n发送对象/消息为空,发送失败!\n");   // 在自己的聊天界面中显示
        }
    }
});
```

```
1 个用法
private void sendDataToServerPrivate(String text){
    privateChatOutputArea.append(new Date().toString() + "\n");   // 在自己的聊天界面中显示
    privateChatOutputArea.append("#"+userName+": " + text + "\n");   // 在自己的聊天界面中显示
    try {
        // 发送数据
        printStream = new PrintStream(socket.getOutputStream());
        printStream.println(text);
        printStream.flush();
        // 清空自己当前的会话框
        privateChatInputMsgInput.setText("");
    } catch (IOException el) {
        el.printStackTrace();
    }
}
```

私聊对象在线时:服务端转发私聊对象消息并储存消息到历史消

息表中;不在线时:服务端不转发私聊对象消息而是将消息储存在待发送消息表中

```java
/**
 * 私聊流程(利用userName取得客户端的Socket对象,从而取得对应输出流,将私聊信息发送到指定客户端)
 * @param userName 私聊的用户名
 * @param msg 私聊的信息
 */
2个用法
private void privateChat(String userName, String msg) throws IOException, ParseException {
    String curUser = this.userName;
    Date date = new Date();
    if(map.containsKey(userName)){
        Socket client = map.get(userName);    // 取得私聊用户名对应的客户端
        printStream = new PrintStream(client.getOutputStream());    // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println(curUser + "悄悄对你说:" + msg +"\n");
        printStream.println("--------------------------------\n");
        output.append(date.toString());
        output.append(curUser + "私聊" + userName + "说:" + msg +"\n");    // 服务器端显示,用于调试
        sqLiteJDBC.insertHistoryMessage(curUser,userName,msg, strFormat.format(new Date()));
    }else{//用户离线,发送离线消息
        output.append(curUser + "向" + userName + "发送离线消息:" + msg +"\n");    // 服务器端显示,用于调试
        sqLiteJDBC.insertWaitToSendMessage(curUser,userName,msg, strFormat.format(new Date()));
    }
}
```

3.2.4.2 聊天室聊天

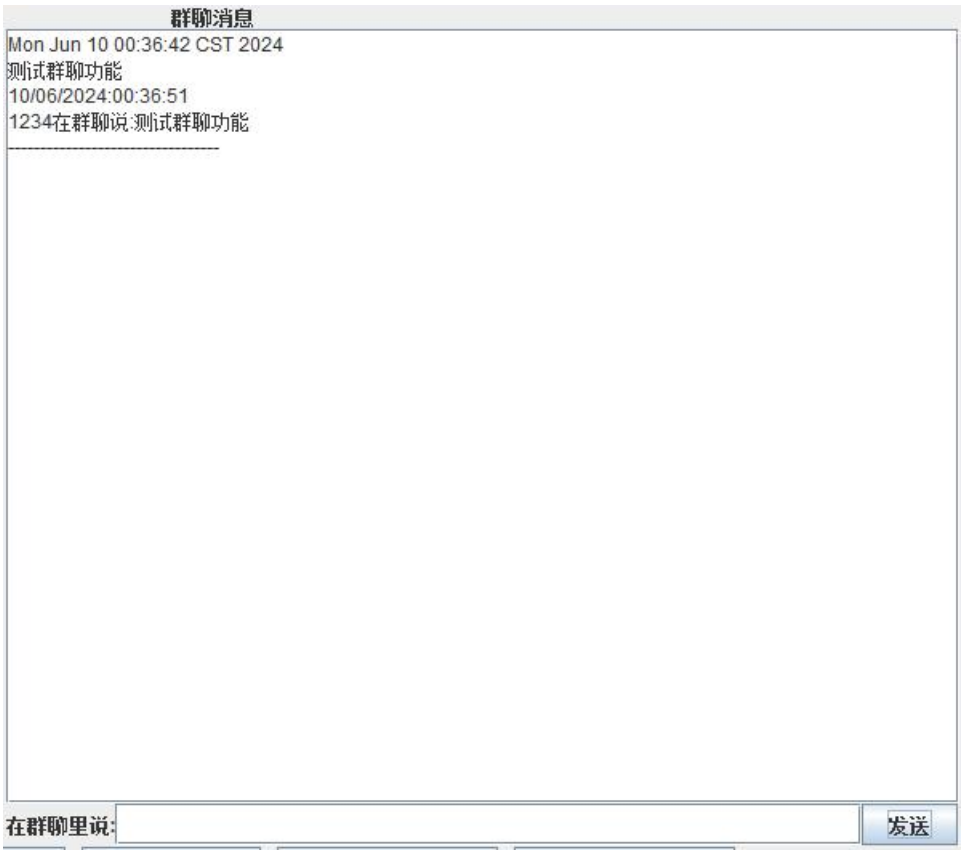
客户端发送信息到服务端,注意信息前缀与私聊时不同

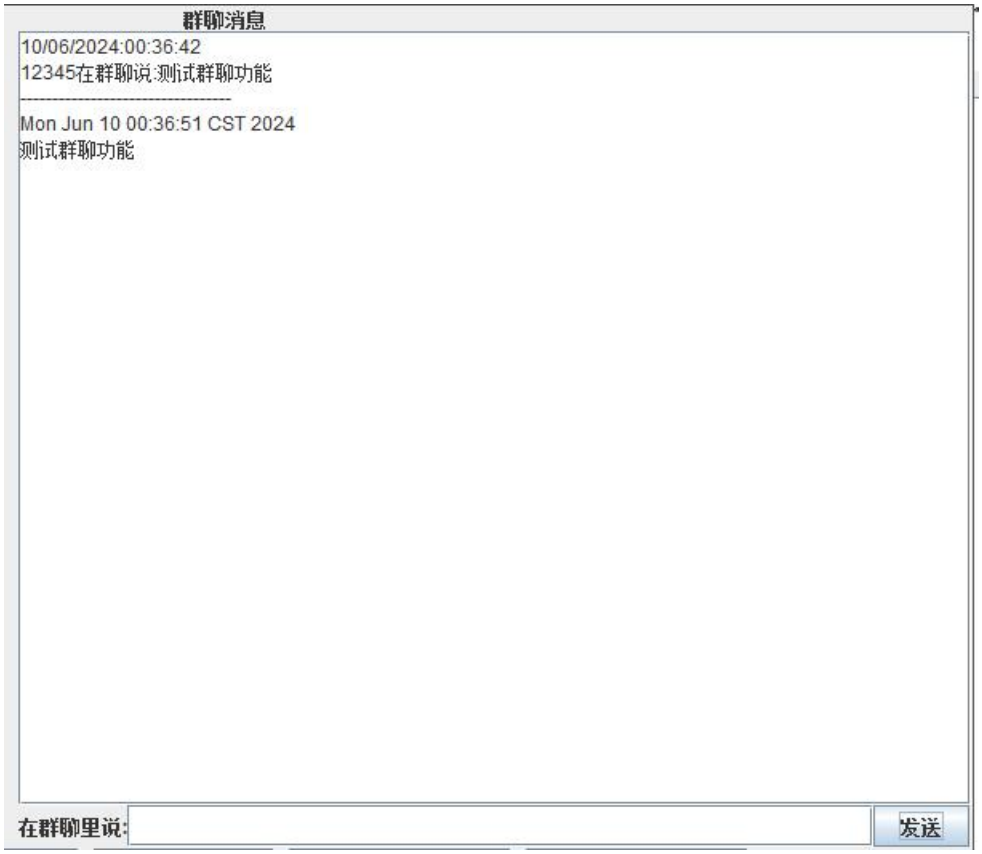
```java
publicChatInputSend.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if(publicChatInputMsgInput.getText()!=""){
            String msg = "#@nobody-" + publicChatInputMsgInput.getText();
            publicChatOutputArea.append(new Date().toString() + "\n");    // 在自己的聊天界面中显示
            publicChatOutputArea.append(publicChatInputMsgInput.getText() + "\n");    // 在自己的聊天界面中显示
            sendDataToServerPublic(msg);
        }
        else{
            publicChatOutputArea.append(new Date().toString() + "\n");    // 在自己的聊天界面中显示
            publicChatOutputArea.append("#系统消息\n发送消息为空,发送失败!\n");    // 在自己的聊天界面中显示
        }
    }
});
```

服务端转发信息到所有在线用户

```
else if(msg.startsWith("#@nobody-")){// 群聊信息
    String msg1 = msg.split( regex: "#@nobody-")[1];
    groupChat(socket, msg1);
}
```

```
/**
 * 群聊流程(将Map集合转换为Set集合,从而取得每个客户端Socket,将群聊信息发送给每个客户端)
 * @param socket 发出群聊的客户端
 * @param msg 群聊信息
 */
1 个用法
private void groupChat(Socket socket,String msg) throws IOException, ParseException {
    String userName = this.userName;    // 遍历Set集合找到发起群聊信息的用户
    Set<Map.Entry<String,Socket>> set = map.entrySet();    // 将Map集合转换为Set集合
    output.append(userName + "在群聊说:" + msg +"\n");    // 在服务器上显示，用于调试
    // 遍历Set集合将群聊信息发给每一个客户端(除了自己以外)
    for(Map.Entry<String,Socket> entry : set){
        //取得客户端的Socket对象
        if (!entry.getValue().equals(socket)) {
            Socket client = entry.getValue();
            printStream = new PrintStream(client.getOutputStream());    //取得client客户端的输出流
            printStream.println("public:");
            printStream.println(strFormat.format(new Date())+"\n");
            printStream.println("public:");
            printStream.println(userName + "在群聊说:" + msg +"\n");
            printStream.println("public:");
            printStream.println("--------------------------------\n");
        }
    }
}
```





3.2.5 历史消息记录

服务端将即时发送的私聊消息转发的同时将消息记录在历史消息表

```
2 个用法
private void privateChat(String userName, String msg) throws IOException, ParseException {
    String curUser = this.userName;
    Date date = new Date();
    if(map.containsKey(userName)){
        Socket client = map.get(userName);    // 取得私聊用户名对应的客户端
        printStream = new PrintStream(client.getOutputStream());    // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println(curUser + "悄悄对你说:" + msg +"\n");
        printStream.println("-------------------------------\n");
        output.append(date.toString());
        output.append(curUser + "私聊" + userName + "说:" + msg +"\n");    // 服务器端显示，用于调试
        sqLiteJDBC.insertHistoryMessage(curUser,userName,msg, strFormat.format(new Date()));
    }else{//用户离线,发送离线消息
        output.append(curUser + "向" + userName + "发送离线消息:" + msg +"\n");    // 服务器端显示，用于调试
        sqLiteJDBC.insertWaitToSendMessage(curUser,userName,msg, strFormat.format(new Date()));
    }
}
```

客户端发送查看历史消息请求

```
button5.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if(!Objects.equals(privateChatInputUserNameInput.getText(),  b: ""))
            sendDataToServer( text: "#history-"+privateChatInputUserNameInput.getText());
        else{
            privateChatOutputArea.append(new Date().toString() + "\n");    // 在自己的聊天界面中显示
            privateChatOutputArea.append("#系统消息\n请求对象为空,发送失败!\n");    // 在自己的聊天界面中显示
        }
    }
});
```

服务端返回对应客户端和对应用户的历史消息

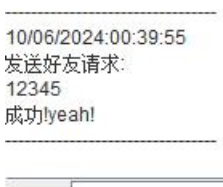
```
else if (msg.startsWith("#history-")) { //用户查询与某用户的历史消息,格式为:#history-userName
    String sendUserName = msg.split( regex: "#history-")[1];
    // 发送历史信息
    historyChat(socket,sendUserName);
}
```

```
private void historyChat(Socket socket,String userName) throws IOException {
    String curUser = this.userName;
    List<List<String>> historyMsg = sqLiteJDBC.getHistoryChat(curUser,userName);
    printStream = new PrintStream(socket.getOutputStream());    // 获取客户端的输出流,将信息发送到指定客户端
    for (List<String> i : historyMsg){
        System.out.println(i.get(0));
        System.out.println(i.get(1));
        System.out.println(i.get(2));
        printStream.println(i.get(0)+"\n"+i.get(1) + " 对你说: " + i.get(2) +"\n");
        printStream.println("--------------------------------\n");
    }
    output.append(curUser + "查询了ta跟"+userName+"的最近十条历史消息\n");    // 服务器端显示，用于调试
}
```

3.2.6 好友管理

3.2.6.1 发送好友请求

客户端发送请求对象到服务端




```
button3.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if(!Objects.equals(privateChatInputUserNameInput.getText(),  b: ""))
            sendDataToServer( text: "#addFriend-"+privateChatInputUserNameInput.getText());
        else{
            privateChatOutputArea.append(new Date().toString() + "\n");    // 在自己的聊天界面中显示
            privateChatOutputArea.append("#系统消息\n请求对象为空,发送失败!\n");    // 在自己的聊天界面中显示
        }
    }
});
```

服务端将好友申请添加到好友申请表中, 若请求对象在线则发送好友申请通知, 不在线则存入离线消息

```
else if (msg.startsWith("#addFriend-")) { //用户添加某好友,格式为:#addFriend-userName
    String addUserName = msg.split( regex: "#addFriend-")[1];
    printStream = new PrintStream(socket.getOutputStream());    // 获取客户端的输出流,将信息发送到指定客户端
    sqLiteJDBC.insertFriendApplication(userName,addUserName,strFormat.format(new Date()));
    if(map.containsKey(addUserName)){
        Socket client = map.get(addUserName);    // 取得私聊用户名对应的客户端
        printStream2 = new PrintStream(client.getOutputStream());    // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream2.println(strFormat.format(new Date())+"\n");
        printStream2.println("#系统消息:\n");
        printStream2.println(userName+"请求添加你为好友!");
        printStream.println("--------------------------------\n");
        printStream2.flush();
    }else{
        sqLiteJDBC.insertWaitToSendMessage( sendUserName: "#系统消息:",addUserName, message: userName+"请求添加你为好友!",strFormat.format(new
    }
    printStream.println(strFormat.format(new Date())+"\n");
    printStream.println("发送好友请求: "+addUserName+" 成功!yeah!\n");
    printStream.println("--------------------------------\n");
    printStream.flush();
    output.append(userName+"请求成为"+addUserName+"的好友\n");
}
```

sendUserName	getUserName	applicationId	time
1234	12345	8	10/06/2024:00:39:55

3. 2. 6. 2 查看好友请求

客户端发送请求到服务端

```
button6.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { sendDataToServer( text: "#queryFriendApplication"); }
});
```

服务端查询数据库并返回未处理的好友请求

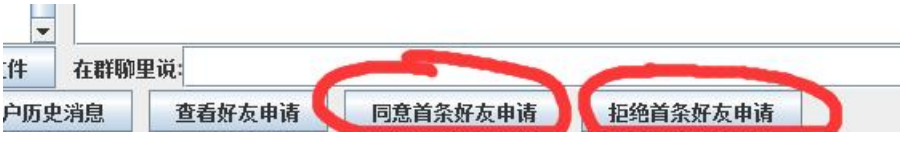
```
else if (msg.equals("#queryFriendApplication")){
    List<List<String>> friends = sqLiteJDBC.getFriendApplication(userName);
    printStream = new PrintStream(socket.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
    printStream.println(strFormat.format(new Date())+"\n");
    printStream.println("#系统消息:\n");
    printStream.println("好友请求:\n");
    for(List<String> i : friends){
        printStream.println("时间:"+i.get(0)+"\n");
        printStream.println("请求人:"+i.get(1)+"\n");
    }
    printStream.println("---------------------------------\n");
    printStream.flush();
}
```

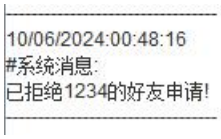
3.2.6.3 同意/拒绝好友请求

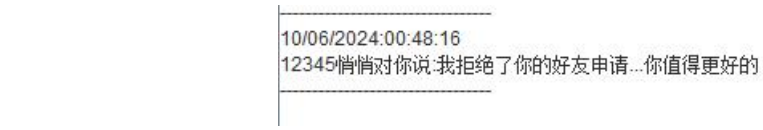
客户端发送同意/拒绝首条好友请求消息到服务端

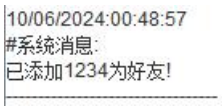
```
button7.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { sendDataToServer( text: "#passFirstFriendApplication"); }
});
button8.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { sendDataToServer( text: "#refuseFirstFriendApplication"); }
});
```

服务端删除好友请求表里的消息并根据同意/拒绝决定是否在好友表中添加好友关系










```
else if (msg.equals("#passFirstFriendApplication")){
    String sendUserName = sqLiteJDBC.getFriendApplicationOne(userName);
    if(sendUserName == null){
        printStream = new PrintStream(socket.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println("#系统消息:\n");
        printStream.println("很可惜...没有人请求添加你为好友哦^_^\n");
        printStream.println("---------------------------------\n");
    }else{
        sqLiteJDBC.deleteFriendApplication(userName,sendUserName);
        if(sqLiteJDBC.queryFriend(sendUserName,userName)==0)
            sqLiteJDBC.addFriend(userName,sendUserName);
        printStream = new PrintStream(socket.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println("#系统消息:\n");
        printStream.println("已添加"+sendUserName+"为好友!"+"\n");
        printStream.println("---------------------------------\n");
    }
}
```

```
else if(msg.equals("#refuseFirstFriendApplication")){
    String sendUserName = sqLiteJDBC.getFriendApplicationOne(userName);
    if(sendUserName == null){
        printStream = new PrintStream(socket.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println("#系统消息:\n");
        printStream.println("很可惜...没有人请求添加你为好友哦^_^\n");
        printStream.println("---------------------------------\n");
    }else{
        sqLiteJDBC.deleteFriendApplication(userName,sendUserName);
        printStream = new PrintStream(socket.getOutputStream());     // 获取私聊客户端的输出流,将私聊信息发送到指定客户端
        printStream.println(strFormat.format(new Date())+"\n");
        printStream.println("#系统消息:\n");
        printStream.println("已拒绝"+sendUserName+"的好友申请!"+"\n");
        printStream.println("---------------------------------\n");
        privateChat(sendUserName, msg: "我拒绝了你的好友申请...你值得更好的");
    }
}
```

3.2.6.4 查看在线/离线好友

客户端发送请求,服务端查询数据库和在线用户 map 后返回列表

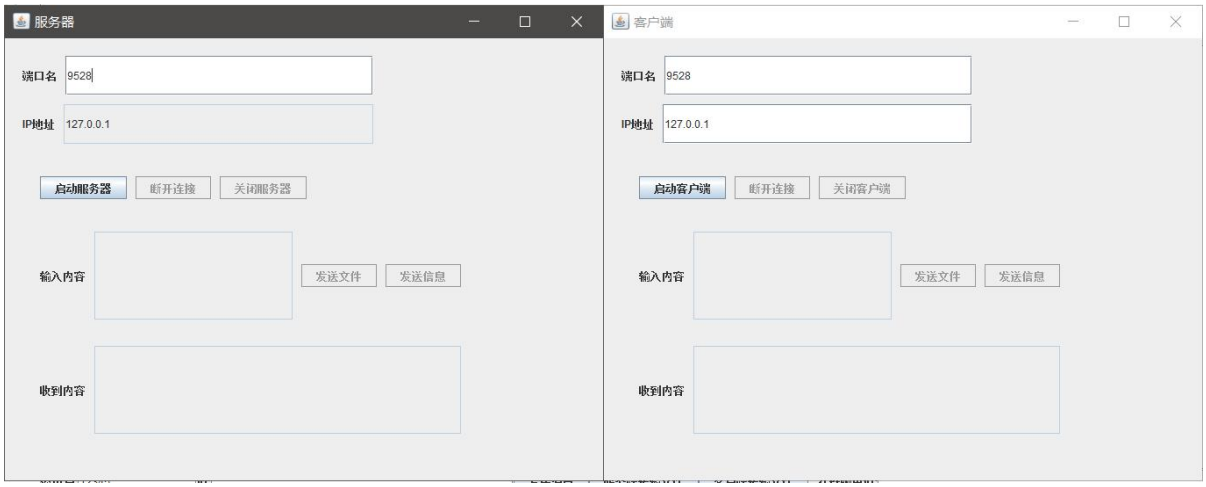
```
button1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { sendDataToServer( text: "#queryOnlineFriend"); }
});
button2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { sendDataToServer( text: "#queryDeadFriend"); }
});
```

```
else if (msg.equals("#queryOnlineFriend")) { //用户查询在线好友
    List<String> friends = sqLiteJDBC.queryFriend(userName);
    printStream = new PrintStream(socket.getOutputStream());     // 获取客户端的输出流,将信息发送到指定客户端
    printStream.println(strFormat.format(new Date())+"\n");
    printStream.println("#系统消息:"+"\n");
    printStream.println("在线好友:\n");
    printStream.flush();
    for(String i : friends){
        if(map.containsKey(i)){
            printStream.println(i);
        }
    }
    printStream.println("--------------------------------\n");
    printStream.flush();
}
else if (msg.equals("#queryDeadFriend")) { //用户查询离线好友
    List<String> friends = sqLiteJDBC.queryFriend(userName);
    printStream = new PrintStream(socket.getOutputStream());     // 获取客户端的输出流,将信息发送到指定客户端
    printStream.println(strFormat.format(new Date())+"\n");
    printStream.println("#系统消息:"+"\n");
    printStream.println("离线好友:"+"\n");
    printStream.flush();
    for(String i : friends){
        if(!map.containsKey(i)){
            printStream.println(i);
        }
    }
    printStream.println("--------------------------------\n");
    printStream.flush();
}
```

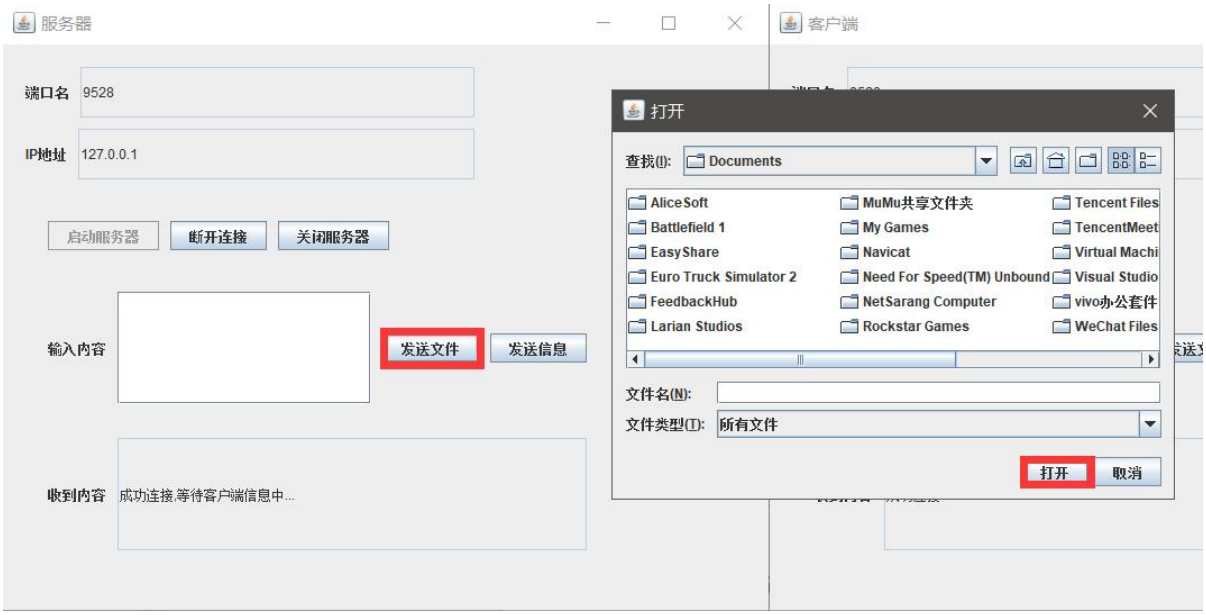
3.2.7 客户端即时文件收发

客户端点击 服务端传输文件/客户端传输文件 ，就会打开一个作为服务端/客户端传输文件的新窗口

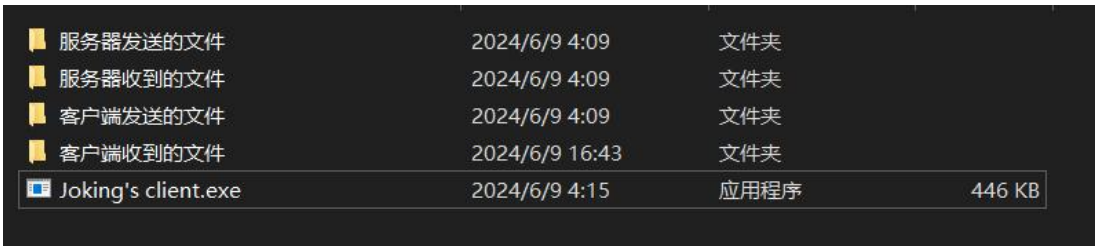
```java
// 文件发送按钮监听
privateChatInputSendFileServer.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        serverUserInputPanel serverUserInputPanel = new serverUserInputPanel();
        serverUserInputPanel.run();
    }
});
privateChatInputSendFileClient.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        clientUserInputPanel clientUserInputPanel = new clientUserInputPanel();
        clientUserInputPanel.run();
    }
});
```

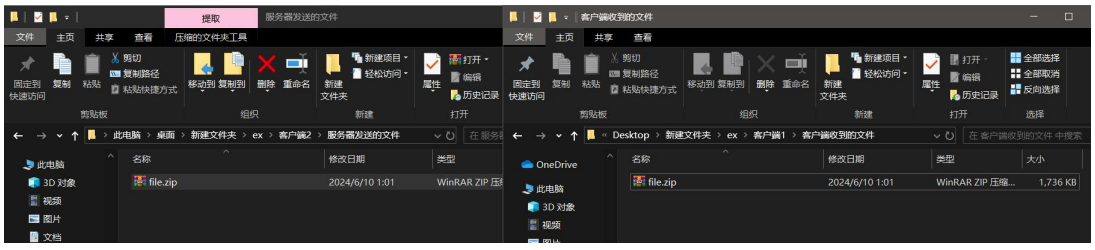


当传输文件的服务器和客户端输入对应的 IP 地址和端口并且客户端在服务器启动的 5 秒内启动时，服务器和客户端就会连接上，此时便可进行文件传输(同时也可以进行即时文字交流)



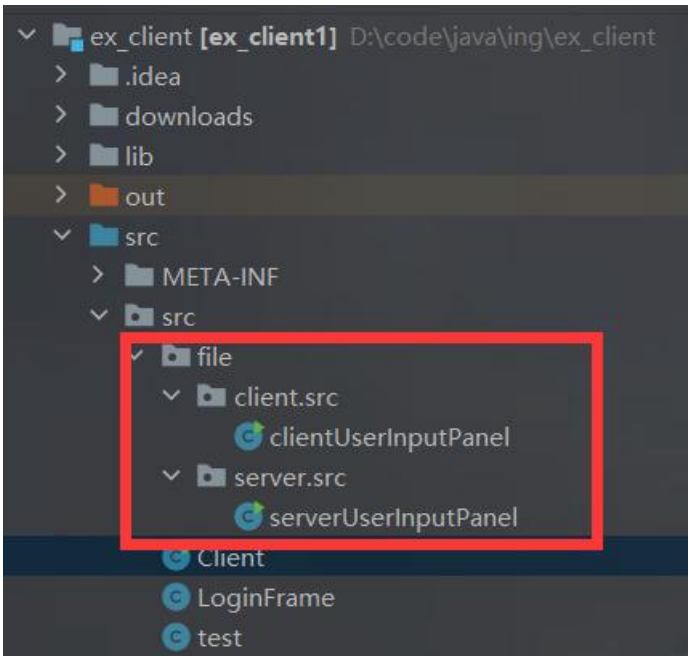
点击发送文件并选择文件后,发送方可以在 服务器/客户端发送的文件 文件夹下的 file.zip 找到发送的文件,接收方可以在 服务器/客户端接收的文件 文件夹下的 file.zip 找到发送的文件





点击关闭窗口即可安全关闭连接并退出

传输文件的代码在客户端代码的 file 目录下



代码先将文件打包成压缩包后使用字节流传输实现文件传输

发送文件:

```java
public void sendFile(String fileName) throws IOException {
    if (output != null) {
        // 创建压缩文件
        // 获取当前工作目录的路径
        String currentDirectory = System.getProperty("user.dir");
        // 构造压缩文件的路径，这里假设压缩文件名为 "file.zip"
        String zipFilePath = currentDirectory + File.separator + "\\服务器发送的文件\\file.zip";
        File zipFile = new File(zipFilePath);FileOutputStream fos = new FileOutputStream(zipFile);
        ZipOutputStream zos = new ZipOutputStream(fos);
        // 要压缩的文件
        File fileToZip = new File(fileName);
        // 添加文件到压缩包
        byte[] buffer = new byte[1024];
        ZipEntry zipEntry = new ZipEntry(fileToZip.getName());zos.putNextEntry(zipEntry);
        FileInputStream fis = new FileInputStream(fileToZip);
        int length;
        while ((length = fis.read(buffer)) > 0) {zos.write(buffer, off: 0, length);}
        // 完成压缩
        zos.closeEntry();zos.close();
        // 发送压缩文件
        FileInputStream zis = new FileInputStream(zipFile);
        DataOutputStream dos = new DataOutputStream(client.getOutputStream());
        byte[] bytes = new byte[1024];
        while ((length = zis.read(bytes)) != -1) {
            //System.out.println("i" + i + " res: " + res);
            byte[] newByteArr = reviseArr(bytes, length);;
            newByteArr[1] = 1;  // 表示第二个位置上的值为1时表示传输的是文件
            dos.write(newByteArr, off: 0, len: length+2);
            dos.flush();
        }
        output.flush(); // 确保消息发送完毕
    }
```

接收文件：

```java
if (bytes[1] == 1) // 说明传的是文件
{
    //String filePath = String.format("./directoryTest/src/用户%d传送来的IO流的框架图.png", sendUser);
    try {
        String currentDirectory = System.getProperty("user.dir");
        String FilePath = currentDirectory + File.separator + "\\服务器收到的文件\\file.zip";
        bosFile = new BufferedOutputStream(new FileOutputStream(FilePath, append: true));
        //System.out.println(bosFile.toString());
        bosFile.write(bytes, off: 2, len: read-2);
        bosFile.flush();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    if (read<1026)    // 说明是最后一次在传送文件，所以传送的字节数才会小于字节数组by的大小
    {
        clientOutput.setText("用户发送的文件传输完成\n");
    }
```

4 参考文献

基于 C/S 模式的简单聊天程序

https://www.cnblogs.com/Liu-xing-wu/p/14320376.html

Java 版——一个简易的 QQ 聊天室程序

https://www.cnblogs.com/pengsay/p/14659608.html

超详细带你用 Java 实现 QQ 的聊天功能

https://developer.aliyun.com/article/1041926

Java Socket：飞鸽传书的网络套接字 | 二哥的 Java 进阶之路 (javabetter.cn)

https://javabetter.cn/socket/socket.html_01%E3%80%81ping-%E4%B8%8E-telnet

循序渐进 Java Socket 网络编程（多客户端、信息共享、文件传输）

https://www.cnblogs.com/feijian/p/4461087.html