浙江大学

本科实验报告

课程名称：	Java 应用技术
实验名称：	Project2-多客户端的纯文本聊天服务器
姓　　名：	姚昊辰
学　　院：	计算机学院
系：	计算机
专　　业：	计算机科学与技术
学　　号：	3210102946
指导教师：	翁恺

2024 年 1 月 1 日

浙江大学实验报告

实验名称：Project2-多客户端的纯文本聊天服务器 实验类型：设计实验

一、 实验内容：

- 实现一个多客户端的纯文本聊天服务器，能同时接受多个客户端的连接，并将任意一个客户端发送的文本向所有客户端（包括发送方）转发。

二、 源码及注释

- Client

```
public class Client{
    Run | Debug
    public static void main(String[] args) throws IOException {
        // 创建一个BufferedReader对象，用于读取用户的输入
        BufferedReader userInputReader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print(s:"Enter server IP address: ");
        // 读取用户输入的服务器IP地址
        String serverIP = userInputReader.readLine();
        System.out.print(s:"Enter server port number: ");
        // 读取用户输入的服务器端口号，并将其转换为整数
        int serverPort = Integer.parseInt(userInputReader.readLine());
        // 创建一个Socket对象，用于与服务器建立连接
        Socket socket = new Socket(serverIP, serverPort);
        System.out.print(s:"Enter your username: ");
        // 创建一个新的BufferedReader对象，用于读取用户的输入
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        // 创建一个PrintWriter对象，用于向服务器发送数据
        PrintWriter out = new PrintWriter(socket.getOutputStream());
        // 创建一个BufferedReader对象，用于从服务器接收数据
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        // 创建一个新的线程，用于不断地从服务器读取数据并将其打印到控制台
```

```
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    while (true){
                        System.out.println(in.readLine());
                    }
                } catch (IOException e) {
                    // 如果在读取数据时发生异常，打印异常的堆栈跟踪
                    e.printStackTrace();
                }
            }
        }).start();
        // 读取用户的输入，并将其发送到服务器，直到用户输入"quit"为止
        String line = reader.readLine();
        while (!"quit".equalsIgnoreCase(line)){
            out.println(line);
            out.flush();
            line = reader.readLine();
        }
        // 关闭PrintWriter、BufferedReader和Socket，以释放网络资源
        out.close();
        in.close();
        socket.close();
    }
}
```

- Server

```java
public class Server {
    // 创建一个线程安全的List，用于存储所有已连接的客户端的Socket对象
    protected static List<Socket> sockets = new Vector<>();
    Run | Debug
    public static void main(String[] args) throws IOException {
        // 创建一个ServerSocket对象，监听2946端口
        ServerSocket server = new ServerSocket(port:2946);
        boolean flag = true;
        System.out.println(x:"Server listening...\n");
        while (flag){
            try {
                // 接受客户端的连接请求，返回一个代表客户端的Socket对象
                Socket accept = server.accept();
                // 将新连接的客户端的Socket对象添加到sockets列表中
                synchronized (sockets){
                    sockets.add(accept);
                }
                // 显示所有已连接的客户端
                displayConnectedClients();
                // 为每个新连接的客户端创建一个新的线程，处理客户端的请求
                Thread thread = new Thread(new ServerThread(accept));
                thread.start();
            }catch (Exception e){
                // 如果在接受连接或处理请求时发生异常，结束循环，停止接受新的连接
                flag = false;
                e.printStackTrace();
            }
        }
        // 关闭ServerSocket，释放资源
        // 关闭ServerSocket，释放资源
        server.close();
    }
    // 显示所有已连接的客户端的信息
    public static void displayConnectedClients() {
        System.out.println(x:"Connected Clients:");
        // 遍历sockets列表，打印每个客户端的地址和端口信息
        synchronized (sockets) {
            for (Socket clientSocket : sockets) {
                System.out.println("Client@" + clientSocket.getRemoteSocketAddress());
            }
        }
        System.out.println(x:"--------------------\n");
    }
}
```

- ServerThread

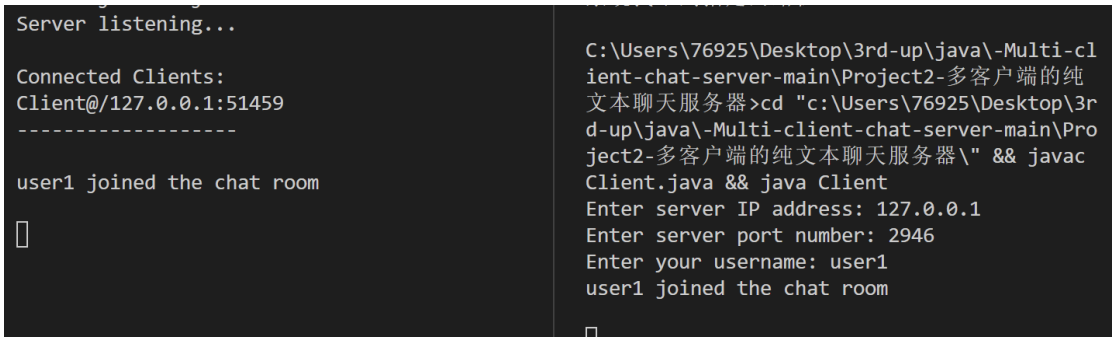
```java
public class ServerThread extends Server implements Runnable{
    // 用于存储所有已经使用的用户名
    private static Set<String> usernames = new HashSet<>();
    Socket socket;
    String username;
    public ServerThread(Socket socket){
        // 初始化socket
        this.socket = socket;
    }
    @Override
    public void run() {
        try {
            // 创建BufferedReader用于读取客户端发送的消息
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            // 读取并设置用户名
            setUsername(reader.readLine());
            // 检查用户名是否唯一，如果不唯一则要求用户更改用户名
            while (!isUsernameUnique()) {
                String msg="Username " + username + " is already taken. Please change your name.";
                PrintWriter out = null;
                synchronized (sockets){
                    out = new PrintWriter(socket.getOutputStream());
                    out.println(msg);
                    out.flush();
                }
                setUsername(reader.readLine());
            }
            // 用户名唯一，用户加入聊天室
            System.out.println(username+" joined the chat room\n");
            print(username+" joined the chat room\n");
```

```
            // 用户名唯一，用户加入聊天室
            System.out.println(username+" joined the chat room\n");
            print(username+" joined the chat room\n");
            boolean flag = true;
            // 循环读取并打印用户发送的消息
            while (flag)
            {
                String line = reader.readLine();
                if (line == null){
                    flag = false;
                    continue;
                }
                String timestamp = getCurrentTimestamp();
                String msg = timestamp + " | " +username+":"+line;
                System.out.println(msg);
                print(msg);
            }
            // 用户退出聊天室
            closeConnect();
        } catch (IOException e) {
            try {
                // 发生异常，用户退出聊天室
                closeConnect();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }
    }
```

```
}
// 检查用户名是否唯一
private boolean isUsernameUnique() {
    synchronized (usernames) {
        if (usernames.contains(username)) {
            return false;
        } else {
            usernames.add(username);
            return true;
        }
    }
}
// 获取当前时间戳
private String getCurrentTimestamp() {
    SimpleDateFormat dateFormat = new SimpleDateFormat(pattern:"yyyy-MM-dd HH:mm:ss");
    return dateFormat.format(new Date());
}
// 设置用户名
private void setUsername(String user) {
    this.username = user;
}
// 向所有已连接的客户端发送消息
private void print(String msg) throws IOException {
    PrintWriter out = null;
    synchronized (sockets){
        for (Socket sc : sockets){
            out = new PrintWriter(sc.getOutputStream());
            out.println(msg);
```

```
    // 向所有已连接的客户端发送消息
    private void print(String msg) throws IOException {
        PrintWriter out = null;
        synchronized (sockets){
            for (Socket sc : sockets){
                out = new PrintWriter(sc.getOutputStream());
                out.println(msg);
                out.flush();
            }
        }
    }
    // 关闭与客户端的连接
    public void closeConnect() throws IOException {
        System.out.println(username+" left the chat room\n");
        print(username+" left the chat room\n");
        synchronized (sockets){
            sockets.remove(socket);
        }
        socket.close();
        displayConnectedClients();
    }
}
```

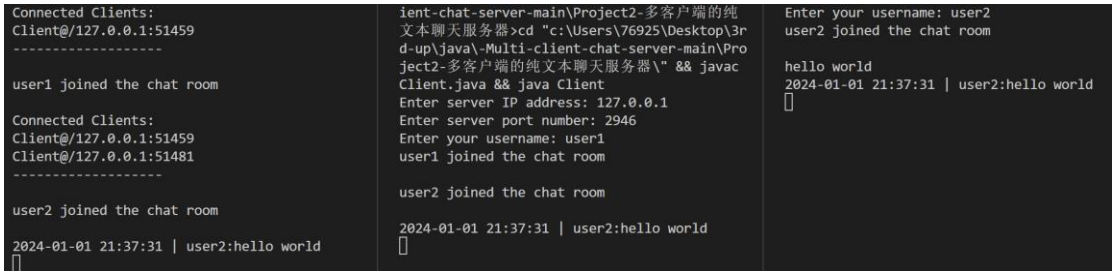
三、 程序运行截图



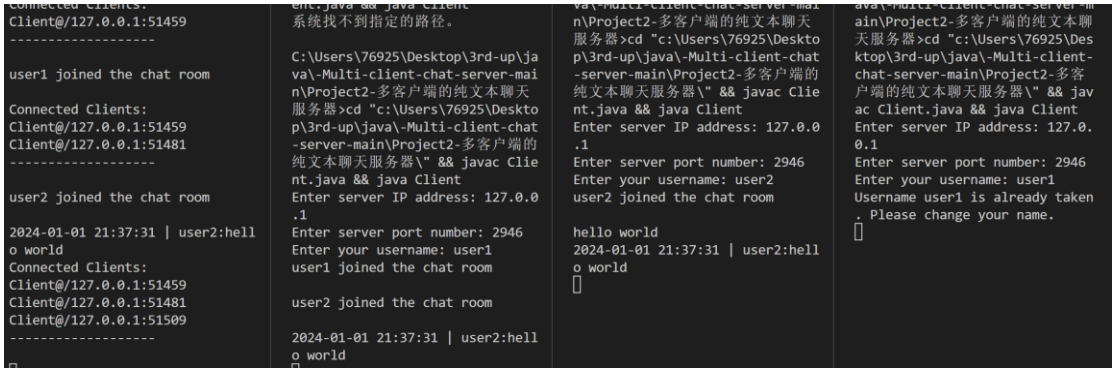
服务器和客户端的 IP 地址为 127.0.0.1，服务器端口号设置为 2946

User1 登录，server 和 client 运行界面

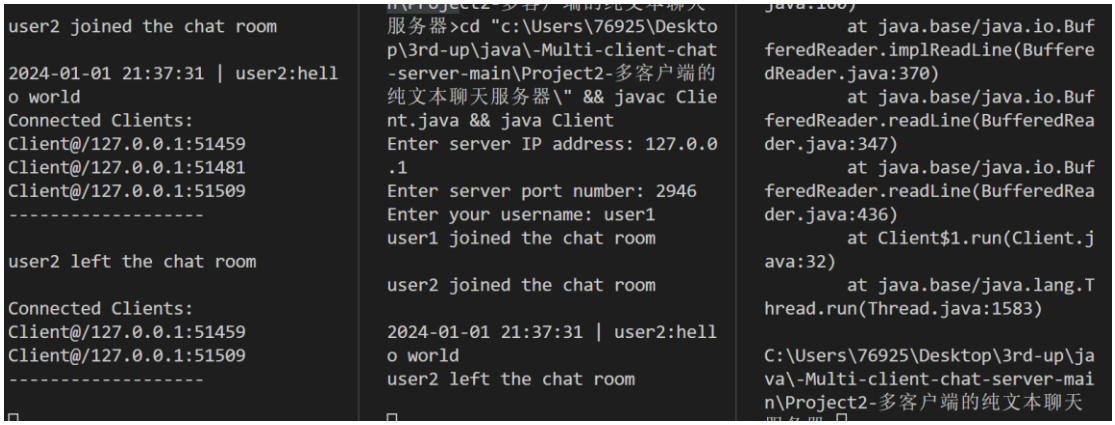
User2 登录，server 和 client1、client2 运行界面



User2 发送消息“hello world”



当有一个新的客户端想使用其他正在使用的用户名，则会显示要求更名



User2 发出 quit 结束连接，程序抛出异常并退出程序

四、 实验心得

在完成这个多用户聊天室项目的过程中，我深入学习了 Java 中的 Socket 编程，掌握了使用 ServerSocket 监听连接请求和通过 Socket 建立客户端与服务器之间双向通信的基本原理。通过为每个客户端连接创建独立线程，我提高了系统的并发处理能力。使用 BufferedReader 和 PrintWriter 实现输入输出流，使得客户端和服务器之间能够进行实时通信。在异常处理方面，针对 Socket 编程可能遇到的 IOException 等异常情况，我加强了适当的异常处理，以保障程序的稳定性。通过将时间戳添加到消息中，我不仅提供了消息发送的时间信息，还通过 SimpleDateFormat 类简化了时间戳的格式化。确保用户名的唯一性成为一个重要的考虑因素，通过使用 Set 数据结构记录已存在的用户名，有效避免了重名问题，提高了系统的健壮性。在用户体验方面，通过显示连接和断开信息、及时将消息显示到聊天窗口，我努力提升了用户在系统中的交互体验。通过合理组织代码，使用多个类来分别负责不同功能，提高了代码的可读性和维护性。这个实验让我更好地理解了网络编程的概念，提升了 Java 编程和异常处理的技能，对于处理多用户连接的实际应用场景有了更深入的认识。