华东师范大学数据学院上机实践报告

Computer Network & Coding Lab 3

课程名称： 计算机网络	年级：2023	上机实践成绩：
指导老师：张召	**姓名**：陈子谦	
上机实践名称：Socket编程	**学号**：10235501454	**上机实践日期**：

一、题目要求

基于 **Socket 编程** 实现一个简易的聊天程序 **EasyChat**，要求实现如下基本功能：

- 服务端：
  - 实时维护在线用户列表
  - 支持客户端私聊、群聊
  - 心跳机制检查用户是否在线
- 客户端：
  - 验证登入系统、无异常登出
  - 拉取在线用户列表
  - 私聊，发送消息给指定用户
  - 群聊，群发消息给在线用户

测试步骤

1. 服务端启动后，等待客户端连接
2. 客户端输入账号密码后（服务器端可用哈希表模拟），验证登录后连接到服务器端
3. 客户端按照预先定义的命令格式，输入命令
4. 输入 getUserList，获取在线用户列表
5. 输入 @userId 消息内容，发送消息给指定用户
6. 输入 @all 消息内容，群发消息给除了自己的在线用户
7. 输入 exit，客户端断开连接，服务器删除该用户连接

二、项目实现思路

1、系统架构设计

本项目采用经典的C/S（客户端/服务器）架构，基于Java Socket技术实现网络通信功能。整体系统架构如下：

- **服务器端（Server）**：
  - 作为系统的核心，负责接收客户端连接请求、处理客户端发送的各类消息、管理在线用户状态
  - 采用多线程技术处理并发连接，每个客户端连接由独立的ClientHandler线程处理
  - 集中管理所有用户状态，包括用户认证、在线状态监控、消息转发等核心功能

    - 维护全局用户列表，通过ConcurrentHashMap实现线程安全的用户管理
    - 实现心跳检测机制，定期检查客户端连接状态，及时清理非活动连接
- **客户端（Client）**：
    - 提供用户友好的命令行交互界面，便于用户登录、发送消息、查询在线用户等操作
    - 管理与服务器的连接状态，包括连接建立、心跳维护、异常处理等
    - 通过独立线程实时接收服务器推送的消息，包括私聊消息、群聊消息、系统通知等
    - 定时发送心跳包确保服务器能感知客户端状态，防止异常断开
    - 支持多种命令格式，使用简单直观的命令前缀（如@）识别消息类型

2、核心设计

- **Message**：
    - 作为整个系统的通信基础，定义消息的标准格式，实现Serializable接口，支持对象序列化传输
    - 封装四个核心属性：type（消息类型）、sender（发送者）、receiver（接收者）、content（内容）同时提供完整的getter/setter方法，便于消息处理
- **ChatServer**：
    - 服务器主类，负责启动服务、监听客户端连接请求
    - 管理ServerSocket，处理端口监听和连接接收
    - 维护全局clientHandlers集合，存储所有在线客户端的处理器
    - 创建并启动ClientHandler线程处理客户端连接
- **ClientHandler**：
    - 作为服务器端处理单个客户端连接的核心类
    - 实现Runnable接口，以独立线程运行
    - 负责与特定客户端的所有交互，包括消息接收、处理和发送
    - 维护客户端的会话状态，包括用户ID、最后心跳时间等
    - 实现心跳检测逻辑，定期检查客户端活跃状态
    - 处理客户端异常断开，确保资源及时释放和状态更新
- **UserManager**：
    - 模拟数据库功能，存储预设的用户账号和密码
    - 提供authenticate方法验证用户身份
- **ChatClient**：
    - 客户端主类，负责与服务器建立连接并处理用户交互
    - 管理Socket连接和输入输出流
    - 处理用户登录认证过程
    - 启动消息接收线程和心跳发送线程
    - 解析用户输入命令，转换为Message对象发送给服务器
    - 处理服务器返回的各类消息，并展示给用户

3、通信协议与消息机制

- **消息类型（Type）设计**：
  - `LOGIN`：客户端登录请求，携带用户名和密码
  - `LOGIN_SUCCESS`：服务器返回登录成功响应
  - `ERROR`：各类错误提示信息
  - `GET_USER_LIST`：请求获取在线用户列表
  - `USER_LIST`：返回当前在线用户列表
  - `PRIVATE_MSG`：私聊消息，指定特定接收者
  - `GROUP_MSG`：群发消息，发送给所有在线用户
  - `HEARTBEAT`：心跳包，维持连接活跃性
  - `LOGOUT`：客户端主动退出
  - `SYSTEM`：系统通知消息，如用户上线下线提醒
- **消息传输机制**：
  - 利用Java的ObjectOutputStream和ObjectInputStream实现Message对象的序列化和反序列化
  - 每个消息包含完整的发送者、接收者信息，服务器根据这些信息进行消息路由
  - 消息格式统一，使得系统能够以一致的方式处理不同类型的通信需求
- **消息流向**：
  - 客户端→服务器：用户操作产生的各类请求（登录、发消息、请求用户列表等）
  - 服务器→客户端：系统响应、消息转发、状态更新等
  - 服务器→多客户端：群发消息、系统广播等

4、心跳机制与连接管理

- **客户端心跳发送**：
  - 客户端登录成功后，启动独立的心跳线程
  - 心跳线程每10秒发送一次HEARTBEAT类型的消息
  - 心跳消息包含客户端ID，服务器据此识别心跳来源
- **服务器心跳检测**：
  - 服务器为每个ClientHandler维护lastHeartbeatTime时间戳
  - 每次收到客户端消息（包括心跳）时更新时间戳
  - 后台线程每5秒扫描所有连接，检查心跳时间
  - 若超过30秒未收到心跳，判定客户端已断开，清理相关资源

5、消息路由与转发策略

- **私聊消息路由**：
  - 客户端通过"@userId 消息内容"格式指定接收者
  - 服务器根据receiver字段查找目标ClientHandler
  - 若目标在线，直接转发；否则返回错误信息

- **群聊消息广播**：
  - 客户端使用"@all 消息内容"格式发送群聊消息
  - 服务器遍历clientHandlers集合，将消息发送给除发送者外的所有在线用户
- **系统消息广播**：
  - 用于通知用户状态变化（如用户上线、下线）
  - 服务器主动生成SYSTEM类型消息，广播给所有在线用户

三、功能实现情况

1、用户认证与登入登出管理

- **客户端登录流程**：
  - 用户输入服务器地址、端口、用户ID和密码
  - 客户端创建LOGIN类型的Message对象，将用户ID和密码封装发送，等待服务器验证响应，根据响应决定后续操作
  - 实现代码：

```
public boolean login(String userId, String password) {
    this.userId = userId;
    try {
        Message loginMsg = new Message("LOGIN", userId, "server", password);
        sendMessage(loginMsg);

        Message response = (Message) in.readObject();
        if ("LOGIN_SUCCESS".equals(response.getType())) {
            System.out.println(response.getContent());
            startHeartbeat();
            startMessageListener();
            return true;
        } else {
            System.out.println("Login Failed: " + response.getContent());
            closeConnection();
            return false;
        }
    } catch (IOException | ClassNotFoundException e) {
        System.out.println("登录过程出错: " + e.getMessage());
        closeConnection();
        return false;
    }
}
```

- **服务器验证过程**：
  - ClientHandler接收LOGIN消息，提取用户ID和密码
  - 调用UserManager.authenticate进行验证
  - 验证成功后，将用户加入在线列表，返回LOGIN_SUCCESS，验证失败则返回ERROR消息

```
// 验证用户
String userId = loginMsg.getSender();
String password = loginMsg.getContent();
```

```
if (!userManager.authenticate(userId, password)) {
    sendMessage(new Message("ERROR", "server", userId, "用户名或密码错误"));
    closeConnection();
    return;
}

// 登录成功
this.userId = userId;
clientHandlers.put(userId, this);
sendMessage(new Message("LOGIN_SUCCESS", "server", userId, "登录成功"));
broadcastUserList();
```

登出处理

系统支持两种登出方式，确保资源正确释放：

- **主动登出**：
  - 用户输入"exit"命令，客户端发送LOGOUT消息，向其他用户广播用户离开消息
  - 服务器接收后关闭连接，清理资源
- **被动检测**：
  - 通过心跳机制检测到客户端异常断开
  - 自动清理相关资源，更新在线用户列表，广播系统消息通知其他用户

2、在线用户列表维护

用户状态管理

- **数据结构选择**：
  - 使用ConcurrentHashMap<String, ClientHandler>存储在线用户
  - 键为用户ID，值为对应的ClientHandler实例
- **状态更新机制**：
  - 用户登录成功后，将其ClientHandler加入Map
  - 用户登出或连接断开时，从Map中移除
  - 每次状态变化后，向所有在线用户广播最新用户列表

用户列表同步

- **主动获取**：
  - 客户端输入"getUserList"命令请求最新列表
  - 服务器收到请求后，生成包含所有在线用户ID的字符串，以USER_LIST类型消息返回给请求客户端
  - 实现代码：

```
private void sendUserList() {
    StringBuilder userList = new StringBuilder();
    for (String user : clientHandlers.keySet()) {
        userList.append(user).append(",");
    }
    if (userList.length() > 0) {
        userList.deleteCharAt(userList.length() - 1);
    }
    sendMessage(new Message("USER_LIST", "server", userId,
userList.toString()));
}
```

- **自动广播**:
  - 当有用户登入或登出时，服务器自动向所有在线用户广播最新列表，实现代码如下:

```
private void broadcastUserList() {
    for (ClientHandler handler : clientHandlers.values()) {
        handler.sendUserList();
    }
}
```

3、消息传输与处理

私聊功能

- **客户端发送**:
  - 客户端检测到输入以"@"开头，且后面不是"all" 解析出接收者ID和消息内容，然后创建PRIVATE_MSG类型的Message对象，代码如下:

```
if (input.startsWith("@")) {
    int spaceIndex = input.indexOf(" ");
    if (spaceIndex != -1) {
        String receiver = input.substring(1, spaceIndex);
        String content = input.substring(spaceIndex + 1);

        if (receiver.equals("all")) {
            // 群聊处理
        } else {
            sendMessage(new Message("PRIVATE_MSG", userId, receiver, content));
            System.out.println("[PRIVATE->"+receiver+"] 我: " + content);
        }
    }
}
```

- **消息路由**:
  - 服务器收到PRIVATE_MSG消息，根据receiver字段查找目标用户
  - 如果目标用户在线，直接转发消息，如果不在线，返回错误提示

```
case "PRIVATE_MSG":
    String receiverId = message.getReceiver();
    ClientHandler receiver = clientHandlers.get(receiverId);
    if (receiver != null) {
        receiver.sendMessage(message);
    } else {
        sendMessage(new Message("ERROR", "server", userId, "用户 " +
receiverId + " 不在线"));
    }
    break;
```

- **客户端显示**：
  - 接收方收到PRIVATE_MSG消息后，显示带有发送者信息的消息
  - 发送方在本地显示带有接收者信息的消息副本

群聊功能

- **客户端发送**：
  - 客户端检测到"@all"开头的命令，创建GROUP_MSG类型的Message对象，receiver设为"all"

```
if (receiver.equals("all")) {
    sendMessage(new Message("GROUP_MSG", userId, "all", content));
    System.out.println("[ALL] 我: " + content);
}
```

- **消息广播**：
  - 服务器收到GROUP_MSG消息，遍历所有在线用户
  - 排除发送者自己，向其他所有用户转发消息

```
private void broadcastMessage(Message message) {
    for (Map.Entry<String, ClientHandler> entry : clientHandlers.entrySet())
{
        if (!entry.getKey().equals(userId)) { // 不发给自己
            entry.getValue().sendMessage(message);
        }
    }
}
```

- **客户端显示**：
  - 接收方显示带有[GROUP]前缀和发送者信息的消息
  - 发送方本地显示确认信息

4、心跳检测与连接管理

系统通过定时发送和监测心跳包确保连接状态的及时更新：

- 客户端登录成功后启动心跳线程，每10秒发送一次HEARTBEAT类型消息

```
private void startHeartbeat() {
    new Thread(() -> {
        while (isRunning) {
            try {
                Thread.sleep(10000); // 每10秒发送一次心跳
                sendMessage(new Message("HEARTBEAT", userId, "server", ""));
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }).start();
}
```

- 服务器接收任何类型消息时更新lastHeartbeatTime，HEARTBEAT只更新时间戳，不需要其他处理

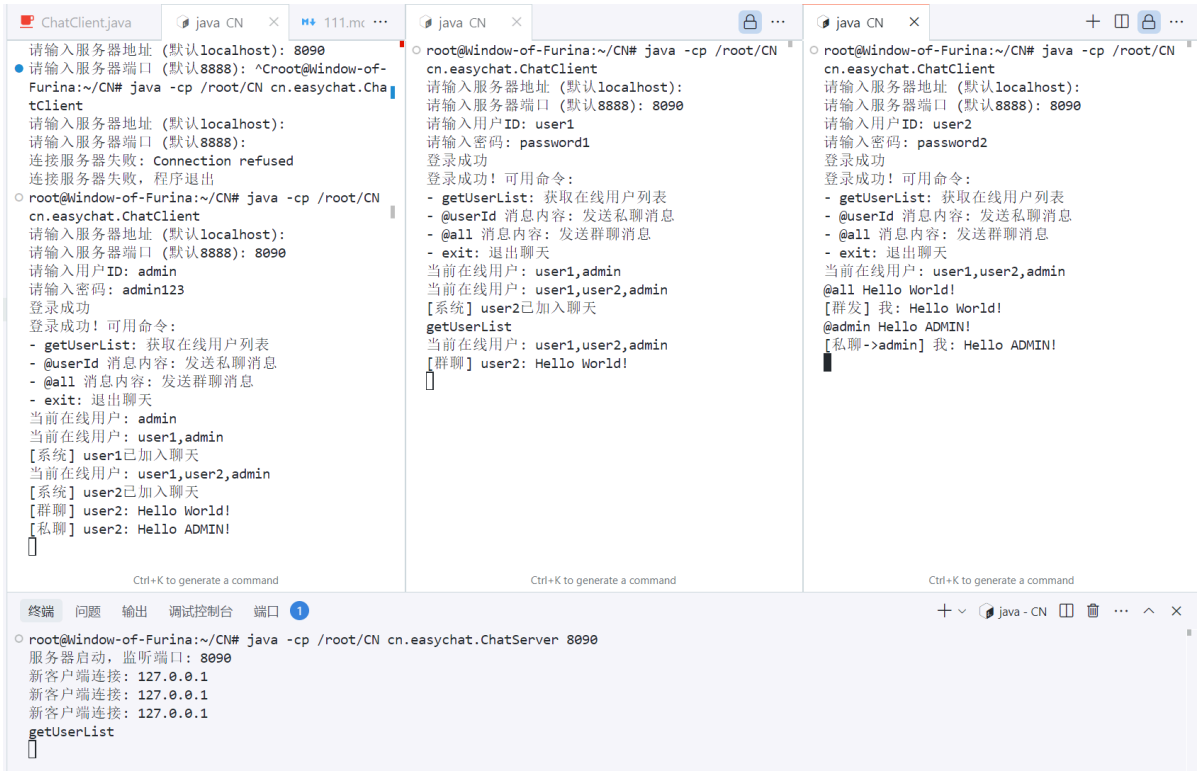
```
private void processMessage(Message message) {
    lastHeartbeatTime = System.currentTimeMillis(); // 更新最后心跳时间

    switch (message.getType()) {
        case "HEARTBEAT":
            // 仅更新心跳时间，不需要其他处理
            break;
        // 其他消息处理...
    }
}
```

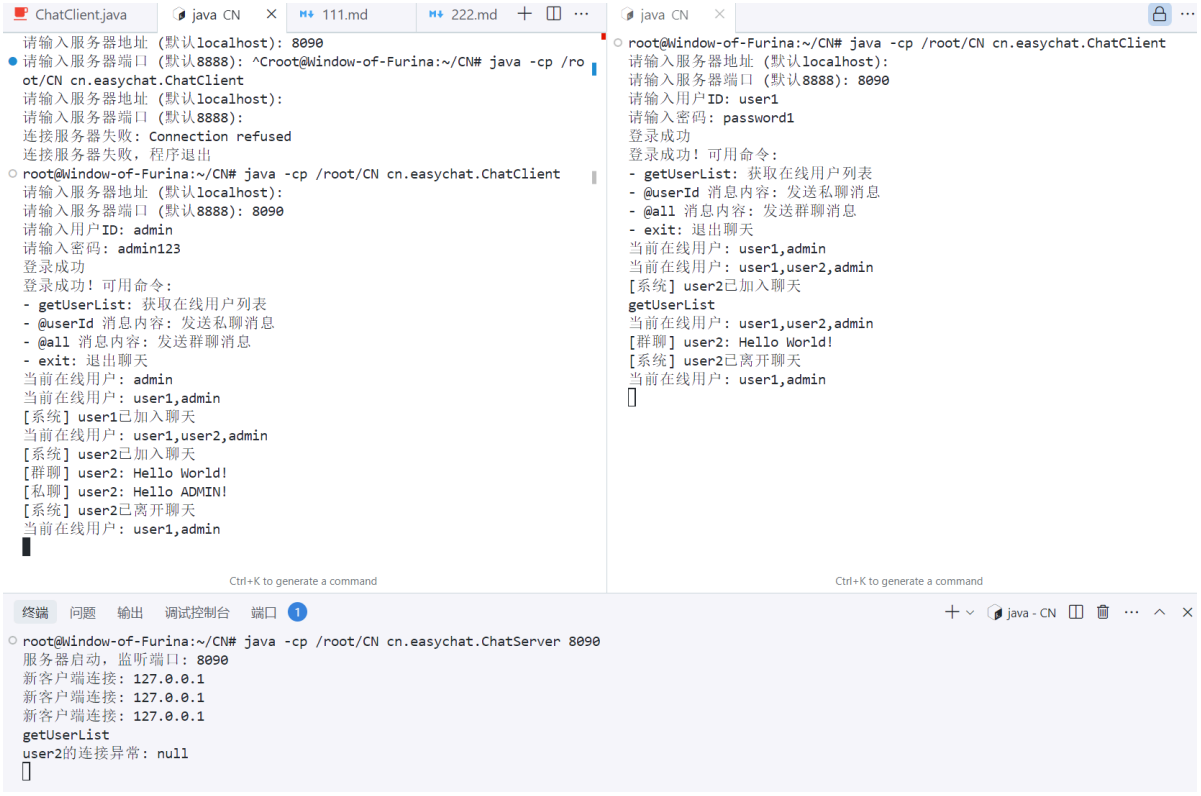
- **超时检测**：
  - 后台线程定期检查所有连接的心跳状态
  - 如30秒未收到心跳，判定连接断开，主动清理资源，更新用户状态，广播系统消息通知其他用户

四、总结

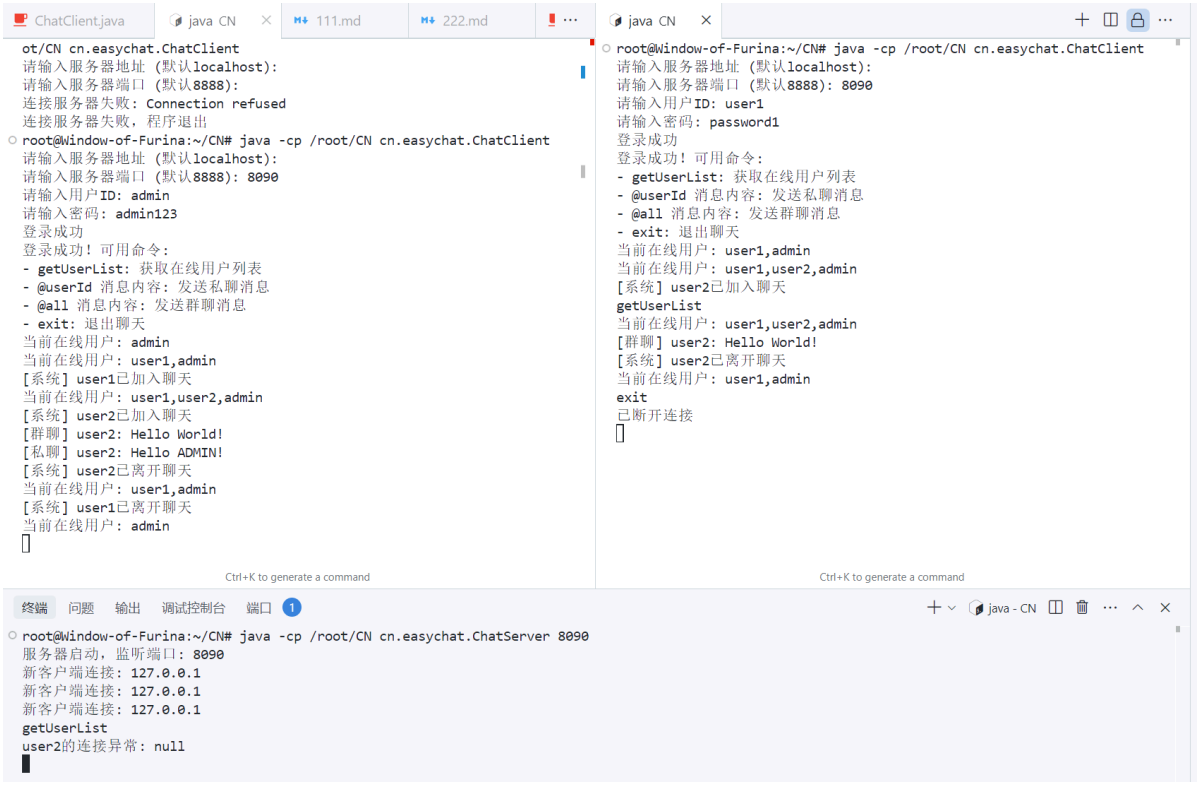
用户登陆、私聊群聊消息发送、获取用户List运行截图



user意外退出时的运行消息截图



用户输入exit自行退出的截图



通过本次实验，我深入理解了基于Socket的网络通信机制，以及如何构建一个功能完整的实时通信系统。系统通过多线程技术处理并发连接，解决了实时通信系统中的高并发挑战实现了在线用户状态的实时同步，通过心跳机制和消息广播保持了系统状态的一致性。EasyChat系统虽然简单，但实现了完整的聊天功能，支持私聊和群聊两种模式，满足不同的通信需求。通过实现一个完整的聊天系统，我不仅巩固了Java编程的基础知识，还掌握了Socket编程、多线程并发、资源管理等高级技术。在解决各种技术问题的过程中，我提升了分析问题和解决问题的能力，同时也加深了对计算机网络原理的理解。