|  |  |
| --- | --- |
| **计算机网络**  **课程设计报告** | |
| 基于Java Socket的音视频聊天室  的设计与实现 | |
|  | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
| **成员1** | 软件2104 柳景荠 2204215059 |
| **成员2** | 软件2104 程奕浩 2213311084 |

# 摘要及说明

1. 最终成果：共开发3个大版本的IP电话，**均支持文字+音视频通信**。
   1. Version1为同一局域网下无服务器的IP电话。通信双方需至少有一方知晓对方的IP、port。
   2. Version2为**同一局域网**下**有服务器**的IP电话。加入了服务器，使用数据库保存用户信息及用户间好友关系。用户**无需知道**对方的IP和port。
   3. Version3为**广域网**下的IP电话。我们将server程序**部署到云服务器**上，具有公网IP的服务器作为中转，用户可以在**不同网络**下联网进行通信。（此版本目前仅实现文本通信，音视频通信尚有bug未能解决（从公网向NAT背后的client发送UDP报文段似乎无法送到））。
2. 本报告中，设计思路汇报了整个项目的经过及概要的设计骨架。重要细节说明详细解释了部分较为关键的细节实现。Bug及解决方案记录了我们在完成本项目时印象最深刻的几个bug。

**目 录**

[摘要及说明 II](#_Toc166736174)

[引言 1](#_Toc166736175)

[1 设计思路 1](#_Toc166736176)

[1.1 尝试视频通话阶段 1](#_Toc166736177)

[1.1.1 TCP实现视频通话的尝试 1](#_Toc166736178)

[1.1.2 UDP实现视频通话的尝试 1](#_Toc166736179)

[1.2 不局限于同一局域网下的IP电话 2](#_Toc166736180)

[1.2.1 P2P通信的初步设想 2](#_Toc166736181)

[1.2.2 经服务器转发的IP电话设计与实现 2](#_Toc166736182)

[1.3 应用层通信协议设计 3](#_Toc166736183)

[1.3.1 UDP 3](#_Toc166736184)

[1.3.2 TCP 3](#_Toc166736185)

[2 架构设计 6](#_Toc166736186)

[2.1 C/S架构分析 6](#_Toc166736187)

[2.2 C/S架构实现 6](#_Toc166736188)

[3 重要细节实现说明 8](#_Toc166736189)

[3.1 代码结构 8](#_Toc166736190)

[3.1.1 说明 8](#_Toc166736191)

[3.1.2 代码结构展示 8](#_Toc166736192)

[3.2 Client重点细节 8](#_Toc166736193)

[3.2.1 Client主体类的生命历程 8](#_Toc166736194)

[3.2.2 UDP部分 9](#_Toc166736195)

[3.3 Server重点细节 10](#_Toc166736196)

[3.3.1 Server主体类 10](#_Toc166736197)

[3.3.2 SingleTcpThread 11](#_Toc166736198)

[3.3.3 CommunicationThread 11](#_Toc166736199)

[4 令人印象深刻的Bug及解决方案 12](#_Toc166736200)

[4.1 初始时的多余空信息 12](#_Toc166736201)

[4.2 Socket的ObjectInputStream与ObjectOutputStream的顺序问题 12](#_Toc166736202)

[4.3 ObjectInputStream.readObject() 阻塞式读取带来的易错bug 12](#_Toc166736203)

[5 效果展示 15](#_Toc166736204)

[5.1 基本功能展示 15](#_Toc166736205)

[5.2 音视频展示 16](#_Toc166736206)

[6 总结与感悟 17](#_Toc166736207)

[7 参考资料 18](#_Toc166736208)

# 引言

网络通信是现代社会中不可或缺的一部分，从即时消息传递到音视频通话，网络通信技术的进步极大地丰富了人们的交流方式。然而，现有的许多解决方案在灵活性、实时性以及跨网络通信方面存在不足。本文旨在设计并实现一个基于Socket编程的多功能网络聊天室，特别是支持音视频通话的功能，并探讨其在不同网络环境中的应用。

# 设计思路

在最初的尝试阶段，我们使用TCP实现了在局域网下的文本对话功能。具体而言，当对话双方之一知晓对方的IP和port时，可以通过TCP进行文本通信。这一阶段验证了课堂所学知识，实现了基本的网络通信功能。

## 尝试视频通话阶段

### TCP实现视频通话的尝试

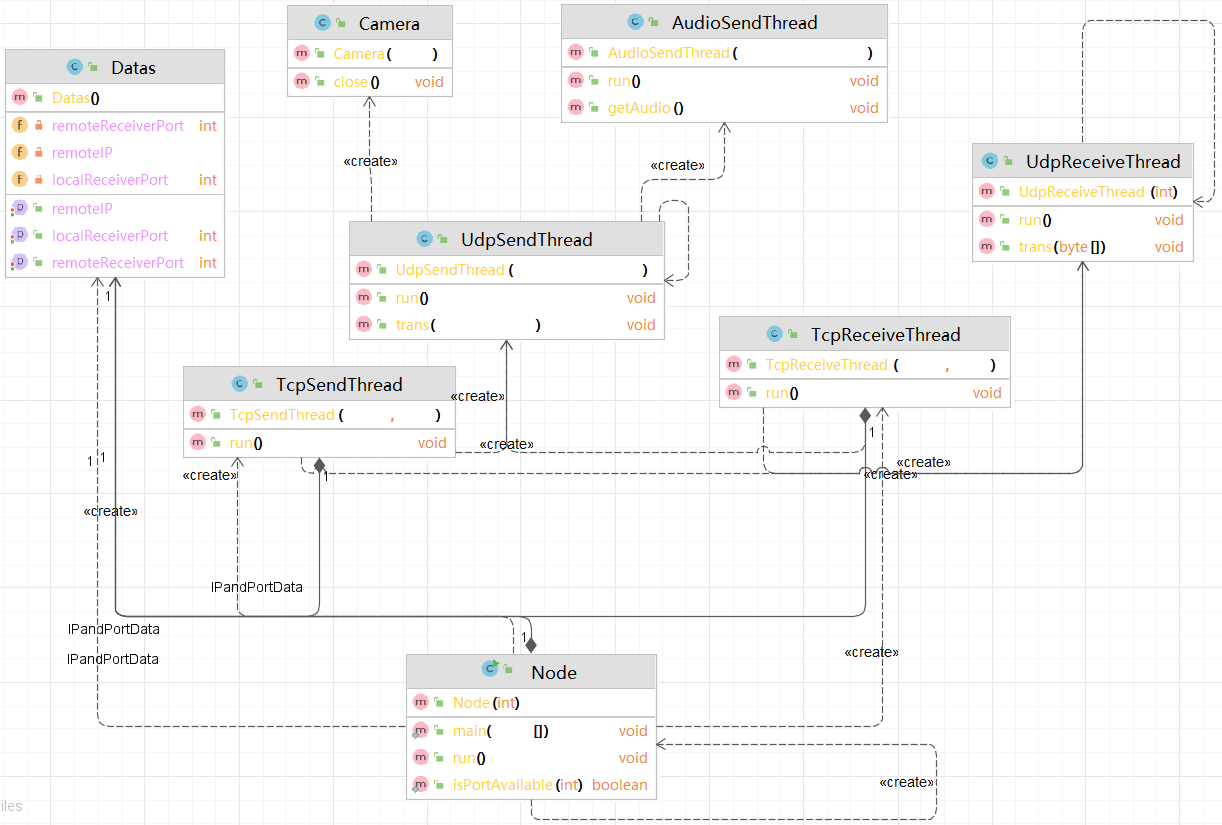
我们首先尝试用TCP实现视频通话，具体方法是通过webcam（com.github.sarxos.webcam.\*）不断拍照，并将每一张照片作为一个TCP分组发送。但实践证明TCP并不适合视频通信，延迟和卡顿太大。

### UDP实现视频通话的尝试

我们改用UDP协议进行尝试。具体步骤如下：

1. 双方都开启serversocket等待accept，其中一方知晓对方的IP和port，进行search操作。连接成功后，发起方的serversocket关闭，双方使用已建立的TCP连接进行文本通信。
2. 文本通信中，若任意一方发送“/video”指令，则双方各自打开与当前TCP端口号相同的UDP接收端，开启视频通信。
3. 双方打开webcam不断拍照，每张照片作为一个UDP分组发送到对方的UDP接收端。
4. 在视频通信过程中，文本通信仍可继续，双方可发送预设命令终止视频或文本通话。

为了实现音视频同步通信，我们在webcam拍照的同时，再开一个线程接收麦克风信号，将每段麦克风数据包装成UDP分组发送，并在每个分组开头加一个标识，用数字表示当前分组是audio还是video。UDP接收端在接收到分组时，先检查标识，再分别处理。

至此，支持text、video+audio的局域网IP电话基本完成。然而，此时系统的局限性在于，两个用户之间需要满足在同一局域网下，并且至少一方需要知道对方的IP地址和port。

## 不局限于同一局域网下的IP电话

### P2P通信的初步设想

为了突破局域网的限制，我们首先考虑了P2P通信的可能性。设想通过一个服务器获取对方的IP地址和端口号，随后双方直接通信。然而，经过学习我们了解到，要实现不经过服务器的通信，需要复杂的NAT穿透技术，这对于我们的实现来说较为复杂。

### 经服务器转发的IP电话设计与实现

我们决定采用服务器中继的方式来实现跨网络的IP电话通信。

UDP通信设计

1. 每对用户开启text通信之后，任一用户发送/video指令（一个特殊的TCP报文段），服务器开启一个专用于当前两个用户的UDP socket。
2. 服务器向两个用户发送一个TCP响应，告知服务器该UDP socket的port，之后两个用户均向该DatagramSocket发送UDP报文段，服务器进行转发。

TCP通信设计

1. 所有TCP通信采用ObjectOutputStream与ObjectInputStream。每一种功能的TCP报文对应一个我们定义的类。
2. 服务器维护一个全线程可见的HashMap<Integer,SocketWorker>。
   1. 其中SocketWorker为我们定义的一个工具类，功能为存储一个tcp socket及它的ObjectOutputStream与ObjectInputStream。SocketWorker是出于避开bug及简化程序而建立。
3. 每个用户登录时，服务器生成一个tcpsocket（socketworker），将其与用户ID一起放入hashmap，并生成一个专用线程来服务该用户。
4. 专用线程最外层的while对hashmap.containkey(userid)进行检测。当hashmap中还存有该用户的tcpsocket，说明该用户仍未与朋友进行联络。
5. 当该用户主动或被动地进行联络，主动方将双方的socket从hashmap中抽出。这一举动会导致被联络方的专用线程结束。
6. 至此，联络双方在服务器中共用一个线程。
7. 当用户请求视频通话时，服务器告知用户服务器的UDP接收端口，客户端开始UDP通信。

## 应用层通信协议设计

### UDP

包格式：

1个int: 源用户ID

1个int: 真实数据长度

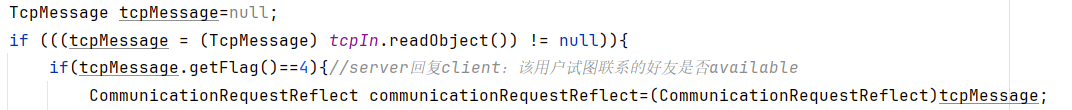
1个int: 数据类型（1代表video，2代表audio）

### TCP

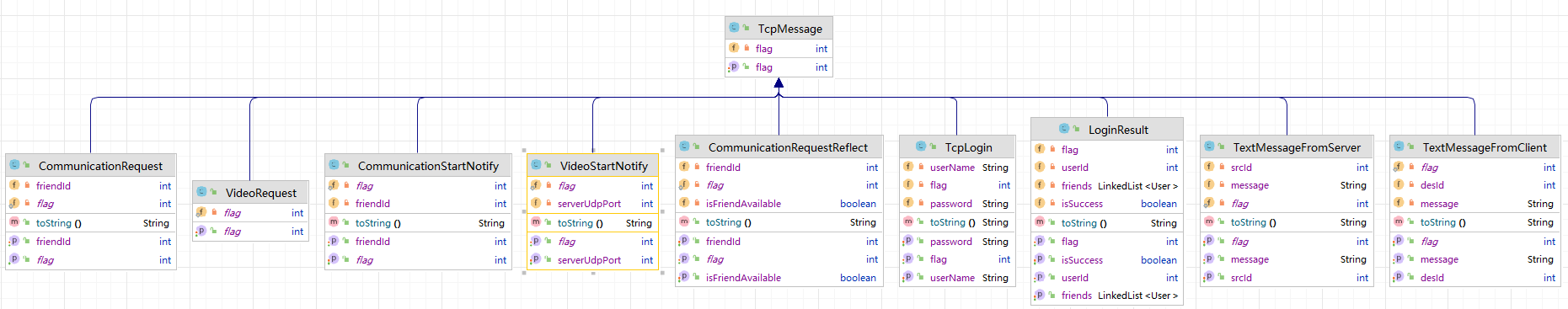
消息类型及其对应的操作：

TcpMessage：int flag

TcpMessage是以下所有报文类的父类。

接收方接收时，首先读取为(TcpMessage) tcpIn.readObject()，然后根据flag值将其转换为具体的报文类。

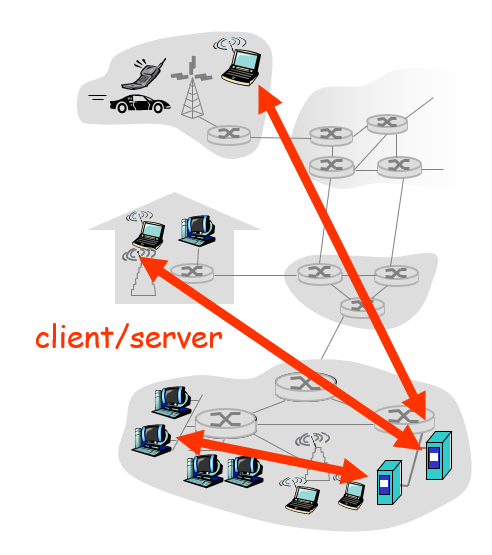
|  |  |  |
| --- | --- | --- |
| 消息码Flag | 消息类型及方向 | 详细描述 |
| 0（弃用） | client→server | 唤醒socket |
| 1 | TcpLogin：client→server  进行登录 | String username，String password |
| 2 | LoginResult ：server→client  回复登录 | boolean isSuccess  （若登录成功）int userId，LinkedList<User> friends |
| 3 | CommunicationRequest：  client→server  选择好友进行联系 | int 好友ID |
| 4 | CommunicationRequestReflect：server→client  反馈选择好友的结果 | boolean isFriendAvailable，int 该好友ID |
| 5 | CommunicationStartNotify：  server→client  告知被发起者开启通话 | int 发起者ID |
| 6 | TextMessageFromClient：  client→server  文本通信 | int 目的用户的ID，String message |
| 7 | TextMessageFromServer：  server→client  文本通信 | int 源用户的ID，String message |
| 8 | VideoRequest：  client→server  发起video | - |
| 9 | VideoStartNotify：  server→client  发起video | int：server的port |
| 10（弃用） | server→client  已准备好，可以开始video |  |
| 11（暂未使用） | client→server  终止video | int：srcID，int：desID |
| 12（暂未使用） | server→client  终止video |  |
| 13（暂未使用） | client→server  终止通信 |  |
| 14（暂未使用） | server→client  终止通信 |  |



报文类的类图

# 架构设计

C/S（Client/Server）架构是一种经典的网络架构模式，通常用于分布式计算系统。在这种架构中，客户端（Client）负责发起请求，服务器（Server）负责处理请求并返回结果。客户端和服务器之间通过网络进行通信，这使得资源和数据处理能力可以集中在服务器端，而客户端则主要负责用户交互和请求的发送。



## C/S架构分析

在我们的聊天软件项目中，我们采用了CS架构。服务器集中处理和管理所有用户的消息和连接，使得系统的控制和维护更加方便。同时，服务器可以进行统一的数据存储、用户验证和权限管理，确保系统的安全性和稳定性。服务器可以使用强大的硬件资源和优化的算法来处理大量并发请求，而客户端只需处理用户界面和简单的逻辑操作，这样可以更好地利用资源，提高系统性能。通过CS架构，客户端和服务器可以独立开发和升级，便于系统的扩展和功能的增加。新的客户端功能可以快速部署，而无需大幅修改服务器端代码。

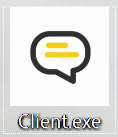
## C/S架构实现

客户端设计：客户端应用程序负责捕捉用户输入（如文本消息、音视频数据等），并将这些数据打包成特定的协议格式，通过网络发送给服务器。此外，客户端还负责接收服务器的响应数据，并将其显示给用户。为了支持音视频通话，客户端需要集成音视频采集和编码功能。

服务器设计：服务器应用程序接收来自各个客户端的请求，解析请求数据，根据请求类型进行相应的处理（如消息转发、用户验证等），并将处理结果返回给相应的客户端。服务器需要维护所有在线用户的连接信息，以便进行消息的高效转发和广播。

网络通信：客户端和服务器之间通过TCP和UDP协议进行通信。TCP协议用于可靠的文本消息传输，确保消息的顺序和完整性；UDP协议用于音视频数据传输，以减少延迟和提高实时性。服务器需要监听特定的端口，处理来自客户端的TCP连接请求和UDP数据包。

通过以上设计和实现，我们的聊天软件能够在局域网和跨网络环境中提供高效、可靠的文本和音视频通信服务。CS架构不仅简化了系统的开发和维护，还确保了良好的用户体验和系统性能。



# 重要细节实现说明

## 代码结构

### 说明

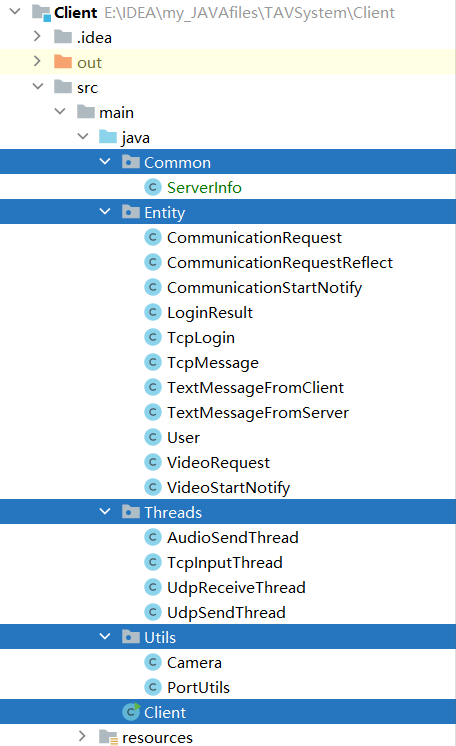
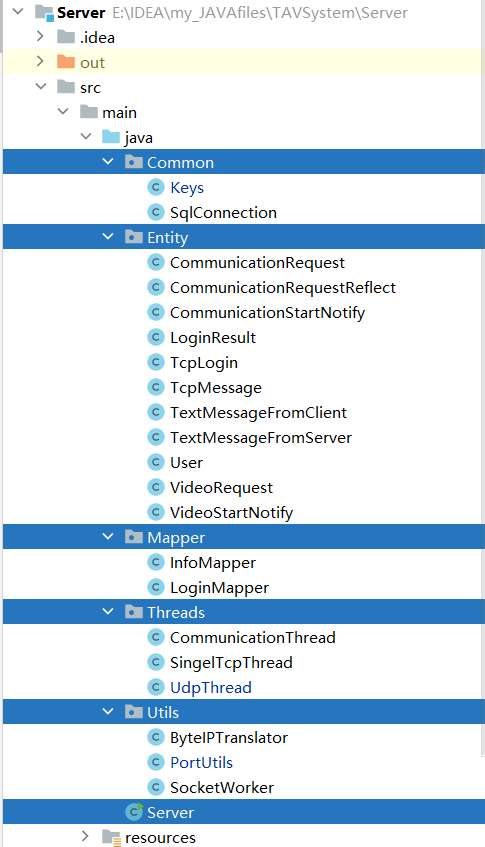
Client与Server均包含：

* + 1. 主体类-main()方法在此处。
    2. Utils-工具类。
    3. Threads-线程类。实现Runnable接口的各种线程类。
    4. Entity-实体类。包含所有报文类以及User类。
    5. Common-常数据类。包含全局不变的数据，汇总便于修改。

Server还包含：

1. Mapper-映射类。用于jdbc连接数据库。

### 代码结构展示

 Client： Server:

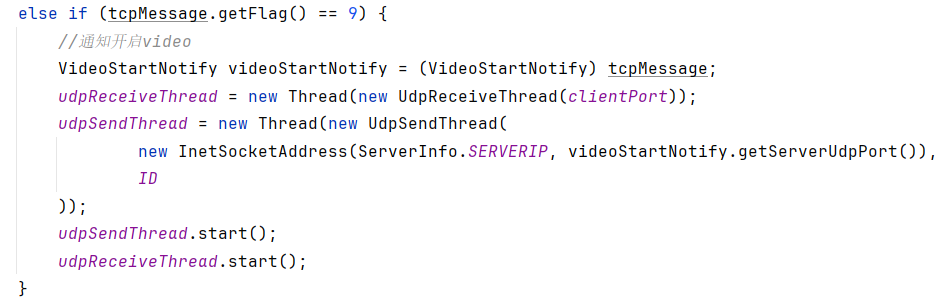
## Client重点细节

### Client主体类的生命历程

1. 程序开始运行->Client类对必要的变量进行初始化，包括scanner、socket等。Client类用简单的逻辑完成用户的登录输入、连接服务器、发送登录报文。

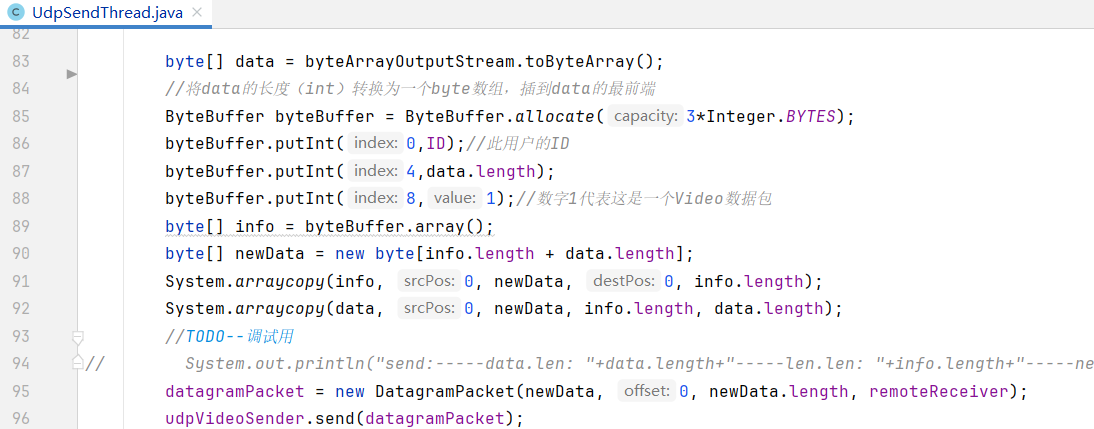
2. 登录成功后：Client类开启两个线程：listenTcpMessage专职于监听Server传来的报文，并作出反应；TcpInputThread线程类，Client类将scanner交给它。这个线程类专职于扫描用户输入，并根据不同输入采取行动。至此，Client主体类的生命历程结束。

### UDP部分

1.当listenTcpMessage线程监听到Server的开启音视频通知时，它打开udp收发线程

2.udpSendThread中，对UDP报文格式的处理：

先放入本用户ID，其次是这个报文中真正的数据的长度，最后是区分video于audio数据的Flag，其后跟着真正的数据。这个顺序是由报文的被处理顺序决定的。

Server将首先接收到报文，Server首先需要弄清楚这个报文是谁发的，它要传给谁，于是先取出ID。之后，Server需要知道真是数据长度。至于是video还是audio，Server并不关心。所以，Server可以顺序地取出前两个int，并去掉第一个int ID，将长度、Flag、数据重新打包发给另一个用户。

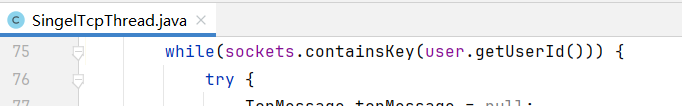
## Server重点细节

### Server主体类

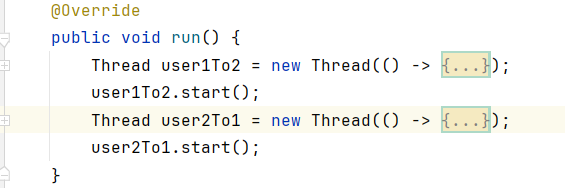
Server主体类简单而重要，它是一个while(true)循环，循环内进行serversocket.accept()，等待client的连接。当一个连接被建立，它开启一个专用线程SingleTcpThread，并将socket以及HashMap（前文提过）传给该线程。

### SingleTcpThread

这个线程负责一个用户在他于朋友联络上之前的一切事务：登录、尝试联络朋友、等待被朋友联络。

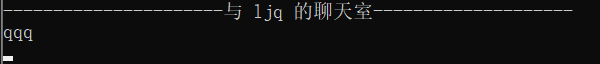
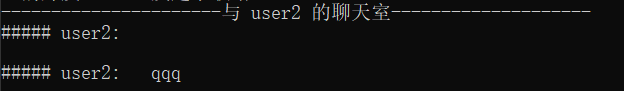
其中“等待被朋友联络”的实际方式为监测该用户的socket是否还在HashMap中（前文提过），若发现不在了，说明这个用户被朋友联络，此线程应该终止。若此用户主动联系上了朋友，则应该将自己的与朋友的socket从HashMap取出，并将两个socket一同交给一个新线程：CommunicationThread。

### CommunicationThread

 这个线程中，Server开启两个线程，分别收听并转发一个用户的信息。

# 令人印象深刻的Bug及解决方案

## 初始时的多余空信息

本项目的每一个版本中，都出现了这个bug：当联络发起方或接收方开始向对方发送第一个消息时，会首先发送出去一个内容为“”的空消息。然后再正常传输。

解决方案：尚未解决

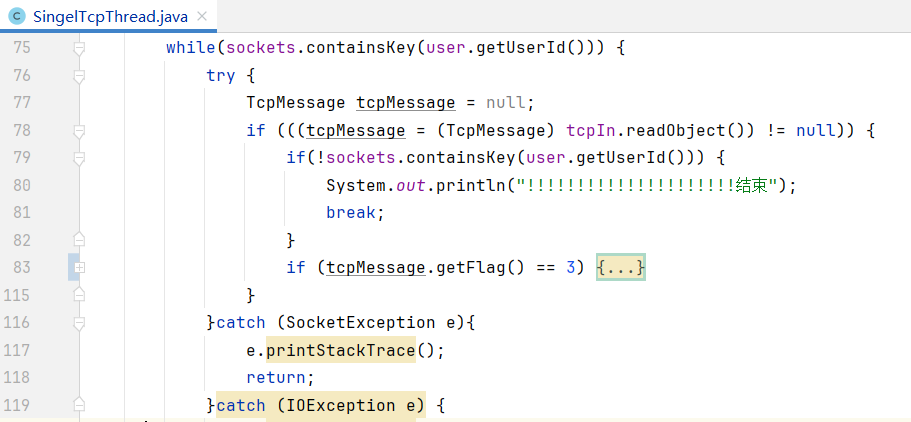
## Socket的ObjectInputStream与ObjectOutputStream的顺序问题

在构造ObjectInputStream与ObjectOutputStream时，如果先构造ObjectInputStream，后构造ObjectOutputStream，会发生死锁：程序既不报错，也不正确地向前运行。我们使用的IDE：IDEA的debug功能也再此处失效。最后只能通过输出做标记的方式发现这个问题。

解决方案：查找资料解决 https://blog.csdn.net/gary0917/article/details/83655937

## ObjectInputStream.readObject() 阻塞式读取带来的易错bug

前文提到：SingleTcpThread通过检测sockets.containsKey(user.getUserId())来判断是否继续循环。



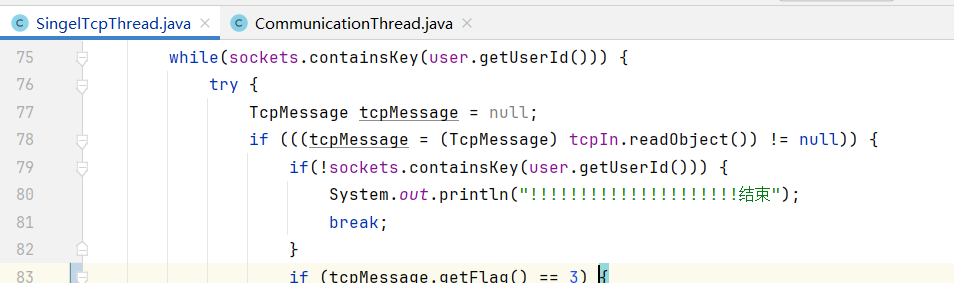
然而，当sockets.containsKey(user.getUserId())判断为真，即认为该用户仍未进行联络后，紧随其后的tcpIn.readObject()对程序进行了阻塞。若另外的用户在tcpIn.readObject()读取到数据之前，发起了对此用户的联络，则会发生：

tcpIn.readObject错误地继续执行（实际上它不该执行了），其后的代码块也一并错误地执行。

这会导致严重的错误：此处的tcpIn.readObject()与communicationThread中的tcpIn.readObject()争夺用户发来的数据。它们将序列化的对象弄得混乱残缺。于是真正需要这个数据的代码块无法识别，报错invalid code 00等。直接导致被联络方无法发送信息。

此bug隐蔽而严重，消磨了我们大量的时间经历。

解决方法：communicationThread中的tcpIn.readObject()在第一次运行之前会Thread.sleep()一小段时间。被联络方在被联络后立即发出一个无意义的“垫子”tcp包，让这里的tcpIn.readObject()读取，阻塞结束。而此处阻塞式读取后面再加一次sockets.containsKey(user.getUserId())检测，确保其后的代码不会错误地执行。



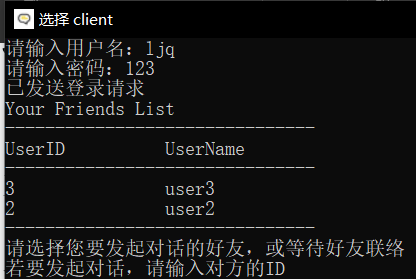


# 效果展示

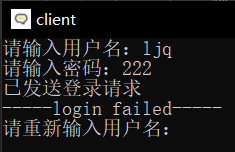
## 基本功能展示

在我们的聊天软件中，客户端之间通过服务器进行消息传递，实现了实时的文本通信。这一功能的实现展示了以下几点：

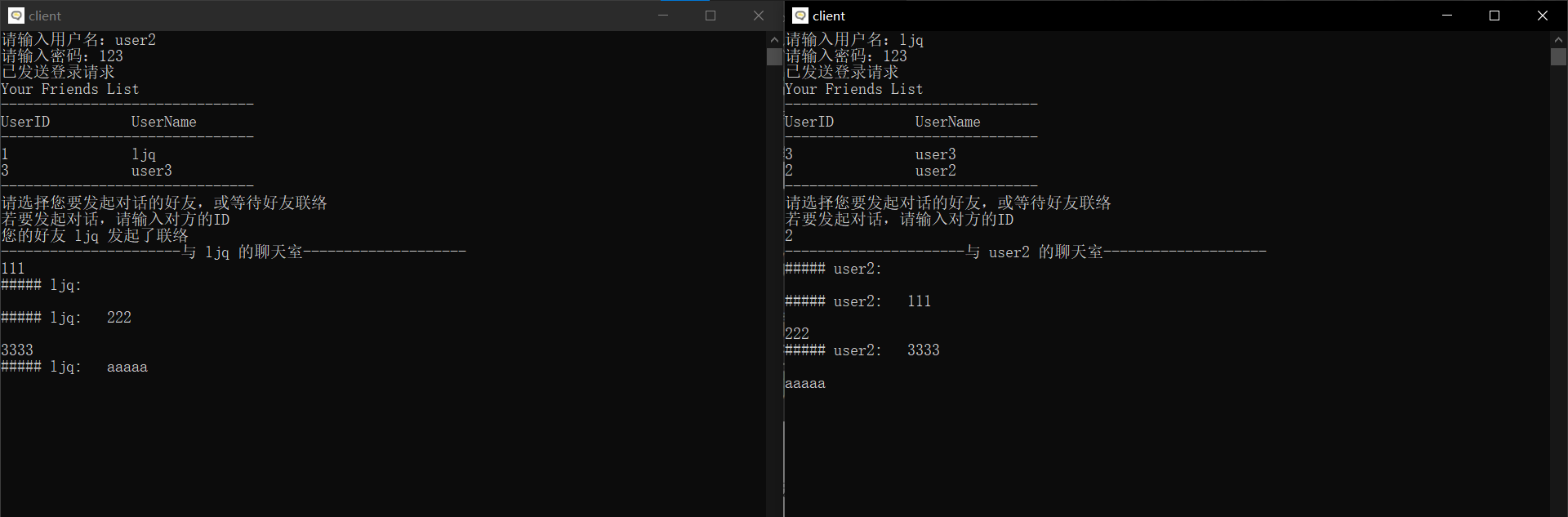
（1）用户登录与验证：用户首先通过客户端进行登录操作，输入用户名和密码。服务器接收登录请求后，验证用户的身份，并返回登录结果。

登录成功后，用户可以查看好友列表并选择好友进行通信

登录失败页面如下。



（2）消息发送与接收：用户选择好友后，可以通过客户端输入文本消息并发送。客户端将消息通过TCP协议发送至服务器，服务器接收消息后，解析并转发给目标用户。目标用户的客户端接收到消息后，在聊天窗口显示。如下图所示。



## 音视频展示

在我们的聊天软件中，除了基本的文本消息传递功能，还实现了客户端之间通过服务器进行视频通话的功能。

（1）实时视频通信：用户在进行文本聊天时，可以发送视频通话请求。服务器接收到请求后，通知目标用户并建立UDP连接进行视频数据传输。双方用户的客户端通过摄像头采集视频数据，并通过UDP协议发送至服务器，再由服务器转发给对方用户，实现实时视频通信。

（2）音频传输：在视频通话的同时，系统还支持音频数据的传输。客户端通过麦克风采集音频数据，并将音频数据打包成UDP数据包，通过服务器进行转发。接收方客户端解码音频数据，并通过扬声器播放，实现同步的音视频通话。

（3）多线程处理：客户端在进行视频通话时，采用多线程技术实现音视频数据的采集、编码、发送和接收处理。

效果如下所示。



# 总结与感悟

XXX

# 参考资料

[1] [游乐场预约管理系统分析与设计\_预约景点系统c++\_雨落俊泉的博客-CSDN博客](https://yijunquan.blog.csdn.net/article/details/122026190)

[2] [基于RBAC权限管理实现的中小学成绩管理系统\_用rbac做校园管理系统\_Hydrion-Qlz的博客-CSDN博客](https://blog.csdn.net/qq_46311811/article/details/122444924)

[3] [关于Java中Socket通信时使用ObjectInputStream与ObjectOutputStream的顺序问题-CSDN博客](https://blog.csdn.net/gary0917/article/details/83655937)