

# 计算机网络 第一次实验

---

- 姓名：秦泽斌
- 学号：2212005
- 专业：物联网工程

## 实验内容

---

1. 给出你聊天协议的完整说明。
2. 利用C或C++语言，使用基本的Socket函数完成程序。不允许使用CSocket等封装后的类编写程序。
3. 使用流式Socket完成程序。
4. 程序应有基本的对话界面，但可以不是图形界面。程序应有正常的退出方式。
5. 完成的程序应能支持多人聊天，支持英文和中文聊天。
6. 编写的程序应该结构清晰，具有较好的可读性。
7. 现场演示。
8. 提交程序源码、可执行代码和实验报告。

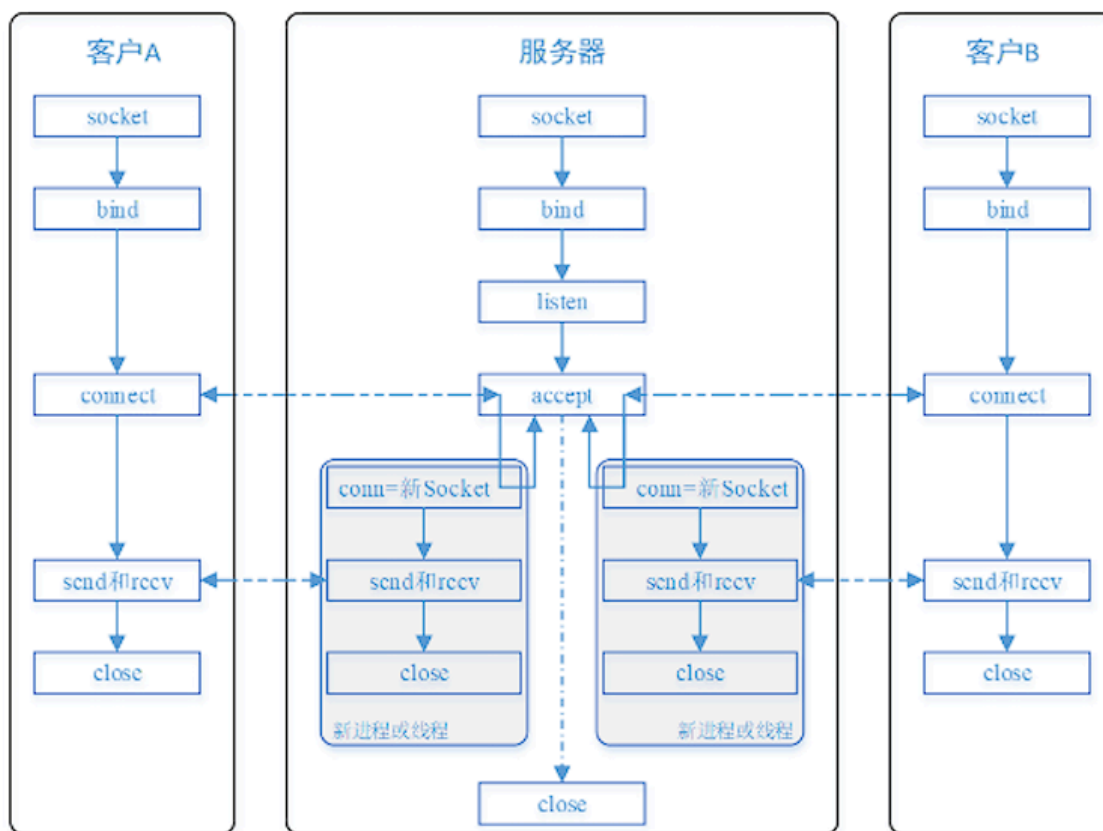
## 一、协议设计

---

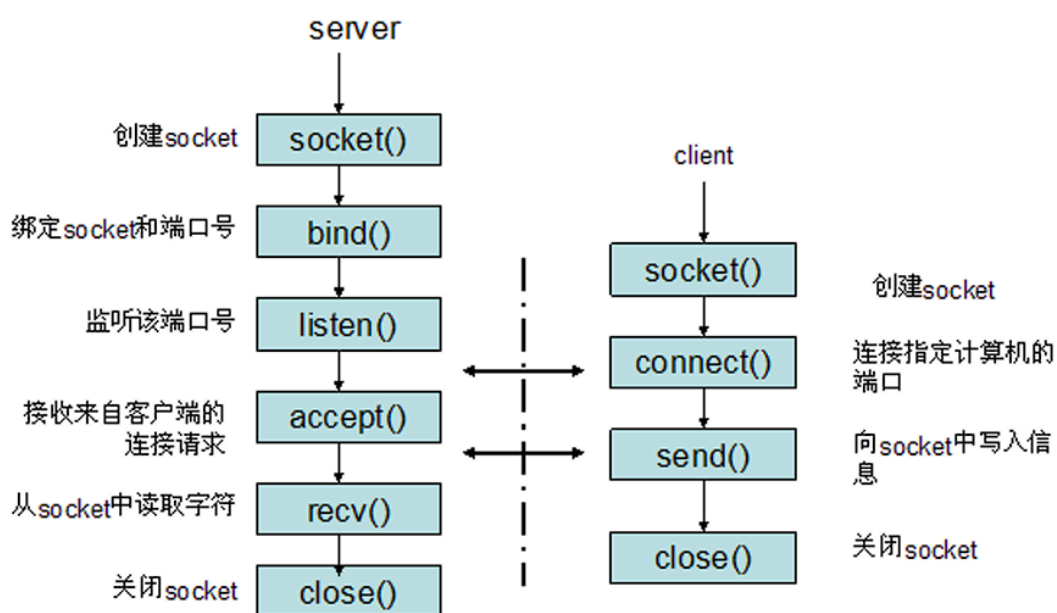
### 1.流式套接字

socket 又称套接字，是网络应用层与传输层的编程接口，根据传输层协议的不同，可分为UDP Socket和TCP Socket，本次实验采用 TCP Socket。

流式套接字（SOCK\_STREAM）：该类套接字提供了面向连接的、可靠的、数据无错并且无重复的数据发送服务，而且发送的数据时按顺序被接受的。所有利用该套接字进行传递的数据均被视为连续的字节流且无长度限制。这对数据的稳定性、正确性和发送/接受顺序要求严格的应用十分适用，TCP协议使用该接口，但其对线路的占用率相对比较高。流式套接字的实现屡见不鲜，如远程登录（TELNET）、文件传输协议（FTP）等使用了流式套接字。



一个通用的 TCP Socket连接过程如下所示：



针对多人聊天室的设计，应该引入多线程技术，为每个客户端开启一个线程来单独维护相应的聊天。

## 2.消息设计

- server端通过输入目标IP地址和端口号来创建聊天室，并为每个接入的客户端创建一个单独的线程来维护其聊天窗口，并会在有客户端加入或退出时显示当前在线用户数量变化，同时server端也会显示客户端发在公屏上的消息
- client端通过输入目标连接的IP地址和端口号来连接相应的聊天室，然后客户端会要求输入一个昵称作为本客户端在聊天室的称呼。
- 发送消息：客户端通过使用特定格式的输入来进行发送消息的动作。通过输入“open:”前缀来向公屏发送消息，通过使用“用户名:”前缀来向某一个单独的用户发送消息。

- 获取当前在线用户列表：客户端通过输入“users”来获取当前在线用户列表，以便和某一个用户进行单独聊天。
- 客户端退出：客户端通过输入“quit”来进行退出聊天室的操作。

## 二、程序内容

---

### 1.程序设计

程序主要由server.h, server.cpp, client.h和client.cpp四个文件组成。其中Server需要长时间在线，接收消息并作为中转站，接收来自Client发送的聊天请求并进行转发。

#### Server

对于Server端，其功能顺序为：建立Socket -> 绑定本地信息 -> 启动监听 -> 监听到请求之后进行accept并分配新的socket用于负责转发 -> 对话结束后关闭Socket

- **建立socket**: 使用WinSock2.h和WS2tcpip.h库，建立serverSocket，并检查是否创建成功
- **绑定服务端信息**: 按照Socket中对socket绑定的要求，将IP、port、family等信息输入后使用bind函数进行绑定
- **监听**: 利用 while(true) 的形式进行阻塞式的监听，使用的函数就是listen函数，并设置一个队列长度是5
- **创建新socket、新线程并进行通讯**:
  - Server会提前创建好一个socket数组，数组长度和监听队列长度相同。
  - 创建一个单独的线程并利用accept函数，使用socket数组中的一个接收，并接收用户发来的信息进行名称的登记。在成功接收之后向客户端发送一条信息，告诉客户端已经成功建立链接；并在Server的命令行上输出一条显示连接成功和当前用户数量的消息。如果接收失败，则会显示error信息。
  - 连接成功之后创建一个新的线程，新线程只用来负责接收消息并显示，也就是线程中有一个while(true)，循环体内部就是recv和解析过后对应各种行为。主线程则是一直进行监听，在while(true)内运行listen函数，监听远程连接是否到来。

#### Client

对于客户端，其内容相比服务器端就简单一些，其功能顺序为：建立Socket -> 设定本地的地址信息 -> 设定远端的地址信息 -> 发送请求 -> 进行通话 -> 对话结束后关闭Socket。

- **建立socket**: 过程同Server中的建立socket步骤
- **设定本地地址信息**: 同样将IP、port等进行输入，但是注意这里和server的区别是这里没有bind函数
- **设定远端的地址信息**: 设置方式同本地，只是内容换成服务器端的地址信息
- **发送请求**: 使用connect函数建立链接，建立成功则会向Server发送一条自身信息登记的消息，并接收到远端发来的提示信息并进行相应显示；建立失败回显error信息
- **进行通讯**: 和Server端方式相同，创建一个新的线程进行接收和内容显示、本线程进行发送。
- **会话结束**: 在通讯过程中，如果Client发送的消息是quit，则结束本次通话。通话结束以后释放之前建立的两个socket，并结束本程序。或者是当Server关闭，则Client这边会显示连接失败。

## 2.程序实现

代码详见cpp源文件，此处只针对核心代码进行讲解

### (1).server

这段代码使用Windows的多线程机制 ( `CreateThread` ) 来处理不同客户端的消息。

#### 函数头部

```
DWORD WINAPI recvMessage(LPVOID tempPara) {  
    para* p = (para*)tempPara;  
    int num = p->number;  
}
```

- 该函数 `recvMessage` 是一个线程函数，用于处理来自客户端的消息。
- `tempPara` 是传入的参数，它被强制转换为 `para*` 类型，其中存储了与客户端相关的信息。
- `num` 表示当前处理的客户端编号。

#### 接收消息的主循环

```
while (1) {  
    if (recv(ClientSockets[num], SrecvBuf, SBUF_SIZE, 0) < 0) {  
    }  
}
```

- 使用 `recv` 函数从客户端套接字 `ClientSockets[num]` 接收数据，并存储在 `SrecvBuf` 缓冲区中。
- 如果 `recv` 返回小于 0 的值，说明接收失败（具体的错误处理未实现）。

#### 处理“获取用户列表”的请求

```
if (isGetList(SrecvBuf)) {  
    char usrList[SBUF_SIZE];  
    memset(usrList, 0, SBUF_SIZE);  
    usrList[0] = 'u'; // 第一个字母定义为 'u'  
    int posi = 1;  
    for (int tempIndex = 0; tempIndex < ListenMax; tempIndex++) {  
        if (ClientInformation[tempIndex].used) {  
            usrList[posi] = char(ClientInformation[tempIndex].len);  
            posi += 1;  
            strcat(usrList, ClientInformation[tempIndex].name);  
            posi += ClientInformation[tempIndex].len;  
        }  
    }  
    send(ClientSockets[num], usrList, sizeof(usrList), 0);  
}
```

- 如果接收到的消息是请求用户列表（判断 `isGetList(SrecvBuf)`），则服务器构造一个包含所有在线用户信息的 `usrList` 列表，并将其发送回请求的客户端。
- 列表中每个用户的名称以 `char` 类型的长度开头，紧跟着用户名。

## 处理用户退出的消息

```
else if (SrecvBuf[0] == 'q') {
    int len = int(SrecvBuf[1]);
    char* nameString = new char[len + 1];
    for (int index = 0; index < len; index++)
        nameString[index] = SrecvBuf[index + 2];
    nameString[len] = '\0';
    cout << "***用户: " << nameString << " 下线了.\n";
    ConnectedNumber -= 1;
    cout << "***当前用户数量为 " << ConnectedNumber << "\n";
}
```

- 如果收到的消息以 'q' 开头，表示某个用户退出。
- 服务器从消息中提取用户的名称，并通知所有其他客户端该用户已下线。
- 然后关闭该用户的套接字，并将该用户的信息从服务器列表中删除。

## 广播或私发消息

```
if (isOpen(dstName)) {
    // 公共消息广播给所有在线用户
    for (int i = 0; i < ListenMax; i++) {
        if (ClientInformation[i].used && ClientSockets[i] != INVALID_SOCKET) {
            send(ClientSockets[i], tempBuf, sizeof(tempBuf), 0);
        }
    }
} else {
    // 私发消息给目标用户
    int dst = 0;
    for (; dst < ListenMax; dst++) {
        if (ClientInformation[dst].used && issameStr(ClientInformation[dst].name,
            dstName)) {
            send(ClientSockets[ClientInformation[dst].number], tempBuf,
                sizeof(tempBuf), 0);
            break;
        }
    }
}
```

- 如果 `dstName` 是公屏聊天，则消息会被广播给所有在线用户。
- 否则，消息会发送给指定的目标用户。如果目标用户不在线或不存在，还会向发送方返回错误消息。

以下代码创建了一个服务器套接字，等待客户端连接，并为每个连接的客户端创建一个独立的线程来处理消息通信。

## 创建服务器套接字

```
ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
if (ServerSocket == INVALID_SOCKET) {
    cout << "socket error:" << WSAGetLastError() << endl;
    return 0;
}
```

- 使用 `socket` 函数创建一个套接字，指定使用 **IPv4** (`AF_INET`)，**流式套接字** (`SOCK_STREAM`，即 TCP 协议)。
- 如果 `socket` 返回 `INVALID_SOCKET`，表示创建失败，程序输出错误信息并返回。

### 初始化服务器地址信息

```
ServerAddr.sin_family = AF_INET;
getServerIP(ServerIP);
inet_pton(AF_INET, ServerIP, &ServerAddr.sin_addr.S_un.S_addr);
ServerPort = getPort();
ServerAddr.sin_port = htons(ServerPort);
```

- `ServerAddr.sin_family = AF_INET`：指定地址族为 IPv4。
- `inet_pton` 函数将服务器的 IP 地址从字符串转换为网络字节序格式，并存储在 `ServerAddr.sin_addr` 中。
- `htons(ServerPort)`：将服务器端口号转换为网络字节序。

### 绑定套接字到服务器地址

```
bind(ServerSocket, (SOCKADDR*)&ServerAddr, sizeof(SOCKADDR));
cout << "***服务器已成功启动，本聊天室的IP地址是：" << ServerIP << "\t端口号为：" <<
ServerPort << endl;
```

- 使用 `bind` 函数将服务器套接字绑定到指定的 IP 地址和端口号。
- 如果绑定成功，输出服务器启动成功的信息。

### 循环监听客户端连接请求

```
while (1) {
    if (ConnectedNumber == 0)
        cout << "***等待客户端接入中.....\n";
    listen(ServerSocket, ListenMax);
```

- 使用 `listen` 函数让服务器进入监听状态，`ListenMax` 指定最大连接队列长度。
- 如果当前没有已连接的客户端，输出提示信息等待新的连接。

### 接收客户端连接请求

```
for (int i = 0; i < ListenMax; i++)
    if (ClientSockets[i] == 0)
        break;
ClientSockets[i] = accept(ServerSocket, (SOCKADDR*)&ClientAddrs[i], &snaddr);
```

- 服务器通过 `accept` 函数接受客户端的连接请求，并为每个连接的客户端分配一个新的套接字。
- `ClientSockets[i]` 用来存储新连接的客户端套接字。

## 处理客户端连接成功

```
if (ClientSockets[i] != INVALID_SOCKET) {
    ClientInformation[i].number = i;
    ConnectedNumber += 1;
    cout << "***客户端连接成功，当前用户数量为: " << ConnectedNumber << endl;
```

- 如果 `accept` 返回的客户端套接字有效，表示连接成功。
- 更新已连接的客户端数量 `ConnectedNumber`，并在控制台输出提示信息。

## 接收客户端发送的用户名

```
recv(ClientSockets[i], SrecvBuf, SBUF_SIZE, 0);
if (isName(SrecvBuf)) {
    ClientInformation[i].len = int(SrecvBuf[1]);
    int indexOfName = 2;
    while (SrecvBuf[indexOfName] != '\0') {
        ClientInformation[i].name[indexOfName - 2] = SrecvBuf[indexOfName];
        indexOfName++;
    }
    ClientInformation[i].name[int(SrecvBuf[1]) + 1] = '\0';
    ClientInformation[i].used = true;
}
```

- 服务器通过 `recv` 函数接收客户端发送的用户名。
- 如果收到的消息是用户名 (`isName(SrecvBuf)`)，则提取并保存该用户名到 `ClientInformation[i].name`。

## 向客户端发送连接成功的消息

```
time_t now = time(0);
char* dt = ctime(&now);
memset(SsendBuf, 0, SBUF_SIZE);
strcat(SsendBuf, dt);
strcat(SsendBuf, hello);
send(ClientSockets[i], SsendBuf, SBUF_SIZE, 0);
```

- 获取当前时间，并将其作为连接成功的消息的一部分发送给客户端。

## 为每个客户端创建接收消息的线程

```
ClientThreads[i] = CreateThread(NULL, NULL, recvMessage, &ClientInformation[i],
0, NULL);
if (ClientThreads[i] == 0) {
    cout << "线程创建失败，程序终止! \n";
    ConnectedNumber -= 1;
    return -1;
}
```

- 每当有新客户端连接时，服务器为该客户端创建一个独立的线程，用于接收和处理消息。线程函数为 `recvMessage`，并将 `ClientInformation[i]` 作为参数传递。
- 如果线程创建失败，服务器输出错误信息，并减少已连接客户端数量。

## (2).client

以下代码实现客户端程序，能够连接到指定的聊天室服务器，并支持向所有人或指定用户发送消息、获取在线用户列表以及正确退出聊天室。

### 初始化Windows Sockets API

```
WSADATA wsadata;
if (WSAStartup(MAKEWORD(2, 2), &wsadata) != 0) {
    cout << "载入socket失败\n";
    system("pause");
    return -1;
}
```

- 使用 `WSAStartup` 函数加载Windows Sockets API，参数 `MAKEWORD(2, 2)` 表示使用Winsock 2.2 版本。
- 如果加载失败，程序输出错误信息并返回。

### 创建客户端套接字

```
LocalhostSocket = socket(AF_INET, SOCK_STREAM, 0);
if (LocalhostSocket == INVALID_SOCKET) {
    cout << "Socket创建失败，错误代码为：" << WSAGetLastError() << endl;
    WSACleanup();
    return -2;
}
```

- 使用 `socket` 函数创建一个套接字，指定使用 **IPv4** (`AF_INET`)，**流式套接字** (`SOCK_STREAM`，即TCP协议)。
- 如果 `socket` 返回 `INVALID_SOCKET`，表示创建失败，程序输出错误信息并清理资源。

### 绑定本地套接字地址

```
LocalhostAddr.sin_family = AF_INET;
LocalhostAddr.sin_port = htons(ClientPort);
inet_pton(AF_INET, LocalIP, &LocalhostAddr.sin_addr.S_un.S_addr);
```

- 设置本地套接字地址结构，指定地址族为 **IPv4**，端口号为 `ClientPort`（需提前设置），IP地址为本地 `LocalIP`。
- `inet_pton` 用于将IP地址从文本形式转换为网络字节序。

### 获取并设置目标服务器信息

```
RemoteAddr.sin_family = AF_INET;
char* DstIP = getIP(); // 获取目标服务器的IP地址
int htonsINT;
cin >> htonsINT;
RemoteAddr.sin_port = htons(htonsINT);
inet_pton(AF_INET, DstIP, &RemoteAddr.sin_addr.S_un.S_addr);
```

- 服务器地址族设置为 **IPv4**，端口号通过用户输入获得。



- 用户需要输入目标服务器的IP地址和端口号，并将这些信息转换为网络字节序后存储在 `RemoteAddr` 中。

### 获取客户端的用户名并封装

```
char inputName[CBUF_SIZE];
cin.getline(inputName, CBUF_SIZE);
userName[0] = char(getLen(inputName));
int indexOfInput = 0;
while (inputName[indexOfInput] != '\0') {
    userName[indexOfInput + 1] = inputName[indexOfInput];
    indexOfInput++;
}
userName[indexOfInput + 2] = '\0';
```

- 用户输入的昵称会保存到 `userName` 数组中，第一个字符保存用户名的长度。
- 随后封装用户信息用于发送给服务器，`sendName` 数组用于构建发送格式，第一位是 `'n'` 表示是名称信息，第二位是名称长度，后续是实际用户名。

### 连接远程服务器并发送用户名

```
if (connect(LocalhostSocket, (SOCKADDR*)&RemoteAddr, sizeof(RemoteAddr)) !=
    SOCKET_ERROR) {
    // 发送用户名给服务器
    send(LocalhostSocket, sendName, int(userName[0] + 2), 0);
}
```

- 使用 `connect` 函数尝试连接远程服务器，如果连接成功，则发送用户的昵称信息到服务器。
- 昵称信息是由前面构建的 `sendName` 数组传递的。

### 显示聊天室欢迎信息

```
time_t now = time(0);
char* dt = ctime(&now);
cout << "***连接成功！欢迎来到我们的聊天室！" << endl << "-----"
-----" << dt;
cout << "***本聊天室使用规则：" << endl
    << "***1、使用open:向所有人发送信息：" << endl
    << "***2、使用‘用户名’:向某位用户单独发送消息：" << endl
    << "***3、使用users来获取当前在线用户：" << endl
    << "***4、使用quit正确推出聊天室：" << endl;
```

- 连接成功后，客户端显示当前时间以及聊天室的使用规则。

### 创建接收消息的线程

```
HANDLE recv_thread = CreateThread(NULL, NULL, recvMessagec, NULL, 0, NULL);
```

- 使用 `CreateThread` 创建一个新的线程，用于接收来自服务器的消息。`recvMessagec` 是接收消息的处理函数。

## 发送消息或执行命令

```
while (1) {
    cin.getline(input, CBUF_SIZE);

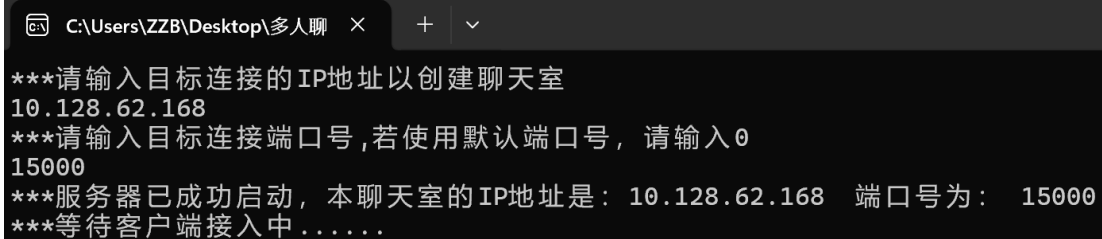
    if (!strcmp(input, "users\0")) {
        send(LocalhostSocket, input, 11, 0);
    } else if (ismyQuit(input)) {
        // 发送退出信息并关闭连接
    } else {
        // 发送消息给指定用户
    }
}
```

- 用户通过输入不同的指令来发送消息或执行命令：
  - 输入 `users` 命令会请求服务器返回在线用户列表。
  - 输入 `quit` 命令会通知服务器该客户端准备退出，并正确关闭连接。
  - 输入以用户名开头的消息将发送给指定的用户，消息格式为 `目标用户名:消息内容`。

## 三、实验结果

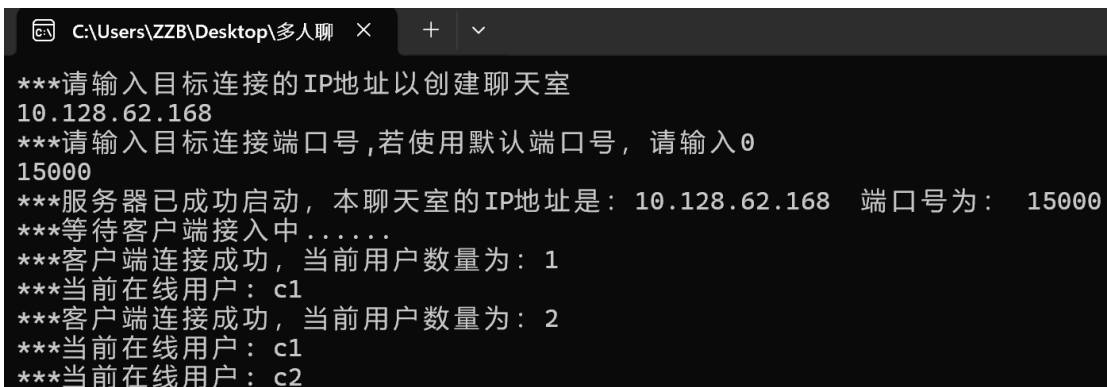
### 1. server

- 启动服务器



```
***请输入目标连接的IP地址以创建聊天室
10.128.62.168
***请输入目标连接端口号,若使用默认端口号, 请输入0
15000
***服务器已成功启动，本聊天室的IP地址是：10.128.62.168 端口号为：15000
***等待客户端接入中.....
```

- 多个用户登录



```
***请输入目标连接的IP地址以创建聊天室
10.128.62.168
***请输入目标连接端口号,若使用默认端口号, 请输入0
15000
***服务器已成功启动，本聊天室的IP地址是：10.128.62.168 端口号为：15000
***等待客户端接入中.....
***客户端连接成功，当前用户数量为：1
***当前在线用户：c1
***客户端连接成功，当前用户数量为：2
***当前在线用户：c2
```

- 收到公屏消息

```
C:\Users\ZZB\Desktop\多人聊 × + v
***请输入目标连接的IP地址以创建聊天室
10.128.62.168
***请输入目标连接端口号,若使用默认端口号, 请输入0
15000
***服务器已成功启动, 本聊天室的IP地址是: 10.128.62.168 端口号为: 15000
***等待客户端接入中.....
***客户端连接成功, 当前用户数量为: 1
***当前在线用户: c1
***客户端连接成功, 当前用户数量为: 2
***当前在线用户: c1
***当前在线用户: c2
-----Sat Oct 19 21:41:32 2024
来自c1的消息: 大家好, 我是c1!
```

- 用户退出聊天室

```
C:\Users\ZZB\Desktop\多人聊 × + v
***请输入目标连接的IP地址以创建聊天室
10.128.62.168
***请输入目标连接端口号,若使用默认端口号, 请输入0
15000
***服务器已成功启动, 本聊天室的IP地址是: 10.128.62.168 端口号为: 15000
***等待客户端接入中.....
***客户端连接成功, 当前用户数量为: 1
***当前在线用户: c1
***客户端连接成功, 当前用户数量为: 2
***当前在线用户: c1
***当前在线用户: c2
-----Sat Oct 19 21:41:32 2024
来自c1的消息: 大家好, 我是c1!
***用户: c1 下线了.
***当前用户数量为 1
```

## 2. client

- 连接到聊天室

```
C:\Users\ZZB\Desktop\多人聊 × + v
***请输入目标连接的聊天室服务器IP:
10.128.62.168
***请输入目标连接的聊天室服务器IP端口号:
15000
***请设置聊天室中显示的昵称:
c1
***连接成功! 欢迎来到我们的聊天室!
-----Sat Oct 19 21:40:22 2024
***本聊天室使用规则:
***1、使用open:向所有人发送信息;
***2、使用'用户名':向某位用户单独发送消息;
***3、使用users来获取当前在线用户;
***3、使用quit正确推出聊天室;
```

- 向公屏发送消息

```
C:\Users\ZZB\Desktop\多人聊 × + v

***请输入目标连接的聊天室服务器 IP:
10.128.62.168
***请输入目标连接的聊天室服务器 IP 端口号:
15000
***请设置聊天室中显示的昵称:
c1
***连接成功! 欢迎来到我们的聊天室!
-----Sat Oct 19 21:40:22 2024

***本聊天室使用规则:
***1、使用 open:向所有人发送信息;
***2、使用 '用户名':向某位用户单独发送消息;
***3、使用 users 来获取当前在线用户;
***3、使用 quit 正确推出聊天室;
open:大家好, 我是 c1!
-----Sat Oct 19 21:41:32 2024

来自 c1 的消息: 大家好, 我是 c1!
```

- 向某一用户发送消息

```
C:\Users\ZZB\Desktop\多人聊 × + v

***请输入目标连接的聊天室服务器 IP:
10.128.62.168
***请输入目标连接的聊天室服务器 IP 端口号:
15000
***请设置聊天室中显示的昵称:
c1
***连接成功! 欢迎来到我们的聊天室!
-----Sat Oct 19 21:40:22 2024

***本聊天室使用规则:
***1、使用 open:向所有人发送信息;
***2、使用 '用户名':向某位用户单独发送消息;
***3、使用 users 来获取当前在线用户;
***3、使用 quit 正确推出聊天室;
open:大家好, 我是 c1!
-----Sat Oct 19 21:41:32 2024

来自 c1 的消息: 大家好, 我是 c1!
c2:hello!C2,I am c1!
```

- 获取当前在线用户列表

```
C:\Users\ZZB\Desktop\多人聊 × + v

***请输入目标连接的聊天室服务器 IP:
10.128.62.168
***请输入目标连接的聊天室服务器 IP 端口号:
15000
***请设置聊天室中显示的昵称:
c1
***连接成功! 欢迎来到我们的聊天室!
-----Sat Oct 19 21:40:22 2024

***本聊天室使用规则:
***1、使用 open:向所有人发送信息;
***2、使用 '用户名':向某位用户单独发送消息;
***3、使用 users 来获取当前在线用户;
***3、使用 quit 正确推出聊天室;
open:大家好, 我是 c1!
-----Sat Oct 19 21:41:32 2024

来自 c1 的消息: 大家好, 我是 c1!
c2:hello!C2,I am c1!
users
1. c1
2. c2
```

- 退出聊天室

```
C:\Users\ZZB\Desktop\多人聊 × + v
***请输入目标连接的聊天室服务器IP:
10.128.62.168
***请输入目标连接的聊天室服务器IP端口号:
15000
***请设置聊天室中显示的昵称:
c1
***连接成功! 欢迎来到我们的聊天室!
-----Sat Oct 19 21:40:22 2024
***本聊天室使用规则:
***1、使用open:向所有人发送信息;
***2、使用'用户名':向某位用户单独发送消息;
***3、使用users来获取当前在线用户;
***3、使用quit正确推出聊天室;
open:大家好, 我是c1!
-----Sat Oct 19 21:41:32 2024
来自 c1的消息: 大家好, 我是c1!
c2:hello!C2,I am c1!
users
1. c1
2. c2
quit
***感谢使用!
连接中断! .
请按任意键继续. . . 请按任意键继续. . . |
```