

# 聊天室--实验报告

学号：2010764    姓名：高森森

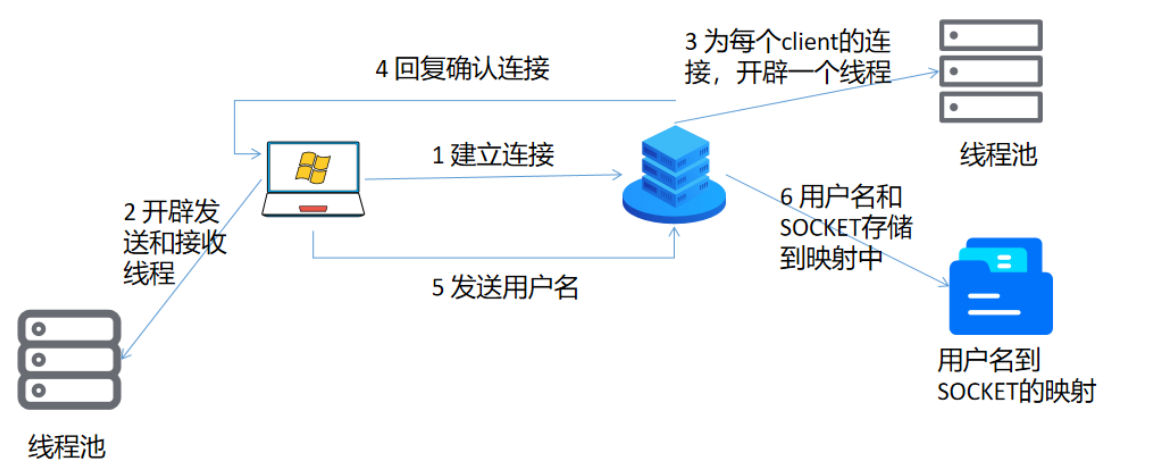
## 一、协议设计

### 1. 消息类型、语法、语义

消息类型	语法	语义
初始化用户名消息类型	用户建立与服务端的连接后，会提醒输入用户名，用户输入的字符串即为初始化用户名消息类型	用户输入的用户名字符串将存储到服务端的用户名映射中
退出群聊消息类型	"quit"	当用户发送"quit"消息，该用户的客户端会关闭，服务端检测到连接关闭，也会将服务端中存储该用户的映射置为"下线状态"
查询消息类型	"ls"	当用户发送"ls"消息，服务端会将当前所有在线用户信息进行处理，发送给该用户
私聊消息类型	"@user_name:message"	当用户使用"@user_name:message"，服务端会提取目标用户名和目标消息，将该消息转发给目标用户
群发消息类型	不包含上述特殊语法的消息默认为群发消息类型	当服务端接收到群发消息，将该群发消息转发给聊天室内除发送消息的所有用户

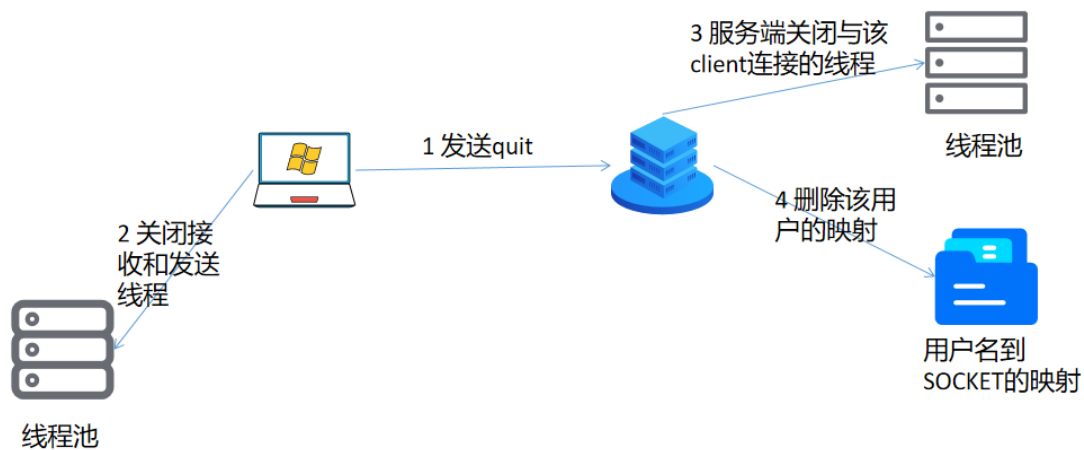
### 2. 时序

#### (1) 初始化用户名消息类型



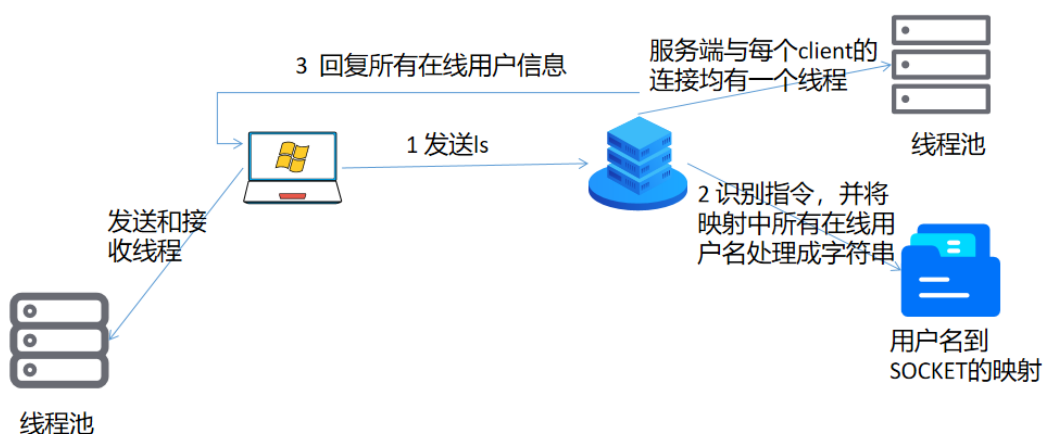
- 客户端初始化Winsock, 创建Socket, 并连接到服务端的IP地址和端口
- 客户端开辟两个线程, 其中一个线程用于向服务端发送消息, 另一个线程用于接收来自服务端的消息
- 服务端创建监听的Socket, 当接收到客户端连接, 服务端将开辟一个线程用于与该客户端的连接
- 当连接建立完成, 服务端将连接成功的消息回复给客户端
- 客户端提醒用户输入用户名, 并将该用户名发送到服务端
- 服务端创建用户名到SOCKET的映射, 并将接收到的用户名存储到该映射中

## (2) 退出群聊消息类型



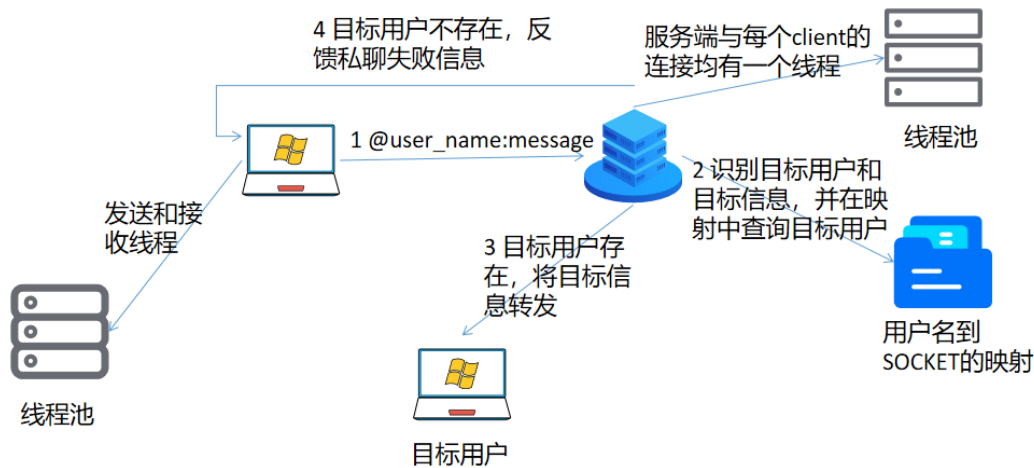
- 客户端发送"quit"消息到服务端
- 客户端检测发送的消息为"quit", 将发送消息和接收消息的线程均关闭, 客户端的Socket也将关闭
- 服务端接收到消息, 判断消息类型, 接收到的消息与"quit"相同, 判定为退出群聊消息类型
- 服务端关闭与该客户端连接的线程
- 服务端在用户名到SOCKET的映射中删除该客户端的映射

## (3) 查询消息类型



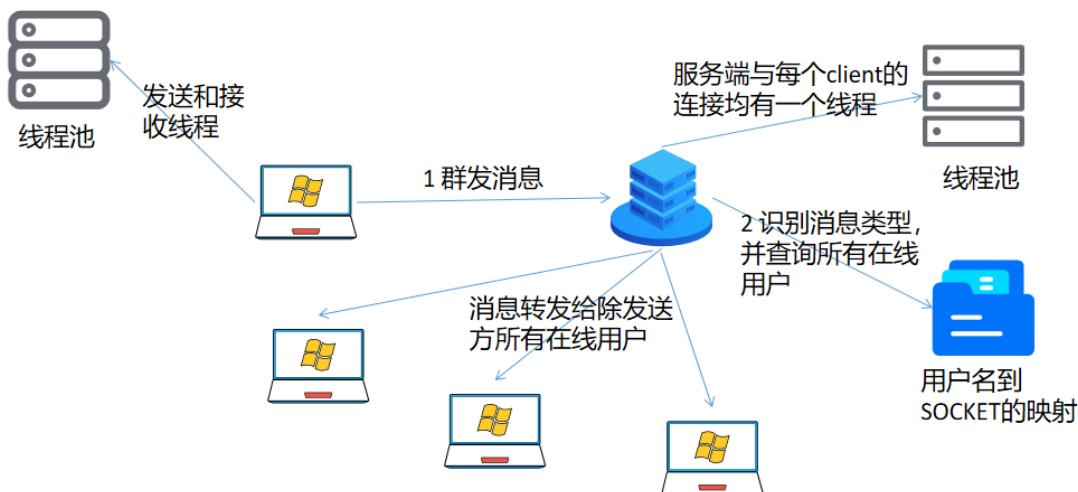
- 客户端发送"ls"消息到服务端
- 服务端接收到消息, 判断消息类型, 接收到的消息与"ls"相同, 判定为查询消息类型
- 服务端遍历用户名到SOCKET的映射, 将映射中所有在线用户名处理成字符串
- 将所有在线用户名信息回复给发送"ls"消息的客户端

## (4) 私聊消息类型



- 客户端发送"@user\_name:message"格式的消息
- 服务端接收到消息，判断消息类型，通过定位字符串中的'@'和':', 判定为私聊消息类型
- 服务端对消息字符串进一步处理，提取出'@'和':'之间的目标用户名和':'之后的目标信息
- 服务端在用户名到SOCKET的映射，查询目标用户名
- 如果在映射中能够查找到目标用户名，则将目标信息转发给目标用户
- 如果在映射中不能找到目标用户名，则将私聊失败的信息回复给发起私聊的客户端

## (5) 群发消息类型



- 客户端发送消息
- 服务端接收到消息，判断消息类型，当消息不是上述的几种类型，判定为群发消息类型
- 服务端在用户名到SOCKET的映射查询当前所有在线的用户
- 服务端将消息转发给除发送消息的所有在线用户

## 二、各模块功能

### 1. 客户端

#### (1) 创建Socket、建立连接、初始化用户名、开辟发送和接收线程

```
//客户端创建Socket
ClientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (ClientSocket == INVALID_SOCKET) {
    cout << "客户端Socket创建失败" << endl;
    WSACleanup();
    return 0;
}
```

```

}

//要连接服务器的IP地址，端口号
struct sockaddr_in Server;
Server.sin_family = AF_INET;
inet_pton(AF_INET, "127.0.0.1", &Server.sin_addr.s_addr);
Server.sin_port = htons(CONNECT_PORT);

//连接服务端
isError = connect(ClientSocket, (SOCKADDR*)&Server, sizeof(Server));
if (isError == SOCKET_ERROR) {
    cout << "连接服务端失败"<< endl;
    closesocket(ClientSocket);
    WSACleanup();
    return 1;
}

cout << "请输入客户端用户名" << endl;
cin.getline(user_name, 20);

//将用户名发送给服务端
send(ClientSocket, user_name, 20, 0);
//-----
// 创建两个线程，一个接收线程，一个发送线程
HANDLE hThread[2];
hThread[0] = CreateThread(NULL, 0, Recv, (LPVOID)&ClientSocket, 0, NULL);
hThread[1] = CreateThread(NULL, 0, Send, (LPVOID)&ClientSocket, 0, NULL);

WaitForMultipleObjects(2, hThread, TRUE, INFINITE);

```

首先客户端使用socket函数创建Socket，并判断创建的Socket是否是有效的；之后便通过connect函数连接到服务端的IP地址和端口，并判断是否连接成功。为了初始化客户端的用户名，并在服务端存储客户端的用户名，客户端将提醒用户输入用户名，并将用户名发送到服务端，服务端在接收到该用户名后，将它和连接的客户端Socket存储到用户名到SOCKET的映射中。

最后，我们创建两个线程，一个线程绑定客户端的发送消息模块，用于向服务端发送消息；另一个线程绑定客户端的接收消息模块，用于接收处理服务端发送到客户端的消息。

## (2) 接收消息模块

```

DWORD WINAPI Recv(LPVOID lparam_socket) {
    int isRecv; //判断是否接收到消息
    SOCKET* recvSocket = (SOCKET*)lparam_socket; //使用指针的原因，需要指向连接socket的地址
    while (1)
    {
        char recvMessage[MESSAGE_LEN];
        isRecv = recv(*recvSocket, recvMessage, MESSAGE_LEN, 0);
        if (isRecv > 0 && flag == 1) //接收到消息，且没有退出群聊，因为是双线程，所以需要判断flag
        {
            SYSTEMTIME time;
            GetLocalTime(&time);
            cout << time.wYear << ":" << time.wMonth << ":"
                << time.wDay << ":" << time.wHour << ":"
                << time.wMinute << ":" << time.wSecond << endl;
            cout << "接收到消息--" << recvMessage << endl;

```

```

        cout << "-----" << endl;
        cout << "请输入你要发送的消息" << endl;
        msgRemind = false;
    }
    else
    {
        closesocket(*recvSocket);
        return 0;
    }
}
}

```

我们使用recv函数接收来自服务端的消息，并判断是否接收成功。如果接收成功，我们使用SYSTEMTIME类型的变量和GetLocalTime函数获取当前的时间，随后将当前时间的年、月、日、时、分、秒输出作为时间标签。

这里涉及到两个实现细节：

- 我们定义了一个全局变量flag，用于判断当前客户端是否关闭连接。因为发送和接收消息是开辟的双线程，发送消息可通过发送"quit"消息直接关闭连接，因此在输出接收到的消息时需要判断该客户端是否还在线。
- 我们定义了一个全局变量msgRemind，用于判断是否已经输出"请输入你要发送的消息"的提示信息。因为发送和接收消息是开辟的双线程，且两个线程均有该提示信息，若不加判断，将导致提示信息混乱的现象。

### (3) 发送消息模块

```

DWORD WINAPI Send(LPVOID lparam_socket) {
    int isSend;
    SOCKET* sendSocket = (SOCKET*)lparam_socket; //使用指针的原因，需要指向连接
    socket的地址

    while (1)
    {
        if (msgRemind)
        {
            cout << "请输入你要发送的消息" << endl;
            char sendMessage[MESSAGE_LEN];
            cin.getline(sendMessage, MESSAGE_LEN); //getline可以识别空格，遇到换行自动结束
            msgRemind = true;

            if (string(sendMessage) == "quit") //表示该用户要退出群聊
            {
                flag = 0;
                closesocket(*sendSocket);
                WSACleanup(); //调用WSACleanup函数来解除与Socket库的绑定并且释放Socket库所
                占用的系统资源
                return 0;
            }
            else
            {
                isSend = send(*sendSocket, sendMessage, MESSAGE_LEN, 0);
                if (isSend == SOCKET_ERROR) //-1表示出错
                {
                    cout << "发送信息失败:" << WSAGetLastError() << endl;
                    closesocket(*sendSocket);
                    WSACleanup(); //调用WSACleanup函数来解除与Socket库的绑定并且释放Socket
                    库所占用的系统资源
                }
            }
        }
    }
}

```

```

        return 0;
    }
    else
    {
        SYSTEMTIME time;
        GetLocalTime(&time);
        cout << time.wYear << ":" << time.wMonth << ":" << time.wDay << ":" << time.wHour << ":" << time.wMinute << ":" << time.wSecond << endl;
        cout << "消息发送成功" << endl;
        cout << "-----" << endl;
        Sleep(100);
    }
}
}
}

```

与接收消息模块相似，发送消息模块也通过msgRemind判断是否需要进行用户输入提示信息。随后判断用户输入消息的类型，如果是"quit"，则关闭客户端与服务端的连接；如果不是"quit"，则将该消息发送到服务端，并判断消息是否发送成功。如果发送成功，则将时间标签和消息发送成功在客户端进行输出。

## 2. 服务端

### (1) 创建监听Socket和多线程连接

```

// 创建一个监听的SOCKET
// 如果有connect的请求就新创建一个线程
SOCKET ListenSocket;
ListenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (ListenSocket == INVALID_SOCKET) {
    cout << "监听Socket创建失败" << endl;
    WSACleanup();
    return 0;
}
//使用多线程处理多个请求
while (1) {
    sockaddr_in client_addr;
    int len = sizeof(sockaddr_in);

    SOCKET AcceptSocket = accept(ListenSocket, (SOCKADDR*)&client_addr,
    &len);
    if (AcceptSocket == INVALID_SOCKET) {
        cout << "接收客户端Socket失败" << endl;
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }
    else {
        // 创建线程，并且传入与client通讯的套接字
        HANDLE hThread = CreateThread(NULL, 0, handlerRequest,
        (LPVOID)AcceptSocket, 0, NULL);
        CloseHandle(hThread);
    }
}
}

```

对于服务端，创建一个监听的Socket用于监听要与服务端连接的客户端Socket，并判断创建的监听Socket是否有效。随后循环判断是否监听到了客户端连接，如果监听到了客户端连接，先判断客户端Socket是否为有效的Socket，如果有效，则开辟一个线程处理与该客户端的连接。

## (2) 与客户端交互模块

```
DWORD WINAPI handlerRequest(LPVOID lparam)
{
    SOCKET ClientSocket = (SOCKET)(LPVOID)lparam;
    client_map[ClientSocket] = 1; //表示client在线

    char user_name[20]; //用于接收用户名

    recv(ClientSocket, user_name, 20, 0); //接收来自客户端的用户名

    //把名字和SOCKET绑定
    client_name[string(user_name)] = ClientSocket;

    SYSTEMTIME time;
    GetLocalTime(&time);
    cout << time.wYear << ":" << time.wMonth << ":"
          << time.wDay << ":" << time.wHour << ":"
          << time.wMinute << ":" << time.wSecond << endl;

    cout << "日志--用户: " << user_name << "加入聊天" << endl;
    cout << "-----" << endl;

    int isRecv, isSend;
    int flag = 1; //该客户端是否退出
    do {
        char recvMessage[MESSAGE_LEN];
        char sendMessage[MESSAGE_LEN];
        char speMessage[MESSAGE_LEN];
        isRecv = recv(ClientSocket, recvMessage, MESSAGE_LEN, 0);

        if (isRecv > 0)
        {
            GetLocalTime(&time);
            cout << time.wYear << ":" << time.wMonth << ":"
                  << time.wDay << ":" << time.wHour << ":"
                  << time.wMinute << ":" << time.wSecond << endl;
            cout << "日志--用户" << user_name << "的消息:" << recvMessage << endl;
            cout << "-----" << endl;
            if (string(recvMessage) == "ls") //查看当前在线所有用户
            {
                listAll(user_name);
            }
            else if (recvMessage[0] == '@')
            {
                sendTotarget(user_name, recvMessage);
            }
            else
            {
                sendToall(user_name, recvMessage);
            }
        }
    }
    else
```

```

    {
        flag = 0; //退出了群聊
        client_map[ClientSocket] = 0;
    }
} while (isRecv != SOCKET_ERROR && flag != 0);

GetLocalTime(&time);
cout << time.wYear << ":" << time.wMonth << ":"
    << time.wDay << ":" << time.wHour << ":"
    << time.wMinute << ":" << time.wSecond << endl;
cout << "日志--用户: " << user_name << "离开了聊天室" << endl;
cout << "-----" << endl;

closesocket(ClientSocket);
return 0;
}

```

在服务端，我们具有两个映射client\_map和client\_name。其中client\_map为SOCKET到int的映射，用于判断客户端是否在线；client\_name为string到SOCKET的映射，用于存储用户名到客户端Socket的映射。

服务端先接收来自客户端的用户名，并将用户名和Socket存储到client\_name映射，并在client\_map中标记该用户在线。随后将时间标签和用户加入群聊的消息在服务端进行输出。

接着是循环处理接收到客户端的消息。当服务端接收到客户端发送的消息，将时间标签、接收到的消息和发送客户端的信息进行打印输出，之后在服务端判断消息的类型。如果接收到的消息为"ls"，则为查询消息类型，调用listAll()函数处理（下面将详细介绍）；如果接收到的消息为'@'开头，则判定为私聊消息类型，调用sendTotarget()函数发送给目标用户；如果接收到的消息不属于上述类型，则为群聊消息类型，调用sendToall()函数转发给当前所有在线用户。

如果服务端检测到客户端关闭连接，将时间标签和客户离开信息作为日志进行输出。

### (3) 查询所有在线用户模块

```

//列出当前所有用户
void listAll(char* user_name)
{
    char sendMessage[MESSAGE_LEN];
    strcpy_s(sendMessage, "当前在线用户: ");

    //找出所有在线用户
    for (auto it : client_name)
    {
        if (client_map[it.second] == 1) //说明该用户在线
        {
            strcat_s(sendMessage, it.first.data());
            strcat_s(sendMessage, " ");
        }
    }

    send(client_name[user_name], sendMessage, MESSAGE_LEN, 0);
}

```

该模块的参数user\_name是发送"ls"消息的用户名，我们遍历client\_name映射，并通过client\_map映射判断当前所有在线用户，将它们的用户名拼接成一个字符串，通过send()函数回复给用户\_name客户端。



#### (4) 发送给目标用户模块

```
//私聊信息
void sendTotarget(char* send_user, char* msg)//msg为recvMessage
{
    char target_user[20];
    char sendMessage[MESSAGE_LEN];
    for (int j = 1; j < MESSAGE_LEN; j++) {
        if (msg[j] == ':') { // ":"后为发送的消息
            target_user[j - 1] = '\0';
            for (int z = j + 1; z < MESSAGE_LEN; z++) {
                sendMessage[z - j - 1] = msg[z];
            }
            strcat_s(sendMessage, "(来自");
            strcat_s(sendMessage, send_user);
            strcat_s(sendMessage, "的私聊信息)\0");
            break;
        }
        else {
            target_user[j - 1] = msg[j];
        }
    }
    if (client_name.find(string(target_user)) == client_name.end() || client_map[client_name[string(target_user)]] == 0)
    {
        SYSTEMTIME time;
        GetLocalTime(&time);
        cout << time.wYear << ":" << time.wMonth << ":" << time.wDay << ":" << time.wHour << ":" << time.wMinute << ":" << time.wSecond << endl;
        cout << "日志--用户"<<send_user<<"私聊用户"<< target_user<<"失败，因为该用户不存在或者已下线"<< endl;
        cout << "-----" << endl;

        //还要把信息发送给send_user
        strcpy_s(sendMessage, "私聊失败：用户");
        strcat_s(sendMessage, target_user);

        strcat_s(sendMessage, "不存在或者已下线");

        int isSend = send(client_name[string(send_user)], sendMessage, MESSAGE_LEN, 0);
        if (isSend == SOCKET_ERROR)
        {
            GetLocalTime(&time);
            cout << time.wYear << ":" << time.wMonth << ":" << time.wDay << ":" << time.wHour << ":" << time.wMinute << ":" << time.wSecond << endl;
            cout << "日志--私聊失败信息反馈给用户" << send_user << "失败" << endl;
            cout << "-----" << endl;
        }
    }
    else
    {
        SOCKET target = client_name[string(target_user)];
        if (client_map[target] == 1)
        {
```

```

        int isSend = send(target, sendMessage, MESSAGE_LEN, 0);
        if (isSend == SOCKET_ERROR)
        {
            SYSTEMTIME time;
            GetLocalTime(&time);
            cout << time.wYear << ":" << time.wMonth << ":"
                << time.wDay << ":" << time.wHour << ":"
                << time.wMinute << ":" << time.wSecond << endl;
            cout << "日志--用户" << send_user << "向用户" << target_user << "发送信息失败" << endl;
            cout << "-----" << endl;
        }
    }
}
}

```

该模块的两个参数，其中send\_user是指发送消息的用户名，msg指未经处理的消息。我们通过定位msg中的'@'和':'两个符号，识别消息中的用户名和待转发的消息。接着我们去client\_name映射中查找目标用户，如果未找到或者找到后在client\_map映射中显示该用户已经下线，则将私聊失败的信息回馈给send\_user用户，并在服务端将时间标签和私聊失败作为日志进行输出。

如果目标用户存在且未下线，则将待转发的消息通过服务端转发给目标用户。

## (5) 群发模块

```

//将信息发送给所有用户
void sendToAll(char* send_user, char* msg) //msg为recvMessage
{
    char sendMessage[MESSAGE_LEN];
    strcpy_s(sendMessage, "用户");
    strcat_s(sendMessage, send_user); // data函数直接转换为char*
    strcat_s(sendMessage, ": ");
    strcat_s(sendMessage, msg);

    for (auto it : client_name)
    {
        if (it.first != string(send_user) && client_map[it.second] == 1)
        {
            int isSend = send(it.second, sendMessage, MESSAGE_LEN, 0);
            if (isSend == SOCKET_ERROR)
            {
                SYSTEMTIME time;
                GetLocalTime(&time);
                cout << time.wYear << ":" << time.wMonth << ":"
                    << time.wDay << ":" << time.wHour << ":"
                    << time.wMinute << ":" << time.wSecond << endl;
                cout << "日志--用户" << send_user << "向用户" << it.second << "发送信息失败" << endl;
                cout << "-----" << endl;
            }
        }
    }
}
}

```

该模块的两个参数，其中send\_user指发送信息的用户名，msg指待群发的消息。我们遍历client\_name映射，并排除掉send\_user用户，随后将msg发送给经client\_map判断所有在线的用户。如果服务端转发给某个用户时失败，将时间标签和失败信息作为日志进行输出。

## 三、程序界面展示及运行说明

### 1. 多客户端连接到服务端

<pre>C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 a ***** welcome user: a ***** 使用ls查询当前在线所有用户 ***** ***** 使用@用户名:信息  私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息</pre>	<pre>C:\Users\gaosensen\Desktop\流式套接字客户端.exe 2022:10:22:19:3:18 日志一用户: a加入聊天 ----- 2022:10:22:19:3:32 日志一用户: b加入聊天 ----- 2022:10:22:19:3:39 日志一用户: c加入聊天 -----</pre>
<pre>C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 b ***** welcome user: b ***** 使用ls查询当前在线所有用户 ***** ***** 使用@用户名:信息  私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息</pre>	<pre>C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 c ***** welcome user: c ***** 使用ls查询当前在线所有用户 ***** ***** 使用@用户名:信息  私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息</pre>





a,b,c三个客户端依次连接到服务端，服务端有相应用户加入聊天的日志输出

### 2. 客户端使用查询类型消息

```
C:\Users\gaosensen\Desktop\流式套接字客户端.exe
请输入客户端用户名
a
*****
welcome user: a
***** 使用ls查询当前在线所有用户 *****
***** 使用@用户名:信息  私聊发送信息 *****
***** 使用quit退出聊天室 *****
请输入你要发送的消息
ls
2022:10:22:19:6:48
消息发送成功
-----
2022:10:22:19:6:48
接收到消息--当前在线用户: a b c
-----
请输入你要发送的消息
```





客户端a发送查询类型消息"ls"，并得到所有当前在线用户的消息反馈。

### 3. 客户端使用群发类型消息

 C:\Users\gaosensen\Desktop\流式套接字客户端.exe 2022:10:22:19:6:48 接收到消息--当前在线用户: a b c ----- 请输入你要发送的消息 2022:10:22:19:9:28 接收到消息--用户b: 大家好 ----- 请输入你要发送的消息	 C:\Users\gaosensen\Desktop\流式套接字.exe 2022:10:22:19:6:48 日志--用户a的消息:1s ----- 2022:10:22:19:9:28 日志--用户b的消息: 大家好 ----- -
 C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 b ***** welcome user: b ***** 使用1s查询当前在线所有用户 ***** ***** 使用0用户名:信息 私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息 大家好 2022:10:22:19:9:28 消息发送成功 ----- 请输入你要发送的消息	 C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 c ***** welcome user: c ***** 使用1s查询当前在线所有用户 ***** ***** 使用0用户名:信息 私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息 2022:10:22:19:9:28 接收到消息--用户b: 大家好 ----- 请输入你要发送的消息


客户端b发送群发类型消息"大家好", 客户端a, c均接收到该消息, 服务端也有b客户端发送消息的日志。

## 4. 客户端使用私聊类型消息

 C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入你要发送的消息 2022:10:22:19:12:26 接收到消息--a, 你好(来自c的私聊信息) ----- 请输入你要发送的消息	 C:\Users\gaosensen\Desktop\流式套接字.exe 2022:10:22:19:9:28 日志--用户b的消息: 大家好 ----- 2022:10:22:19:12:26 日志--用户c的消息:@a:a, 你好 -----
 C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 b ***** welcome user: b ***** 使用1s查询当前在线所有用户 ***** ***** 使用0用户名:信息 私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息 大家好 2022:10:22:19:9:28 消息发送成功 ----- 请输入你要发送的消息	 C:\Users\gaosensen\Desktop\流式套接字客户端.exe 请输入客户端用户名 c ***** welcome user: c ***** 使用1s查询当前在线所有用户 ***** ***** 使用0用户名:信息 私聊发送信息 ***** ***** 使用quit退出聊天室 ***** 请输入你要发送的消息 2022:10:22:19:9:28 接收到消息--用户b: 大家好 ----- 请输入你要发送的消息 @a:a, 你好 2022:10:22:19:12:26 消息发送成功 ----- 请输入你要发送的消息 -

客户端c发送私聊类型消息"@a:a, 你好", 可以看到只有a接收到了消息, 而b没有接收到, 且服务端有相应的日志输出。

## 5. 客户端使用退出群聊类型消息

 C:\Users\gaosensen\Desktop\流式套接字.exe

```
2022:10:22:19:15:55
日志--用户: a离开了聊天室
-----
```

a客户端发送退出群聊类型消息"quit", 服务端显示a离开聊天室的日志。

也可以通过直接关闭客户端终端窗口的方式退出群聊。

```
C:\Users\gaosensen\Desktop\流式套接字客户端.exe
请输入客户端用户名
b
*****
***** welcome user: b *****
***** 使用ls查询当前在线所有用户 *****
***** 使用@用户名:信息 私聊发送信息 *****
***** 使用quit退出聊天室 *****
请输入你要发送的消息
大家好
2022:10:22:19:9:28
消息发送成功

请输入你要发送的消息
@a:你还在吗?
2022:10:22:19:19:25
消息发送成功

2022:10:22:19:19:25
接收到消息--私聊失败: 用户a不存在或者已下线
请输入你要发送的消息

C:\Users\gaosensen\Desktop\流式套接字.exe
2022:10:22:19:15:55
日志--用户: a离开了聊天室
-----
2022:10:22:19:19:25
日志--用户b的消息:@a:你还在吗?
-----
2022:10:22:19:19:25
日志--用户b私聊用户a失败, 因为该用户不存在或者已下线
-----
```

当a客户端已经离开群聊后，b客户端对其私聊，将会得到a不存在或者已下线的消息反馈，服务端日志中也会显示私聊失败的相关信息。

## 四、实验过程中遇到的问题及分析

### 1. 提示用户输入紊乱

由于客户端的发送消息和接收消息使用两个线程，并且在发送完消息和接收完消息，都要对用户进行“请输入你要发送的消息”的提示，有时候会出现紊乱的情况，即发送消息线程未输入消息但是已经进行了“请输入你要发送的消息”的提示，这时接收消息的线程接收到消息，在输出接收到的消息后，也会进行“请输入你要发送的消息”的提示，这时就会出现紊乱。

我们对程序进行改进，设置变量msgRemind用于判断是否已经进行提示，成功解决该问题。

### 2. 私聊类型消息没有反馈

由于发送私聊类型消息的用户可能输入错误的用户名，或者该用户已经下线，我们需要对发送私聊类型消息的用户一个反馈。

我们对程序进行改进，如果私聊目标用户不存在或者已经下线，会给发送私聊消息的用户“用户不存在或者已下线”的反馈，如果目标用户存在，则不会有反馈。