# 计算机网络课程实验报告

郭裕彬 2114052 物联网工程

## 实验1：利用Socket编写一个聊天程序

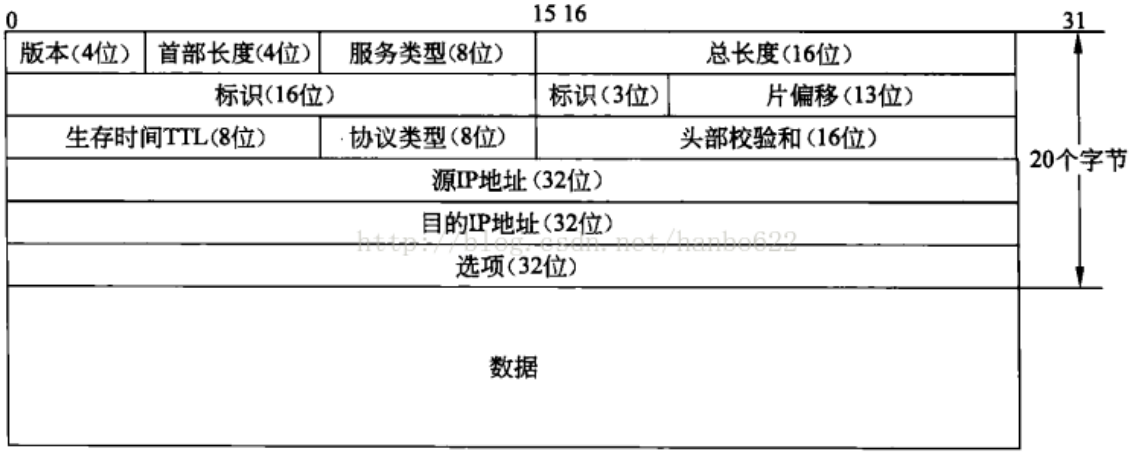
### 实验要求：

- 给出你聊天协议的完整说明。
- 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。
- 使用流式套接字、采用多线程（或多进程）方式完成程序。
- 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。
- 完成的程序应能支持多人聊天，支持英文和中文聊天。
- 编写的程序应该结构清晰，具有较好的可读性。
- 在实验中观察是否有数据的丢失，提交源码和实验报告。

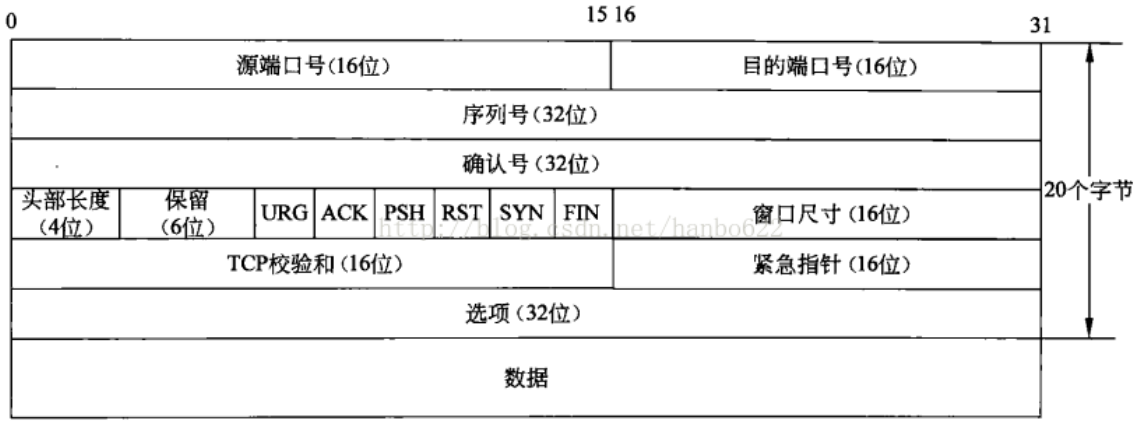
## 协议设计

### 整体结构

- 数据包首部由IPv4数据包组成，其中协议结构如下：




- 传输层协议使用的是TCP协议，其结构如下：

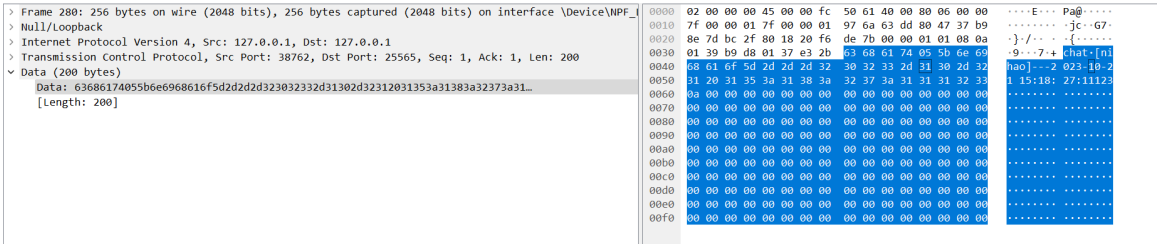



- 之后的包内容就是本次实验中自主设计的消息数据部分，定义了一个消息结构体：

```
  string type;//消息类型
  string t;//发送时间
  string text;//文本
  string usr;//发送用户
  uint8_t tl;//用户昵称长度
}Message;
```

并设计了对应的转换函数将其转换成字符串流传输，在会在后面详细说明。

附上通过Wireshark监听127.0.0.1的本地Loopback接口的25565号端口，抓取到的服务端向客户端转发消息的解包结果：



## 消息类型

消息类型由Message结构体中的type部分确定，共有`chat`、`exit`、`join`三种类型，分别对应正常的客户端与服务端进行聊天通信、客户端与服务端断开连接离开聊天室和客户端连接服务端进入聊天室的消息。

## 语法语义

消息通过转换函数从结构体转为字符串后，其顺序构成为

- 消息类型，占两个字节，由四个字母表示，"chat"表示正常消息，"join"表示进入聊天室，"exit"表示退出聊天室；
- 用户昵称长度，占一个字节，以十六进制数形式存储；
- 一个'['字符；
- 用户昵称，最长长度为15个字节；
- 一个"]---"字符串；
- 发送时间，以"yyyy-mm-dd hh:mm:ss"为格式的字符串；
- 一个':'字符；
- 消息文本，最长长度为150个字节；

## 时序

- 首先遵循TCP协议，进行客户端与服务端的三次握手
- 客户端获取用户输入的昵称后，向服务端发送一个带有消息的数据包，包中的消息类型赋值为`join`，正常存储用户昵称长度和昵称、发送时间，消息文本为空。
- 服务端收到这个数据包后，首先解析出消息类型为`join`，直接读取数据包中的用户昵称，向连接到服务端的所有客户端发送消息，内容为"xxx进入了聊天室"。
- 客户端可以向服务端正常发送聊天消息，输入消息文本后，客户端将对应的用户昵称、长度、发送时间、消息文本等转换成字字符串附加到数据包中发送给服务端。
- 服务端收到数据包后，解析出消息类型为`chat`，向连接到服务端的所有客户端转发消息。

- 客户端获取到用户输入".exit"的消息文本后，向服务端发送一个带有消息的数据包，包中的消息类型赋值为`exit`，正常存储用户昵称长度和昵称、发送时间，文本为空。
- 服务端收到这个数据包后，解析出消息类型为`exit`，直接读取数据包中的用户昵称，断开对应的客户端的连接，向连接到客户端的所有客户端发送消息，内容为"xxx离开了聊天室"。

## 各模块功能

总体来说，服务端使用每个客户端一个线程的方式来实现多客户端通信，首先加载和初始化socket，绑定服务端的socket和IP地址，无限循环监听，等待客户端的连接，当有新的连接进入时，为其创建一个线程来维护与这个客户端的通信。在线程内部，无限循环完成接收该客户端消息和转发消息至所有连接的客户端的操作直至该客户端断开连接，之后关闭socket，清理线程。

客户端加载和初始化socket，绑定socket和IP地址，尝试连接服务端。分别使用接收和发送两个线程来处理消息的收发，发送线程负责获取用户输入并发送三类消息；接收线程负责从服务端接收数据包输出字符串，两个线程的等待时间均为无限，直至用户输入".exit"退出聊天室。

### 辅助处理函数

结构体转换为字符串通过定义的函数messageToStr()来实现，返回的字符串形式为[name]---yyyy:mm:dd hh:mm:ss:text；type字段恒为四个字符，因此可以通过读取字符串第五个字节的数据获取用户昵称的长度，从而准确截取到用户昵称。字符串第五个字节以后的数据为要显示在窗口中的文本，通过定义函数来截取。

```
//从接收的字符串中获得用户的昵称 Server.cpp
string strToMessName(string s)
{
    string name;
    int namelen=s[4];
    name=s.substr(6,namelen);
    return name;
}
```

```
//从接收的字符串中获取消息形式：[xxx]---yyyy:mm:dd hh:mm:ss:text
Server.cpp
string strToSubstr(string s)
{
    string ss;
    ss=s.substr(5);
    return ss;
}

//消息转换为可发送的字符串 Client.cpp
string messageToStr(Message msg)
{
    string str;
    str+=msg.type+(char)msg.tl+'['+msg.usr+(string)"]---
"+msg.t+':'+msg.text+'\n';
    return str;
};
```

## 服务端主函数循环客户端连接

在一系列初始化完成后，服务端循环监听来自客户端的连接，每当使用accept()监听到并完成一个新的连接请求时，使用一个usr结构体记录socket和地址信息，将这个结构体放入Vector中保存记录，为其创建一个新的处理线程，负责接收其消息和转发消息到所有已连接的客户端。

```
//定义用户结构体
typedef struct{
    SOCKET sock;
    sockaddr_in addr;
}user;

//in main()
while(1)
    {
        int len=sizeof(sockaddr_in);
        client=accept(ListenSock,(sockaddr*)(&ClientAddr),&len);
        if(client==INVALID_SOCKET)
        {
```

```
            cout<<"创建客户失败"<<endl;
            break;
        }
        //为监听到的用户创建一个处理线程
        user usr;
        usr.addr=ClientAddr;
        usr.sock=client;
        ClientVector.push_back(usr);
        hThread=CreateThread(NULL,0,HandleThread,
(LPVOID)&usr,0,NULL);
        if(hThread==NULL)
        {
            cout<<"创建线程失败"<<endl;
            break;
        }
    }
```

### 服务端处理线程负责收发消息

对线程处理函数传入user*类型的参数后，该线程循环接收并转发消息，使用recv()函数接收，成功接收则将接收数组使用strToSubstr()函数赋值为截取的显示文本，判断消息类型是否为退出或进入，是则将接收数组的内容更新为对应的“[昵称]+退出/进入了聊天室”，对于退出的消息类型，关闭该socket；获取当前时间，向ClinetVector容器内所有的socket转发显示文本，同时服务器端输出日志“yyyy-mm-dd hh:mm:ss-来自[地址]的消息转发至[地址]成功/失败”，其中地址的获取使用了inet_ntoa函数来转换成带小数点的地址字符串。由于测试都在本机上进行，因此显示的地址都为定义的127.0.0.1。

```
DWORD WINAPI HandleThread(LPVOID LpParam)
{
    int RecFlag;
    int SendFlag;
    user* client =(user*)LpParam;
    SOCKET sock=client->sock;
    char RecBuf[MAXBUFSIZE];
    char SendBuf[MAXBUFSIZE];

    //循环接收并转发消息
    while(1)
```

```
    {
        RecFlag=recv(sock,RecBuf,MAXBUFSIZE,0);
        if(RecFlag!=SOCKET_ERROR)
        {
            string text=RecBuf;
            strcpy_s(RecBuf,strToSubstr(text).c_str());

            //消息类型为退出时，转发退出信息
            if(strToMessType(text)==0)
            {
                string exitStr=strToMessName(text)+"离开了聊天室\n";
                strcpy_s(RecBuf,exitStr.c_str());
                closesocket(sock);
            }
            //消息类型为进入时，转发进入信息
            else if(strToMessType(text)==2)
            {
                string exitStr=strToMessName(text)+"进入了聊天室\n";
                strcpy_s(RecBuf,exitStr.c_str());
            }
            //正常转发信息
            time_t t=time(0);
            char tt[100]={0};
            for(int i=0;i<ClientVector.size();i++)
            {
                strftime(tt,sizeof(tt),"%Y-%m-%d
%H:%M:%S",localtime(&t));

 SendFlag=send(ClientVector[i].sock,RecBuf,MAXBUFSIZE,0);
                if(SendFlag==SOCKET_ERROR)
                {
                    cout<<tt<<"-来自"<<inet_ntoa(client-
>addr.sin_addr)<<":"<<strToMessName(text)<<"的消息转发至"
<<inet_ntoa(ClientVector[i].addr.sin_addr)<<"失败"<<endl;
                    ClientVector.erase(ClientVector.begin()+i);
                    i--;
                }
                else
```

```
                    cout<<tt<<"-来自"<<inet_ntoa(client-
>addr.sin_addr)<<":"<<strToMessName(text)<<"的消息转发至"
<<inet_ntoa(ClientVector[i].addr.sin_addr)<<"成功"<<endl;
            }
        }
        else{
            cout<<"接收失败"<<endl;
            break;
        }
    }
    return 0;
}
```

### 客户端发送线程负责发送消息

对线程处理函数传入COSKET类型的参数后，该线程负责发送用户输入的消息。初始化部分，请求用户输入一个小于16个字节的字符串作为昵称，并使用send()向服务端发送一个经由messageToStr函数处理的、消息文本为空的join类型的数据包，告知服务端新连接用户的相关信息；之后循环接受消息的输入并将其处理发送，若用户输入的内容为".exit"，将消息类型修改为exit，否则都为chat，发送消息，判断是否发送成功；若为退出类型，则关闭socket，清理相关资源。

```
//发送线程
DWORD WINAPI SendThread(LPVOID LpParam)
{
    //初始化线程并获取用户昵称
    SOCKET sock=(SOCKET)LpParam;
    char usrname[15];
    char SendBuf[MAXBUFSIZE];
    cout<<"请输入昵称"<<endl;
    cin.getline(usrname,20);
    if(strlen(usrname)>15)
    {
        cout<<"长度超限"<<endl;
        closesocket(sock);
        WSACleanup();
    }
    uint8_t namelen=strlen(usrname);
```

```
    char sendContent[MAXBUFSIZE-50]={};

    //向服务器发送进入聊天室的信息
    Message joinMessage;
    joinMessage.type="join";
    time_t joint=time(0);
    char jt[100]={0};
    strftime(jt,sizeof(jt),"%Y-%m-%d %H:%M:%S",localtime(&joint));
    joinMessage.usr=usrname;
    joinMessage.t=jt;
    joinMessage.tl=namelen;
    strcpy_s(SendBuf,messageToStr(joinMessage).c_str());
    send(sock,SendBuf,MAXBUFSIZE,0);

    //循环接受消息的输入并发送
    while(1)
    {
        cin.getline(sendContent,MAXBUFSIZE);
        Message usrMessage;
        time_t t=time(0);
        char tt[100]={0};
        strftime(tt,sizeof(tt),"%Y-%m-%d %H:%M:%S",localtime(&t));
        usrMessage.t=tt;
        usrMessage.text=sendContent;
        usrMessage.usr=usrname;
        usrMessage.tl=namelen;

        //判断消息类型是退出还是正常聊天
        if(usrMessage.text==".exit")
        {
            usrMessage.type="exit";
        }
        else
        {
            usrMessage.type="chat";
        }

        //发送消息
        strcpy_s(SendBuf,messageToStr(usrMessage).c_str());
```

```
            int sendStatus=send(sock,SendBuf,MAXBUFSIZE,0);
            if(sendStatus==SOCKET_ERROR)
            {
                cout<<"发送失败"<<endl;
            }
            else
            {
                cout<<"发送成功"<<endl;
            }

            //退出
            if(usrMessage.text==".exit")
            {
                closesocket(sock);
                WSACleanup();
                //exit(0);
            }
        }
}
```

### 客户端接收线程负责接收消息

因为简化设计的客户端并不需要储存其他用户的相关信息，因此只是简单地通过一个线程循环接收客户端发送的包，并将全部数据显示在窗口。

```
//接收线程
DWORD WINAPI ReceiveThread(LPVOID LpParam)
{
    SOCKET sock=(SOCKET)LpParam;
    char RecBuf[MAXBUFSIZE];
    int RecFlag;

    //循环接收服务端发送的包
    while(1)
    {
        RecFlag=recv(sock,RecBuf,MAXBUFSIZE,0);
        if(RecFlag!=SOCKET_ERROR)
        {
```

```
            string text=RecBuf;
            cout<<text;
            memset(RecBuf,'\0',sizeof(RecBuf));
        }
        else break;
    }
    return 0;
}
```

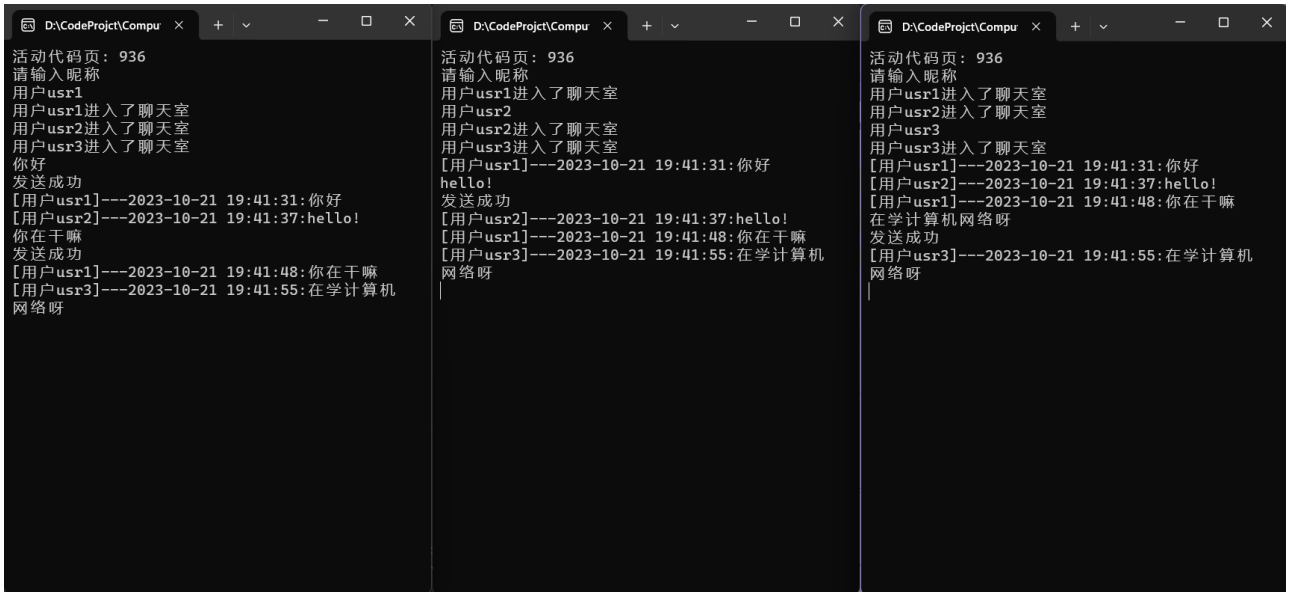
### 客户端主函数连接并开启线程

客户端记录本机的地址、端口信息后，使用connect()尝试连接服务端，成功连接后分别建立一个发送线程和一个接收线程，并将两个线程的等待时间设置为无限。

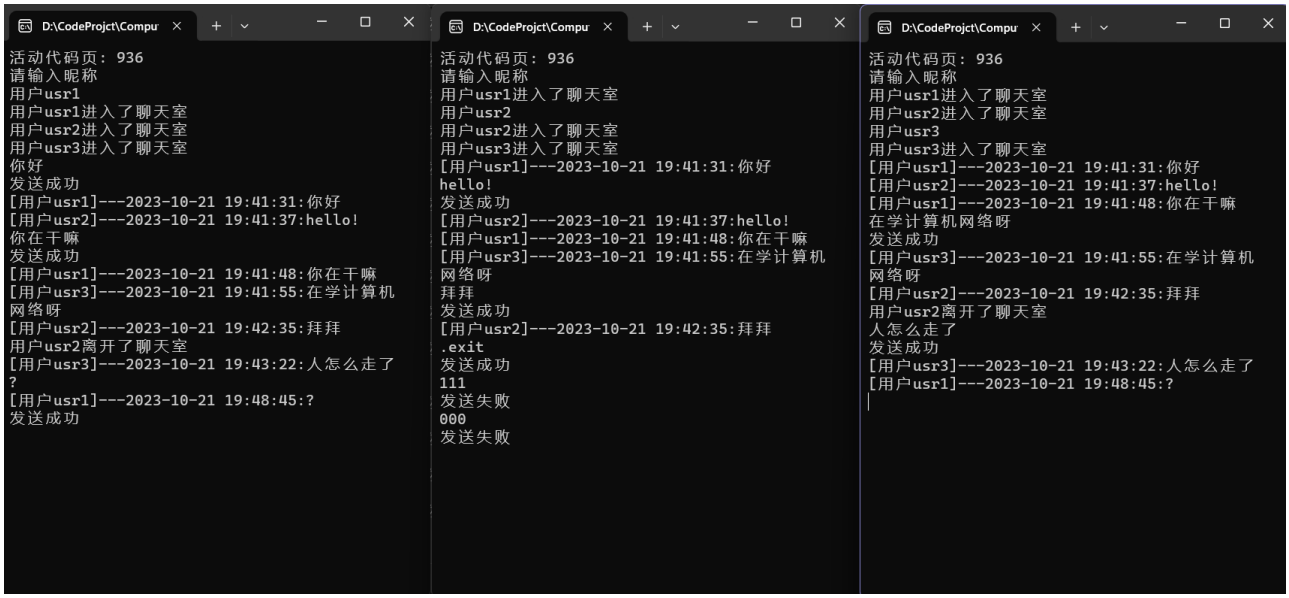
```
    SOCKET sock;
    sockaddr_in clientAddr;
    //……etc
    clientAddr.sin_family=AF_INET;
    clientAddr.sin_addr.s_addr=inet_addr(HOST);
    clientAddr.sin_port=htons(HOST_PORT);
    int len=sizeof(sockaddr_in);
    //连接服务端
    if(connect(sock,(sockaddr*)&clientAddr,len)==SOCKET_ERROR)
    {
        cout<<"sock断开连接!"<<endl;
        closesocket(sock);
        WSACleanup();
        exit(0);
        return 0;
    }
    //建立两个线程，分别用于收发
    hThreadRec=CreateThread(NULL,0,SendThread,(LPVOID)sock,0,NULL);
    hThreadSend=CreateThread(NULL,0,ReceiveThread,
(LPVOID)sock,0,NULL);
    //设置等待时间无限
    WaitForSingleObject(hThreadRec,INFINITE);
    WaitForSingleObject(hThreadSend,INFINITE);
```

## 运行结果

使用三个客户端连接服务端，昵称分别设置为用户usr1、用户usr2和用户usr3，键入回车后，会向连接至服务端的所有客户端发送用户进入聊天室的消息。键入昵称并回显进入消息后，即可开始聊天。输入消息内容，按下回车，本客户端会显示发送成功或失败的消息，所有客户端都会接收到来自服务端的消息，显示在窗口上。




当客户端输入.exit的消息并键入回车后，客户端会向服务端发送退出聊天室的数据包，并关闭自己的socket，之后便不能再向客户端发送消息；而其余客户端会受到服务器发送的该客户端退出聊天室的消息。




服务端后台的消息记录如下，转发失败和接收失败的是断开连接的线程最后一次活动，之后不再从该socket接收或向其发送消息。

```
2023-10-21 19:41:48-来自127.0.0.1:用户usr1的消息转发至127.0.0.1成功-[用户usr1]---2023-10-21 19:41:48:你在干嘛
2023-10-21 19:41:55-来自127.0.0.1:用户usr3的消息转发至127.0.0.1成功-[用户usr3]---2023-10-21 19:41:55:在学计算机网络呀
2023-10-21 19:41:55-来自127.0.0.1:用户usr3的消息转发至127.0.0.1成功-[用户usr3]---2023-10-21 19:41:55:在学计算机网络呀
2023-10-21 19:41:55-来自127.0.0.1:用户usr3的消息转发至127.0.0.1成功-[用户usr3]---2023-10-21 19:41:55:在学计算机网络呀
2023-10-21 19:42:35-来自127.0.0.1:用户usr2的消息转发至127.0.0.1成功-[用户usr2]---2023-10-21 19:42:35:拜拜
2023-10-21 19:42:35-来自127.0.0.1:用户usr2的消息转发至127.0.0.1成功-[用户usr2]---2023-10-21 19:42:35:拜拜
2023-10-21 19:42:35-来自127.0.0.1:用户usr2的消息转发至127.0.0.1成功-[用户usr2]---2023-10-21 19:42:35:拜拜
2023-10-21 19:42:40-来自127.0.0.1:用户usr2的消息转发至127.0.0.1成功-用户usr2离开了聊天室
2023-10-21 19:42:40-来自127.0.0.1:用户usr2的消息转发至127.0.0.1失败用户usr2离开了聊天室
2023-10-21 19:42:40-来自127.0.0.1:用户usr2的消息转发至127.0.0.1成功-用户usr2离开了聊天室
接收失败
2023-10-21 19:43:22-来自127.0.0.1:用户usr3的消息转发至127.0.0.1成功-[用户usr3]---2023-10-21 19:43:22:人怎么走了
2023-10-21 19:43:22-来自127.0.0.1:用户usr3的消息转发至127.0.0.1成功-[用户usr3]---2023-10-21 19:43:22:人怎么走了
2023-10-21 19:48:45-来自127.0.0.1:用户usr1的消息转发至127.0.0.1成功-[用户usr1]---2023-10-21 19:48:45:?
2023-10-21 19:48:45-来自127.0.0.1:用户usr1的消息转发至127.0.0.1成功-[用户usr1]---2023-10-21 19:48:45:?
```

## 问题及分析

- 想要单独提取可变长的昵称时，尝试遍历数组以得到\0或关键字符来截取昵称，但实现起来十分困难：

  通过在定长或可知长度的字符串后添加一个或几个字节（视具体表现数据而定）的空间用于存储后续某个字段的长度，一起添加到传递的数据包的字符串中，后取读取时视为数来读，即可准确地获取那个字段的字符串。

- sockaddr与sockaddr_in混用问题：

  两者实际占用的空间是一样的，因此可以通过强制类型转换来复用，挑选适合后续使用的结构体来方便使用，在函数明确要求另一个类型结构体的时候使用强制类型转换即可。

- 服务端日志记录时，无法记录客户端的地址信息等：

  将原本设计的socket容器改为结构体user的容器，作为句柄向服务端收发线程传入，此时线程就可以读取user中sockaddr_in的部分，获取到该socket对应的源地址进行日志记录。