实验一 socket编程【完成hub版】

刘星雨 无08 2020010850

实验目的

1. 理解socket套接字在网络模型中的位置与作用
2. 掌握socket接口的编程方式，实现两台电脑之间的聊天功能

实验内容及结果

1. 实现chat_client.py的通信功能，与助教的提供的代码进行测试

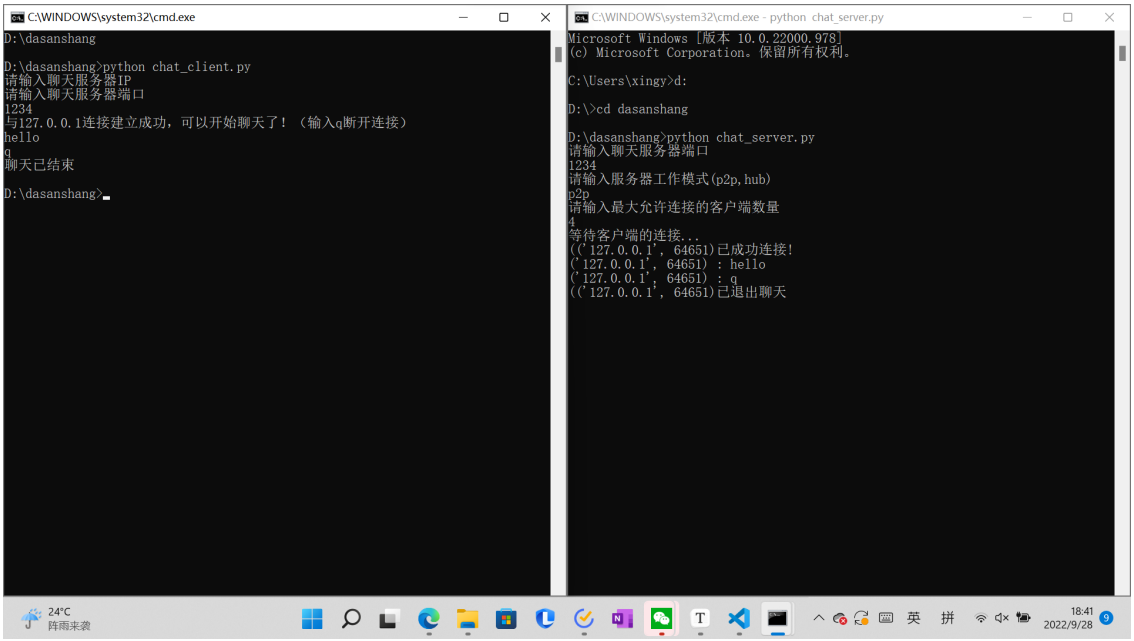
   选择echo的接口结果：

```
PS C:\Users\xingy> & C:/ProgramData/Anaconda3/python.exe d:/dasanshang/chat_client.py
请输入聊天服务器IP
请输入聊天服务器端口
10000
与101.6.65.34连接建立成功，可以开始聊天了！（输入q断开连接）
hello
收到：hello
q
聊天已结束
```

   选择hub的接口的结果，与同学同时进行测试：

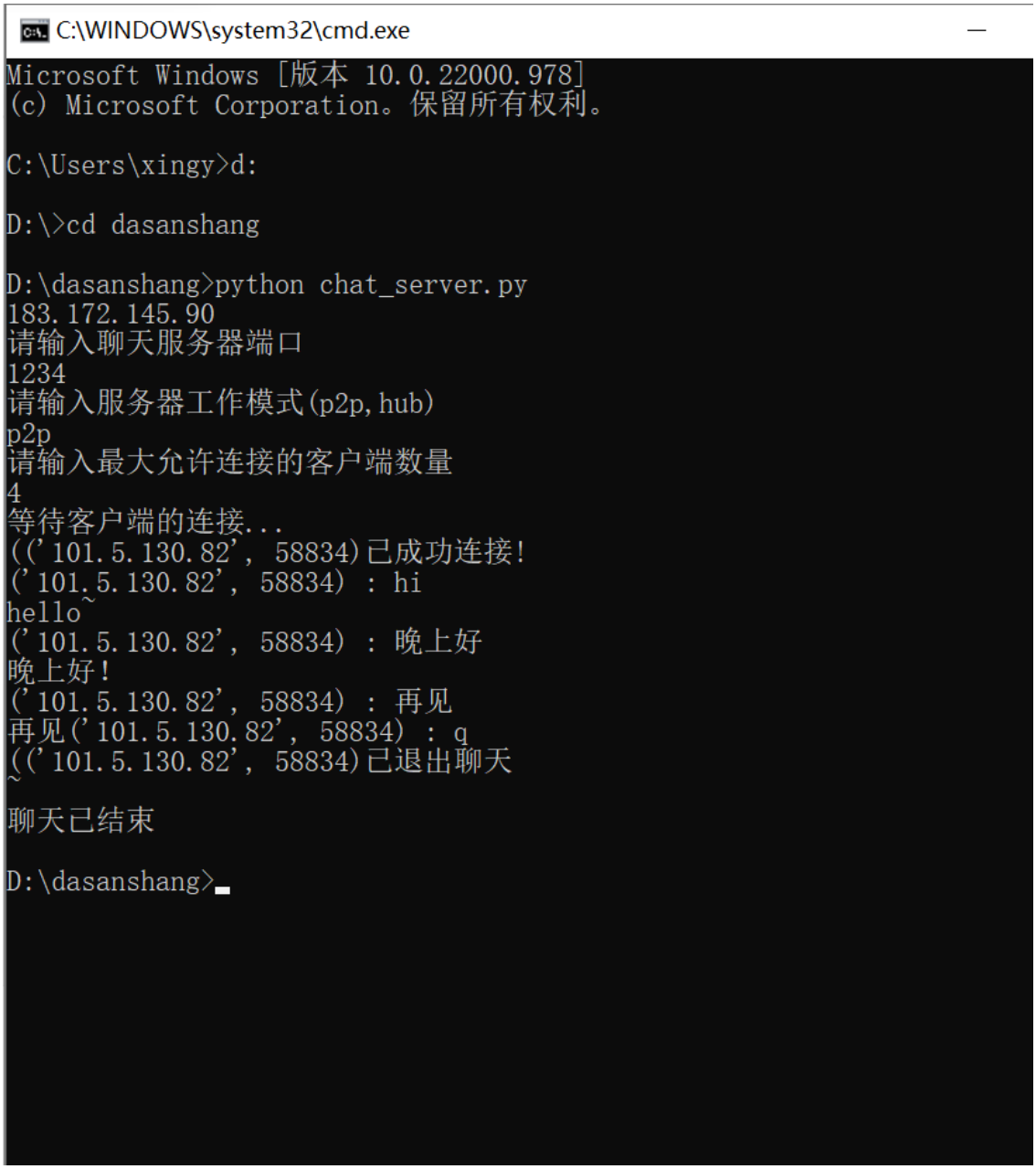
   和同学一起连接助教的hub，但是似乎没有能够看到对方的消息，按照道理不是我的client的问题。

2. 实现chat_server.py的p2p功能，与本机ip进行通信



3. p2p功能与异地ip进行通信

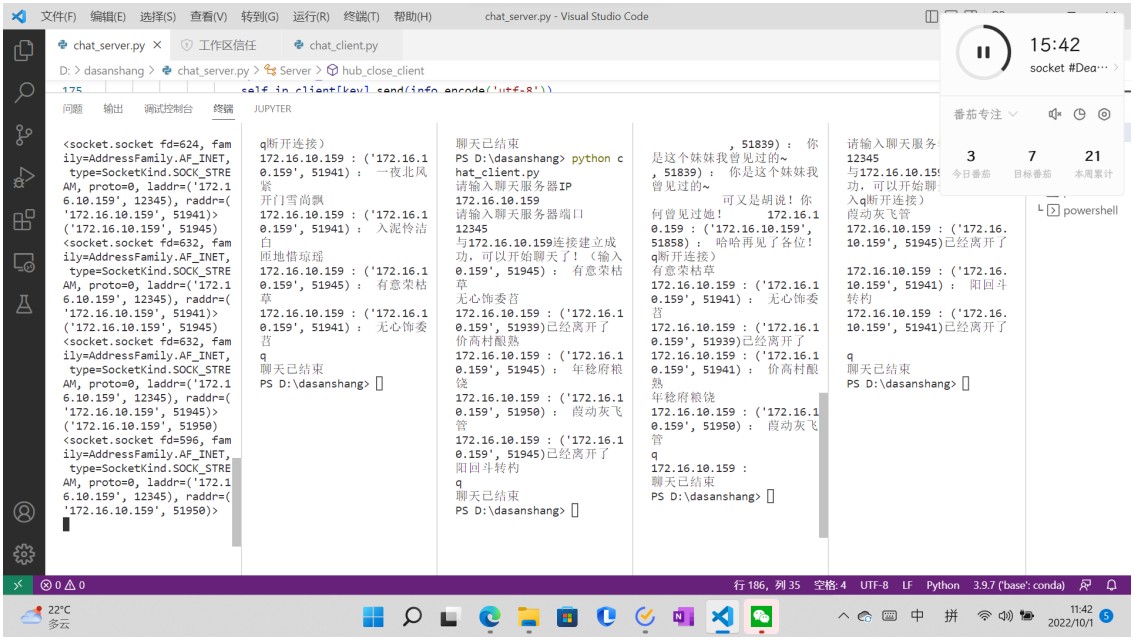
服务端：



客户端：

```
请输入聊天服务器IP
183.172.145.90
请输入聊天服务器端口
1234
与183.172.145.90连接建立成功，可以开始聊天了！（输入q断开连接）
hi
183.172.145.90 : hello~
晚上好
183.172.145.90 : 晚上好！
再见
q
聊天已结束
(nmos) PS C:\Users\18525\Documents\WeChat Files\wxid_7scsykue6rw522\FileStorage\File\2022-09>
```

4. 自己的hub的本机测试



代码及思路

chat_client.py

- 初始化生成一个socket
  - 其中需要选择网络协议【这里选择为ipv4--AF_INET】、通信类型【这里选择TCP--即SOCK_STREAM】
- 获得服务端的ip和端口号
  - 助教服务器ip为：101.6.65.34
  - 本机ip为：127.0.0.1
  - 和同学联机测试时的ip由chat_server.py中的get_ip( )函数获得
- 连接客户端和服务器
  - 利用socket类下的函数connect( )来补全start_connection（）
- 接收消息
  - 输入不为q时，一直观测键盘输入，编码发给服务器。若输入为q，关闭client的socket。
- 发送消息
  - 只要有消息就打印下消息的来源和内容，如果会报错，则打印“会话已结束”
- 利用多线程threading来实现消息同时收发

```
import socket
from threading import Thread
import time
```

```python
#echo端口：10000
#聊天室：10001

BUFFER_SIZE = 1024

class Client():
    def __init__(self):
        #----------------------------------------------
        # TODO:  初始化客户端socket
        self.domain=socket.AF_INET#ipv4
        self.type=socket.SOCK_STREAM#tcp stream sockets
        self.protocol=socket.IPPROTO_TCP
        self.client = socket.socket(self.domain,self.type,self.protocol)
        #创建socket
        #----------------------------------------------
        self.ip, self.port = self.set_ip_port()  # 通过命令行标准输入，设置服务器IP与端口

    def set_ip_port(self):
        print("请输入聊天服务器IP")
        ip = input()
        #ip = "127.0.0.1"##
        #ip="101.6.65.34"
        print("请输入聊天服务器端口")
        port = input()
        return ip,int(port)

    def start_connection(self):
        #----------------------------------------------
        # TODO:  通过socket连接至对应IP与端口
        # self.client.XXXXXX
        self.client.connect((self.ip,self.port))
        #----------------------------------------------
        print("与" + self.ip + "连接建立成功，可以开始聊天了！（输入q断开连接）")
        # 为接受消息和发送消息分别开启两个线程，实现双工聊天
        Thread(target=self.send_msg).start()
        Thread(target=self.recv_msg).start()
        #self.send_msg()

    def send_msg(self):
        #----------------------------------------------
        # TODO:  在本函数中实现Socket消息的接收，并实现输入q退出的功能
        # 提示：需要循环结构
        while True:
            data=input()
            self.client.send(bytes(data,'utf-8'))
            if data=='q':
                self.client.close()
                break
            #取消多线程的代码
          #  time.sleep(1)
           # self.recv_msg()

        #----------------------------------------------
        a = 1
```

```
    def recv_msg(self):
        #-----------------------------------------------
        # TODO:  在本函数中实现Socket消息的接收
        # 提示：需要循环结构
        # 提示：send_msg子进程退出并关闭socket时会报错，因此需要用try except结构进行异常处
        #-----------------------------------------------
        try:
            # 可能报错的语句
            while True:
                data=self.client.recv(BUFFER_SIZE)
                print(str(self.ip)+' : '+data.decode('utf-8'))
                #取消多线程之后的代码：
              #  time.sleep(1)
              #  self.send_msg()
            a = 1
        except:
            # 如果报错了，则执行下面的内容（退出循环）
            print("聊天已结束")
            self.client.close()
            b = 1

if __name__ == '__main__':
    client = Client()
    client.start_connection()
```

chat_server.py

- 生成自己的socket
- 规定自己的ip地址和端口号
  - 本机地址127.0.0.1
  - 利用hostname()和gethostbyname()获得ip本机ip地址
- 利用bind()绑定服务器socket和ip地址和端口
- 设置服务端的timeout、listen数量、运行模式
- 利用accept（）来与客户端建立连接
  - 如果是hub则要将建立的client及其对应的地址加入字典self.ip_client
- threading双线实现收发消息
  - 对于hub还说，每一个client都需要一个threading
- 接收消息
  - recv（）
  - 若接收的内容是'q',则关闭与客户端的socket
- 发送消息
  - send()
  - 若已经关闭了与客户端的socket，再发送会报错，此时输出'聊天已结束'
- 对于hub收发消息设置为同一个，因为要广播接收的内容

```
import socket
from threading import Thread

BUFFER_SIZE = 1024
```

```
class Server():
    def __init__(self):
        #-----------------------------------------------
        # TODO:  初始化服务端socket

        self.domain=socket.AF_INET
        self.type=socket.SOCK_STREAM
        self.protocol=socket.IPPROTO_TCP
        self.server = socket.socket(self.domain,self.type,self.protocol)
        #-----------------------------------------------
        #self.ip = "127.0.0.1"          # 服务器IP为local host，即本机
        self.ip=self.get_ip()
        print(self.ip)
        self.port = self.set_port()    # 通过命令行标准输入，设置服务器端口

        #-----------------------------------------------
        # TODO:  为服务socket绑定IP与端口
        # self.server.XXXXXXXX(params)
        self.addr=(self.ip,self.port)
        self.server.bind(self.addr)
        #-----------------------------------------------
        self.mode = self.get_mode() #工作方式

        #-----------------------------------------------
        # TODO:  设置服务端默认timeout时间(必须有)
        # self.server.XXXXXXXX(params)
        self.server.settimeout(6000)
        #-----------------------------------------------
        self.max_clients = self.set_max_clients()

        #-----------------------------------------------
        # TODO:  设置服务端最大连接的客户端数量(必须有)
        # self.server.XXXXXXXX(params)
        self.server.listen(self.max_clients)
        #-----------------------------------------------


        if self.mode == 'p2p':
            # 必做：实现p2p聊天
            self.start_p2p_listen()
        else:
            # 选做：实现聊天室服务器功能
            self.ip_client = {}
            self.start_hub_listen()
    def get_ip(self):
        hostname=socket.gethostname()
        ip=socket.gethostbyname(hostname)
        return ip

    def set_port(self):
        print("请输入聊天服务器端口")
        port = input()
        return int(port)

    def get_mode(self):
```

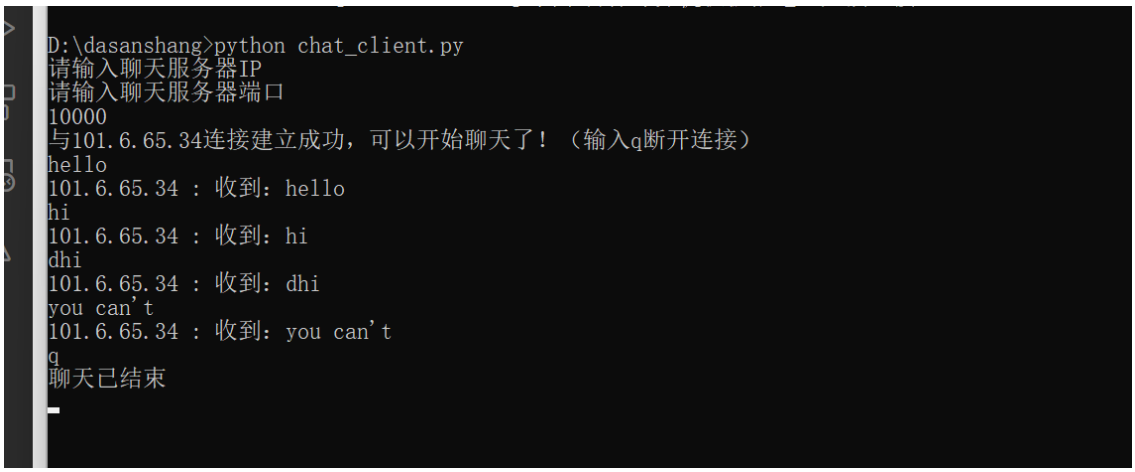
```
        print("请输入服务器工作模式(p2p,hub)")
        mode = input()
        return mode

    def set_max_clients(self):
        print("请输入最大允许连接的客户端数量")
        max_clients = input()
        return int(max_clients)


 ##############################################################################
#
    # 工作方式1：p2p连接服务器
    def start_p2p_listen(self):
        #---------------------------------------------
        # TODO:  等待建立连接(必须有)，当用户连接时打印消息，如(ip, port)已成功连接
        # 提示：需要循环结构
        # self.server.XXXXXXXX(params)
        while True:
            print('等待客户端的连接...')
            self.connection,self.address=self.server.accept()
            print('('+str(self.address)+'已成功连接!')
            break
        #---------------------------------------------
        #---------------------------------------------
        # TODO:  为接受消息和发送消息分别开启两个线程，实现双工聊天
        # 此处仅需替换param位置的参数；根据上一个位置的返回值仅需更改
        Thread(target=self.p2p_send_msg,args=(self.connection,)).start()  #args参
数是client，是客户端的socket
        Thread(target=self.p2p_recv_msg,args=(self.connection,)).start()
        #---------------------------------------------
        a = 1

    def p2p_send_msg(self,client):
        #---------------------------------------------
        # TODO:  实现发送消息功能
        # 提示1：字符串必须先encode才能发送
        # 提示2：获得标准输入参考本例程其他函数
        # 提示3：需要循环结构
        # 提示4：当recv_msg收到用户退出通知，并关闭socket后，此子进程会报错，需要通过try
except进行异常处理
        #---------------------------------------------
        try:
            # 可能报错的语句
            while True:
                data_send=input()
                client.send(bytes(data_send,'utf-8'))
            a = 1
        except:
            # 如果报错了，则执行下面的内容（退出循环）
            print("聊天已结束")
            b = 1

    def p2p_recv_msg(self,client):
        #---------------------------------------------
```

```
        # TODO:  实现接受消息功能，客户端发送q则退出，并打印退出消息，如(ip, port)已退出聊天
        # 提示1：接收到的消息必须先decode才能转换为字符串
        while True:
            self.recv_data=client.recv(BUFFER_SIZE)
            print(str(self.address)+' : '+self.recv_data.decode('utf-8'))

            if self.recv_data.decode('utf-8')=='q':
                print("("+str(self.address)+"已退出聊天")
                self.connection.close()
                self.server.close()
                break
        # 提示2：打印到标准输出参考本例程其他函数
        # 提示3：需要循环结构
        #--------------------------------------------
        c = 1



 ##############################################################################
#
    # 工作方式2：hub聊天室服务器（广播各个用户发送的信息）
    def start_hub_listen(self):
        #--------------------------------------------
        # 选做
        # TODO:  等待建立连接(必须有)，当用户连接时打印消息，如(ip, port)已成功连接
        # 提示1：需要循环结构
        # 提示2：推荐使用字典数据格式，利用self.ip_client将(ip,port)与client的键值对进行保存，方便管理多个用户
        # 提示3：在循环结构中，每个用户连接后利用此命令开启进程
Thread(target=self.hub_msg_process,args=(parm1,parm2 self.ip_client)).start()
        # 提示4：各个线程之间不会对传入参数进行拷贝，因此ip_client会由主线程动态更新
        # self.server.XXXXXXXX(params)
        while True:
            print('等待客户端的连接...')
            client,address=self.server.accept()
            #print(str(address)) -->  ('ip',port)
            print('('+str(address)+'已成功连接!')
            self.ip_client[address]=client
            Thread(target=self.hub_msg_process,args=
(client,address,self.ip_client)).start()

        #--------------------------------------------
        d = 1

    def hub_msg_process(self,current_client, current_address, ip_client):
        #--------------------------------------------
        # 选做
        # TODO:  接受当前client发送的消息，并广播给其他所有client；当某一用户发送q时，退出该用户，并将其退出消息广播至其他所有用户
        # 提示1：需要循环结构
        # 提示2：需要调用self.hub_close_client函数退出用户线程并实现上述退出消息广播至其他所有用户的功能
        # 提示3：利用ip_client字典进行广播；for key, value in ip_client.items()；广播时，不能广播到自己
        #--------------------------------------------
```

```
        for key, value in ip_client.items():
            print(key)
            print(value)
        while True:
            try:
                recv_data=current_client.recv(BUFFER_SIZE).decode('utf-8')
                info=str(current_address)+' :  '+recv_data
            except:
                self.hub_close_client(current_client,current_address)

            if recv_data=='q':
                self.hub_close_client(current_client,current_address)
                break
            for key,value in ip_client.items():
                if key!=current_address:
                    self.ip_client[key].send(info.encode('utf-8'))

    def hub_close_client(self, client, address):
        #-----------------------------------------------
        # 选做
        # TODO:  关闭该客户socket连接，将其退出消息广播至所有其他在线用户
        # 提示：从字典中删除元素:del(ip_client[key])
        client.close()
        words=str(address)+"已经离开了"
        for key,value in self.ip_client.items():
            if key!=address:
                self.ip_client[key].send(words.encode('utf-8'))
        del(self.ip_client[address])

        #-----------------------------------------------
        f = 1

if __name__ == '__main__':
    server = Server()
```

思考题

1. 实验中提供的代码框架使用多线程分别处理消息接收与消息发送，若取消多线程部分，会出现什么现象？并分析原因



取消多线程就意味着一次只能运行recv()和send（）函数中的一个，收发信息的次序就必须提前确定，例如更改之后只能收一条消息发一条消息了。thread意味着不停地运行两个函数，如果没有thread，运行了send后，程序自己是不会从send的while循环里面出来运行recv的。就算是让两个函数内部互相调用了，例如现在更改之后的代码，收发也是不能同时的。

以下是去掉thread之后实现的一发一收的代码：

```
    def start_connection(self):
        #--------------------------------------------
        # TODO:  通过socket连接至对应IP与端口
        # self.client.XXXXXX
        self.client.connect((self.ip,self.port))
        #--------------------------------------------
        print("与" + self.ip + "连接建立成功，可以开始聊天了！（输入q断开连接）")
        # 为接受消息和发送消息分别开启两个线程，实现双工聊天
        #Thread(target=self.send_msg).start()
        #Thread(target=self.recv_msg).start()
        self.send_msg()
    def send_msg(self):
        #--------------------------------------------
        # TODO:  在本函数中实现Socket消息的接收，并实现输入q退出的功能
        # 提示：需要循环结构
        while True:
            data=input()
            self.client.send(bytes(data,'utf-8'))
            if data=='q':
                self.client.close()
                break
            #取消多线程的代码
            time.sleep(1)
            self.recv_msg()

        #--------------------------------------------
        a = 1
    def recv_msg(self):
        try:
            # 可能报错的语句
            while True:
                data=self.client.recv(BUFFER_SIZE)
                print(str(self.ip)+' : '+data.decode('utf-8'))
                #取消多线程之后的代码：
                time.sleep(1)
                self.send_msg()
            a = 1
        except:
            # 如果报错了，则执行下面的内容（退出循环）
            print("聊天已结束")
            self.client.close()
            b = 1
```

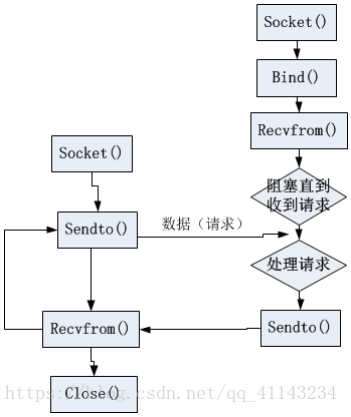
2. 除多线程外，有无其他方式实现socket双工通信?

   经查询，select模块可以利用轮询机制，重复查看是否有信息需要处理，可以实现全双工异步通信。

   [(4条消息) 无需多线程，实现全双工（异步）通信_csq225226的博客-CSDN博客]

3. 若使用基于UDP的socket，聊天软件能否正常工作？二者再使用上有何不同？

   按照普通的聊天室需求，基于UDP的socket，聊天软件也是可以正常工作的。但是，UDP协议是面向非连接的协议，没有建立连接的过程，所以它的通信效率高；也正因为如此，它的可靠性不如TCP协议高。UDP只适用于一次只传送少量数据、对可靠性要求不高的应用环境，一次只能发送的数据不能超过64KB 。



   如图，没有connect（）和listen（）这样与建立连接相关的函数