程设第十二次作业
20377383 樊思涵

题目一：利用socket和多线程，实现支持多人对话的聊天室。具体地，实现Manager和Chatter 两个类，Chatter只需和Manager之间建立一对一联系，而Manager则负责广播或转发所有用户的消息。请在实际中找个场景运用。

相关要求如下：

1. 实现Manager类, 服务器，管理成员进入和离开聊天室，接收成员消息并广播
2. 实现Chatter类, 用户, 向管理员发送加入和退出请求，发送和接收消息
3. Manager类使用多线程服务多个用户
4. Chatter用户发送和接收消息需要依赖不同线程进行
5. Manager类具备定向转发功能，比如Chatter可以在消息中通过@指定特定用户，这样Manager将仅转发给被指定用户。
6. Chatter在离开时，自动保存聊天记录到硬盘（包括时间、发信人，信息）。
7. Manager也应保存所有聊天室记录到硬盘。

实现框架

需要一个终端专门实现 Manager 类作为服务器
多个用户（终端）可以使用 Chatter 类作为用户与 Manager 对接实现聊天功能
进行实现时，传入第一个参数为 server 则作为服务器，第一个参数为 client 则为作为聊天室的成员，并读取第二个参数作为用户名。
Ip，port 均采取默认的’127.0.0.1’，8080

Manager

初始化

```
class Manager:
    def __init__(self,socket,addr):
        self.ip = addr[0]
        self.port = addr[1]
        self.socket=socket
        self.username = 'NA'                                #不会被采用的临时username
        self.id = str(self.ip)+'('+str(self.port)+')'       #每个对接的用户的主键
```

发送、接受、广播信息方法

```
def send_msg(self,msg,username):
    try:
        self.socket.send(("%s %s: %s" %(str(time.strftime("%Y-%m-%d %H:%M:%S")), username, msg)).encode("utf-8"))
        return True
    except Exception as e:
        print("send error %s" % e)
        return False

def recv_msg(self):
    try:
        data = self.socket.recv(1024).decode("utf-8")
        if data == "quit" or not data:
            return False
        return data
    except:
        return False

def broadcast(self,msg,username):
    for c in clients.values():
        c.send_msg(msg,username)
```

Manager 需被执行的主体程序

```
def connect_client(self):
    try:
        print(f"{self.id} 尝试连接")
        file = open('log.txt','a')
        file.write(f"{self.id}尝试连接\n")
        data = self.recv_msg()
        if not data:
            return
        self.username = data
        print(f"用户{self.username} {self.id}已连接")
        file.write(f"用户{self.username} {self.id}已连接\n")
        iports[self.username] = self.id
        c.socket.send("已连接".encode("utf-8"))
        while True:
            data = self.recv_msg()
            if not data:
                break
            elif data.split(' ')[0] == '@':
                try:
                    data_lis = data.split(' ')
                    username_tmp = data_lis[1]
                    data_new=''
                    for i in range(2,len(data_lis)):
                        data_new += data_lis[i]
                    clients[iports[username_tmp]].send_msg(data_new,self.username)
                except Exception as e:
                    print("send error %s" % e)
            else:
                print(f"用户{self.username} {self.id}发送了: {data}")
                file.write(f"用户{self.username} {self.id} 发送了: {data}\n")
                self.broadcast(data,self.username)
    except Exception as e:
        print("Exception: %s" % str(e))
    finally:
        print(f"{self.id} 断开连接")
        file.write(f"{self.id} 断开连接\n")
        file.close()
        self.socket.close()
        clients.pop(self.id)
```

Manager 类的多线程实现过程

```
127     elif sys.argv[1] == 'server':
128         clients = {}
129         iports = {}
130         server = socket(AF_INET, SOCK_STREAM)
131         server.setsockopt(SOL_SOCKET,SO_REUSEADDR,1)
132         server.bind((HOST,PORT))
133         server.listen(10)
134         print("服务器已开启，正在监听{}".format(server.getsockname()))
135         while True:
136             conn, addr = server.accept() #在此阻塞
137             c = Manager(conn,addr)
138             clients[c.id] = c
139             t = Thread(target=Manager.connect_client, args=(c,))
140             t.start()
141
```

Chatter

初始化

```
class Chatter:
    def __init__(self,ip,port,username):
        """
        在初始化中完成多线程的收发消息
        """
        self.file = open(username+'.txt','a')
        self.file.write('username 聊天记录\n')
        client = socket(AF_INET,SOCK_STREAM)
        try:
            client.connect((ip,port))
            self.R = True
            t1 = Thread(target=self.send_msg, args=(client,self.file))
            t2 = Thread(target=self.recv_msg, args=(client,self.file))
            client.send(username.encode("utf-8"))
            t1.start()
            t2.start()
            t1.join()
            t2.join()
        except Exception as e:
            print("client error %s" % e)
        finally:
            print("连接已被关闭")
            self.file.close()
            client.close()
```

发送、接受信息方法

```
def send_msg(self,c,file):
    time.sleep(0.5)
    while True:
        data = input('')
        c.send(data.encode("utf-8"))
        file.write(data+'\n')
        if data == "quit":
            self.R = False
            break

def recv_msg(self,c,file):
    while self.R:
        try:
            data = c.recv(1024).decode("utf-8")
            if not data:
                break
            print(data)
            file.write(data+'\n')
        except Exception as e:
            print("recv error %s" % e)
```

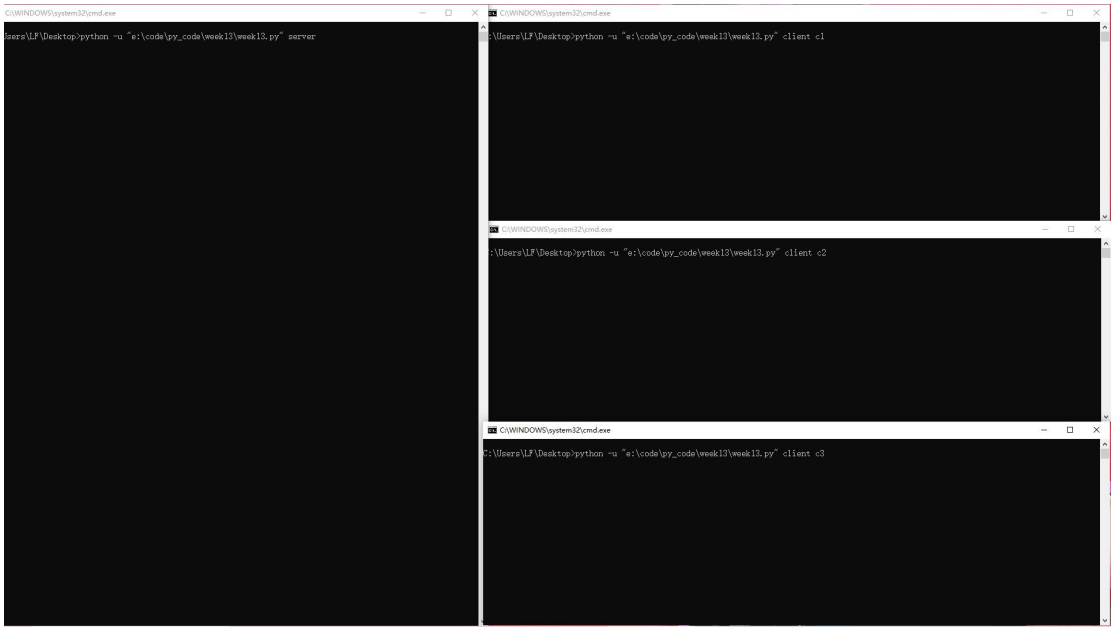
Chatter 类的在主程序中的实现方法

```
if __name__ == "__main__":
    if sys.argv[1] == 'client':
        Client = Chatter(HOST,PORT,sys.argv[2])
```

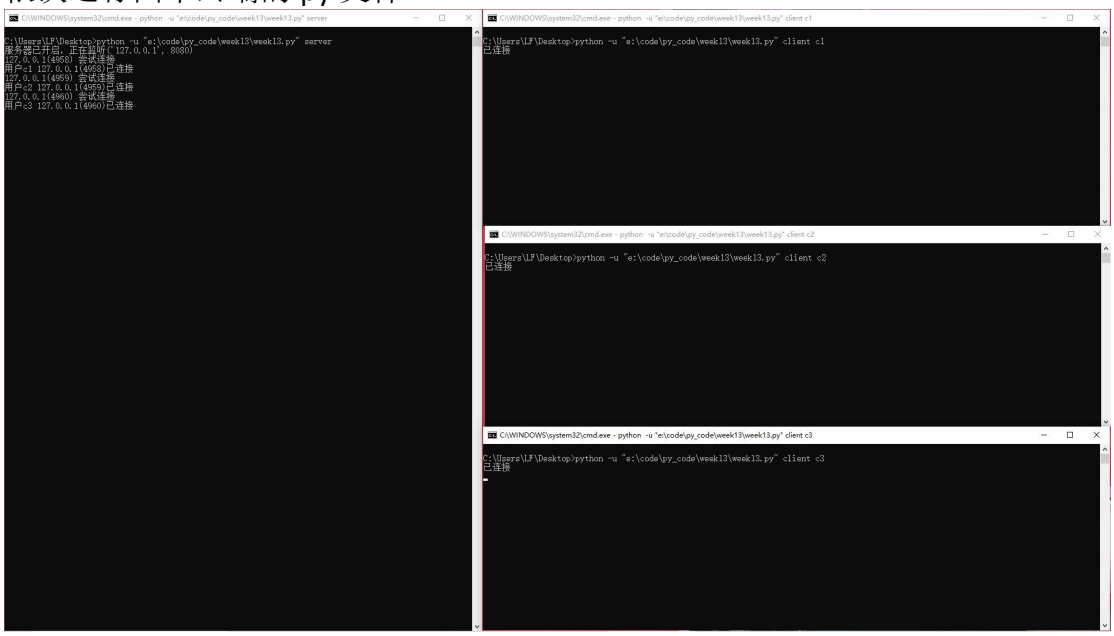
最终聊天室实现

先在本地测试

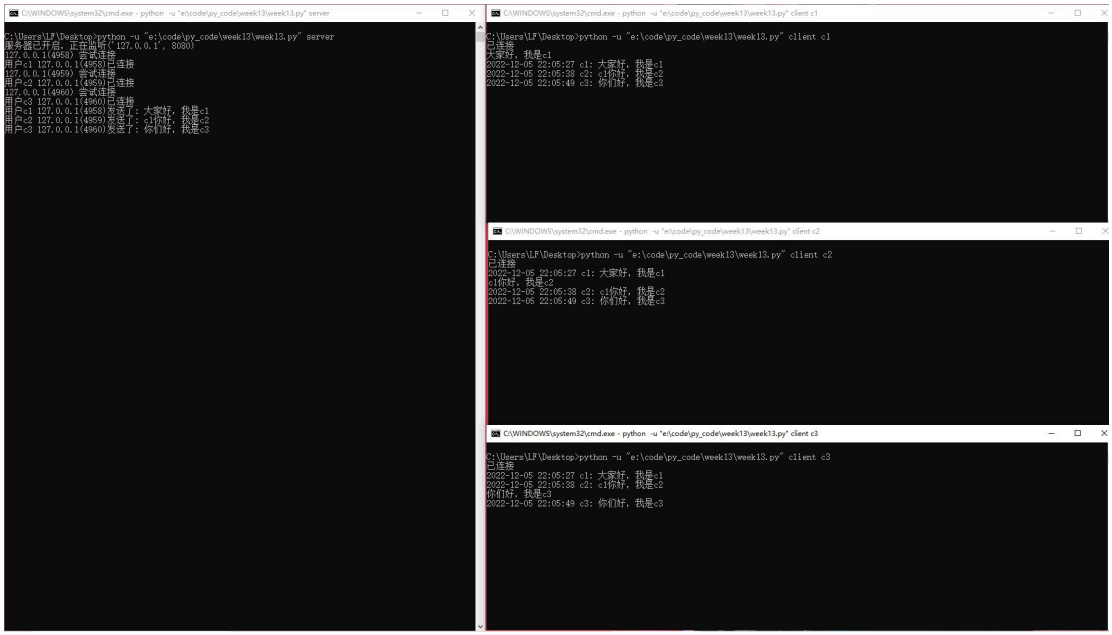
一个终端作为 server，三个终端作为 client



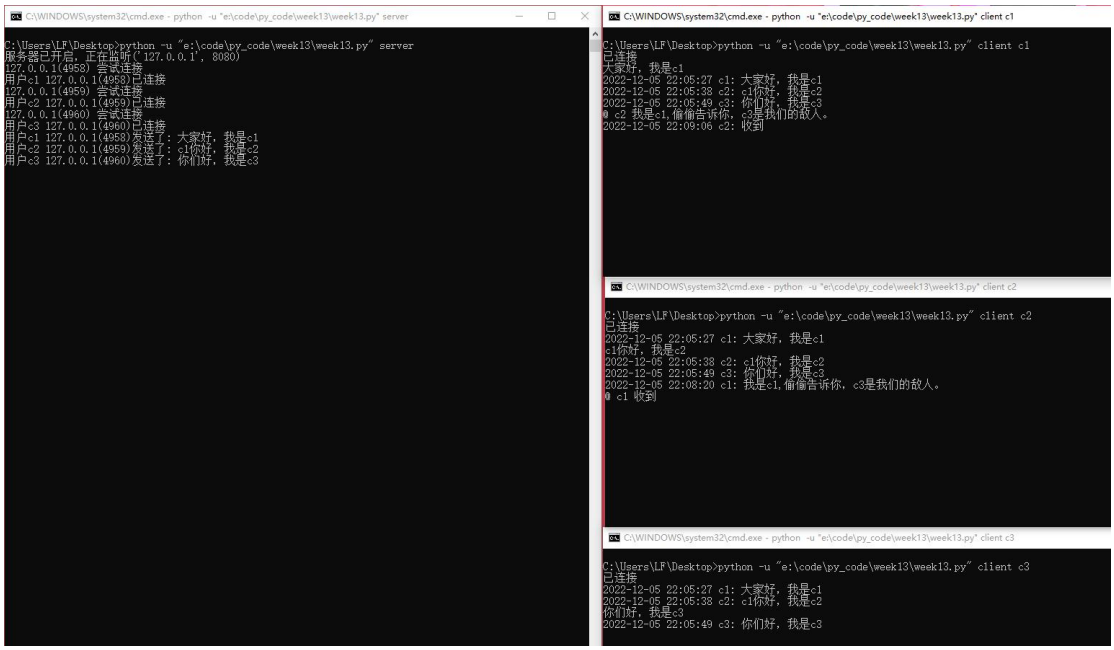
依次运行四个终端的 py 文件



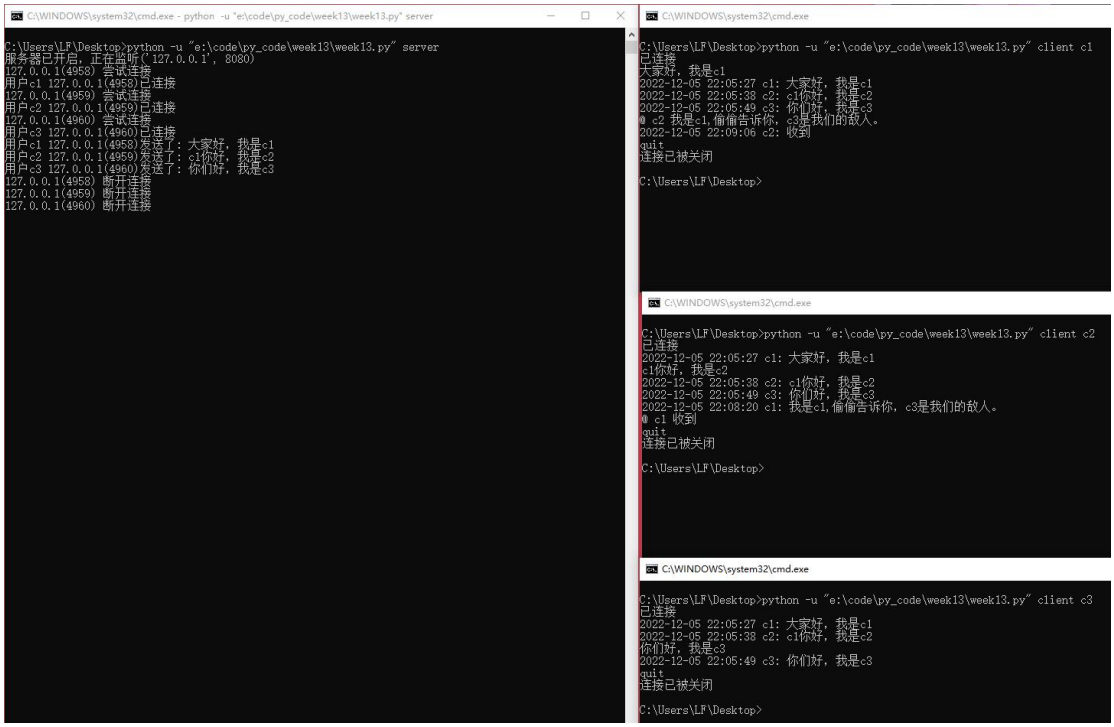
在公共区域发送消息



私聊发送消息



用户退出群聊



保存内容展示