# 通信与网络实验一：Socket 编程实验报告

无 06
闫珺博
2020010796

## 一、client 功能部分：

### 1. socket 初始化

```
#
# TODO:  初始化客户端socket
self.client =socket.socket()
#------------------------------
```

初始化 socket 对象的实例

### 2. 通过 socket 连接至对应 IP 与端口

```
# TODO:  通过socket连接至对应IP与端口
self.client.connect((self.ip, self.port))
```

self.ip 和 self.port 在调用 set_ip_port()后被设置为服务器的 ip 地址和端口号，以元组形式作为 socket 内建方法 connect()的参数

### 3. 实现 Socket 信息的发送

```
def send_msg(self):
    #--------------------------------------------
    # TODO:  在本函数中实现Socket消息的发送，并实现输入q退出的功能
    # 提示：需要循环结构
    #--------------------------------------------
    msg=""
    while msg !='q':
        msg = input()
        self.client.send(msg.encode())

    self.client.close()
```

msg 接收标准输入流中的信息，当输入信息不是 q 时，通过 socket 内建方法 send()向 server 发送 UTF-8 编码的 msg

### 4. 实现 Socket 信息的接收

```
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结构进行异常处理
    #--------------------------------------------
    try:
        # 可能报错的语句
        while 1:
            msg=self.client.recv(BUFFER_SIZE).decode()
            print(msg)
    except:
        # 如果报错了，则执行下面的内容（退出循环）
        print('连接断开')
```

由 socket 内建方法 recv()实现信息接收，解码后赋给 msg 再打印出来

二、server 功能部分：

1. server 初始化

```
class Server():
    def __init__(self):
        #-----------------------------------------------
        # TODO:  初始化服务端socket
        self.server =socket.socket()
        #-----------------------------------------------
        self.ip = "127.0.0.1"          # 服务器IP为local host，即本机
        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.bind((self.ip,self.port))
        #-----------------------------------------------
        self.mode = self.get_mode()

        #-----------------------------------------------
        # TODO:  设置服务端默认timeout时间(必须有)
        self.server.settimeout(600)
        #-----------------------------------------------
        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.listen(self.max_clients)
        #-----------------------------------------------
```

初始化 server 端 socket，绑定 ip 和 port，设置 timeout 时间以及最大连接客户端数量

2. p2p 连接服务器

(1). 等待建立连接

```
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 1:
        client_socket,client_addr=self.server.accept()
        client_ip=client_addr[0]
        client_port=client_addr[1]
        print("("+client_ip+","+str(client_port)+")成功连接")
    #-----------------------------------------------
    # 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=(client_socket,)).start()
        Thread(target=self.p2p_recv_msg,args=(client_socket,)).start()
    #-----------------------------------------------
```

由 socket 内建方法 accept()实现堵塞式等待连接并返回 socket 对象和地址(ip,port)

(2). 发送信息

```
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:
        client_ip,client_port=client.getpeername()
        # 可能报错的语句
        while 1:
            msg = input()
            msg="server:"+msg
            client.send(msg.encode())
    except:
        # 如果报错了，则执行下面的内容（退出循环）
        print("("+client_ip+","+str(client_port)+")退出连接")
```

与客户端发送消息类似，利用 send()方法

(3). 接收信息

```
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
    # 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
    #--------------------------------------------
    client_ip,client_port=client.getpeername()
    while 1:
        msg=client.recv(BUFFER_SIZE).decode()
        if msg=='q':
            print("("+client_ip+","+str(client_port)+")已退出聊天")
            break
        print("("+client_ip+","+str(client_port)+"):"+msg)
```

循环实现打印接收到的消息，如果接收到 q 则打印客户端退出消息

3. hub 聊天室服务器

(1). 等待建立连接

```
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 1:
        current_client,current_address=self.server.accept()
        client_ip=current_address[0]
        client_port=current_address[1]
        print("("+client_ip+","+str(client_port)+")成功连接")
        self.ip_client[(client_ip,client_port)]=current_client
        Thread(target=self.hub_msg_process,args=(current_client, current_address, self.ip_client)).start()
```

循环结构，每多一个 client 连接就令开启一个进程，以(ip，port)元组作为 key，对应用户的 socket 对象作为 value 构建字典

(2). 接收消息并广播

```python
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
    #-------------------------------------------
    while 1:
        msg=current_client.recv(BUFFER_SIZE).decode()
        if msg=='q':
            self.hub_close_client(current_client, current_address)
            break
        else:
            msg="("+current_address[0]+","+str(current_address[1])+"):"+msg
            for key, value in ip_client.items():
                if(value==current_client):
                    continue
                else:
                    value.send(msg.encode())
```

利用 ip_client 字典对除了当前 ip,port 的 socket 对象发送从当前客户端接收到的信息，如果接收到 q 则调用关闭客户端 socket 的方法 hub_close_client()

**(3).关闭当前客户端 socket 连接并广播退出消息**

```python
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])
    #-------------------------------------------
    msg="("+address[0]+","+str(address[1])+")已退出聊天"
    print(msg)
    del(self.ip_client[address])
    client.close()
    for key, value in self.ip_client.items():
        value.send(msg.encode())
```

从字典中删去退出的客户端并广播退出消息

## 三、client、server 正确性验证：

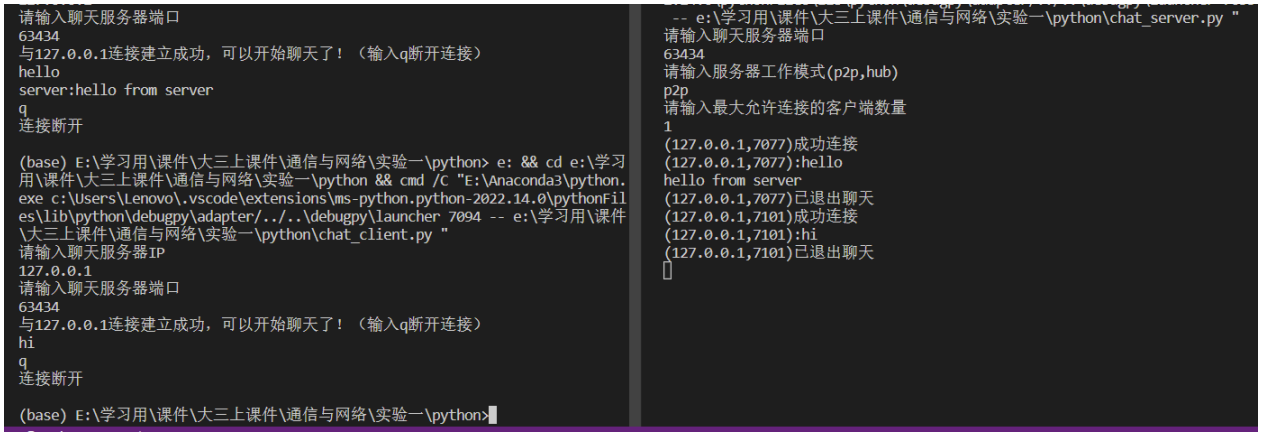
### 1.client 和 server 实现 p2p 通信

```
\学习用\课件\大三上课件\通信与网络\实验一\python\chat_client.py "
请输入聊天服务器IP
127.0.0.1
请输入聊天服务器端口
63434
与127.0.0.1连接建立成功，可以开始聊天了！（输入q断开连接）
hello
server:hello from server
q
连接断开

(base) E:\学习用\课件\大三上课件\通信与网络\实验一\python>
```

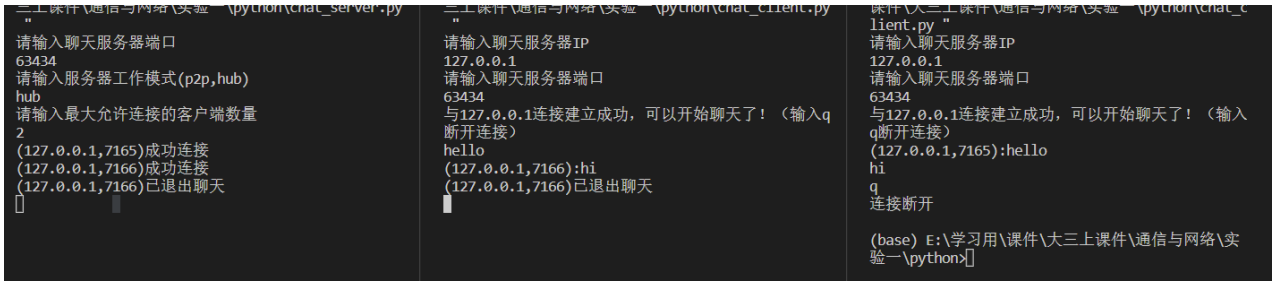
```
-- e:\学习用\课件\大三上课件\通信与网络\实验一\python\chat_server.py "
请输入聊天服务器端口
63434
请输入服务器工作模式(p2p,hub)
p2p
请输入最大允许连接的客户端数量
1
(127.0.0.1,7077)成功连接
(127.0.0.1,7077):hello
hello from server
(127.0.0.1,7077)已退出聊天
```

client 和 server 可以正常地互相收发信息




当前 client 断开连接后新的 client 也能正常连接到 server，验证了 client 和 server 的 p2p 部分的正确性

**2. 两个 client 连接到聊天室 server**




两个 client 连接到 server，可以看到广播的消息，其中一个退出也可以正常收到广播的退出消息，验证了 server 的 hub 部分的正确性

## 四、思考题

（1） 本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，若取消代码中的多线程部分，会出现什么现象？请分析现象原因。

**答：**如果去掉多线程部分，则 client 只能收/发消息，p2p 服务器只能收/发消息，聊天室服务器不能连接大于一个 client，并且也只能收/发消息。

以 client 去掉多线程部分为例：


注释掉多线程部分，先调用 send_msg(),再调用 recv_msg()

如图，client 发送消息 server 可以正常收到并打印出来，但 client 不能接受到 server 发送的消息

```
def send_msg(self):
    #--------------------------------------------
    # TODO:  在本函数中实现Socket消息的发送，并实现输入q退出的功能
    # 提示：需要循环结构
    #--------------------------------------------
    msg=""
    while msg !='q':
        msg = input()
        self.client.send(msg.encode())

    self.client.close()
```

观察 send_msg()实现，在通信过程中，while 循环一直在进行。因此如果不使用多进程，self.send_msg()一直没有执行完成，程序运行不到 self.recv_msg()，也就不能正常进行信息的接收。其他地方的多线程也类似，都是为了同时处理多个在通信建立后一直执行的循环语句。如果去掉收发方法的循环，也可以实现双向通信，但要等待对方发来信息才能发下一条信息。

（2） 除多线程外，有无其他方式实现 Socket 双工通信？

**答：**根据网络资料，select 模块可以在单线程网络服务器程序中管理多个套接字连接，进而实现双工通信。

（3） 若使用基于 UDP 的 socket，聊天软件是否能正常工作？二者在使用上有何不同？

**答：**在更改部分代码后可以正常工作。

| | 客户端 | 服务器端 |
|---|---|---|
| 基于 TCP 的 socket | 创建 socket 对象，设置服务器的 ip 地址和断开号，使用 connect()连接到服务器，用 send()和 recv()向服务器发送信息和从服务器接收信息 | 创建 socket 对象，用 bind()绑定 ip 地址和端口号到 socket 上，用 listen()开始监听。用 accept()阻塞接受客户端的连接请求,用 send()和 recv()发送信息和接收信息 |
| 基于 UDP 的 socket | 创建 socket 对象，用 bind()绑定 ip 地址和端口号到 socket 上，向服务器发送请求，并等待接收服务器回复的响应。 | 创建 socket 对象，并绑定 ip 和端口号。服务器用 recvfrom()循环等待用户请求，并在接收到请求后发送对应响应。 |

基于 UDP 的 socket 客户端不需要事先 connect 到服务器端，但需要绑定 ip 地址和端口号；服务器端不需要阻塞接受客户端的连接请求