网络聊天室 设计文档

16307110258 赵海凯 包括系统结构，应用程序协议、关键代码说明

项目&&功能简介

本聊天室软件基于UDP协议，能够应用在网络丢包率较小的环境下，能够同时支持比较大规模的用户同时在线。同时还实现了GUI界面，方便用户使用。

此外，本聊天室还引入了大厅模式，即用户登陆后会首先进入到大厅主界面，所有用户都可以在大厅进行聊天并且进行一系列操作如查看所有在线用户，查看所有已经开通的聊天室，进行私聊等。

实现功能： 1.开通新的聊天房间 2.展示所有已开通的房间 3.展示所有已经登录的用户 4.离开聊天室 5.在聊天室中查看所有在该聊天室的用户

设计协议(基于UDP)

服务端

由于基于UDP协议的客户端与服务器是无连接的，因此不可靠。所以相比于TCP来说，服务端不需要listen与accept操作。 当服务端要把信息通过套接字传到客户端的时候，因为socket只能传递二进制格式的信息，所以要把信息进行封装成 二进制格式，封装函数如下

```
def sendMessage(sock,clientAddr,messType,actionType,data):
    """
    把信息从服务器发送到客户端
    前两位是报头header 用于告诉客户端这是属于什么类型的信息
    :param sock:套接字
    :param clientAddr:(IP,PORT)
    :param messType:报头第一个bit
    :param actionType:报头第二个bit
    :param data:所传输的数据
    :return:
    """
    sendData = messType.encode('utf-8')+actionType.encode('utf-
    8')+struct.pack('<I',len(data))+data.encode('utf-8')
    sock.sendto(sendData,clientAddr
```

其中messType+actionType 共同组成了一个2bit的报头(header)，用于告知客户端该信息的类型。封装后的二进制格式信息为：2bit的报头+4bit数据长度+实际数据

客户端

基于UDP的客户端也无需connect操作。 类似的，客户端给服务器发送信息，也要把信息封装成二进制格式，封装方法与服务端的完全一样。

其中服务端发送信息的各种报头与其对应的信息如下表格所示

header	对应信息
00	通知客户端该用户登录成功
01	告知客户端输入密码错误
02	告知客户端用户不存在
03	告知客户端该用户已经在线
04	用户名已经被注册
05	向所有用户更新有新用户上线
06	告知全体用户某用户退出大厅（退出登录）
07	创建新房间
08	告知注册成功
09	向大厅发送所有在线用户名字
0A	在大厅展示所有在线房间
11	在大厅聊天框中发送用户信息
12	发送房间信息
13	发送私聊信息
14	展示所有在房间内的用户名字
15	用户退出房间
16	向房间所有活跃用户发送新用户进入房间的消息
17	强制销毁客户端某用户的房间界面
18	告知大厅所有用户某房间被删除了

关键代码说明

这里只解析部分主要代码，至于变量的声明，辅助函数等请参考代码源文件。

服务端

1.首先创建套接字，与端口8888绑定，地址采用localhost

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #UDP
host = socket.gethostbyname(socket.gethostname())
# server side 与端口 8888绑定
s.bind(('localhost', 8888))
```

2.处理服务端接收到的信息，解包

首先，客户端发来的信息都会存储在一个队列当中，只要队列不空，该函数就会依次取出客户端发来的信息，并进行解包，即把二进制数据转换为正常的数据，得到该信息的报头，长度以及具体传输的内容。只要调用这个函数，服务端会一直在一个While True函数中，会一直等待接收来自客户端的信息。 这里要考虑的主要是数据传输的时候要设置一个缓冲区，避免一次性传输大量数据导致网络拥塞。但写到这里我才猛然惊醒ＵＤＰ没有拥塞控制，即使网络出现拥塞也不会使得源主机的发送速率降低。主要的逻辑分支是是否已经接受到header，缓冲区大小是否满足接收完整数据包，以及接受了完整数据包后的操作。

具体实现如下

```
def process(socket,recvPackets):
    """
    处理服务端接收到的信息
    :param socket:
    :param recvPackets: Queue 一个队列
    :return:
    """
    all_buffer = '' #buffer zone
    single_buffer =''  #接受的单个数据包部分

    flag = False  #报头是否得到标识
    header = ''

    supposedLength = 0# 数据部分应该接收的长度
    recvLength = 0 # 当前数据报已经接受的长度

    while True:

        while not recvPackets.empty():

            tmp_data,clientAddr = recvPackets.get()

            all_addr.add(clientAddr)

            tmp_data = tmp_data.decode('utf-8')

            print("tmp_data",tmp_data)

            all_buffer = all_buffer + tmp_data

            if not tmp_data:
                break

            while True:
                if not flag: #未接收到报头
                    current_len = len(all_buffer)
                    if current_len >=6:
                        header = all_buffer[0:2]
                        print("header "+header)
                        supposedLength =struct.unpack('<I',all_buffer[2:6].encode('utf-8'))
[0]
                        flag = True
```

```
                        recvLength = 0
                        all_buffer = all_buffer[6:]
                        single_buffer = ''
                        if len(all_buffer) >= supposedLength: #缓冲区够大
                            single_buffer = all_buffer[:supposedLength]
                            print("single buffer",single_buffer)
                            all_buffer = all_buffer[supposedLength:]
                            flag = False #重置为未接收到报头

 data_process(single_buffer,header,socket,clientAddr,user_password)
                            recvLength = 0
                            single_buffer = ''
                            header = ''
                        else: #缓冲区不够大
                            break
                    else:
                        break

                else: #接收到报头

                    tmp_len = supposedLength - recvLength
                    if len(all_buffer) < tmp_len: #缓冲区不够大，不能收到完整数据包

                        single_buffer += all_buffer #把所有数据放在单个包中,准备继续接收
                        print("single buffer",single_buffer)
                        recvLength +=len(all_buffer)
                        all_buffer = '' #reset
                        break

                    else: # 已经收到完整数据包
                        single_buffer +=all_buffer[0:tmp_len]
                        print("single buffer",single_buffer)
                        all_buffer = all_buffer[tmp_len+1:]
                        data_process(single_buffer.decode('utf-
8'),header,socket,clientAddr,user_password)
                        recvLength = 0
                        single_buffer = ''
                        header = ''
                        flag = False

    socket.close()
```

3.解包之后，根据客户端发来的信息的报头，采取相应的操作，同时服务器向客户端回传信息，告知进行了操作之后的信息。 代码都有详细的注释，可以阅读下面代码以了解具体实现方式。

```
def data_process(data,header,sock,client_addr,user_password):

    if header == '01':
        #注册
        index = data.find('@@')
        user =  data[:index]
        password = data[index+2:]
```

```
        if user in user_password:

            sendMessage(sock,client_addr,'0','4','用户名已经被注册')

            return

        sendMessage(sock,client_addr,'0','B','注册成功')
        write_to_file(user,password)

    elif header =='03':
        #用户登录
        index = data.find('@@')
        user =  data[:index]
        password = data[index+2:]
        user_password = read_file('password.txt')

        if user not in user_password:
            sendMessage(sock,client_addr,'0','2',u'用户不存在')
            return
        if user_password[user]!=password:
            sendMessage(sock,client_addr,'0','1',u'密码错误')
            return
        for u in user_dict:
            if user_dict[u] == user:
                sendMessage(sock,u,'0','3',u'用户已经在线')
                return
        user_time[user] = datetime.datetime.now()
        sendMessage(sock,client_addr,'0','0',u'用户登录成功')
        for u in all_addr:
            if u != client_addr: #向所有用户告知
                sendMessage(sock,u,'0','5',user)
        user_dict[client_addr]  =  user
        time.sleep(1)

        # send all online user to the currnet user
        online_user = ""
        for u in user_dict:
            online_user = online_user+user_dict[u]+'#'
        online_user = online_user[:len(online_user)-1]
        # 向大厅发送所有在线用户的名字
        sendMessage(sock,client_addr,'0','9',online_user)
        time.sleep(1)

        online_room = ""
        for r in room_list:
            online_room = online_room + r+ '#'
        online_room = online_room[:len(online_room)-1]
        sendMessage(sock,client_addr,'0','A',online_room)

    elif header == '04':
        #进入房间 data 是房间名字

        all_room_user = room_user[data]
```

```
        for addr in all_room_user:
            #向房间所有活跃用户发送进入房间的信息
            sendMessage(sock,addr,'1','6',data+'#'+user_dict[client_addr])

        sendData = ''
        room_user[data].append(client_addr)
        sendData +=data
        sendData +='#'
        for u in room_user[data]:
            sendData = sendData + user_dict[u] + '#'
        sendData = sendData[:len(sendData)-1]
        sendMessage(sock,client_addr,'1','4',sendData)


    elif header == '05':
        #退出房间

        #data 房间名
        if client_addr in room_user[data]:
            room_user[data].remove(client_addr)

        print("用户"+user_dict[client_addr]+"退出房间"+data)

        all_room_user = room_user[data]

        for addr in all_room_user:
            #向房间所有活跃用户发送退出房间的信息
            sendMessage(sock,addr,'1','5',user_dict[client_addr])


    elif header =='06':
        # 创建新房间后向所有用户告知
        room_list.append(data)
        room_user[data] = []
        for addr in all_addr:
            sendMessage(sock,addr,'0','7',data)

    elif header =='11' :
        for addr in all_addr:
            sendMessage(sock,addr,'1','1','%s: %s\n' % ( user_dict[client_addr], data))

    elif header =='12' :
        #房间信息
        index = data.find("@@")
        room_name = data[0:index]
        room_mess = data[index+2:]
        for addr in room_user[room_name]:
            sendMessage(sock,addr,'1','2','%s#%s: %s\n' %
(room_name,user_dict[client_addr],room_mess))

    elif header == '13': #私人信息

        index = data.find("@@")
```

```
        user_name = data[0:index]
        user_mess = data[index+2:]
        for addr in all_addr:
            if user_dict[addr] == user_name:
                sendMessage(sock,addr,'1','3','%s#%s: %s\n' % (user_dict[client_addr],
user_dict[client_addr],user_mess))
                break

    elif header =='14': #管理员踢人了

        index = data.find('#')
        room_name = data[:index]
        kick_user = data[index+1:]

        for u in user_dict:
            if user_dict[u] == kick_user:
                kick_addr = u
                room_user[room_name].remove(kick_addr)
                break

        left_addr = room_user[room_name]
        for addr in left_addr:
            sendMessage(sock,addr,'1','5',kick_user)

        #destroy 被踢用户的UI界面
        sendMessage(sock,kick_addr,'1','7',room_name)

    elif header =='15':

        #用户退出登录，告知其他所有在线用户
        for addr in all_addr:
            if user_dict[addr] != data:
                sendMessage(sock,addr,'0','6',data)
```

UDP是基于无连接的，因此不需要和客户端建立连接，直接发送到客户端的地址即可。

```
def sendMessage(sock,clientAddr,messType,actionType,data):
    """
    把信息从服务器发送到客户端
    前两位是报头header 用于告诉客户端这是属于什么类型的信息
    :param sock:
    :param clientAddr:
    :param messType:
    :param actionType:
    :param data:
    :return:
    """
    sendData = messType.encode('utf-8')+actionType.encode('utf-
    8')+struct.pack('<I',len(data))+data.encode('utf-8')
    sock.sendto(sendData,clientAddr)
```

主函数

这里使用的是单线程，在后台执行recvData这个函数，会一直接收来自客户端的信息，然后开一个Queue,保存在队列当中。 然后调用receive函数，依次取出队列中的数据进行处理

```
def recvData(socket,recvPackets):
    """
    把接收到的信息都存储在recvPackets里
    :param socket:
    :param recvPackets:
    :return:
    """
    while True:
    data,addr = socket.recvfrom(1024)
    recvPackets.put((data,addr))

if __name__ == "__main__":
    print("服务端正在工作")
    user_point = read_point('../points.txt')
    thread = threading.Thread(target = recvData,args=(s,recvPackets))
    thread.start()
    receive(s,recvPackets)
```

客户端

这一部分我实现了一个客户端的类，所有函数都定义在该类内 其中建立连接的函数如下，当一个新的客户端启动的时候，会随机为客户端分配一个端口号，并且启动一个新的线程，在后台执行接收来自服务端信息的函数。

```
def connect(self):
    if self.socket!=None: #已经连接
        return
    try:
        self.socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        host = socket.gethostbyname(socket.gethostname())
        port = random.randint(6000,8887) #随机为客户端分配一个端口号
        self.socket.bind((host,port))
    except Exception:
        raise
    self.thread = threading.Thread(target=self.recvData)
    self.thread.start()
```

2.处理客户端接收到的信息，解包。 这部分与服务端的函数功能一样，这里不再赘述。

3.解包之后客户端将根据不同的信息在GUI界面上进行相应的操作，具体看代码实现，有详细的注释

```
def dataProcess(self, data, header):
        """

        :param data:
```

```
        :param header:
        :return:
        """

        """
        下面是header的设计

        """

        print("data:" + data)

        if header == '00':
            # login successful
            self.client.stop_flag = '0'
        elif header == '01':
            # password wrong
            self.client.stop_flag = '1'
        elif header == '02':
            # user not exists
            self.client.stop_flag = '2'
        elif header == '03':
            # user already online
            self.client.stop_flag = '3'

        elif header == '04':
            # username already exists
            self.client.stop_flag = '4'

        elif header == '05':
            # 用户登录
            self.client.main_app.all_player.insert(Tkinter.END, data)
            print("用户登录" + data)

        elif header == '06':
            # 用户退出大厅
            print("用户退出" + data)
            for i in range(self.client.main_app.all_player.size()):  # 删除退出用户
                if self.client.main_app.all_player.get(i) == data:
                    print(self.client.main_app.all_player.get(i))
                    self.client.main_app.all_player.delete(i)
                    break

        elif header == '07':
            # 创建新房间
            print("创建新房间" + data)
            self.client.main_app.allroom.insert(tkinter.END, data)

        elif header == '0B':
            # register successful
            self.client.stop_flag = 'B'

        elif header == '09':
            # 在大厅里展示所有的在线用户
```

```
            alluser = data.split('#')
            for u in alluser:
                if u != '':
                    self.client.main_app.all_player.insert(tkinter.END, u)

        elif header == '0A':
            # display all room
            allroom = data.split('#')
            for r in allroom:
                if r != '':
                    print(r)
                    self.client.main_app.allroom.insert(tkinter.END, r)

        elif header == '11':
            # 大厅信息
            print("大厅信息" + data)
            self.client.main_app.datingtext.insert(tkinter.END, data)

        elif header == '12':
            # 房间信息
            roomIndex = data.find('#')
            roomName = data[0:roomIndex]
            message = data[roomIndex + 1:]
            if roomName in self.client.room_app:
                # 把信息插入到房间信息界面上
                self.client.room_app[roomName].main_text.insert(tkinter.END, message)


        elif header == '13':
            # 私聊
            userIndex = data.find('#')
            userName = data[0:userIndex]
            message = data[userIndex + 1:]

            # 如果已经正在私聊
            if userName in self.client.person_app:
                self.client.person_app[userName].main_text.insert(tkinter.END, message)
            # 否则创建新窗口
            else:
                self.client.main_app.new_person = userName
                self.client.main_app.after_idle(self.client.main_app.create_new_person)
                time.sleep(1)
                self.client.person_app[userName].main_text.insert(tkinter.END, message)

        elif header == '14':
            # 这里的data是房间内所有用户的名称
            roomIndex = data.find('#')
            roomName = data[0:roomIndex]
            alluserdata = data[roomIndex + 1:]
            alluser = alluserdata.split('#')
            for u in alluser:
                if u != '':
                    self.client.room_app[roomName].all_room_user.insert(tkinter.END, u)
```

```
        elif header == '15':
            # 用户退出房间
            # data 是 用户名
            for i in range(self.client.room_user.size()):  # 删除退出用户
                if self.client.room_user.get(i) == data:
                    self.client.room_user.delete(i)
                    break

        elif header == '16':

            index = data.find('#')
            roomName = data[0:index]
            newUser = data[index + 1:]
            self.client.room_app[roomName].all_room_user.insert(tkinter.END, newUser)


        elif header == '17':
            self.client.room_pointer.master.destroy()
```

使用说明：

运行步骤 1.先打开sever文件夹，在命令行输入：

```
python server.py
```

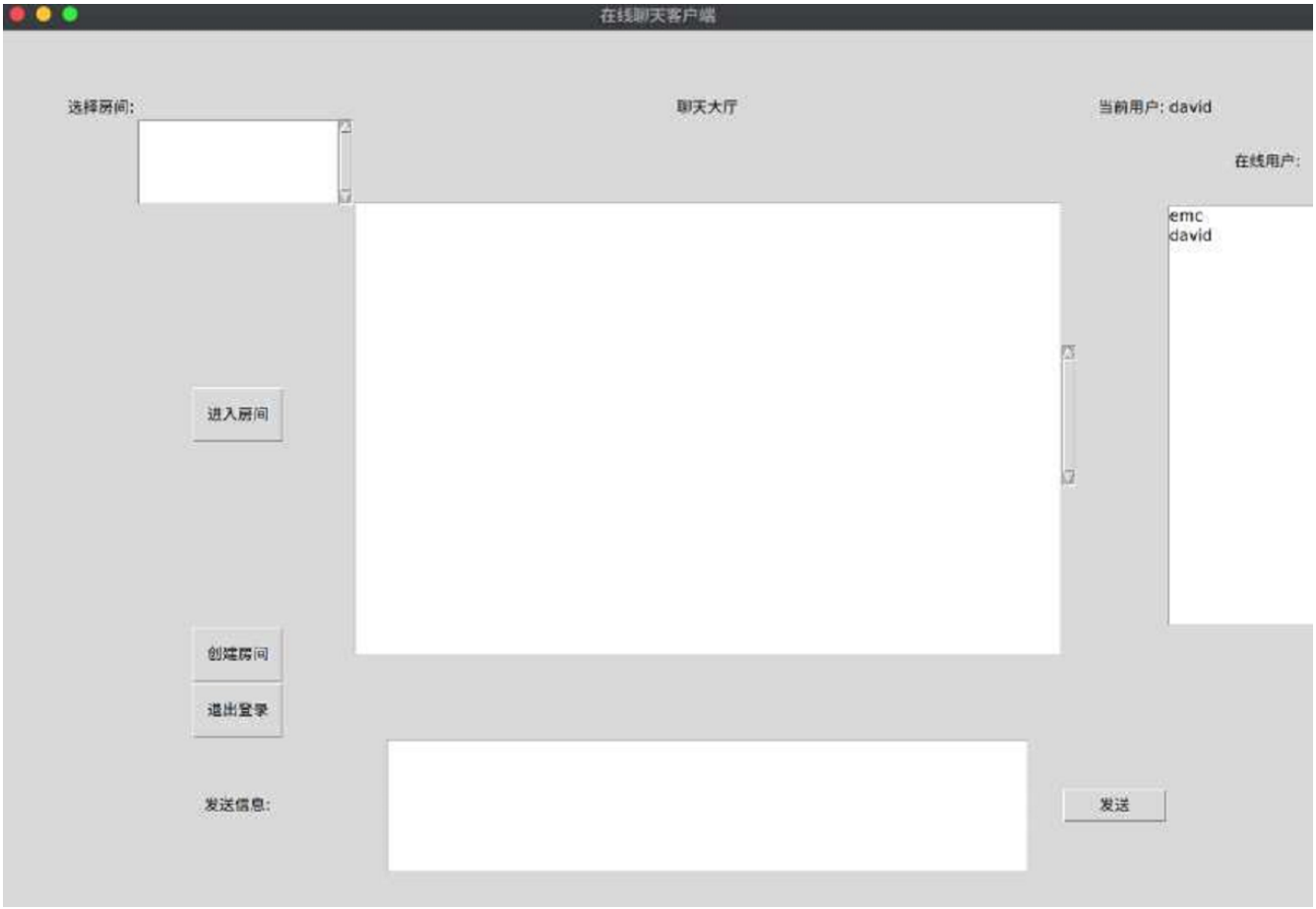
2.再打开client文件夹，新开一个终端，命令行输入

```
python client.py
```

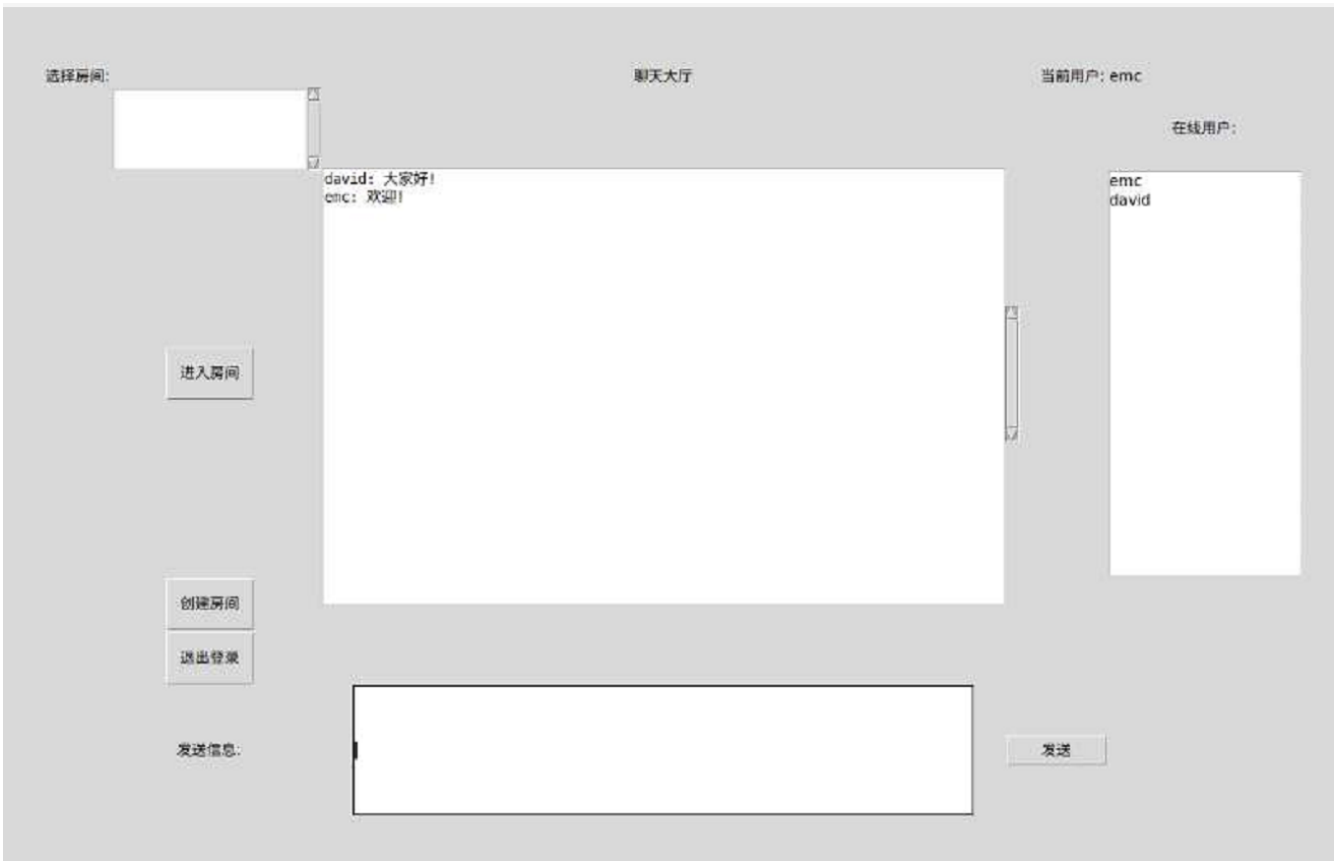
就能打开一个新的终端

注册账号界面:

开了两个client终端


在线聊天客户端
选择房间:
聊天大厅
当前用户: david
在线用户:
emc
david
进入房间
创建房间
退出登录
发送信息:
发送

大厅发言功能:


选择房间:
聊天大厅
当前用户: emc
在线用户:
david: 大家好!
emc: 欢迎!
emc
david
进入房间
创建房间
退出登录
发送信息:
发送

房间聊天室功能：

私聊功能：