

# 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。

本聊天室中，服务方是作为管理员这一身份实现的，要想进行服务方的操作，请先使用

账号：admin

密码: admin

登录界面。

此外，本聊天室还引入了大厅模式，即用户（包括服务方的管理员）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。

### 服务方实现功能：

- 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.踢走某在聊天室的用户
- 5.离开聊天室
- 6.关闭聊天室，向所有在大厅的用户更新该聊天室被删除，并且强制关闭所有在聊天室的用户的聊天窗口
- 7.在聊天室中查看所有在该聊天室的用户
- 8.发言积分(附加功能)
- 9.大厅聊天(附加功能)

### 客户方实现功能：

- 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

6.发言积分(附加功能)

7.大厅聊天(附加功能)

具体功能介绍请查看用户文档

## 项目文件架构

. ├── client  
| ├── client.py 客户端代码 | ├── gui.py UI界面 | ├── points.txt 用于保存所有用户发言积分 |── server |──  
password.txt 用于保存所有用户密码与账户名 |── server.py 服务端代码

## 设计协议(基于UDP)

服务端：

由于基于UDP协议的客户端与服务端是无连接的，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

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

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。

```
def sendData(self,data_type,action_type,data):
    """
    data_type:报头第一个bit
    action_type:报头第二个bit
    data:所传输的数据
    """
    sendMessage = data_type.encode('utf-8')+action_type.encode('utf-8')+struct.pack('<I',len(data))+data.encode('utf-8')
    #发给服务器
    self.socket.sendto(sendMessage,self.ADDR) #self.ADDR = (self.IP,self.port)
```

其中客户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	请求注册
01	请求登录
02	请求进入某房间
03	请求退出房间
04	创建新房间
05	发送大厅消息
06	发送房间消息
07	发送私聊消息
08	管理员踢走用户 <sup>[1]</sup>
09	用户退出登录
10	删除房间 <sup>[2]</sup>

注：header 为08和10的对应功能是服务端才有的，之所以在客户端实现，是因为本项目中服务端是作为一个账户名为admin的特殊用户来体现的。即admin将拥有上帝视角，具有服务方的一切功能，充当服务方。

## 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 receive(socket, recvPackets):

    """
    处理服务端接收到的信息
    :param socket: 服务端套接字
    :param recvPackets: Queue 一个队列，存储客户端发来的信息
    :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') #把socket传来的二进制文件解码

            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:
            #获得header
            header = all_buffer[0:2]
            #获得数据长度
            supposedLength =struct.unpack('<I',all_buffer[2:6].encode('utf-8'))
[0] #已经接收到header
            flag = True
            recvLength = 0
            #所要传输的数据
            all_buffer = all_buffer[6:]
            single_buffer = ''

            if len(all_buffer) >= supposedLength: #缓冲区够大
                single_buffer = all_buffer[:supposedLength]
                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,user_point)
            recvLength = len(single_buffer)
            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
            recvLength +=len(all_buffer)
            all_buffer = ''
            break

        else: # 已经收到完整数据包
            single_buffer +=all_buffer[0:tmp_len]
            all_buffer = all_buffer[tmp_len+1:]
            data_process(single_buffer.decode('utf-
8'),header,socket,clientAddr,user_password,user_point)
            #reset
            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 dataProcess(data, header, sock, client_addr, user_password, user_point):
    """
    :param data:
    :param header:
    :param sock:
    :param addr:
    :return:
    """

    if header == '00':
        #注册
        index = data.find('@@')
        user = data[:index]
        password = data[index+2:]

        if user in user_password:
            #告知客户端该用户名已经被注册
            sendMessage(sock, client_addr, '0', '4', '用户名已经被注册')
            return
        #存储新注册用户的密码
        user_password[user] = password
        #告知客户端注册成功
        sendMessage(sock, client_addr, '0', '8', '注册成功')
        write_password(user, password)
        user_point[user] = 0
        #write_point(user, 0)

    elif header == '01':
        #用户登录
        index = data.find('@@')
        user = data[:index]
        password = data[index+2:]
        user_password = read_password('password.txt')
        if user not in user_password:
            #告知客户端用户不存在
            sendMessage(sock, client_addr, '0', '2', u'用户不存在')
            return

        if user_password[user] != password:
            #告知客户端密码错误
            sendMessage(sock, client_addr, '0', '1', u'密码错误')
            return

        for u in user_dict:
            #遍历搜索要登录的用户，看是否已经在线
            if user_dict[u] == user:
                sendMessage(sock, client_addr, '0', '3', u'用户已经在线')
```

```

        return

sendMessage(sock,client_addr,'0','0',u'用户登录成功')
for u in all_addr:
    if u != client_addr: #向所有用户告知新用户登录,以便在大厅界面更新所有在线用户
        sendMessage(sock,u,'0','5',user)
user_dict[client_addr] = user
#time.sleep(1)

online_user = ""
for u in user_dict:
    online_user = online_user+user_dict[u]+'#'
online_user = online_user[:len(online_user)-1]
# 向新登录用户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]
#向新登录用户发送所有已经创建的房间
sendMessage(sock,client_addr,'0','A',online_room)

elif header == '02':
    #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更新该用户进入房间的信息
        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]
    #给该用户更新 所有在房间内的在线用户
    sendMessage(sock,client_addr,'1','4',sendData)

elif header == '03':
    #退出房间
    #data 房间名
    if client_addr in room_user[data]:
        room_user[data].remove(client_addr)
    all_room_user = room_user[data]

    for addr in all_room_user:
        #向房间所有活跃用户发送该用户退出房间的信息,实现房间在线用户实时更新
        sendMessage(sock,addr,'1','5',user_dict[client_addr])

```



```

elif header == '04':
    # 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 == '05':
    #发送大厅信息 并更新发言积分

    user_point[user_dict[client_addr]]+=5
    write_point(user_dict[client_addr],user_point[user_dict[client_addr]])
    for addr in all_addr:
        sendMessage(sock, addr, '1', '1', '%s: %s\n' % ( user_dict[client_addr], data))

elif header == '06':
    #发送房间信息 并更新发言积分
    user_point[user_dict[client_addr]]+=3
    write_point(user_dict[client_addr],user_point[user_dict[client_addr]])
    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 == '07':
    #发送私人信息 并更新发言积分
    user_point[user_dict[client_addr]]+=1
    write_point(user_dict[client_addr],user_point[user_dict[client_addr]])
    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 == '08':
    #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]

```

```

#向所有在房间的用户更新该用户被踢出的信息，以更新当前房间的在线用户
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 == '09':
    #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:
            del user_dict[addr]
            all_addr.remove(addr)
            break

    for addr in all_addr:
        if user_dict[addr] != data:
            sendMessage(sock, addr, '0', '6', data)

elif header == '10':
    #删除房间
    room_name = data
    room_list.remove(room_name)
    for addr in user_dict:
        #向大厅所有用户发送聊天室被关闭的信息
        sendMessage(sock, addr, '1', '8', room_name)
    for addr in room_user[room_name]:
        #销毁所有在聊天室的用户的窗口界面
        sendMessage(sock, addr, '1', '7', room_name)
    del room_user[room_name]

```

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，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: 服务端回传的信息
    :param header: 信息的header
    :return:
    """

    if header == '00':
        #服务端回传'00'即登陆成功，GUI会弹出登录成功的提示
        self.client.stop_flag = '0'
    elif header == '01':
        #密码输入错误
        self.client.stop_flag = '1'
    elif header == '02':
        #用户不存在
        self.client.stop_flag = '2'
    elif header == '03':
        #用户已经在线
        self.client.stop_flag = '3'
    elif header == '04':
        #注册时用户名已经存在
        self.client.stop_flag = '4'
    elif header == '05':
        # 用户登录，大厅界面的在线用户会插入新登陆的用户
        self.client.main_app.all_player.insert(Tkinter.END, data)

    elif header == '06':
        # 用户退出大厅
        for i in range(self.client.main_app.all_player.size()):
            # 在大厅在线用户栏中删除退出用户
            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 ，在大厅的所有房间列表中更新新创建的房间
        self.client.main_app.allroom.insert(tkinter.END, data)

    elif header == '08':
        #注册成功
        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':
    #在大厅中列出所有房间
    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，更新用户的发言积分并且在聊天框中插入信息
    self.client.main_app.points.delete(0)
    self.client.main_app.points.insert(tkinter.END,read_point('../points.txt')
[self.client.username])
    self.client.main_app.datingtext.insert(tkinter.END,data)

elif header == '12':
    #房间信息，更新用户的发言积分并且在聊天框中插入信息
    self.client.main_app.points.delete(0)
    self.client.main_app.points.insert(tkinter.END,read_point('../points.txt')
[self.client.username])
    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':
    userIndex = data.find('#')
    userName = data[0:userIndex]
    message = data[userIndex+1:]
    #如果已经正在私聊，直接在私聊的聊天框中插入信息
    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)

    self.client.main_app.points.delete(0)
    self.client.main_app.points.insert(tkinter.END,read_point('../points.txt')
[self.client.username])

elif header == '14':
    #这里的data是房间内所有用户的名称
    """
    列出房间所有在线用户
    """
    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':
    #显示房间用户
    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':
    #销毁房间界面
    self.client.room_pointer.master.destroy()

elif header == '18':
    # 给所有在大厅的用户告知房间被删除了
    for i in range(self.client.main_app.allroom.size()):
        if self.client.main_app.allroom.get(i) == data:
            self.client.main_app.allroom.delete(i)
            break

```

其余辅助类的实现请参考代码。

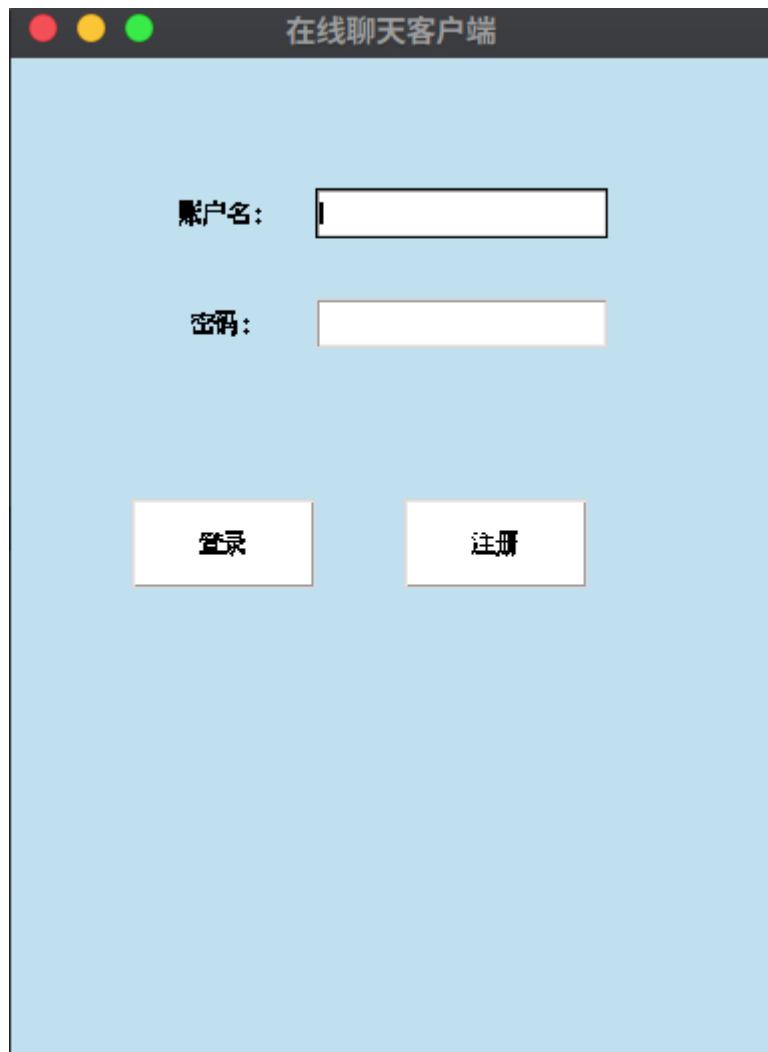
## GUI界面

gui.py

基于Python内置的第三方库TKinter来实现GUI界面。

由于GUI代码比较冗长，详细注释请查看源码。每一个分界面分别由一个Python类实现，此处仅介绍实现的主要类及其功能：

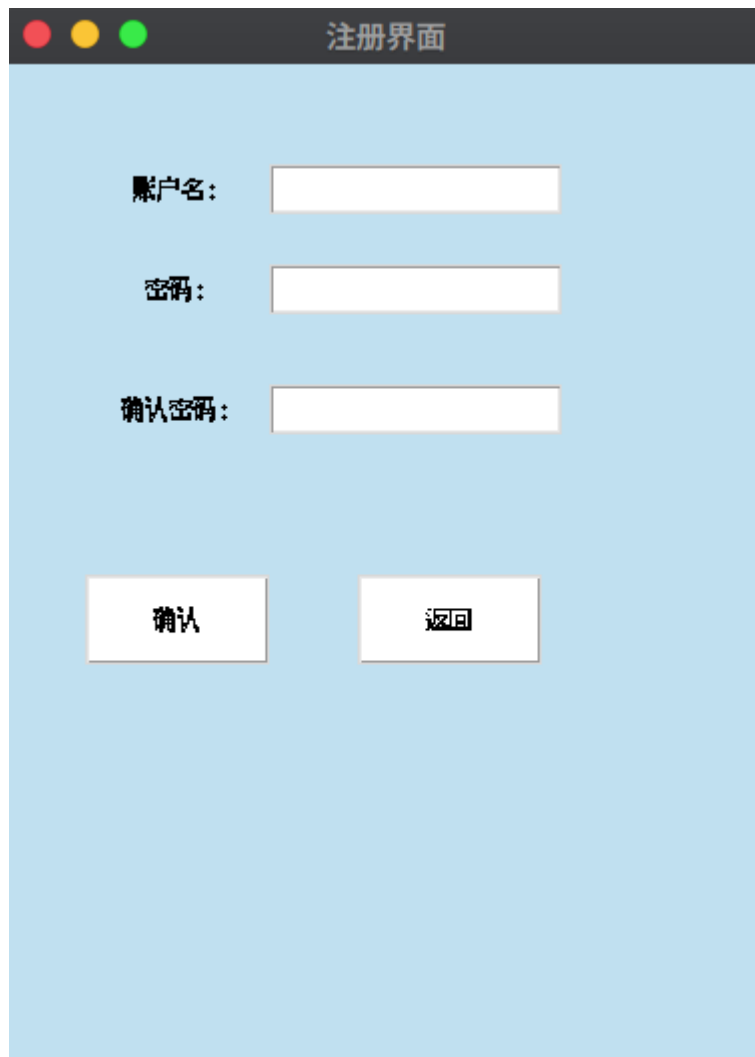
登录主界面



```
class LoginApp(Tkinter.Frame):

    def __init__(self, master=None, clientp = None):
        Tkinter.Frame.__init__(self, master)
        self.configure(bg='#c0e0f0')
        self.master.configure(bg='#c0e0f0')
        self.grid(row = 0, column = 0, padx = 60, pady = 60)
        self.master.title('在线聊天客户端')
        self.master.geometry('380x500+300+150')
        self.client = clientp    # 客户端主类指针
        self.createWidgets()
```

注册主界面



注册界面

用户名:

密码:

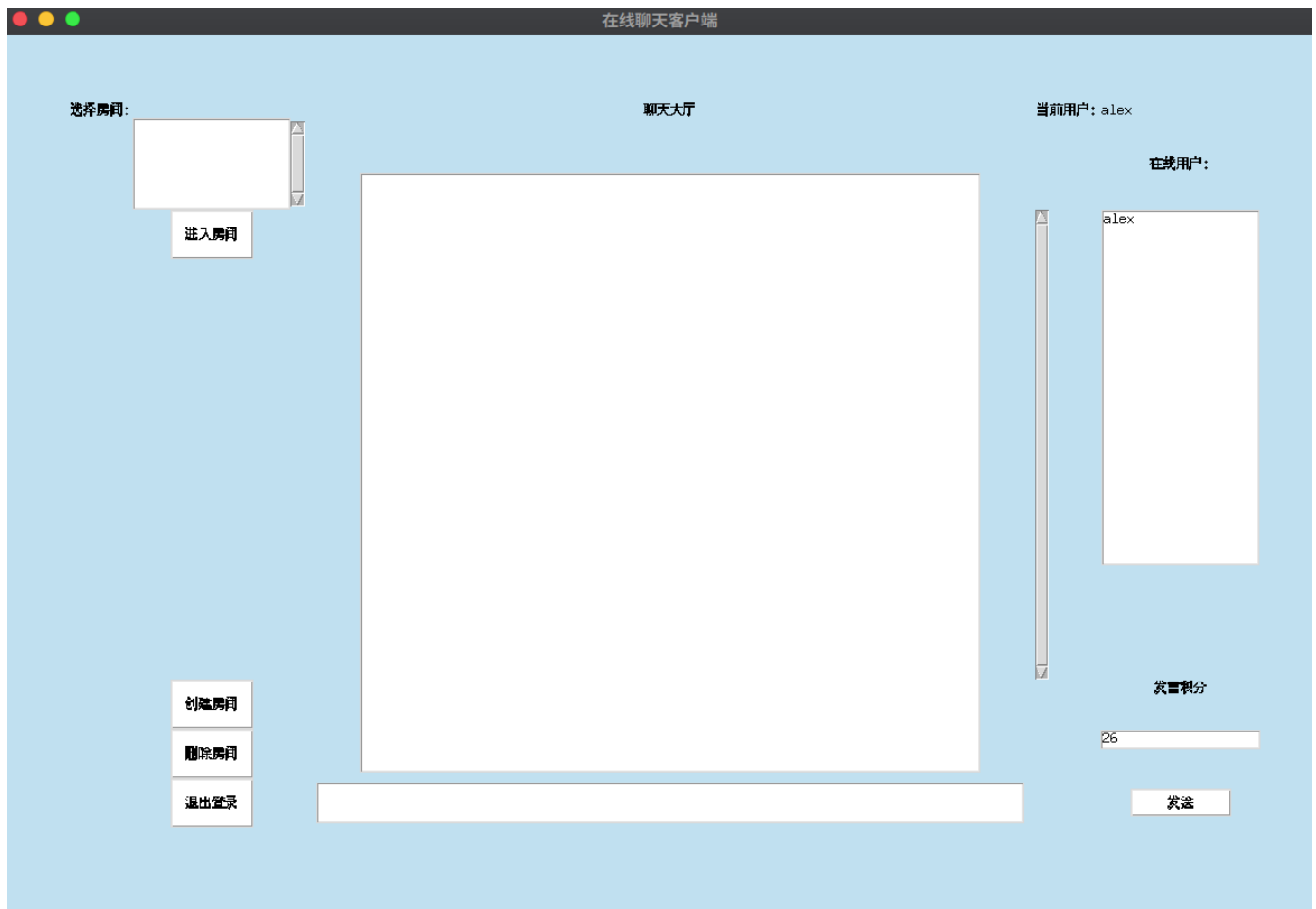
确认密码:

确认 返回

```
class Register_application(Tkinter.Frame):  
    """  
    注册主界面  
    """  
    def __init__(self, master=None, clientp = None):  
        Tkinter.Frame.__init__(self, master,bg='#c0e0f0')  
        self.grid(row = 0,column = 0,padx = 40,pady = 40)  
        self.client = clientp          # 获得客户端主类指针  
        self.master.title('注册界面')  
        self.createWidgets()
```

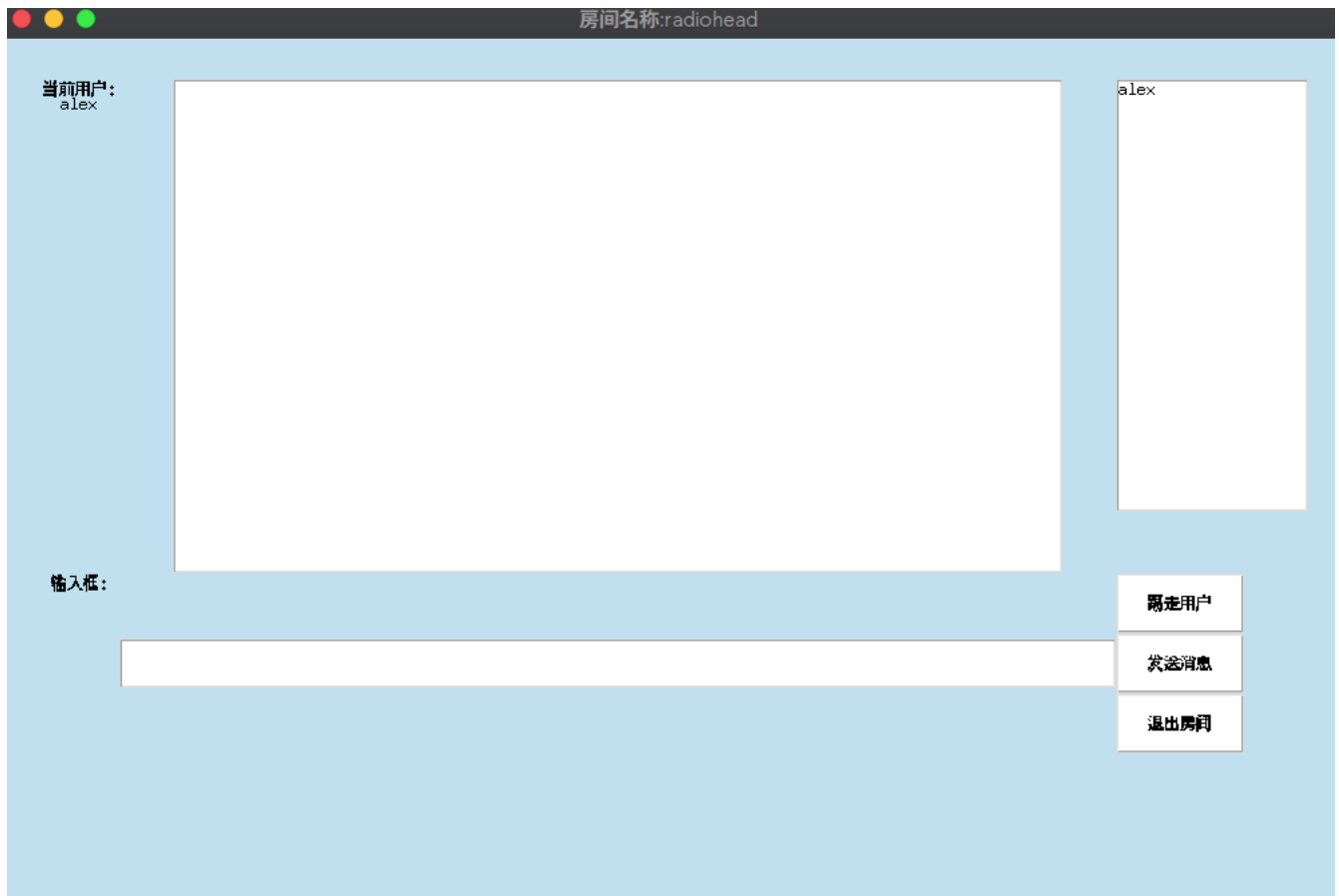
大厅主界面





```
class Main_Room_application(Tkinter.Frame):  
    """  
  
    大厅主界面  
  
    """  
    def __init__(self, master=None, clientp=None, username = None):  
        Tkinter.Frame.__init__(self, master, bg='#c0e0f0')  
        self.grid(row = 0, column = 0, padx = 60, pady = 60)  
        self.master.protocol("WM_DELETE_WINDOW", self.back_login)  
        self.client = clientp  
        self.client.main_app = self # 此时已经打开主界面，在主类中记录  
        self.client.username = username  
        self.username = username # 当前用户名  
        self.new_person = ''  
        self.createwidgets()  
        self.client.user_point = read_point('../points.txt')[username]
```

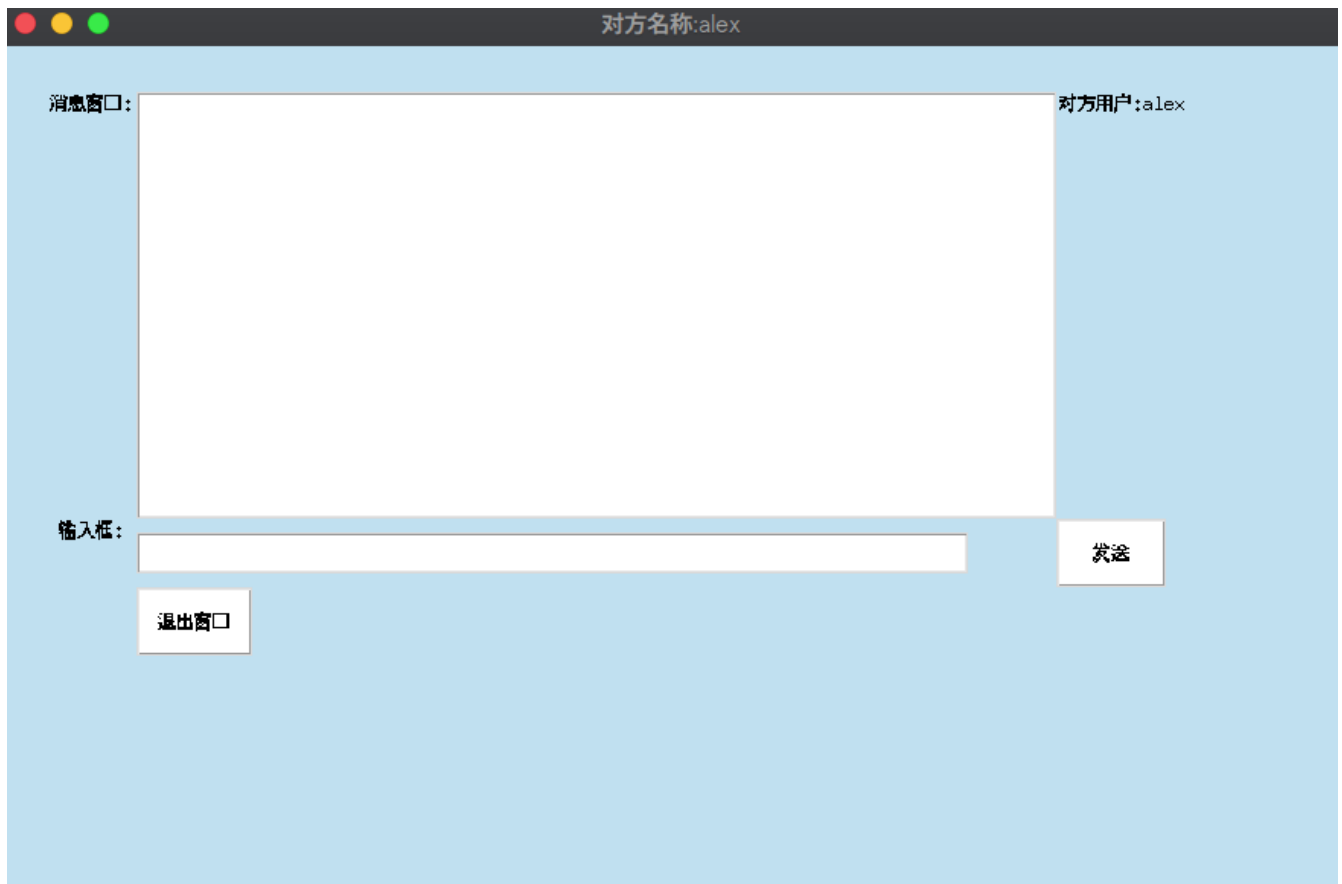
房间主界面



```
# 房间主界面
class Main_Chat_application(Tkinter.Frame):

    def __init__(self, master=None, clientp = None, room_name = None, user_name=None):
        Tkinter.Frame.__init__(self, master, bg='#c0e0f0')
        self.grid(row = 0, column = 0, padx = 30, pady = 30)
        self.master.configure(bg='#c0e0f0')
        self.master.title('房间名称:' + room_name)
        self.master.geometry('1000x650+450+150')
        self.master.protocol("WM_DELETE_WINDOW", self.back_room) # 捕获窗口关闭事件
        self.client = clientp
        self.username = user_name
        self.client.room_pointer = self
        self.room_name = room_name      # 房间名称
        self.createWidgets()
```

私聊主界面



```
class Main_Person_application(Tkinter.Frame):

    def __init__(self, master=None, clientp = None, person_name = None):
        Tkinter.Frame.__init__(self, master, bg='#c0e0f0')
        self.master.configure(bg='#c0e0f0')
        self.grid(row = 0, column = 0, padx = 30, pady = 30)
        self.master.title('对方名称:' + person_name)
        self.master.geometry('880x550+500+150')
        self.master.protocol("WM_DELETE_WINDOW", self.back_chat) # 捕获窗口关闭事件
        self.client = clientp
        self.person_name = person_name      # 对方名称
        self.createWidgets()
```