Socket 编程项目——聊天室

本聊天室采用 C/S 架构，前端使用 PyQt5 设计，通过 Socket 在客户端和服务器端之间进行通信，由我一个人完成。

【功能】

- 比较美观易用的 GUI
- 登录、注册功能
- 服务器端会打印正在处理的信息，方便观察及调试
- 支持登陆界面的键盘操作（Tab 换行，Enter 登陆）
- 使用 SQLite 存储用户名及其密码
- 账号只能在一处登陆，重复登陆会被拒绝
- 自己设计的包含 header 和消息体的协议
- 登录注册过程中的错误处理机制
- 聊天界面中，有新消息自动滚动到底部
- 有用户登陆、退出会发送广播并更新用户列表
- 支持多行发送（Enter 换行）
- 发送消息后自动清空输入栏
- 支持向所有人发送信息（广播）
- 支持向一部分人发送信息（组播）
- 支持向单个人发送信息
- 一键全选/全不选联系人
- 支持发送文件，如图片、txt 文件等

【文件目录】

|

| chat.py　　聊天室主页面

| client.py　　客户端

| header.py　　协议 header 的定义

| login.py　　登陆界面

| server.py　　服务器端

| - image　　GUI 的背景图像

|　| chat_bg.jpg　聊天室的背景

【使用方法】

先启动服务器：

python3 server.py

再启动（多个）客户端：

python3 client.py

【具体实现】

· **协议设计**

聊天室的工作流程是：用户通过客户端把 Request 发送给服务器端，Request 由 header 和消息体组成。服务器端解析 Request，根据 Request 给不同的客户端发送不同的 Response，Response 同样包括

header 和消息体。客户端收到 Response 后，解析 Response，根据 Response 在用户界面上执行不同的操作。

header 与消息体用 ‘\r\n’ 分隔开。数据都用 utf-8 编码后传输。

协议的 header 设计如下：

Request：

header	含义	消息体格式
100	登出	Username
101	登陆	Username ‘\r\n’ Password
102	注册	Username ‘\r\n’ Password
106	发送消息	多个用 ‘\t’ 隔开的 Username ‘\r\n’ Message
108	得到所有在线的用户	NULL
109	发送文件	文件内容（二进制形式）

Response：

header	含义	消息体格式
200	发送文件	文件内容（二进制形式）
201	密码错误	NULL
202	重复登陆	NULL
203	登出成功	Username
204	登陆成功	Username
205	新用户登陆	Username

206	注册成功	NULL
207	注册重复	NULL
208	消息发送出错	NULL
209	发送消息	Sender ‘\r\n’ Message
210	在线用户	多个用 ‘\r\n’ 隔开的 Username

- **服务器端**

在启动服务器的时候，会链接到存有用户名和密码的数据库，如果没有这样的数据库就会创建：

```
#·连接数据库，没有会自动创建文件
if·not·os.path.exists("info.db"):
····conn·=·sqlite3.connect("info.db")
····c·=·conn.cursor()
····c.execute('CREATE·TABLE·user(username·TEXT·PRIMARY·KEY·NOT·NULL,·password·TEXT)')
····conn.commit()
····conn.close()

db·=·sqlite3.connect('info.db')
cursor·=·db.cursor()
db.commit()
```

接下来服务器端会初始化一个 socket 连接到一个地址并开始 listen，服务器会通过这个 socket 向客户端发送消息。服务器端使用 select 模块对所有 socket 的合集进行监听（开始时只有服务器端的一个 socket）。这里调用 select 是为了完成非阻塞式的 I/O。select 会监听合集中所有句柄是否发生变化，所以从 select 函数的第一个返回值可以得到有哪些 socket 发生了改变。如果改变的 socket 是服务器端的 socket，说明有新用户要求链接；否则，是已经登陆的用户发送了信息，交由 handle 函数处理。在轮询过程中，如果出现异常，就把对

应的 socket 从合集中去除。

```
while True:
    #开始select监听,对input_list中的服务器端server进行监听
    readable, writable, exceptional = select.select(connections, outputs, [])
    #handle inputs
    #循环判断是否有客户端连接进来,当有客户端连接进来时select将触发
    for s in readable:
        #有新用户连接
        if s == sock:
            conn, client_address = s.accept()
            connections.append(conn)
            users[conn] = None
        #老用户发来信息，需要处理
        else:
            try:
                data = s.recv(BUFFER_SIZE)
                assert len(data) > 0, 'server receive empty message'
                handle(s, data)
            except Exception:
                sock.close()
                sys.exit()

    #处理异常的情况
    for s in exceptional:
        print('exception condition on', s.getpeername())
        #Stop listening for input on the connection
        connections.remove(s)
        if s in outputs:
            outputs.remove(s)
        sock.close()
        sys.exit()
```

handle 函数会调用 split 函数解析客户端发来的数据，用 if-elif 语句根据 header 执行不同的操作，定义在 server.py 中。比如如果 header 是 login，就会先在数据库中查找用户名和密码是否正确，再看用户是否已经登陆，登陆成功就会更新 connects 和 users 这两个词典，然后把这个消息发送给所有其他的 socket。

```
elif splt[0] == c_login:
    #在数据库中查找user
    cursor.execute("select * from user where username = '{0}' and password = '{1}'".format(splt[1], sp
    user = cursor.fetchone()
    if user is None:
        s.sendall(str(s_password_wrong).encode('utf-8'))
        print('password wrong')
    elif user[0] in connects.keys():
        #用户已经登陆了
        s.sendall(str(s_login_repeat).encode('utf-8'))
        print('login repeat')
    else:
        #登录成功
        connects[user[0]] = s
        users[s] = user[0]
        print(user[0])
        #给这个socket发送login_success
        s.sendall((str(s_login_success) + '\r\n' + user[0]).encode('utf-8'))
        print('login success' + user[0])
        #s向其他socket发送新登录的用户名
        print('send: ' + str(s_new_login) + '\r\n' + str(user[0]))
        send_to_others(s, str(s_new_login) + '\r\n' + str(user[0]))
        print('send new login success' + user[0])
```

其中，connects 和 users 是完成服务器端的两个重要数据结构，记录在线的用户的套接字和用户名，分别可以从用户名得到 socket 和从 socket 得到用户名。

除了登陆，服务器端还完成了：

登出　　从 socket 的合集中移除相应的 socket, 从 connects 和 users 中删除相应的项

注册　　在数据库中寻找用户名是否已经被使用，如果没有被使用就报告注册成功

在线用户　将 users 字典所有的 key 发送给请求该操作的 socket

发送消息　先调用 split 函数解析出要发送给谁，然后把信息发送给这些接收者

发送文件　将文件发送给所有在线的用户

- **客户端**

客户端同样先初始一个 socket，然后链接到服务器端，在这个过程中同样考虑了链接过程中的异常处理。客户端会导入定义在 login.py 和

chat.py 中的登陆界面和聊天界面，先打开登陆界面，用户登陆成功之后会关闭登陆界面，打开聊天界面。

登陆界面和聊天界面用 PyQt5 设计，同时为了前后端的链接，添加了一些信号函数，供后端调用，并且前端的按钮、label 等控件也可以通过 socket 发送数据。

客户端导入 threading 和 select 模块，通过使用多线程和 select 来实现异步处理多个连接的功能。

客户端的核心函数是 receive，通过调用 select 监听所有在线的用户。

receive 可以处理的操作有：

接收文件	接收服务器发过来的文件，并且在聊天界面显示一个对话框，询问是否要接收文件，如果确定接收，就把这个文件保存到本地
在线用户	在聊天页面中把所有现在用户显示在用户栏中
登陆成功	登陆成功之后，需要关闭登陆界面，在打开聊天页面之前需要更新用户列表，即调用在线用户功能，然后需要更新 label，把 label 的内容更改为自己的用户名
新登陆	在聊天框里添加一行：**进入了聊天室，同时，更新用户列表
重复登陆	拒绝登陆行为，打开对话框，提示重复登陆了
密码错误	拒绝登陆行为，提示不存在这样的用户名或密码
注册成功	弹出对话框，告知用户注册成功
注册重复	拒绝注册行为，弹出对话框，告知用户用户名重复

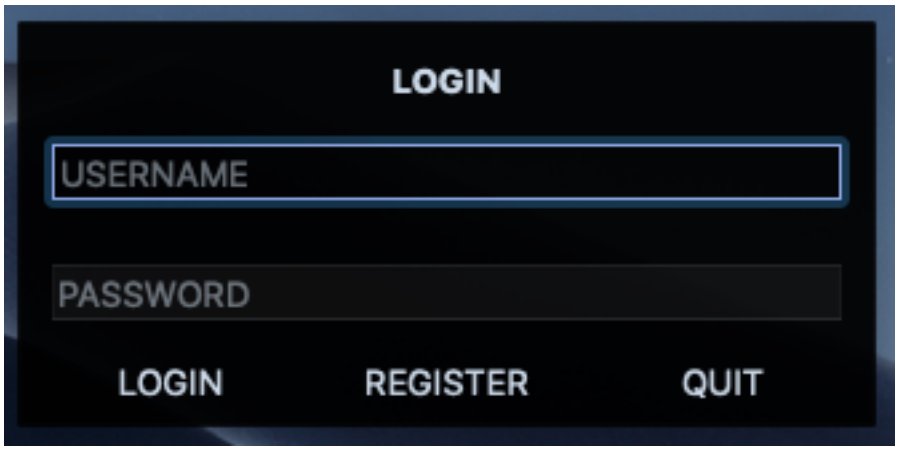
发送消息	通过 split 解析消息，得到发送者和消息，在聊天框中显示消息
登出	在聊天框中显示：**退出了聊天室，在用户列表中删除相应的用户名

· **前端**

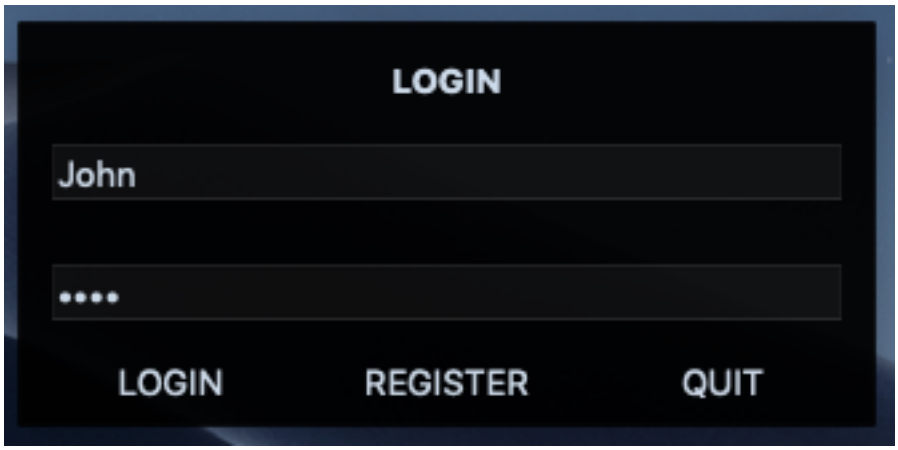
前端使用 PyQt5 设计，有登陆界面和聊天界面。要使后端可以改变前端内容，需要定义信号，即 pyqtSignal。

【成果展示】

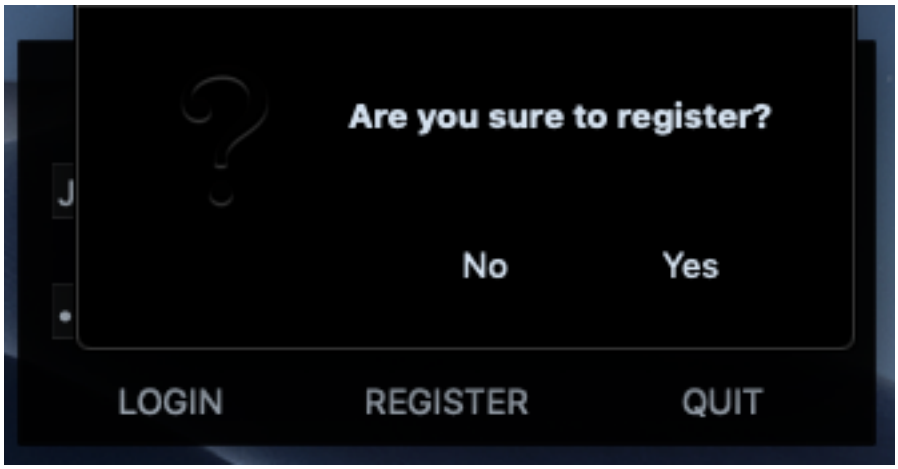
接下来从登陆注册界面开始，展示本聊天室的功能。登陆注册页面如下所示：



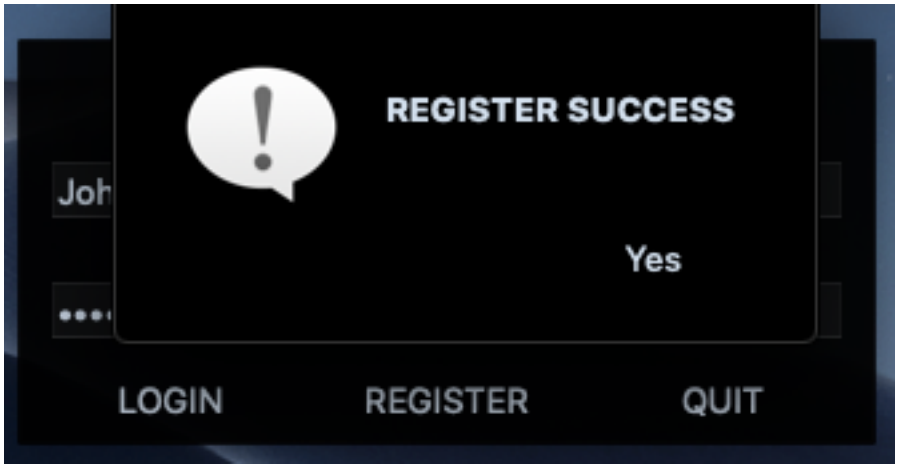
界面背景是半透明的黑色；输入的密码不可见，会变成点；使用键盘上的 Tab 可以换行；输入用户名和密码后按 Enter 可以登陆，登陆后进入聊天室主页面。



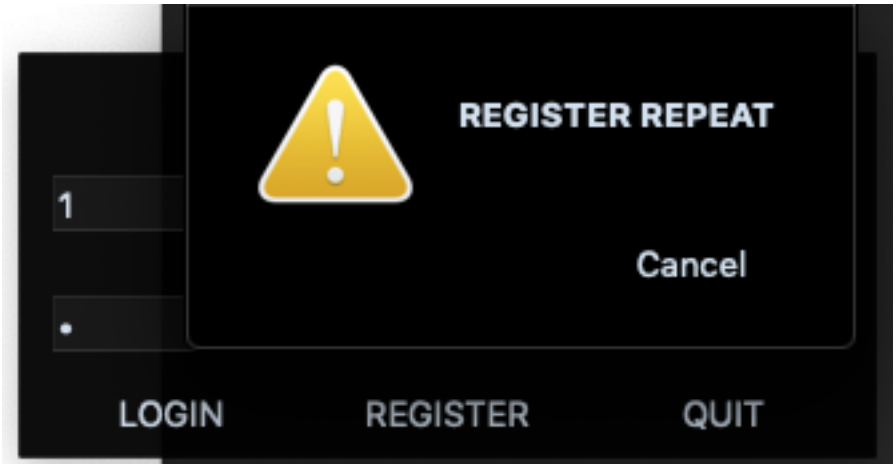
如果点击 REGISTER，会再次询问你是否为现在的用户名和密码注册一个新账号：



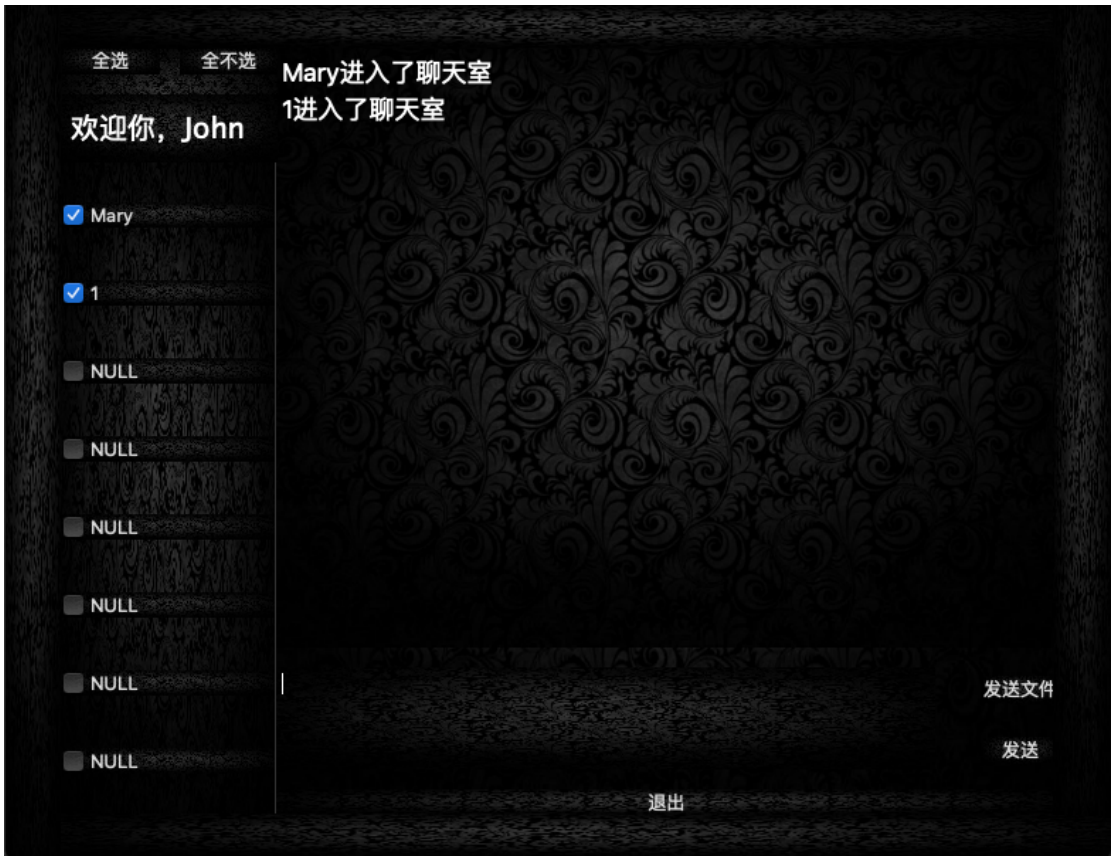
点击 Yes 继续注册，如果这个用户名之前没有被注册过，就会提示你注册成功：



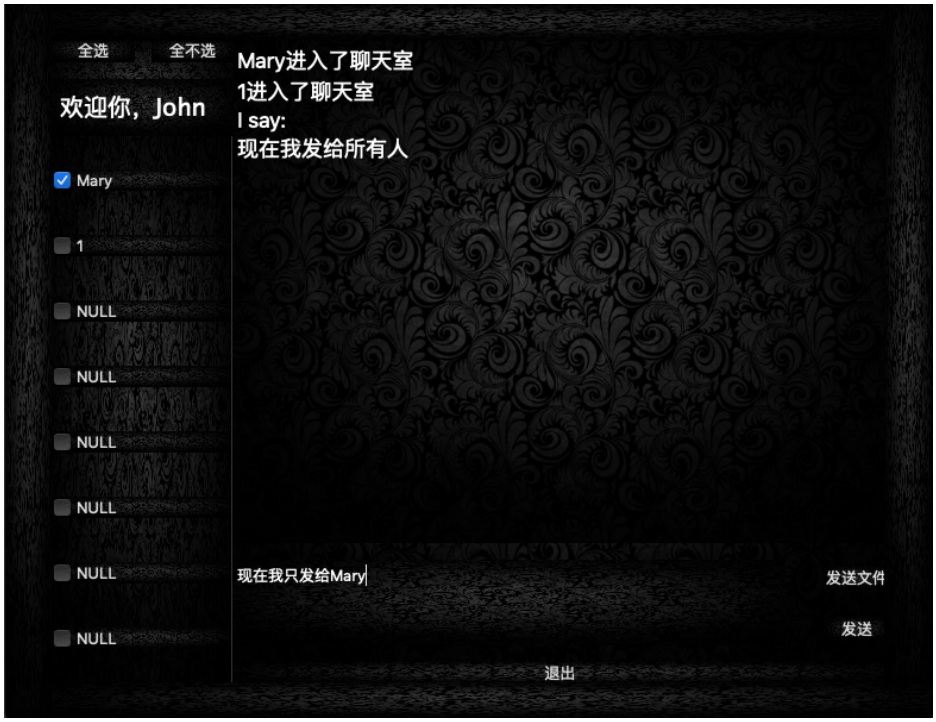
如果用户名已经被注册过，也会发出提醒：

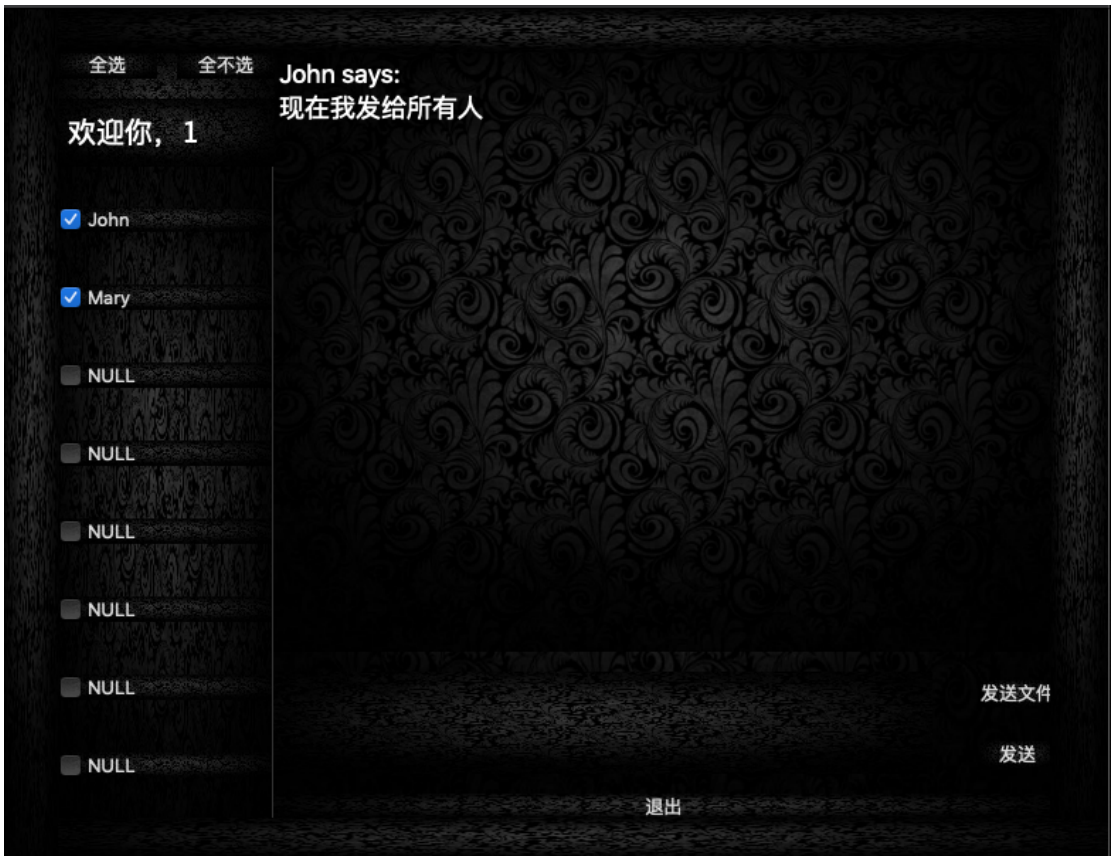


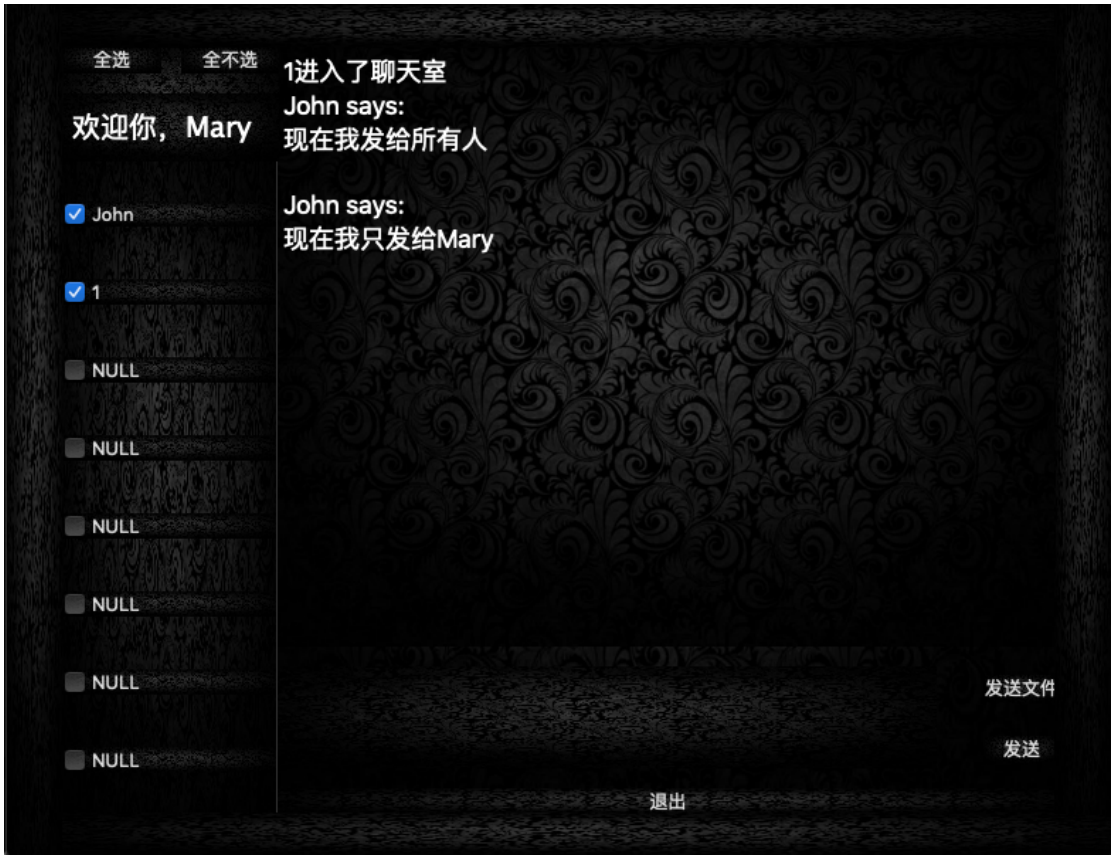
登陆成功后会进入聊天界面：



进入聊天界面时，会自动刷新左侧的用户栏，显示现在在线的用户。同时会在用户栏上方显示自己的用户名。可以在左侧用户栏选择想要发送消息给哪些用户，选中的即为要发送，不选的即为不发送。比如：

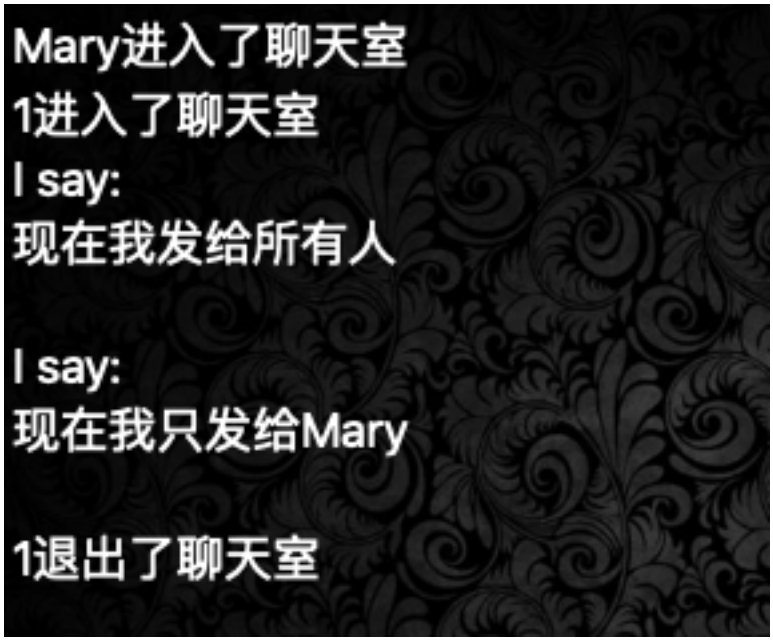

全选
全不选
欢迎你，John
Mary
1
NULL
NULL
NULL
NULL
NULL
NULL
Mary进入了聊天室
1进入了聊天室
I say:
现在我发给所有人
现在我只发给Mary
发送文件
发送
退出


全选
全不选
欢迎你，1
John
Mary
NULL
NULL
NULL
NULL
NULL
NULL
John says:
现在我发给所有人
发送文件
发送
退出



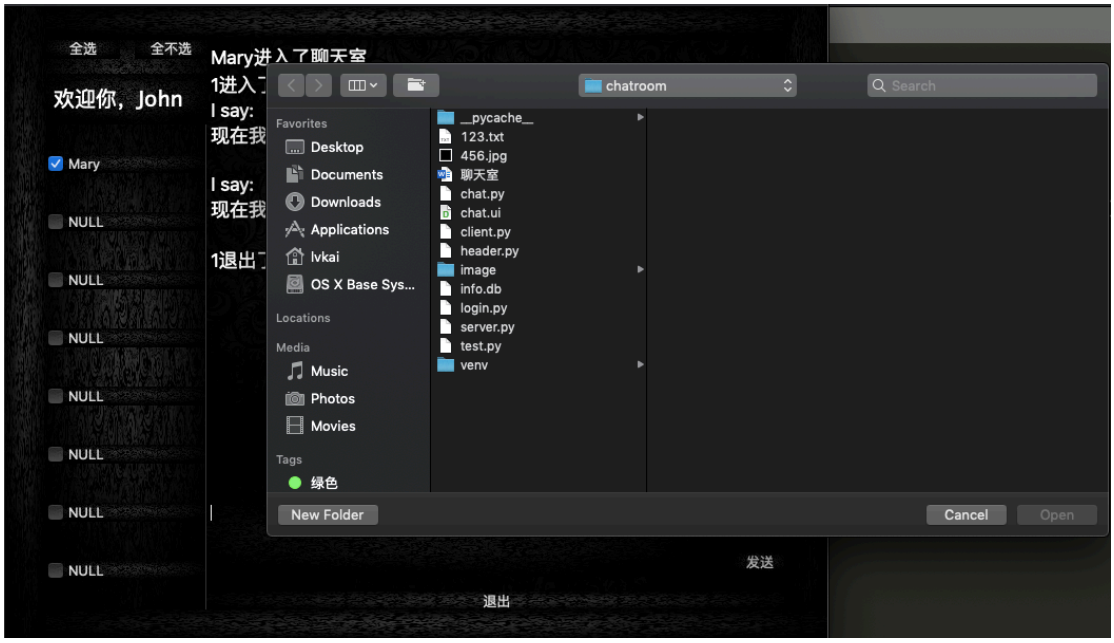
可以看出，当 John 只勾选 Mary，不勾选 1 时，只有 Mary 收到了消息，1 并没有收到消息。用户可以通过全选、全不选来选择用户。

有用户退出聊天室时，会有消息提醒：



同时如果有新消息，聊天框会自动滚动到最底部。

最后是发送文件功能：



会提醒其他用户是否接收：

确定为会下载到本地

【未来工作】

现在协议的设计不够安全，包括密码在内都是明文传输，容易被窃听。将来可以通过添加 ssl 库，像 https 与 http 的关系那样，将数据加密

后再传输。