

纽约大学-上海ICS聊天系统：规范和实施指南

[全面的 架构](#)

[模块：分度器 和 组 管理](#)

[模块：闲谈 效用](#)

[模块：客户 面 状态 机器](#)

[背景 状态 机器](#)

[协议 编码](#)

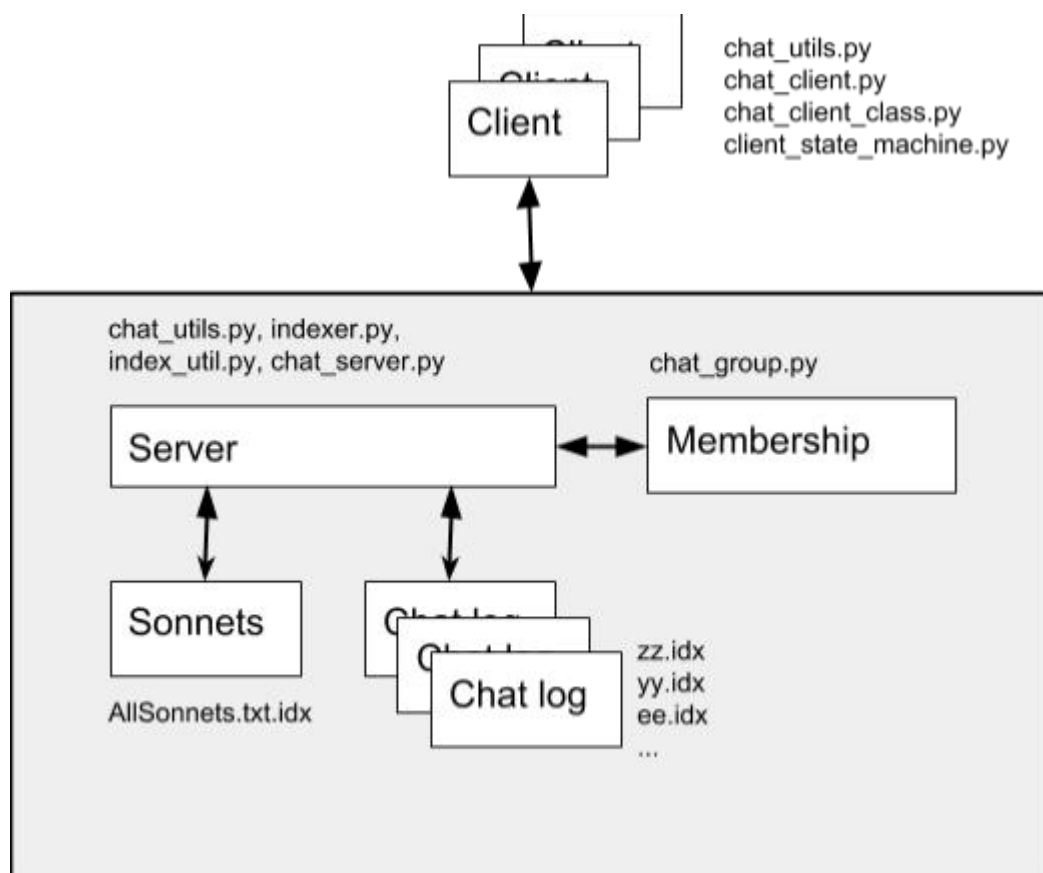
[模块：客户 面 状态 机器](#)

[模块：服务器 面](#)

说明：

- 单元项目1：索引器
- 单元项目2：集团管理
- 单元项目3：共两周
 - [部分 1：客户 面 状态 机器](#)
 - [部分 2：服务器 面 消息 处理 积分](#)

聊天系统架构



我们的聊天系统的整体架构和主要组件，以及组成系统的文件。

这是一个典型的分布式客户机-服务器系统，其中多个客户机与一个中心服务器进行交互。从概念上讲，这就是微信的构建方式。客户端彼此直接交互，但是实际发生的是服务器正在来回传递消息，并添加其他功能（如索引历史记录）。

可以有多个客户端，每个客户端要么空闲，要么积极参与与一组其他客户端的一个聊天会话。你可以把一个客户想象成微信的一个普通用户。**我们的系统很简单：它只允许在一组中聊天。**

该服务器有几个额外的模块：

- 会员管理模块，看看谁在和谁聊天。
- 一个聊天日志，每个用户一个。这允许用户用关键字搜索她过去的聊天历史。
- 一个十四行诗数据库，所以用户可以在她不聊天的时候问一首诗。

构成系统的文件，客户端：

- `chat_cmdl_client.py`，聊天客户端类。都是给定的。不需要改变它；的确，改变将由你承担风险！：)
- `chat_state_机器.py`：处理与聊天系统交互的主要事件。你实现它。

组成系统、服务器端的文件

- `chat_服务器`。给出了代码的一部分。您需要实现一个事件处理功能。
- `索引器`。索引信息和十四行诗。你已经在UP1中实现了它。
- `AllSonnets.txt`，`罗马.txt`。十四行诗和罗马到数字的转换。
- `聊天组`。会员处理。您已经在UP2中实现了。

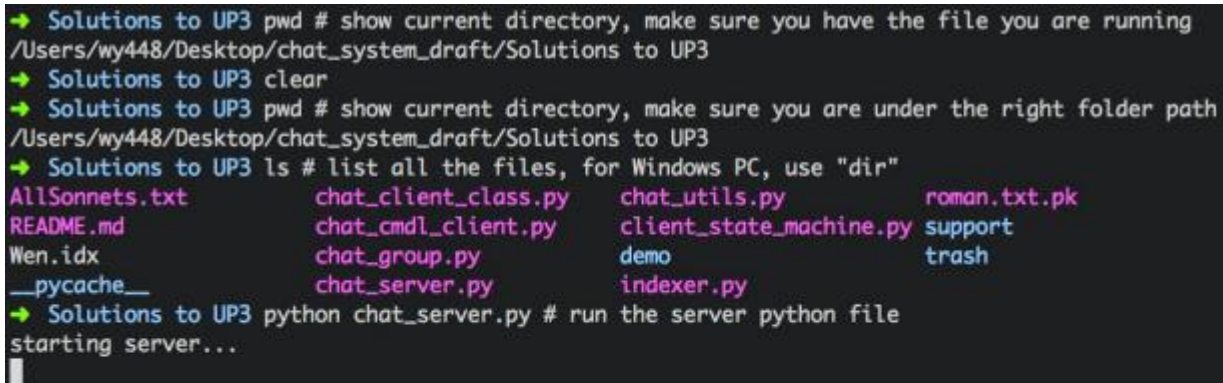
`index_util.py`和`chat_util.py`是我们提供的实用程序文件/模块。

完成后，您可以按以下方式运行它：

- 在一个控制台上：“`python chat_server.py`”。这将启动服务器。
- 在另一个控制台上：“`python chat_cmdl_client.py`”。这将启动客户端

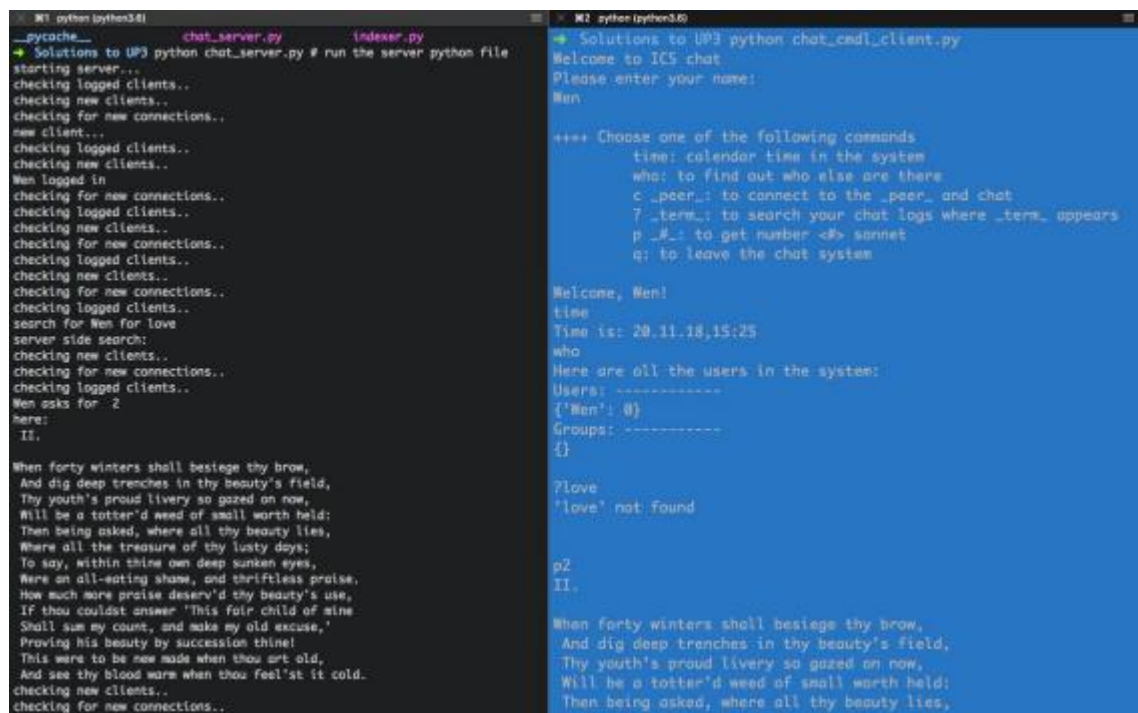
[请参见关于如何在单独的机器上运行聊天客户端和服务器的附录。](#)

当服务器启动时，你应该有以下截图：



```
→ Solutions to UP3 pwd # show current directory, make sure you have the file you are running
/Users/wy448/Desktop/chat_system_draft/Solutions to UP3
→ Solutions to UP3 clear
→ Solutions to UP3 pwd # show current directory, make sure you are under the right folder path
/Users/wy448/Desktop/chat_system_draft/Solutions to UP3
→ Solutions to UP3 ls # list all the files, for Windows PC, use "dir"
AllSonnets.txt      chat_client_class.py  chat_utils.py         roman.txt.pk
README.md           chat_cmdl_client.py   client_state_machine.py support
Wen.idx             chat_group.py         demo                  trash
__pycache__         chat_server.py        indexer.py
→ Solutions to UP3 python chat_server.py # run the server python file
starting server...
```

下面是一个屏幕截图，包括一个启动服务器（左）和一个客户端（右）：



```
M1: python (python3.6)
_pycache_ chat_server.py indexer.py
+ Solutions to UP3 python chat_server.py # run the server python file
starting server...
checking logged clients..
checking new clients..
checking for new connections..
new client...
checking logged clients..
checking new clients..
Wen logged in
checking for new connections..
checking logged clients..
checking new clients..
checking for new connections..
checking logged clients..
checking new clients..
checking for new connections..
checking logged clients..
search for Wen for love
server side search:
checking new clients..
checking for new connections..
checking logged clients..
Wen asks for 2
here:
II.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a totter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.
checking new clients..
checking for new connections..

M2: python (python3.6)
+ Solutions to UP3 python chat_cmdl_client.py
Welcome to ICS chat
Please enter your name:
Wen

++++ Choose one of the following commands
time: calendar time in the system
who: to find out who else are there
c _peer_: to connect to the _peer_ and chat
? _term_: to search your chat logs where _term_ appears
p _#_: to get number <#> sonnet
q: to leave the chat system

Welcome, Wen!
time
Time is: 20.11.18,15:25
who
Here are all the users in the system:
Users: -----
{'Wen': 0}
Groups: -----
{}

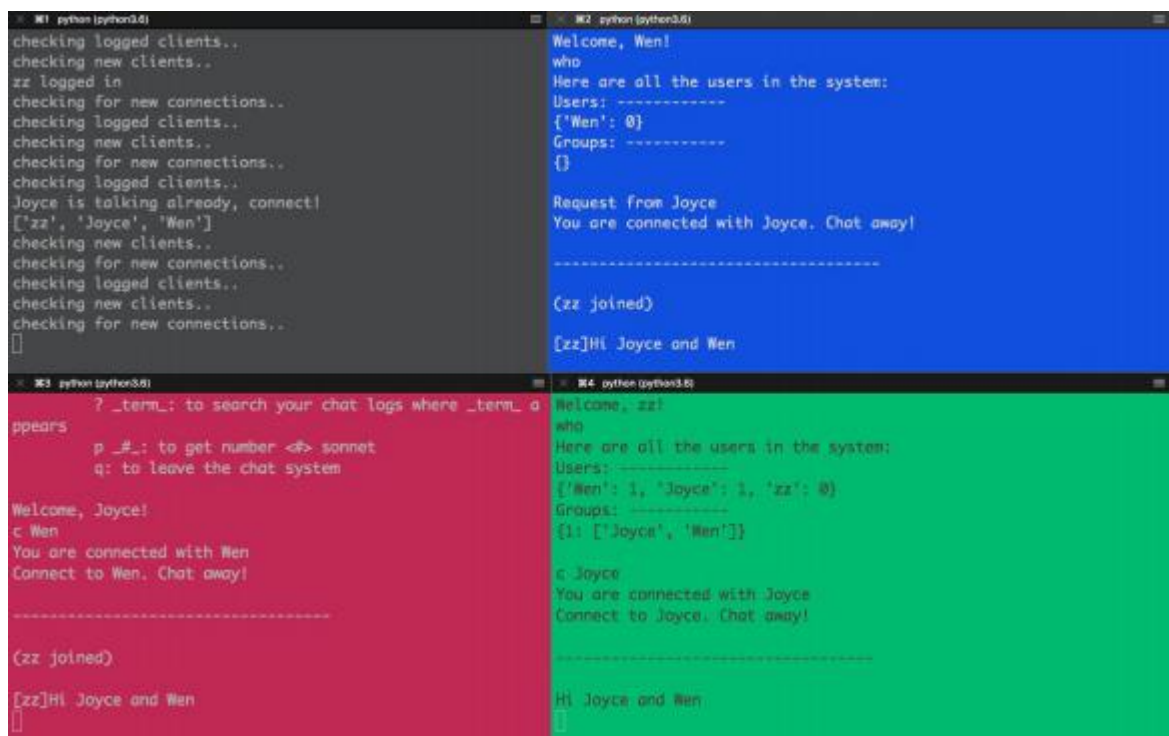
?love
'love' not found

p2
II.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a totter'd weed of small worth held:
Then being asked, where all thy beauty lies,
```

下面是聊天开始的截图，顺序是：

- 服务器（左上）启动
- 温（右上）先加入；问题一个“谁”：他是唯一的一个
- Joyce（左下）加入下一个：连接到Wen
- zz（右下）加入了最后一个：连接到Joyce，因此加入了小组对话



```
M1: python (python3.6)
checking logged clients..
checking new clients..
zz logged in
checking for new connections..
checking logged clients..
checking new clients..
checking for new connections..
checking logged clients..
Joyce is talking already, connect!
['zz', 'Joyce', 'Wen']
checking new clients..
checking for new connections..
checking logged clients..
checking new clients..
checking for new connections..
[]

M2: python (python3.6)
Welcome, Wen!
who
Here are all the users in the system:
Users: -----
{'Wen': 0}
Groups: -----
{}

Request from Joyce
You are connected with Joyce. Chat away!

-----
(zz joined)
[zz]Hi Joyce and Wen

M3: python (python3.6)
? _term_: to search your chat logs where _term_ a
ppears
p _#_: to get number <#> sonnet
q: to leave the chat system

Welcome, Joyce!
c Wen
You are connected with Wen
Connect to Wen. Chat away!

-----
(zz joined)
[zz]Hi Joyce and Wen
[]

M4: python (python3.6)
Welcome, zz!
who
Here are all the users in the system:
Users: -----
{'Wen': 1, 'Joyce': 1, 'zz': 0}
Groups: -----
{1: ['Joyce', 'Wen']}

c Joyce
You are connected with Joyce
Connect to Joyce. Chat away!

-----
Hi Joyce and Wen
[]
```

模块

下面描述了每个模块，并在适用时提供了实现指南。有很多模块。但是，您已经完成了两个关键的任务，并且您只需要实现两个函数，一个在客户端上，另一个在服务器上。

索引器和集团管理

UP1和UP2。

索引器：

- 类PIndex存储和索引的十四行诗
- 类索引索引客户端之间的聊天，并响应搜索。集团管理层：
- 记录对等体连接和离开系统的时间
- 响应系统中成员的查询（通过客户端发出的“谁”命令）
- 让一个对等体连接到另一个体（e。g。c彼得“）
- 让一个同伴退出一个组（通过客户端的“再见”）

实用程序功能

聊天_utils.py是作为一个模块导入。它有几件事值得一提：

```
6  CHAT_IP = socket.gethostbyname(socket.gethostname())
7  CHAT_PORT = 1112
8  SERVER = (CHAT_IP, CHAT_PORT)
9
10 menu = "\n++++ Choose one of the following commands\n \
11         time: calendar time in the system\n \
12         who: to find out who else are there\n \
13         c _peer_: to connect to the _peer_ and chat\n \
14         ? _term_: to search your chat logs where _term_ appears\n \
15         p _#_: to get number <#> sonnet\n \
16         q: to leave the chat system\n\n"
```

第6-8行给出了客户端连接到服务器时的服务器地址和端口。你不需要担心它。目前，该服务器与客户机运行在同一台机器上。稍后，我们将扩展如何连接到运行在其他机器上的服务器。

```
18  S_OFFLINE   = 0
19  S_CONNECTED = 1
20  S_LOGGEDIN  = 2
21  S_CHATTING  = 3
```

以上是客户端可以处于的四种状态。事实上，在我们当前的实现中，我们将不会使用S_CONNECTED。

插座注意事项：

对于程序在互联网上相互交谈，他们使用插座。这是一个我们不会讨论的高级话题。现在，把插座当成你和朋友通话的电话。我们提供了两个实用程序例程：

- `mysend (s, msg)` 接受一个字符串`msg`，并发送一个套接字`s`。
- `myrecv (s)`在味精中返回一个字符串。

我们的代码已经设置了套接字，所以您不需要实现它们。

演示：你可以在演示文件夹中找到4个简单的演示文件。

客户端`demo.py`

客户端`demo_multi_client.py`

服务器`demo.py`

服务器`demo_multi_client.py`

状态机

您需要根据下图正确地更新用户状态。

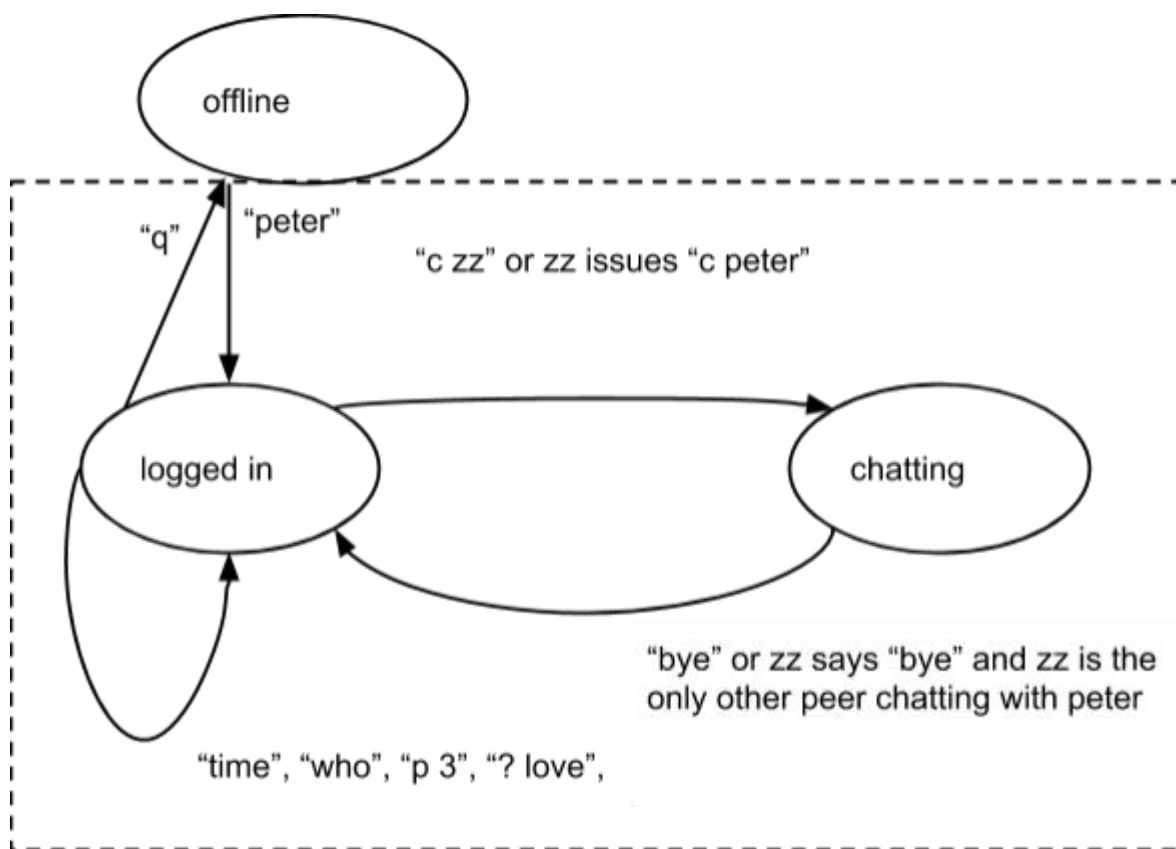
```
18  S_OFFLINE   = 0
19  S_CONNECTED = 1
20  S_LOGGEDIN  = 2
21  S_CHATTING  = 3
```

快速浏览一下wiki页面：[http://en. 维基百科.org/wiki/有限状态机器](http://en.维基百科.org/wiki/有限状态机器)

状态机的基本步骤是在某个事件之后从一个状态移动到另一个状态。该转换可能会产生一些操作。当我们描述如何实现客户端状态机时，它会变得更加清楚。py.

下面是针对我们的聊天客户端的一个简化的状态机。虚线框是客户端状态机的逻辑。

py.



协议代码

就像人们必须共享一种共同的语言（或一组符号）才能进行通信一样，客户机和服务器也必须共享一组代码，以便他们理解请求的意图。下面总结了客户端和服务端之间的消息。每个消息(e.g. {“动作”：“登录”，“名称”：“zz”})是一个字典，当服务器和客户端之间交换时，它们由json打包/解压缩。dump/json. 装载

以下所有消息都假定您以zz的身份登录。

演示：json如何工作的一个示例可以在提供的演示文件夹中找到。

状态S OFFLINE中的操作（到S LOGGEDIN）

当用户输入：“他的昵称”时，

服务器响应：“登录成功”或“名称已存在”

```
客户端发送 {“操作”：“登录”、“名称”：“zz”}  
服务器响应 {“操作”：“登录”，  
    “状态”：[“好”，“重复”，  
    “味精”：[“登录成功”，“名称已存在”]  
}
```

状态S LOGGEDIN中的操作

当用户类型为“时间”时，

服务器用字符串编码的时间进行响应

```
“客户端发送”：{“动作”：“时间”}  
“服务器响应”：{“动作”：“时间”，“味精”：“13: 40: 33”}
```

当用户类型为“谁”时，

服务器与系统中的成员和聊天组进行响应

```
客户端发送 {“操作”：“列表”}  
服务器响应 {“动作”：“列表”，“msg”：“{zz: 0, wen: 1, cc: 1}”}
```

当用户类型：“？”爱

服务器响应聊天历史记录，其中包含“爱”的聊天记录

```
客户端发送 {“动作”：“搜索”、“目标”：“爱”}  
“服务器响应”：{“动作”：“搜索”，“msg”：[“（zz）12: 50: 01我爱ICS”，  
“（温）13: 22: 22我不喜欢ICS”]}
```

当用户输入：“p3”时，服务器用十四行诗#3（或III）进行响应

```
“客户端发送 {“动作”：“诗”，“目标”：“3”}  
服务器响应 {“”，“动作”：“诗”，“味精”：“III”。当四十个冬天将被围攻时  
你的眉毛，……”}
```


从S_LOGGEDIN到S_CHATTING的操作

当用户zz类型：“cpeter”时，服务器应该：

1. 让他知道他成功地联系上了彼得，
2. 让彼得（和其他已经和彼得聊天的人）知道zz加入了
3. 或者让zz知道连接不成功的原因

```
//Successfully
“客户端发送”：{ “动作”：“连接”、“目标”：“peter” }
“服务器响应zz”：{
    “动作”：“连接”、“状态”：“成功”、“味精”：“连接到彼得” }
“服务器响应peter”：{
    “action”：“connect”, “status”：“request”, “from”：“zz”, msg：“zz joined”}
“服务器响应组成员1”：{
    “action”：“connect”, “status”：“request”, “from”：“zz”, msg：“zz joined”}
“服务器响应组成员2”：{
    “action”：“connect”, “status”：“request”, “from”：“zz”, msg：“zz joined”}

//案例1失败
客户端发送 “{” 动作” “连接” “目标” “zz” }
“服务器响应zz”：{
    “操作”：“连接”、“状态”：“自我”、“味精”：“无法连接到您自己” }

//案例2失败
“客户端发送”：{ “动作”：“连接”、“目标”：“peter” }
“服务器响应zz”：{
    “行动”：“连接”，“状态”：“无用户”，“msg”：“彼得不是”
联机（无用户） “}
```

当Peter（同伴）输入：“czz（你）”时，zz将会连接到Peter

```
“对等方发送”：{ “操作”：“连接”、“状态”：“请求”、“来自”：“Peter” }
服务器发送 “{” 动作：“连接”、“目标”：“zz”、“msg”：“您已连接到彼得 “}
```

状态S_CHATTING中的操作

当用户类型为“hi”时

```
“客户端发送”：{ “行动”：“交换”，“消息”：“嗨”，“从”：“zz” }
“服务器响应zz”：
```

没有回应，只要通过文本(i。e. 对zz小组里的每一个同伴说

“服务器响应到其他组成员，如果存在”：

```
{  
  “行动”：“交换”，“来自”：“zz”，“消息”：“嗨”]  
}
```

从状态S_CHATTING到S_LOGGEDIN的操作

当用户类型：“bye”

客户端发送“{”操作”“断开连接”}

“服务器发送”：不需要响应；zz离开聊天组

当其他用户输入“再见”时，我是唯一剩下的一个

“服务器响应“：{“动作”：“断开连接”，“味精”：“所有人都离开了，你是孤独的”}

UP3实现1：客户端状态机。py

您只需要修改客户端状态机。我们鼓励高级学生阅读聊天客户端.py（这是主条目）和chat_client_class.py。我们已经处理了登录、注销、设置连接等工作。在这两个文件中。当chat_client初始化时，它将拥有类ClassSM的一个成员，登录后，它将输入S_LOGGEDIN，这就是您的工作开始的地方。也就是说，只在客户端处于状态S_LOGGEDIN后才调用proc。

```
51     def proc(self, my_msg, peer_msg):
52         self.out_msg = ''
53         #=====
54         # Once logged in, do a few things: get peer listing, connect, search
55         # And, of course, if you are so bored, just go
56         # This is event handling instate "S_LOGGEDIN"
57         #=====
58         if self.state == S_LOGGEDIN:
59             # todo: can't deal with multiple lines yet
60             if len(my_msg) > 0:
61
62                 if my_msg == 'q':
63                     self.out_msg += 'See you next time!\n'
64                     self.state = S_OFFLINE
65
66                 elif my_msg == 'time':
67                     mysend(self.s, json.dumps({"action": "time"}))
68                     time_in = json.loads(myrecv(self.s))["results"]
69                     self.out_msg += "Time is: " + time_in
70
```

这是您需要完成的功能。它需要三个参数：

- my_msg: 此用户的传出消息
- peer_code, peer_msg: 来自其同行的代码和相关的传入消息。

这个函数的输出存储在自存储中。out_msg。

上面的代码显示了处理“q”和“时间”的示例。“时间”命令的写入方式是典型的：通过套接字发送消息，并记录任何要自我输出的内容。out_msg。

代码中总共有2个传递语句可以找到您需要完成的地方，如下所示：第一个传递语句：

```

107
108         if len(peer_msg) > 0:
109             peer_msg = json.loads(peer_msg)
110             if peer_msg["action"] == "connect":
111
112                 # -----your code here-----#
113                 print(peer_msg)
114                 pass
115
116
117
118                 # -----end of your code-----#
119
120     #=====
121     # Start chatting, 'bye' for quit

```

第二个传递语句:

```

120     #=====
121     # Start chatting, 'bye' for quit
122     # This is event handling instate "S_CHATTING"
123     #=====
124     elif self.state == S_CHATTING:
125         if len(my_msg) > 0:      # my stuff going out
126             mysend(self.s, json.dumps({"action":"exchange", "from":["" + self.s.name, peer.name], "data": my_msg}))
127             if my_msg == 'bye':
128                 self.disconnect()
129                 self.state = S_LOGGEDIN
130                 self.peer = ''
131         if len(peer_msg) > 0:    # peer's stuff, coming in
132
133
134             # -----your code here-----#
135             peer_msg = json.loads(peer_msg)
136             print(peer_msg)
137             pass
138
139
140
141             # -----end of your code-----#
142
143             # Display the menu again
144             if self.state == S_LOGGEDIN:
145                 self.out_msg += menu
146     #=====
147     # invalid state
148     #=====

```

UP3实现2: chat_server.py

服务器，将在chat_server中实现.py，将接收来自客户端的各种各样的消息（消息，如{“动作”：“时间”}，{“动作”：“断开连接”}等）。服务器的主要工作是响应所有这些消息。为了处理它们，服务器类维护了大量的字典。最重要的是：

自我.logged_name2sock: 将客户端的名称映射到其套接字

自我.logged_sock2name: 与上面的内容相反；将一个套接字映射到客户端名称

自我.组: 组管理部分，记账系统中同行的状态

自我.索引: 将客户端的名称映射到其聊天索引

```
18 class Server:
19     def __init__(self):
20         self.new_clients = [] #list of new sockets of which the user id is not known
21         self.logged_name2sock = {} #dictionary mapping username to socket
22         self.logged_sock2name = {} # dict mapping socket to user name
23         self.all_sockets = []
24         self.group = grp.Group()
25         #start server
26         self.server=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
27         self.server.bind(SERVER)
28         print(SERVER)
29         self.server.listen(5)
30         self.all_sockets.append(self.server)
31         #initialize past chat indices
32         self.indices={}

```

如果您想知道服务器是如何启动的，这里是主循环。你真的不需要了解很多，但有一个想法很好：

```
183 #=====
184 # main loop, loops *forever*
185 #=====
186 def run(self):
187     print('starting server...')
188     while(1):
189         read,write,error=select.select(self.all_sockets,[],[])
190         print('checking logged clients..')
191         for logc in list(self.logged_name2sock.values()):
192             if logc in read:
193                 self.handle_msg(logc)
194         print('checking new clients..')
195         for newc in self.new_clients[:]:
196             if newc in read:
197                 self.login(newc)
198         print('checking for new connections..')
199         if self.server in read :
200             #new client request
201             sock, address=self.server.accept()
202             self.new_client(sock)
203
204 def main():
205     server=Server()
206     server.run()
207
208 main()

```

基本上，服务器通过它的每个套接字进行循环，适当地处理它们，无论是使用它们来接收和处理消息，登录它们，还是在它自己的套接字的特殊情况下，接受连接请求。

您需要完成函数`handle_msg()`。看看`handle_msg()`的前几行话，

```
89 #=====
90 # main command switchboard
91 #=====
92 def handle_msg(self, from_sock):
93     #read msg code
94     msg = myrecv(from_sock)
95     if len(msg) > 0:
```

我们看到它从发送消息的客户端的套接字`_sock`中获取一个参数。为了真正获得消息，我们需要使用`myrecv`（来自`_sock`）。

现在，根据代码，服务器将向客户端发送消息。在客户端连接和断开连接的情况下，将调用服务器的组对象的“连接”和“断开连接”函数。

你的工作将是填写`(el)` if块，处理，有5个传递语句围绕“行动”：“时间”，你需要完成。

- {“操作”：“交换”，“消息”：“<astr>”}
- {"action": "list"}
- {“动作”：“诗”，“目标”：“<十四行诗编号>”}
- {"action": "search", "target": 'love'}

```
179 #=====
180 #           time
181 #=====
182 elif msg["action"] == "time":
183     ctime = time.strftime('%d.%m.%y,%H:%M', time.localtime())
184     mysend(from_sock, json.dumps({"action": "time", "results": ctime}))
185 #=====
186 #           search: : IMPLEMENT THIS
187 #=====
```

“动作”：“列表”和“动作”：“列表”的情况是最容易开始的。在这两种情况下，您只需要以字符串的形式发送回适当的消息。

对于“行动”：“诗”，你得到了一个错误的实现。你的工作就是纠正它。提示：回想一下，当代码是“动作”：“诗”时，你收到的信息是形式的
{“action”: “poem”, “target”: “<sonnet number>”}.

“行动”：“交换”将会更具挑战性。

对于“操作”：“交换”，您收到的消息的形式是{“操作”：“交换”，“消息”：“<astr>”}，其中字符串是由客户端发送的内容。把它发给客户组中的每个人！最后，不要忘记向索引各自消息否则，搜索(? 命令)不会工作。

执行顺序：

1. chat_server.py :

```
elif msg["action"] == "poem":  
    pass
```

2. chat_server.py

```
elif msg["action"] == "list":  
    pass
```

3. 聊天状态机器.py (状态=登录)

```
if len(peer_msg) > 0:  
    peer_msg = json.loads(peer_msg)  
    if peer_msg["action"] == "connect":  
        pass
```

4. chat_state_machine.py (状态= chatting) chat_server.py

```
if len(peer_msg) > 0:    # peer's stu  
    peer_msg = json.loads(peer_msg)  
    pass
```

```

elif msg["action"] == "exchange":
    from_name = self.logged_sock2name[from_sock]
    # Finding the list of people to send to
    # and index message
    pass
    the_guys = self.group.list_me(from_name)[1:]
    for g in the_guys:
        to_sock = self.logged_name2sock[g]
        pass

```

5. 聊天服务器。py

```

elif msg["action"] == "search":
    pass # get search search_rslt

```

在不同的机器上运行聊天客户端和服务端

使用chat_cmdl_client -d地址连接到远程服务器（在子网内）示例：

```

➔ Solutions to UP3 python chat_cmdl_client.py -d 127.0.0.1
Welcome to ICS chat
Please enter your name:

```

有趣的事情尝试

以下是一些你可以创造性地做的例子：

- 在S_ALONE状态下，“乒乓球等等”，服务器响应“乒乓等等”
- 在S_CHATTING的状态下，“_flip_所说的是真的”，服务器发送给对等点“[zz] _flip__true是说什么”