设计文档

一、总体思路

笔者通过多线程以及 UDP 协议下的 socket 建立客户端和服务器之间的连接。对于用户、商店和商品，分别设计了相应的数据结构，并通过两个函数分别处理客户端的请求和服务器的请求，其中都按不同命令做出了不同的处理。

二、具体实现

（一）UDP 连接的建立

本程序使用 UDP 协议下的 socket 进行客户端和服务器之间的通信。由于笔者使用的是 Python 语言，因此可以利用 Python 内置的 socket 库（当然，还有一个 socketserver 库，但笔者没有尝试）。Socket 有多种类型，比如 SOCK_STREAM 是对应于 TCP 协议的；而对于 UDP 协议，socket 中也有对应的 SOCK_DGRAM 类型，即数据报套接字。因此，我们将从这里入手建立 UDP 连接。

在客户端，可以通过这样一条语句来建立连接：

```
self.client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

这里的 client 是定义在 Client 类中的，这个将在下文展开。可以看到，socket 库中有一个 socket()函数，其中有两个参数，第一个是 AF_INET，指定的是网络协议，这里说明是使用 IPv4 的协议；而第二个参数就是刚刚提到的 socket 类型，在 UDP 协议之下，应当是 SOCK_DGRAM。

对于服务器，构建方式也是一样的：

```
self.serv = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

但这条语句只是发起连接，真正的通信还没有建立起来。因此，我们首先设置服务器，服务器的 IP 地址设置为 127.0.0.1，端口为 6667，而客户端的 IP 地址与服务器相同，端口在 10000-65535 之间随机产生。它们的地址是一个由 IP 地址和端口号组成的二元组，socket 中的 bind 函数所接受的参数也是一个二元的元组。

接下来就要使用 socket 中的 bind 的函数了。它的功能即是将地址和套接字绑定，保证服务器和客户端能在固定的端口进行监听和发送数据。以服务器为例：

```
loc_ip = '127.0.0.1'
loc_port = 6667
loc_addr = (loc_ip, loc_port)
self.serv = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
self.serv.bind(loc_addr)
```

在建立连接之后，就要发送和接收数据了。Socket 中有 recvfrom()函数和 sendto()函数，分别提供接收数据和发送数据的功能。

以客户端为例，笔者写了两个函数以方便调用：

```
def recv_msg(self):
    BUFSIZE = 2048
    msg, addr = self.client.recvfrom(BUFSIZE)
    print 'Received from ' + str(addr[0]) + ' : ' + msg
    return msg, addr

def send_msg(self, msg):
    self.client.sendto(msg, self.Serv_Addr)
    print 'Message sent to server'
```

对于 recv_msg()来说，设置了一个 2048 字节的缓冲区大小，同时，recvfrom()函数会返回接收到的信息和地址，这里把它们分别赋给 msg 和 addr。对于 send_msg()来说，调用了 socket 当中的 sendto()函数，将信息 msg 发给服务器，其地址是 Serv_Addr。

服务器和客户端的构造方式相同。

现在，服务器和客户端都具备收发数据的功能，然而，如何让它们同时运行呢？这就需要使用多线程来实现并行的运行了。

（二）多线程处理

查阅资料后，笔者得知 socket 中有 threading 库能够实现多线程。通过 threading 建立多线程有多种方式，笔者选择直接调用 threading 中的 Thread 函数，其参数是一个函数，即表明这个线程要运行什么样的函数。自然，在客户端和服务器，都需要两个线程，一个用来收取对方发来的数据，另一个用来发送消息，这样才能实现网上商城的各功能。对于每个线程，都需要一直运行，比如服务器需要一直在 6667 端口监听，因此笔者使用 while 循环来实现。

```
client = Client()

def clin_read():
    while True:
        client.recv_msg()

def clin_send():
    while True:
        msg = raw_input()
        if not msg:
            break
        client.send_msg(msg)
```

client 属于 Client()类，即对它进行初始化。而接下来的 clin_read()和 clin_send()函数则实现了上述的功能。

然后定义一个线程池 threads，当然，也可以直接在建立后就运行：

```
threads = []
```

对于发数据的线程和收数据的线程，先将它们分别加入地址池，再一起运行：

```
send = threading.Thread(target= clin_send)
threads.append(send)
read = threading.Thread(target= clin_read)
threads.append(read)

for t in threads:
    t.start()
```

对于服务器来说也是类似的：

```
server = Server()

def serv_recv():
    while True:
        msg, addr = server.recv_msg()
        server.handle_client_command(msg, addr)

def serv_self():
    while True:
        msg = raw_input()
        server.handle_server_command(msg)

threads = []
```

serv_recv()函数用来接收客户端的数据，并调用 handle_client_command()函数来对用户的命令进行响应和处理，该函数将在下文展开；而 serv_self()函数则对应于发送数据和管理员用户的操作，并调用 handle_server_command()函数来处理管理员用户的各种操作，将在下文展开。

线程的建立也是类似的：

```
send = threading.Thread(target=serv_self)
threads.append(send)
read = threading.Thread(target=serv_recv)
threads.append(read)

for t in threads:
    t.start()
```

客户端和服务器的线程都建立好之后，我们就可以来思考如何建立网上商城了。

（三）相关数据结构

要构建网上商城，我们要考虑相关的数据结构。商城当中主要有三种类型的角色，第一是用户，第二是商店，第三是商品，笔者对三者分别设置了对应的数据结构。

在 Python 当中，字典是一种很合适的数据结构，因为它是由键及其对应的值组成，这样，不仅构造了对应关系，在寻找某个特定对象的时候也更加方便，因此，笔者主要使用字典来进行构造。

① 用户

对于商城中用户来说，主要有三个信息：用户地址，用户名，用户 ID，这三者之间需要建立映射关系。因此笔者设置如下：

```
userName = {} # addr : name
userID = {} # name : id
```

userName 是一个字典，键代表着用户地址，值代表着用户名。如果要在同一个客户端登录不同的用户，必须保证之前的用户已经下线。用户可以给自己起名字，服务器将地址和这个名字对应起来。

同样地，userID 也是一个字典，键代表用户名，值代表用户的 ID，这个 ID 是根据已注册的用户情况，由服务器给出的。例如，第一个注册的用户的 ID 是 00000，第十个用户的 ID 则是 00009。笔者在输出 ID 时使用了五位数字。若用户开店，店的 ID 和用户 ID 相同，因此这个 ID 是很有必要的。可以看到，用户 ID 是和用户名建立联系的，因为不论用户处于什么状态，其名字和 ID 都是对应的，而对于地址则未必都有这样的联系。

当然，还要通过 ID 找名字，通过名字找地址（在线的话），因此笔者写了两个辅助的函数便于调用：

```
def get_address(name):
    for addr in userName.keys():
        if userName[addr] == name:
            return addr
    return False

def get_name(id):
    for name in userID.keys():
        if userID[name] == id:
            return name
    return False
```

两个函数分别是 get_address()和 get_name()，这样，对于用户来说的三个信息之间的相互对应关系就建立起来了。其中 get_address()还能够用于判断用户是否在线，若它返回了 False，则说明在 userName 的键当中找不到对应用户名的地址，说明该用户已经登出了。

② 商店

商城支持用户开店并增加新的商品，同时查看商品及顾客等。因此，对于商店，主要有商店 ID，商店中顾客，商店中商品，店主这四个信息，对于商品，将在下文展开。

与商店有关的数据结构如下（商品除外）：

```
shopID = [] # shop_id == owner_id
in_Shops = {} # user_addr : shop_id(owner_id)
customers = {} # shop_id(owner_id) : customer_name
Goods = {} # shop_id(owner_id) : goods_id
```

shopID 存放了所有商店的 ID，它和店主的 ID 相同，给程序编写带来了便利。同时，可以直接从中判断谁开了店。

In_Shops 是一个字典，键是用户地址，值是商店 ID（亦即店主 ID），这样，就可以很方便的知道每位用户是否在店内，在哪家店内。若用户不在任何店内，设置其值为-1，包括服务器端的管理员用户‘admin’。

```
in_Shops['admin'] = -1
```

customers 也是一个字典，键是商店 ID，值是一个列表，存放着对应商店 ID 的顾客信息（用户名），这样，要实现店主或顾客查看店内所有顾客就很方便了。

Goods 同样是一个字典，键是商店 ID，值是一个列表，存放着对应商店 ID 的商品信息（商品 ID）。同样，店主或顾客就可以查看店内的商品了。

此外，我们可以通过一个 has_Shop()函数来判断一个用户是否有商店，这对于程序的编写是很重要的，因为如果一个用户是店主的话，ta 所能执行的操作和普通用户不同：

```
# determine if a user has shop
def has_Shop(owner_id):
    for id in shopID:
        if owner_id == id:
            return True
    return False
```

③ 商品

对于商品来说，主要的信息有商品 ID，商品名和商品价格三个，那么其实数据结构也很简单，使用两个字典，通过商品 ID 作为索引即可。

```
goodsID = [] # list of all the goods in the Mall
goodsName = {} # goods_id : name
goodsPrice = {} # goods_id : price
```

goodsID 存放着商城中所有出现的商品的 ID，而每个商店中有的商品则不尽相同。店主可以从商城中已有商品里添加，也可以添加新的商品，这新的商品会被加入商城的商品清单中。

goodsName 和 goodsPrice 就很简单了，键是商品 ID，值分别是商品名和商品价格。

笔者没有使用类，比如 user 类或 shop 类等，是觉得字典的方式更为直接和简单，但没有考虑开销的问题，就以一个简洁的方式来实现了。

现在，数据结构已经建立好，可以真正来构建网上商城，实现客户端和服务器的不同命令了。

（四）客户端

客户端的代码很少，因为处理请求是在服务器端，因此，客户端只需要实现收发数据即可。

首先，设定好服务器的地址：

```
serv_ip = '127.0.0.1'
serv_port = 6667
serv_addr = (serv_ip, serv_port)
```

然后，设置 Client 类，其中有初始化函数__init__()，包含了第（一）部分所提到的建立 UDP 协议下的 socket 连接，然后，还有上文提到的 recv_msg()和 send_msg()。

接着就是第（二）部分提到的内容，构造 clin_read()和 clin_send()函数，建立线程并启动，就可以向服务器发送数据了。

由于代码很短，贴出源代码如下：

```
import socket
import random
import sys
import threading
from time import sleep

serv_ip = '127.0.0.1'
serv_port = 6667
serv_addr = (serv_ip, serv_port)

class Client():
    def __init__(self):
        try:
            self.client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            client_ip = '127.0.0.1'
            client_port = random.randint(10000, 65535)
            client_addr = (client_ip, client_port)
            self.client.bind(client_addr)
            self.Serv_Addr = serv_addr
            print    'Client initialized!'
        except:
            print 'Initialization failed. Please try again.'
            sys.exit()

    def recv_msg(self):
        BUFSIZE = 2048
        msg, addr = self.client.recvfrom(BUFSIZE)
        print 'Received from ' + str(addr[0]) + ' : ' + msg
        return msg, addr

    def send_msg(self, msg):
        self.client.sendto(msg, self.Serv_Addr)
        print 'Message sent to server'

client = Client()

def clin_read():
    while True:
        client.recv_msg()
```

```
def clin_send():
    while True:
        msg = raw_input()
        if not msg:
            break
        client.send_msg(msg)

threads = []

if __name__ == '__main__':
    print """Hey!! Welcome to the E-Mall! You can use the following commands:

/login UserName: log in to the server with UserName. The server will give you an ID, and your shop(if there is one) shares the same ID.
/logout UserName: Log out. You can go back anytime and we reserve your Name and ID.
/shops: View all the shops in the E-Mall so that you can enjoy your shopping.
/enter ShopID: Enter a shop and view the goods.
/goods: View the goods in the shop. If you are a shop owner and not in others' shop, you can view the goods in your shop.
/customers: View the customers in the(your) shop.(similar to '/goods')
/buy goodsID: Buy one product in the shop.
/leave: Leave a shop.
/addgoods goodsID [goodsName] [goodsPrice]: Add new goods to your shop. If there has been any product in our E-Mall directory, just input the ID.

Hope you have a good time!
Copyright: CHEN Xinkai
"""
    send = threading.Thread(target= clin_send)
    threads.append(send)
    read = threading.Thread(target= clin_read)
    threads.append(read)

    for t in threads:
        t.start()
```

（五）服务器

服务器端的程序是商城的核心，它含有两个主要的函数 handle_client_command()和handle_server_command()，分别用来处理客户端请求和服务器请求。

数据结构及相关函数、线程建立过程已在上文介绍，就不再赘述。

1. handle_client_command(msg, addr)

首先看一下处理客户端命令的函数。它的参数分别是 msg 和 addr，即客户端发来的信息和客户端的地址。

使用 split 函数对用户发来的信息进行切分，每个空格后的词都进行提取，并存到 commd 中：

```
commd = msg.split(' ')
```

由于服务器是通过客户端发来的消息进行操作，因此这个切分十分重要。根据要求，commd 里面的第一个值 commd[0]是用户的命令，以“/”开头，这样就可以通过不同的命令作出不同的应答。

通过以下语句进入 if 判断语句：

```
if commd[0] in ['/login', '/logout', '/shops', '/enter', '/goods', '/customers', '/buy', '/leave', '/addgoods']
```

然后，就可以开始处理不同命令了。对于每个命令，都使用了 try 和 except 来捕获错误，在下文的代码中就不再展示。

① /login Username

这是登录操作。用户需要在输入“/login”之后再输入用户名，并以空格隔开。多输或少输都将不能登录。

代码如下：

```
# logged in or not?
if commd[1] in userID.keys() and get_address(commd[1]) != False and addr != get_address(commd[1]):
    self.send_msg('You have logged in somewhere else. Log out first!\n', addr)
else:
    if addr in userName.keys():
        self.send_msg(str(addr) + ', you have already logged in!\n', addr)
    else:
        userName[addr] = (commd[1])
        # one user name corresponds to one id
        if commd[1] not in userID.keys():
            # user id starts from 00000
            userID[commd[1]] = '%05d' % len(userID.keys())
        # make sure that this address in not in any shop
        in_Shops[addr] = -1
        self.send_msg(commd[1] + ' successfully logged in! Your ID is ' + userID[commd[1]] + '.\n',
                      addr)
        print commd[1] + ' logged in from ' + str(addr) + '\n'
```

首先，判断该用户是否在异地登录：commd[1]是输入的用户名，若它已经被注册，但所对应的地址不是当前客户端的地址 addr（在 get_address()不为 False 的情况下，即该用户名仍然在线），服务器就会告知客户端“You have logged in somewhere else. Log out first!”。本商城不允许异地登录。

然后，判断该地址是否已经在本地登录，通过对 userName 的 keys 进行寻找，若该地址存在，说明地址和名字存在着映射，说明这个用户名已经登录了，服务器就会告知客户端“you have already logged in!”。

若不存在上述情况，则允许进行登录。将 userName 中的’addr’键的值设为输入的值 commd[1]，即用户名。然后判断在 userID 中是否有该用户名，这是为了保证名字和 ID 是一一对应且唯一的。若 userID 的 key 中没有，则由服务器分配 ID，ID=已注册用户数（从 0 开始编号）因此有 :userID[commd[1]] = '%05d' % len(userID.keys())。这里做了一个简单的处理，就是让所有 ID 都是五位，不足的位数以 0 补齐。

然后设置 in_Shops 的值，上文中提到，当没有进店时，该值为-1，因此刚登录时将in_Shops[addr]设为-1。

最后，就是向用户发送消息了。同时，在服务器上也会显示登录情况。笔者没有做 GUI 界面，因此就只是简单的 print 出来了。

② /logout UserName

登出操作。用户可以暂时的离开商城，但其用户名及对应的 ID 不会被删去，而地址和用户名之间的映射则会被删除，因为在同一个客户端可能会有几个用户登录。

代码如下：

```
# determine if the user/address has logged in
if (addr not in userName.keys()) or (commd[1] not in userID.keys()):
    self.send_msg('Hello ' + commd[1] + ', you have not logged in.\n', addr)
else:
    if addr != get_address(commd[1]):
        self.send_msg('Please input your own name.\n', addr)
    else:
        # if the user is in a shop, kick him/her out!
        if in_Shops[addr] >= 0:
            shop_id = in_Shops[addr]
            if customers.has_key(shop_id):
                if len(customers[shop_id]) > 0:
                    customers[shop_id].remove(commd[1])
            # notify the owner
            owner_name = get_name(shop_id)
            owner_addr = get_address(owner_name)
            if owner_addr != False and owner_addr != addr:
                self.send_msg('User ' + userName[addr] + ' has left your shop.\n', owner_addr)
            del in_Shops[addr]
        self.send_msg('Dear ' + commd[1] + ', you logged out successfully!\n', addr)
        # delete the key 'addr' and its value(username) in userName
        del userName[addr]
        print commd[1] + ' logged out from ' + str(addr) + '\n'
```

首先判断输入的用户名是否已经登录或是否存在：若 userName 的 key 当中没有当前地址，说明该地址尚未登录；或者，若输入的用户名不在 userID 的 key 当中，说明该用户尚未注册，也就还没有登录。这种情况下，服务器会告知客户端“You have not logged in.”

然后，判断的用户名是否是自己的用户名，否则将会导致把别的用户下线的可能。判断的语句是 if addr != get_address(commd[1])，如果这样的话，服务器会告知客户端“Please input your own name.”。

经过上述判断，登出的操作被允许，开始登出的过程。如果 in_Shops[addr] >= 0，即用户在某家店中，则要获取这家店的 ID，将本地址从本店顾客中删去，并通过 shop_id 获取店主信息，通知 ta 本地址已经离店。其中，要判断店主是否在线，若不在线，即 get_address(owner_name)=False，就不会发给对应的店主了。最后，将 in_Shops[addr]这个键删去(del)。

在删除了进店信息后，还要删除地址和名字之间的映射，以免影响下一次登录。需要 del userName[addr]，然后服务器会显示该用户登出。

③ /shops

查看当前的商店情况。顾客通过这一命令可以得知谁开了商店，从而选择进入哪家店。若还没有人开店，系统会告知“There is currently no shop. You can create one.”。

代码如下：

```
if len(shopID) == 0:
    self.send_msg('There is currently no shop. You can create one.\n', addr)
else:
    for shop in shopID:
        # The 'Shops' dictionary stores shop IDs and the corresponding names of the owners
        name = get_name(shopID)
        self.send_msg('Shop: ' + shop + '  Owner name: ' + name + '  Owner ID: ' + userID[name],
                      addr)
```

在这里，主要看 shopID 里面的信息，若为空则说明没有店，若非空，则获取店主名字并和 ID 一起输出。

④ /enter ShopID

进入商店。用户需要明确商店的 ID 以进入一家商店。在查看店铺情况后，用户可以进店选购。

代码如下：

```
if (commd[1] in shopID) is False:
    self.send_msg('There is no Shop ' + commd[1] + '. Please try again.' + '\n', addr)
elif in_Shops[addr] != -1:
    if in_Shops[addr] == commd[1]:
      self.send_msg('You have already in Shop ' + commd[1], addr)
    else:
        self.send_msg('You cannot enter another shop before leaving one.\n', addr)
else:
    if customers.has_key(commd[1]) is False:
        customers[commd[1]] = []
    customers[commd[1]].append(userName[addr])
    self.send_msg('You have entered Shop ' + commd[1] + '\n', addr)
    in_Shops[addr] = commd[1]
    if Goods.has_key(commd[1]):
        for goods_id in Goods[commd[1]]:
            goods_name = goodsName[goods_id]
            goods_price = goodsPrice[goods_id]
            self.send_msg(
                'Product ' + goods_id + ': Name: ' + goods_name + ' Price: ' + goods_price + '\n',
                addr)
    else:
        self.send_msg('No goods in shop.\n', addr)
    owner_name = get_name(commd[1])
    owner_addr = get_address(owner_name)
    customer_name = userName[addr]
    if owner_addr != False:
        self.send_msg('User ' + customer_name + ' has entered your shop\n', owner_addr)
```

首先判断是否有这家店，若没有的话，服务器告知客户端“There is no shop xxx. Please try again”。然后，判断该地址对应用户是否已经在店里，若用户是重复输入 enter，则会显示“You have already in Shop xxx”；若想要进入别的店，则需要先离店，服务器会告知客户端“You cannot enter another shop before leaving one.”。

接下来是进店的操作。对这家店来说，多了一名顾客，若之前没有顾客，则要建立一个列表，即 customers[commd[1]] = []，然后才能加入当前用户。用户收到消息“You have entered Shop xxx.”。然后，存储用户的进店信息，将 in_Shops[addr]改为商店 ID。

进入商店时，还要显示商品，这时，判断 Goods 中是否有当前商店 ID 这个 key，即判断店内是否有商品，若没有，则返回“No goods in shop.”；若有，则在 Goods[commd[1]]中将商品信息发给用户，包括 ID、名字和价格，后两者可以通过之前设置的 goodsName 和 goodsPrice 得到。

进店还要通知店主，从输入的商店 ID 获得店主的名字和地址，在其在线的情况下告知

ta“xxx has entered your shop.”。

⑤ /leave

离开商店，用户无需指定商店 ID。

代码如下：

```
# see if the user is not in any shop
if in_Shops[addr] == -1:
    self.send_msg('You are not in any shop', addr)
else:
    shop_id = in_Shops[addr]
    user_name = userName[addr]
    if len(customers[shop_id]) == 1:
        del customers[shop_id]
    else:
        customers[shop_id].remove(user_name)
    in_Shops[addr] = -1
    # notify the owner
    owner_name = get_name(shop_id)
    owner_addr = get_address(owner_name)
    if owner_addr != False and owner_addr != addr:
        self.send_msg('User ' + user_name + ' left your shop.\n', owner_addr)
    self.send_msg('You have left Shop ' + shop_id + '.\n', addr)
```

首先判断用户是否在商店中，若不是，服务器将告知“You are not in any shop”。

然后，开始离店。将该用户从 customers[shop_id]中去除，将 in_Shops[addr]设置为-1，即未进店时的初始值。

最后，告知店主，“User xxx left your shop.”，告知该客户“You have left Shop xxx.”。

⑥ /goods

查看店内商品。

代码如下：

```
# if this user has shop and is not in other shops, return the info of his own shop
if in_Shops[addr] == -1 and has_Shop(userID[userName[addr]]):
    shop_id = userID[userName[addr]]
    if Goods.has_key(shop_id):
        for goods_id in Goods[shop_id]:
            goods_name = goodsName[goods_id]
            goods_price = goodsPrice[goods_id]
            self.send_msg(
                'Product ID: ' + goods_id + ', Name: ' + goods_name + ', Price: ' + goods_price + '\n
                addr)
    else:
        self.send_msg('No goods in your shop.\n', addr)
# return the info of the shop in which user is staying
elif in_Shops[addr] in shopID:
    shop_id = in_Shops[addr]
    if Goods.has_key(shop_id):
        for goods_id in Goods[shop_id]:
            goods_name = goodsName[goods_id]
            goods_price = goodsPrice[goods_id]
            self.send_msg(
                'Product ID: ' + goods_id + ' Name: ' + goods_name + ' Price: ' + goods_price + '\n',
                addr)
    else:
        self.send_msg('No goods in shop.\n', addr)
else:
    self.send_msg('This command is unavailable. Try other commands.\n', addr)
```

当用户不在任何店中，即 in_Shops[addr] == -1 时，该条命令的作用取决于用户的身份。若用户有开店，即 has_Shop()，系统会告知用户自己店里的商品，从 Goods[shop_id]中获取包括商品的 ID、名称以及价格的信息。当然，若店中无货，用户会被告知“No goods in your shop.”。

若用户没有开店，您将会被告知“This command is unavailable. Try other commands.”，即您不能使用这条指令。

当用户在某家店中的时候，可以随时查看店内商品，店主也可能会添加新商品（这会通知所有店内的顾客）。系统将告知商品的 ID、名称以及价格。若商店中没有商品，系统将告知“No goods in shop.”。

⑦ **/customers**

查看店内顾客。

代码如下：

```
# if this user has shop and is not in other shops, return the info of his own shop
if in_Shops[addr] == -1 and has_Shop(userID[userName[addr]]):
    shop_id = userID[userName[addr]]
    if customers.has_key(shop_id):
        self.send_msg('Current customers: ', addr)
        for customer in customers[shop_id]:
            self.send_msg(customer + '\n', addr)
    else:
        self.send_msg('No customer in shop.\n', addr)
# if the user in any shop..
elif in_Shops[addr] in shopID:
    shop_id = in_Shops[addr]
    # read the customer info into a list and then send together
    self.send_msg('Current customers: ', addr)
    cust = []
    for customer in customers[shop_id]:
        cust.append(customer)
    self.send_msg(cust + '\n', addr)
else:
    self.send_msg('This command is unavailable. Please try other commands.\n', addr)
```

和/goods 命令类似，当用户不在任何店中时，若是店主，则可以查看自己的店里的顾客情况，如果没有顾客，系统则会告知“No customer in shop.”；若不是店主，则无法使用这条指令，系统将会告知“This command is unavailable. Please try other commands.”。

当您在店中的时候，您可以随时查看同时在店中的顾客（包括您自己）。

购买店内商品。

代码如下：

```
# if the user is not in any shop, tell him/her
if in_Shops[addr] == -1:
    self.send_msg('You are not in any shop.\n', addr)
# if the user trying to buy something in his/her own shop, tell him/her
elif in_Shops[addr] == userID[userName[addr]]:
    self.send_msg('It is your own shop.\n', addr)
else:
    if len(commd) == 1 or commd[1] not in goodsID:
        self.send_msg('Invalid input.\n', addr)
    elif Goods.has_key(commd[1]) is False:
        self.send_msg('No good in shop.\n', addr)
    else:
        self.send_msg('You bought product ' + commd[1] + ' (Name: ' + goodsName[
            commd[1]] + ') successfully at the price: ' + goodsPrice[commd[1]] + '.\n', addr)
        shop_id = in_Shops[addr]
        # notify the owner
        owner_name = get_name(shop_id)
        owner_addr = get_address(owner_name)
        self.send_msg(
            userName[addr] + ' has bought product ' + commd[1] + ' (Product Name: ' + goodsName[
                commd[1]] + ').\n', owner_addr)
```

若用户不在任何店中（in_Shops[addr] == -1），自然无法买东西。若用户在自己的店里（in_Shops[addr] ==user_id），就无需买东西了。

若不是上述情况，就要判断店内是否有东西，没有的话告知“No good in shop”，如果有的话，则告知用户购买成功（这个商场没有货币这种东西。。），并通知店主。

⑨ **/addgoods GoodsID [GoodsName] [GoodsPrice]**

店主增加商品。

代码如下：

```
if has_Shop(userID[userName[addr]]) == False:
    self.send_msg('You do not even own a shop!\n', addr)
else:
    owner_name = userName[addr]
    shop_id = userID[owner_name]
    # the value of "Goods" is a list
    if Goods.has_key(shop_id) is False:
        Goods[shop_id] = []
    if commd[1] not in Goods[shop_id]:
        Goods[shop_id].append(commd[1])
        if commd[1] not in goodsID:
            goodsID.append(commd[1])
            goodsName[commd[1]] = commd[2]
            goodsPrice[commd[1]] = commd[3]
            print 'New product added to directory. Product ID: ' + commd[1] + ' Name: ' + commd[
                2] + ' Price: ' + commd[3] + '\n'
            # if new goods added, notify all the other shop owners
            for ownerid in shopID:
                ownername = get_name(ownerid)
                owneraddr = get_address(ownername)
                if owneraddr != False and owneraddr != addr:
```

```
                    self.send_msg(
                        'New product added to directory. Product ID: ' + commd[1] + ' Name: ' +
                        commd[
                            2] + ' Price: ' + commd[3] + '\n', owneraddr)
                self.send_msg('New product added to shop. Product ID: ' + commd[1] + ' Name: ' + commd[
                    2] + ' Price: ' + commd[3] + '\n', addr)
            else:
                self.send_msg(
                    'This product has been registered. The name and price is automatically attached.\n',
                    addr)
                self.send_msg(
                    'New product added to shop. Product ID: ' + commd[1] + ' Name: ' + goodsName[
                        commd[1]] + ' Price: ' + goodsPrice[commd[1]] + '\n', addr)
                # notify all the customers in shop
            if customers.has_key(shop_id):
                for customer in customers[shop_id]:
                    customer_addr = get_address(customer)
                    self.send_msg(
                        'New product added to shop. Product ID: ' + commd[1] + ', name: ' + goodsName[
                            commd[1]] + ', price: ' + goodsPrice[commd[1]], customer_addr)
        else:
            self.send_msg('You have added this one!\n', addr)
```

代码较长，首先判断用户是否有商店，若没有的话则告知“You do not even own a shop!”，然后，获得店主信息和商店信息。

判断店中是否有商品，若没有的话要创建一个列表（Goods[shop_id] = []）。将商品 ID 加入其中。如果店主增加了一个新的商品（if commd[1] not in goodsID），将它加入商城的商品清单 goodsID 中，并保存其名字和价格。同时，要通知所有的店主“New product added to directory.”。

而店主其实可以只输入商城商品清单中已有商品的 ID 即可增加商品。本商城要求商品的唯一性，包括价格的唯一性，因此名字和价格会自动匹配。系统会告知店主“This product has been registered. The name and price is automatically attached.”。

增加新商品后，店主会收到“New product added to shop.”，而在店内的每个顾客（通过 customers[shop_id]找到）都会收到上新通知。

至此，客户的命令已经处理完毕。

2. handle_server_command(msg)

和客户请求一样，msg 也需要 split，才能进行处理。

① /msg [UserID]

向用户发消息。

代码如下：

```
# if only input '/msg', broadcast to all the users
if len(commd) == 1:
    print 'Please input your message: '
    note = raw_input()
    self.broadcast(note, userName.keys())
else:
    print 'Please input your message: '
    note = raw_input()
    # admin can input more than one ID
    for user_id in commd[1:]:
        # leave out invalid IDs
        if user_id not in userID.values():
            print user_id + ' is an invalid id.\n'
            continue
        user_name = get_name(user_id)
        user_addr = get_address(user_name)
        # leave out users who are offline
        if user_addr not in userName.keys():
            print user_id + ' has logged out.\n'
            continue
        self.send_msg('[Server]: ' + note + '\n', user_addr)
```

管理员有两种选择，第一是不输入所要发送的用户 ID，即开始广播，这样服务器会对所有在线的用户进行广播。可以看到，这样用了一个 broadcast 函数：

```
def broadcast(self, msg, client_addrs):
    if len(client_addrs):
        print 'Broadcasting...\n'
        for client_addr in client_addrs:
            # leave out the users that are offline
            if client_addr not in userName.keys():
                print client_addr + ' has logged out.\n'
                continue
            self.send_msg('[Server]: ' + msg + '\n', client_addr)
        print 'Broadcast Ending...\n'
    else:
        print 'No user. Broadcast Failed.\n'
```

即对 userName.keys()中的地址进行广播，不在其中的则已经下线了。

如果管理员指定了用户 ID，则获取他们的地址后进行消息的发送。这里有两个判断，第一个是剔除无效的 ID（userID.values()中是否存在），另一个是剔除下线的用户（userName.keys()中是否存在）。

若输入多个用户 ID，用空格隔开即可，程序中使用了 for 循环。

② /opennewshop UserID

开一家新的店。管理员需要输入店的 ID。

代码如下：

```
if len(commd) == 1:
    print 'Please input a shop id!\n'
else:
    if commd[1] in shopID:
        print 'This shop already existed!\n'
    else:
        if commd[1] not in userID.values():
            print 'This user have not registered.\n'
        else:
            # admin can input more than one ID
            for user_id in commd[1:]:
                user_name = get_name(user_id)
                if user_name not in userName.values():
                    print user_id + ' has logged out. \n'
                else:
                    shopID.append(user_id)
                    user_addr = get_address(user_name)
                    self.send_msg(
                        ' ' + user_name + ', your shop opens successfully! Add some goods first!\n'
                        user_addr)
                    print 'New shop opened. Shop ID: ' + user_id + '\n'
                    self.broadcast(
                        'New shop opened!! Shop ID: ' + user_id + '. Go and have a look!\n',
                        userName.keys())
```

首先，判断店是否存在（if commd[1] in shopID），或对应的用户是否存在（if commd[1] not in userID.values()）。然后开店，在 shopID 中增加一个新的 ID，获得用户的地址并告知“your shop opens successfully! Add some goods first!”，此外，每个在线的用户也会收到新开店的通知。

③ /enter ShopID

进入一家商店。

代码如下：

```
if len(commd) == 1:
    print 'You should input a shop name!\n'
else:
    # if admin is already in a shop
    if in_Shops['admin'] != -1:
        print 'You cannot enter another shop.\n'
    else:
        if (commd[1] in shopID) is False:
            print 'Invalid shop name. Please try again.\n'
        else:
            print 'You have entered Shop ' + commd[1] + '\n'
            in_Shops['admin'] = commd[1]
```

管理员进入商店不会被加到顾客名单中，也不会通知店主。当然，管理员不能同时进入两家店，因此要判断 if in_Shops['admin'] != -1。

进店后，服务器端会显示“You have entered Shop xxx.”，然后 admin 的进店情况进行更新，in_Shops['admin']改为现在的商店 ID。

④ /leave

离开商店。

代码如下：

```
if in_Shops['admin'] == -1:
    print 'You are not in any shop!\n'
else:
    shop_id = in_Shops['admin']
    in_Shops['admin'] = -1
    print 'You have left Shop ' + shop_id + '\n'
```

若 in_Shops['admin'] == -1，说明管理员不在任何店中，不可使用此操作；若管理员在某家店中，将 in_Shops['admin']修改为初始值-1 即可。

⑤ /goods

查看商品。

代码如下：

```
# if admin not in any shop, show the E-Mall Directory
if in_Shops['admin'] == -1:
    print 'You are not in any shop. Goods in the E-Mall Directory:\n'
    if len(goodsID) == 0:
        print 'Nothing.\n'
    else:
        for id in goodsID:
            print 'Goods id: ' + id + ' goods name: ' + goodsName[id] + ' goods price: ' + \
                  goodsPrice[id] + '\n'
# if admin is in a shop, show the goods in shop
else:
    shop_id = in_Shops['admin']
    if Goods.has_key(shop_id):
        product_IDs = Goods[shop_id]
        for product_id in product_IDs:
            print 'Goods id: ' + product_id + ' goods name: ' + goodsName[
                product_id] + ' goods price: ' + goodsPrice[product_id] + '\n'
    else:
        print 'No goods in shop.\n'
```

若管理员用户不在任何店内（in_Shops['admin'] == -1），则显示商城商品清单中的所有商品。判断 goodsID 是否为空，非空的话就输出其中的商品。

若管理员用户进入了某家店，则显示该店的商品的 ID、商品名和商品价格。若没有商品，则显示“No goods in shop.”

⑥ /customers

查看店内的顾客。

代码如下：

```
if in_Shops['admin'] == -1:
    print 'You are not in any shop.\n'
else:
    shop_id = in_Shops['admin']
    if customers.has_key(shop_id):
        for customer in customers[shop_id]:
            print 'Name: ' + customer + ' id: ' + userID[customer]
        print '\n'
    else:
        print 'No customer in this shop (except you, of course).\n'
```

若管理员不在任何店内（in_Shops['admin'] == -1），服务器会显示“You are not in any shop.”。若所在的店中无顾客，服务器会显示“No customer in this shop (except you, of

course).”。否则，服务器会显示顾客的名字和 ID。

⑦ /shops

查看商城中的商店。

代码如下：

```
if len(shopID) > 0:
    for shop in shopID:
        print 'Shop id:' + shop + ' Owner:' + get_name(shop)
else:
    print 'There is no shop at all.\n'
```

若没有任何人开店（len(shopID) == 0），则会显示“There is no shop at all.”。否则，服务器将会显示商店名和店主的名字。

⑧ /users

查看商城中的用户。

代码如下：

```
if userName:
    for name in userName.values():
        user_id = userID[name]
        if has_Shop(user_id):
            print 'Name: *' + name + ' ID: ' + user_id
        else:
            print 'Name: ' + name + ' ID: ' + user_id
    print '\n'
else:
    print 'There is no user at all.\n'
```

此命令将会显示所有用户的名字和 ID，若某个用户是店主，则名字前会有‘*’来标识。若还没有用户注册，服务器将显示“There is no user at all.”。

判断 userName 是否为空即可判断是否有用户。然后分两种情况，对于店主（has_Shop(user_id)），会在名字前多一个“*”。

⑨ /closeshop ShopID

关闭商店。

代码如下：

```
if len(commd) == 1:
    print 'You should specify a shop id.\n'
else:
    if (commd[1] in shopID) is False:
        print 'No such shop! Try again.\n'
    else:
        # notify the owner
        owner_name = get_name(commd[1])
        owner_addr = get_address(owner_name)
        if owner_addr != False:
            self.send_msg('[Server]: Your shop is being closed\n', owner_addr)
        # remove all the customers(if any)
        if customers.has_key(commd[1]):
            for customer in customers[commd[1]]:
                cust_addr = get_address(customer)
                self.send_msg(
                    '[Server]: Sorry, the shop in which you are shopping has been closed\n',
                    cust_addr)
                in_Shops[cust_addr] = -1
            del customers[commd[1]]
        # remove all the goods(if any)
        if Goods.has_key(commd[1]):
            del Goods[commd[1]]
        # remove this shop
        shopID.remove(commd[1])
```

此命令将会关闭某个商店，先判断是否有这家店。

然后，获取店主的信息，并通知店主“Your shop is being closed”。然后，若店中有顾客，通知店内的顾客“Sorry, the shop in which you are shopping has been closed”。对每个顾客，将 in_Shops 的值重置为-1，然后将这家店的顾客信息 customers[commd[1]]删除。

店中的商品信息 Goods[commd[1]]也将被清空。最后，shopID 中把这家店删去。

至此，对于所有命令的处理已经结束，本程序终于实现。

三、小结

这次网上商城的编写是一次有成就感也有趣的过程，也贯穿着发现问题、解决问题的乐趣。刚开始不知道如何搭建框架，后来在设计数据结构时思路比较混乱，最后又一直 debug，考虑各种情况，终究是实现了。出于时间关系，笔者没有做 GUI，有机会将会尝试做一下。总而言之，这一次做 PJ 的过程令笔者受益匪浅。