网络商城系统 设计文档

王傲

15300240004

数据通信与计算机网络

COMP130017.01

指导教师：肖晓春

目录：

内容简介：

本次实验实现了基于 UDP 的网络商城系统，同时给出了客户端和服务器端的源代码，基于自主设计的协议的基础上实现了要求的功能，并有所创新，能够在安全的基础上很好的完成网络购物功能。本文是网络商城系统的设计文档。

关键字：网络商城，UDP，图形界面，PyQt，DES 加密，时间戳，多线程

1. 基本简介

1.1 实现功能

本次完成的网络商城，实现了基于 DES 加密、使用时间戳防止重放攻击的安全 UDP 通信；对于用户而言，实现了登陆、用户头像、查看商城中商店、查看一段时间内的消息、查看余额、充值、查看商店中商品、购物、查看当前店内其他顾客等功能；对于拥有商店的用户，还实现了为自己的商店登记新商品、查看自己店内当前顾客、查看自己店内当前剩余商品等功能；对于管理员，实现了群发或单独发送消息、查看商城内商店信息、商城用户信息、为已注册用户开店、关闭现有商店、查询用户余额、注册新用户功能，其中注册新用户时实现了拖拽式上传头像图片的功能。除此之外，当用户进入某家商店、在某家商店购物、离开某家商店时，系统均会向店主发送消息；店家登记新商品后，系统也会通知当前店内的顾客；管理员关闭某家商店后，系统也会通知当前店内的顾客。

同时，为了实现监听服务器发送的消息，每当有新的消息就弹出弹窗提示用户，在客户端实现了两个线程，子线程用定时器定时向服务器索取最新消息，有的话则用弹窗提示；在服务器端，为了防止上传、下载头像占用主 socket 时间过长，同样实现了多线程，用新的 socket 或者线程来处理头像，减轻 I/O 压力。

1.2 实验平台

本次 PJ 在 MacBook Pro Mid 15’上完成，操作系统为 macOS High Sierra 10.13.2,使用 Python 2.7.14,主要使用的库有：

- PyQt5 5.9.2 及相关依赖库: 主要用于实现图形界面和相关操作功能（利用 brew 安装）
- pyDes 2.0.1: 实现通讯过程中的 DES 加密算法，避免明文传输
- socket: 用于实现客户端和服务器端的基于 UDP 的通信

- json: 协议基于 JSON 格式的字符串
- sys, os: 使用某些系统调用
- threading: 用于在服务器端实现多线程，处理头像传送或接收

使用 UDP socket 进行通信，端口号为 65432，头像端口号为 65431。传输信息和协议基于 JSON 字符串实现，通过字典的键和键值传递信息。由于 PJ 要求不允许使用数据库，因此主要的数据存储在 txt 文件中，一部分需要实时变化的数据存储在内存中。

本次 PJ 主要基于 PyQt 实现，其中最为重要的机制是信号 signal 和槽函数的连接，使用 connect 函数连接，如 self.LoginButton.clicked.connect(self.send_message) 语句，当按钮 LoginButton 发出被点击信号 clicked 时，槽函数 send_message 被调用，使用 socket 发送信息。

对于弹窗机制，使用 PyQt 提供的 QThread 实现多线程而不使用 Python 原生的 threading，可以使用信号和槽来通信，更加方便；对于接收头像机制，使用 Python 原生的 threading，不需要信号和槽，实现简洁。

特别说明，在 mac 上使用时必须先使用指令

```
sudo sysctl -w net.inet.udp.maxdgram=65535
```

更改 UDP 缓冲区的大小，不然登陆后用户的头像会因为缓冲区不足而无法接收。

1.3 相关文件

本次 PJ 的文件主要分为两个部分：Client 和 Server。

Client 文件夹中：

- main.py: 主文件，通过 connect 函数将不同对象的信号和槽函数连接起来，使其形成一个可以互相沟通的整体。使用时直接在 Client 文件夹中运行 main.py 即可使用客户端功能。
- IP.py: 定义了网络连接的一些具体参数和通用函数，包括 IP 地址、端口号、DES 加密函数、DES 解密函数、DES 密钥、获取时间戳函数、通用的错误提示窗口、消息弹窗等。其余的每个源文件均需要包含 IP.py。
- Login.py: 登陆窗口的实现，包括清空窗口、发送登录数据等功能。

- Userinterface.py: 用户界面的实现，提供了查看用户姓名和头像、查看商场内商店、查看某一段时间内的消息、查看余额并充值的功能，并提供了进入选中的商店和自己的商店管理界面的入口。同时，在文件中实现了多线程收信。
- GoodsList.py: 展示进入的商店的商品栏的窗口，并提供了点击购买的功能，每点击一次购买一件，并将余额扣除相应的数额。如果商品购买完毕，便会下架，无法继续购买；如果余额不足，也会无法继续购买。此外，还提供了顾客窗口，可以查询当前在同一家店内的顾客情况。
- MyShop.py: 用户自己商店的管理界面，提供登记新商品、查看自己店内顾客情况、查看自己店内商品情况的功能。
- Admininterface.py: 管理员界面，提供查看管理员姓名和头像、单发和群发消息、查看商店信息、查看用户信息、为已注册用户开店、关闭商店、查看用户余额的功能，并提供了进入注册新用户界面的入口。
- Admin_GoodsList.py: 管理员使用的商品列表界面，与 GoodsList.py 中内容和功能类似，但不提供点击购买功能。
- NewUser.py: 注册新用户界面，输入用户名和密码来注册，并提供拖拽式上传用户头像的功能。
- black_flat.qss: GUI 界面渲染文件。
- tmp: 用于暂时存储和加载服务器传来的用户头像，同时里面有图像 default.jpg，当注册新用户未拖拽上传头像时，以 default.jpg 作为默认头像上传。
- icon: 存储用于按钮、列表上的辅助图像。

Server 文件夹中：

- server.py: 实现 server 功能的最主要的文件，根据服务器的请求回复相应数据，实现相应功能。在终端中运行可直接使用服务器端功能。
- info/user.txt: 存储用户数据，包含用户 ID、密码、姓名、头像图片位置等信息（为了安全，可以使用加密存储，这里为了方便，使用明文存储）。
- info/admin.txt: 存储管理员数据，内容与 user.txt 类似。
- info/shops.txt: 存储商店信息，包括商店 ID、商店名、店主姓名。

- info/goods.txt: 存储商品信息，包括商品 ID、商品名、所属商店 ID、售价。
- info/messages.txt: 存储已经发送的消息，每条包括日期、时间、发送方 ID、接收方 ID、内容。
- image: 存储用户和管理员的头像。

除此之外，还使用了四个字典结构：good_num、user_in_shop、user_remain、user_address，其中 good_num 是商品剩余数量，user_in_shop 记录当前用户所在商店（不在商店中的为“none”），user_remain 是用户的余额，user_address 存储用户的地址，用于给子线程返回消息。这四者均会因为用户的操作而比较频繁的变化，因此放在内存中。

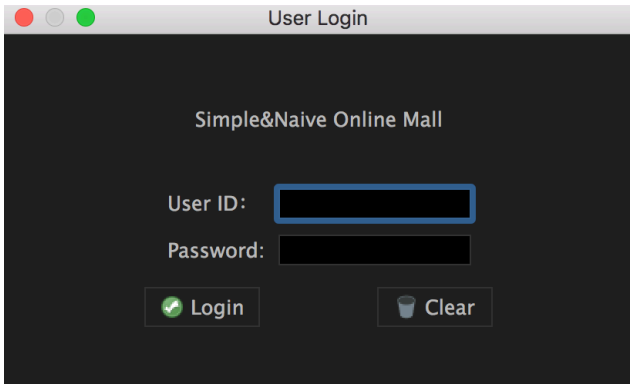
2. 流程实现

这里通过不同的模块来分述相关功能。

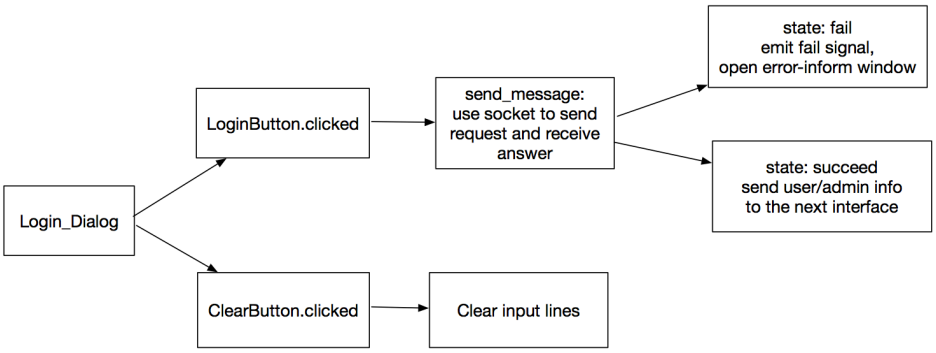
2.1 客户端

2.1.1 登陆模块

登陆窗口下图所示：



登陆流程如下图所示：

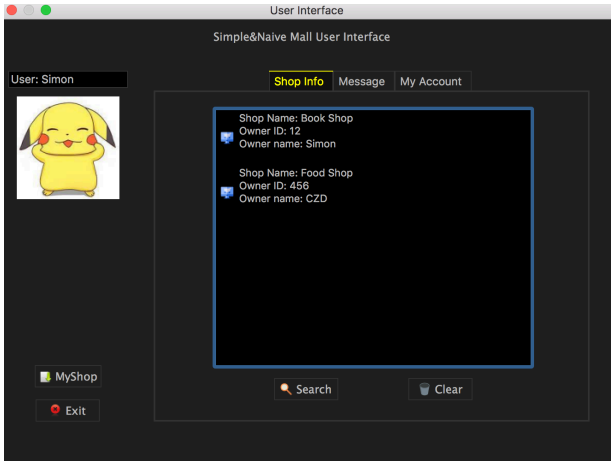


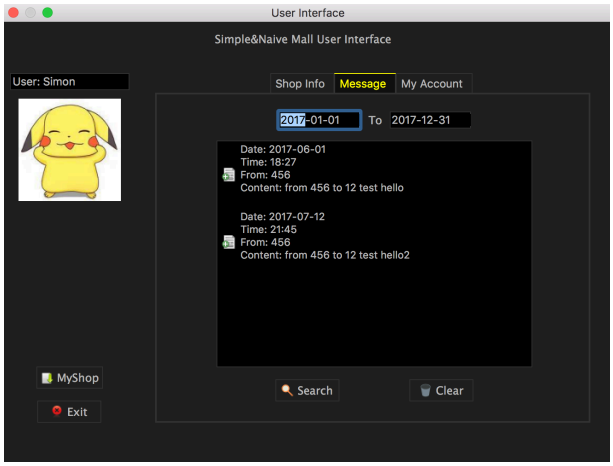
登陆模块支持使用 ID 或者姓名登陆。

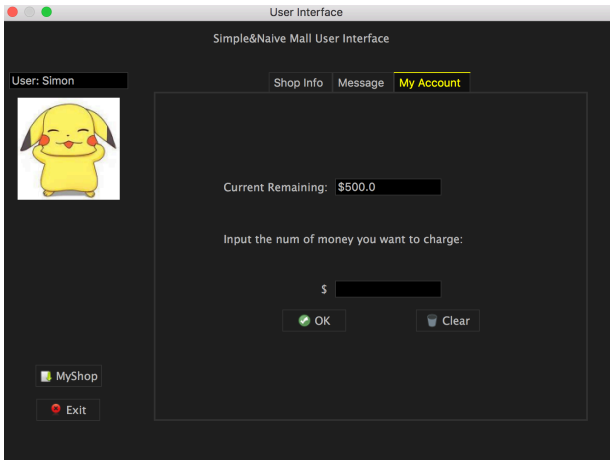
当点击 ClearButton 时，调用 clear 函数，ID 和密码输入行会被清空；当点击 LoginButton 时，调用 send_message 函数，设置“request”字段为“login”表示请求登陆，同时加上 ID 和密码字段，加上时间戳，加密后发送给 server 端；接收到 server 端响应后，如果“state”字段为“fail”，证明用户名/ID 或密码错误，拒绝登陆，调用 Error_Dialog 提醒用户登陆失败；如果“state”字段为“succeed”，表明登陆成功，server 还会发送登陆者的姓名和身份级别（用户/管理员），登陆窗口根据不同用户级别发送相应的 signal 打开使用界面，同时利用 connect 函数传递相应身份信息。

2.1.2 用户界面模块

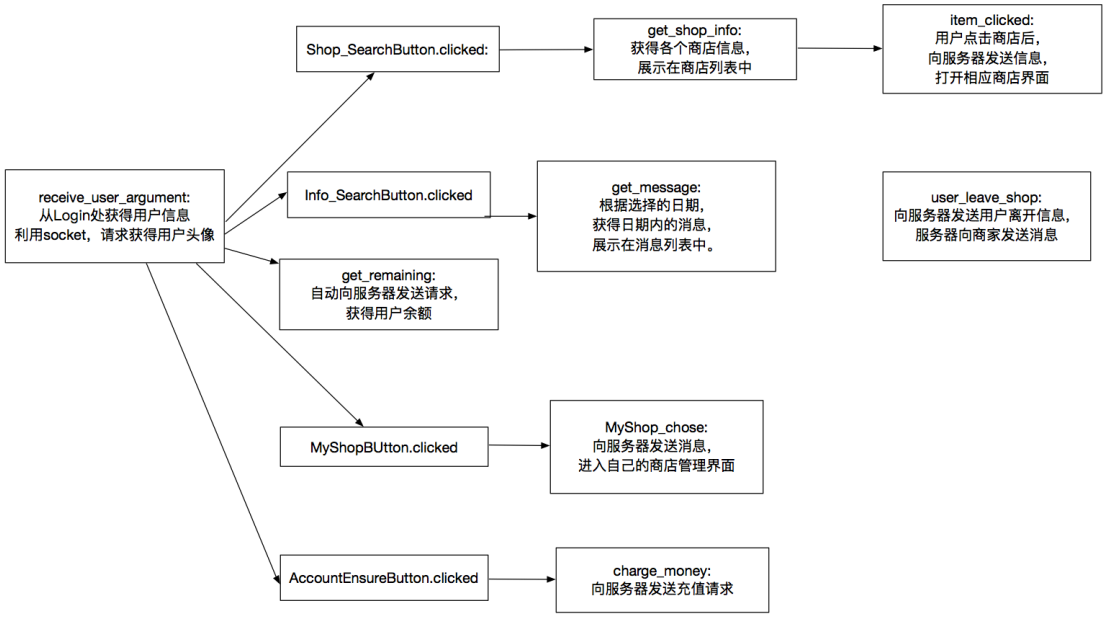
用户界面样式如下：







用户界面模块流程图如下（clear/empty 机制基本十分类似，下面不再给出）：



当用户登陆成功后，登录窗口发送 login_succeed_user 信号，说明是用户登陆，receive_user_argument 函数接收，打开用户界面 Userinterface_Dialog，同时接收用户 ID、用户名信息，然后用户界面依据接收的 ID 向服务器请求用户头像，成功获取后于用户名一起显示在窗口左上角。

点击获取商店按钮后，调用 get_shop_info 函数，发送“shopinfo”请求，获得商店信息，然后逐条的加载进商店列表，显示出商店名、商家 ID、商家姓名。

点击获取的任意一个商店，发送 item_clicked 信号和相关参数，打开相应的商店界面。同时服务器会向商家发送消息，记录顾客进入商店。

设定好时间区间后，点击获取信息按钮，调用 get_message 函数，发送“messageinfo”请求，获得相应时间区间内该用户收到的消息。

系统会自动调用 get_remaining 函数，获取并显示用户的余额。每当用户余额发生变化（充值/购物）时，均会自动调用此函数。

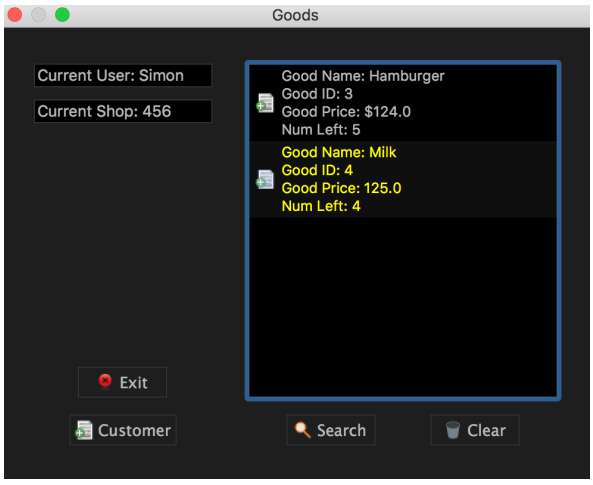
点击我的商店按钮，调用 MyShop_chose 函数，进入我的商店界面。

输入充值金额后，点击 AccountEnsureButton 按钮，调用 charge_money 函数，向服务器发送“charge_money”请求，对账户进行充值。根据服务器返回的状态：成功或失败（如有非法字符等），显示正确的余额。

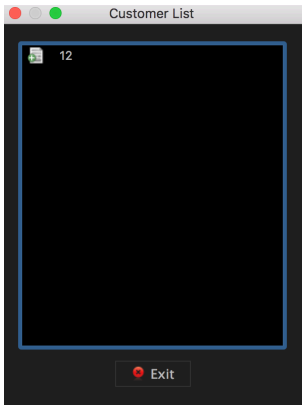
特别的，用户界面收到用户离开商店的信号（不是离线的信号）后，会调用 user_leave_shop 函数，向服务器发送消息，服务器向商家发送消息，记录顾客离开。

2.1.3 商店界面模块

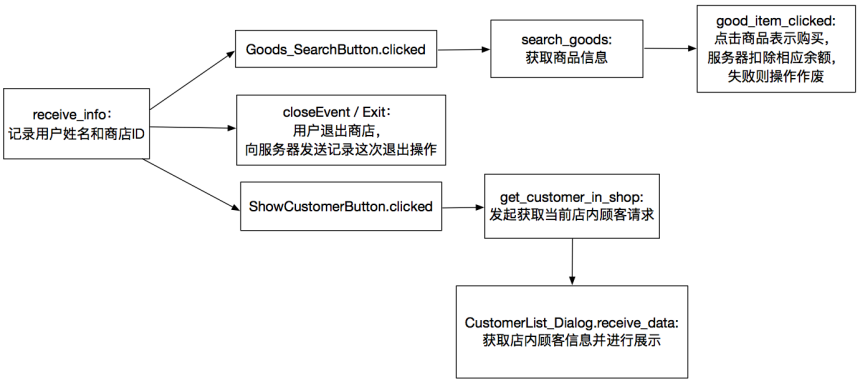
界面样式如下：



查询店内顾客界面（显示用户 ID）：



商店界面流程如下：



当用户在用户界面点击列表中的某个商店后，发送 shop_chose 信号，GoodsList_Dialog 调用 receive_info 函数接收，获得用户姓名和商店 ID（店主 ID）。

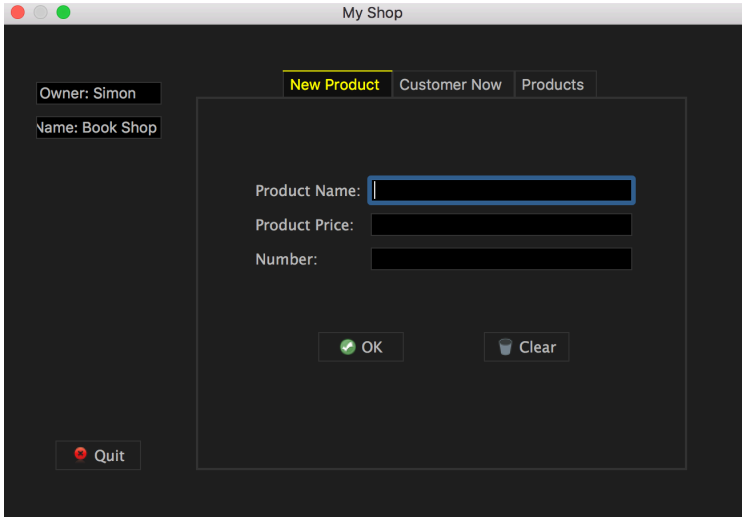
点击查询商品列表按钮，调用 search_goods 函数，向服务器发送“getgoods”请求，获得店内商品详细信息。点击某件商品，调用 good_item_clicked 函数，向服务器发送“buygood”请求，购买商品。服务器首先会判断余额是否足够，然后判断剩余商品数量是否不为 0，决定是否扣除相应费用，最后返回此次购买的状态；客户端接收到服务器返回的状态后，正确的更新用户余额，完成购买操作；如果余额足够而商品数量不够的话，商品便不会再显示。

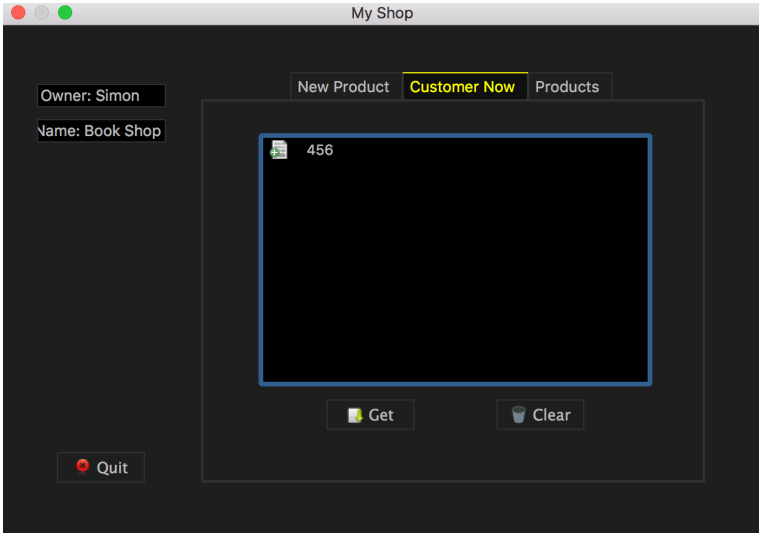
用户点击 ExitButton，或者点击红叉关闭窗口触发 closeEvent 的话，均会发送 exit_shop 信号，用户界面接收到信号后向服务器发送用户离开请求，服务器向商家发送顾客离开消息。

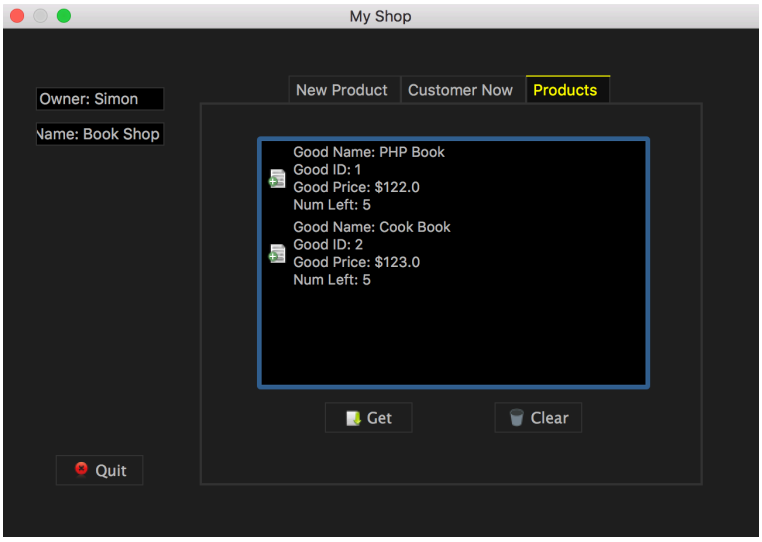
用户点击 ShowCustomerButton 按钮，调用 get_customer_in_shop 函数，打开 CustomerList_Dialog 窗口，同时调用 receive_data 函数获取当前店内顾客 ID，展示在顾客列表中。

2.1.4 我的商店模块

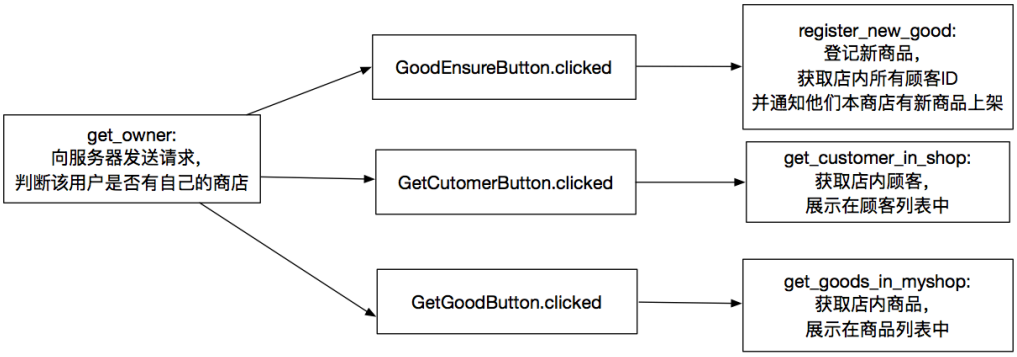
我的商店管理界面如图：







我的商店流程如下：



当用户在用户界面点击我的商店按钮后，发送 myshop 信号，MyShop_Dialog 界面调用 get_owner 函数接收，获取用户 ID 和用户名，然后向服务器发送“if_shop_exists”请求，查询用户是否拥有商店；如果存在，就能获得商店名并打开界面，否则发送出错信号。

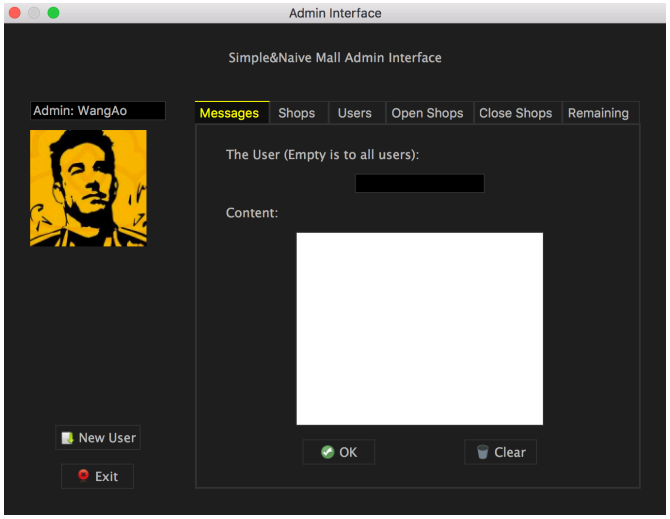
用户输入商品名、商品价格、商品数量后，点击确认按钮，调用 register_new_good 函数，向服务器发送“register_new_good”请求，登记新商品；成功后，发送“get_customer_in_shop”请求，获得店内顾客 ID，然后发送“inform_new_good”请求，告知服务器通知相应顾客，服务器向相应店内顾客发送新商品上架消息。

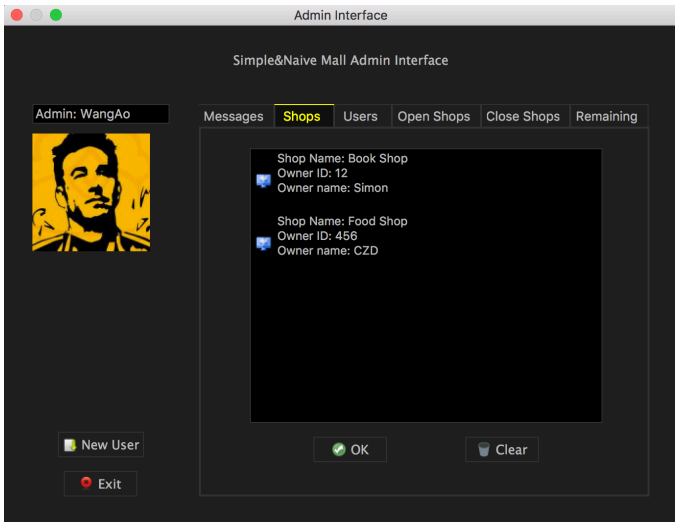
用户点击查询顾客按钮，调用 get_customer_in_shop 函数，向服务器发送“get_customer_in_shop”请求，获得店内顾客 ID 并展示在顾客列表中。

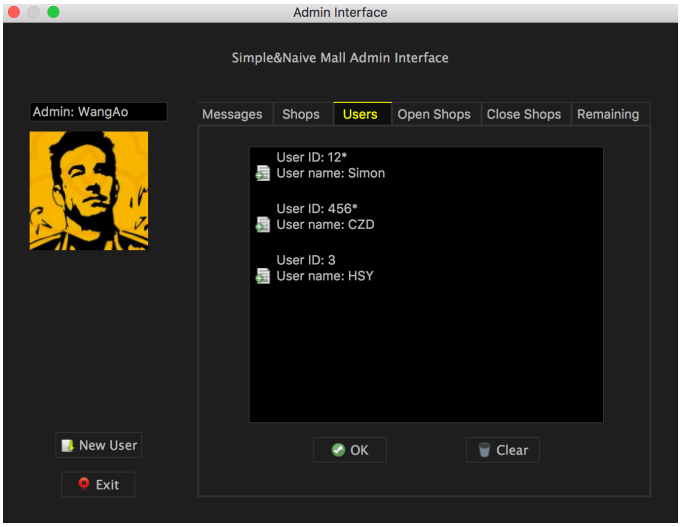
用户点击查询商品按钮，调用 get_goods_in_myshop 函数，向服务器发送“getgoods”请求，获取自己店内商品的信息，数目为 0 的不会获得，最后展示在商品列表中。

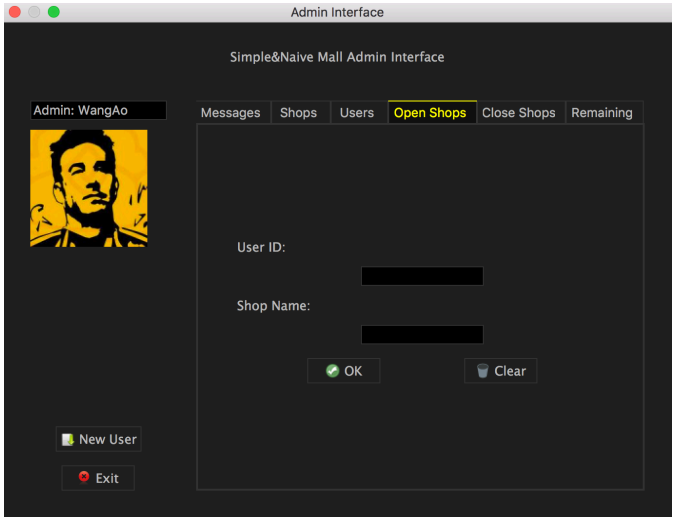
2.1.5 管理员界面模块

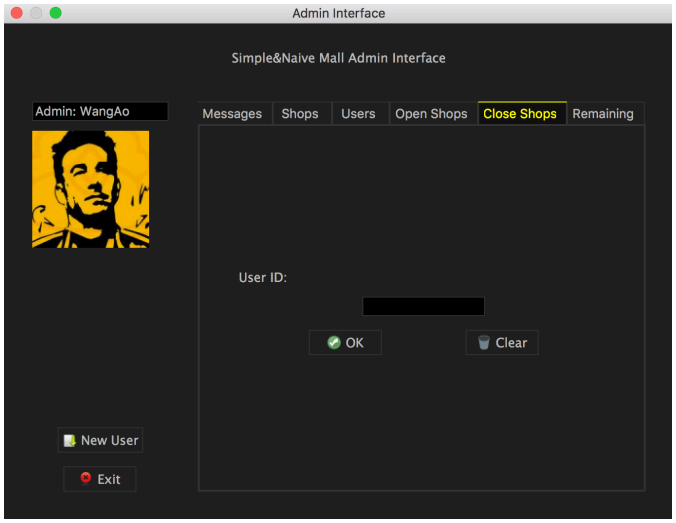
管理员界面如图：

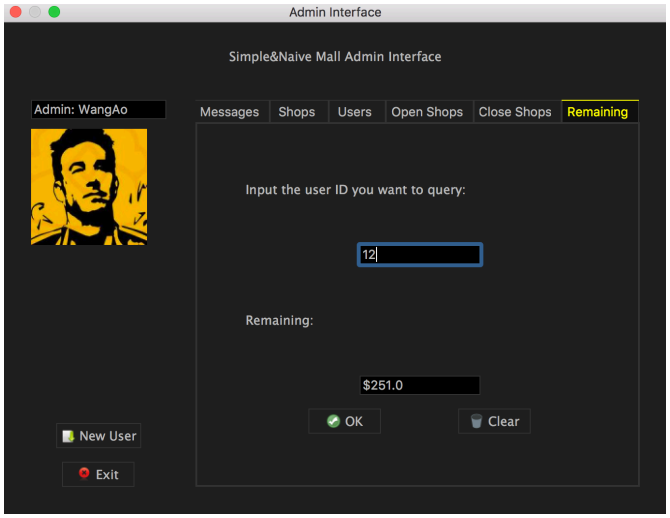




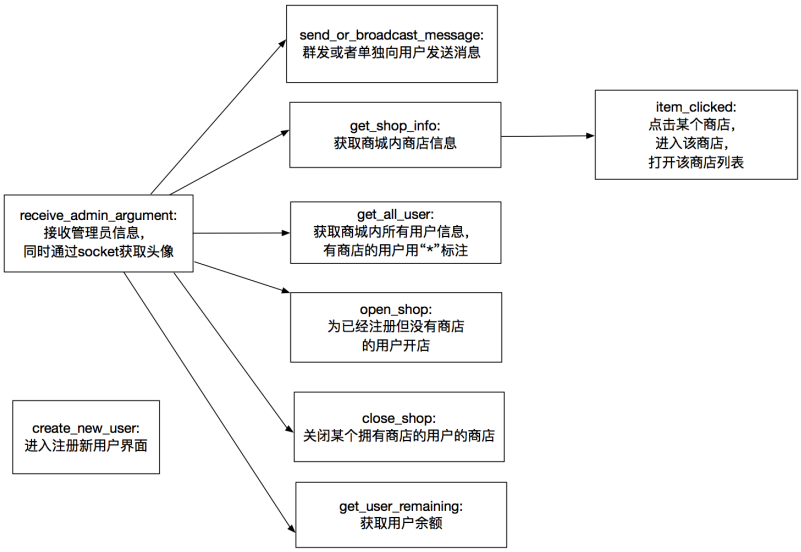

Admin Interface
Simple&Naive Mall Admin Interface
Admin: WangAo
Messages
Shops
Users
Open Shops
Close Shops
Remaining
User ID: 12*
User name: Simon
User ID: 456*
User name: CZD
User ID: 3
User name: HSY
New User
Exit
OK
Clear


Admin Interface
Simple&Naive Mall Admin Interface
Admin: WangAo
Messages
Shops
Users
Open Shops
Close Shops
Remaining
User ID:
Shop Name:
OK
Clear
New User
Exit


Admin Interface
Simple&Naive Mall Admin Interface
Admin: WangAo
Messages
Shops
Users
Open Shops
Close Shops
Remaining
User ID:
OK
Clear
New User
Exit



管理员界面流程如下：



管理员填写要发送的用户的 ID（为空时为群发）和要发送的消息内容，点击确认按钮，调用 send_or_broadcast_message 函数，向服务器发送“inform”或者“broadcast”请求，单发或者群发消息。

管理员点击查询商店按钮，调用 get_shop_info 函数，发送“shopinfo”请求，接收商店信息。点击列表中的某家商店，可以进入该商店，查询店内商品和店内顾客，操作与之前类似。

管理员点击查询用户按钮，调用 get_all_user 函数，发送“get_all_user”请求，获取商城内所有用户信息；其中，拥有商店的用户以“*”标识。

管理员输入用户名和商店名后，点击确认按钮，调用 open_shop 函数，发送“open_shop”请求，服务器检查用户是否存在并且是否拥有商店，符合条件则开通商店并返回状态信息。

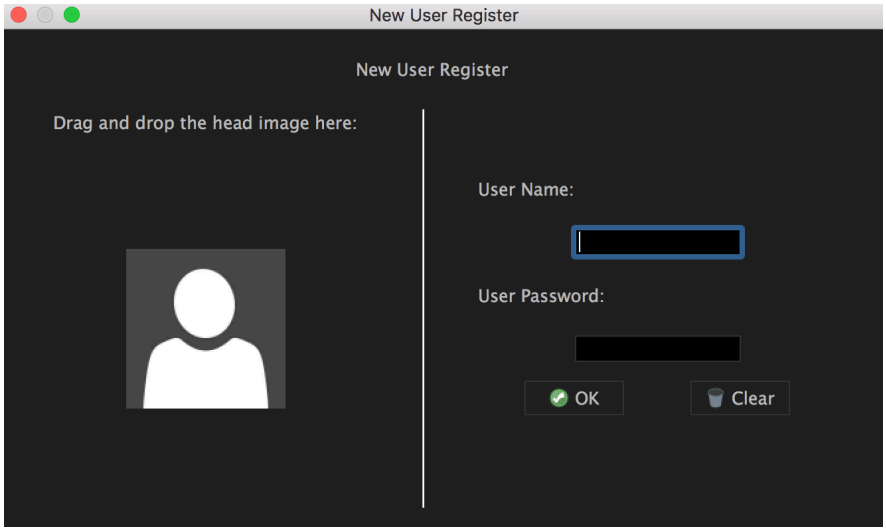
管理员输入用户 ID 后，点击确认按钮，调用 close_shop 函数，发送“close_shop”请求，服务器检查商店是否存在，成功则删除商店信息并返回状态信息。

管理员输入要查询的用户 ID 后，点击查询按钮，调用 get_user_remaining 函数，发送“get_user_remaining”请求，获得用户余额并显示出来。

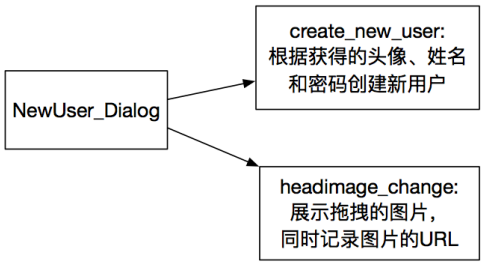
此外，管理员点击注册新用户按钮后，会调用 create_new_user 函数，打开注册新用户界面。

2.1.6 注册新用户模块

注册新用户界面如图：



注册新用户流程如下：



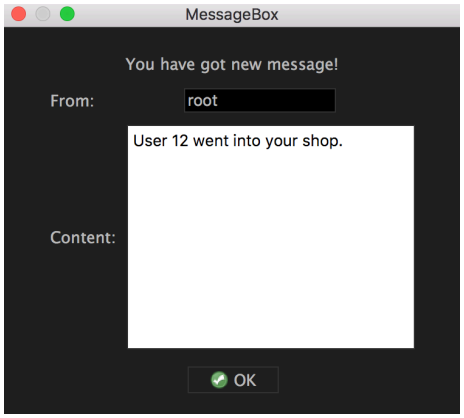
这里实现了拖拽式上传头像的功能。为了实现拖拽的功能，重载了 QLabel 类。默认图片存储在 tmp 文件夹中，为 default.jpg。将图片拖拽到默认图片位置，默认图片将转换为拖拽的图片（只能是 jpg 格式图片），同时记录该图片的 URL，为上传做准备。

管理员填写需要注册的用户名和密码并选择头像后，点击确认按钮，调用 create_new_user 函数，发送“register_new_user”请求，进行登记；服务器确认不重名后，注册用户并发送状态信息。

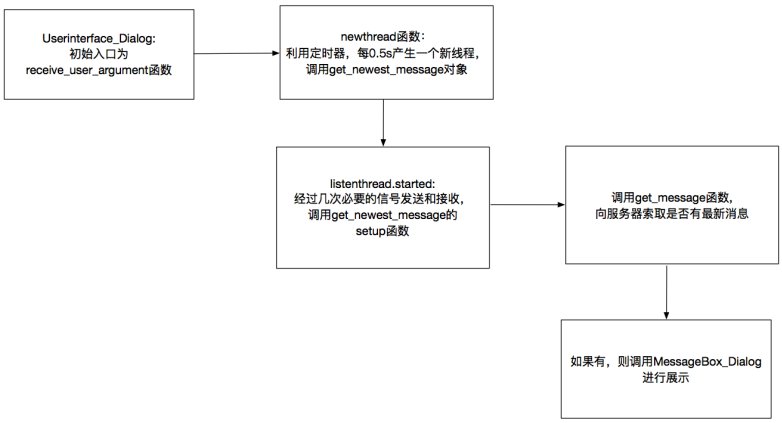
注意：Mac 和 Windows 的 URL 处理有区别：Mac 上 NewUser.py 第 186 行 URL 从 7 开始，Windows 为 8，这是 URL 不同造成的。（exe 文件可以在 Windows 上直接运行,而如果要在 Windows 上运行 py 文件需要进行修改，将 7 改为 8。）

2.1.7 获取服务器消息模块

消息弹窗界面如下：



多线程实现实时获取消息的机制如下：



由于只有用户需要接收消息，管理员不用，所以在 Userinterface_Dialog，即用户界面实现多线程接收消息。又由于用户界面的入口是 receive_user_argument 函数，所以在这个函数中调用 newthread 函数，设置一个定时器，每 0.5s 向服务器发送一次消息请求，看是否有最新的发向自己的消息，如果有，则调用 MessageBox_Dialog 对象显示。

2.2 服务器端

服务器端的代码逻辑比较简单，根据客户端发送的“request”字段的键值进行相应的响应，处理文件中或者内存中的数据并返回相应的数据和状态信息。

由于服务器端函数非常多，逻辑又比较简单，所以这里不作说明，结合下面的协议分析、源代码和注释就可以理解。

需要注意，服务器每接收到一条信息，都会对时间戳进行判断，五秒之前的则不予处理，考虑网络延迟的同时防止重放攻击；服务器发送的所有数据，包括头像图片在内，均使用 DES 算法加密（加密、解密头像的时间较长，可能会导致登录时有些延迟）。

对于多线程，每次服务器收到登记新用户的请求，都会在上传头像前开辟一个新线程，用于接收头像；接收完毕后，线程自动结束。

3. 使用协议

这里具体的分析使用的协议格式。

发送的 JSON 字符串结构如下：

```
{"timestamp": 1513690665.0, "password": "123", "request": "login", "id": "12"}
```

这里只在客户端向服务器发送数据时加上时间戳，因为客户端接收数据的时间很短，所以不考虑重放攻击。

1. 登陆

客户端：

- request：login
- id
- password
- timestamp

服务器端：

- state: fail

或

- state: succeed
- authority: user / admin
- id
- name

2. 获取余额

客户端：

- request: get_user_remaining
- user_id
- timestamp

服务器端：

- state: fail

或

- state: succeed
- remaining

3. 获取头像

客户端：

- request: headimage
- id
- authority
- timestamp

服务器端：

- 直接传输图片

4. 获取商店信息

客户端：

- request: shopinfo
- timestamp

服务器端：

- 含有 id、shopname、ownername 的字典组成的字典

5. 获取一定时期内消息

客户端：

- request： messageinfo
- date1
- date2
- id
- timestamp

服务器端：

- 含有 date、time、from、content 的字典组成的字典

6. 发送用户离开商店信号

客户端：

- request： user_leave_shop
- user_id
- owner_id
- timestamp

服务器端：

服务器仅将用户离开某商店的信息写入 txt 文件，不向客户端发送数据。

7. 用户充值

客户端：

- request: charge_money
- user_id
- num
- timestamp

服务器端：

- remaining

8. 通知服务器用户进入商店

客户端：

- request: user_in_shop

- shop_id
- user_id
- timestamp

服务器端：

服务器仅将用户进入某商店的消息记下，不向客户端发送数据。

9. 获取商店内商品信息

客户端：

- request: getgoods
- shop_id
- timestamp

服务器端：

- 含有 good_id、good_name、price、num 的字典组成的字典

10. 购买商品

客户端：

- request: buygood
- good_id
- good_price
- user_id
- shop_id
- timestamp

服务器端：

- state: fail

或

- state: succeed
- num

11. 获取当前店内顾客

客户端：

- request: get _customer_in_shop
- shop_id

- timestamp

服务器端：

用户 ID 组成的字典

12. 判断某个用户的商店是否存在

客户端：

- request: if_shop_exists
- user_id
- timestamp

服务器端：

- shop_name（存在则为商店名，否则为“none”）

13. 登记新商品

客户端：

- request: register_new_good
- goodname
- goodprice
- goodnum
- shopid
- timestamp

服务器端：

- state

14. 通知客户新商品上架

客户端：

- request: inform_new_good
- shop_id
- goodname
- customer
- timestamp

服务器端：

服务器仅将新商品上架的消息写入 txt 文件以供用户查询，不向客户端发送数据。

15. 获取店内商品

客户端：

- request: getgoods
- shop_id
- timestamp

服务器端：

含有 good_id、good_name、price、num 的字典组成的字典

16. 群发消息

客户端：

- request: broadcast
- content
- timestamp

服务器端：

- state

17. 单发消息

客户端：

- request: inform
- user_id
- content
- timestamp

服务器端：

- state

18. 获取全部用户

客户端：

- request: get_all_user
- timestamp

服务器端：

含有 user_id、user_name 的字典组成的列表

19. 为已存在的用户开店

客户端：

- request: open_shop
- user_id
- shop_name
- timestamp

服务器端：

- state

20. 关闭某个用户的商店

客户端：

- request: close_shop
- user_id
- customer
- timestamp

服务器端：

- state

21. 登记新用户

客户端：

- request: register_new_user
- name
- password
- timestamp

服务器端：

- state

22. 子线程向服务器请求新消息

客户端：

- request: get_newest_message
- user_id
- timestamp（这里 timestamp 不仅起到防止重放攻击的作用，也用作时间比对，查找最新的消息）

服务器端：

- state: succeed
- from_id
- content

或

- state: fail