**FDU定制电动车拍拍卖软件需求分析及设计规格说明**

1. **需求分析：**

**1.1 概述**

实现这个拍卖网络应用，需要两套程序，一套是客户端，一套是服务端。这两个进程之间通过socket在网络上发送和接收报文。

**1.2功能需求：**

**1.2.1总体需求：**

1. GUI界面实现
2. 本次参拍人信息存储
3. 采用传输层基于UDP的实现策略

**1.2.2 server端：**

1. 向定制用户发送消息，用以提示信息
2. 列出当前参拍者情况（当前的登录用户信息（id、socket地址、用户名、密码）、当前中标人群信息（投标价格，投标时间、用户id））
3. 将某参拍者踢出竞拍室
4. 开始车牌拍卖，把本次拍卖情况通知已登录的用户（当前拍卖是否开始、当前登录人数、车牌拍卖信息（数目和范围）、起拍价、最低竞价价格、最低竞价出价时间、当前时间）
5. 关闭车牌拍卖，把本次拍卖结果通知每一位用户：（最高中标价、最低中标价、平均中标价、每个客户是否中标，对应的中标价格为多少）
6. 广播当前最新的拍卖状态（当前拍卖是否开始、当前登录人数、车牌拍卖信息（数目和范围）、起拍价、最低竞价价格、最低竞价出价时间、当前时间）
7. 提供注册功能，并分配一个id号给注册者
8. 客户端登录时，比对后台数据，验证注册时的id和对应密码，并返回登录成功或失败的消息
9. 接受客户端出价，审查其合理性，对整个投标列表进行排序，计算出中标人群
10. 接受客户端离开请求，查看是否可以离开，并告知相应结果

**1.2.3 client端：**

1. 注册一个用于参加拍卖的账户（输入用户名和密码，得到分配的id）
2. 用id和密码登录服务器
3. 主动向服务器询问当前的拍卖状态：（当前拍卖是否开始、当前登录人数、车牌拍卖信息（数目和范围）、起拍价、最低竞价价格、最低竞价出价时间、当前时间）
4. 出价，（在服务方开始拍卖了才奏效）
5. 退出拍卖，需要征得服务器允许才能退出

**1.3性能需求：**

1. 可应用于分组丢失率非常小的网络环境
2. 支持比较大规模用户同时在线

**2.系统设计**

**2.1采用技术：**

pyqt5, protobuf, qt designer, threading, time, pandas，xlrd（1.2.0版本，这个版本和我电脑配置适应）,socket

pyqt5, qt designer, 以及相关的tools用于ui界面设计

protobuf, 包括下载的编译器、和python库，用于协议设计

threading，用于多线程编程

time，用于取得当前时间，投标时间

pandas，xlrd，用于处理excel文档，处理客户信息。

socket 用于客户端和服务端通信

**2.2体系结构：**

C/S 模式

**2.3数据/类设计：**

**2.3.1类：**

**2.3.1.1总体类设计：**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 名称 | 位置 | 简介 |
| 1 | Server | server\_message.py | 服务端核心，收发消息，对消息进行处理 |
| 2 | Client | client\_message.py | 客户端核心，收发消息，对消息进行处理 |
| 3 | Ui\_serverwindow | server\_ui/serverwindow.py | 服务端ui主界面，包括显示消息和操作的按钮 |
| 4 | Ui\_Dialog | server\_ui/kickdialog.py | 服务端踢人操作对话框ui |
| 5 | Ui\_Dialog | server\_ui/msgdialog.py | 服务端发消息操作对话框ui |
| 6 | Ui\_openauctiondialog | server\_ui/openauctiondialog.py | 服务端开始拍卖操作对话框，输入拍卖初始信息参数ui |
| 7 | Ui\_LoginWindow | client\_ui/loginwindow.py | 客户端登录界面ui，包括输入框和按钮 |
| 8 | Ui\_SigninDIalog | client\_ui/signindialog.py | 客户端注册界面ui，包括输入框和按钮 |
| 9 | Ui\_MainWindow | client\_ui/bidwindow.py | 客户端拍卖界面，包括消息显示和操作的按钮ui |
| 10 | Ui\_Biddialog | client\_ui/biddialog.py | 客户端竞价对话框ui，包括价格输入框和按钮 |
| 11 | Ui\_infodialog | client\_ui/infodialog.py | 提示消息对话框ui，比如输入信息错误提示，密码错误提示等等 |
| 12 | SignalBdWindow | client\_ui/loginwindow.py | 信号类，用于在不同窗口间传递信号 |

**2.3.1.2 关键类Server：**

**2.3.1.2.1属性：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 简介 | 数据类型 | 字段含义 | 初始值 |
| self.sock | 程序套接字 | socket.socket()类 | [0,10000] |  |
| self.client\_pool | 客户池 | 字典  {int:list[(str,int),str,str]} | {id:[套接字二元组,  client.name,  client.key]} | {} |
| self.id\_now | 当前可分配的id号 | int |  | 0 |
| self.is\_auction\_begin | 拍卖是否开始，1表示开始 | int | 0：未开始  1：拍卖开始 | 0 |
| self.licence\_num | 本次拍卖车牌总数 | int |  | 10 |
| self.licence\_range | 本次拍卖车牌范围 | tuple:（int,int） | (最小的车牌号，最大的车牌号) | （0，9） |
| self.floor\_price | 起拍价 | int |  | 1000 |
| self.price\_range | 合理出价区间 | list:[int,int] | [当前可接受的最低价，当前可接受的最高价] | [1000,1060] |
| self.current\_lowest\_bid | 当前最低竞拍价格 | tuple：（int,double,int） | (出价价格，出价时间,出价人id) | (1000,0,0) |
| self.bidder\_pool | 当前中标人list | list[tuple(int,int,int)] | list[((出价价格，出价时间,出价人id))] | [] |
| self.mean\_bid\_price | 平均中标价格 | int |  | 0 |
| self.df | 本次拍卖所有客户信息的表单（储存在excel中，df是读取的电子表格） |  |  | pandas.read\_excel(‘client\_pool.xlsx’) |

**2.3.1.2.2方法：（只介绍关键部分）**

run()

* 思路

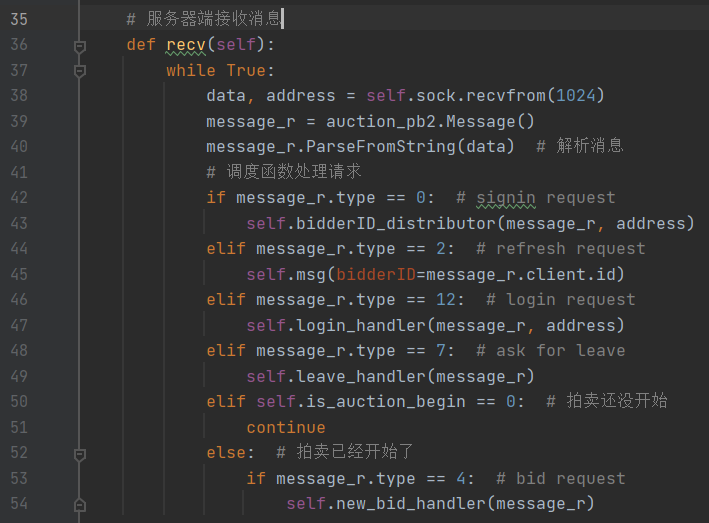
服务器要一直处于接受消息状态的，采用一个循环不停地recv消息，对收到的每条消息进行预处理，是一个dispatcher，收到一条消息，解码，进行类型判断，并分配到对应的处理函数。

解析消息调用的是protobuf\_pb2.py里面的函数，是编译器自动生成的代码，解析消息可以把一个字节流解析成消息类（Message），然后通过 . 来取得属性值。

另外还有一个序列化函数，调用了以后把消息类序列化为字节流，通过socket发送。

其中，如果拍卖没有开始，是不会接受客户的竞价请求的，直接把客户的请求包丢掉。

* 代码



msg()

* 思路

功能是，把当前拍卖状态群发或发给定指定用户

输入：

data：string类型，表示要输出的数据

bidderID=-1时，代表群发

否则bidderID表示发给id为该值的用户

具体实现：

先把当前的拍卖状态写入消息体，再把要发送的数据data（也就是发给用户的话语）写入消息体

如果是单发就直接在客户池（当前在线的客户）中找到id为bidderID的客户的套接字二元组，发过去

群发就遍历一遍客户池，得到地址发给所有客户

* 代码



bidderID\_distributor()

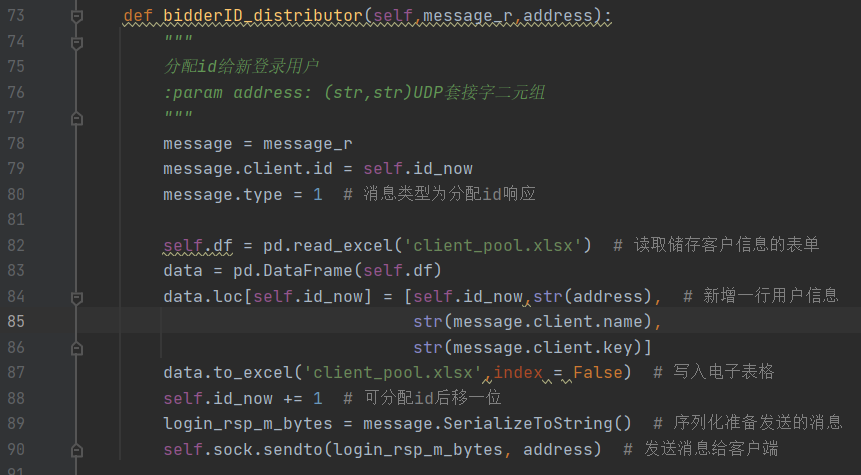
* 思路

输入是收到的客户发来的消息类Message，udp套接字二元组

这个函数的功能是在客户进行注册时，分配一个id给当前客户，把该客户的信息（id、套接字地址、姓名、密码加入储存着客户信息表单client\_pool.xlsx中，分配id的消息类型为1。

发消息给客户通知分配的id值

* 代码



loginhandler()

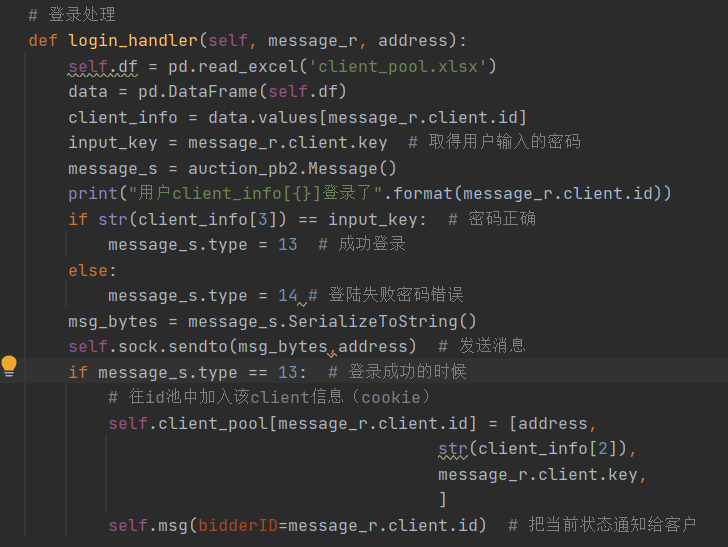
* 思路

检索储存着所有客户信息的表单，根据登录请求消息中的id号码查找对应的密码，如果就允许登录，否则拒绝登录。这两类返回消息的类型分别是13和14。

登录成功后需要把该客户加入当前在线的客户池self.client\_pool中

最后把当前的拍卖状态发给该客户

* 代码



new\_bid\_handler()

* 思路

功能是，新的投标价格到来，对当前拍卖价格重新排序，根据情况修改当前中标人群池（self.bidder\_pool），回传给客户出价成功或出价失败的消息。

如果新价格不合理，返回出价失败

如果新价格不在价格区间内，表示投标失败

如果新价格合理：确定是否要把该投标人加入中标人群中

如果当前投标人少于车牌数目，那么直接把该人加入中标人群中

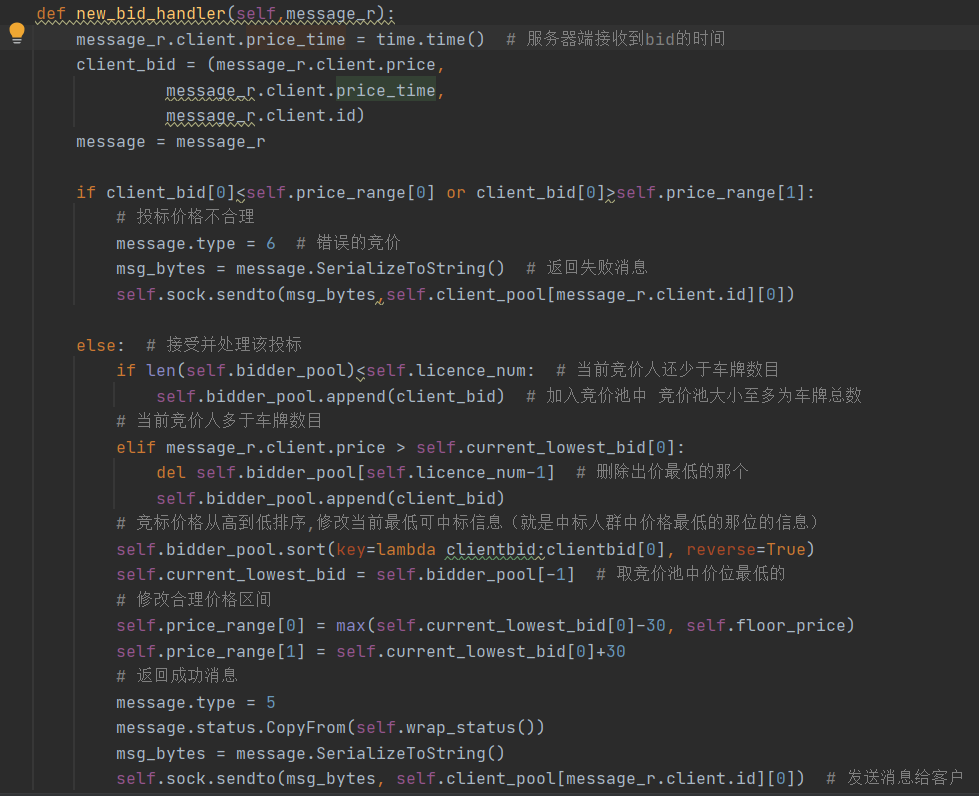
如果当前人数多余车牌数目，看新投标价格与中标人群中价格最低的相比是不是更大，如果更大，就把中标人群中价格最低的那个人踢出来，把当前的人加进去，然后对中标人群重新排序，自大到小储存在self.bidder\_pool中。

然后合理出价区间最低值就是可中标的人群中价格最低的那位的钱减30元和初始设定的底价中的较大值

最高值就是可中标人群中价格最低的那位的钱加30元

这里的意思是，客户端的出价必须在这个范围内才能够出价成功。（这里参考了上海市车牌拍卖的规则），所以大量客户同时出价，在处理前面的客户的出价时，价格区间已经变了，如果后续到来的客户出价请求不在这个出价范围内，那么就出价失败，所以抢车牌是拼手速的，比如这时候我看见的合理出价区间是这样的，我根据这个区间出价了，但是到服务器那边后，出价区间已经变了，我的出价就无效了。

* 代码



leave\_handler()

* 思路

处理客户端离开请求，

遍历self.bidder\_pool看看这个客户是不是在当前中标人群中，

如果在就拒绝他的离开请求，message.type = 9

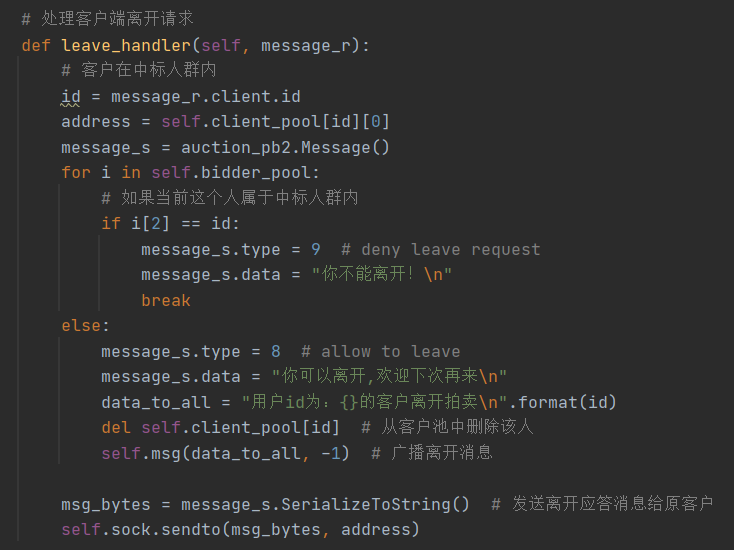
并通知该人不能离开

如果不在就允许它的离开请求, message.type = 8

通知该人允许离开

同时还要把这个人离开的消息广播

* 代码

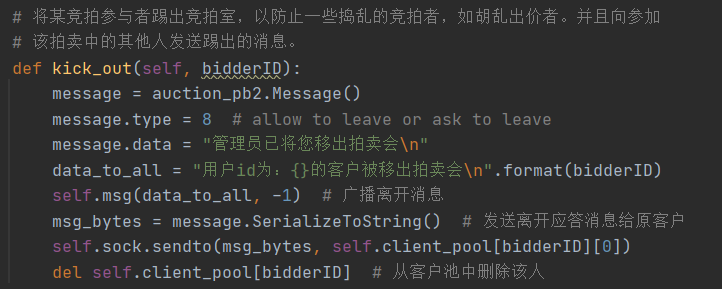


kick\_out()

* 思路

把id号为bidderID的客户踢出当前拍卖室，消息类型为8，并且广播这个消息，从当前在线的客户池中删除该人。

* 代码



open\_new\_auction()

* 思路

开始拍卖，调用self.msg()函数广播拍卖开始消息

* 代码



close\_auction()

* 思路

先用self.msg()函数广播拍卖结束的消息

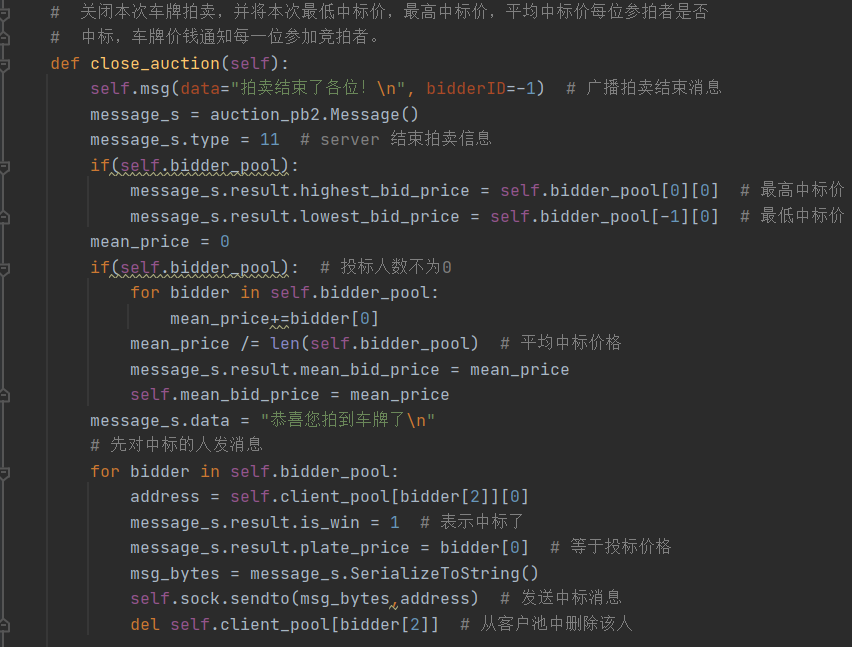
然后对每个人发送拍卖结果，消息类型为11。

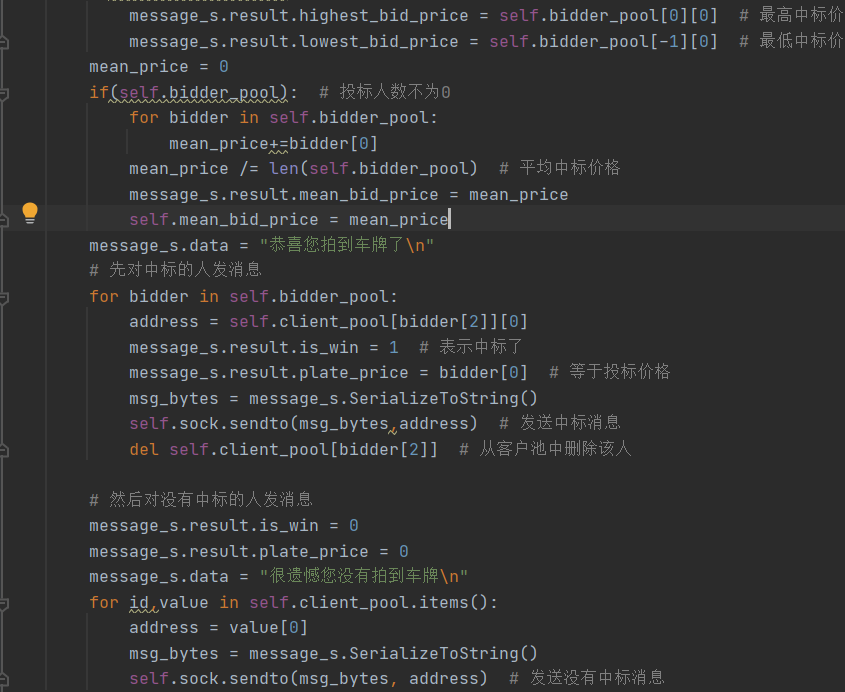
计算最高、最低中标价格和平均中标价格。（在中标人群池self.bidder\_pool中计算）

对于self.bidder\_pool中的人（中标的人）发送中标消息，和对应的中标价格，发送完后从当前在线客户池self.client\_pool中删除这些人

对其他人（当前在线客户池中剩余的人）发送未中标消息

* 代码





**2.3.1.2 关键类Client：**

**2.3.1.2.1属性：**

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 简介 | 数据类型 | 初始值 |
| self.sock | 程序套接字 | socket.socket()类 |  |
| self.name | 客户端用户名 | str | ‘Alice’ |
| self.id | 客户端id | int | -1 |
| self.key | 客户端密码 | str | ‘’ |

**2.3.1.2.2方法（只展示关键部分）**

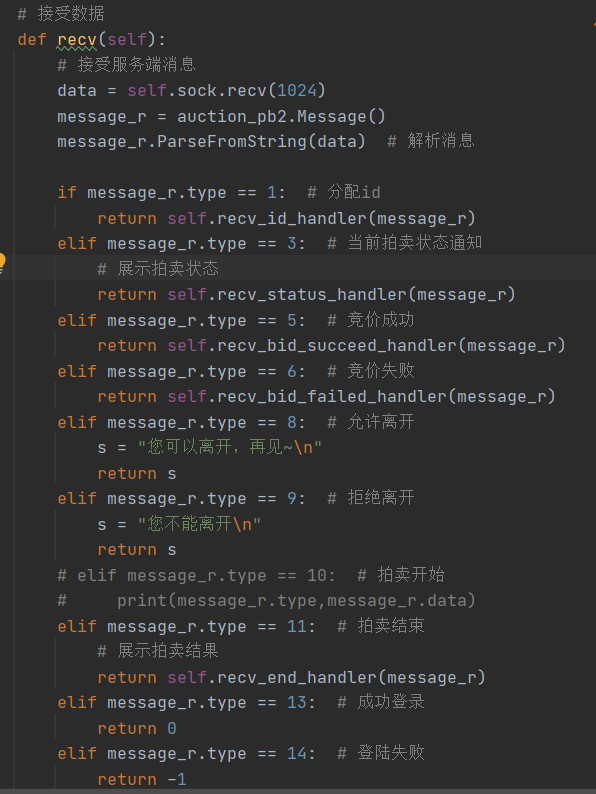
recv()

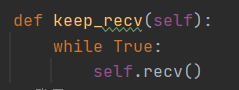
* 思路

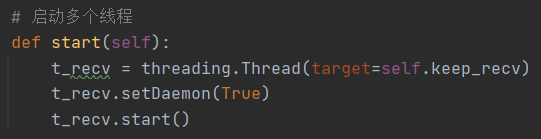
客户端在登录时也需要循环接受消息，和服务器一样，这个循环接受消息的动作需要一直进行，此外还需要支持客户端的系列操作如竞价，所以把这个循环接受recv作为一个线程来一直运行。

等待服务器发送通知消息，或者踢人消息。每个消息到来，先解码，判断类型，然后调用相应的处理函数，处理函数返回的内容是字符串（除了登陆成功和登陆失败两类消息），然后客户端把相应的字符串显示在公告栏上。供顾客观看。

* 代码





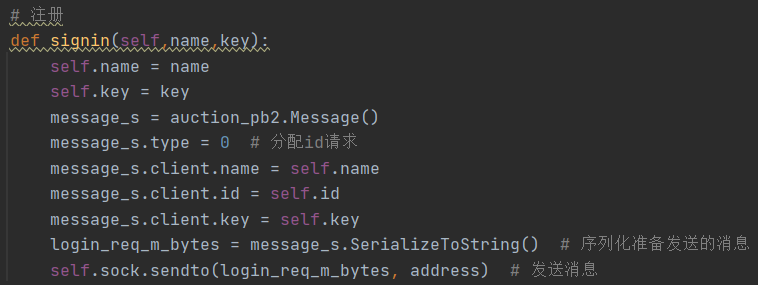


signin()

* 思路

注册，发送用户名和密码，消息类型为0，表示请求分配id，发送消息给服务器

* 代码

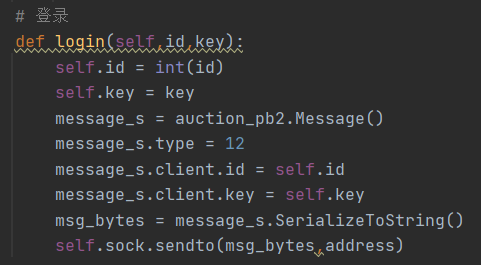


login()

* 思路

客户端发送登录请求，输入id和密码，消息类型为12，发送给服务器

* 代码

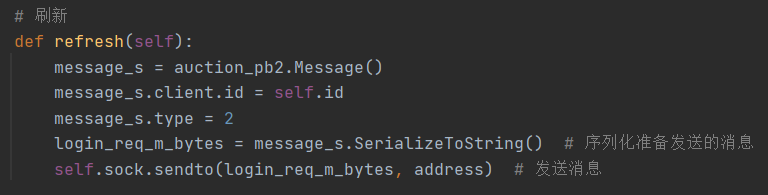


refresh()

* 思路

消息类型为2，表示请求当前拍卖状态，以自己的id为名义，发送给服务器

* 代码

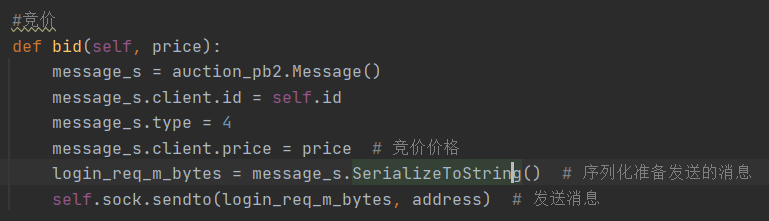


bid()

* 思路

消息类型为4，把竞价价格和id发给服务器

* 代码

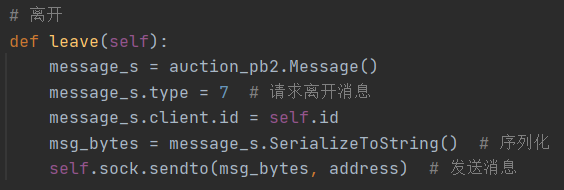


leave()

* 思路

消息类型为7，把id发给服务器，表示该id对应的客户请求离开

* 代码



**2.4 接口设计**

**2.4.1 协议层**

**2.4.1.1总体思路 protobuf**

考虑到协议层的设计目标：可扩展的、简单的、高效、人类可读的、容易兼容的。

本次pg采用了google的protobuf协议。

需要在官网下载相关的软件proto编译器，人工编写的协议的类型为proto，编写的语言和c++类似。

这个协议体积小、序列化速度快（比XML和JSON快20-100倍）、传输速度快

并且加密性好，抓包只能看到序列化后的字节，并不能了解其含义。

最重要的是它可扩展、向后兼容性好，不必破坏原有的数据格式，就可以直接对进行更新，而且还是跨平台、跨语言的。只需要维护proto文件就可以了。

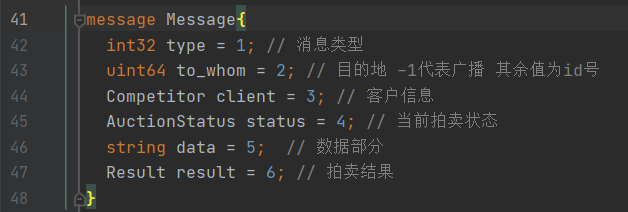
不过它也有一些缺点，它通用性比较差，因为绝大多数的行业标准使用XML或者json，来编写，这个是谷歌使用的工具。此外他以二进制流方式存储，也就是人类不可读，需要通过proto文件才能了解到数据结构

**2.4.1.2 auction.proto文件编写：**

每个域后面跟的‘=int’是一个id号码，并不是初始值。

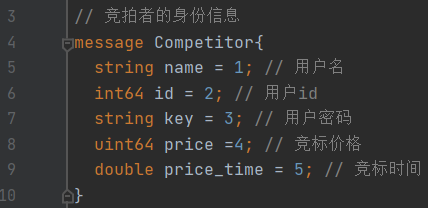
Message：

这个是总的消息体，消息体包括消息类型、目的id、客户信息、当前拍卖状态、数据（传送的string话语）、拍卖结果这几大类。



其中嵌套的消息体的结构体如下：

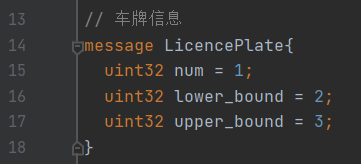
Competitor:



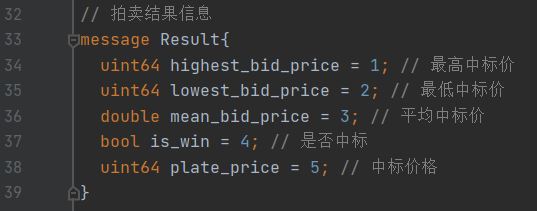
AuctionStatus:



LicencePlate:



Result:



**2.4.1.3 编译proto文件：**

本次pg使用了将proto编译成python的编译器，可在谷歌官网下载。

编写好auction.proto文件

在该文件所在目录下打开cmd

输入下列命令进行编译，生成auction\_pb2.py（在同目录下）,要使用其中的类和方法直接在其他的.py文件中import这个auction\_pb2.py就可以了



**2.4.1.4 消息类型**

下面显示不同的消息类型：也就是Message类中的type域的值

|  |  |
| --- | --- |
| 消息代码 | 简介 |
| 0 | client分配id请求 |
| 1 | server分配id响应 |
| 2 | client向server请求消息 |
| 3 | server向client发消息 |
| 4 | client竞价请求 |
| 5 | sever返回竞价成功消息 |
| 6 | server 返回竞价失败消息 |
| 7 | client请求离开消息 |
| 8 | server应答允许离开消息 |
| 9 | server应答拒绝离开消息 |
| 10 | 备用 |
| 11 | server结束拍卖 |
| 12 | client 登录信息 |
| 13 | server 返回登录成功信息 |
| 14 | server 返回登录失败信息 |

**2.4.2 UI**

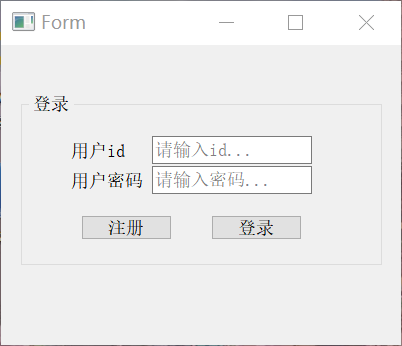
**2.4.2.1概述:**

使用了python的pyqt5库和一些相关的工具，使用Qt designer初步设计界面生成.ui文件，再用相应的工具把.ui文件转成.py代码，最后再自己编写信号和槽函数。

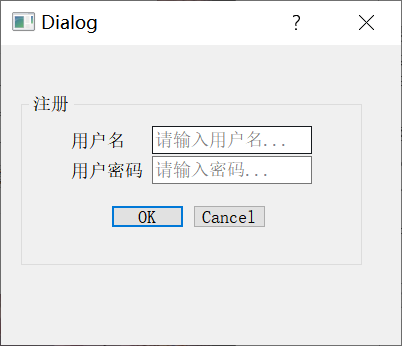
在客户端主要有两个界面，一个是登录界面，一个是竞价界面，服务器端有一个管理拍卖界面。此外，这两大类（client&server）分别还有许多对话窗口，比如竞价时输入价格的对话框、踢人时输入id的对话框等等。下面将依次介绍：

**2.4.2.2客户端界面**

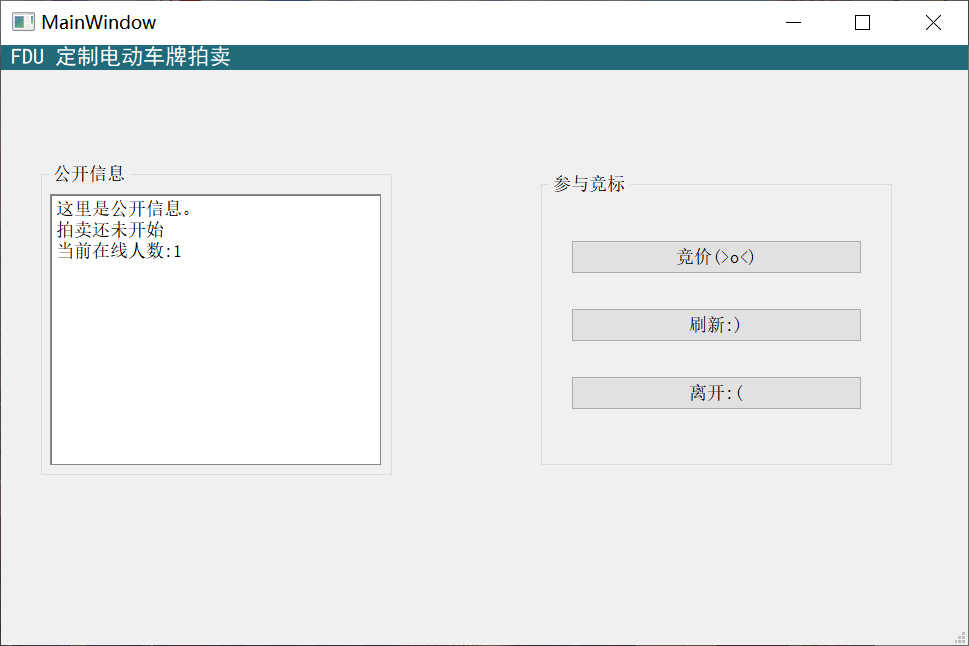
**2.4.2.2.1登录界面**



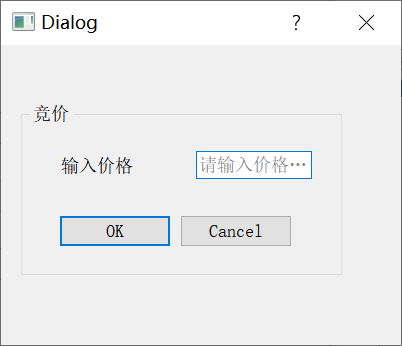
**2.4.2.2.2注册对话框**



**2.4.2.2.3竞价界面**

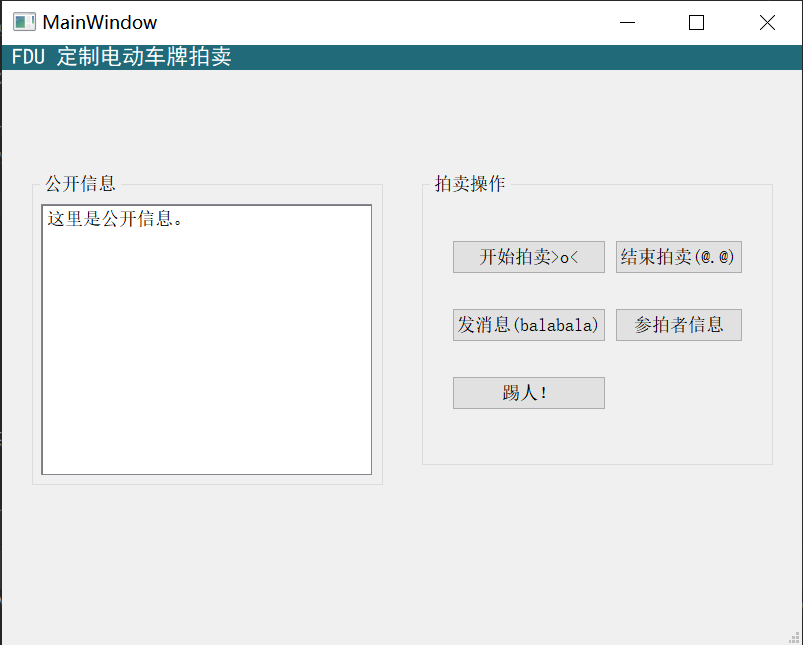


**2.4.2.2.4竞价对话框**

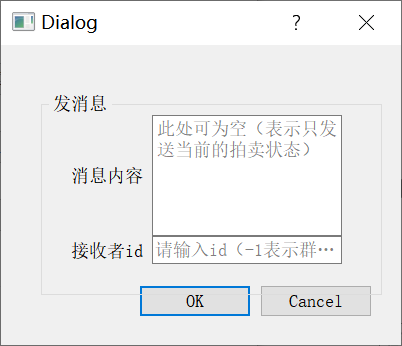
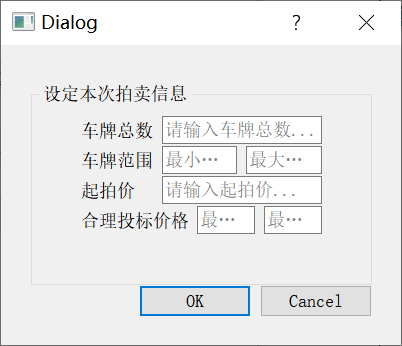


**2.4.2.3服务端主界面：**

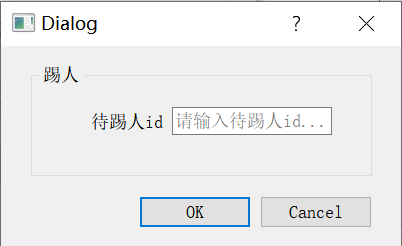
**2.4.2.3.1拍卖管理主界面：**



**2.4.2.3.2开始拍卖输入信息对话框** **2.4.2.3.3发送消息对话框**



**2.4.2.3.4踢人对话框**

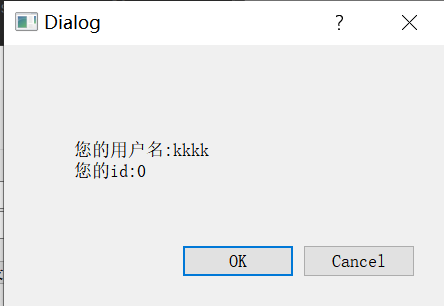


**2.4.2.4具体设计：（以客户端为例子）**

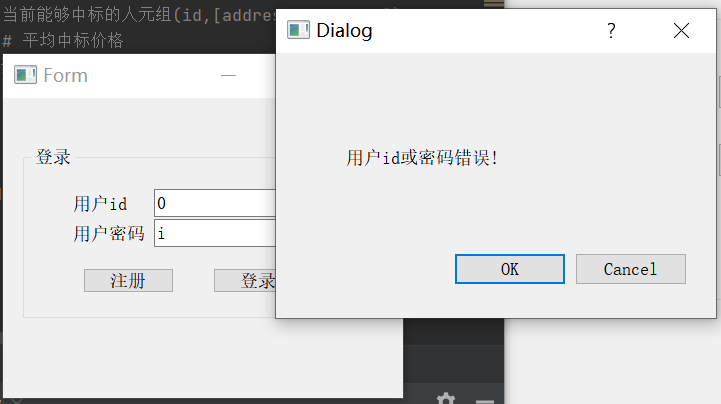
1．登录窗口

在登录窗口，可以先进行注册，

注册完后会弹出一个对话框，告诉分配的id号码，客户需要记住这个id，然后使用id和密码进行登录



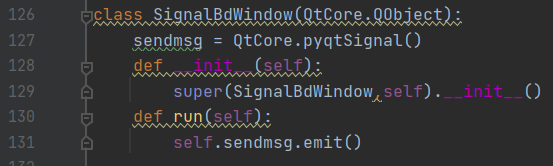
注册完后使用分配得到的id号码和密码进行登录，如果id和密码不匹配，则会弹出消息框：



如果匹配，则登陆界面自动关闭，然后自动打开竞价窗口

2.窗口间信号传递

这里的自动打开涉及到不同窗口之间信号的传递，于是我定义了一个信号量类

【loginwindow.py】

当登录成功后，登录窗口会发送一个消息

【loginwindow.py】

然后信号被绑定在竞标窗口的show槽函数上，竞标窗口收到信号，就会show了

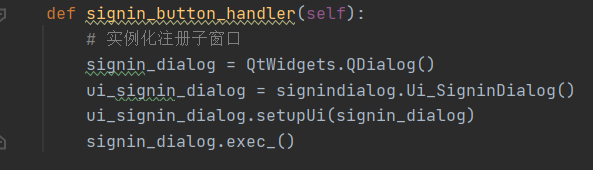
【loginwindow.py】

1. 对话框弹出

客户端和服务器端按下一个按钮，有的会在消息展示框中显示消息，有的会生成一个对话框，都是被绑定在相应的槽函数上

比如登录界面的注册按钮，绑定注册对话框：

是在这个槽函数中实现的

【loginwindow.py】

4.消息类和ui类的接口：

以客户端的登录窗口为例

首先实例化一个客户消息类Client（），为client

【loginwindow.py】

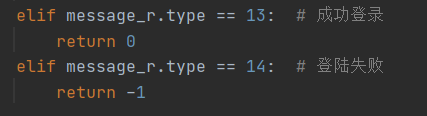
然后把这个消息类作为参数传给登录窗口

【loginwindow.py】

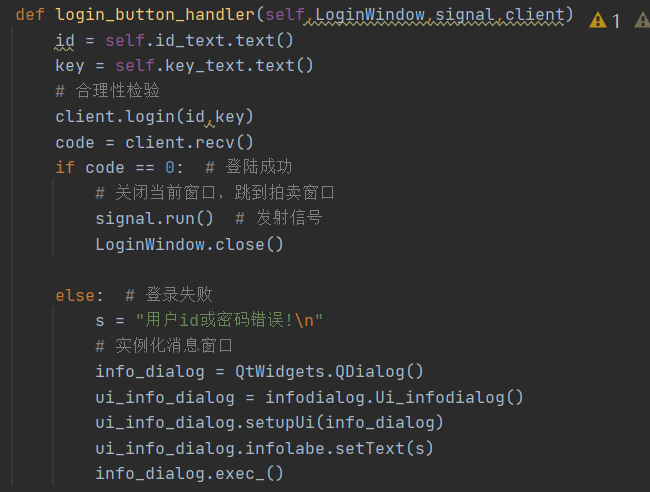
登录窗口在进行登录操作时，收集用户填写在文本框里面的id号码和密码

作为参数传给消息类，也就是调用client的login（）函数，然后client消息类就会发消息给server。

client会接受server回传的消息，client的recv()对其进行解码并判断类型，调用相应的处理函数，如果是成功类型的消息，client会返回0，否则返回-1

【client\_message.py】

根据client消息类返回的参数，登录窗口会显示不同的界面，或是提示id或密码错误，或是跳转到竞价窗口。

【loginwindow.py】

5.关于消息展示：

在客户端竞价窗口，需要展示服务器发来的消息，比如踢人、当前拍卖状态等等，这个消息展示是要一直循环进行的，因为不知道服务器什么时候会发消息过来，所以用一个线程来一直跑这个函数。

通过这个函数来实现:



首先同样通过传参把客户端消息类client传给竞价窗口类的成员函数，然后调用消息类的方法，recv()方法对收到的服务器的消息进行解码：解码完了分析消息类型，调用相应的处理函数，处理函数返回一个string，这个string就是应当展示的信息。竞价窗口得到string，就把这个string展示在消息栏中。

此外，如果消息是（您可以离开，再见）或者是（拍卖结束了），说明或者客户端请求离开、或者服务器踢人、或者拍卖结束了，此时客户端就被离线了，解绑所有的按钮，使得按钮失效，无法再向服务器发送消息。而服务器端会把客户端的信息从当前在线人群池众删除。客户端通过这种方式离开拍卖会。

**3.参考文献**

[1]. proto buffer谷歌官方文档

<https://developers.google.com/protocol-buffers/docs/pythontutorial>

[2]. 上海市个人非营业性客车额度拍卖须知

<https://www.alltobid.com/contents/16/69.html>

[3]. Python Socket

<https://docs.python.org/3/library/socket.html>

[4]. PyQt5 Reference Guide

<https://www.riverbankcomputing.com/static/Docs/PyQt5/>

[5]. PyCharm安装PyQt5及其工具（Qt Designer、PyUIC、PyRcc）详细教程

<https://blog.csdn.net/qq_32892383/article/details/108867482>