基于UDP通信的网络拍卖行

胡天晓
14300240007
14保密管理

项目简介

概述

本软件可以实现网络拍卖功能，分为服务器端和客户端。只能同时运行一个服务器端，但可以运行多个客户端。

Github: https://github.com/TianxiaoHu/AuctionOnline

网络环境

本程序的应用场景基于分组丢失率非常小的网络环境，为此采取基于UDP的实现策略。

为方便测试，所有通信在本地回环地址127.0.0.1上进行。程序启动时会自动初始化网络环境，服务器端端口为20210，客户端端口为10000～65535中某一随机端口。

若初始化成功，程序界面出现初始化成功提示，若出现错误（如端口号被占用等），程序将自动退出。

编程语言

本拍卖行使用python进行开发，python版本为2.7.12。

采用轻量级GUI库Tkinter进行用户图形化界面的交互。

运行方式

Windows

为方便windows 平台使用，已使用pyinstaller打包为独立的exe可执行文件，用户双击server.exe或client.exe即可运行。

Mac/Linux

首先在终端中更改目录到当前目录：

```
$ cd directory/to/AuctionOnline
```

启动服务器端：

```
$ python server.py
```

```
$ python client.py
```

功能一览

服务器端

程序启动后，如果初始化成功，则会弹出提示语，如果因为某些原因初始化失败（如端口占用等），程序将自动退出。

显示帮助

点击页面上方的 `Help` 按钮即可显示支持的服务器端指令集及详细用法。

指令集

- `/auctions`

  用于查看当前所有拍卖房间。

- `/opennewauction auctionname base gap`

  用于创建新的拍卖房间，`auctionname` 为房间名，`base` 为起拍价，`gap` 为每次出价最低增加的价格。

- `/users`

  用于列出当前所有在线用户。

- `/list auctionname`

  用于列出某房间的拍卖信息。包括：房间名、起拍价、每次竞价最低增加价格、当前最高价出价者、房间内在线用户（不包括已经离开的拍卖者）及出价历史。

指令集

- `/msg username1 username2 ... | message`

  用于对一个或多个用户发送消息，`username` 为用户登录时的用户名。最后一个用户名和消息之间用 `|` 分隔。

- `/broadcast auctionname message`

  用于对某个拍卖房间进行房间广播，所有在房间中的用户会收到消息。

- `/kickout username1 username2 ...`

  用于踢出房间内用户，支持同时踢出多人，但不能踢出当前出价最高的用户（拍卖已结束的房间除外）。

指令集

- `/finish auctionname`

  用于结束房间内拍卖。房间内拍卖结束后，房间相当于被锁定。用户无法继续出价，不在房间内的用户将无法再加入房间，房间内的所有用户都可被服务器踢出，用户可以使用 `/leave` 指令离开房间。

- `/close auctionname`

  用于关闭拍卖房间并从服务器端删除该房间的历史信息。拍卖房间内必须没有任何用户。

- `/restart auctionname`

  用于重新启动某房间内的拍卖。房间内所有拍卖历史将被清空，拍卖信息被还原为初始信息。
  房间内所有用户将收到拍卖已重启的提示信息。
  已经在房间内的用户不会被踢出。

传输日志

为方便历史追溯及客户端动作记录与核对，服务器端设置了传输日志窗口，可以显示出每个用户的网络地址及与服务器交互的详细动作信息。

当传输日志窗口信息过多时，可以点击 `Clear log` 按钮清空日志。

退出程序

点击 `Exit` 按钮即可退出程序，释放内存。

客户端

显示帮助

点击页面上方的 `Help` 按钮即可显示支持客户端指令集及详细用法。

若用户输入了错误的指令或指令格式，服务器端会反馈用户*Invalid input*提示信息。

指令集

- `/login username`

  用于以 `username` 身份登录到服务器端。

- `/auctions`

  用于查看当前所有拍卖房间。

- `/join auctionname`

  用于加入某个拍卖房间，注意每个用户只能同时在一个房间内。
  房间内所有在线用户将会收到有人加入的信息。

指令集

- `/list`

  用于查看当前房间价格信息及出价历史。

- `/bidders`

  用于查看当前房间的所有在线用户。

- `/bid price`

  用于为当前房间拍卖物品出价。
  房间内所有用户将会收到出价信息。

- `/pubmsg message`

  用于向当前房间内所有在线用户统一发送房间内公开消息。

指令集

- `/primsg username1 username2 ... | message`

  用于向任意个当前房间内的用户发送私人消息。注意最后一个 `username` 和 `message` 之间需要用 `|` 分隔。

- `/leave`

  用于退出当前房间，但在当前房间拍卖未结束且当前用户出价最高时无法退出。
  房间内所有用户将会收到有人离开房间的消息。

- `/exit`

  用于退出登录并永久清除服务器端所有用户信息。若用户在房间内则会自动执行 `/leave` 指令，**房间内所有用户将会收到其退出的消息，服务器端也会收到用户退出登录的消息。**
  在当前房间拍卖未结束且当前用户出价最高时无法退出。

退出程序

当用户执行完 `/exit` 指令后，点击 `Exit` 按钮即可退出程序，释放内存。

代码结构

服务器端

全局变量

```
global AuctionRoom, User, AddMapID, IDMapAdd
AuctionRoom, User = [], []
AddMapID, IDMapAdd = {}, {}
```

为方便调用，有关用户及房间的信息被存储在全局变量中。由于每个用户有地址（唯一）和用户名两种标识，因此用两个全局字典存储其相对应关系。

AES加密模块

考虑到有必要对服务器端和客户端对话进行加密以防止中间人（MITM）攻击，故采用AES加密对服务器端和客户端回话进行处理。

采用成熟的python加密模块Crypto进行加密。

由于Crypto中的AES模块只能对字符数为16的整数倍的字符串进行加密，因此在调用之前需要将其先用`'\0'`填充至16的整数倍长度再进行传输。同理，接收之后也需要去除其中的填充字符。

在接下来服务器端和客户端回话过程中，发送之前均先用统一密钥加密，接收后再解密。

AES加密模块

```
from Crypto.Cipher import AES

padding = '\0'
pad = lambda x: x + (16 - len(x) % 16) * padding
unwrap = lambda x: x.replace('\0', '')
key = '1234567890abcdef'
mode = AES.MODE_ECB
encryptor = AES.new(key, mode)
decryptor = AES.new(key, mode)

def AESencrypt(plaintext):
    plaintext = pad(plaintext)
    return encryptor.encrypt(plaintext)

def AESdecrypt(ciphertext):
    plaintext = decryptor.decrypt(ciphertext)
    return unwrap(plaintext)
```

Room类

当服务器端创建了新的拍卖房间时，将会初始化一个Room类的示例，并将其房间名添加到全局列表AuctionRoom中。

下面列出Room类的初始化函数细节，其他类接口仅列出函数名及参数，具体可以参见python源代码。

Room类

```
class Room():

    def __init__(self, name, baseprice, gap):
        self.name = name
        self.bidder = []
        self.history = []
        self.base_price = float(baseprice)
        self.highest_price = float(baseprice)
        self.highest_user = ''
        self.gap = float(gap)
        self.closed = False
```

Room类

```
class Room():

    def add_bidder(self, bidder_ID):
        pass

    def remove_bidder(self, bidder_ID):
        pass

    def draw_bid_history(self):
        pass

    def draw_bidder_address(self):
        pass

    def draw_bidder_ID(self):
        pass

    def update_bid_info(self, UserID, price):
        pass
```

Server类

当服务器端启动时会初始化一个Server类，若初始化成功，则屏幕上将出现成功的提示消息，若失败，则程序会弹出提示后自动退出。

服务器端定义了三个有关数据收发的接口，分别为

- send_message

  用于向某一IP地址发送消息。

- receive_message

  用于监听客户端传来的消息。

- broadcast

  用于向多个IP地址批量发送消息，常用于房间内广播等。

Server类

下面列出Server初始化细节及数据收发接口的实现。

```
import socket

class Server():
    def __init__(self, local_port=20210):
        try:
            local_IP = '127.0.0.1'
            local_address = (local_IP, local_port)
            self.s = socket.socket(socket.AF_INET,
                                   socket.SOCK_DGRAM)
            self.s.bind(local_address)
            window.feedback_message.insert(1.0,
                'AuctionOnline Server Initialized!\n')

        except:
            window.feedback_message.insert(1.0,
                'Fail to create Server UDP socket...\n')
            time.sleep(2)
            sys.exit()
```

Server类

```
class Server():

    def receive_message(self):
        message, address = self.s.recvfrom(2048)
        plaintext = AESdecrypt(message)
        window.log_message.insert(1.0, 'received: ' +
            plaintext + ' from '+ str(address) + '\n')
        return plaintext, address

    def send_message(self, message, address):
        ciphertext = AESencrypt(message)
        self.s.sendto(ciphertext, address)
        window.log_message.insert(1.0, message +
                    ' send to ' + str(address) + '\n')

    def broadcast(self, message, client_addresses):
        window.log_message.insert(1.0,
                            'Broadcasting ended!\n')
        for client_address in client_addresses:
            self.send_message(message, client_address)
        window.log_message.insert(1.0,
                            'Broadcasting...\n')
```

Server类

```
class Server():

    def deal_client_command(self, message, address):
        fields = message.split(' ')
        if fields[0] in ['/login', '/auctions', '/join',
                         '/list', '/bidders', '/bid',
                         '/pubmsg', '/primsg','/leave',
                         '/exit']:
            if fields[0] == '/login':
                pass

            if fields[0] == '/auctions':
                ... ...

        else:
            self.send_message('Invalid input!', address)
```

Server类

```
class Server():

    def deal_server_command(self, message):
        window.command_entry.delete(0, END)
        window.error_label['text'] = ''
        fields = message.split(' ')
        if fields[0] in ['/msg', '/list', '/kickout',
                         '/restart', '/opennewauction',
                         '/auctions','/users', '/finish',
                         '/close', '/broadcast']:
            if fields[0] == '/msg':
                pass

            if fields[0] == '/list':
                ... ...

        else:
            window.error_label['text'] = 'Invalid input!'
```

GUI

采用轻量级图形界面库Tkinter,
由 `Button` 、 `Entry` 、 `Text` 、 `Label` 和 `MessageBox` 组件构成。

多线程

程序需要两个线程同时进行，主线程用于处理服务器端输入的指令，另外开启一个线程专门用于循环监听端口，处理客户端发来的信息。

`ListennerThread` 继承自 `threading.Thread` ，将其初始化方式重载为监听端口。

多线程

```
import threading

class ListenerThread(threading.Thread):
    def run(self):
        while True:
            data, address = server.receive_message()
            server.deal_client_command(data, address)

if __name__ == '__main__':
    listener_thread = ListenerThread()
    listener_thread.setDaemon(True)
    listener_thread.start()
    window.root.mainloop()
```

在主线程中将子线程设为与主线程同时结束，同时启动GUI线程处理服务器端的指令。

客户端

客户端结构相对于服务器端结构较为简单，由于客户端不保存状态信息，故其没有服务器端的全局变量和Room类。

客户端中的AES加密模块、GUI、多线程和服务器端逻辑相同或相似，此处省略。

Client类

当客户端启动时会初始化一个Client类，若初始化成功，则屏幕上将出现成功的提示消息，若失败，则程序会自动退出。

客户端只定义了两个关于数据收发的接口，分别为

- send_message

  用于向服务器端发送消息。

- receive_message

  用于监听服务器端传来的消息。

其他

在开发过程中，结合实际应用场景对服务器端和客户端指令集进行了修改与扩充，如取消了服务器端的 `/enter` 指令，改为更加详细的 `/list` 指令，以及设置 `/kickout` 等指令可以直接指定房间名作为参数等。除此之外，还增加了一些实用性的功能如房间内广播、用户私信等。

Thank you for listening