当代数据库数据库管理系统 Homework1:寻宝游戏(一)

10185501409 池欣宁

实验环境

1. MongoDB community @ 4.4
2. MongoDB 可视化： MongoDB Compass
3. Flask 框架
4. Flask APSschedule
5. 前端：html+ Jinja2
6. Python 3.8
7. venv 虚拟环境与python包版本管理

   venv 对应的包各版本如下：

```
(venv)  chixinning@mymac ~/Desktop/hw1 virtualenv venv
created virtual environment CPython3.8.5.final.0-64 in 715ms
  creator CPython3Posix(dest=/Users/chixinning/Desktop/hw1/venv, clear=False, gl
obal=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle
, via=copy, app_data_dir=/Users/chixinning/Library/Application Support/virtualen
v)
    added seed packages: APScheduler==3.6.3, Flask==1.1.2, Flask_APScheduler==1.
11.0, Flask_PyMongo==2.3.0, Jinja2==2.11.2, MarkupSafe==1.1.1, Werkzeug==1.0.1,
attrs==20.2.0, certifi==2020.6.20, chardet==3.0.4, click==7.1.2, idna==2.10, ini
config==1.1.1, itsdangerous==1.1.0, numpy==1.19.2, packaging==20.4, pandas==1.1.
3, pip==20.2.3, pluggy==0.13.1, py==1.9.0, pymongo==3.11.0, pyparsing==2.4.7, py
test==6.1.1, python_dateutil==2.8.1, pytz==2020.1, requests==2.24.0, setuptools=
=50.3.0, six==1.15.0, toml==0.10.1, tzlocal==2.1, urllib3==1.25.10, wheel==0.35.
1, xlwt==1.3.0
  activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,Pyt
honActivator,XonshActivator
```

8. pytest 测试与coverage

支持的游戏场景

homework1所描述的场景

1. 每个游戏玩家都有一定数量的金币、宝物。有一个市场供玩家们买卖宝物。玩家可以将宝物放到市场上**挂牌**，自己确定价格。其他玩家支付足够的金币，可购买宝物。
2. 宝物分为两类：一类为**工具，它决定持有玩家的工作能力**；一类为**配饰，它决定持有玩家的运气**。
3. 每位玩家每天可以通过**寻宝**获得一件宝物，宝物的价值由玩家的运气决定。每位玩家每天可以通过**劳动**赚取金币，赚得多少由玩家的工作能力决定。（游戏中的一天可以是现实中的1分钟、5分钟、10分钟。自主设定。）
4. 每个宝物都有一个自己的名字（尽量不重复）。每位玩家能够**佩戴**的宝物是有限的（比如一个玩家只能佩戴一**个工具和两个配饰**）。多余的宝物被放在存储箱中，不起作用，但可以拿到市场出售。
5. 在市场上**挂牌**的宝物必须在存储箱中并仍然在存储箱中，直到宝物被卖出。挂牌的宝物可以被**收回**，并以新的价格重新挂牌。当存储箱装不下时，运气或工作能力值最低的宝物将被系统自动回收。
6. 假设游戏永不停止而玩家的最终目的是**获得最好的宝物**。（不停提升运气 提升共工作能力，获得好的饰品和最好的工具 循环）

额外实现的更符合游戏完整逻辑的功能

7. 玩家查看个人主页(基本信息)
8. 玩家查看个人存储箱
9. 玩家查看个人操作历史，玩家可以通过输入operation_type查看自己进行的操作历史。
10. 玩家售卖物品支持改价，当有treasure被某玩家挂牌出售时，支持玩家直接修改该宝物价格，而不是通过先回收再重新出售
11. 玩家查看个人出售列表
12. 玩家初次进入游戏时的注册操作，同时记录玩家登陆密码
13. 玩家登陆进入游戏
14. 玩家忘记密码时，回看个人密码。

数据库设计

需求分析：了解用户需求，确定软件的基本功能

用户需求

用户：游戏玩家

用户属性：

1. 一定数量的金币
2. 随身携带：一个工具类宝物和两个运气类宝物，随身携带的宝物**决定**着当天玩家在自动寻宝的时间段过程中获得的收益。
3. 存储箱：一定数量的宝物(分位工具类)
4. 宝物：
    1. 类型：宝物功能用途
    2. 等级：宝物等级决定了用户某种事件的结果。

用户操作：

Type1: 系统自动帮助玩家进行的,即以游戏中的"天"为单位。

1. 寻宝(afschedule定期巡航)：

    details:寻宝根据玩家佩戴的饰品进行，使用randint在level处保证随机性，对于这一个可多人在线的游戏而言，系统后台的宝物数据库的个数/种类对每个登陆游戏玩家是不透明的，但对每个玩家而言，系统后台的宝物数据库为不可更改的同一宝物数据库。

2. 打工：(afschedule定期巡航)

    根据佩戴的物品的labour属性帮助玩家打工

> 说明：为了使寻宝和打工操作更符合真实的游戏逻辑，默认flask 框架所构建的web应用app.py是全天候运行，即游戏后台服务器一直处于后台状态，所以系统自动帮助玩家进行的活动，默认只要flask run，即遍历players列表的所有玩家，自动帮助玩家进行打工和寻宝操作。**【无论玩家是否真实登陆游戏查看或进行相关操作】**

Type2: 系统自动帮助玩家进行的,即系统帮助玩家的"自动"丢弃机制。

- 回收

  当玩家**进行寻宝**前，判断玩家存储箱的大小，如果存储箱的大小超过maxlen，则自动回收level最低的宝物，这里working宝物和fortune宝物一起被排序。

  当玩家想在市场上确定购买宝物时，系统自动帮助玩家进行不必要的宝物的回收操作。

  同时，当玩家想将宝物从市场上回收时，也有此操作。

  总结：当玩家进行所有涉及到存储箱数量改变的操作时，均应该使用此操作。

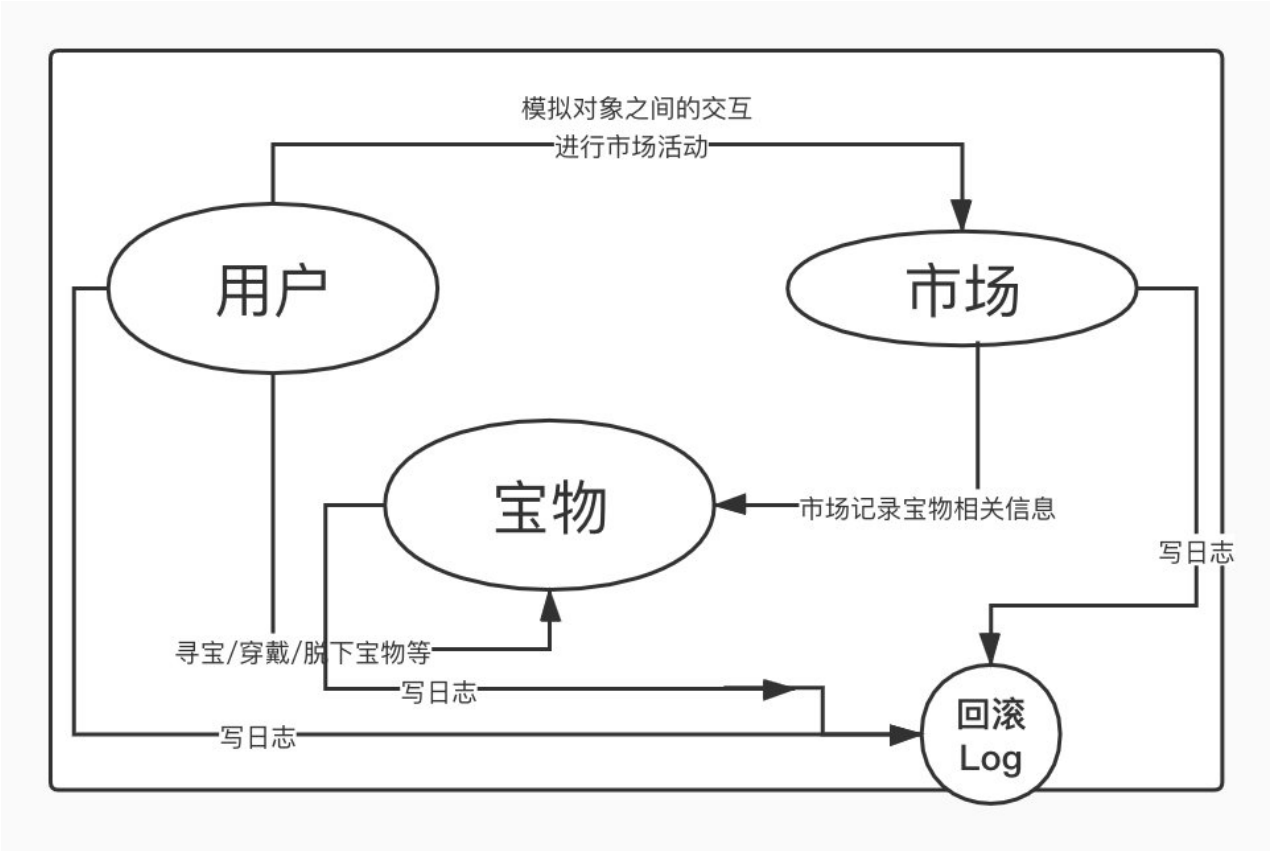
Type3: 玩家自主进行的操作(opearation)

1. 玩家浏览自己的个人基本信息(个人主页)
2. 玩家浏览自己的存储箱
3. 玩家浏览市场
4. 玩家在市场挂牌售卖宝物
5. 玩家在市场对售卖宝物进行改价
6. 玩家将宝物从市场上回收
7. 玩家查看个人主页(基本信息)
8. 玩家查看个人存储箱
9. 玩家查看个人操作历史，玩家可以通过输入operation_type查看自己进行的操作历史。
10. 玩家售卖物品支持改价，当有treasure被某玩家挂牌出售时，支持玩家直接修改该宝物价格，而不是通过先回收再重新出售
11. 玩家查看个人出售列表
12. 玩家初次进入游戏时的注册操作，同时记录玩家登陆密码
13. 玩家登陆进入游戏
14. 玩家忘记密码时，回看个人密码。

概念模型设计：确定数据库需要记录的信息(Raw Info)

从上述用户需求可得，确定Object即Nosql类型数据库的collection类型如下：

1. 宝物(treasure)
2. 宝物库(Box)
3. 用户(player)
4. 市场(market)
5. 历史(history)

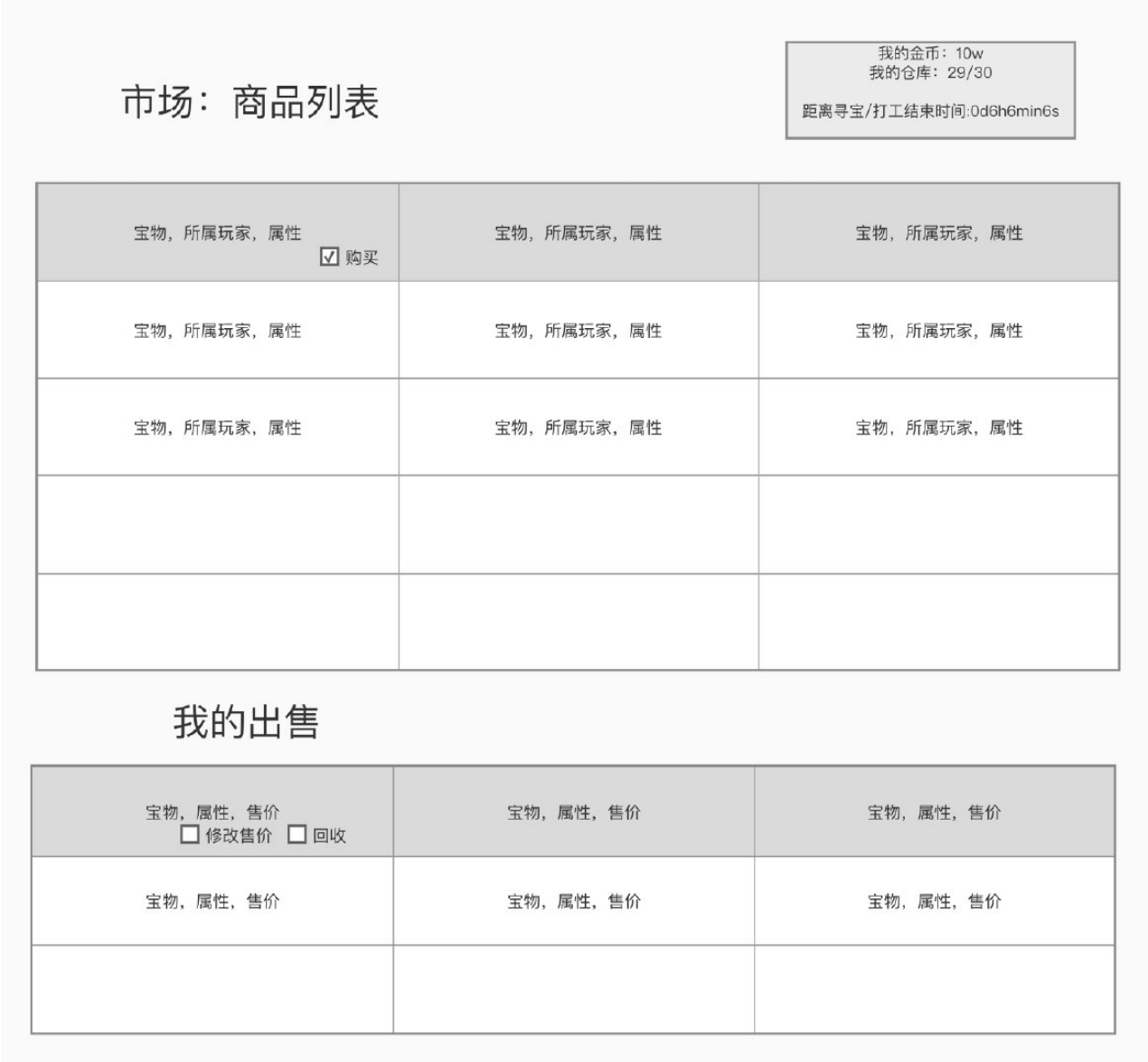


模拟UI

模拟用户操作而模拟用户访问频率，得到初始的DB design Schema，通过**模拟CRUD次数**对数据库结构进行调整。

用户界面UI

市场列表UI



UI模拟结论,按照访问频率从高到低排列

正常玩家的操作：

Most Frequent:

用户登陆，即进入用户主界面，所以Player这个collection是用户访问最多的文档集，由于用户访问的频繁行，所以要尽可能多的涉及用户有关的常用信息嵌入到player 这个collection的单个对象中，便于用户需要时夺得及时的访问，以及减少数据库其他文档集的频繁访问，以增强用户体验。

查看仓库

用户点击主页面的仓库按钮，即可查看仓库与更改takeaway等。

与市场有关的交易操作

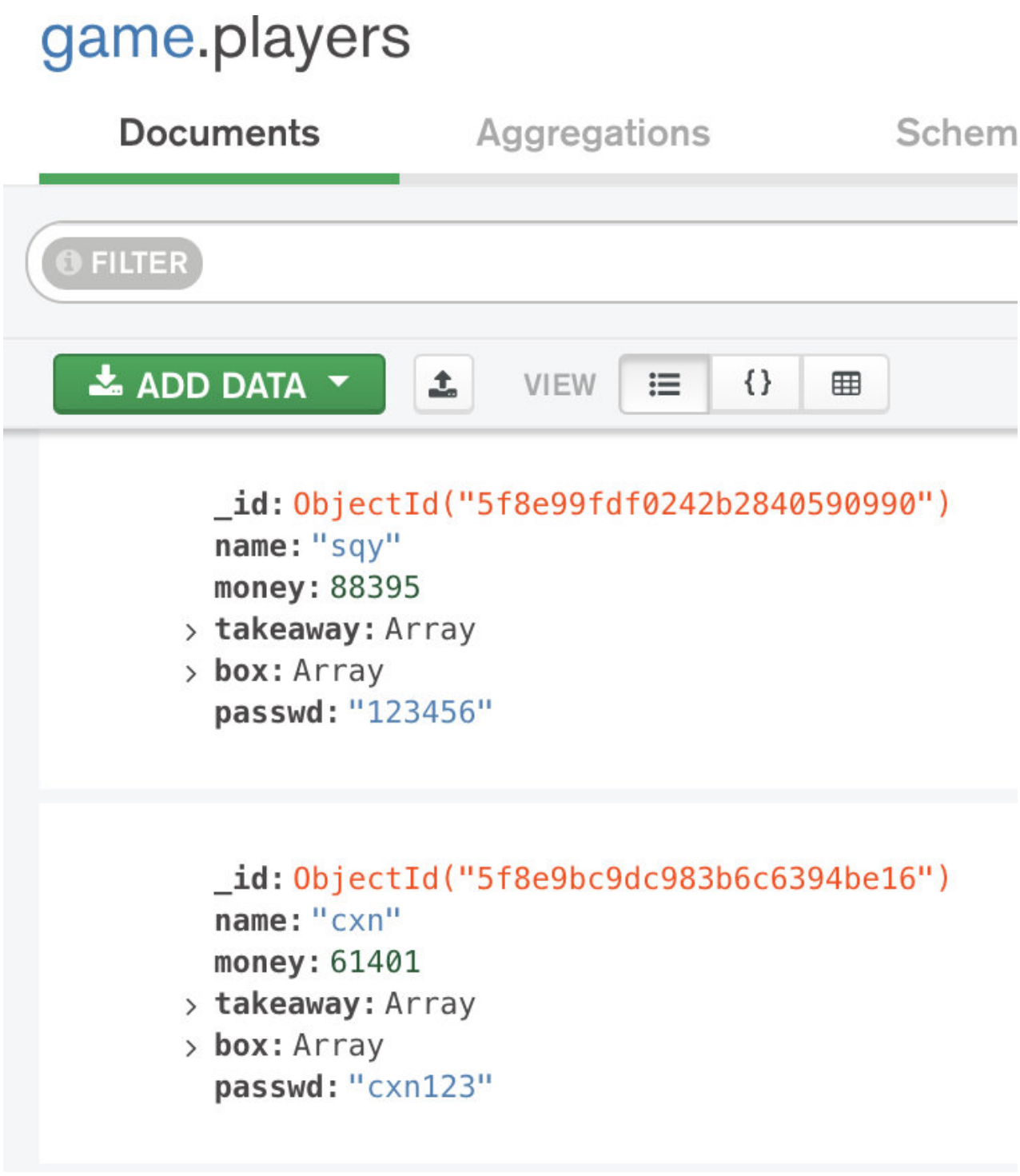
通常而言，用户一般在浏览完自身信息和存储库信息后，才会去选择去市场进行相关交易，以增强自身能力，防止存储箱溢出等。

Raw schema

1. Player{OID,storage[Treasure:{OID},],takeaway{OID1,OID2,OID3}attribute:{money, working,fortune}，myself[working:[],fortune:[]]}
2. Market{Treasure:{[player:OID],OID,price,attribute:{working,fortune}}}
3. workingtreasure(寻宝):{OID,attribute:{working,fortune}}
4. Fortune treasure(寻宝):{OID,attribute:{working,fortune}}
5. Box:{ownername:[]}

在实际编写过程中最终的数据库设计

players


game.players
Documents
Aggregations
Schem
FILTER
ADD DATA
VIEW
{}
_id: ObjectId("5f8e99fdf0242b2840590990")
name: "sqy"
money: 88395
takeaway: Array
box: Array
passwd: "123456"
_id: ObjectId("5f8e9bc9dc983b6c6394be16")
name: "cxn"
money: 61401
takeaway: Array
box: Array
passwd: "cxn123"

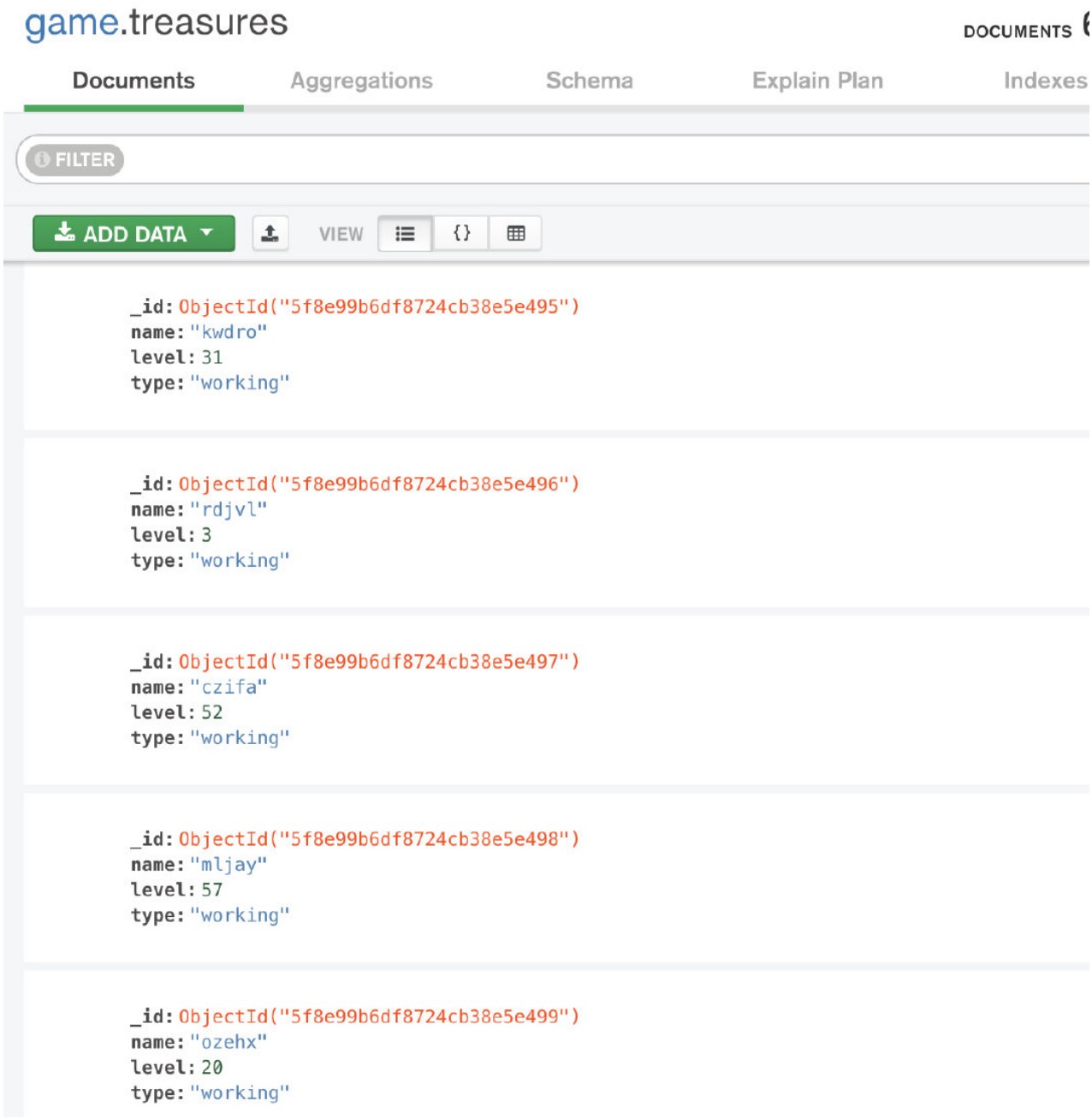
```
_id: ObjectId("5f8e99fdf0242b2840590990")
name: "sqy"
money: 88395
takeaway: Array
  0: Object
    name: "rope"
    level: 10
    type: "working"
  1: Object
  2: Object
box: Array
  0: Object
  1: Object
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object
  8: Object
  9: Object
  10: Object
  11: Object
  12: Object
  13: Object
  14: Object
  15: Object
  16: Object
  17: Object
  18: Object
passwd: "123456"
```

player collection 插入的document内容如下：

- username: string 类型，为玩家用户名，不可重复，其中的不可重复性由 mongoDB 提供的`players.create_index([("name", pymongo.ASCENDING)], unique=True`保证。
- money：int 类型，该玩家当前拥有的金币数量
- Takeaway:array 类型，其中array中存放的是玩家随身携带的treasure类型的document.
- Box:array类型，其中array中存放的是玩家随身携带的treasure类型的document.
- Passed：str类型，为玩家的登陆密码

treasure

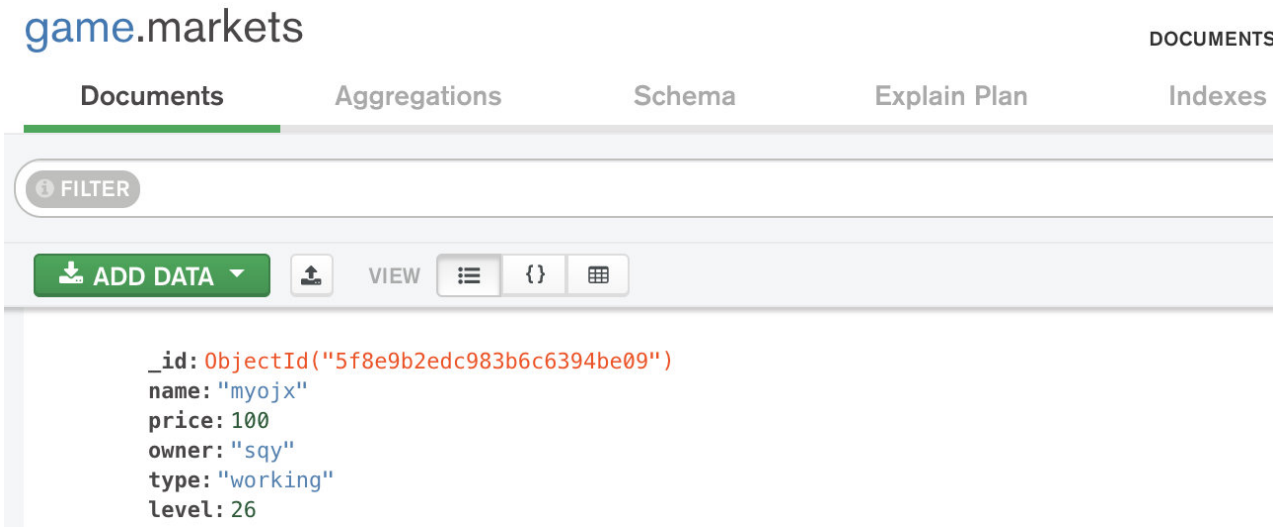
treasure文档集，相当于系统默认的宝物库，对于每个初次注册登陆的玩家，为保证游戏起始的公平性，宝物库相同，【可以模拟玩家“出生”的偶然性，模拟不同玩家有不同质量的宝物库~游戏后续升级的🍑之一】



宝物库，只记录宝物的最基本信息，即宝物名称，注意，这里将宝物名称作为唯一的宝物索引不同的宝物。我自行编写`generate_basic_db.py`文件使用random函数随机生成的宝物名称。

- Name:str宝物名称
- Level:宝物等级
- Type:working 即为工具类宝物，fortune即为玩具类宝物。

markets

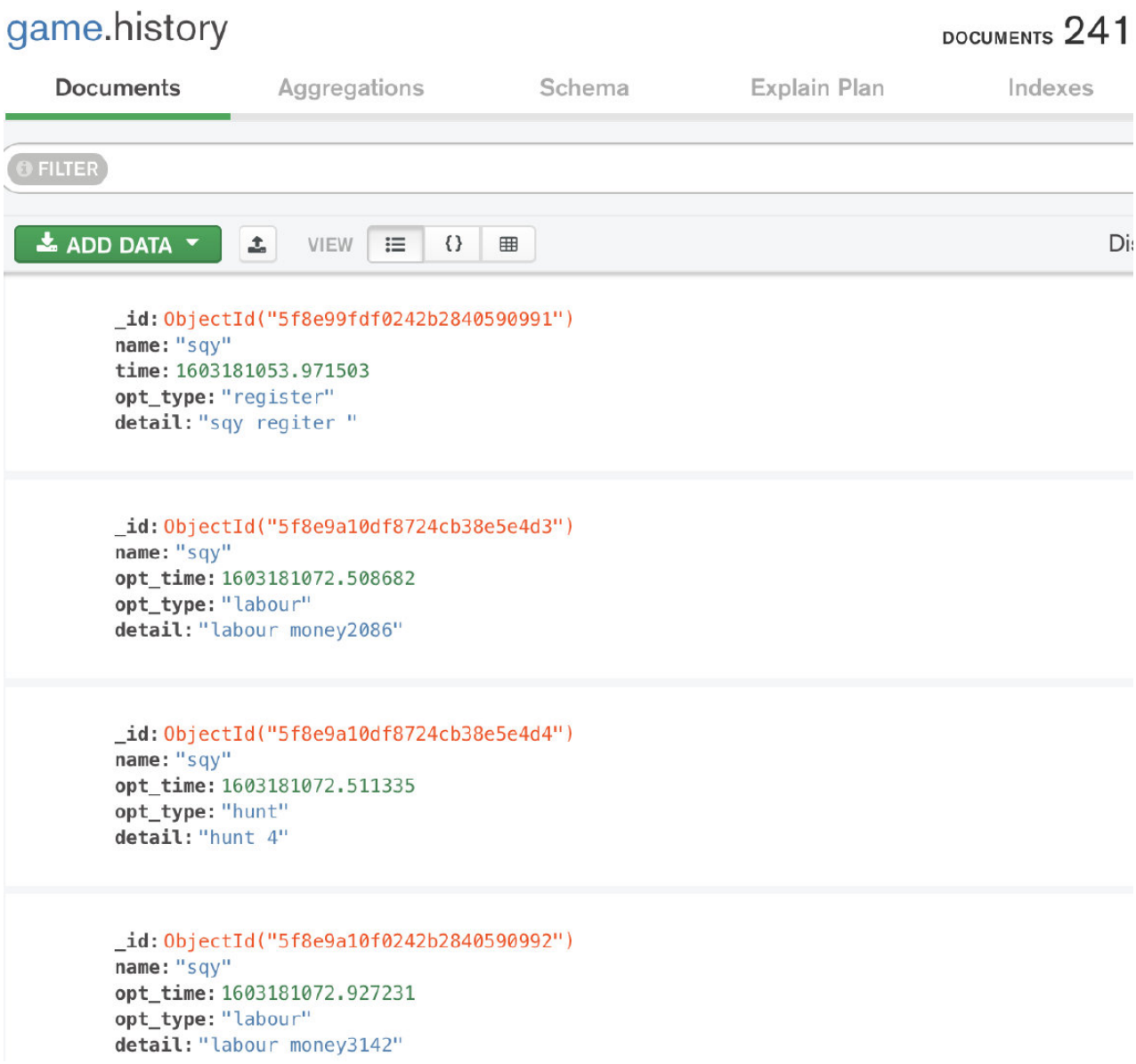


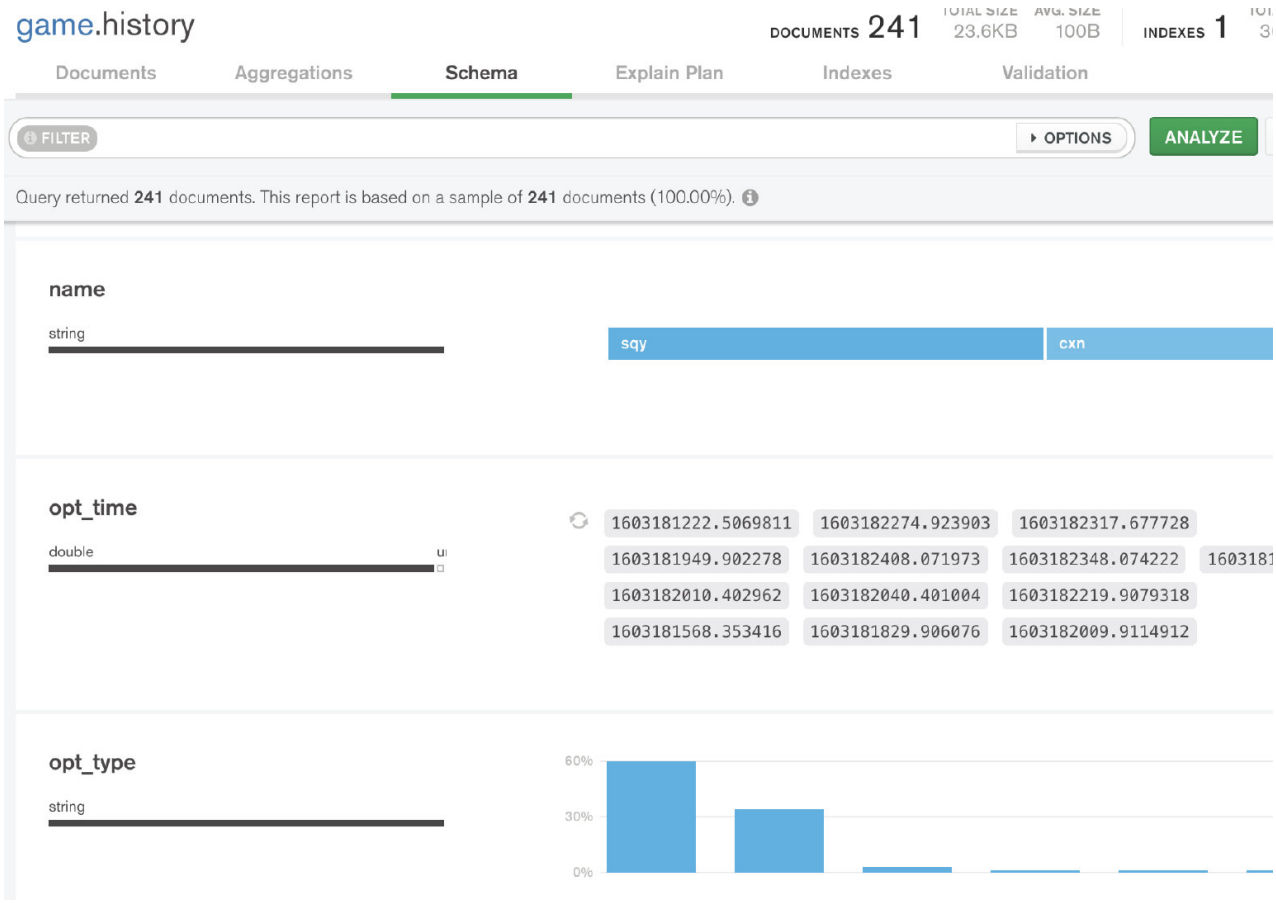
当玩家准备在市场上售出时，向markets 集合插入一条信息，如上图所示。

- Name:str类型 记录宝物名称
- Price: int类型，记录宝物出售的价格，功能支持宝物售卖者进行改价。
- Owner:str类型，记录出售者的名称。更符合正常游戏逻辑的设计。

  因为我在初始化宝物库，即treasure collection的时候，只向其中插入了62个document，且所有玩家在第一次寻宝(hunt)之前，自身的幸运等级和工作能力等级被初始化相同，所以，有很高的概率在游戏的初始阶段，不同玩家寻到的宝物是有大比例重叠的。所以需要在市场信息中记录出售玩家以**区分同名宝物**，玩家在后续的购买功能时，需要以(treasurename,sellername)来唯一指定购买的是哪一款宝物。
- level:int类型，记录该宝物的等级。更符合正常游戏逻辑的设计。

history


game.history
DOCUMENTS 241
Documents
Aggregations
Schema
Explain Plan
Indexes
FILTER
ADD DATA
VIEW
{}
_id: ObjectId("5f8e99fdf0242b2840590991")
name: "sqy"
time: 1603181053.971503
opt_type: "register"
detail: "sqy regiter "
_id: ObjectId("5f8e9a10df8724cb38e5e4d3")
name: "sqy"
opt_time: 1603181072.508682
opt_type: "labour"
detail: "labour money2086"
_id: ObjectId("5f8e9a10df8724cb38e5e4d4")
name: "sqy"
opt_time: 1603181072.511335
opt_type: "hunt"
detail: "hunt 4"
_id: ObjectId("5f8e9a10f0242b2840590992")
name: "sqy"
opt_time: 1603181072.927231
opt_type: "labour"
detail: "labour money3142"



history文档集的设计：

- name: str类型，记录操作的玩家
- Opt_time:float类型，由time()库函数获得，记录操作发生的时间
- opt_type:str类型；操作类型：包括玩家主动进行的操作和系统帮助玩家自动进行的操作
- Detail：操作细节：str类型

对比raw schema与final schema

问题1:是否需要额外新建立box collection?

我在一开始设计的时候，对于玩家的宝库仓库放在哪里纠结了很久。在raw schema的设计时，准备新增一个box collection。打算只在玩家player中嵌套一个"box"key,它的value指向box collection中的索引，但是，经过后面游戏具体设计实现发现，自动寻宝功能//穿戴，脱下宝物功能//买卖功能等，即所有游戏的核心功能，都围绕着玩家box的更改进行。所以不惜牺牲player collection的存储size略微庞大，来获得这种对于player box的交互的快速，即减少CRUD的时间。

同理，对于玩家穿戴的操作，涉及到玩家"takeaway"这个key所对应的value的更新，但由于takeaway的上限容量更小，所以直接将takeaway嵌套入玩家collection。

问题2:是否设置玩家自身的属性ability当作一个Key?

我起初，想在玩家的document中增加,"ability"Key，即玩家的能力属性，由游戏的实现逻辑可得，能力属性决定了玩家寻宝和打工时的收获。但也由于游戏的实现逻辑较简单，即玩家实时的ability并没有即时的显示在玩家的个人信息主页，所以玩家自身的属性ability当作一个Key，并不是有十分的必要性。因为如果在游戏逻辑没有要求的情况下，每次玩家在进行wear/unwear操作的时候，都需要对此进行update.

但是，如果将游戏逻辑补充完整到，在玩家的个人信息主页，实时的显示玩家的ability，那么，abilityKey就很有必要作为玩家的一个Key，也是通过additional information的存储以提高用户体验。

问题3:我为什么没有选择增加"on-sale-flag"Key?

游戏设计的逻辑要求，需要保证玩家自己挂牌在市场上售卖的宝物，不能供玩家穿戴时选择，即玩家穿戴时，要看自己的存储库，去除掉那些onsale的宝物。同时，由于玩家也需要直接看到自己哪些物品挂在市场（或者说哪些物品在背包中但其实被锁定住因为挂在市场），所以建立on-sale-flag属性（这样不需要到market中遍历sell是自己的物品才能知道哪些是我已挂在市场的宝物，而且便于维护）。但是，对于on-sale-flage属性，若选择在所有宝物库初始化时置为0，不符合宝物被用户获得才有可出售性的逻辑，所以，应该选择在寻宝时，被加入进玩家宝物库(box)时维护。但是，box中给每个物品贴"on-sale-flag"标签，需要遍历box获得我的出售。但是，box中物品也是过多，不方便查找属性，且还需要遍历整个box字典才能知道那些东西在市场或者不在市场上。

问题4:如何高效获得“我的出售”

问题3讨论了我为什么没有选择增加“on-sale-flag",为了维护游戏逻辑的完整性，我需要记录我的出售。在市场出售列表中，因为属于不同玩家的同名宝物售卖的可能性，我该如何获得我的出售呢?

一种选择方案是，在player document中再额外增加一个冗余信息。

另一种选择方案是，在market的document中，对于每一个宝物，额外增加一个seller的Key，作为除了name of treasure以外的secondary Index。

对比这两种选择方案，第一种需要通过"player_name"作为索引搜索B树这个mongoDB 的索引结构。在每一次出售时，修改working_treasure_onsale和fortune_treasure_onsale数组，在每一次挂牌回收时，需要遍历这两个数组进行删除。

第二种通过'treasure_name_onmarket'和'seller_name'这一个二元的索引结构，主要还是“seller_name"这个索引，获得“我的出售”。在挂牌回收时，也是同样的操作。

对比两种方案，显然值得选择第二种。

问题4:为什么不单独区分working treasure 和fortune treasure?

为了系统丢弃时考虑。

系统自动帮助玩家回收宝物，需要将working treasure和fortune treasure作为同等权的宝物丢弃，所以在这里，我没有将working treasure 和fortune treasure分开，考虑到玩家可佩戴的不同种类的宝物的最大值和系统可寻宝宝物库的个数，可以赋不同权，在这里等权的情况下，单独区分working treasure和fortune treasure没有太大的意义。

问题5: 为什么要存储冗余(重复)信息

玩家可读可操作自由操作sell为自己的object或进行buy操作。由于有查看market操作，需要快速向玩家展示所有market中的信息，故组织成good ，price，sell格式，当然也可以只考虑每个good在原来user的treasure的_id，单个人认为这样对于如果程序出错，数据库出现问题时维护起来会很累，因为需要反向找到_id，再对market进行操作，所以组织成这样，而且对于仅访问（如查看 毕竟查看操作必然比交易操作多）market某个object，不需要去直接对用户个人属性进行访问，防止造成不必要的数据损失。

问题6:嵌套文档的索引均省去了"_id"的存放，而是使用名称

在理论部分，可以使用"OID"指向不同集合的document，但是在操作的具体实现中，我发现。"_id"一般都是用hash生成的不是特别readable的值，所以变换思路，有比较易于想到的name作为索引的值。

同时，在问题5中，可以知道在数据库设计时，以不同的key作为index，可以提高CRUD的效率，

程序设计与游戏最终支持的功能展示

为了保证说明文档的简介性，只保留实现思路，不保留代码。

后端

Welcome to the game

register

用户注册，不同玩家不能拥有相同的用户名，以"name"作为unique索引，避免重复。

注册初始化时，默认玩家working_level=fortube_level=10,也是什么都不佩戴时的level。

- 玩家姓名： cxn
- 玩家的存储箱：

物品名称	物品属性	物品类型

- 玩家的随身携带：

物品名称	物品属性	物品类型
rope	10	working
dice	9	fortune
diya	9	fortune

- 玩家的金钱： 1000

homepage

用户查看主页，使用render_templates,前端采用Jinjia2渲染。

- 玩家姓名： cxn
- 玩家的存储箱：

物品名称	物品属性	物品类型
tsgrx	28	fortune
myojx	26	working
kbdhl	25	working
uwmkb	31	working
uwmkb	31	working
kwdro	31	working
kbdhl	25	working
pbzfe	26	fortune
kbdhl	25	working
tsgrx	28	fortune
qwndo	26	working
tsgrx	28	fortune

- 玩家的随身携带：

物品名称	物品属性	物品类型

物品名称 物品属性 物品类型

rope	10	working
dice	9	fortune
diya	9	fortune

- 玩家的金钱： 13064

look_history

用户查看历史，使用render_templates,前端采用Jinjia2渲染。

cxn 1603260221.025433 hunt hunt 2

cxn 1603260221.467038 labour labour money5038

cxn 1603260221.4696362 hunt hunt 3

cxn 1603260323.9668732 labour labour money6074

cxn 1603260323.9678478 hunt hunt 3

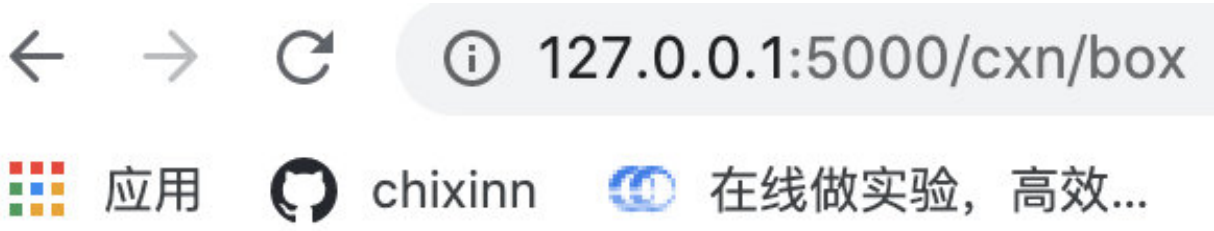
cxn 1603260334.406998 labour labour money7074

cxn 1603260334.407539 hunt hunt 0

cxn 1603260383.962097 labour labour money8092

Look_box

用户查看存储箱，使用render_templates,前端采用Jinjia2渲染。



cxn玩家的存储箱

物品名称 物品属性 物品类型

tsgrx 28 fortune

myojx 26 working

kbdhl 25 working

uwmkb 31 working

uwmkb 31 working

kwdro 31 working

kbdhl 25 working

pbzfe 26 fortune

kbdhl 25 working

tsgrx 28 fortune

qwndo 26 working

tsgrx 28 fortune

myojx 26 working

uwmkb 31 working

kwdro 31 working

uwmkb 31 working

kbdhl 25 working

kbdhl 25 working

qwndo 26 working

kwdro 31 working

unwear

用户脱下身上takeaway着的某物品。毕竟takeaway有上限，先脱下才能再穿上。

从身上脱下成功diya 宝物

sell：首次出售

大家伙都来的市场

挂牌成功

您当前拥有26132元

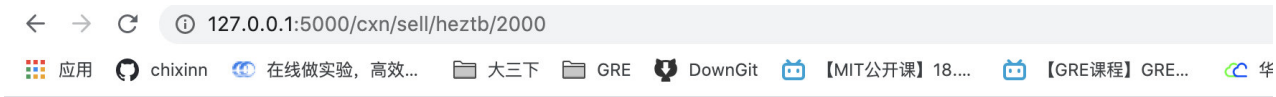
物品名称 物品等级 物品类型卖家 售价
kbdhl 1000 cxn working 25

Sell:改价

用户将宝物挂牌出售。

你还在卖着哦，改价成功！

Sell_Error_detect



咋搞的啊，又输入错宝物名字了,宝物不在存储库中是不能售卖的哦

Look_mysale

用户查看我的出售，使用render_templates,前端采用Jinjia2渲染。

大家伙都来的市场

欢迎查看我的售卖～

您当前拥有元

物品名称 物品等级 物品类型卖家 售价
kbdhl 4000 cxn working 25
myojx 10 cxn working 26

withdraw

用户将某宝物挂牌回收。

收回宝物 myojx 成功

wear

佩戴 myojx 宝物成功

buy

玩家 cxn 支付宝到账 4000

再次查看市场时，该宝物已从市场上删除。

大家伙都来的市场

您当前拥有6209元

物品名称 物品等级 物品类型卖家 售价

前端以index.html为例子

index.html

```
{% extends 'base.html' %}
{% block content %}
<ul>
    {% for user in users %}
    <li><p>
        玩家姓名:
        {{user.name}}
    </li>
    <li><p>
        玩家的存储箱: <br>
        <table>
        <tr> <th>物品名称</th> <th>物品属性</th>  <th>物品类型</th></tr>
        {% for item in user.box%}
            <tr>
                <td>{{item.name}}</td>
                <td>{{item.level}}</td>
                <td>{{item.type}}</td>
            </tr>
        {% endfor %}
    </table>
    </li>
    <li><p>
        玩家的随身携带: <br>
        <table>
        <tr> <th>物品名称</th> <th>物品属性</th>  <th>物品类型</th></tr>
        {% for item in user.takeaway%}
            <tr>
                <td>{{item.name}}</td>
                <td>{{item.level}}</td>
                <td>{{item.type}}</td>
            </tr>
        {% endfor %}
    </table>
    </li>
    <li><p>
        玩家的金钱:
        {{user.money}}
    </li>
    {% endfor %}
</ul>
{% endblock %}
```

因为我的前端也是为了做这个实验现学的，所以设计的界面很简陋。

即Jinjia2将我后段传入的content(这里是User)内容传入，嵌套进html语句中，基于的base是base.html，为了使templates有统一的版式。

程序编写核心点：

- 设计到玩家box更改操作，只要是使box_size增加的函数，都要检查box容量上限，即使调用自动自动丢弃函数。所以，有可能前后两次update的操作。
- 同时，还有一处比较关键和核心的地方时，一次update后的索引重建：

```
if len(box1) >= 10:
    recovery_treasure(username)
box = player['box']     # 回收宝物后需要重新索引宝箱数据，这个很关键，我一开始没想到
box.append({"name":treasure,"level":treasure_to_buy["level"],"type":treasure_to_buy["type"]})
players.update_one({"name": username}, {"$set": {"box": box}})
```

- 挂牌出售时，支持玩家改价的功能很重要，否则对于mongoDB来说会显示两条玩家信息
- 还有很多程序设计的细节，要点展示在代码注释。
- **自动寻宝/打工的整体逻辑**借鉴了这个代码：APS Scheduler自动巡航

Test 测试

因为一开始我的pytest一直会有找不到fixture的报错，找了一圈也没有找到相应的解决问题，导致pytest一直无法运行。

后来又看了一下github上的JSON_EXAMPLE,对`setup.cfg`重写后，解决了问题。

测试结果如下

```
(venv)  ✘ chixinning@mymac ~/Desktop/hw1/HuntFinal pytest
============================= test session starts ==============================
platform darwin -- Python 3.8.6, pytest-6.1.1, py-1.9.0, pluggy-0.13.1
rootdir: /Users/chixinning/Desktop/hw1/HuntFinal
collected 1 item

test/test_opt.py .                                                       [100%]

============================== warnings summary ================================
../Database-Management-System-master/python_flask/test/test_opt.py:13: 1 warning
test/test_opt.py: 1311 warnings
  /Users/chixinning/Desktop/hw1/Database-Management-System-master/python_flask/test/te
st_collection_names instead.

test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
  /Users/chixinning/Desktop/hw1/HuntFinal/action/opt.py:132: DeprecationWarning: inser
    history.insert({"name":username,"time":time(),"opt_type":"login","detail":username

test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
test/test_opt.py::test_opt_get
  /Users/chixinning/Desktop/hw1/HuntFinal/action/opt.py:87: DeprecationWarning: insert
    history.insert({"name":username,"time":time(),"opt_type":"register","detail":usern

-- Docs: https://docs.pytest.org/en/stable/warnings.html
======================= 1 passed, 1324 warnings in 3.82s =======================
```

总结

这次的数据库作业实践，让我第一次完整的体会到了web应用的构建流程，并以此应用为驱动，获得了很多web应用开发的知识。使刚刚结束的计算机网络课程中的http协议，前端、后端、数据设计、测试等，有了脱离于纸的理解与认识，同时对python语言本身也得到了熟悉，也逐渐学会去看一些报错信息。

如最近一次报错信息：

`NoneType' object is not subscriptable`我之前一直以为是我选择的数据结构不能索引的问题，仔细排查以后，才发现是索引的空对象。

同样，为了不使我的web应用看起来过于简陋，我再一次在菜鸟教程上学习了html语言，并第一次尝试自己在html中使用unorderedlist , table等语言。

同时，我在web应用开发准备初期，在网上浏览flask应用开发demo时，首次接触到了Jinjia2渲染，并实现了后端到前端的传值和搭建。

上面是我在准备初期的一些学习书签。

不过，纸上得来终觉浅，绝知此事要躬行，这次实验使我收获颇丰，获益匪浅。