《当代数据管理》实验报告

姓名：郑逸潇	**学号：10192100406**
项目名称：《寻宝游戏》	**编号：Project1**

数据库设计

分析

1、分析宝物和宝物类型文档集是否区分

每个不同的宝物(item)由于状态会不相同(佩戴中/在背包/挂牌中)，因此每个宝物(item)都需要一个文档来存储，而宝物的种类则用不同的文档集来存储。

宝物的信息(名字，简介，功能等)需要考虑占用的存储和查询效率，来决定是将宝物的信息存在 item 中还是 item 文档只存储状态和一个指向宝物种类的序号(ObjectId)

个人选择是将宝物信息直接存在了每个 item 中，因此消耗的存储空间会变大一些，但是无需每次每个物体要取其在宝物种类中的 id，这种做法较为简易，减少了一次与数据库的交互。

2、分析用户和宝物文档集是否需要分离

若宝物文档嵌入在用户文档中，那么每次进行宝物的增加、减少时需要频繁的扩大、减小文档，且商城购买时需要遍历所有用户取出其正在挂牌中的 item，较为繁琐，个人不建议嵌入。

个人想过以下几种方案：

1. 宝物文档嵌入用户文档，但是维护另一个文档集 market_item，其内容为一个 ObjectId 指向用户的宝物文档和价格属性。
2. 宝物文档不嵌入用户文档，将价格属性直接加入 item 文档中，然后用户文档集中添加 tool 和 ornament 属性，内容分别是工具和饰品的宝物 ObjectId，但在 item 文档中不添加其属于的用户 ObjectId。
3. 宝物文档不嵌入用户文档，将价格属性直接加入 item 文档中，在 item 文档中添加其属于的用户 ObjectId(记为 uid)，但在用户文档中不添加任何关于 item 的信息。(由于我们可以在每次操作的时候将所有属于此用户的文档全部拿出来，理论上是可以实现所有功能的)
4. 宝物文档不嵌入用户文档，将价格属性直接加入 item 文档中，然后用户文档集中添加有关 item 的属性，内容是宝物的 ObjectId，在 item 文档中添加其属于的用户 ObjectId(记为 uid)。
5. 宝物文档不嵌入用户文档，将价格属性直接加入 item 文档中，然后用户文档集中添加有关 item 数量的属性，内容是宝物的数量，在 item 文档中添加其属于的用户 ObjectId(记为 uid)。

为了商城的实现简易以及需要联动改动的操作减少，个人觉得应该需要将宝物(item)文档与其所对应的用户(user)分离开，因此排除第一种方案；

因而考虑第二种方案，考虑 item 文档不加所属用户 id 而用户文档添加拥有 item id，此方案在“购买”操作时需要同时更新双方用户的数据库，此时需要进行两次数据库操作，且不为原子操作，所以可能会发生错误，即在更新完其中一个 user 时，服务器宕机使得第二个 user 并没有更新上，此时某个 item 会处于同时属于两个人或不属于任何人状态**且有时候用户难以发现**，而且此宝物可能会流传很久后才被发现，无日志的话不可追溯，**引发安全问题**。

因而考虑第三种方案，只在 item 侧添加属于 user 的属性，此方案在“购买”时不会出现 item 不会出现第二种方案的 bug(若更新为原子操作)，若单纯是更改金币的时候出现 bug 用户可以察觉，也可以反馈。但是此方案需要频繁访问 item 数据库，取出所有与 user 相关的 item 在很多操作的时候并不是必要的，**效率较低**。

因而考虑第四种方案，此方案可以解决“购买”宕机问题，因为可以对 user 和 item 双方都进行一遍判定来确保此物确实在此时属于此用户，可以防止出 bug 的物品流传出去难以追踪(因为在出售物品的时候肯定要先判定此物品属于此用户才可以进行)，此方案可行。

再考虑第五种方案，此方案在 user 端只存数字，代表数量，而不存具体的 ObjectId，首先这种方案在文档存储方面肯定消耗较少存储空间，其次是在获取属于某用户的 item 时，可以对 item 的 belong uid 建立索引，这样可以直接获取相同 uid 索引的所有文档，对比第四种方案效率应该不会差(理论上应该是将第四种的文档种的存储空间花费在了索引上，虽然没有具体了解是如何构建的索引，不知道索引消耗的空间大还是 ObjectId 放在 user 中大，但是效率应该也好一点)，而且这种方案在编写代码的时候也比较简单，只需更改数字，而不需要扩大 or 缩小数组。

因此我选择的是第五种方案，简单且不易错，当然咱们这种小游戏应该不会出大问题，因此所有方案都可以试一试。

文档集(collections)

用户(user)

```
{
    'username':username,
    'password':password,
    'gold_num':xxxx,
    'work_efficiency':xx,
```

```
        'lucky_value':xx,
        'wear':{
                'tool_num':x,
                'ornament_num':x,
                'totipotent_num':x,
        },
        'backpack':xx,
        'auto_clean':0/1,
        'auto_work':0/1,
        'finish':0/1,
}
```

物品(item)

```
{
        'buid':xxxx,
        'name':itemname,
        'grade':1/2/3/4/5/6,
        'info':'xxxx',
        'type':'tool'/'ornament'/'totipotent',
        'work_efficiency':xx,
        'lucky_value':xx,
        'state':'wear'/'backpack'/'onsale',
        'price':xxx,
}
```

物品种类(item_type)

```
{
        myid = xx,
        name = itemname,
        prob = xx,
        grade =1/2/3/4/5/6,
        info = 'xxx',
        type = 'tool'/'ornament'/'totipotent',
        work_efficiency = xx,
        lucky_value = xx,
}
```

[hint] 需要在 item 文档的 buid 上建立索引，提高查找效率

```
db.item. createIndex({"buid":1})
```

数据库 CRUD

插入 user/item 文档时直接构建一个 python 字典 insert_one 即可

```
db.user.insert_one(user)
db.item.insert_one(item)
```

更新 user 佩戴/取下/丢弃背包中物品时使用如下情况：

```
# 佩戴宝物
db.user.update_one({'_id': ObjectId(uid)},
                   {'$inc': {
                    'work_efficiency': item['work_efficiency'],
                    'lucky_value': item['lucky_value'],
                    'wear.'+item['type']+'_num': 1,
                    }})
db.item.update_one({'_id': ObjectId(iid)}, {'$set': {'state':
'wear'}})

# 取下宝物
db.user.update_one({'_id': ObjectId(uid)},
                   {'$inc': {
                    'work_efficiency': -item['work_efficiency'],
                    'lucky_value': -item['lucky_value'],
                    'wear.'+item['type']+'_num': -1,
                    }})
db.item.update_one({'_id': ObjectId(iid)}, {'$set': {'state':
'backpack'}})

# 丢弃宝物(这里可以将丢弃 改成 把item的state改成invalid,便于回溯)
db.user.update_one({'_id': ObjectId(uid)}, {'$inc': {'backpack':
-1}})
db.item.delete_one({'_id': ObjectId(iid)})
```

将背包中最差的 num 个宝物删除使用如下情况（较难）：

```
# 删除背包最差的num个宝物（较难）
items = list(db.item.aggregate([
    {'$match': {'buid': user['_id'], 'state':'backpack'}},
    {'$project': {'sum': {"$add": ["$work_efficiency",
"$lucky_value"]}}},
    {'$sort': {'sum': 1}},
    {'$limit': num},
]))
```

```
# 将 items 删除
for item in items:
    db.item.delete_one({'_id': item['_id']})
# 更新 user
db.user.update_one({'_id': user['_id']}, {'$inc': {'backpack': -
num}})
```

购买时使用如下情况：

```
db.user.update_one({'_id': ObjectId(item['buid'])},
                   {'$inc': {
                       'gold_num': item['price'],
                       'backpack': -1,
                   }})
# 更新买家
db.user.update({'_id': ObjectId(uid)},
               {'$inc': {
                   'gold_num': -item['price'],
                   'backpack': 1,
               }})
# 更新 item 信息
db.item.update({'_id': ObjectId(iid)},
               {'$set': {
                   'buid': ObjectId(uid),
                   'state': 'backpack',
                   'price': 0,
               }})
```

出售回收类同，只需执行更新 item 信息即可

```
# 出售操作（只需设置宝物状态和价格即可）
db.item.update({'_id': ObjectId(iid)},
               {'$set': {
                   'state': 'onsale',
                   'price': price,
               }})

# 回收操作
db.item.update({'_id': ObjectId(iid)},
               {'$set': {
                   'state': 'backpack',
                   'price': 0,
               }})
```

工作/寻宝较为简单：

```
# 工作（由于金币我设置了上限，未了解 MongoDB 关于某一属性的范围，因此是直接 set）
db.user.update_one({'_id': ObjectId(uid)},
                   {'$set': {
                       'gold_num': user['gold_num']
                   }})


# 寻宝（更新用户信息，宝物插入见第一条）
db.user.update_one({'_id': ObjectId(uid)},
                   {'$inc': {
                       'gold_num': -10*times,
                       'backpack': 1,
                   }})
```

开启/关闭自动清理/自动工作也比较简单：

```
# 开启/关闭自动清理
db.user.update_one({'_id': ObjectId(uid)}, {
    '$set': {'auto_clean': 0}})

# 开启/关闭自动工作
db.user.update_one({'_id': user['_id']}, {
    '$set': {'auto_work': 1}})
```

JSON / HTTP 接口

[hint] 细节请查看 json-http.xlsx

URL(GET)	操作	参数（json）	返回值（json）

127.0.0.1:8000/test/user/	注册/登录/退出登录	{
		'username':xxx,
		'password':xxx
		}
			{
			'success'/'error':xxx
			}

127.0.0.1:8001/test/getall/	获取用户所有信息	无参，一般是获得session，因此需先登录	{
			'success'/'error':xxx,
			'wearitems':[item文档],
			'backpackitems':[item文档],
			}

127.0.0.1:8002/test/work/	工作	{
		'work':'auto'/'manual'/'once'
		}
		(设置自动手动或单纯工作一次，自动时无法进行工作一次操作)
			{
			'success'/'error':xxx
			}

127.0.0.1:8003/test/hunt/	寻宝	{
		'times':int
		}
		(设定不超过10)
			{
			'success'/'error':xxx,
			'item':item文档,
			}

127.0.0.1:8004/test/operate/	佩戴/取下/丢弃	{
		'f':'wear'or'backpack'or'discard',
		'iid':'宝物的id'
		}
			{
			'success'/'error':xxx
			}

127.0.0.1:8005/test/market/	浏览/购买/出售/回收	{
		'f':'view'or'buy'or'sell'or'retrieve',
		'iid':'宝物的id'(当f为view时无用),
		'price':'int'(仅当f为sell时有用)
		}
			{
			'success'/'error':xxx
			}

127.0.0.1:8006/test/settings/	设置（仅自动清理）	{
		'setting':'xxx', 'operate':'xxx'
		}
			{
			'success'/'error':xxx
			}

功能实现

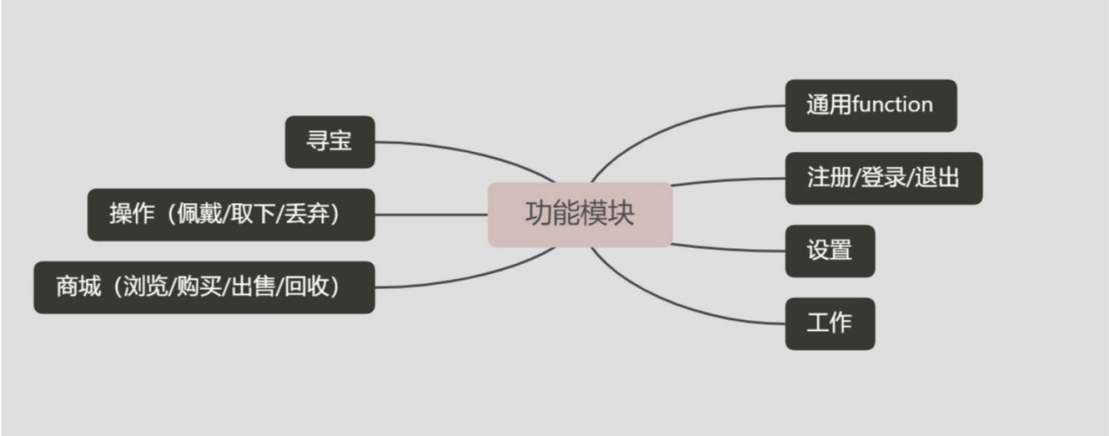
所有功能分为

(注册/登录/退出)
工作
寻宝
(佩戴/取下/回收)
商城(浏览/购买/出售/回收)
设置(仅包含自动工作)

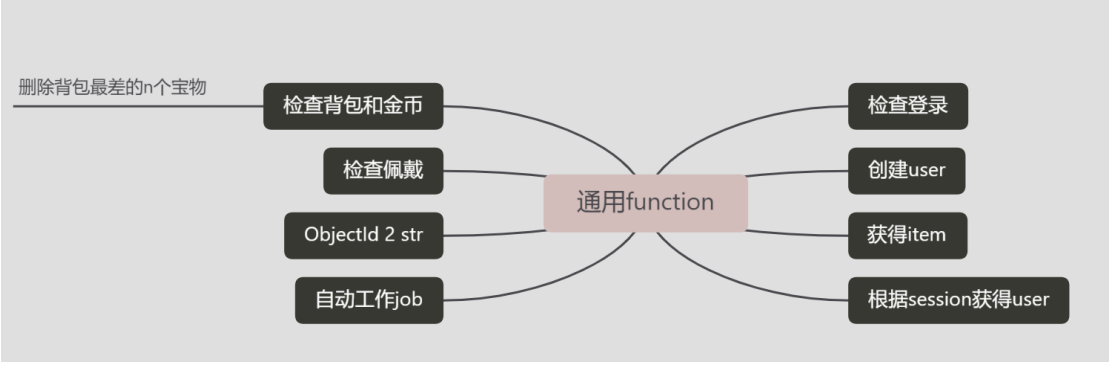
其中需要注意的点有：

- 写入数据库时需要 try 来防止并发请求
- 每个视图函数都要检查数据库是否连接
- 除登录/注册外每个视图函数执行前都需要检查是否已登录
- 购买/寻宝 时都需要考虑金币数量和背包剩余空间
- 由于 json 文件无法识别 ObjectId，因此需要有一个 ObjectId to str 的函数将每个文档转换

功能模块如下：



通用 function（便于视图函数调用）如下：



注册/登录/退出

注册

注册时需要考虑用户名 username 的问题，本项目以 username 作为不同用户的区分，因此需判断 username 是否在数据库中已存在。若无，则可以初始化 user 文档并写入数据库，然后返回到登录界面；若有则返回用户名重复，请修改用户名的提示。
其中写入数据库时需要 try 来防止并发请求，若并发请求，则打印报错，并返回服务器错误的提示

```
    # 判断是注册操作
    if f == 'register':

        # 用户名是否可用
        user = db.user.find_one({'username':username})
        if user is not None:
            return JsonResponse({'error':'用户名重复，请重新输入
'})

        # 编写文档，直接存入，密码未 hash
        user = create_user(username, password)

        # 向数据库插入数据 要 try 防止并发
        try:
            db.user.insert_one(user)

        except Exception as e:
            print('--- concurrent write error! ---')
            return JsonResponse(SERVER_ERROR)

        return JsonResponse({'success':'注册成功，请登录'})
```

登录

登录时需判断用户名和密码是否匹配，若无用户名 or 密码不匹配则输出“用户名或密码错误，请重试”。若用户名和密码匹配则设置 session，使服务器记住当前用户的浏览器。

```
    # 判断是登录操作
    elif f == 'login':

        # 对比数据库用户和密码，然后进入个人界面
        user = db.user.find_one({'username':username})
```

```
        # 用户名不存在或密码错误
        if user is None or user['password'] != password:
            return JsonResponse({'error':'用户名或密码错误，请重试
'})

        # 登录进入，且设置 session
        else:
            request.session['username'] = user['username']
            request.session['uid'] = str(user['_id'])
            return JsonResponse({'success':'登录成功'})
```

退出登录

```
    # 判断是退出登录操作
    elif f == 'logout':
         # 删除 session 值
        if 'username' in request.session and 'uid' in
request.session:
            del request.session['username']
            del request.session['uid']

            return JsonResponse({'success':'退出登录成功'})
        else:
            return JsonResponse({'error':'您尚未登录，请登录'})
```

将设置好的 session 删除跳转到登录界面即可

工作

工作是最简单的一部分，只需接收到前端的请求，将 session 中记录的 user 从数据库中读取出并修改其金币数量再写回数据库即可。**需要 try**

后续加入了自动工作功能，我使用了 apscheduler 来实现定时任务，此时需要在项目开启时建立一个 Background Scheduler，然后向其中添加每个自动工作的用户的 job，让其自动工作即可

```
 max_gold = 99999

    # 获取 username, uid, 和 user 文档
    username, uid, user = get_user(request)
```

```
    try:
        work = request.GET['work']
    except Exception as e:
        print('请使用规定 json 格式')
        return JsonResponse({'error':'请使用规定 json 格式'})

    if work == 'auto':
        if user['auto_work'] == 0:
            try:
                sched.add_job(auto_work, trigger, args=[uid],
id=uid)
                db.user.update_one({'_id':user['_id']},
{'$set':{'auto_work':1}})
                return JsonResponse({'success':'自动工作开始，将会
每 10min 为您自动工作一次'})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)
        else:
            return JsonResponse({'error':'您已在自动工作中'})

    elif work == 'manual':
        if user['auto_work'] == 1:
            try:
                sched.remove_job(uid)
                db.user.update_one({'_id':user['_id']},
{'$set':{'auto_work':0}})
                return JsonResponse({'success':'自动工作停止'})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)
        else:
            return JsonResponse({'error':'您并未开启自动工作'})

    elif work == 'once':
        if user['auto_work'] == 0:
            # 更改 user 数据
            user['gold_num'] += 10 * user['work_efficiency']

            # 金币数量能超过上限
            if user['gold_num'] > max_gold:
                user['gold_num'] = max_gold
```

```
            # 更新 user 数据库
            try:
                db.user.update_one({'_id': ObjectId(uid)}, {
                                    '$set': {'gold_num':
user['gold_num']}})
                return JsonResponse({'success':'成功工作，金币增加
了'})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

        else:
            return JsonResponse({'error':'您正在自动工作无法进行手
动工作！'})

    else:
        print('请使用规定 json 格式')
        return JsonResponse({'error':'请使用规定 json 格式'})
```

寻宝

寻宝分成两个部分，第一步根据幸运值获得宝物，第二步修改宝物以及获得此宝物的用户信息存入数据库。

宝物获取

个人采用了一种**资源池**的概念，建立一个大小为 n 的数组，其中的数据为整型，代表了某种宝物(我在 item_type 时设定了一个 myid 定义的就是宝物的 id 号，人手动输入，保证不相同)，然后数据库读取 item_type 中所有的文档，对于每个宝物，获取其概率和等级，然后据此使用一算法：

向资源池添加的数量 = n * 概率 * (1 + 幸运值 * 等级 * a)(a 为固定参数)

然后数组中个数不足 n 的按 0 添加

此算法可以根据幸运值的提升来提高获得物品的可能性，而且幸运值越高等级越高的物品获得概率越大，等级越低的物品获得概率虽然也增大但是增大幅度比等级高的物品小。**十连抽按此算法也能提高中奖率哦**(当然也可以多加一个关于多抽的参数)

当然此算法还可以优化，项目不做具体优化。

```
    import numpy as np
    import random
    import math

    # 构建 item 资源池
    items_all_num = 100000
    items_poor = []
    item_list = db.item_type.find()
    for item_type in item_list:
        if item_type['myid'] != 0:
            prob = float(item_type['prob'])
            num = int(prob *
(1+lucky_value*math.exp(item_type['grade'])*0.0005) *
items_all_num) # 可能性算法
            items_poor.extend(list(np.full(num,
item_type['myid'])))

    # 用 0 补充
    if (items_all_num-len(items_poor)) > 0:
        items_poor.extend(list(np.zeros(items_all_num-len(items
_poor))))

    # 从 item 资源池中获取一个 item，并封装成 item 文档
    items = []
    for i in range(times):
        items.append(create_item(db.item_type.find_one({'myid':
int(random.choice(items_poor))})))
    return items
```

更改宝物和用户信息

将宝物的所属用户 id 更改，将用户的背包中物品数量更改，然后写入数据库即可。**需要 try**

这里需要判断背包是否已满，也需要清理背包空间(将最垃圾的一个不在挂牌的宝物删掉)。此处我将把是否自动清理背包设置成用户文档的一个属性，便于用户自己调整。当然这个操作在购买的时候也会遇上，所有可以**封装成一个 check_backpack 函数**调用。

```
    # 根据次数和幸运值获取宝物
    items = get_items(times, int(user['lucky_value']))
```

```
# 向数据库插入新的 item 数据
for item in items:
    item['buid'] = user['_id']  # 更改属于的用户 id
    try:
        db.item.insert_one(item)
    except Exception as e:
        print('--- concurrent write error! ---')
        return JsonResponse(SERVER_ERROR)

# 更新 user 数据库
try:
    db.user.update_one({'_id':ObjectId(uid)},
    {'$inc':{
        'gold_num':-10*times,
        'backpack':1,
    }})
except Exception as e:
    print('--- concurrent write error! ---')
    return JsonResponse(SERVER_ERROR)

return JsonResponse({
    'success':'成功获得宝物！以下是您获得的宝物：',
    'items':obj2str(items),
})
```

佩戴/取下/回收

这三个请求都需要判断 item 是否属于当前浏览器操作用户，防止恶意操作。

佩戴

佩戴宝物时需要判断是否达到佩戴上限(本项目设置佩戴上限为 2 工具 2 饰品，另加最后终极宝物佩戴后会触发您已过关的提示，当然终极宝物佩戴时不能佩戴其他任何宝物，且只能佩戴一件)

还需要判断此宝物是否在背包中(正在挂牌额物品无法佩戴，已佩戴的物品防止用户恶意操作)

若可以佩戴，则将宝物的状态改为 wear，并根据宝物的属性增加用户的工作

效率和幸运值，再将用户的正在佩戴的饰品数量或工具数量更新，最后将更新好的数据写入数据库**同样需要 try**

```
    # 如果是佩戴请求
    if f == 'wear':

        # 判断是否还能佩戴
        flag, warning = check_wear(user, item)
        if flag == False:
            return JsonResponse({'error':warning})

        # 判断是否在背包里
        if item['state'] == 'wear':
            return JsonResponse({'error':'您已佩戴此宝物'})
        elif item['state'] == 'onsale':
            return JsonResponse({'error':'此宝物正在挂牌中，无法佩戴'})

        else:
            # 更新数据库
            try:
                db.user.update_one({'_id':ObjectId(uid)},
                {'$inc':{
                    'work_efficiency':item['work_efficiency'],
                    'lucky_value':item['lucky_value'],
                    'wear.'+item['type']+'_num':1,
                }})
                db.item.update_one({'_id':ObjectId(iid)},{'$set':{'state':'wear'}})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

        return JsonResponse({'success':'佩戴成功！'})
```

取下

取下时需要判断此宝物是否正在佩戴，若不在佩戴则不执行取下操作。

若可以取下，则将宝物的状态改为 backpack，更新工作效率和幸运值，更新佩戴数量，写回数据库**同样需要 try**

```
    # 如果是取下请求
```

```
    elif f == 'backpack':
        # 判断是否佩戴中
        if item['state'] == 'wear':
            # 更新数据库
            try:
                db.user.update_one({'_id':ObjectId(uid)},
                {'$inc':{
                    'work_efficiency':-item['work_efficiency'],
                    'lucky_value':-item['lucky_value'],
                    'wear.'+item['type']+'_num':-1,
                }})
                db.item.update_one({'_id':ObjectId(iid)},
{'$set':{'state':'backpack'}})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse({SERVER_ERROR})
        else:
            return JsonResponse({'error':'此宝物并未佩戴！'})

        return JsonResponse({'success':'取下成功！'})
```

丢弃

丢弃请求需要判断此宝物是否在背包中(不在挂牌也不在佩戴)，然后将用户中背包宝物数量-1，写回数据库**try**

```
    # 如果是丢弃请求
    elif f == 'discard':

        # 判断是否在背包中
        if item['state'] == 'wear':
            return JsonResponse({'error':'您已佩戴此宝物，无法丢弃
'})
        elif item['state'] == 'onsale':
            return JsonResponse({'error':'此宝物正在挂牌中，无法丢
弃'})

        else:
            # 更新数据库
            try:
                db.user.update_one({'_id':ObjectId(uid)},
{'$inc':{'backpack':-1}})
```

```
                db.item.delete_one({'_id':ObjectId(iid)})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

        return JsonResponse({'success':'丢弃成功！'})
```

商城

这也是整个项目中最难的一部分，分为出售界面和购买界面。其中购买界面只显示不属于当前用户的 onsale 的 item，可供当前用户购买；而出售界面则显示属于当前用户的不在佩戴中的 item，可供当前用户将背包中的物品出售/回收。这里可以发很多攻击的请求，我将部分不合理操作的请求在后端屏蔽。

进行所有操作的时候都需要判断操作对象 item 是否存在。若不存在则返回操作错误。

浏览

```
    try:
        f = request.GET['f']

        # 如果只是浏览市场
        if f == 'view':
            items =
list(db.item.find({'buid':{'$ne':ObjectId(uid)},
'state':'onsale'}))
            return JsonResponse({
                'success':'以下为商城中正在出售的宝物：',
                'items':obj2str(items)
            })

        iid = request.GET['iid']
        # 获取 iid 所对应的 item 文档
        item = db.item.find_one({'_id':ObjectId(iid)})
    except Exception as e:
        print('请使用规定 json 格式')
        return JsonResponse({'error':'请使用规定 json 格式'})
```

购买

首先判断当前 item 是否在 onsale 状态，其次判断是否当前用户和 item 所属用户不相同(防止有人闲得慌)。然后再判断是否能够购买(使用寻宝时封装的函数)，若能购买，则更新卖家信息，将其金币数量+price，背包中宝物数量-1；然后更新买家信息，金币数-price，背包中宝物数量+1；最后更新 item 信息，将状态改成 backpack，属于的用户改为买家，然后 price 设为 0(初始值，不出售的宝物价格均为 0)，更新数据库**try**。

```
    # 购买请求
    if f == 'buy':
        # # 判断是否为当前用户操作，防止自己购买自己的宝物
        # if str(item['buid']) == str(uid):
        #     return JsonResponse({'error':'这是您自己的宝物！'})

        # 判断是否 onsale
        if item['state'] == 'onsale':

            # 判断是否能购买
            flag, warning = check_gold_backpack(user,
item['price'], 1)
            if flag is False:
                return JsonResponse({'error':warning})

            # 更新数据库
            try:
                # 更新卖家
                db.user.update_one({'_id':ObjectId(item['buid']
)},
                {'$inc':{
                    'gold_num':item['price'],
                    'backpack':-1,
                }})
                # 更新买家
                db.user.update({'_id':ObjectId(uid)},
                {'$inc':{
                    'gold_num':-item['price'],
                    'backpack':1,
                }})
                # 更新 item 信息
                db.item.update({'_id':ObjectId(iid)},
                {'$set':{
                    'buid':ObjectId(uid),
                    'state':'backpack',
                    'price':0,
```

```
                }})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

            return JsonResponse({
                'success':'成功购买宝物！快去佩戴吧！',
                'item':obj2str(item),
                })

        # 若不是 onsale
        else:
            return JsonResponse({'error':'此宝物并没有挂牌出售！
'})
```

出售

首先判断当前 item 是否为当前用户所有，然后判断 item 是否在背包中(或在挂牌中，要是挂牌中就更新挂牌价格)，成立则将宝物状态设置为 onsale，价格设置为传过来的参数。更新数据库**try**。

```
    # 出售请求
    elif f == 'sell':
        # 判断是否为当前用户操作，防止恶意篡改
        if str(item['buid']) != str(uid):
            return JsonResponse({'error':'您没有此宝物！'})

        # 获取挂牌价格
        try:
            price = int(request.GET['price'])
        except Exception as e:
            print('--- 价格输入有误 ---')
            return JsonResponse({'error':'输入价格有误！'})

        # 判断是否在背包中(或者正在售卖，此时为更新价格)
        if item['state'] == 'backpack' or item['state'] ==
'onsale':

            # 更新数据库
            try:
                db.item.update({'_id':ObjectId(iid)},
                {'$set':{
```

```
                    'state':'onsale',
                    'price':price,
                }})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

            return JsonResponse({'success':'成功挂牌宝物！等待有缘人购买吧！'})

        # 如果宝物状态是佩戴中则无法挂牌出售
        else:
            return JsonResponse({'error':'您正在佩戴此宝物！无法挂牌出售！'})
```

回收

首先判断当前 item 是否为当前用户所有，然后判断 item 是否在挂牌中，若成立则将 item 状态设置为 backpack，价格设置为 0，更新数据库**try**。

```
    # 回收请求
    elif f == 'retrieve':
        # 判断是否为本人操作，防止恶意篡改
        if str(item['buid']) != str(uid):
            return JsonResponse({'error':'您没有此宝物！'})

        # 判断是否在 onsale 中
        if item['state'] == 'onsale':

            # 更新数据库
            try:
                db.item.update({'_id':ObjectId(iid)},
                {'$set':{
                    'state':'backpack',
                    'price':0,
                }})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)
```

```
            return JsonResponse({'success':'成功回收挂牌宝物！'})

        # 如果宝物状态不是 onsale
        else:
            return JsonResponse({'error':'此宝物并未挂牌出售！'})
```

设置

自动清理背包

根据请求将 user 文档中的 auto_clean 修改为对应的值即可

```
    # 如果设置的是自动删除功能
    if setting == 'auto_clean':

        # 判断操作是自动还是手动并改动
        if operate == 'auto':
            # 更新数据库
            try:
                db.user.update_one({'_id':ObjectId(uid)},
{'$set':{'auto_clean':1}})
                return JsonResponse({'success':'设置自动删除宝物
成功！'})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

        elif operate == 'manual':
            # 更新数据库
            try:
                db.user.update_one({'_id':ObjectId(uid)},
{'$set':{'auto_clean':0}})
                return JsonResponse({'success':'设置自动删除宝物
成功！'})
            except Exception as e:
                print('--- concurrent write error! ---')
                return JsonResponse(SERVER_ERROR)

        else:
            return JsonResponse({'error':'请使用规定 json 格式'})
```