华东师范大学数据学院数据库管理实验报告

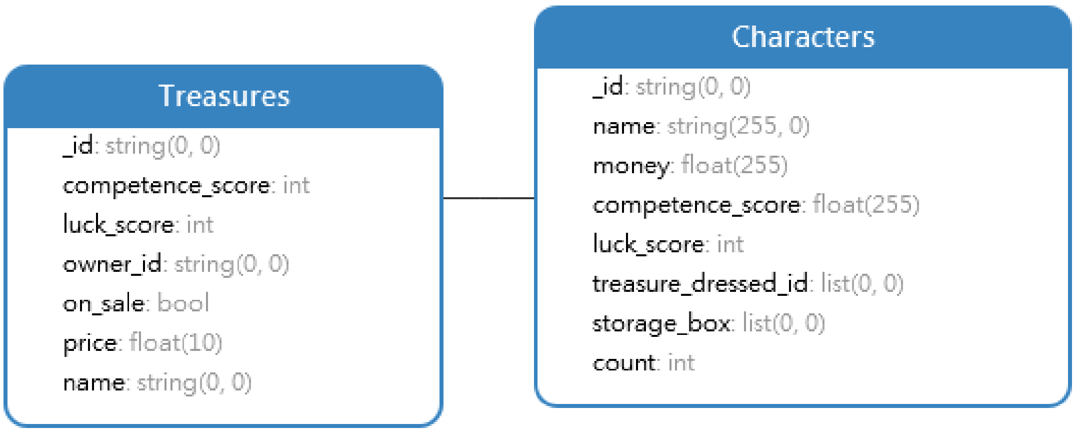
课程名称： 数据库管理	**年级：2017**
指导教师：周煊	**姓名：熊双宇**

0. 环境

- Python 3.6.8
- Flask 1.1.1
- Werkzeug 0.16.0

1. 数据库设计

1.1 Collection

1. 

2. **collection and index**

- Treasures Schema:

  {

  "_id": string **[Index]**

  "name": string **[Index]** <u>name + datetime</u>

  "competence_score": int

  "luck_score": int

  "owner_id": string

  "on_sale": bool

  "price": float

  }

  - Characters Schema:

    {

    "_id": string **[Index]**

"name": string **[Index]**

"money": float

"competence_score": float

"luck_score": int

"treasured_dressed_id": list

"storage_box": list

}

1.2 应用访问数据库(CRUD操作)

```
import pymongo
- query: result = db.collection.find_one() #  Returns a single document, or None if no matching document is found.

- insert:
    - result = db.collection.insert_one() #  Returns an instance of InsertOneResult.
    - result_id =  db.collection.insert_one().inserted_id

- drop: result = db.collection.drop_one()

-  update:

  db.collection.update_one(
      {'_id': ObjectId(result['_id'])},
      {'$inc':
                {},
      '$addToSet ':
                {}
      }
```

2. JSON HTTP API

2.1 API format

```
import requests
URL = 'http://127.0.0.1:5000'
```

2.1.1 API for register, log in and log out

Functions	url	input	methods	output
register	URL + '/register'	json: {'name': string, 'password': string}	GET/POST	1.successfully register and login, a character is inserted into ddatabase2.error because of duplicate 'name'
login	URL + '/login'	json: {'name': string,	GET/POST	1. successfully login, session['username'] is set 2. error because of no existing

Functions	url	input	methods	output
	/login	'password': string}		'name' in database or 'password' not match to name'
logout	URL + '/logout'	None	GET	1. successfully: session['username'] is popped, return '/login' page 2. if no users is logged in， redirect to the '/login' page

2.1.2 API for automatically hunting treasures and working

2.1.2.1 Config

```
class Config(object):  # 创建配置，用类
    # 任务列表
    JOBS = [
        {  # 第1个任务，每隔15min执行一次
            'id': 'hunt_treasure',
            'func': '__main__:hunt_treasure',  # 方法名
            'args': (),  # 入参
            'trigger': 'interval',  # interval表示循环任务
            'seconds': 900,  # 15min
        },
        {  # 第2个任务，每隔15min执行一次
            'id': 'work_money',
            'func': '__main__:work_money',  # 方法名
            'args': (),  # 入参
            'trigger': 'interval',  # interval表示循环任务
            'seconds': 900,  # 15min
        }
    ]


app.config.from_object(Config())  # 为实例化的flask引入配置
```

2.1.2.2 hunt_treasure

Functions	config	algorithm
hunt _treasure	1. one user's storage box can have treasures less than **MAX_TREASURE(20)**; 2. one user hunt one treasure according to **luck_score** every **Time(15)** minutes.	None

```
MAX_TREASURE = 20

def hunt_treasure():
    global MAX_TREASURE
    print(datetime.datetime.utcnow(), 'All characters auto-hunt for a 
treasure')

    for character in characters.find():
        new_t_id = generate_treasure(character, 
character['competence_score'], character['luck_score'])
        if character['count'] >= MAX_TREASURE:
            throw_t = throw_treasure(character)
            if throw_t is None:
                print('Hunt failed: the character\'s storage box is full of on-sale treasures')
                return 1
        characters.update_one(
            {'_id':  ObjectId(character['_id'])},
            {'$addToSet': {'storage_box': str(new_t_id)},  # add a value, unless exists
             '$inc': {'count': 1}
             }
        )
        print("All characters successfully hunt a treasure")
    return 0

```

Output:

- Error:
  - if user's **storage_box** is full, throw a treasure with lowest **luck_score**, and store the new treasure in box;
  - if user's box is full of treasures all for sale , hunt will fail and function will return 1 as error.
- Success: add a treasure stored in **storage_box**, print out "All characters successfully hunt a treasure"

2.1.2.3 work_money

Functions	config	algorithm
work_money	1. one user work for money according to **competence_score** every **Time(15)** minutes.	1. if user's **competence_score** <= 100, the user will get 101$; 2. else get (**competence_score**% 100)$

Output:

print 'All characters have worked one time!'

```
def work_money():
    print(datetime.datetime.utcnow(), 'All characters auto-worked for money one time')

    for character in characters.find():
        if character['competence_score'] <= 100:
            money = 101
        else:
            money = character['competence_score']

        characters.update_one(
            {'_id': ObjectId(character['_id'])},
            {'$inc': {'money': (money % 100)}}
        )
       print(character['money'])
    return 0
```

2.1.2.4 event listener

```
def my_listener(event):
    if event.exception:
        print('The job crashed :(')
    else:
        print('The job worked :)')

scheduler.add_listener(my_listener, EVENT_JOB_EXECUTED | EVENT_JOB_ERROR)
```

2.1.3 API for dressing and taking off treasures

2.1.3.1

Treasure has one of the following properties:

- Tool: the treasure's **competence_score** > 0, and **luck_score** is 0;
- Accessory: the treasure's **luck_score** > 0, and **competence_score** is 0.

2.1.3.2 Principle:

Every user can dress on limited number of treasures with different properties:

```
MAX_TOOLS = 1 # a user can dress on at most one treasure with  'Tool'property
MAX_ACCESSORIES = 2 # a user can dress on at most two treasure with 'Accessory'property
```

2.1.3.3 function

Function	url	input	methods
dress_treasure	URL + '/characters/dress/<task>'	1. json: {'name': string, 'treasure_name': list}; 2. task: on/off.	POST

Output:

- Error:
  - if the length of treasure list > 3, function will just return an error;
  - if treasures in list can't be found in database, an error will be returned;
- check property of treasure, if treasure isn't a tool or a accessory, return an error;
  - if treasure is not belong to user, it will not be dressed on.
- Success:
  - 'task' is 'on': move treasures' id in **'storage_box'** into **'treasure_dressed_id'** , change the **competence_score** and **luck_score**, and profile of the user;
  - If 'task' is 'off': move treasures' id in **treasure_dressed_id** into **storage_box**, change the **competence_score** and **luck_score**, and profile of the user;

2.1.4 API for looking through market

Function	url	input	methods	output
get_all_onsale_treasures	URL + '/market'	None	GET	information of all treasures with 'True' for the key **'on_sale'**

2.1.4 API for transaction, quotation and undo-quotation treasure

2.1.4.1 transaction

Function	url	input	methods
transaction_treasures('purchase')	URL + '/market/transaction/purchase'	json: {'name': string, 'treasure_name': string}	POST

Output:

- Error:
  - If user not exists in database, return {'Error': 'name is invalid!'}
  - If treasure not exits in database, return {'Error': 'treasure_name is invalid!'}
  - If treasure not for sale, return {'Error': 'treasure is not on_sale now!'}
  - If user can't afford treasure, return {'Error': 'you have money:{}, which is less than the price:{}'.format(character['money'], treasure['price']) }
- Success:
  - Update seller's and purchaser's profile:
    - **'money'**
    - **'count'** : the number of treasures in **storage_box**
    - **’storage_box'**
  - Update treasure's information: change the **'owner_id'**, set **'on_sale'** key as False, update the **'price'** as 0;
  - return the treasure information.

2.1.4.2 quotation

Function	url	input	methods
transaction_treasures('for_sale')	URL + '/market/transaction/for_sale'	json: {'name': string, 'treasure_name': string, 'price': float}	POST

Output:

- Error:
  - If treasure's **'on_sale'** is true, return {"Error":'treasure has been on_sale!'}
  - If treasure is not belong to user, return {'Error': 'treasure isn't in you storage_box!'}
- Success:
  - treasure's information: set **'on_sale'** as True, change **'price'**
  - return the treasure information

2.1.4.3 undo-quotation

Function	url	input	methods
transaction_treasures('off_sale')	URL + '/market/transaction/off_sale'	json: {'name': string, 'treasure_name': string}	POST

Output:

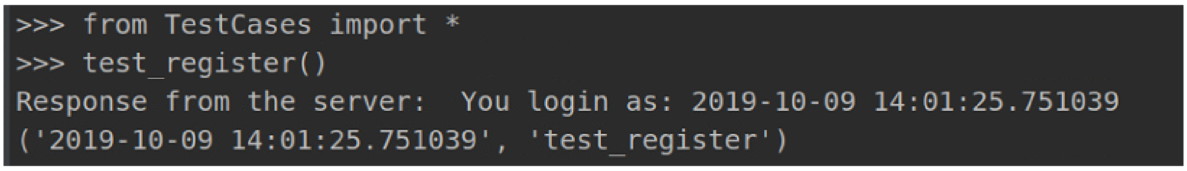
- Error:
  - If treasure is not for sale, return {'Error': 'the treasure has been off_sale!'}
  - If treasure is not belong to user, return {'Error': 'treasure isn't in you storage_box!'}
- Success:
  - treasure's information: set **'on_sale'** as False change **'price'**
  - return the treasure information

3. 测试案例

3.1 测试注册角色

```python
def test_register():
    '''
    1. username is universal time (to avoid duplicate name)
    2. password is 'test_register'
    3. use requests.post(url,json) to send request
    4. return the name and password
    '''
```

Output:

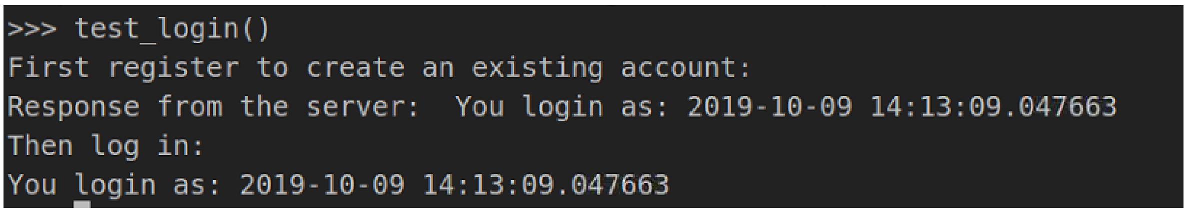


```
>>> from TestCases import *
>>> test_register()
Response from the server:  You login as: 2019-10-09 14:01:25.751039
('2019-10-09 14:01:25.751039', 'test_register')
```

3.2 测试登陆角色

```python
def test_login():
    '''
    1. First register as a user
    2. Then use requests.post(url,json) to login
    '''
```

Output:

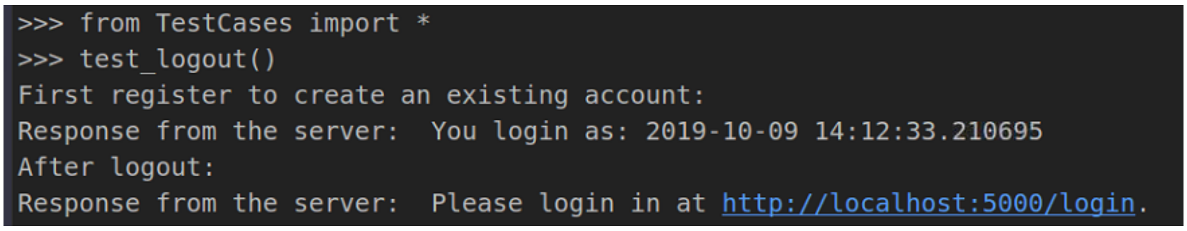


```
>>> test_login()
First register to create an existing account:
Response from the server:  You login as: 2019-10-09 14:13:09.047663
Then log in:
You login as: 2019-10-09 14:13:09.047663
```

3.3 测试登出角色

```
def test_logout():
    '''
    1. First register as a user
    2. Then log in the user
    3. Finally use requests.get(url) to log out
    '''
```

Output:



```
>>> from TestCases import *
>>> test_logout()
First register to create an existing account:
Response from the server:  You login as: 2019-10-09 14:12:33.210695
After logout:
Response from the server:  Please login in at http://localhost:5000/login.
```

3.4测试穿脱宝物

```
def test_dress_treasure(task, name = None, treasure_name_list  = None):
    '''
    1. If user is None, register and log in one
    2. Add a treasure to the user's storage_box(use the function:
'add_treasure()' which will be described later)
    3. Test the dress function
    '''
```

Output:

```
>>> test_dress_treasure('on')
Response from the server:  You login as: 2019-10-09 14:14:36.806625
Response for adding a treasure:  {
  "result": [
    {
      "competence_score": 0,
      "luck_score": 3,
      "name": "Test_treasure",
      "on_sale": false,
      "owner_id": "5d9deb4df906509ba676f320",
      "price": 0
    }
  ]
}

Response for dressing a treasure:  {
  "Dress": [
    {
      "_id": "5d9deb4df906509ba676f320",
      "competence_score": 0.0,
      "luck_score": 3,
      "name": "2019-10-09 14:14:36.806625",
      "storage_box": [],
      "treasure_dressed_id": [
        "5d9deb4df906509ba676f321"
      ]
    }
  ]
}

```

We can see the treasure's **'_id'** is added to user's **'treasure_dressed_id'**!

3.5 测试浏览市场

```python
def show_market():
    '''
    1. use requests.get(url)
    '''
```

Output:

```
>>> show_market()
Market:  {
  "All on-sale treasures": [
    {
      "competence_score": 0,
      "luck_score": 3,
      "name": "Test_treasure: 2019-10-09 14:17:30.236729",
      "on_sale": true,
      "owner_id": "5d9debfaf906509ba676f322",
      "price": 0
    },
    {
      "competence_score": 0,
      "luck_score": 3,
      "name": "Test_treasure: 2019-10-09 14:19:50.554916",
      "on_sale": true,
      "owner_id": "5d9deb4df906509ba676f320",
      "price": 0
    },
    {
      "competence_score": 0,
      "luck_score": 3,
      "name": "Test_treasure: 2019-10-09 14:40:22.060260",
      "on_sale": true,
      "owner_id": "5d9df15629b355340263264d",
      "price": 0
    }
  ]
}

```

All treasures' **'on_sale'** are True

3.6 测试宝物交易、挂牌和收回

```
def test_transaction(task, treasure_name=None, name=None, price=0):
    '''
    1. 交易：task='purchase'；挂牌：task='for_sale';收回：task='off_sale'
    2. If name is None, register and login a user
    3. If user has no treasure(treasure_name is None), add a treasure into the user's storage_box
    4. Use requests.post(url, json) to send a request
'''
```

3.6.1 宝物交易

Output:

```
>>> test_transaction('purchase',buyer_name = "2019-10-09 14:14:36.806625", treasure_name = "Test_treasure: 2019-10-09 14:19:50.554916")
Response for transaction task purchase :  {
  "purchase": [
    {
      "money": 0,
      "on_sale": false,
      "owner_id": "5d9deb4df906509ba676f320",
      "owner_name": "2019-10-09 14:14:36.806625",
      "price": 0,
      "treasure_id": "5d9dec86f906509ba676f325",
      "treasure_name": "Test_treasure: 2019-10-09 14:19:50.554916",
      "user_id": "5d9deb4df906509ba676f320"
    }
  ]
}
```

After transaction:

- treasure's **'on_sale'** is changed to False]
- treasure's **'owner_id'** matches to buyer's **'_id'**

3.6.2 挂牌

```
>>> test_transaction('for_sale', price = 1)
Response from the server:  You login as: 2019-10-09 14:19:50.306879
Response for adding a treasure:  {
  "result": [
    {
      "competence_score": 0,
      "luck_score": 3,
      "name": "Test_treasure: 2019-10-09 14:19:50.554916",
      "on_sale": false,
      "owner_id": "5d9dec86f906509ba676f324",
      "price": 0
    }
  ]
}

Transaction result:  {
  "for_sale": [
    {
      "on_sale": true,
      "owner_id": "5d9dec86f906509ba676f324",
      "owner_name": "2019-10-09 14:19:50.306879",
      "price": 1,
      "treasure_id": "5d9dec86f906509ba676f325",
      "treasure_name": "Test_treasure: 2019-10-09 14:19:50.554916"
    }
  ]
}
```

We can see:

- treasure's **'on_sale'** change from False to True;
- treasure's **'price'** change from 0 to 1;

3.6.3 收回

Output:

```
>>> test_transaction('off_sale',buyer_name = "2019-10-09 14:14:36.806625", treasure_name = "Test_treasure: 2019-10-09 14:19:50.554916")
Response for transaction task off_sale :  {
  "off_sale": [
    {
      "money": 1,
      "on_sale": false,
      "owner_id": "5d9deb4df906509ba676f320",
      "owner_name": "2019-10-09 14:14:36.806625",
      "price": 0,
      "treasure_id": "5d9dec86f906509ba676f325",
      "treasure_name": "Test_treasure: 2019-10-09 14:19:50.554916",
      "user_id": "5d9deb4df906509ba676f320"
    }
  ]
}
```

After off_sale:

- The treasure's **'on_sale'** is changed to False
- **'price'** is changed to 0

3.7 测试自动寻宝和工作

测试代码:

```
def test_auto_jobs():
    with app.app_context():
        # scheduler.get_jobs()
        for job_id in scheduler.get_jobs():
            print(job_id)
            print(job_id.next_run_time)
```

输出结果:

```
hunt_treasure (trigger: interval[0:15:00], next run at: 2019-10-10 00:17:05 CST)
2019-10-10 00:17:05.555141+08:00
work_money (trigger: interval[0:15:00], next run at: 2019-10-10 00:17:05 CST)
2019-10-10 00:17:05.555258+08:00
```

4. 其他相关函数解释

```
URL = 'http://127.0.0.1/'
```

4.1 给数据库添加角色

Function	input	output
add_character(name, hashpass)	args: name(string), hashpass(string)	{'name': new_character['name'], 'date': new_character['date']}

- 该函数将在用户注册时被调用;
- 将用户注册和角色创建分开的初衷是想实现一个用户多个角色，后默认一个用户只有一个角色，即用户等价于角色；

4.2 给角色添加宝物

Function	url	input	method
add_treasure()	URL + '/treasure/add'	json: {'name': string, 'treasure_name':list, 'competence_score': int, 'luck_score':int}	POST

Output:

- Error:
  - if 'treasure_name' is not a list, return an error: {"Error: ": 'treasure_name {} is not a list'.format(treasure_name)}
  - if **storage_box** of user with 'name' is full, return an error: {'Error': 'The owner's storage is full of {} treasures!'.format(MAX_TREASURE)}
- Success:
  - Treasure will be added to **storage_box** of user with **'name'**

PS:

- 该函数将只测试案例中被调用，如给宝物挂牌时为了防止角色没有宝物，将先为其添加一件宝物;
- 该函数的初衷是允许角色按照自己的能力，将多件宝物合成出新的宝物给自己【未实现】；

4.3 计算角色能力和运气

4.3.1 能力

Function	input
cal_competence(task, t_score, num)	task: 'on'/'off', t_score: int, c_score: int, num: int

```python
def cal_competence(task, t_score, c_score, num):
    '''
    function will be called in 'dress_treasure'
    task: on/off (on: dress on a treasure, off: take off a treasure)
    t_score: the competence_score of treasure
    c_score: the competence_score of character
    num: the number of treasures characer has dressed on
    '''
```

该函数在角色穿脱宝物时调用;

Output:

- Success: return the competence score after dressing on or taking off a treasure;
- Error:
    - if the competence score > MAX_COMPETENCE_SCORE, return None as error;
    - if user want to take off a treasure, but he/she has dressed on no treasures, return None as error;

4.3.2 运气

同理与计算能力函数

4.4 丢弃存储柜里的宝物

Condition: 在**‘storage_box’**中挑选出运气值最低的的宝物

Function	input	output
throw_treasure(character)	None	the **’_id** of treasure with lowest **'luck_score'**

- 该函数将在角色存储柜满后调用
- 被丢弃的宝物在数据库中不会被删除，只是**'owner_id'** 被置为None，该设计的初衷是回收所有被丢弃的宝物给系统的宝物库，合成具有新的属性的宝物【‘合成宝物’未实现】

Output:

- Error: Not found a treasure satisfied with the **Condition**， return {'Error':'Found throw_treasure with lowest luck!'}
- Success: return the id of treasure satisfying the **Condition**

4.5 给角色生成宝物

Function	input
generate_treasure(character, competence_score, luck_score)	args: 1. character: document; 2. competence_score: float; 3. luck_score: int

Output:

- Error:
    - if the character is None, return ({'Error': 'The character is None!'};

  - if the treasure generated has duplicate name with one treasure in
  - database, return {'Error':'There exists two duplicated name in DB!' }
- Success: return the _**'id'** of the treasure generated;

Algorithm:

- 宝物的luck_score分为50个等级,competence_score有11个档次:

  ```
  treasures_luck_level = [num for num in range(50)]
  treasures_competence_score = [0, 28, 70, 96, 120, 250, 345, 427, 513,
  624, 718]
  ```

  宝物名字的第一部分从以下列表中选出:

  ```
  treasure_names_list = ['烛龙', '帝江', '英招', '饕餮', '白泽', '应龙', '九尾
  狐', '青鸟', '狻猊', '穷奇', '凤凰']
  ```

- 给新的宝物设置属性
  - 名字：为了防止重名，采用这样的格式：'part1_part2_part3'
    - part1: 随机在treasure_names_list中选出;
    - part2: 角色名称;
    - part3: 世界时间

      ```
      import datetime
      datetime.datetime.utcnow()
      ```

  - competence_score 和 luck_score:
    - 用分位数将'**treasures_luck_level**'划分为6部分，计算角色的'**luck_score**' 属于哪部分;
    - 然后分别在'**treasures_luck_level**'和'**treasures_competence_score**'中从最低级到比该部分更高一级中随机选择一个值，作为新的宝物的luck_score和competence_score
    - 由于宝物被划分为两种属性，即作为tool的宝物的luck_score 为0， 作为accessory的宝物的competence_score为0，所以对新的宝物的luck_score和competence_score值随机挑选出一个置为0，并且保证另一个不为0。

4.6 测试schedule函数

```
def test_auto_jobs():
    from flask_apscheduler import APScheduler

    with app.app_context():
        for job_id in scheduler.get_jobs():
            print(job_id)
            print(job_id.next_run_time)

            # assert job_id.next_run_time.replace(tzinfo=None) >=
(datetime.datetime.utcnow() +
datetime.timedelta(minutes=15)).replace(tzinfo =None)
            # assert job_id.next_run_time.replace(tzinfo=None) <=
(datetime.datetime.utcnow() +
datetime.timedelta(minutes=15)).replace(tzinfo =None)
```

思路：打印出schedule.get_jobs中的所有job的下一次运行时间

初衷是查看下一次运行时间和本次运行时间的间隔是否是设置的时间间隔。