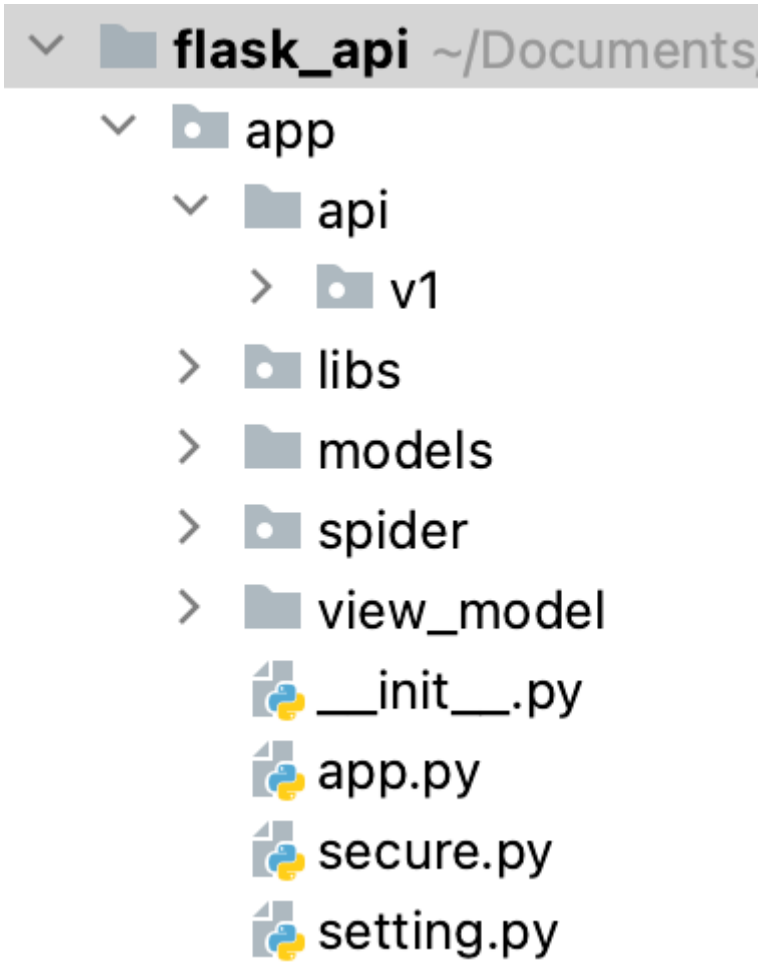
flask-api 使用手册

此项目是笔者从之前项目中精简而来，旨在方便开发者简单开启后端接口服务

代码设计思想与文件管理方式源于*MVC+RESTful*

以下为项目代码组织结构，之后会一一讲解

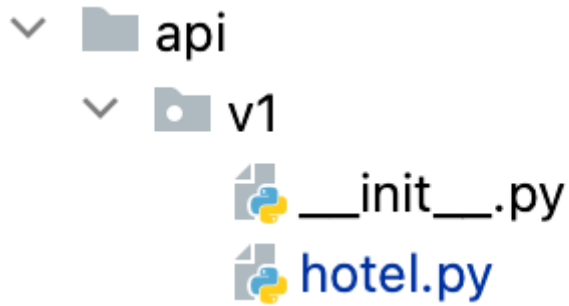


1、视图函数的注册

由于采用RESTful接口规范，于是将服务器中数据当成资源进行接口设计

故推荐将每个数据类型的视图函数单独写入一个py文件

例：



将hotel的所有数据视作服务器上一类资源，并需要编写相关视图函数以提供接口服务

api为视图函数组织文件夹，v1代表版本号（供后续迭代升级），各类资源的视图函数集中到单独的py文件中，如hotel

代码组织较为繁琐，不推荐一上手便去深入细节，重点关注使用方式

视图函数注册方式

1. **请在各类资源的py文件当中导入Redprint，并实例化该对象**

   以hotel.py举例

   即在hotel.py当中添加以下代码

```
from app.libs.redprint import Redprint
api = Redprint('hotel')
```

2. **请将实例化后的Reprint对象注册到上层Blueprint对象中**

   以hotel.py为例

   即在v1/__init__文件当中指定区域添加代码

```
#---->please add register code in this section <----#
   hotel.api.register(bp_v1)

#---->please add register code in this section <----#
```

   若添加新模块，需要在该文件当中导入相应的包

   之后，在标注区域添加XX.api.register(bp_v1)

3. **请根据对资源的请求动作设计url**

如获取hotel信息，那他的动作是get,故url为"\v1\hotel\get"(蓝图层的'v1'(构建Blueprint时传入)，红图层的'hotel'(构建Redprint时传入),请求动作'get')

视图函数和url的绑定方式如下

```
@api.route('/get', methods=['GET'])
def get_hotel():
    return 'hotel!!!'
```

路由绑定涉及到python装饰器语法，较为复杂，不做讲解，会用即可

2、数据库的映射与查询

为专注于业务逻辑的开发，避免繁琐sql语句的编写，flask对本来就很好用的Sqlalchemy库进行封装，形成flask_sqlalchemy

该库核心思想是结合面向对象和数据库的操作，以对象映射数据库操作（详细知识请参考ORM模型）

对于刚入门的开发者而言，切忌沉湎于技术细节，做到能用会用即可，等到进阶阶段可选择性了解

接下来介绍如何使用

flask_sqlalchemy使用

数据库表的映射

笔者在实际开发中又对flask_sqlalchemy进行继承优化,形成Base类

代码如下

```
class Base(db.Model):
    __abstract__ = True
    status = Column(SmallInteger, default=1)
    create_time = Column('create_time', Integer)

    def __init__(self):
        self.create_time = int(datetime.now().timestamp())

    def set_attrs(self, attrs_dict):
        for key, value in attrs_dict.items():
            if hasattr(self, key) and key != 'id':
                setattr(self, key, value)
```

```
    #定义了此方法 可以通过o['key']来访问实例变量
    def __getitem__(self, item):
        return getattr(self, item)

    @property
    def create_datetime(self):
        if self.create_time:
            return datetime.fromtimestamp(self.create_time)
        else:
            return None

    def delete(self):
        self.status = 0
```

此部分代码存于models文件夹中的base.py，在软件架构层面位于MVC的M层即Model层

既然是可以实现对象到数据库的映射，那如何映射呢 ---> 通过定义类变量

可以看到在这个基类当中定义了2个基础属性

1. status -- 数据库当中的数据一般使用的是软删除的方式，即用一个状态位代表是否删除
2. create_time -- 对于每个数据项都会记录一下加入数据库的时间以供后续数据分析

容易发现，该基类定义的是一些基础属性，数据库当中不需要存在这样的表，所以用

```
__abstract__ = True
```

向sqlchemy指示该类不需要进行数据库映射

如果需要将对象映射到数据库当中需要继承Base类，并添加特有的属性

代码体现

```
class Hotel(Base):
    __tablename__ = 'hotel'
    id = Column(Integer, primary_key=True, autoincrement=True)
    price = Column(Integer)
    location = Column(String(100))
    description = Column(String(600))
    src = Column(String(100))
    comments = Column(String(1000))
```

在执行完这个代码之后，数据库中会自动进行数据映射

以下为本地数据库的查询结果

```
mysql> show tables;
+-----------------+
| Tables_in_hotel |
+-----------------+
| hotel           |
+-----------------+
```

```
mysql> describe hotel;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| status      | smallint      | YES  |     | NULL    |                |
| create_time | int           | YES  |     | NULL    |                |
| id          | int           | NO   | PRI | NULL    | auto_increment |
| price       | int           | YES  |     | NULL    |                |
| location    | varchar(100)  | YES  |     | NULL    |                |
| description | varchar(600)  | YES  |     | NULL    |                |
| src         | varchar(100)  | YES  |     | NULL    |                |
| comments    | varchar(1000) | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
```

至此，是不是发现在代码开发的时候，一句sql语句都没写，却神奇地操作了数据库

读至此，应该会有一个疑问：我怎么指定要操作哪个服务器上的哪个数据库呢

数据库连接的配置信息应该写到flask_api\app\secure.py当中

配置方式如下

```
SQLALCHEMY_DATABASE_URI =
('mysql+pymysql://root:123456@localhost:3306/Hotel')
```

各参数含义：

1. mysql : 数据库的类型
2. pymysql: 数据库的驱动
3. root: 数据库的用户名
4. 123456数据库的密码
5. localhost: 数据库的ip地址
6. 3306：数据库服务的端口号
7. Hotel: 数据库名

参考以上实例，可以在\secure.py中进行数据库连接到配置

需要注意的是，为保证数据库中表的正确映射，请在指定区域导入相应包（会用即可，有空可琢磨一下为什么）

导入的位置是app__init__.py文件的首部（已指定了位置）

```
# ---->please import model package in this section <----#
from app.models.hotel import Hotel

# ---->please import model package in this section <----#
```

数据项的插入

将数据项的插入操作放到视图函数中进行实验

```
@api.route('/set', methods=['GET'])
def set_hotel():
    hotel = Hotel()
    with db.auto_commit():
        hotel.price = 348
        hotel.location = '南京'
        hotel.description = '位于6朝古都的南京'
        hotel.src = '如家'
        hotel.comments = '舒服的一次住宿'
        db.session.add(hotel)
    return 'hotel saved in db'
```

使用postman访问该url时，才进行数据插入

构建url，进行访问

执行函数之后，查询数据库得

```
mysql> select * from hotel;
+--------+-------------+----+-------+----------+--------------------------+--------+----------------------+
| status | create_time | id | price | location | description              | src    | comments             |
+--------+-------------+----+-------+----------+--------------------------+--------+----------------------+
|      1 |  1701348333 |  1 |   348 | 南京     | 位于6朝古都的南京        | 如家   | 舒服的一次住宿       |
+--------+-------------+----+-------+----------+--------------------------+--------+----------------------+
```

需要注意的是，对数据库进行插入操作时，需放到上下文语境当中运行

这是因为，上下文语境中是一个数据库事务，当发生错误时要进行数据库回滚，于是提供了一个auto_commit上下文管理器

因为涉及到数据库的基本概念，以及python中高级特性，不理解没关系，还是会用即可，有时间可以看看源码，着重理解上下文管理器是干什么的，以及怎么定义的会对python的设计思想有进一步的认识

在此给出固定代码

```
    with db.auto_commit():

        ######################
        #       obj.a = ''  #set the attribution
        #       obj.d = ''  #set the attribution
        ######################
        db.session.add(obj)
```

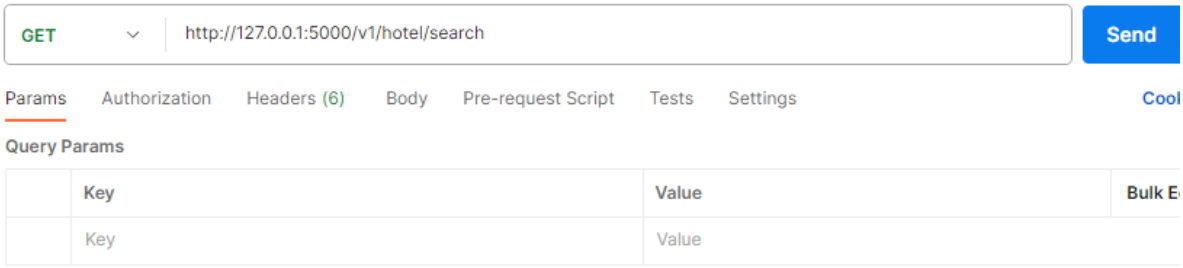
数据库的查询

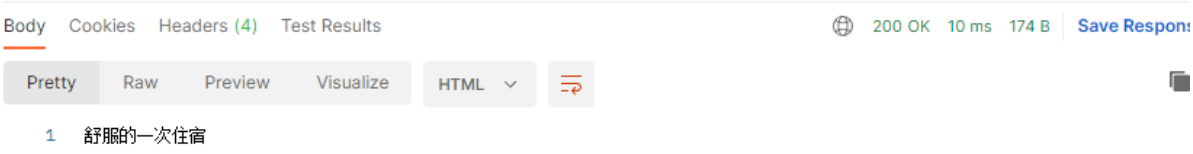
同样的编写视图函数进行讲解

```
@api.route('/search', methods=['GET'])
def search_hotel():
    hotel = Hotel.query.filter_by(id=1).first()
    return hotel.comments
```

可以发现查询操作是**定义成模型类的类方法** ，类名指定了要查询的表

以下是postman的实验





有关查询操作，sqlcemy给出了很多api，可查看sqlchemy的api文档进行使用

3、模型序列化的使用方式

因为搭建的是api接口框架，故希望每次返回的都是json格式的数据

对象的序列化比较复杂，笔者进行了相应的封装，便于大家使用

使用方式

1. 请在需要使用序列化函数的文件首部先导入jsonify

```
from flask.json import jsonify
```

2. 请在类当中指明需要进行序列化的属性，即定义keys函数，并指明想要序列化的属性，哪怕只有一个也**一定要以列表形式返回**

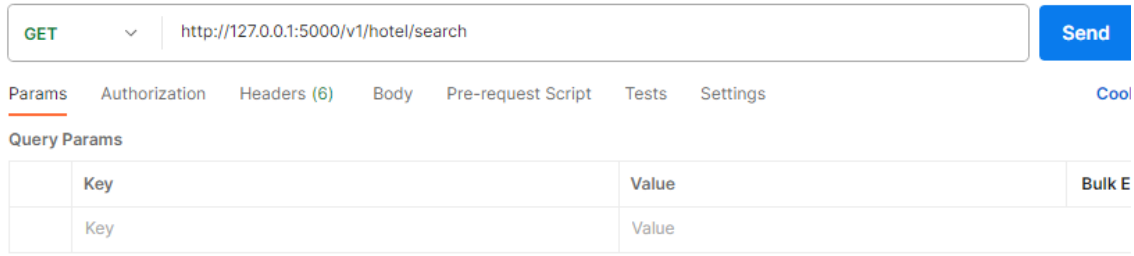
   以Hotel类为例：

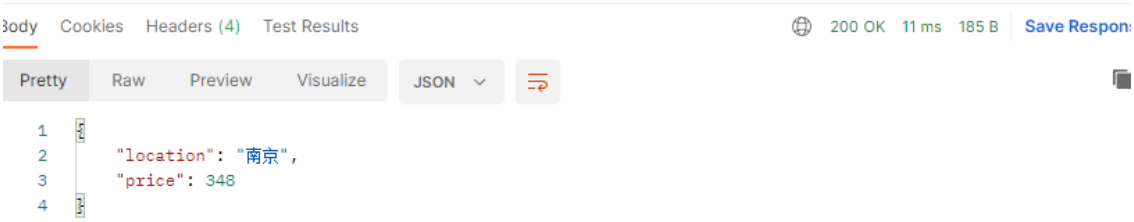
```
class Hotel(Base):
    __tablename__ = 'hotel'
    id = Column(Integer, primary_key=True, autoincrement=True)
    price = Column(Integer)
    location = Column(String(100))
    description = Column(String(600))
    src = Column(String(100))
    comments = Column(String(1000))

    def keys(self):
        return ['price', 'location']#specify the attributions you
want to jsonify
```

   编写视图函数进行测试

```
@api.route('/search', methods=['GET'])
def search_hotel():
    hotel = Hotel.query.filter_by(id=1).first()
    return jsonify(hotel)
```

   以下是postman的实验





果然只序列化了指定的属性

4、视图层使用推荐

由于目前对于数据的获取源自于数据库，但数据库当中的数据未必方便可视化

于是在传递给前端进行可视化之前，需要有层ViewModel进行数据调整

需结合具体的业务进行开发

视图层的代码请放到app\view_model文件夹内

5、工具函数定义

在开发过程中，可能会有一些工具类的函数定义

请将工具代码放到app\libs当中

6、RESTful抛异常的方式

在实际开发过程中，由于用户传递的参数未知，以及代码逻辑难免会出现异常情况

于是，抛异常的操作在所难免

由于是api接口服务，所以对于异常的抛出也希望是以json的格式返回给客户端

api异常抛出方式

1. 请从app\libs\error.py 导入 APIException
2. 在定义自己的异常类，并继承APIException

```
class Error(APIException):
    code = 400
    error_code = 1006
    msg = '错误'
```

   - 其中code代表的是错误码，需要根据http协议查询该异常应该抛出的错误码
   - error_code是根据项目进行定义的，需要建立一个文档以供api调用者参考使用
   - msg是异常的描述信息
3. 在代码逻辑中嵌入异常抛出

   示意代码：

```
if err:
    raise Error()
```

   客户端会接受到json格式的异常信息，格式如下

```
{
    "code":400,
    "error_code":1006,
    "msg":"错误"
}
```

7、爬虫函数编写

接口的开发有时需要调用别人的api，或者需要自己编写爬虫进行数据爬取

请将爬虫代码写到app\spider文件夹中