day92 flask

内容回顾

1. 什么是接口?

   - interface类型，Python没有，Java/C#语言才有。用于约束实现了该接口的类中必须有某些指定方法。
   - api也可以成为一个接口。

2. 抽象类和抽象方法

   他既具有约束的功能又具有提供 子类继承方法 的功能，Python中通过abc实现。

3. 重载和重写?

4. flask和django的区别?

5. 什么是数据库链接池？以及作用?

6. sqlhelper

7. 面向对象的上下文管理

8. 上下文管理和SQLHelper

```
import pymysql
from DBUtils.PooledDB import PooledDB

class SqlHelper(object):
    def __init__(self):
        self.pool = PooledDB(
            creator=pymysql,  # 使用链接数据库的模块
            maxconnections=6,  # 连接池允许的最大连接数，0和None表示不限制连接数
            mincached=2,  # 初始化时，链接池中至少创建的链接，0表示不创建
            blocking=True,  # 连接池中如果没有可用连接后，是否阻塞等待。True，等待；
False，不等待然后报错
            ping=0,
            # ping MySQL服务端，检查是否服务可用。# 如：0 = None = never, 1 =
default = whenever it is requested, 2 = when a cursor is created, 4 = when a
query is executed, 7 = always
            host='127.0.0.1',
            port=3306,
            user='root',
            password='222',
            database='cmdb',
            charset='utf8'
        )

    def open(self):
        conn = self.pool.connection()
```

```
        cursor = conn.cursor()
        return conn,cursor

    def close(self,cursor,conn):
        cursor.close()
        conn.close()

    def fetchall(self,sql, *args):
        """ 获取所有数据 """
        conn,cursor = self.open()
        cursor.execute(sql, args)
        result = cursor.fetchall()
        self.close(conn,cursor)
        return result

    def fetchone(self,sql, *args):
        """ 获取所有数据 """
        conn, cursor = self.open()
        cursor.execute(sql, args)
        result = cursor.fetchone()
        self.close(conn, cursor)
        return result

    def __enter__(self):
        return self.open()[1]

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(exc_type, exc_val, exc_tb)


db = SqlHelper()
```

今日概要

- wsgi
- 创建flask对象
  - 模板
  - 静态文件
- 路由系统
  - 路由的应用：装饰器（推荐）、方法
  - 动态路由
- 视图
  - FBV
  - CBV
- 模板
  - 继承
  - include
  - 自定义标签
- 特殊装饰器
  - before_request充当中间件角色

今日详细

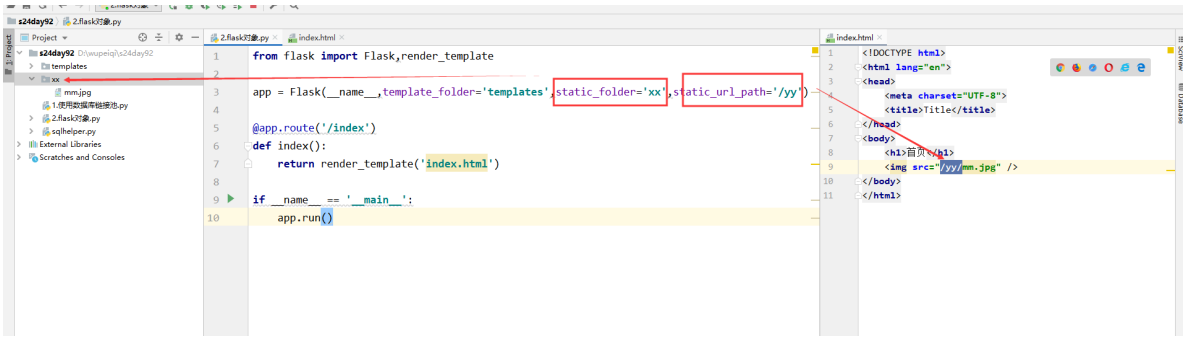
1.wsgi 找源码的流程

```
from werkzeug.serving import run_simple
from werkzeug.wrappers import BaseResponse

def func(environ, start_response):
    print('请求来了')
    response = BaseResponse('你好')
    return response(environ, start_response)


if __name__ == '__main__':
    run_simple('127.0.0.1', 5000, func)
```

```
"""
    1.程序启动，等待用户请求到来
        app.run()
    2.用户请求到来 app()
        app.__call__
"""

from flask import Flask

app = Flask(__name__)

@app.route('/index')
def index():
    return 'hello world'

if __name__ == '__main__':
    app.run()
```

2.flask对象

静态文件的处理。



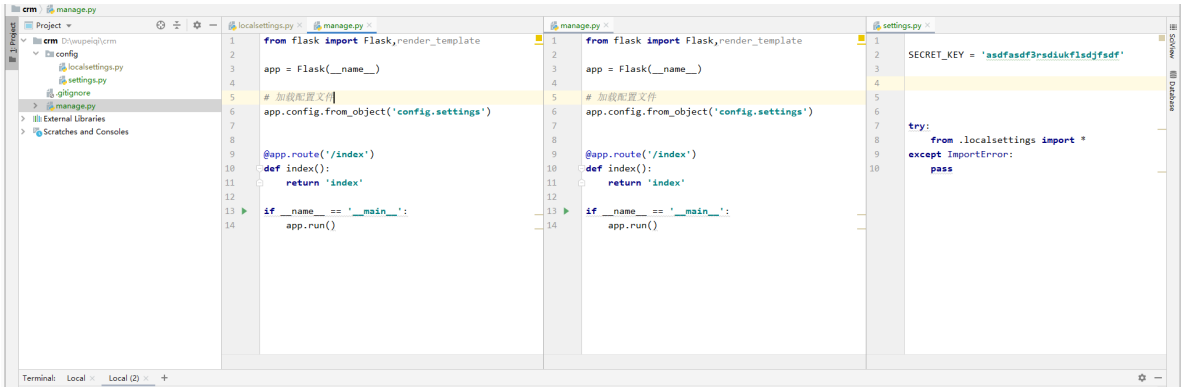
推荐

```python
from flask import Flask,render_template

app = Flask(__name__,template_folder='templates',static_folder='static')

@app.route('/index')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run()
```

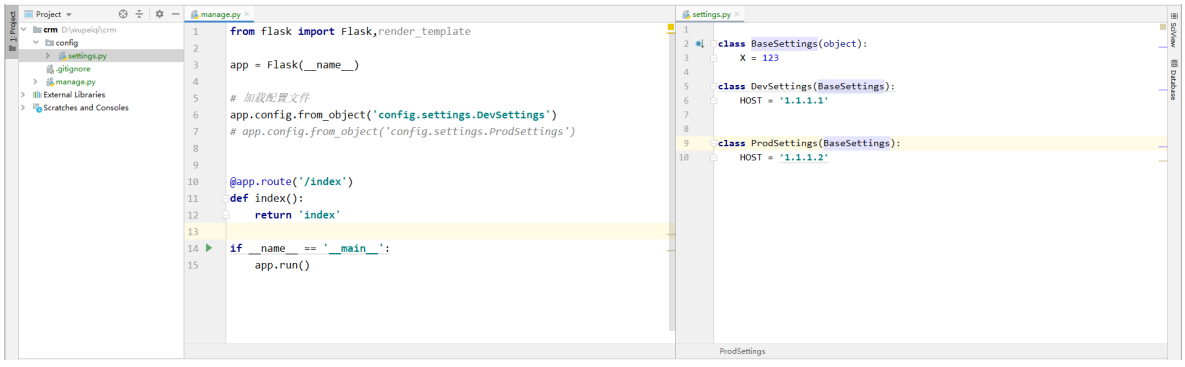
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>首页</h1>
    <img src="/static/xx/xx/mm.jpg" />

    <!-- 建议 -->
    <img src="{{ url_for('static',filename='xx/xx/mm.jpg')}}" />
</body>
</html>
```

3.配置文件

3.1 基于全局变量



3.2 基于类的方式



4.路由系统

- 路由的两种写法

```
def index():
    return render_template('index.html')
app.add_url_rule('/index', 'index', index)


# 公司里一般用这种方式
@app.route('/login')
def login():
    return render_template('login.html')
```

- 路由加载的源码流程

```
- 将url和函数打包成为 rule 对象
- 将rule对象添加到map对象中。
- app.url_map = map对象
```

- 动态路由

```
@app.route('/login')
def login():
    return render_template('login.html')

@app.route('/login/<name>')
def login(name):
    print(type(name))
    return render_template('login.html')

@app.route('/login/<int:name>')
def login(name):
    print(type(name))
    return render_template('login.html')
```

- 支持正则表达式的路由

```
from flask import Flask,render_template

app = Flask(__name__)


from werkzeug.routing import BaseConverter
class RegConverter(BaseConverter):
    def __init__(self, map, regex):
        super().__init__(map)
        self.regex = regex
app.url_map.converters['regex'] = RegConverter

@app.route('/index/<regex("\d+"):x1>')
def index(x1):
    return render_template('index.html')

if __name__ == '__main__':
```

```
    app.run()
```

5.视图

5.1 FBV

```
def index():
    return render_template('index.html')
app.add_url_rule('/index', 'index', index)


# 公司里一般用这种方式
@app.route('/login')
def login():
    return render_template('login.html')
```

5.2 CBV

```
from flask import Flask,render_template,views

app = Flask(__name__,)

def test1(func):
    def inner(*args,**kwargs):
        print('before1')
        result = func(*args,**kwargs)
        print('after1')
        return result
    return inner

def test2(func):
    def inner(*args,**kwargs):
        print('before2')
        result = func(*args,**kwargs)
        print('after2')
        return result
    return inner


class UserView(views.MethodView):
    methods = ['GET',"POST"]

    decorators = [test1,test2]


    def get(self):
        print('get')
        return 'get'

    def post(self):
        print('post')
        return 'post'

app.add_url_rule('/user', view_func=UserView.as_view('user')) # endpoint
```

```
if __name__ == '__main__':
    app.run()
```

6.模板

6.1 基本用法

flask比django更加接近Python。

```
from flask import Flask,render_template

app = Flask(__name__,)

def func(arg):
    return '你好' + arg

@app.route('/md')
def index():
    nums = [11,222,33]
    return render_template('md.html',nums=nums,f=func)


if __name__ == '__main__':
    app.run()
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>头</h1>
        {% block content %} {% endblock %}
    <h1>底</h1>
</body>
</html>
```

```
<form action="">
    <input type="text">
    <input type="text">
    <input type="text">
    <input type="text">
    <input type="text">
</form>
```

```
{% extends 'layout.html' %}


{% block content %}
    <h1>MD</h1>
    {% include 'form.html' %}
    {{ f("汪洋") }}
{% endblock %}
```

6.2 定义全局模板方法

```
from flask import Flask,render_template

app = Flask(__name__,)

@app.template_global() #  {{ func("赵海宇") }}
def func(arg):
    return '海狗子' + arg

@app.template_filter() # {{ "赵海宇"|x1("孙宇") }}
def x1(arg,name):
    return '海狗子' + arg + name


@app.route('/md/hg')
def index():
    return render_template('md_hg.html')

if __name__ == '__main__':
    app.run()
```

注意：在蓝图中注册时候，应用返回只有本蓝图。

7.特殊装饰器

```
from flask import Flask,render_template,request

app = Flask(__name__)

@app.before_request
def f1():
    if request.path == '/login':
        return
    print('f1')
    # return '123'

@app.after_request
def f10(response):
    print('f10')
    return response

@app.route('/index')
def index():
    print('index')
    return render_template('index.html')
```

```
if __name__ == '__main__':
    app.run()
```

多个装饰器

```
from flask import Flask,render_template,request

app = Flask(__name__)

@app.before_request
def f1():
    print('f1')

@app.before_request
def f2():
    print('f2')

@app.after_request
def f10(response):
    print('f10')
    return response

@app.after_request
def f20(response):
    print('f20')
    return response

@app.route('/index')
def index():
    print('index')
    return render_template('index.html')

if __name__ == '__main__':
    app.run()
    app.__call__
```

注意：before_after request可以在蓝图中定义，在蓝图中定义的话，作用域只在本蓝图。

8.小细节

```
from flask import Flask,render_template

app = Flask(__name__,)

@app.route('/index')
def index():
    return render_template('index.html')


@app.before_request
def func():
    print('xxx')

def x1():
    print('xxx')
```

```
app.before_request(x1)

if __name__ == '__main__':
    app.run()
```

赠送：threading.local

```
import time
import threading

# 当每个线程在执行 val1.xx=1 ，在内部会为此线程开辟一个空间，来存储 xx=1
# val1.xx,找到此线程自己的内存地址去取自己存储 xx
val1 = threading.local()

def task(i):
    val1.num = i
    time.sleep(1)
    print(val1.num)

for i in range(4):
    t = threading.Thread(target=task,args=(i,))
    t.start()
```