day95 flask+发布系统

内容回顾

1. flask和django的区别?
2. flask请求的声明周期?

```
wsgi
before
视图
after
```

3. wsgi的本质

```
socket服务端
```

4. flask的蓝图的作用?

```
默认flask通过一个脚本就可以实现，但由于项目业务可能会比较多，为了方便业务的拆分，我们会创建多个文件（蓝图）进行管理。
```

5. flask的蓝图和django的app有什么区别?

```
相同点：都是用于做业务拆分 / 需要注册才能使用 / 都可以在自己内部定义模板和静态文件。
不同点：注册位置不同 、flask before/after_request和django中间件他的应用粒度更细 、django的app内置了很多，flask蓝图没有内置。
```

6. 在app=Flask()对象中可以传入 静态文件、模板的配置。
7. 通过app.config读取配置文件（localsettings.py）。
8. 特殊装饰器

```
before_first_reqeust
before_request
after_request
route
template_global
template_filter
```

9. 为flask的视图设置装饰器时，需要

```
- 位置
- functools.wraps
- functools.partial
```

10. Flask可以定义FBV和CBV
11. flask中内置了session，session的数据以加密的形式放入cookie中。
12. flask自己没有模板，而是用的第三方的jinja2模板。

13. threading.local
14. 自定义threading.local，内部维护了一个字典，以 线程/协程 ID为键。
15. 栈的特点：后进先出。
16. LocalStack的类，在Local中维护了一个栈

```
{
    111:{stack:[ctx, ]}
}
```

17. 单利模式（基于文件导入实现）

```
在flask程序中，只会创建两个LocalStack对象。

_request_ctx_stack = LocalStack()
_app_ctx_stack = LocalStack()
```

18. flask的请求流程
    - 请求准备阶段
    - 请求到来
        - 创建两个ctx=RequestContext对象； app_ctx = AppContext对象，将对象放入local中。
        - before/视图/after
        - 销毁ctx / app_ctx
19. g是什么?
20. 数据库链接池：DBUtils
21. 面向对象的with上下文：enter/exit

内容补充

1. 从看flask源码你学到了什么?

```
- 新的编程思路。
    - django、drf数据是通过传递。
    - flask，存储在某个地方，以后用的时候去拿。
    哪种好？两个不同的实现机制，没有好坏之分。
    django好，疑问如果是一个初学者对于django的机制比较好理解，flask学习代价比较大（了解上下文管理职机制之后才能更好的理解）。

- 技术点
    - 单利模式的应用场景
    - LocalProxy
    - 装饰器不注意functools
```

2. 在flask的Local对象中为什么要通过线程ID进行区分?

```
因为在flask中可以开启多线程的模式，当开启多线程模式进行处理用户请求时，需要将线程之间的数据进行隔离，以防止数据混乱。
```

3. 在flask的Local对象中为什么要维持成一个栈?

```
{
    111:{stack:[ctx, ]},
}
```

```
在web runtime 时，栈永远只有1个对象。
    {
        111:{stack:[ctx, ]}
        112:{stack:[ctx, ]}
    }
    {
        111:{stack:[app_ctx, ]}
        112:{stack:[app_ctx, ]}
    }
```

```
在写离线脚本时，才会用在栈中放多个对象。 （创建一个py文件本地运行）

from flask import current_app,g
from pro_excel import create_app

app1 = create_app()
with app1.app_context():  # AppContext对象(app,g) -> local对象
    print(current_app.config) # -1 top app1
    app2 = create_app()
    with app2.app_context():  # AppContext对象(app,g) -> local对象
        print(current_app.config) # top -1 app2
    print(current_app.config) # top -1 app1

写离线脚本且多个上下文嵌套时，才会在栈中添加多个对象。
```

注意：在flask中很少出现嵌套的脚本。

杨泽涛爱喝豆浆

今日概要

- 信号
- flask-script组件
- 蓝图
- 前置知识点：下载代码、上传代码
  - git下载代码
  - git上传代码和执行命令
  - 登录

今日详细

1. 信号（源码）

信号，是在flask框架中为我们预留的钩子，让我们可以进行一些自定义操作。

```
pip3 install blinker
```

根据flask项目的请求流程来进行设置扩展点

- 中间件

```
from flask import Flask,render_template

app = Flask(__name__)


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


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


class MyMiddleware(object):
    def __init__(self,old_app):
        self.wsgi_app = old_app.wsgi_app

    def __call__(self, *args, **kwargs):
        print('123')
        result = self.wsgi_app(*args, **kwargs)
        print('456')
        return result

app.wsgi_app = MyMiddleware(app)

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

- 当app_ctx被push到local中栈之后，会触发appcontext_pushed信号，之前注册在这个信号中的方法，就会被执行。

```
from flask import Flask,render_template
from flask import signals

app = Flask(__name__)

@signals.appcontext_pushed.connect
def f1(arg):
    print('appcontext_pushed信号f1被触发',arg)

@signals.appcontext_pushed.connect
def f2(arg):
    print('appcontext_pushed信号f2被触发',arg)

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

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

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

- 执行before_first_request扩展

```
from flask import Flask,render_template

app = Flask(__name__)

@app.before_first_request
def f2():
    print('before_first_requestf2被触发')

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

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

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

- request_started信号

```
from flask import Flask,render_template
from flask import signals
app = Flask(__name__)

@signals.request_started.connect
def f3(arg):
    print('request_started信号被触发',arg)

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

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

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

- url_value_processor

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)


@app.url_value_preprocessor
```

```
def f5(endpoint,args):
    print('f5')

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

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

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

- before_reuqest

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@app.before_request
def f6():
    g.xx = 123
    print('f6')

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

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

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

- 视图函数
- before_render_template / rendered_template

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@signals.before_render_template.connect
def f7(app, template, context):
    print('f7')

@signals.template_rendered.connect
def f8(app, template, context):
    print('f8')

@app.route('/index/')
```

```
def index():
    return render_template('index.html')

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

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

- after_request

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@app.after_request
def f9(response):
    print('f9')
    return response

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

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

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

- request_finished

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@signals.request_finished.connect
def f10(app,response):
    print('f10')

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

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

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

- got_request_exception

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@app.before_first_request
def test():
    int('asdf')

@signals.got_request_exception.connect
def f11(app,exception):
    print('f11')

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

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

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

- teardown_request

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@app.teardown_request
def f12(exc):
    print('f12')

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

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

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

- request_tearing_down

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@signals.request_tearing_down.connect
def f13(app,exc):
```

```
    print('f13')

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

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

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

- appcontext_popped

```
from flask import Flask,render_template,g
from flask import signals
app = Flask(__name__)

@signals.appcontext_popped.connect
def f14(app):
    print('f14')

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

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

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

总结：关于flask内部共有14+个扩展点用于我们对flask框架内部进行定制，其中有：9个是信号。

```
template_rendered = _signals.signal("template-rendered")
before_render_template = _signals.signal("before-render-template")
request_started = _signals.signal("request-started")
request_finished = _signals.signal("request-finished")
request_tearing_down = _signals.signal("request-tearing-down")
got_request_exception = _signals.signal("got-request-exception")
appcontext_tearing_down = _signals.signal("appcontext-tearing-down")
appcontext_pushed = _signals.signal("appcontext-pushed")
appcontext_popped = _signals.signal("appcontext-popped")

message_flashed = _signals.signal("message-flashed")
```

扩展:flash

flash存值之后只能取一次

```
from flask import Flask,render_template,flash,get_flashed_messages,session
from flask import signals
app = Flask(__name__)
app.secret_key = 'iuknsoiuwknlskjdf'

@app.route('/index/')
def index():
    # flash('123')
    session['k1'] = 123
    return render_template('index.html')

@app.route('/order')
def order():
    # messages = get_flashed_messages()
    # print(messages)
    val = session['k1']
    del session['k1']
    print(val)
    return render_template('order.html')

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

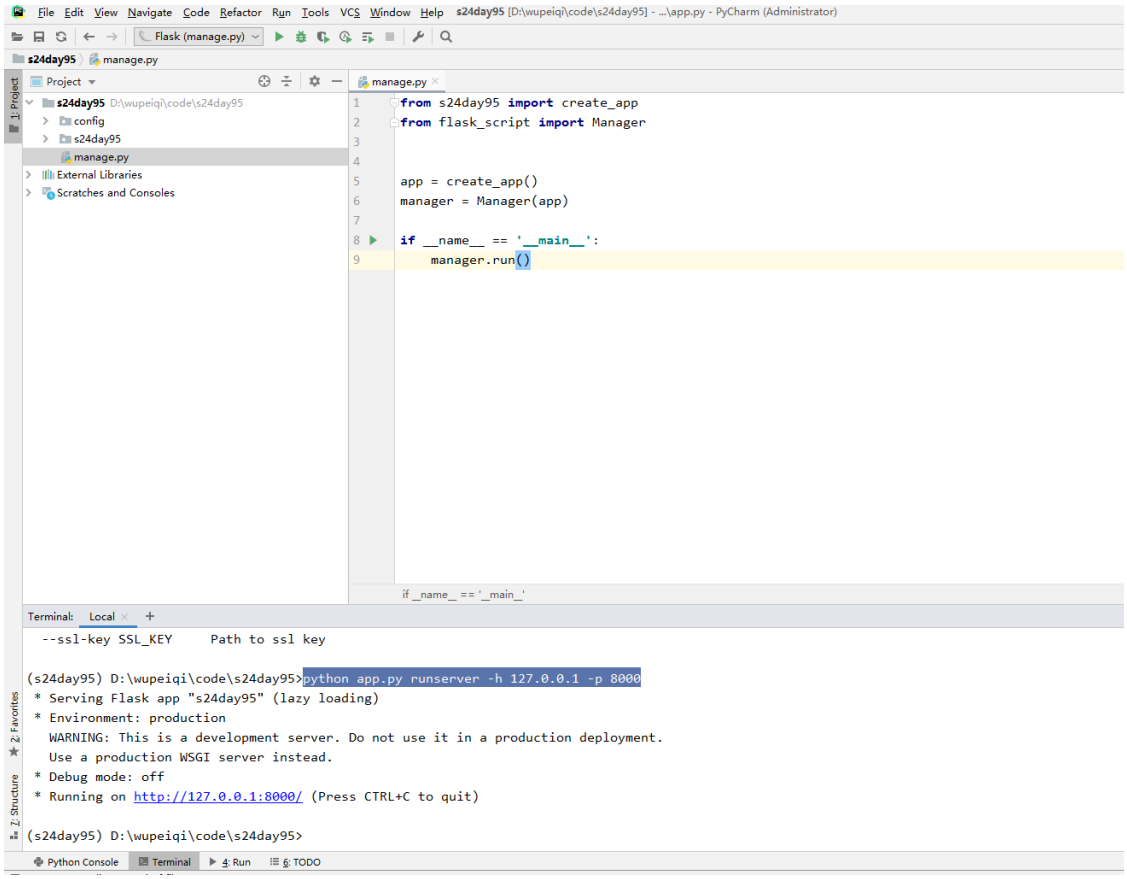
2.flask-script

flask的组件，用于运行flask程序。

- 安装

```
pip3 install flask-script
```

- 使用



- 其他执行命令

```
from s24day95 import create_app
from flask_script import Manager

app = create_app()
manager = Manager(app)

@manager.command
def custom(arg):
    """
    自定义命令
    python manage.py custom 123
    :param arg:
    :return:
    """
    print(arg)

@manager.option('-n', '--name', dest='name')
@manager.option('-u', '--url', dest='url')
def cmd(name, url):
    """
    自定义命令
    执行： python manage.py  cmd -n wupeiqi -u http://www.oldboyedu.com
    :param name:
    :param url:
    :return:
    """
    print(name, url)

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

- 其他

```
结合：flask-migrate / flask-sqlalchemy

python manage.py migrate
```

3.蓝图

目录结构的划分

- 分功能蓝图：s24day95
- 分结构蓝图：bigblue

4.个人作业

基于python实现会议室预定系统（flask）+ DBUtils。

阶段总结

- flask基本使用
- flask源码系列博客
  - 博客
  - github，创建一个仓库，在里面写markdon

预习

https://www.cnblogs.com/wupeiqi/articles/9805296.html