小白学Flask第十四天 | 一文带你彻底了解蓝图是啥！

原创JAP君 Python进击者
2019-12-02原文

Flask系列文章：

小白学Flask第一天 | 我的第一个Flask程序

小白学Flask第二天| app对象的初始化和配置

小白学Flask第三天| 今天把视图函数的路由给讲清楚！

小白学Flask第四天| 把路由转换器玩的更牛逼

小白学Flask第五天 | 详解很重要的request对象

小白学Flask第六天| abort函数、自定义错误方法、视图函数的返回值

小白学Flask第七天| 讲讲cookie和session的操作

小白学Flask第八天| Flask上下文和请求钩子

小白学Flask第九天| 看看模板的那些事（一）

小白学Flask第十天| 宏、继承、包含、特殊变量

小白学Flask第十一天| flask-sqlalchemy数据库扩展包(一)

小白学Flask第十二天| flask-sqlalchemy数据库扩展包(二)

小白学Flask第十三天| 来谈谈数据库迁移、邮箱扩展的那些事！

我用Flask写了一个图书作者管理项目(附完整代码)

主要内容：

1. 为什么学习蓝图？
2. 蓝图是个啥
3. 实战蓝图

为什么要学习蓝图？

我们学习Flask框架，是从写单个文件，执行hello world开始的。我们在这单个文件中可以定义路由、视图函数、定义模型等等。

但这显然存在一个问题：**随着业务代码的增加，将所有代码都放在单个程序文件中，是非常不合适的。这不仅会让代码阅读变得困难，而且会给后期维护带来麻烦。**

如下示例：我们在一个文件中写入多个路由，这会使代码维护变得困难。

```
from flask import Flask

    app = Flask(__name__)

    @app.route('/')

    def index():

        return 'index'


    @app.route('/list')

    def list():

        return 'list'


    @app.route('/detail')

    def detail():
```

```
    return 'detail'


@app.route('/')
def admin_home():
    return 'admin_home'


@app.route('/new')
def new():
    return 'new'


@app.route('/edit')
def edit():
    return 'edit'
```

问题：一个程序执行文件中，功能代码过多。

所以我们需要让代码模块化。**根据具体不同功能模块的实现，划分成不同的分类**，降低各功能模块之间的耦合度。python中的模块制作和导入就是基于实现功能模块的封装的需求。

尝试用模块导入的方式解决： 我们把上述一个py文件的多个路由视图函数给拆成两个文件：app.py和admin.py文件。app.py文件作为程序启动文件，因为admin文件没有应用程序实例app，在admin文件中要使用app.route路由装饰器，需要把app.py文件的app导入到admin.py文件中。

app.py ：

```
# 文件app.py
from flask import Flask
# 导入admin中的内容
from admin import *
app = Flask(__name__)

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

@app.route('/list')
def list():
    return 'list'

@app.route('/detail')
def detail():
    return 'detail'

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

admin.py :

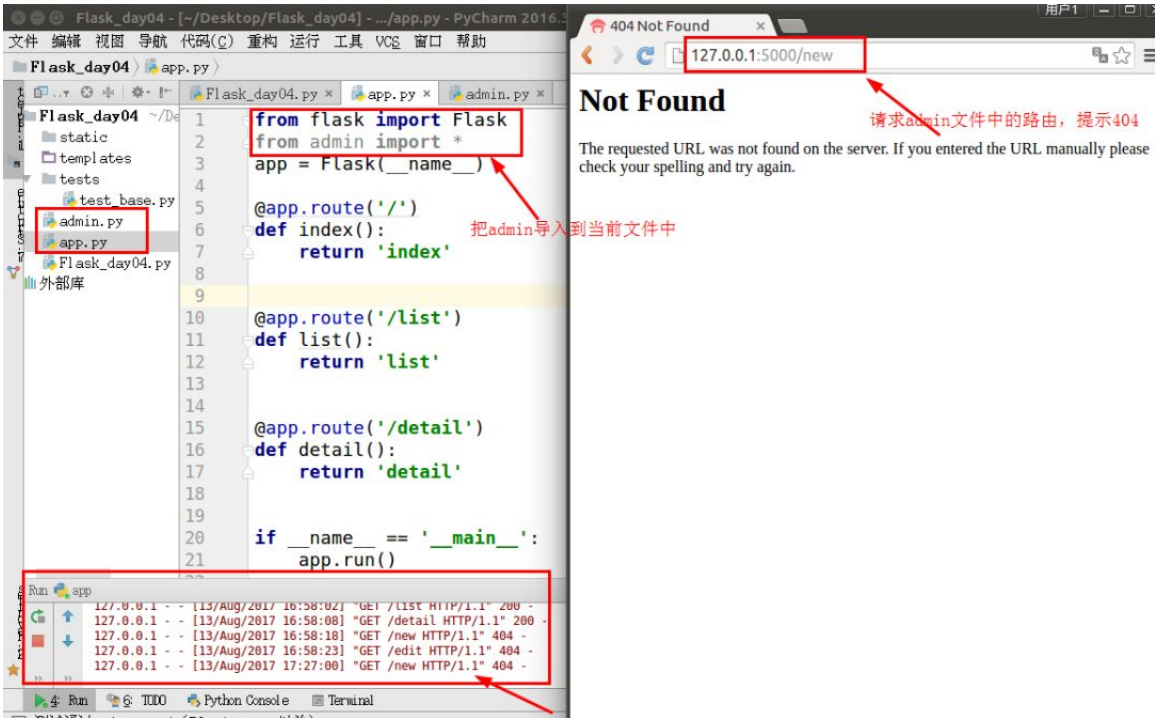
```
# 文件admin.py
from app import app
```

```
@app.route('/')
def admin_home():
    return 'admin_home'


@app.route('/new')
def new():
    return 'new'


@app.route('/edit')
def edit():
    return 'edit'
```

启动app.py文件后，我们发现admin.py文件中的路由都无法访问。 也就是说，**python中的模块化虽然能把代码给拆分开，但不能解决路由映射的问题。**



因此我们就引出了蓝图的概念

蓝图是个啥？

蓝图：

用于实现单个应用的视图、模板、静态文件的集合。

通俗点讲**蓝图就是模块化处理的类**，更加具体点讲，**蓝图就是一个存储操作路由映射方法的容器，主要用来实现客户端请求和URL相互关联的功能。在Flask中，使用蓝图可以帮助我们实现模块化应用的功能。**

蓝图是怎么运行的？

蓝图是**保存了一组将来可以在应用对象上执行的操作。**

注册路由就是一种操作,当在程序实例上调用route装饰器注册路由时，这个操作将修改对象的url_map路由映射列表。当我们在蓝图对象上调用route装饰器注册路由时，它只是在内部的一个延迟操作记录列表defered_functions中添加了一个项。

当执行应用对象的 register_blueprint() 方法时，应用对象从蓝图对象的 defered_functions 列表中取出每一项，即调用应用对象的 add_url_rule() 方法，这将会修改程序实例的路由映射列表。

实战一下，如何在代码中实现蓝图

一、创建蓝图对象

```
#Blueprint必须指定两个参数，admin表示蓝图的名称，__name__表示蓝图所在模块
admin = Blueprint('admin',__name__)
```

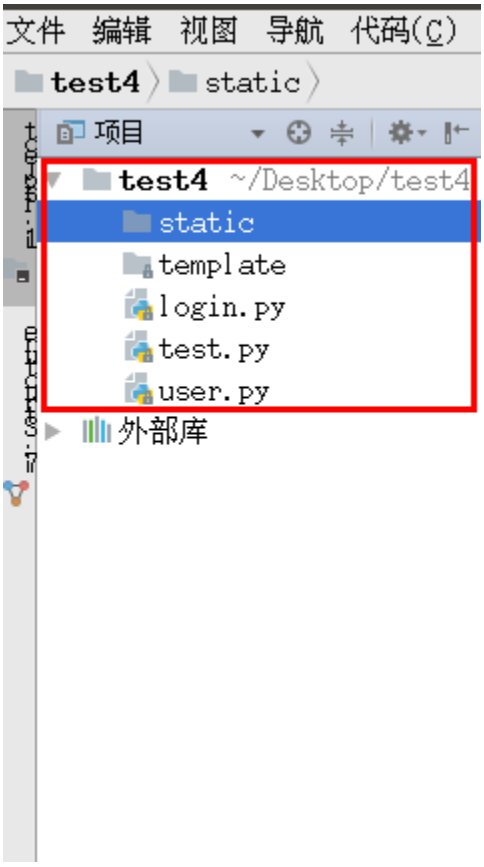
二、注册蓝图路由

```
# 注意下面是admin对象的route方法
@admin.route('/')
def admin_index():
    return 'admin_index'
```

三、在程序实例中注册该蓝图

```
app.register_blueprint(admin,url_prefix='/admin')
```

上面只是简单的三个步骤，下面来通过一个完整的代码来给大家实现一下



创建蓝图：

user.py

```
from flask import Blueprint,render_template

#创建蓝图，第一个参数指定了蓝图的名字。

users = Blueprint('user',__name__)
```

```
@users.route('/user')
def user():
    return render_template('user.html')
```

login.py

```
from flask import Blueprint,render_template
#创建蓝图
logins = Blueprint('login',__name__)

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

程序执行文件 test.py

```
from flask import Flask
#导入蓝图对象
from login import logins
from user import users

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'
```

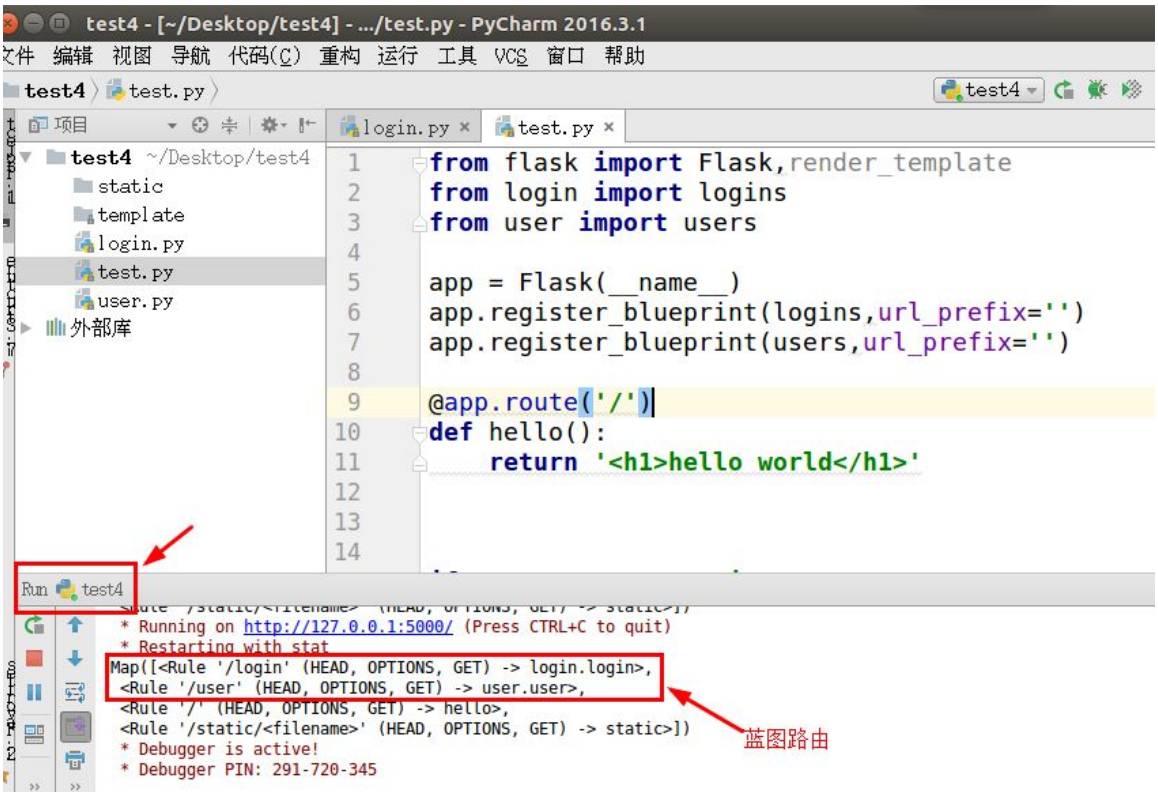
#注册蓝图，第一个参数logins是蓝图对象，url_prefix参数默认值是根路由，如果指定，会在蓝图注册的路由url中添加前缀。

```
app.register_blueprint(logins,url_prefix='')

app.register_blueprint(users,url_prefix='')


if __name__ == '__main__':

    app.run(debug=True)
```



精选留言

暂无...