

1.蓝图引入

思考：

前面几天的写flask项目都是将全部逻辑写在一个py文件里面的，假如我们大型项目，如果全部代码都写到一个文件里面去，几千行代码甚至上万行的代码。如果出现bug，你要从成千上万行的代码中去找bug你会吐血的。

显然按照常规的方法将所有视图写在同一文件不是合适的。

解决这个问题我们首先想到的应该就是模块化。

按照不同的功能分成不同的模块。

试一试：

既然我们想到将视图函数提取出来分成不同模块。那么我们来试试，分成不同模块是否可行。

首先我们创建一个商城：其中有四大基础模块：

购物车模块，用户模块，登录注册模块，商品模块

常规写法

```
# -*- encoding: utf-8 -*-
from flask import Flask

app = Flask(__name__) # type:Flask

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

```
        return 'index'

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

@app.route('/login')
def login():
    return 'cart'

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

if __name__ == '__main__':
    print(app.url_map)
    app.run(debug=True)
```

```
D:\EVNS\flask_bei\Scripts\python.exe D:/pythonfile/pythonProject/demo/main.py
Map([<Rule '/static/<filename>' (HEAD, GET, OPTIONS) -> static>,
<Rule '/' (HEAD, GET, OPTIONS) -> index>,
<Rule '/cart' (HEAD, GET, OPTIONS) -> cart>,
<Rule '/login' (HEAD, GET, OPTIONS) -> login>,
<Rule '/goods' (HEAD, GET, OPTIONS) -> goods>])
* Serving Flask app 'main'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
```

模块化操作

```
4 from app.login import login
5 from app.cart import cart
6
7 app = Flask(__name__) # type:Flask
8
9
10 if __name__ == '__main__':
11     print(app.url_map)
12     app.run(debug=True)
```

```
1 # -*- encoding: utf-8 -*-
2 from app.mall import app
3
4
5 @app.route('/cart')
6 def cart():
7     return 'cart'
8
```

Run: mall (1) x

D:\EVNS\flask_bei\Scripts\python.exe D:/pythonfile/pythonProject/app/mall.py

Traceback (most recent call last):

File "D:/pythonfile/pythonProject/app/mall.py", line 3, in <module>

from app.goods import goods

File "D:/pythonfile/pythonProject/app/goods.py", line 4, in <module>

@app.route('/goods')

NameError: name 'app' is not defined

Process finished with exit code 1

发现 `mall.py` 与其他模块没有任何联系，那是不是应该将模块导进来，导入之后会发现一个循环导入的问题，因为模块中需要与 `app` 对象管理导入了 `mall` 模块。

所以 `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。

`Flask` 中为了解决这个问题，提供了蓝图（`blueprint`）功能，它的主要功能是使模块式开发成为可能。

简单来说，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。

2. 建立蓝图

第一步：创建蓝图对象

```
# -*- encoding: utf-8 -*-
from flask import Blueprint
```

创建蓝图对象，第一个参数是 蓝图的名字，第二个参数 `name` 表示蓝图所在模块，会以这个为根目录寻找静态文件，很模板文件。

```
app_cart = Blueprint('app_cart', __name__)
```

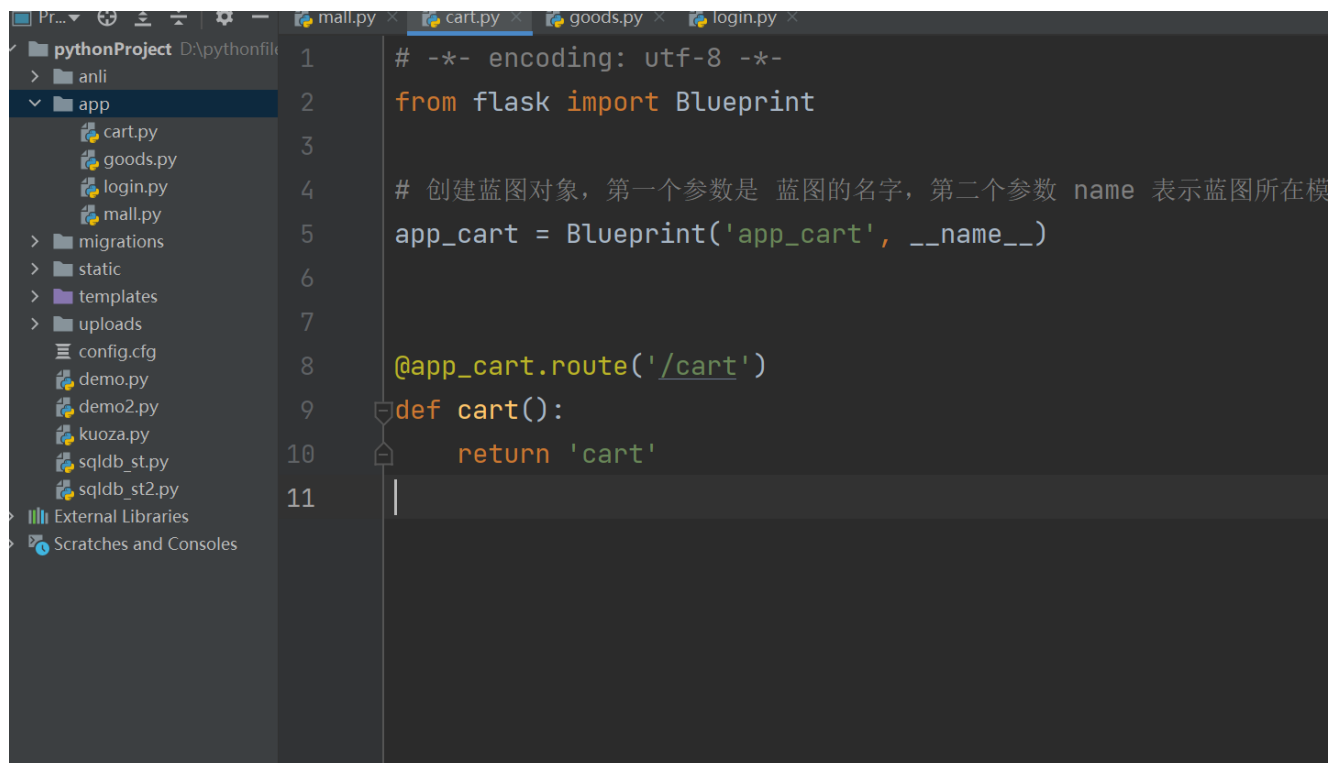
第二步：注册蓝图路由

装饰路由时，使用蓝图对象。

```
@app_cart.route('/cart')
```

```
def cart():
```

```
    return 'cart'
```



第三步：在程序实例中注册该蓝图

```
# -*- encoding: utf-8 -*-
```

```
from flask import Flask
```

```
from app.goods import app_goods
```

```
from app.login import app_login
```

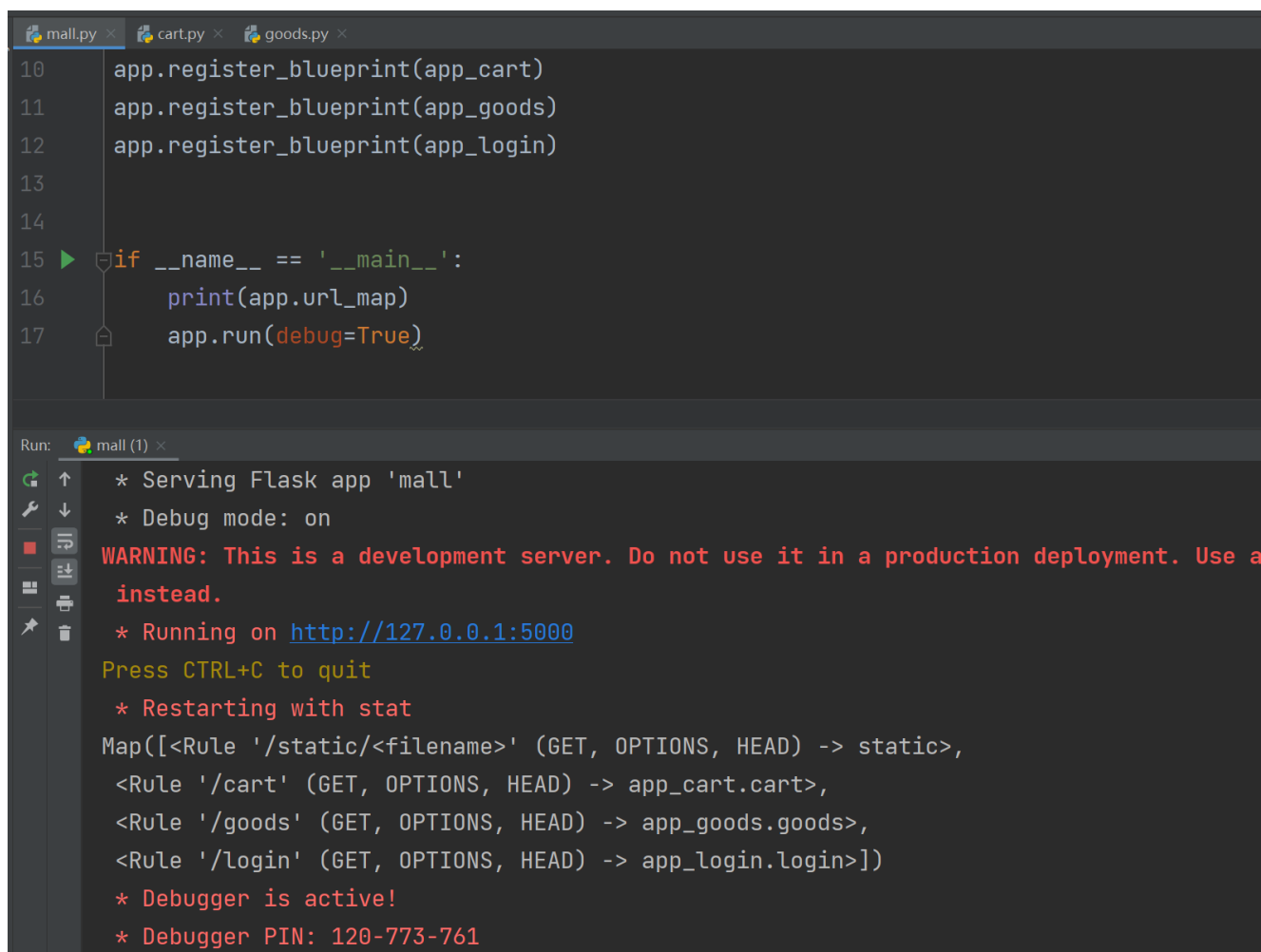
```
from app.cart import app_cart
```

```
app = Flask(__name__) # type:Flask
```

注册蓝图

```
app.register_blueprint(app_cart)
app.register_blueprint(app_goods)
app.register_blueprint(app_login)
```

```
if __name__ == '__main__':
    print(app.url_map)
    app.run(debug=True)
```



The screenshot shows a code editor with three tabs: mall.py, cart.py, and goods.py. The mall.py tab is active, showing the following code:

```
10 app.register_blueprint(app_cart)
11 app.register_blueprint(app_goods)
12 app.register_blueprint(app_login)
13
14
15 if __name__ == '__main__':
16     print(app.url_map)
17     app.run(debug=True)
```

Below the code editor is a terminal window titled 'Run: mall (1) x'. It displays the following output:

```
* Serving Flask app 'mall'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a
instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
Map([<Rule '/static/<filename>' (GET, OPTIONS, HEAD) -> static>,
<Rule '/cart' (GET, OPTIONS, HEAD) -> app_cart.cart>,
<Rule '/goods' (GET, OPTIONS, HEAD) -> app_goods.goods>,
<Rule '/login' (GET, OPTIONS, HEAD) -> app_login.login>])
* Debugger is active!
* Debugger PIN: 120-773-761
```

使用蓝图后查看url_map 观察注册的路由:

蓝图路由可以分为两块, "."前面的是蓝图名称, "."后面的是视图函数名。

3.蓝图运行机制

蓝图是保存了一组将来可以在应用对象上执行的操作。

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。

```
# 强加进 url_map
```

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deferred_functions中添加了一个项。

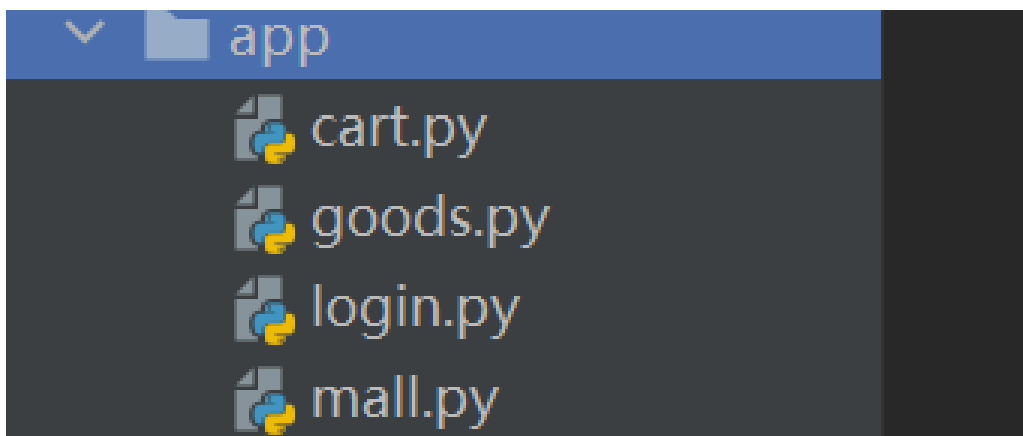
当执行应用对象的 register_blueprint()方法时，应用对象从蓝图对象的deferred_functions 列表中取出每一项，即调用应用对象的add_url_rule() 方法，这将会修改程序实例的路由映射列表。

一个Blueprint并不是一个完整的应用，它不能独立于应用运行，而是必须要注册到某一个应用中。

蓝图的基本设想是当它们注册到应用上时，它们记录将会被执行的操作。当分派请求和生成从一个端点到另一个的 URL 时，Flask 会关联蓝图中的视图函数

4.蓝图资源文件夹

与常规应用一样，蓝图应该包含在一个文件夹中。虽然多个蓝图可以与应用主文件放在相同的文件夹中，但是通常不建议这样做。



这个文件夹会从 `Blueprint` 的第二个参数中推断出来，通常是 `__name__`。

这个参数决定对应蓝图的是哪个逻辑的 `Python` 模块或包。

如果它指向一个存在的 `Python` 包，这个包（通常是文件系统中的文件夹）就是资源文件夹。

如果是一个模块，模块所在的包就是资源文件夹。

你可以访问 `Blueprint.root_path` 属性来查看 资源文件夹是什么：



```
class Blueprint(_PackageBoundObject):
    def init (self, name, import_name,
```

```
static_folder=None,static_url_path=None,  
template_folder=None,url_prefix=None, subdomain=None,  
url_defaults=None,root_path=None):
```

`static_folder` 蓝图中静态文件的查询目录

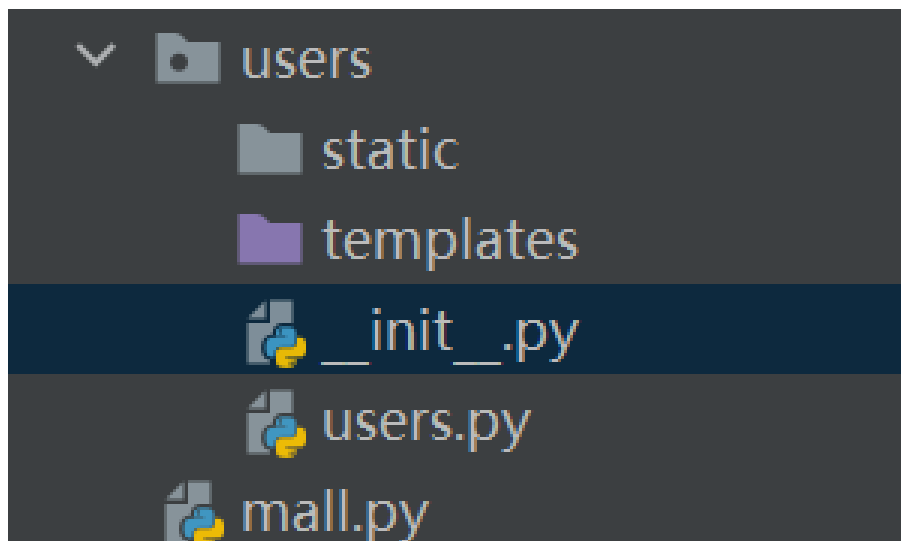
`template_folder` 蓝图中模板文件的查询目录

`url_prefix` 蓝图中url前缀（相当于django主路由）

在蓝图中还可以指定模板文件夹，和静态文件夹。

将模块单独放在一个文件下

比如 用户模块目录如下：




```
users.py × mall.py ×
1  # -*- encoding: utf-8 -*-
2  from flask import Flask
3  from users.users import app_users
4
5
6  app = Flask(__name__) # type: Flask
7
8  # 注册蓝图
9  app.register_blueprint(app_users)
10
11
12  if __name__ == '__main__':
13      print(app.url_map)
14  app.run(debug=True)
```

```
# -*- encoding: utf-8 -*-
from flask import Blueprint
```

创建蓝图对象，第一个参数是 蓝图的名字，第二个参数 `name` 表示蓝图所在模块，会以这个为根目录寻找静态文件，很模板文件。

```
app_users = Blueprint('app_users', __name__,
    template_folder='templates',
    static_folder='static',url_prefix='/users')
```

```
@app_users.route('/usersdemo')
def cart():
```

```
return 'users_demo'
```

资源文件路径可以是绝对的或是相对蓝图资源文件夹的。模板文件夹会被加入到模板的搜索路径中，但是比实际的应用模板文件夹优先级低。

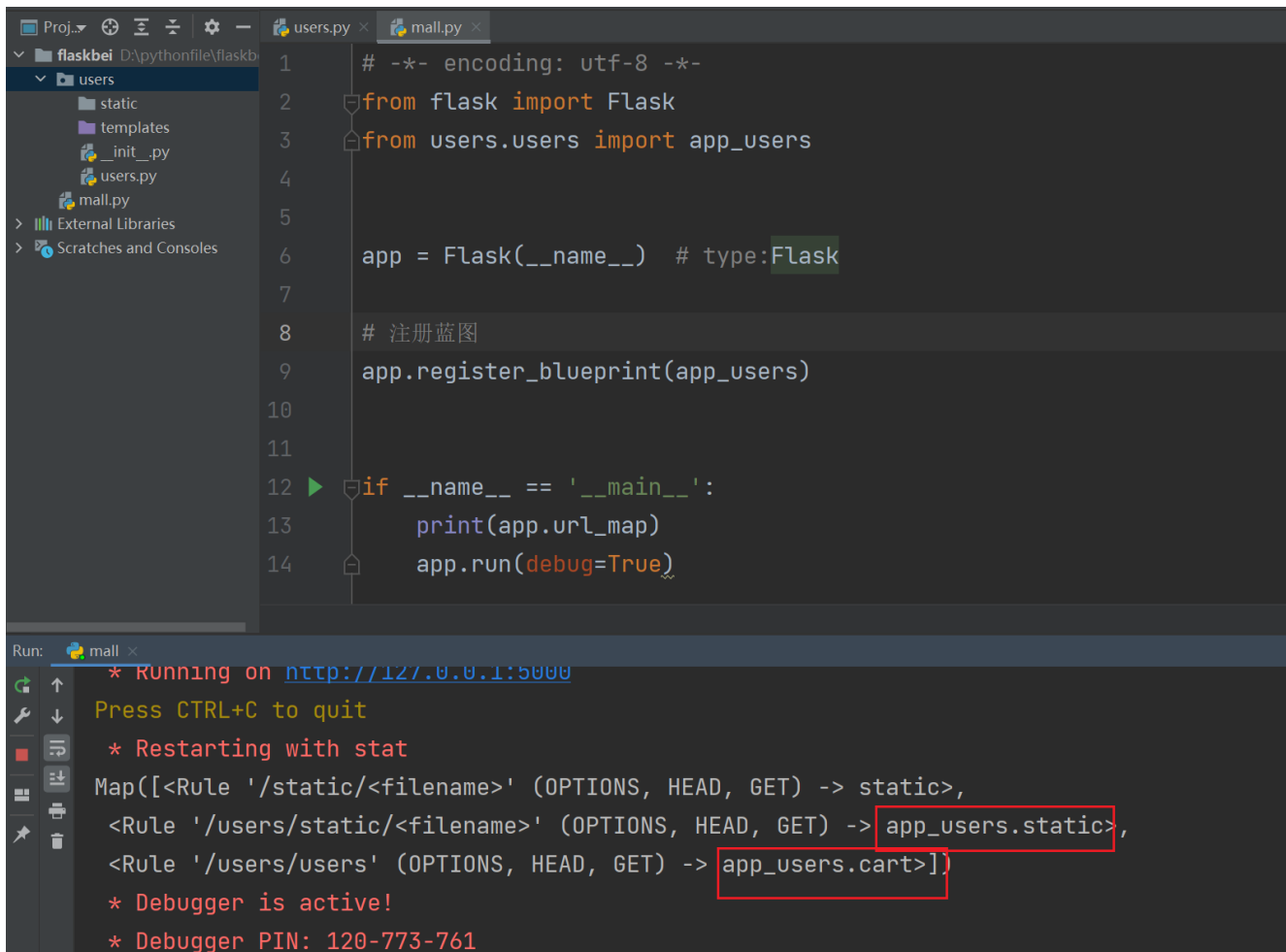
这样，你可以容易地在实际的应用中覆盖蓝图提供的模板。

5.蓝图中的url_for

在蓝图中使用`url_for`生成反向路由。

不能跟应用中一样直接使用。

需要在函数名称前面加上蓝图名称在模板中使用`url_for`



```
1 # -*- encoding: utf-8 -*-
2 from flask import Flask
3 from users.users import app_users
4
5
6 app = Flask(__name__) # type: Flask
7
8 # 注册蓝图
9 app.register_blueprint(app_users)
10
11
12 if __name__ == '__main__':
13     print(app.url_map)
14     app.run(debug=True)
```

Run: mall ×

```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
Map([<Rule '/static/<filename>' (OPTIONS, HEAD, GET) -> static>,
<Rule '/users/static/<filename>' (OPTIONS, HEAD, GET) -> app_users.static>,
<Rule '/users/users' (OPTIONS, HEAD, GET) -> app_users.cart>])
* Debugger is active!
* Debugger PIN: 120-773-761
```

使用蓝图后查看url_map 观察注册的路由：
蓝图路由可以分为两块，"."前面的是蓝图名称，"."后面的是视图函数名。

在模板中使用url_for

```
# 蓝图模板中使用url_for 需要加上蓝图名称作为前缀
<a href="{ { url_for('app_users.register') } }">注册</a>
```

视图中使用url_for

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

```
@app_users.route('/register')
def register():
    return redirect(url_for('app_users.login'))
```