

Day 6 (类)视图和装饰器，中间件

1. 视图view

1.1 视图函数

1.2 *类视图

2. *装饰器 [不改变源码，增加额外的功能]

2.1 重学闭包（大概学会了个半斤八两吧。。）

2.2 类视图装饰器

2.2.2 基本使用

3. 中间件（假）

1. 视图view

Django中的view

可以是方法(函数)

也可以是类

按照django的规则，我们添加的view都要写到app的views.py文件中

1.1 视图函数

略

(django直接调用了url对应的指定方法，中间没有什么特别的处理)

前面都用了这么久了。。。

1.2 *类视图

定义

使用类来定义的视图，称为类视图

作用

在一个类视图内，为同一个路由，提供了多种HTTP请求方式的支持（GET，POST，PUT，DELETE等）

代码可读性与复用性更好

导入

from django.views import View

```
views.py × urls.py × index.html × sales_index.html × excellent_staffs.html ×
1      # 装载redirect和render
2      from django.shortcuts import render, redirect
3
4      # 装载HttpResponse
5      from django.http import HttpResponse
6
7      # 将公用数据库common装载进来
8      from common.models import Customer
9
10     # 导入json库
11     import json
12
13     # 导入django对template的加载工具
14     from django.template.loader import get_template
15
16     # 导入类视图的父类View
17     from django.views import View
18
19
```

使用案例

1. 先写个类视图，重写get和post方法

重写的get方法，响应GET请求，处理查询

重写的POST方法，响应POST请求，处理增加

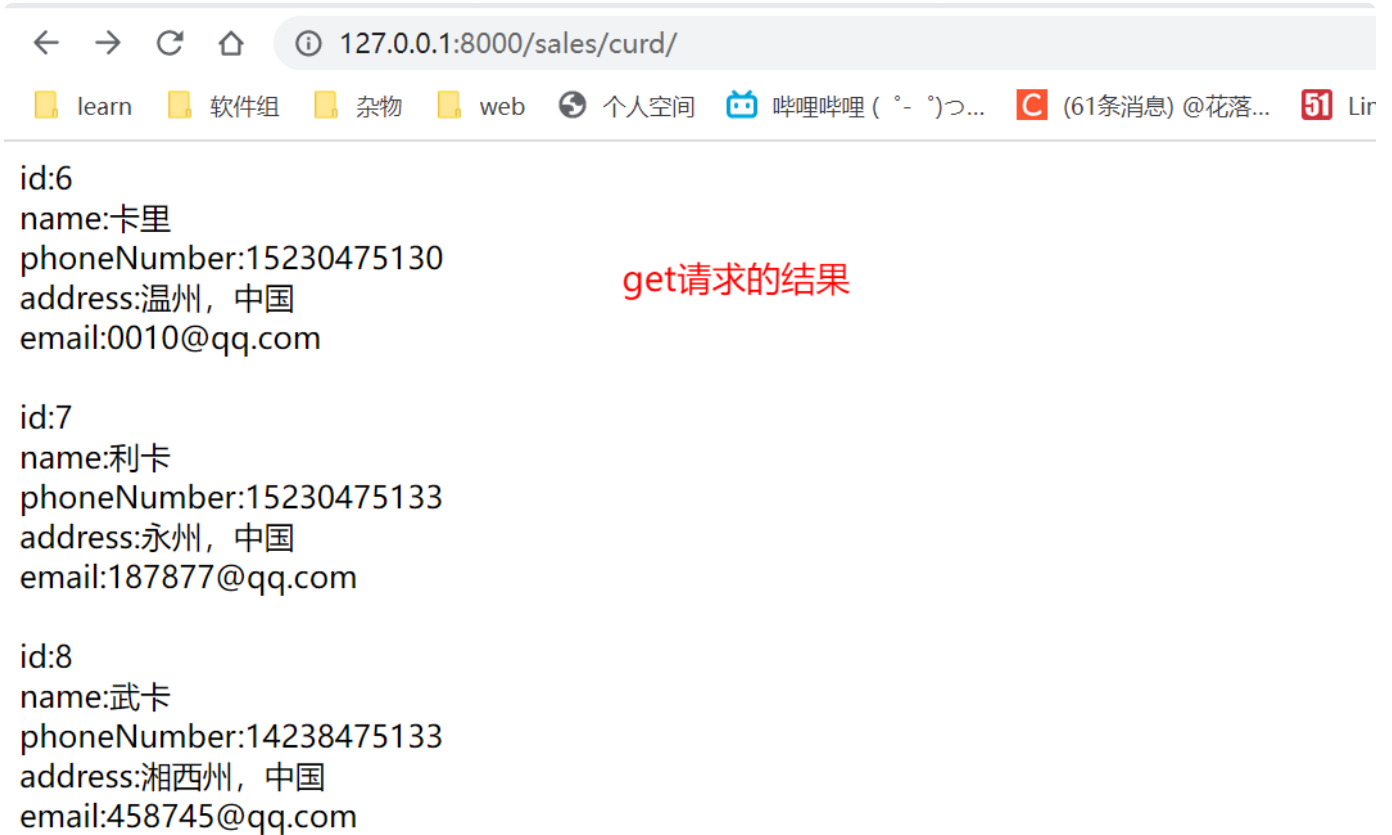
```
1  # 该类为 类视图
2  class Curd(View):                # 继承自Django的View
3      query_set = Customer.objects.all()
4
5      def get(self , request):      # 重写View的get方法用来处理查询
6          ret_Str = ""
7
8          for i in self.query_set:
9              for key, value in i.items():
10                 ret_Str += f"{key}:{value}</br>"
11                 ret_Str += "</br>"
12
13             return HttpResponse(ret_Str)
14
15     def post(self , request):      # 重写View的post方法用来处理增加
16         name = request.POST.get('name')
17
18         phoneNumber = request.POST.get('phoneNumber')
19
20         address = request.POST.get('address')
21
22         email = request.POST.get('email')
23
24         Customer.objects.create(name=name,phoneNumber=phoneNumber,address=address,email=email)
25
26         return HttpResponse("增  OK")
```

2. 绑定路由 `views.类视图名.as_view()`

`as_view()`: 相当于实例化对象

```
path('excellent/', views.show_excellent_staff),  
path('curd/', views.Curd.as_view()))
```

3. 对该路由发起GET请求，结果： (OK)



← → ↻ 🏠 ⓘ 127.0.0.1:8000/sales/curd/

📁 learn 📁 软件组 📁 杂物 📁 web 🔄 个人空间 📺 哔哩哔哩 (゜-゜)つ... 📬 (61条消息) @花落... 📺 51 Lir

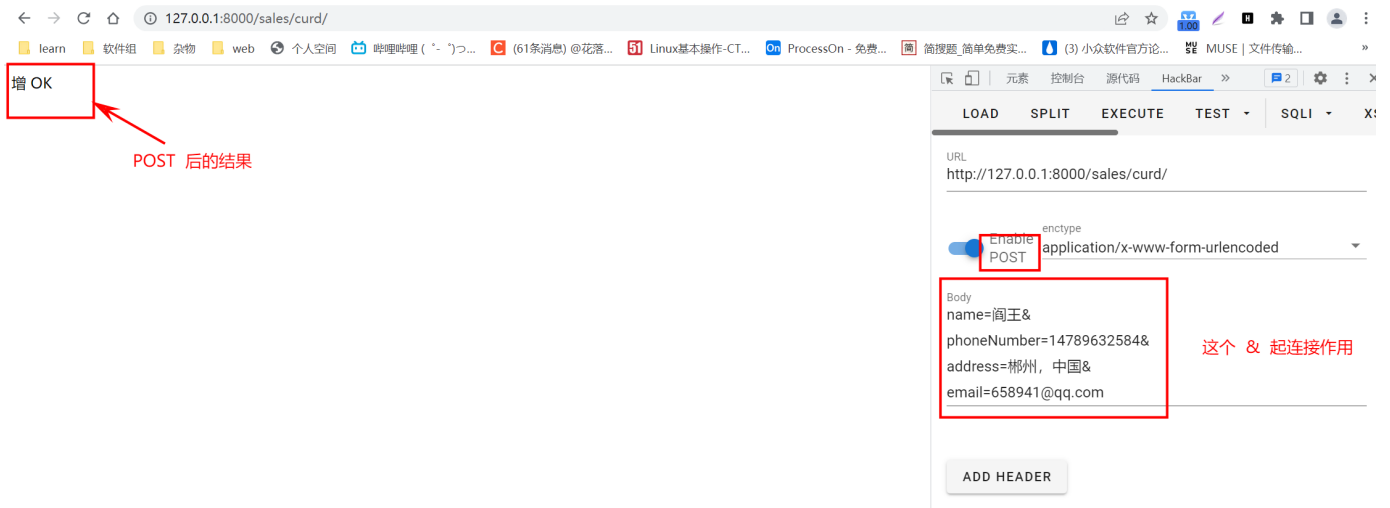
id:6
name:卡里
phoneNumber:15230475130
address:温州, 中国
email:0010@qq.com

id:7
name:利卡
phoneNumber:15230475133
address:永州, 中国
email:187877@qq.com

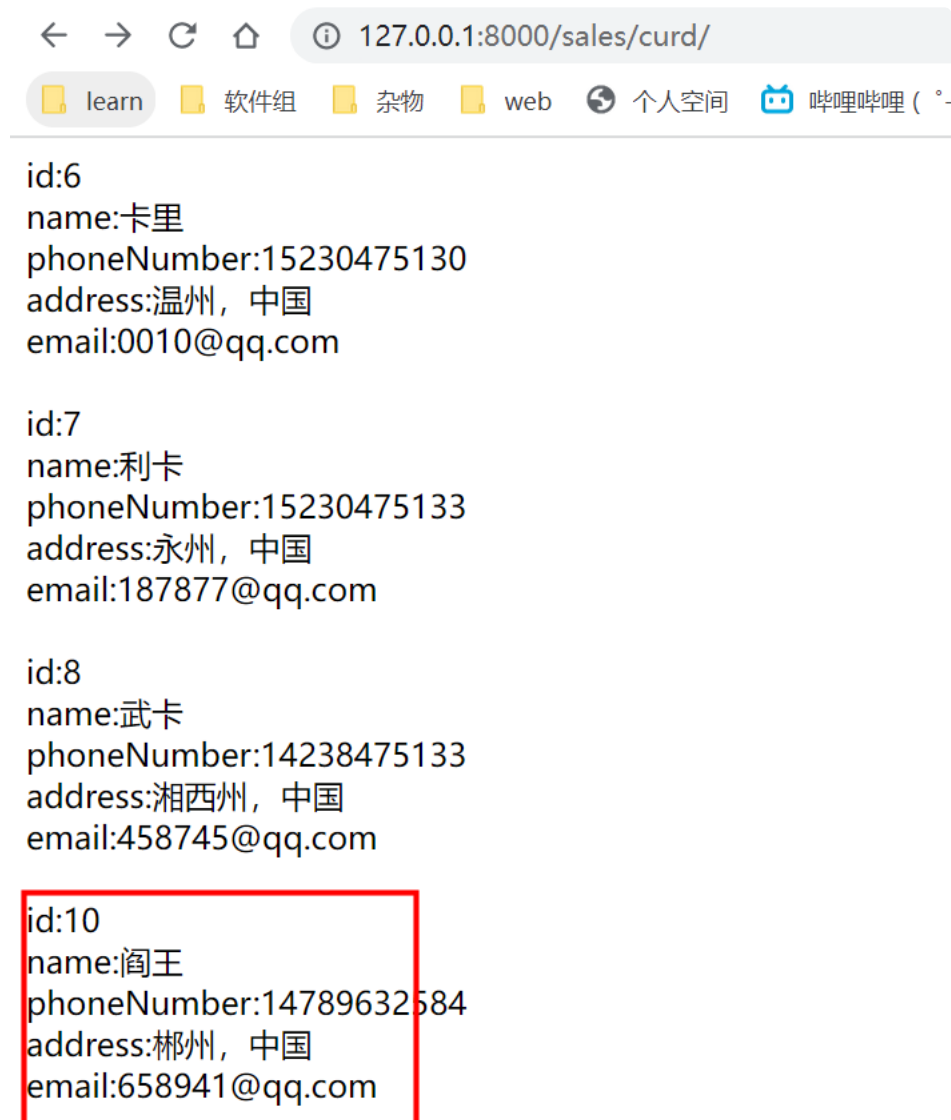
id:8
name:武卡
phoneNumber:14238475133
address:湘西州, 中国
email:458745@qq.com

get请求的结果

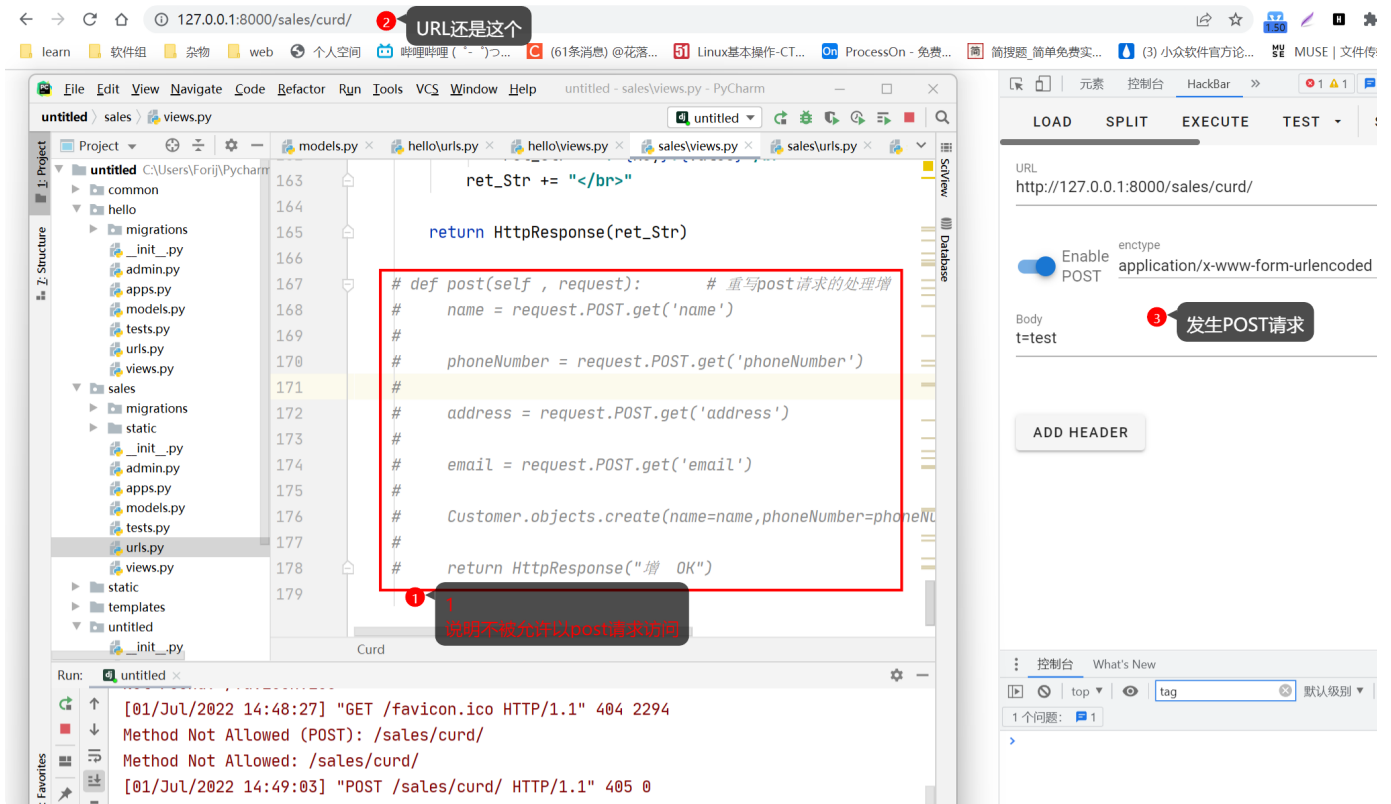
4. 对该路由发起POST请求（又用HackBar模拟POST。。记得关和开CSRF）



再次GET查询 (增加OK)



假如注释掉对post的重写



请求会被拒绝（状态码：405）

2. *装饰器 [不改变源码，增加额外的功能]

介绍

‘装饰’代指为被装饰对象添加新的功能，

‘器’代指器具/工具，

装饰器实质上就是一个闭包函数。

装饰器的返回值是一个函数对象。

作用：

可以让其他函数在不去改变任何代码的前提下增加额外的功能

2.1 重学闭包（大概学会了个半斤八两吧。。）

（主要内容在附件内）

- 定义：闭包就是能够读取外部函数内的变量的函数。
- 作用1：读取函数内部的变量
- 作用2：将外层函数的变量持久地保存在内存中。

 闭包.py

2.2 类视图装饰器

2.2.2 基本使用

【鉴于，sales子应用内容太多了，改用hello子应用操作，且初始化操作已完成】

案例一：为类视图中的所有视图方法都使用装饰器

1. 导入需要的模块

```
from django.utils.decorators import method_decorator # 对(类)方法进行装饰
from django.views import View # 自定义类视图的父类
```

2. 写闭包函数作为装饰器执行的函数（好难理解。。）

```
# 定义闭包函数作为装饰器执行的函数
def switch(func): # 选则过滤。。。 (我这个自定义闭包做装饰器功能鸡肋的很。。。)

    def way(request):
        if request.method == 'GET': # 若是GET请求，控制台打印一串字符。。。
            print("这是个GET请求，请注意查收处理")
        elif request.method == 'POST': # 若是POST请求，控制台打印一串字符。。。
            print("这是个GET请求，请注意查收处理")

        return func(request) # 框出来的就是不理解的。。。算了，记模板。。

    return way
```

重点是别忘了这个模板。。以后自定义装饰器的时候改改就好

▼ 留个模板算了。。以后改一下就能用

Python

📄 复制代码

```
1 def 闭包函数名(func):
2     def 内层函数名(request):
3         if request怎么怎么样:
4             怎么怎么样
5         else:
6             怎么怎么样
7         return func(request)
8     return 内层函数名
```

3. 写类视图，并在类视图上加装饰器

加装饰器后的类视图（为类中每个函数都加上了装饰器）

```
@method_decorator(switch,name='dispatch')# name的取值'dispatch'(全选), 'get'(仅get), 'post', 'put', 'delete', 'options'等
class Things(View):
    def get(self,request):
        return HttpResponse("this is GET")
    def post(self,request):
        return HttpResponse("this is post")
```

4. 绑定路由

添加子路由条目

```
urlpatterns = [  
    path("sayhello/", views.sayhello),  
    path("UID", views.var_uid),  
    path("class_deco/", views.Things.as_view()),  
]
```

5. 效果 (OK)

GET

The screenshot displays a Django development environment. The top part shows the code in `views.py` for the `Things` class view. The `dispatch` method is decorated with `@method_decorator` to handle GET and POST requests. The `get` method returns `HttpResponse("this is GET")`. The bottom part shows the Django development server output, which includes the message "Watching for file changes with StatReloader" and the URL `http://127.0.0.1:8000/`. The server log shows a GET request to `/hello/class_deco/` with a status of 200.

```
if request.method == 'GET':  
    print("这是个GET请求, 请注意查收")  
elif request.method == 'POST':  
    print("这是个POST请求, 请注意查收")  
  
    return func(request)  
    return way  
  
# 加装饰器后的类视图 (为类中每个函数都加上了装饰器)  
@method_decorator(switch, name='dispatch')  
class Things(View):  
    def get(self, request):  
        return HttpResponse("this is GET")  
    def post(self, request):  
        return HttpResponse("this is POST")
```

127.0.0.1:8000/hello/class_deco/

this is GET

GET

Watching for file changes with StatReloader
System check identified no issues (0 silenced).
July 01, 2022 - 16:33:59
Django version 2.2.5, using settings 'untitled.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
这是个GET请求, 请注意查收处理
[01/Jul/2022 16:37:03] "GET /hello/class_deco/ HTTP/1.1" 200 11

POST

his is post

POST

LOADSPLITEXECUTETEST

URL
http://127.0.0.1:8000/hello/class_deco/

Enable POST

enctype
application/x-www-form-urlencoded

Body

3. 中间件（假）

太麻烦辣，主要是闭包搞不透彻，懒得看了。。。开摆。。。。

留个种子，以后要学再看（虽说以后再学是假的/doge）

中间件