在每个请求执行视图之前或者执行完成之后，需要做一些操作，为了避免每个视图都重写重复代码，Flask提供了四个通用函数完成一系列操作，即请求钩子

有些类似django的中间件

@before_first_request
在对应用程序实例的第一个请求之前注册要运行的函数，只会运行一次

@before_request
在每个请求之前注册一个要运行的函数，每一次请求都会执行一次

@after_request
在每个请求之后注册一个要运行的函数，每次请求完成后都会执行。
需要接受一个Response对象作为参数，并返回一个新的Response对象，或者返回接收的Response对象

@teardown_request
注册在每一个请求的末尾，不管是否有异常，每次请求的最后都会执行。

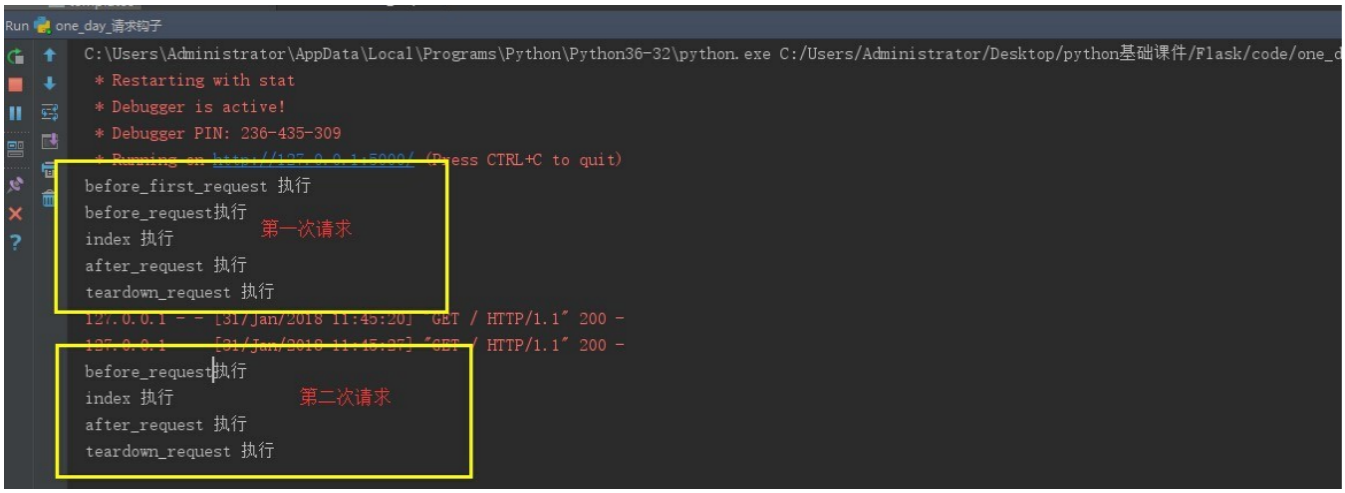
@context_processor
上下文处理器，返回的字典可以在全部的模板中使用

@template_filter('xxxxxx')
增加模板过滤器，可以在模板中使用该函数，后面的参数是名称，在模板中用到

@errorhandler(400)
发生一些异常时，比如404,500，或者抛出异常(Exception)之类的，就会自动调用该钩子函数
    1.发生请求错误时，框架会自动调用相应的钩子函数，并向钩子函数中传入error参数
    2.如果钩子函数没有定义error参数，就会报错
    3.可以使用abort函数来手动终止请求抛出异常，如果要是发生参数错误，可以abort(404)之类的

```
# -*- encoding: utf-8 -*-
# 导入Flask类
import os

from flask import Flask
```

```python
# Flask 接收一个参数 name ，
# 导入模块的目录， flask以这个目录为基础，寻找静态文件目录static和模板目录
templates
app = Flask(__name__)


# 只有在第一次请求的时候执行
@app.before_first_request
def first_request():
    print("before_first_request 执行")


# 每次请求之前执行
@app.before_request
def before_request_():
    print('before_request执行')


# 每次执行完成后没有未处理的异常抛出才会执行
@app.after_request
def after_request(response):
    print('after_request 执行')
    return response

# 即使视图函数有未处理的异常抛出都执行
@app.teardown_request
def teardown_request_(response):
    print('teardown_request 执行')
    return response


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


if __name__ == '__main__':

    # Flask 应用程序实例的方法run启动web服务器
    app.run(debug=True)
```


Run one_day_请求钩子
C:\Users\Administrator\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/Administrator/Desktop/python基础课件/Flask/code/one_d
* Restarting with stat
* Debugger is active!
* Debugger PIN: 236-435-309
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
before_first_request 执行
before_request执行
index 执行
第一次请求
after_request 执行
teardown_request 执行
127.0.0.1 - - [31/Jan/2018 11:45:20] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Jan/2018 11:45:27] "GET / HTTP/1.1" 200 -
before_request执行
index 执行
第二次请求
after_request 执行
teardown_request 执行