1 问题说明

- 导入视图放到最后的原因
  - 循环引用

```python
from flask import Blueprint

goods_bp = Blueprint('goods', __name__)

from . import views  # 此处
```

- Flask Debug模式的作用
  - 后端出现错误 会直接返回真实的错误信息给前端
  - 修改代码后 自动重启开发服务器
- 支持options请求不等价于实现了CORS跨域解决方案

2 处理请求

```
请求报文
GET /users/1?a=1 HTTP/1.1
Content-Type: application/json
...

body  -> file  form  json xml


def func(request, ...):
    request.
```

3 处理响应

```
HTTP/1.1 200 OK
Content-Type: application/json  # 描述body的数据格式的
..
body "{}"


```

返回模板 render_template

重定向

返回json

- return json.dumps()
- return jsonify
    - 转换成json格式字符串
    - 设置了响应头Content-Type: application/json

构造响应头和状态码

```
def func():
  return ret1, ret2, ret3
  #return (ret1, ret2, ret3)

ret = (ret1, ret2, ret3)
ret = func()
r1, r2, r3 = (ret1, ret2, ret3)
r1, r2, r3 = func()
```

cookie

- 设置 resp.set_cookie
- 读取 request.cookies.get()
- 删除 resp.delete_cookie

session

`from flask import session`

flask -> 浏览器session

异常处理

abort error_handler

请求钩子

启动中间件/中间层的作用

[middleware1 -> Class Middleware1

def pre_process

def after_process(response)

....

return resp

middleware2

middleware3]

请求的处理过程 pre_process -> view -> after_process

request 请求支持 处理流程

middleware1.pre_process() -> m2.pre_process() -> m3.pre_process()

-> view() -> m3.after_process() -> m2.after_process() -> m1.after_process() -> client

中间件处理 不区分具体是哪个视图， 对所有视图通通生效

上下文综合案例

分析

- 特定强制需求 -> 装饰器
- 所有视图的需求 -> 请求钩子

请求-> 请求钩子（尝试判断用户的身份 对于未登录用户不做处理 放行） 用g对象保存用户身份信息

g.user_id = 1232 g.user_id =None

-> 普通视图处理 g.user_id

-> 强制登录视图 -> 装饰器

上下文实现的原理 -> Threadlocal 线程局部变量

from flask import request

request -> 全局变量

/articles?channel_id=123 -> request.args.get('channel_id') -> 123 Thread id A

/articles?channel_id=124 -> request.args.get('channel_id') -> 124 Thread id B

request.args = {

'thread_a_id': 123,

'thread_b_id': 124

}

作业

完成 https://learngitbranching.js.org/ 练习