小白学Flask第八天| Flask上下文和请求钩子

原创JAP君 Python进击者
2019-09-10原文

本文内容：

1.Flask的上下文对象

2.请求钩子

Flask的上下文对象

在这篇文章之前，我们学习过request和session这两个小家伙，他们两的功能都非常的强大，我们今天讲上下文对象，也是和他们两有着很大的关系。

request：封装了HTTP请求的内容，针对的是http请求。举例：user = request.args.get('user')，获取的是get请求的参数。

session：用来记录请求会话中的信息，针对的是用户信息。举例：session['name'] = user.id，可以记录用户信息。还可以通过session.get('name')获取用户信息。

首先给大家看段代码：

```
from flask import Flask, request
```

```
app = Flask(__name__)


@app.route('/')
def hello_world(request): #   在这里将request对象作为参数传进来
    data = request.json
    return 'hello world'


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

大家仔细看这段代码会发现我们将request对象作为参数传进视图函数hello_world，如果我们当真要这样去使用request对象，视图函数一多，我们就会发现这样非常的不整洁而且还容易出错。

为了解决这个问题，**利用“上下文对象”将request对象作为全局变量，此时这个request对象就是在这个线程中的全局变量。但是如果这个对象是在A线程当中那么他就是A线程中的全局变量，在其他线程（B线程，C线程...）当中不是全局变量，这样就可以保证对象不会混淆。**

所以我们平时这样写就OK了：

```
from flask import Flask, request


app = Flask(__name__)


@app.route('/')
```

```
def hello_world():
    data = request.json
    return 'hello world'


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

session的道理也是类似的，这里就不多阐述。

除了request和session这类请求上下文对象(request context)，还有一类上下文对象，叫做**应用上下文对象(application context)**。

current_app和g都属于应用上下文对象。

current_app：表示当前运行程序文件的程序实例。
g：处理请求时，用于临时存储的对象，每次请求都会重设这个变量。

current_app在之前的文章中我们也简单介绍过。那么g是啥？

其实它就是一个存储容器，你想往里面存储什么样的数据都可以。

```
from flask import Flask, g


app = Flask(__name__)


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

```
    g.username = "JavaandPython君"
    g.pass = "123"
    return 'hello world'


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

请求钩子

大家可能以前没听说过这个概念，其实非常容易理解，大家都知道钩子是什么，钩子有什么用呢？ 钩住某个东西然后跟它连在一起。

在客户端和服务器交互的过程中，有些准备工作或扫尾工作需要处理，比如：在请求开始时，建立数据库连接；在请求结束时，指定数据的交互格式。为了让每个视图函数避免编写重复功能的代码，Flask提供了通用设施的功能，即请求钩子。

这里给出几个钩子的概念：
1.**before_first_request：**在第一次请求之前运行，只需执行一次，如链接数据库

2.**before_request：**在每一次请求都会执行,可以在这里做权限校验操作，比如说某用户是黑名单用户，黑名单用户登录系统将遭到拒绝访问，可以使用before_request进行权限校验。

3.**after_request**
：在请求之后运行，会接收一个参数，这个参数就是前面的请求处理完毕之后,返回的响应数据，如果需要对响应做额外处理,可以再这里进行。

4.**teardown_request**

：每一次请求之后都会调用，会接受一个参数，参数是服务器出现的错误信息

如何使用它们呢？其实也是非常简单

```
from flask import Flask


app = Flask(__name__)


@app.route('/')
def helloworld():
    return 'hello world'


# 在第一次请求之前运行.
# 例子: 比如连接数据库操作, 只需要执行一次
@app.before_first_request
def before_first_request():
    print('before_first_request')


# 在每一次请求都会执行
# 例子:
可以在这里做权限校验操作，比如说某用户是黑名单用户，黑名单用户登录系统将遭到拒绝访问，可以使用
# before_request进行权限校验
@app.before_request
```

```python
def before_request():
    print('before_request')


# 在请求之后运行
@app.after_request
def after_request(response):
    # response: 就是前面的请求处理完毕之后, 返回的响应数据
    # 如果需要对响应做额外处理,可以再这里进行
    # json.dumps 配置请求钩子
    # response.headers["Content-Type"] = "application/json"
    print('after_request')
    return response


# 每一次请求之后都会调用，会接受一个参数，参数是服务器出现的错误信息
@app.teardown_request
def teardown_request(error):
    # 数据库的扩展, 可以实现自动提交数据库
    print('teardown_request: error %s' % error)



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

关于钩子的知识点，我们只需要去记住这几个点就OK了。

但是这里给大家**延申一个知识**，从上面大家可能会发现我们这些钩子不能够去锁定某个视图函数，例如他不能确定我只有运行A视图函数才去执行钩子里的内容，我运行B视图函数他也会去执行，那么怎样才能够指定视图函数执行指定的钩子内容。

```
from flask import Flask, url_for, request


app = Flask(__name__)


@app.route('/hello')
def helloworld():
    return 'hello world'


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


# 在第一次请求之前运行.
# 例子: 比如连接数据库操作, 只需要执行一次
@app.before_first_request
def before_first_request():
     path = request.path
     if path == url_for("hello"):
        printf("如果是hello视图函数就执行这个")
     elif path == url_for("index"):
```

```
        printf("如果是index视图函数就执行这个")
    return ""


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

通过上面代码，相信你就已经懂怎么做了！

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

持续更新中...

精选留言

暂无...