**Flask 使用上下文让**特定的变量在一个线程中全局可访问，与此同时却不会干扰其他线程。

程序上下文：current\_app与g;请求上下文：session与request;，都为全局变量。

Flask提供**注册通用函数**的功能，注册的函数可在请求被分发到视图函数之前或之后调用；

Flask 视图响应默认的HTTP响应状态码为200，数字代码作为第二个参数返回值，可改变状态码，如:return '<h1>Bad Request</h1>',400;调用make\_response()函数创建响应对象；

重定向函数redirect('URL');abort函数：处理响应错误（抛出异常），如：abort(404)

# **Flask-Bootstrap:**

Flask-Bootstrap 把 [Bootstrap](http://getbootstrap.com/) 打包进一个 扩展，这个扩展主要由一个叫“bootstrap”的蓝本（blueprint）组成。它也可以创建链接从一个CDN上引用Bootstrap资源。

## **跨浏览器支持:**

跨浏览器支持（特别对Internet Explorer < 9的版本来说）通常是 必要的，但是Flask-Bootstrap没有附加这个功能。你可以下载 [html5shiv](https://raw.github.com/aFarkas/html5shiv/master/dist/html5shiv.min.js) 和 [Respond.js](https://raw.githubusercontent.com/scottjehl/Respond/master/dest/respond.min.js) ，把它们放在你程序的静态文件夹，然后像下面的例子那样包含它们**:**

{% import "bootstrap/fixes.html" as fixes %}

{% block head %}

{{super()}}

{{fixes.ie8()}}

{% endblock %}

**Html5shiv:**Html5标签结构

**git :https://github.com/aFarkas/html5shiv.git**

使用[Javascript](https://baike.baidu.com/item/Javascript)来使不支持HTML5的浏览器支持[HTML](https://baike.baidu.com/item/HTML)标签。针对IE浏览器比较好的解决方案是html5shiv。html5shiv主要解决HTML5提出的新的元素不被IE6-8识别，这些新元素不能作为父节点包裹子元素，并且不能应用CSS样式。让CSS 样式应用在未知元素上只需执行 document.createElement(elementName) 即可实现。html5shiv就是根据这个原理创建的。

# **Respond.JS:**

**Git:https://github.com/scottjehl/Respond.git**

Respond.js 是一个快速、轻量的 polyfill，用于为 IE6-8 以及其它不支持 [CSS3](https://www.baidu.com/s?wd=CSS3&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnAu-Phc1m1--uh7BrHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0knHcznjDLPWf3rHmsP1mzr0) Media Queries 的浏览器提供媒体查询的 min-width 和 [max-width](https://www.baidu.com/s?wd=max-width&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnAu-Phc1m1--uh7BrHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0knHcznjDLPWf3rHmsP1mzr0) 特性，实现[响应式网页设计](https://www.baidu.com/s?wd=响应式网页设计&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnAu-Phc1m1--uh7BrHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnW0knHcznjDLPWf3rHmsP1mzr0)（Responsive Web Design）。

程序可以把数据存储在用户会话中，在请求之间“”记住”数据，用户会话是一种私有存储，存在于每个连接到服务器的客户端中，它是请求上下文的变量，名为session,默认情况下保存在客户端cookie中，使用设置的SECRET\_KEY进行加密签名。

Python3与Python2中MYSQL连接也是有区别的

大概就是如下：

app.config**[**'SQLALCHEMY\_DATABASE\_URI'**]** = 'mysql://root:root@127.0.0.1/tushare?charset=utf8' #这是Python2的

但是在Python3进行了改变：

正确的连接应该是这样

app.config**[**'SQLALCHEMY\_DATABASE\_URI'**]** = 'mysql+pymysql://root:root@127.0.0.1/tushare?charset=utf8' #这是Python3的

如果不这样连接会怎么样，恭喜你会爆出一堆你不知道是什么鬼的错误

return \_\_import\_\_('MySQLdb')

ModuleNotFoundError: No module named 'MySQLdb'

网易邮件发送失败原因：

**link:**http://help.163.com/09/1224/17/5RAJ4LMH00753VB8.html

失败代码：

　•554 DT:SPM 发送的邮件内容包含了未被许可的信息，或被系统识别为垃圾邮件。请检查是否有用户发送病毒或者垃圾邮件；

## **Jinja2**

**模板语法**

{{变量/表达式}}

{%语法%}

{#注释#}

#开头，行语法表达式：app.jinja\_env.line\_statement\_prefix = '#'

\*\*\*\*

在使用pycharm创建flask项目时，如果没有用pycharm自带的flask模版方式创建项目(手动创建的项目)，则templates目录下的模版页中的Jinja标签不能自动识别，解决办法：在项目根目录下的.idea文件夹下的xxx.iml文件(xxx是项目名)中<component name="TemplatesService">下添加：

<option name="TEMPLATE\_CONFIGURATION" value="Jinja2" />

随后重新打开项目文件即可

\*\*\*

response对象

2.1 生成response对象

response = make\_response(render\_template(index.html))

2.2 方法

status：响应状态

headers：响应头，设置http字段

set\_coockie：设置一个cookie

**重定向**

redirect(location, code)

其中，location是链接路径，code取值有301和302，301——永久转移，302——临时转移；

# [从Flask-Script迁移到Flask-Cli](http://www.cnblogs.com/lynsyklate/p/7693169.html)

flask从0.11版本开始引入了click提供命令行支持，在此之前我们通常会引入Flask-Script来提供。当我们使用Flask-Script时，你将创建名为manage.py的驱动脚本。我们来看下flask-script与flask-cli之间的对比

#flask-script -> flask-Cli

./manage.py runserver -> flask run

./manage.py shell ->flask shell

如果你想为Flasky应用提供FLask Cli的支持，你可以在flasky.py中这样编写

import os

from app import create\_app

app = create\_app(os.getenv('FLASK\_CONFIG') or 'default')

我们在将会会完全迁移至新的Cli，不在使用manage.py来。

设置FLASK\_APP:

export FLASK\_APP=flask.py

export FLASK\_DEBUG=1

flask run

大部分应用可以使用下面的方式启动

if \_\_name\_\_=='\_\_main\_\_:

app.run()

当你不需要使用CLI支持时你确实可以以这种方式启动。但是当你确实需要新的cli时，你尽可以移除它。

## **使用Flask Shell 命令**

通常，你在shell中的测试或调试shell中添加model类、数据库实例和其他可能与之交互的对象。  
对于FLask-Script，Flasky应用有如下shell上下文定义

def make\_shell\_context():

return dict(app=app, db=db, User=User, Follow=Follow, Role=Role,

Permission=Permission, Post=Post, Comment=Comment)

manager.add\_command("shell", Shell(make\_context=make\_shell\_context))

正如你在上文看到的，make\_shell\_context()函数被add\_command引用来定义shell参数。Flask-Script在启动shell前直接调用这个函数，然后返回一个包含这些symbol的字典。Flask CLI提供了相同功能，但是确实以decorator(装饰器)来判断是不是shell上下文对象。为了和FLask-Script有相同的功能，我们扩展了flasky.py模块。

import os

from app import create\_app, db

from app.models import User, Follow, Role, Permission, Post, Comment

app = create\_app(os.getenv('FLASK\_CONFIG') or 'default')

@app.shell\_context\_processor

def make\_shell\_context():

return dict(app=app, db=db, User=User, Follow=Follow, Role=Role,

Permission=Permission, Post=Post, Comment=Comment)

正如你看见的，作用时相同的，但是你需要@app.shell\_context\_processor装饰器使Flask知道这些shell上下文对象

## **增加Flask拓展的命令**

如果你使用的扩展正未曾为FLask Cli做出适配，那么你得继续使用FLask-Script，Flask migrate提供了对于FLask Cli的更新，直接使用它最新的版本即可。  
对于Flask Migrate， ./manage.py db 直接变成了 flask db。在Flask-Script中我们Flask-Migrate在manage.py中被初始化，在FLaskCli中我们需要将Flask-Script版本中manage.py中的内容移入flasky.py中，只有一些微小的不同。

## **应用命令**

允许你把自己的常用任务(custom work)封装函数在命令行中执行。在CLI中你也只需要添加一个装饰器就能够完成迁移

原先的./manage.py test命令

@manager.command

def test(coverage=False):

"""Run the unit tests."""

# ...

在Flask-Cli中

import click

# ...

@app.cli.command()

@click.option('--coverage/--no-coverage', default=False, help='Enable code coverage')

def test(coverage):

"""Run the unit tests."""

# …

@app.cli.command()装饰器提供了接口给Click。因为在Flask-Script中命令函执行是在应用上下文中的，而Flask-Cli中应用上下文如果你不需要是可以禁用的。在此处，函数的实际代码不需要改变，[但注意coverage选项需要显式使用@click.option装饰器](mailto:但注意coverage选项需要显式使用@click.option装饰器)，而在flask-script中，该选项是自动导出的函数的参数列表。

# [python中base64编码与解码](http://www.cnblogs.com/zanjiahaoge666/p/7242642.html)

在一些项目中，接口的报文是通过base64加密传输的，所以在进行接口自动化时，需要对所传的参数进行base64编码，对拿到的响应报文进行解码；广泛应用于MIME协议，作为电子邮件的传输编码，生成的编码可逆，后一两位可能有“=”，生成的编码都是ascii字符。

**python2中进行Base64编码和解码:**

>>> import base64  
>>> s = '我是字符串'  
>>> a = base64.b64encode(s)  
>>> print a  
ztLKx9fWt/u0rg==  
>>> print base64.b64decode(a)  
我是字符串

python3不太一样：因为3.x中字符都为unicode编码，而b64encode函数的参数为byte类型，所以必须先转码。

import base64

encodestr = base64.b64encode('abcr34r344r'.encode('utf-8'))

print(encodestr)

打印结果为

b'YWJjcjM0cjM0NHI='

b 表示 byte的意思，我们只要再将byte转换回去就好了。。。源码如下

import base64

encodestr = base64.b64encode('abcr34r344r'.encode('utf-8'))

print(str(encodestr,'utf-8'))

打印结果为

YWJjcjM0cjM0NHI=

**flask**虚拟环境下安装**Anconda**自带的库文件，不使用**conda**命令方式：

在Anconda的安装目录下找到pkgs目录，本机为：/usr/local/anaconda3/pkgs/,找到xxx.tar.bz2文件，解压提取lib/python3.\*/site-packages下的文件到虚拟环境venv/lib64/python3.\*/site-packages/下即可

# **swagger在node.js下的使用**

在/home/260137/Swagger/下操作

## 1.swagger-editor

* 在线编辑：http://editor.swagger.io/
* 本地编辑：
  + 用git clone https://github.com/swagger-api/swagger-editor.git 将源码下载到本地 (在/home/260137/Swagger/swagger-editor)
  + cd swagger-editor
  + http-server -p 8000
  + 浏览器登录http://127.0.0.1:8000（与在线编辑界面一样)

**2.swagger-ui**

用git clone https://github.com/swagger-api/swagger-ui.git将源码下载到本地

cd swagger-ui/dist/ （可将dist目录下的文件拷贝到任意位置，执行下面两步）

将index.js里面引入json文件的url改成你的api的json文件的地址 (/static/swagger\_file/swagger.json)

http-server -p 8000

或执行：node index.js

浏览器登录http://127.0.0.1:8000 可看到在线的API文档