Flask入门

Python Flask 目录

本文主要借鉴 letiantian 的文章 http://www.letiantian.me/learn-flask/

一、简介

1、Flask 是一个轻量级的基于 Python 的 Web 框架，支持 Python 2 和 Python 3，简单易用，适合快速开发。封装功能不及Django完善，性能不及Tornado，但是Flask的第三方开源组件比丰富（http://flask.pocoo.org/extensions/），其 WSGI工具箱采用 Werkzeug ，模板引擎则使用 Jinja2 。Flask也被称为 “microframework” ，因为它使用简单的核心，用 extension 增加其他功能。Flask没有默认使用的数据库、窗体验证工具。

本文章中的代码使用 Python 3 运行，建议安装最新版本，本文使用的是 `Python 3.6.4`。

2、其他web框架

（1）Django：比较“重”的框架，同时也是最出名的Python框架。包含了web开发中常用的功能、组件的框架（ORM、Session、Form、Admin、分页、中间件、信号、缓存、ContenType....），Django是走大而全的方向，最出名的是其全自动化的管理后台：只需要使用起ORM，做简单的对象定义，它就能自动生成数据库结构、以及全功能的管理后台。

（2）Tornado：大特性就是异步非阻塞、原生支持WebSocket协议；

（3）Flask：如上

（4）Bottle：是一个简单高效的遵循WSGI的微型python Web框架。说微型，是因为它只有一个文件，除Python标准库外，它不依赖于任何第三方模块。

二、安装

```
$ sudo pip3 install Flask
```

进入python交互模式看下Flask的介绍和版本：

```
$ python3

>>> import flask
>>> print(flask.__doc__)

    flask
    ~~~~~

    A microframework based on Werkzeug.  It's extensively documented
    and follows best practice patterns.

    :copyright: © 2010 by the Pallets team.
    :license: BSD, see LICENSE for more details.

>>> print(flask.__version__)
1.0.2
```

三、初始化Flask

使用Flask写一个显示“Hello World!”的web程序，如何配置、调试Flask。

1、输出 Hello World

（1）按照以下命令建立Flask项目：HelloWorld，后面每一个示例会多次使用到

```
mkdir HelloWorld
mkdir HelloWorld/static
mkdir HelloWorld/templates
touch HelloWorld/server.py
```

（2）static和templates目录是默认配置，其中static用来存放静态资源，例如图片、js、css文件等。templates存放模板文件。

我们的网站逻辑基本在server.py文件中，当然，也可以给这个文件起其他的名字。

（3）在`server.py`中加入以下内容：

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

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

（4）运行server.py

```
$ python3 server.py
 * Running on http://127.0.0.1:5000/
```

（5）打开浏览器访问`http://127.0.0.1:5000/`，浏览页面上将出现`Hello World!`。终端里会显示下面的信息：

```
127.0.0.1 - - [16/May/2014 10:29:08] "GET / HTTP/1.1" 200 -
```

（6）变量app是一个Flask实例，通过下面的方式：

```
1 @app.route('/')
2 def hello_world():
3     return 'Hello World!'
```

当客户端访问`/`时，将响应`hello_world()`函数返回的内容。**注意，这不是返回`Hello World!`这么简单，`Hello World!`只是HTTP响应报文的实体部分，状态码等信息既可以由Flask自动处理，也可以通过编程来制定。**

2、修改Flask的配置

（1）程序名称

```
app = Flask(__name__)
```

上面的代码中，python内置变量`__name__`的值是字符串`__main__`。Flask类将这个参数作为程序名称。当然这个是可以自定义的，比如`app = Flask("my-app")`。

（2）静态资源、模板、参考文档

```
app = Flask("my-app", static_folder="path1", template_folder="path2")
```

更多参数请参考`__doc__`：

```
1 from flask import Flask
2 print(Flask.__doc__)
```

（3）调试模式

上面的server.py中以`app.run()`方式运行，这种方式下，如果服务器端出现错误是不会在客户端显示的。但是在开发环境中，显示错误信息是很有必要的，要显示错误信息，应该以下面的方式运行Flask：

```
app.run(debug=True)
```

将`debug`设置为`True`的另一个好处是，程序启动后，会自动检测源码是否发生变化，若有变化则自动重启程序。这可以帮我们省下很多时间。

（4）绑定IP和端口

默认情况下，Flask绑定IP为`127.0.0.1`，端口为`5000`。我们也可以通过下面的方式自定义：

```
app.run(host='0.0.0.0', port=80, debug=True)
```

`0.0.0.0`代表电脑所有的IP。`80`是HTTP网站服务的默认端口。

由于绑定了80端口，需要使用root权限运行`server.py（会报错）`。也就是：

```
$ sudo python3 server.py
```

四、获取URL参数 （GET请求）

URL参数是出现在url中的键值对，例如`http://127.0.0.1:5000/?disp=3`中的url参数是`{'disp':3}`

1、同上，创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、列出所有的url参数

（1）修改server.py 内容：

```
 1 from flask import Flask, request
 2
 3 app = Flask(__name__)
 4
 5
 6 @app.route('/')
 7 def hello_world():
 8     return request.args.__str__()
 9
10
11 if __name__ == '__main__':
12     app.run(port=5000, debug=True)
```

（2）运行看效果，在浏览器中访问`http://127.0.0.1:5000/?user=Flask&time&p=7&p=8`，将显示：

```
ImmutableMultiDict([('user', 'Flask'), ('time', ''), ('p', '7'), ('p', '8')])
```

较新的浏览器也支持直接在url中输入中文（最新的火狐浏览器内部会帮忙将中文转换成符合URL规范的数据），在浏览器中访问`http://127.0.0.1:5000/?info=这是爱，`，将显示：

```
ImmutableMultiDict([('info', '这是爱，')])
```

（3）获取浏览器传给Flask服务的数据。可以通过`request.full_path`和`request.path`来看一下：

```
 1 from flask import Flask, request
 2
 3 app = Flask(__name__)
 4
 5
 6 @app.route('/')
 7 def hello_world():
 8     print(request.path)
 9     print(request.full_path)
```

```
10     return request.args.__str__()
11
12
13 if __name__ == '__main__':
14     app.run(port=5000, debug=True)
```

浏览器访问`http://127.0.0.1:5000/?info=这是爱,`，运行`server.py`的终端会输出：

```
1 /
2 /?info=%E8%BF%99%E6%98%AF%E7%88%B1%EF%BC%8C
```

3、获取某个指定的参数

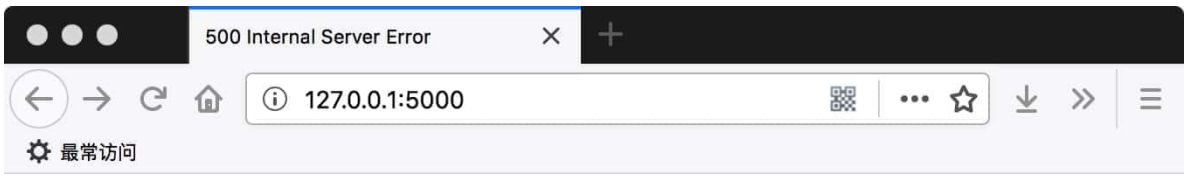
（1）获取指定参数。 例如，要获取键`info`对应的值，如下修改`server.py`：

```
 1 from flask import Flask, request
 2
 3 app = Flask(__name__)
 4
 5 @app.route('/')
 6 def hello_world():
 7     return request.args.get('info')
 8
 9 if __name__ == '__main__':
10     app.run(port=5000)
```

运行`server.py`，在浏览器中访问`http://127.0.0.1:5000/?info=hello`，浏览器将显示：

```
hello
```

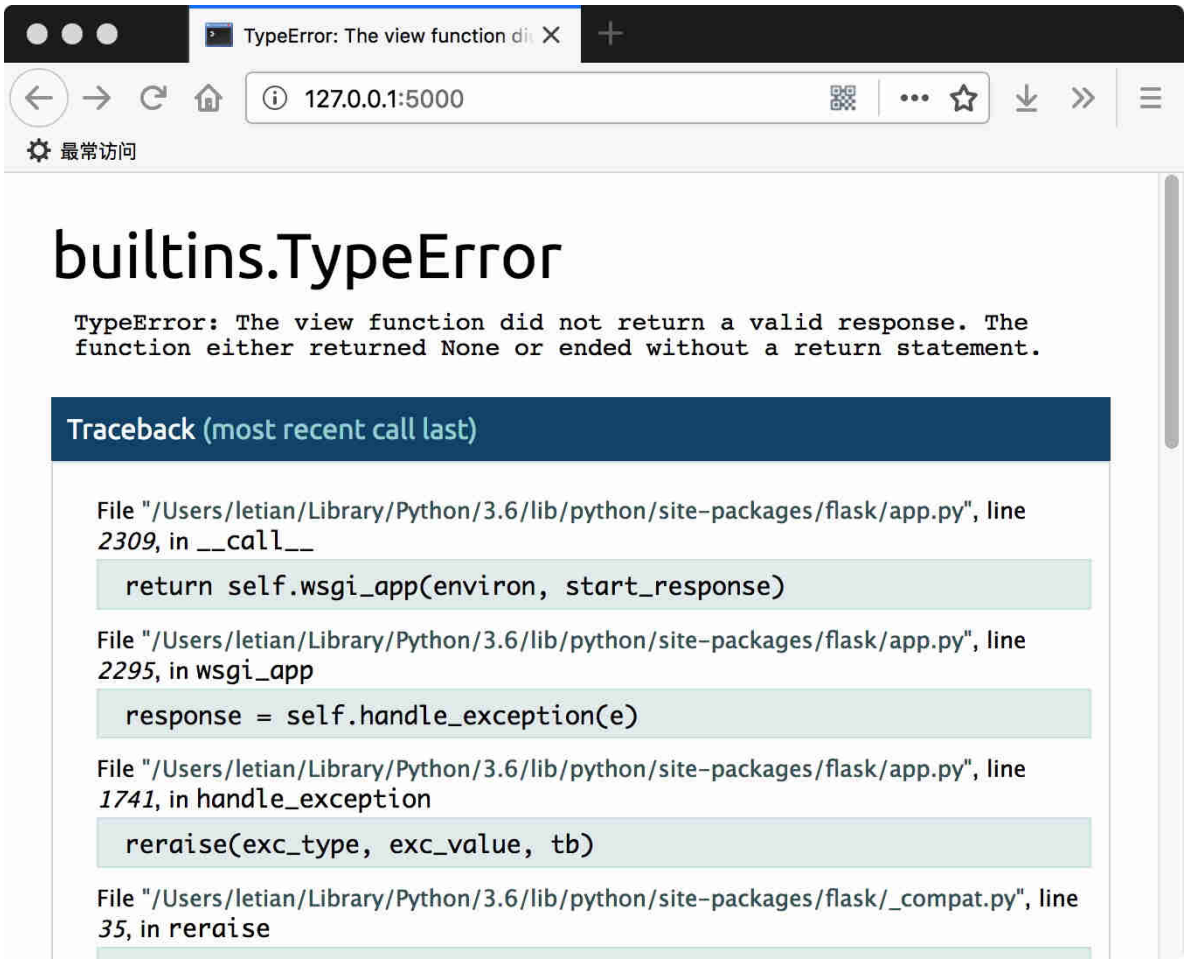
（2）异常和错误。当我们访问`http://127.0.0.1:5000/`时候却出现了500错误，浏览器显示：



Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

如果开启了Debug模式，会显示：



这是因为没有在URL参数中找到`info`。所以`request.args.get('info')`返回Python内置的None，而**Flask不允许返回None。**

解决方法很简单，先判断下它是不是None：

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def hello_world():
    r = request.args.get('info')
    if r==None:
        # do something
        return ''
    return r

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

另外一个方法是，设置默认值，也就是取不到数据时用这个值：

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def hello_world():
    r = request.args.get('info', 'hi')
    return r

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

4、如何处理多值

（1）出现多值。例如，`http://127.0.0.1:5000/?user=Flask&time&p=7&p=8`，仔细看下，`p`有两个值。

如果我们的代码是如下，则在浏览器中请求时，我们只会看到`7`：

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def hello_world():
    r = request.args.get('p')
    return r

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

（2）使用getlist 来获取多值，浏览器输入 `http://127.0.0.1:5000/?user=Flask&time&p=7&p=8`，我们会看到 `['7', '8']` ：

```
from flask import Flask, request

app = Flask(__name__)


@app.route('/')
def hello_world():
    r = request.args.getlist('p')  # 返回一个list
    return str(r)


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

五、获取POST方法传送的数据

作为一种HTTP请求方法，POST用于向指定的资源提交要被处理的数据。我们在某网站注册用户、写文章等时候，需要将数据传递到网站服务器中。并不适合将数据放到URL参数中，密码放到URL参数中容易被看到，文章数据又太多，浏览器不一定支持太长长度的URL。这时，一般使用POST方法。

1、同上，创建项目

```
mkdir HelloWorld
mkdir HelloWorld/static
mkdir HelloWorld/templates
touch HelloWorld/server.py
```

这里使用浏览器模拟工具：requests库

```
$ sudo pip3 install requests
```

2、查看POST数据内容

（1）编写server.py

以用户注册为例子，我们需要向服务器 `/register` 传送用户名 `name` 和密码 `password` 。如下编写 `HelloWorld/server.py`：

```
from flask import Flask, request

app = Flask(__name__)


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


@app.route('/register', methods=['POST'])
def register():
    print(request.headers)
    print(request.stream.read())
    return 'welcome'
```

```
16
17
18 if __name__ == '__main__':
19     app.run(port=5000, debug=True)
```

@app.route('/register', methods=['POST'])是指url /register只接受POST方法。可以根据需要修改methods`参数，例如如果想要让它同时支持GET和POST，这样写：

```
@app.route('/register', methods=['GET', 'POST'])
```

（2）浏览器模拟工具client.py内容如下：

```
1 import requests
2
3 user_info = {'name': 'letian', 'password': '123'}
4 r = requests.post("http://127.0.0.1:5000/register", data=user_info)
5
6 print(r.text)
```

（3）运行HelloWorld/server.py，然后运行client.py

client.py将输出：

```
welcome
```

而HelloWorld/server.py在终端中输出以下调试信息（通过print输出）：

```
1 Host: 127.0.0.1:5000
2 User-Agent: python-requests/2.19.1
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Connection: keep-alive
6 Content-Length: 24
7 Content-Type: application/x-www-form-urlencoded
8
9
10 b'name=letian&password=123'
```

前7行是client.py生成的HTTP请求头，由print(request.headers)输出。

请求体的数据，我们通过print(request.stream.read())输出，结果是：

```
b'name=letian&password=123'
```

3、解析POST数据

（1）修改server.py

上面，我们看到post的数据内容是：b'name=letian&password=123'，我们要把name、password提取出来，使用Flask内置的解析器request.form：

```
1 from flask import Flask, request
2
3 app = Flask(__name__)
```

```
 4
 5 @app.route('/')
 6 def hello_world():
 7     return 'hello world'
 8
 9
10 @app.route('/register', methods=['POST'])
11 def register():
12     print(request.headers)
13     # print(request.stream.read()) # 不要用，否则下面的form取不到数据
14     print(request.form)
15     print(request.form['name'])
16     print(request.form.get('name'))
17     print(request.form.getlist('name'))
18     print(request.form.get('nickname', default='little apple'))
19     return 'welcome'
20
21
22 if __name__ == '__main__':
23     app.run(port=5000, debug=True)
```

`request.form`会自动解析数据。`request.form['name']`和`request.form.get('name')`都可以获取`name`对应的值。对于`request.form.get()`可以为参数`default`指定值以作为默认值。

（2）客户端效果，执行`client.py`请求数据，服务器代码会在终端输出：

```
 1 Host: 127.0.0.1:5000
 2 User-Agent: python-requests/2.19.1
 3 Accept-Encoding: gzip, deflate
 4 Accept: */*
 5 Connection: keep-alive
 6 Content-Length: 24
 7 Content-Type: application/x-www-form-urlencoded
 8
 9
10 ImmutableMultiDict([('name', 'letian'), ('password', '123')])
11 letian
12 letian
13 ['letian']
14 little apple
```

（3）多个值的情况。如果`name`有多个值，可以使用`request.form.getlist('name')`，该方法将返回一个列表。将client.py改一下：

```
1 import requests
2
3 user_info = {'name': ['letian', 'letian2'], 'password': '123'}
4 r = requests.post("http://127.0.0.1:5000/register", data=user_info)
5
6 print(r.text)
```

此时运行`client.py`，`print(request.form.getlist('name'))`将输出：

```
[u'letian', u'letian2']
```

六、处理和响应JSON数据

使用 HTTP POST 方法传到网站服务器的数据格式可以有很多种，比如“获取POST方法传送的数据”讲到的`name=letian&password=123`这种用过`&`符号分割的key-value键值对格式。我们也可以用JSON格式、XML格式。相比XML的重量、规范繁琐，JSON显得非常小巧和易用。

1、同上，创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、处理JSON格式的请求数据。如果POST的数据是JSON格式，`request.json`会自动将json数据转换成Python类型（字典或者列表）。

（1）编写server.py

```
 1 from flask import Flask, request
 2
 3 app = Flask("my-app")
 4
 5
 6 @app.route('/')
 7 def hello_world():
 8     return 'Hello World!'
 9
10
11 @app.route('/add', methods=['POST'])
12 def add():
13     print(request.headers)
14     print(type(request.json))
15     print(request.json)
16     result = request.json['a'] + request.json['b']
17     return str(result)
18
19
20 if __name__ == '__main__':
21     app.run(host='127.0.0.1', port=5000, debug=True)
```

（2）编写client.py模拟浏览器请求：

```
1 import requests
2
3 json_data = {'a': 1, 'b': 2}
4
5 r = requests.post("http://127.0.0.1:5000/add", json=json_data)
6
7 print(r.text)
```

（3）执行效果，运行`server.py`，然后运行`client.py`

`client.py`会在终端输出：3

```
server.py` 会在终端输出：（**注意，请求头中`Content-Type`的值是**`**application/json）
**
```

```
 1 Host: 127.0.0.1:5000
 2 User-Agent: python-requests/2.19.1
 3 Accept-Encoding: gzip, deflate
 4 Accept: */*
 5 Connection: keep-alive
 6 Content-Length: 16
 7 Content-Type: application/json
 8
 9
10 <class 'dict'>
11 {'a': 1, 'b': 2}
```

3、响应JSON - 方案1： json.dumps()

响应JSON时，除了要把响应体改成JSON格式，响应头的`Content-Type`也要设置为`application/json`。

（1）编写server2.py

```
 1 from flask import Flask, request, Response
 2 import json
 3
 4 app = Flask("my-app")
 5
 6
 7 @app.route('/')
 8 def hello_world():
 9     return 'Hello World!'
10
11
12 @app.route('/add', methods=['POST'])
13 def add():
14     result = {'sum': request.json['a'] + request.json['b']}
15     return Response(json.dumps(result),  mimetype='application/json')
16
17
18 if __name__ == '__main__':
19     app.run(host='127.0.0.1', port=5000, debug=True)
```

（2）编写client2.py

```
1 import requests
2
3 json_data = {'a': 1, 'b': 2}
4
5 r = requests.post("http://127.0.0.1:5000/add", json=json_data)
6
7 print(r.headers)
8 print(r.text)
```

（3）执行效果

运行`client.py`，将显示：

```
1 {'Content-Type': 'application/json', 'Content-Length': '10', 'Server':
'Werkzeug/0.14.1 Python/3.6.4', 'Date': 'Sat, 07 Jul 2018 05:23:00 GMT'}
2 {"sum": 3}
```

上面第一段内容是服务器的响应头，第二段内容是响应体，也就是服务器返回的JSON格式数据。

（4）定制HTTP响应头，比如自定义`Server`，可以如下修改`add()`函数：

```
1 @app.route('/add', methods=['POST'])
2 def add():
3     result = {'sum': request.json['a'] + request.json['b']}
4     resp = Response(json.dumps(result),  mimetype='application/json')
5     resp.headers.add('Server', 'python flask')
6     return resp
```

`client2.py`运行后会输出：

```
1 {'Content-Type': 'application/json', 'Content-Length': '10', 'Server': 'python
flask', 'Date': 'Sat, 07 Jul 2018 05:26:40 GMT'}
2 {"sum": 3}
```

4、响应JSON - 方案2：使用 jsonify 工具函数

```
 1 from flask import Flask, request, jsonify
 2
 3 app = Flask("my-app")
 4
 5
 6 @app.route('/')
 7 def hello_world():
 8     return 'Hello World!'
 9
10
11 @app.route('/add', methods=['POST'])
12 def add():
13     result = {'sum': request.json['a'] + request.json['b']}
14     return jsonify(result)
15
16
17 if __name__ == '__main__':
18     app.run(host='127.0.0.1', port=5000, debug=True)
```

七、上传文件

一般也是POST方法

1、同上，创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、上传文件

以上传图片为例：假设将上传的图片只允许'png'、'jpg'、'jpeg'、'gif'这四种格式，通过url`/upload`使用POST上传，上传的图片存放在服务器端的`static/uploads`目录下。

（1）首先在项目`HelloWorld`中创建目录`static/uploads`：

```
mkdir HelloWorld/static/uploads
```

（2）使用werkzeug库，判断文件名是否安全，例如防止文件名是`../../../a.png`：

```
sudo pip3 install werkzeug
```

（3）编写server.py:

```
from flask import Flask, request

from werkzeug.utils import secure_filename
import os

app = Flask(__name__)

# 文件上传目录
app.config['UPLOAD_FOLDER'] = 'static/uploads/'
# 支持的文件格式
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif'}  # 集合类型


# 判断文件名是否是我们支持的格式
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in app.config['ALLOWED_EXTENSIONS']


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


@app.route('/upload', methods=['POST'])
def upload():
    upload_file = request.files['image']
    if upload_file and allowed_file(upload_file.filename):
        filename = secure_filename(upload_file.filename)
        # 将文件保存到 static/uploads 目录，文件名同上传时使用的文件名
        upload_file.save(os.path.join(app.root_path,
app.config['UPLOAD_FOLDER'], filename))
        return 'info is '+request.form.get('info', '')+'. success'
    else:
        return 'failed'


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

解析：

<1> app.config 中的config是字典的子类，可以用来设置自有的配置信息，也可以设置自己的配置信息。

<2> 函数 allowed_file(filename) 用来判断 filename 是否有后缀以及后缀是否在 app.config['ALLOWED_EXTENSIONS'] 中。

<3> 客户端上传的图片必须以 image 标识。

<4> upload_file 是上传文件对应的对象。

<5> app.root_path 获取 server.py 所在目录在文件系统中的绝对路径。

<6> upload_file.save(path) 用来将 upload_file 保存在服务器的文件系统中，参数最好是绝对路径，否则会报错（网上很多代码都是使用相对路径，但是笔者在使用相对路径时总是报错，说找不到路径）。

<7> 函数 os.path.join() 用来将使用合适的路径分隔符将路径组合起来。

（4）定制客户端client.py

```
1 import requests
2
3 file_data = {'image': open('Lenna.jpg', 'rb')}
4
5 user_info = {'info': 'Lenna'}
6
7 r = requests.post("http://127.0.0.1:5000/upload", data=user_info,
files=file_data)
8
9 print(r.text)
```

（5）执行效果。

<1> 运行 client.py，当前目录下的 Lenna.jpg 将上传到服务器。然后，我们可以在 static/uploads 中看到文件 Lenna.jpg。

<2> 要控制上产文件的大小，可以设置请求实体的大小，例如：(不过，在处理上传文件时候，需要使用 try:...except:...)

```
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024 #16MB
```

<3> 获取上传文件的内容可以：

```
file_content = request.files['image'].stream.read()
```

八、Restful URL

简单来说，Restful URL可以看做是对 URL 参数的替代

推荐阅读：阮一峰 理解RESTful架构 http://www.ruanyifeng.com/blog/2011/09/restful.html

1、同上，创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、写代码

（1）编写server.py:

```
 1 from flask import Flask
 2
 3 app = Flask(__name__)
 4
 5
 6 @app.route('/')
 7 def hello_world():
 8     return 'hello world'
 9
10
11 @app.route('/user/<username>')
12 def user(username):
13     print(username)
14     print(type(username))
15     return 'hello ' + username
16
17
18 @app.route('/user/<username>/friends')
19 def user_friends(username):
20     print(username)
21     print(type(username))
22     return 'hello ' + username + 'They are your friends.'
23
24
25 if __name__ == '__main__':
26     app.run(port=5000, debug=True)
```

（2）执行。运行`HelloWorld/server.py`。

<1> 使用浏览器访问`http://127.0.0.1:5000/user/letian`，HelloWorld/server.py将输出：

```
1 letian
2 <class 'str'>
```

<2> 访问`http://127.0.0.1:5000/user/letian/`，响应为404 Not Found

<3> 浏览器访问`http://127.0.0.1:5000/user/letian/friends`，可以看到：

```
Hello letian. They are your friends.
```

`HelloWorld/server.py`输出：

```
1 letian
2 <class 'str'>
```

3、转换类型

由上面的示例可以看出，使用 Restful URL 得到的变量默认为str对象。如果我们需要通过分页显示查询结果，那么需要在url中有数字来指定页数。按照上面方法，可以在获取str类型页数变量后，将其转换为int类型。不过，还有更方便的方法，**就是用flask内置的转换机制，即在route中指定该如何转换。**

（1）修改server： @app.route('/page/int:num')`会将num变量自动转换成int类型。

```
from flask import Flask

app = Flask(__name__)


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


@app.route('/page/<int:num>')
def page(num):
    print(num)
    print(type(num))
    return 'hello world'


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

（2）执行。在浏览器中访问`http://127.0.0.1:5000/page/1`，HelloWorld/server.py将输出如下内容：

```
1
<class 'int'>
```

如果访问的是`http://127.0.0.1:5000/page/asd`，我们会得到404响应。

（3）在官方资料中，说是有3个默认的转换器（看上去够用）：

```
int     accepts integers
float     like int but for floating point values
path     like the default but also accepts slashes
```

（4）一个特殊用法，修改server：

```
from flask import Flask

app = Flask(__name__)


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


@app.route('/page/<int:num1>-<int:num2>')
def page(num1, num2):
    print(num1)
```

```
14     print(num2)
15     return 'hello world'
16
17
18 if __name__ == '__main__':
19     app.run(port=5000, debug=True)
```

在浏览器中访问`http://127.0.0.1:5000/page/11-22`，`HelloWorld/server.py`会输出：

```
1 11
2 22
```

4、编写转换器（自定义）

自定义的转换器是一个继承`werkzeug.routing.BaseConverter`的类，修改`to_python`和`to_url`方法即可。`to_python`方法用于将url中的变量转换后供被@app.route`包装的函数使用，`to_url`方法用于`flask.url_for中的参数转换。

下面是一个示例，将`HelloWorld/server.py`修改如下：

```
 1 from flask import Flask, url_for
 2
 3 from werkzeug.routing import BaseConverter
 4
 5
 6 class MyIntConverter(BaseConverter):
 7
 8     def __init__(self, url_map):
 9         super(MyIntConverter, self).__init__(url_map)
10
11     def to_python(self, value):
12         return int(value)
13
14     def to_url(self, value):
15         return value * 2
16
17
18 app = Flask(__name__)
19 app.url_map.converters['my_int'] = MyIntConverter
20
21
22 @app.route('/')
23 def hello_world():
24     return 'hello world'
25
26
27 @app.route('/page/<my_int:num>')
28 def page(num):
29     print(num)
30     print(url_for('page', num=123))   # page 对应的是 page函数 ，num 对应对应
`/page/<my_int:num>`中的num，必须是str
31     return 'hello world'
32
33
34 if __name__ == '__main__':
35     app.run(port=5000, debug=True)
```

浏览器访问`http://127.0.0.1:5000/page/123`后，`HelloWorld/server.py`的输出信息是：

```
1 123
2 /page/123123
```

九、使用url_for生成链接

工具函数`url_for`可以让你以软编码的形式生成url，提供开发效率

1、同上，创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、写代码

（1）编写server.py:

```
 1 from flask import Flask, url_for
 2
 3 app = Flask(__name__)
 4
 5 @app.route('/')
 6 def hello_world():
 7     pass
 8
 9 @app.route('/user/<name>')
10 def user(name):
11     pass
12
13 @app.route('/page/<int:num>')
14 def page(num):
15     pass
16
17 @app.route('/test')
18 def test():
19     print(url_for('hello_world'))
20     print(url_for('user', name='letian'))
21     print(url_for('page', num=1, q='hadoop mapreduce 10%3'))
22     print(url_for('static', filename='uploads/01.jpg'))
23     return 'Hello'
24
25 if __name__ == '__main__':
26     app.run(debug=True)
```

（2）执行结果。运行`HelloWorld/server.py`，然后在浏览器中访问`http://127.0.0.1:5000/test`，`HelloWorld/server.py`将输出以下信息：

```
1 /
2 /user/letian
3 /page/1?q=hadoop+mapreduce+10%253
4 /static/uploads/01.jpg
```

十、使用redirect重定向网址

`redirect`函数用于重定向，实现机制很简单，就是向客户端（浏览器）发送一个重定向的HTTP报文，浏览器会去访问报文中指定的url。

1、同上，创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、编写代码。使用`redirect`时，给它一个字符串类型的参数就行了。

（1）编写server

```
 1 from flask import Flask, url_for, redirect
 2
 3 app = Flask(__name__)
 4
 5 @app.route('/')
 6 def hello_world():
 7     return 'hello world'
 8
 9 @app.route('/test1')
10 def test1():
11     print('this is test1')
12     return redirect(url_for('test2'))
13
14 @app.route('/test2')
15 def test2():
16     print('this is test2')
17     return 'this is test2'
18
19 if __name__ == '__main__':
20     app.run(debug=True)
```

（2）运行`HelloWorld/server.py`，在浏览器中访问`http://127.0.0.1:5000/test1`，浏览器的url会变成`http://127.0.0.1:5000/test2`，并显示：

```
this is test2
```

十一、使用Jinja2模板引擎

模板引擎负责MVC中的V（view，视图）这一部分。Flask默认使用Jinja2模板引擎。

Flask与模板相关的函数有：

- flask.render_template(template_name_or_list, **context)
  Renders a template from the template folder with the given context.
- flask.render_template_string(source, **context)
  Renders a template from the given template source string with the given context.
- flask.get_template_attribute(template_name, attribute)
  Loads a macro (or variable) a template exports. This can be used to invoke a macro from within Python code.

这其中常用的就是前两个函数。本例中使用了模板继承、if判断、for循环。

1、同上，创建项目

```
mkdir HelloWorld
mkdir HelloWorld/static
mkdir HelloWorld/templates
touch HelloWorld/server.py
```

2、创建并编辑 HelloWorld/templates/default.html

```
<html>
<head>
    <title>
        {% if page_title %}
            {{ page_title }}
        {% endif %}
    </title>
</head>

<body>
    {% block body %}{% endblock %}


```
可以看到，在``标签中使用了if判断，如果给模板传递了`page_title`变量，显示之，否则，不显示。
``标签中定义了一个名为`body`的block，用来被其他模板文件继承。
11.3 创建并编辑HelloWorld/templates/user_info.html
内容如下：
```
{% extends "default.html" %}

{% block body %}
    {% for key in user_info %}

        {{ key }}: {{ user_info[key] }}


    {% endfor %}
{% endblock %}
```

变量`user_info`应该是一个字典，for循环用来循环输出键值对。

3、编辑 server.py

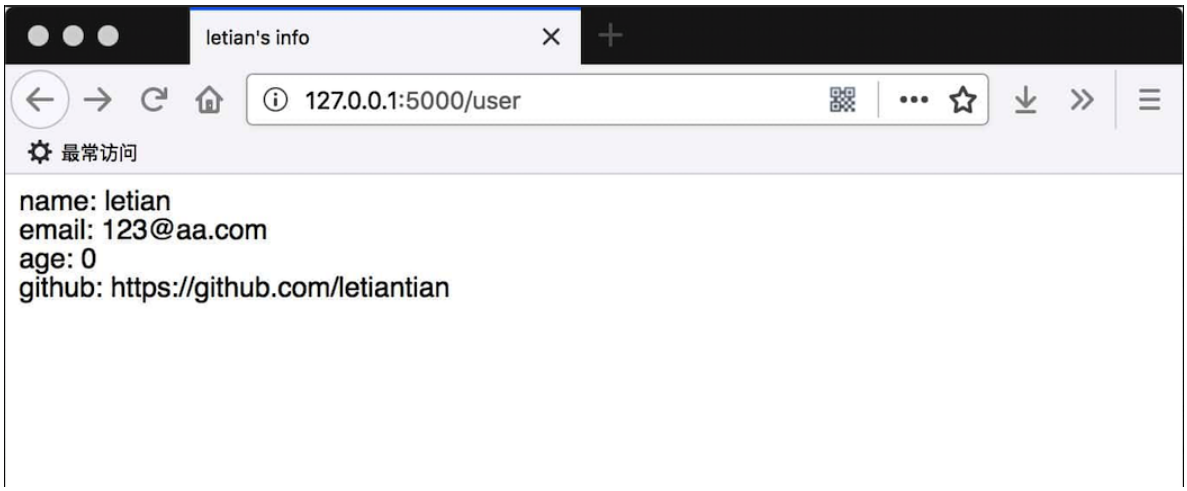
```
from flask import Flask, render_template

app = Flask(__name__)


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


@app.route('/user')
def user():
    user_info = {
```

```
14         'name': 'letian',
15         'email': '123@aa.com',
16         'age':0,
17         'github': 'https://github.com/letiantian'
18     }
19     return render_template('user_info.html', page_title='letian\'s info',
user_info=user_info)
20
21
22 if __name__ == '__main__':
23     app.run(port=5000, debug=True)
```

`render_template()`函数的第一个参数指定模板文件，后面的参数是要传递的数据。

4、运行与测试

运行HelloWorld/server.py，在浏览器中访问`http://127.0.0.1:5000/user`，效果图如下：



查看网页源码：

```
1 <html>
2 <head>
3     <title>
4             letian&amp;#39;s info
5     </title>
6 </head>
7 <body>
8         name: letian <br/>
9         email: 123@aa.com <br/>
10        age: 0 <br/>
11        github: https://github.com/letiantian <br/>
12 </body>
13 </html>
```

十二、自定义404等错误的响应

要处理HTTP错误，可以使用`flask.abort`函数。

1、示例1：简单入门

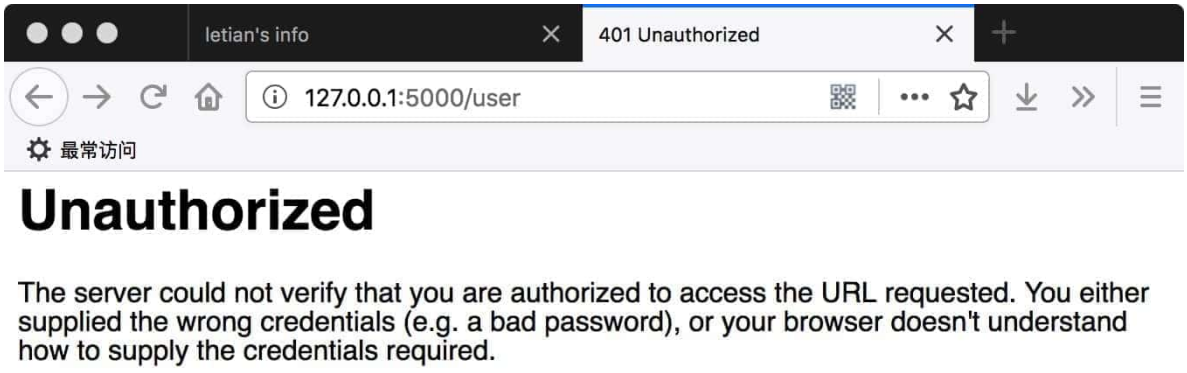
（1）创建项目

```
mkdir HelloWorld
mkdir HelloWorld/static
mkdir HelloWorld/templates
touch HelloWorld/server.py
```

（2）编辑 server.py

```
from flask import Flask, render_template_string, abort

app = Flask(__name__)


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


@app.route('/user')
def user():
    abort(401)  # Unauthorized 未授权
    print('Unauthorized, 请先登录')


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

（3）效果。运行`HelloWorld/server.py`，浏览器访问`http://127.0.0.1:5000/user`，效果如下：



要注意的是，`HelloWorld/server.py`中`abort(401)`后的`print`并没有执行。

2、示例2： 自定义错误页面

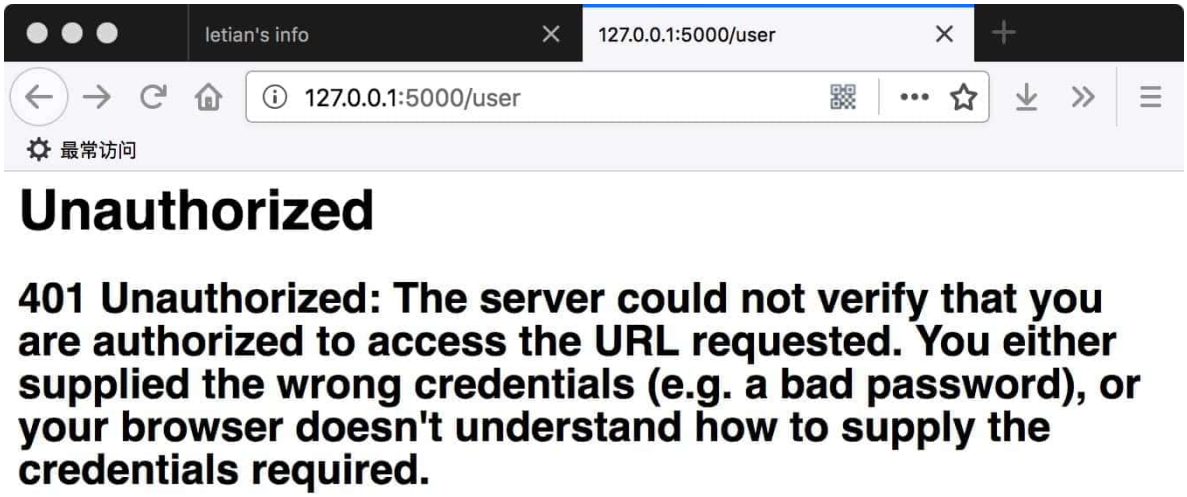
（1）修改server

```
from flask import Flask, render_template_string, abort

app = Flask(__name__)


```

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


@app.route('/user')
def user():
    abort(401)  # Unauthorized


@app.errorhandler(401)
def page_unauthorized(error):
    return render_template_string('<h1> Unauthorized </h1><h2>{{ error_info }}</h2>', error_info=error), 401


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

`page_unauthorized`函数返回的是一个元组，401 代表HTTP 响应状态码。如果省略401，则响应状态码会变成默认的 200。

（2）效果，运行`HelloWorld/server.py`，浏览器访问`http://127.0.0.1:5000/user`，效果如下：



十三、用户会话

session用来记录用户的登录状态，一般基于cookie实现

1、创建项目

```
mkdir HelloWorld
mkdir HelloWorld/static
mkdir HelloWorld/templates
touch HelloWorld/server.py
```

2、编辑 server.py

```
from flask import Flask, render_template_string, \
```

```
    session, request, redirect, url_for

app = Flask(__name__)

app.secret_key = 'F12Zr47j\3yX R~X@H!jLwf/T'


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


@app.route('/login')
def login():
    page = '''
    <form action="{{ url_for('do_login') }}" method="post">
        <p>name: <input type="text" name="user_name" /></p>
        <input type="submit" value="Submit" />
    </form>
    '''
    return render_template_string(page)


@app.route('/do_login', methods=['POST'])
def do_login():
    name = request.form.get('user_name')
    session['user_name'] = name
    return 'success'


@app.route('/show')
def show():
    return session['user_name']


@app.route('/logout')
def logout():
    session.pop('user_name', None)
    return redirect(url_for('login'))


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

代码含义与解析：

（1）`app.secret_key`用于给session加密。

（2）在`/login`中将向用户展示一个表单，要求输入一个名字，submit后将数据以post的方式传递给`/do_login`，`/do_login`将名字存放在session中。

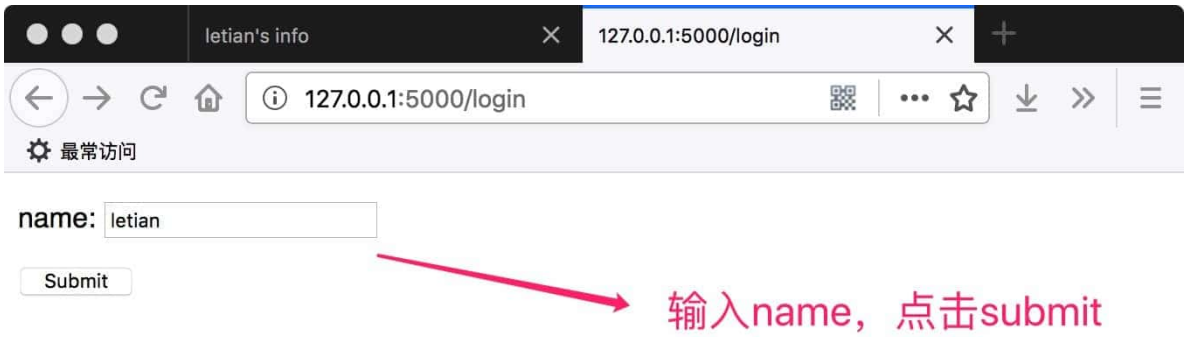
（3）如果用户成功登录，访问`/show`时会显示用户的名字。此时，打开firebug等调试工具，选择session面板，会看到有一个cookie的名称为`session`。

（4）`/logout`用于登出，通过将`session`中的`user_name`字段pop即可。Flask中的session基于字典类型实现，调用pop方法时会返回pop的键对应的值；如果要pop的键并不存在，那么返回值是`pop()`的第二个参数。

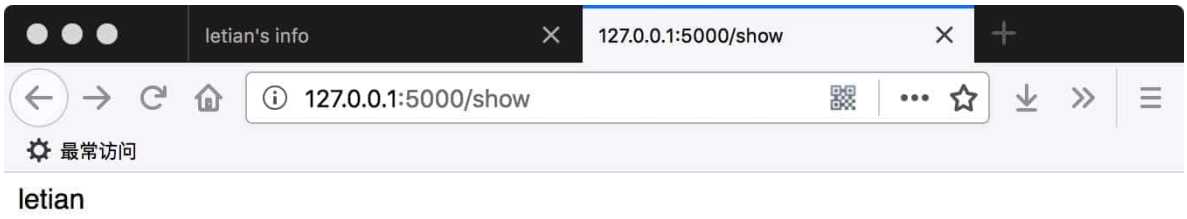
（5）使用`redirect()`重定向时，一定要在前面加上`return`。

3、效果

（1）进入`http://127.0.0.1:5000/login`，输入name，点击submit：



进入`http://127.0.0.1:5000/show`查看session中存储的name：



4、设置session的有效时间，将session的有效时间设置为5分钟。

```
1 from datetime import timedelta
2 from flask import session, app
3
4 session.permanent = True
5 app.permanent_session_lifetime = timedelta(minutes=5)
```

十四、使用Cookie

Cookie是存储在客户端的记录访问者状态的数据。 常用的用于记录用户登录状态的session大多是基于cookie实现的。cookie可以借助`flask.Response`来实现。

推荐阅读：具体原理 http://zh.wikipedia.org/wiki/Cookie

1、创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、代码

（1）修改server.py

```
 1 from flask import Flask, request, Response, make_response
 2 import time
 3
 4 app = Flask(__name__)
 5
 6
 7 @app.route('/')
 8 def hello_world():
 9     return 'hello world'
10
11
12 @app.route('/add')
13 def login():
14     res = Response('add cookies')
15     res.set_cookie(key='name', value='letian', expires=time.time()+6*60)
16     return res
17
18
19 @app.route('/show')
20 def show():
21     return request.cookies.__str__()
22
23
24 @app.route('/del')
25 def del_cookie():
26     res = Response('delete cookies')
27     res.set_cookie('name', '', expires=0)
28     return res
29
30
31 if __name__ == '__main__':
32     app.run(port=5000, debug=True)
```

由上可以看到，可以使用`Response.set_cookie`添加和删除cookie。`expires`参数用来设置cookie有效时间，它的值可以是`datetime`对象或者unix时间戳，笔者使用的是unix时间戳。

```
res.set_cookie(key='name', value='letian', expires=time.time()+6*60)
```

上面的expire参数的值表示cookie在从现在开始的6分钟内都是有效的。

要删除cookie，将expire参数的值设为0即可：

```
res.set_cookie('name', '', expires=0)
```

`set_cookie()`函数的原型如下：

```
set_cookie(key, value='', max_age=None, expires=None, path='/', domain=None,
secure=None, httponly=False)

Sets a cookie. The parameters are the same as in the cookie Morsel object in the
Python standard library but it accepts unicode data, too.
Parameters:

     key – the key (name) of the cookie to be set.
     value – the value of the cookie.
     max_age – should be a number of seconds, or None (default) if the cookie
should last only as long as the client's browser session.

     expires – should be a datetime object or UNIX timestamp.

     domain – if you want to set a cross-domain cookie. For example,
domain=".example.com" will set a cookie that is readable by the domain
              www.example.com, foo.example.com etc. Otherwise, a cookie will
only be readable by the domain that set it.

     path – limits the cookie to a given path, per default it will span the
whole domain.
```

（2）运行与测试

<1> 运行server.py,使用浏览器打开`http://127.0.0.1:5000/add`，浏览器界面会显示:

```
add cookies
```

<2> 查看cookie，如果使用firefox浏览器，可以用firebug插件查看。打开firebug，选择`Cookies`选项，刷新页面，可以看到名为`name`的cookie，其值为`letian`。

在"网络"选项中，可以查看响应头中类似下面内容的设置cookie的HTTP「指令」：

```
Set-Cookie: name=letian; Expires=Sun, 29-Jun-2014 05:16:27 GMT; Path=/
```

<3> 在cookie有效期间，使用浏览器访问`http://127.0.0.1:5000/show`，可以看到:

```
{'name': 'letian'}
```

十五、闪存系统 flashing system

Flask的闪存系统（flashing system）用于向用户提供反馈信息，这些反馈信息一般是对用户上一次操作的反馈。反馈信息是存储在服务器端的，当服务器向客户端返回反馈信息后，这些反馈信息会被服务器端删除。

1、创建项目

```
1 mkdir HelloWorld
2 mkdir HelloWorld/static
3 mkdir HelloWorld/templates
4 touch HelloWorld/server.py
```

2、编写server.py

```
 1 from flask import Flask, flash, get_flashed_messages
 2 import time
 3
 4 app = Flask(__name__)
 5 app.secret_key = 'some_secret'
 6
 7
 8 @app.route('/')
 9 def index():
10     return 'hi'
11
12
13 @app.route('/gen')
14 def gen():
15     info = 'access at '+ time.time().__str__()
16     flash(info)
17     return info
18
19
20 @app.route('/show1')
21 def show1():
22     return get_flashed_messages().__str__()
23
24
25 @app.route('/show2')
26 def show2():
27     return get_flashed_messages().__str__()
28
29
30 if __name__ == "__main__":
31     app.run(port=5000, debug=True)
```

3、效果

（1）打开浏览器，访问`http://127.0.0.1:5000/gen`，浏览器界面显示（注意，时间戳是动态生成的，每次都会不一样，除非并行访问）：

```
access at 1404020982.83
```

（2）查看浏览器的cookie，可以看到`session`，其对应的内容是：

```
.eJyrVopPy0kszkgtVrKKrlZSKIFQSUpWSknhYVXJRm55UYG2tkq1OlDRyHC_rKgIvypPdzcDTxdXA1-XwHLfLEdTfxfPUn8XX6DKWCAEAJKBGq8.BpE6dg.F1VURZa7VqU9bvbC4XIBO9-3Y4Y
```

（3）再一次访问`http://127.0.0.1:5000/gen`，浏览器界面显示：

```
access at 1404021130.32
```

cookie中`session`发生了变化，新的内容是：

```
.eJyrVopPy0kszkgtVrKKrlZSKIFQSUpWSknhYVXJRm55UYG2tkq1OlDRyHC_rKgIvypPdzcDTxdXA1-XwHLfLEdTfxfPUn8XX6DKWLBaMg1yrfCtciz1rfIEGxRbCwAhGjC5.BpE7Cg.Cb_B_k2otqczhknGnpNjQ5u4dqw
```

（4）然后使用浏览器访问`http://127.0.0.1:5000/show1`，浏览器界面显示：

```
['access at 1404020982.83', 'access at 1404021130.32']
```

这个列表中的内容也就是上面的两次访问`http://127.0.0.1:5000/gen`得到的内容。此时，cookie中已经没有`session`了。

如果使用浏览器访问`http://127.0.0.1:5000/show1`或者`http://127.0.0.1:5000/show2`，只会得到：`[]`

4、高级用法

flash系统也支持对flash的内容进行分类。修改`HelloWorld/server.py`内容：

```
from flask import Flask, flash, get_flashed_messages
import time

app = Flask(__name__)
app.secret_key = 'some_secret'


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


@app.route('/gen')
def gen():
    info = 'access at '+ time.time().__str__()
    flash('show1 '+info, category='show1')
    flash('show2 '+info, category='show2')
    return info


@app.route('/show1')
def show1():
    return get_flashed_messages(category_filter='show1').__str__()

@app.route('/show2')
def show2():
    return get_flashed_messages(category_filter='show2').__str__()


if __name__ == "__main__":
    app.run(port=5000, debug=True)
```

某一时刻，浏览器访问`http://127.0.0.1:5000/gen`，浏览器界面显示：

```
access at 1404022326.39
```

不过，由上面的代码可以知道，此时生成了两个flash信息，但分类(category)不同。

使用浏览器访问`http://127.0.0.1:5000/show1`，得到如下内容：

```
['1 access at 1404022326.39']
```

而继续访问`http://127.0.0.1:5000/show2`，得到的内容为空：`[]`

5、在模板文件中获取flash的内容

在Flask中，`get_flashed_messages()`默认已经集成到`Jinja2`模板引擎中，易用性很强。下面是来自官方的一个示例：

```
{% with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}
    <ul class=flashes>
    {% for category, message in messages %}
      <li class="{{ category }}">{{ message }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
```