**Python Flask Web 框架入门**

2018年09月03日 20:31:23 [MHyourh](https://me.csdn.net/sinat_38682860) 阅读数 15998

Flask是一个轻量级的基于Python的web框架。

本文适合有一定HTML、Python、网络基础的同学阅读。

**1. 简介**

这份文档中的代码使用 Python 3 运行。  
是的，所以读者需要自己在电脑上安装Python 3 和 pip3。建议安装最新版本，我使用的是Python 3.6.4。  
安装方法，可以自行谷歌或者百度。  
建议在 linux 下实践本教程中命令行操作、执行代码。

**2. 安装**

通过pip3安装Flask即可：

$ sudo pip3 install Flask

进入python交互模式看下Flask的介绍和版本：

1. $ python3
3. >>> import flask
4. >>> print(flask.\_\_doc\_\_)
6. flask
7. ~~~~~
9. A microframework based on Werkzeug. It's extensively documented
10. and follows best practice patterns.
12. :copyright: © 2010 by the Pallets team.
13. :license: BSD, see LICENSE for more details.
15. >>> print(flask.\_\_version\_\_)
16. 1.0.2

**3. 从 Hello World 开始**

本节主要内容：使用Flask写一个显示”Hello World!”的web程序，如何配置、调试Flask。

**3.1 Hello World**

按照以下命令建立Flask项目HelloWorld:

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

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。

在server.py中加入以下内容：

1. from flask import Flask
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. return 'Hello World!'
9. if \_\_name\_\_ == '\_\_main\_\_':
10. app.run()

运行server.py：

1. $ python3 server.py
2. \* Running on http:*//127.0.0.1:5000/*

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 -

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。

**3.2 修改Flask的配置**

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")。

Flask默认使用static目录存放静态资源，templates目录存放模板，这是可以通过设置参数更改的：

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\_\_)

**3.3 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。

**3.4 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。什么是默认？比如，我们访问网站http://www.example.com，其实是访问的http://www.example.com:80，只不过:80可以省略不写。

由于绑定了80端口，需要使用root权限运行server.py。也就是：

$ sudo python3 server.py

**3.5 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-001>

**4. 获取 URL 参数**

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}。

**4.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**4.2 列出所有的url参数**

在server.py中添加以下内容：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return request.args.\_\_str\_\_()

11. if \_\_name\_\_ == '\_\_main\_\_':
12. app.run(port=5000, debug=True)

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，')])

浏览器传给我们的Flask服务的数据长什么样子呢？可以通过request.full\_path和request.path来看一下：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. print(request.path)
9. print(request.full\_path)
10. return request.args.\_\_str\_\_()

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

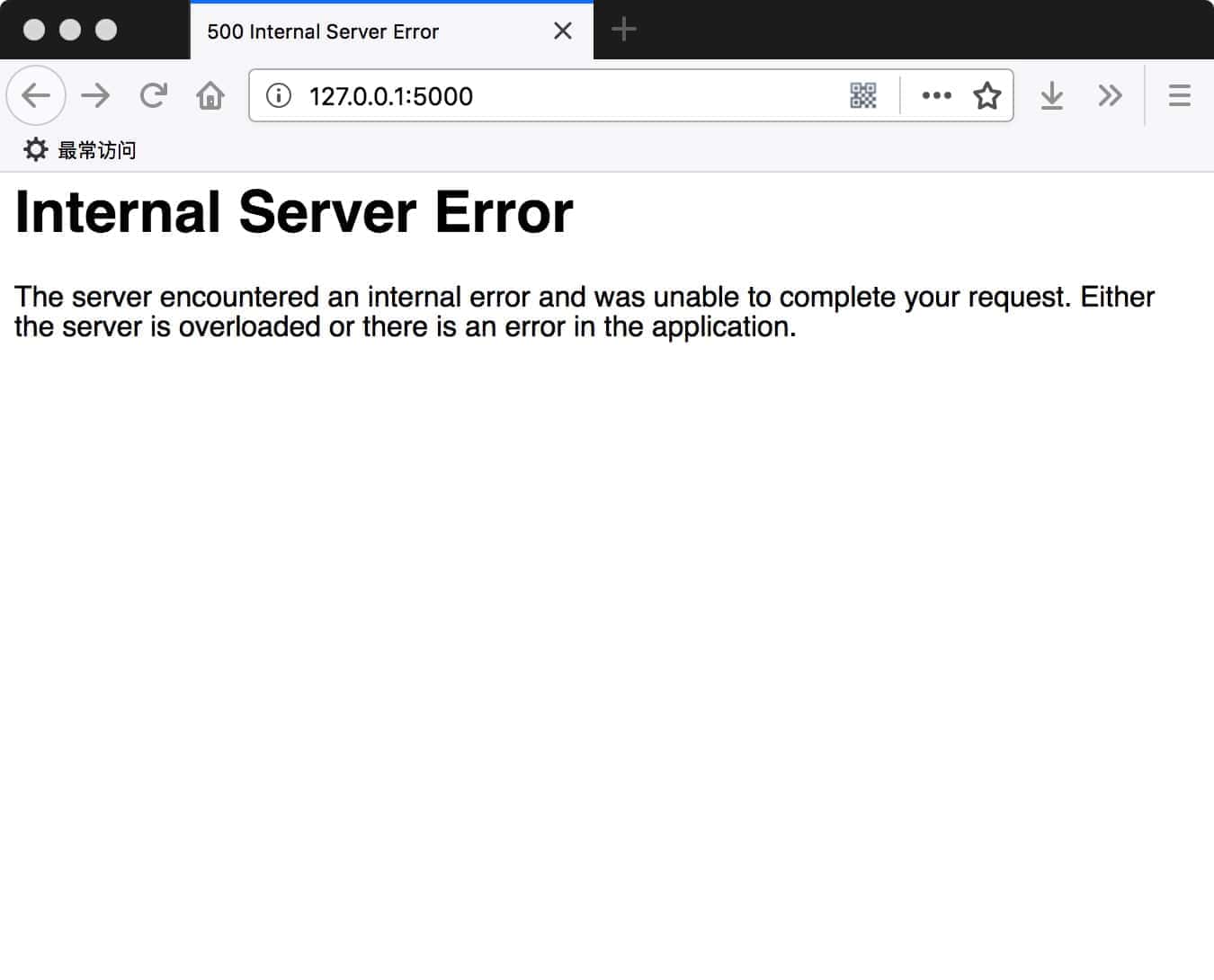
**4.3 获取某个指定的参数**

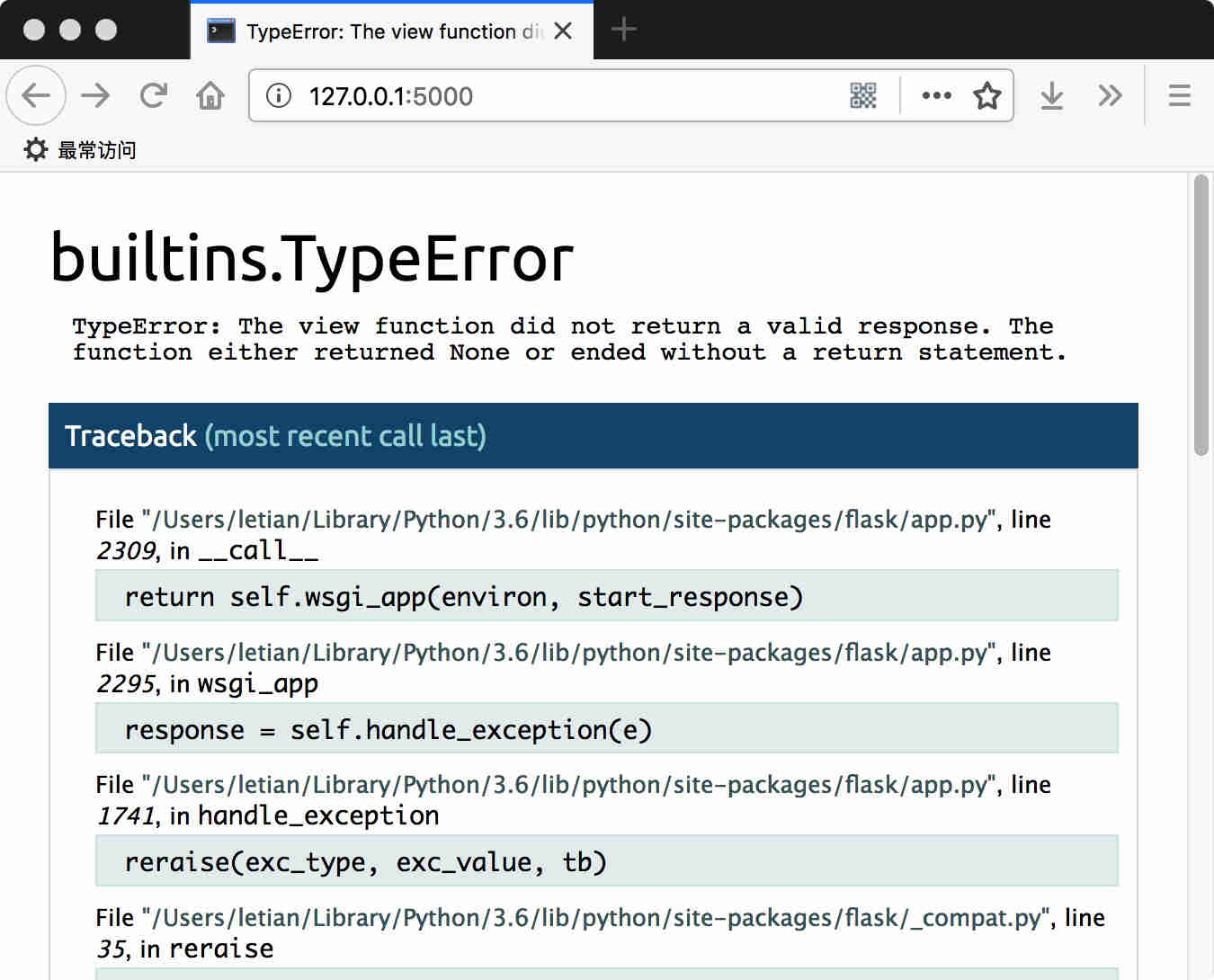
例如，要获取键info对应的值，如下修改server.py：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. return request.args.get('info')
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

不过，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：  
[](https://www.letiantian.me/content/images/learn-flask/flask-001.jpg)

如果开启了Debug模式，会显示：  
[](https://www.letiantian.me/content/images/learn-flask/flask-002.jpg)

为什么为这样？

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，我们先判断下它是不是None：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. r = request.args.get('info')
8. if r==None:
9. *# do something*
10. return ''
11. return r
13. if \_\_name\_\_ == '\_\_main\_\_':
14. 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：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. r = request.args.get('info', 'hi')
8. return r
10. if \_\_name\_\_ == '\_\_main\_\_':
11. app.run(port=5000, debug=True)

函数request.args.get的第二个参数用来设置默认值。此时在浏览器访问http://127.0.0.1:5000/，将显示：

hi

**4.4 如何处理多值**

还记得上面有一次请求是这样的吗？ 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：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. r = request.args.get('p')
8. return r
10. if \_\_name\_\_ == '\_\_main\_\_':
11. app.run(port=5000, debug=True)

在浏览器中请求时，我们只会看到7。如果我们需要把p的所有值都获取到，该怎么办？

不用get，用getlist：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. r = request.args.getlist('p') *# 返回一个list*
9. return str(r)

12. if \_\_name\_\_ == '\_\_main\_\_':
13. app.run(port=5000, debug=True)

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']。

**4.5 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-002>

**5. 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。

本文使用python的requests库模拟浏览器。

安装方法：

$ sudo pip3 install requests

**5.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**5.2 查看POST数据内容**

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。

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

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

18. if \_\_name\_\_ == '\_\_main\_\_':
19. app.run(port=5000, debug=True)

[`@app.route](mailto:%60@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'])

浏览器模拟工具client.py内容如下：

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

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

10. b'name=letian&password=123'

前6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'

**5.3 解析POST数据**

上面，我们看到post的数据内容是：

b'name=letian&password=123'

我们要想办法把我们要的name、password提取出来，怎么做呢？自己写？不用，Flask已经内置了解析器request.form。

我们将服务代码改成：

1. from flask import Flask, request
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. return 'hello world'

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'

22. if \_\_name\_\_ == '\_\_main\_\_':
23. app.run(port=5000, debug=True)

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

10. ImmutableMultiDict([('name', 'letian'), ('password', '123')])
11. letian
12. letian
13. ['letian']
14. little apple

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。所以：

print(request.form.get('nickname', default='little apple'))

输出的是默认值

little apple

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。我们将client.py改一下：

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

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

[u'letian', u'letian2']

**5.4 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-003>

**6. 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「5. 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。

**6.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**6.2 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）。

编写server.py:

1. from flask import Flask, request
3. app = Flask("my-app")

6. @app.route('/')
7. def hello\_world():
8. return 'Hello World!'

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)

20. if \_\_name\_\_ == '\_\_main\_\_':
21. app.run(host='127.0.0.1', port=5000, debug=True)

编写client.py模拟浏览器请求：

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

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：

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

10. <class 'dict'>
11. {'a': 1, 'b': 2}

注意，请求头中Content-Type的值是application/json。

**6.3 响应JSON-方案1**

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。

编写server2.py：

1. from flask import Flask, request, Response
2. import json
4. app = Flask("my-app")

7. @app.route('/')
8. def hello\_world():
9. return 'Hello World!'

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')

18. if \_\_name\_\_ == '\_\_main\_\_':
19. app.run(host='127.0.0.1', port=5000, debug=True)

修改后运行。

编写client2.py：

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

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。

另外，如果需要服务器的HTTP响应头具有更好的可定制性，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}

**6.4 响应JSON-方案2**

使用 jsonify 工具函数即可。

1. from flask import Flask, request, jsonify
3. app = Flask("my-app")

6. @app.route('/')
7. def hello\_world():
8. return 'Hello World!'

11. @app.route('/add', methods=['POST'])
12. def add():
13. result = {'sum': request.json['a'] + request.json['b']}
14. return jsonify(result)

17. if \_\_name\_\_ == '\_\_main\_\_':
18. app.run(host='127.0.0.1', port=5000, debug=True)

**6.5 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-004>

**7. 上传文件**

上传文件，一般也是用POST方法。

**7.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**7.2 上传文件**

这一部分的代码参考自[How to upload a file to the server in Flask](http://runnable.com/UiPcaBXaxGNYAAAL/how-to-upload-a-file-to-the-server-in-flask-for-python)。

我们以上传图片为例：  
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。

首先在项目HelloWorld中创建目录static/uploads：

mkdir HelloWorld/static/uploads

werkzeug库可以判断文件名是否安全，例如防止文件名是../../../a.png，安装这个库：

$ sudo pip3 install werkzeug

server.py代码：

1. from flask import Flask, request
3. from werkzeug.utils import secure\_filename
4. import os
6. app = Flask(\_\_name\_\_)
8. *# 文件上传目录*
9. app.config['UPLOAD\_FOLDER'] = 'static/uploads/'
10. *# 支持的文件格式*
11. app.config['ALLOWED\_EXTENSIONS'] = {'png', 'jpg', 'jpeg', 'gif'} *# 集合类型*

14. *# 判断文件名是否是我们支持的格式*
15. def allowed\_file(filename):
16. return '.' in filename and \
17. filename.rsplit('.', 1)[1] in app.config['ALLOWED\_EXTENSIONS']

20. @app.route('/')
21. def hello\_world():
22. return 'hello world'

25. @app.route('/upload', methods=['POST'])
26. def upload():
27. upload\_file = request.files['image']
28. if upload\_file and allowed\_file(upload\_file.filename):
29. filename = secure\_filename(upload\_file.filename)
30. *# 将文件保存到 static/uploads 目录，文件名同上传时使用的文件名*
31. upload\_file.save(os.path.join(app.root\_path, app.config['UPLOAD\_FOLDER'], filename))
32. return 'info is '+request.form.get('info', '')+'. success'
33. else:
34. return 'failed'

37. if \_\_name\_\_ == '\_\_main\_\_':
38. app.run(port=5000, debug=True)

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。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。

客户端上传的图片必须以image01标识。upload\_file是上传文件对应的对象。app.root\_path获取server.py所在目录在文件系统中的绝对路径。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）。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。

好了，定制客户端client.py：

1. import requests
3. file\_data = {'image': open('Lenna.jpg', 'rb')}
5. user\_info = {'info': 'Lenna'}
7. r = requests.post("http://127.0.0.1:5000/upload", data=user\_info, files=file\_data)
9. print(r.text)

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。

要控制上产文件的大小，可以设置请求实体的大小，例如：

app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024 *#16MB*

不过，在处理上传文件时候，需要使用try:...except:...。

如果要获取上传文件的内容可以：

file\_content = request.files['image'].stream.read()

**7.3 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-005>

**8. Restful URL**

简单来说，Restful URL可以看做是对 URL 参数的替代。

**8.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**8.2 编写代码**

编辑server.py：

1. from flask import Flask
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

11. @app.route('/user/<username>')
12. def user(username):
13. print(username)
14. print(type(username))
15. return 'hello ' + username

18. @app.route('/user/<username>/friends')
19. def user\_friends(username):
20. print(username)
21. print(type(username))
22. return 'hello ' + username

25. if \_\_name\_\_ == '\_\_main\_\_':
26. app.run(port=5000, debug=True)

运行HelloWorld/server.py。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'>

而访问http://127.0.0.1:5000/user/letian/，响应为404 Not Found。

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'>

**8.3 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。

新的服务器代码：

1. from flask import Flask
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

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

18. if \_\_name\_\_ == '\_\_main\_\_':
19. app.run(port=5000, debug=True)

[`@app.route](mailto:%60@app.route)(‘/page/int:num‘)`会将num变量自动转换成int类型。

运行上面的程序，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. 1
2. <class 'int'>

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

在官方资料中，说是有3个默认的转换器：

1. int accepts integers
2. float like int but for floating point values
3. path like the default but also accepts slashes

看起来够用了。

**8.4 一个有趣的用法**

如下编写服务器代码：

1. from flask import Flask
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

11. @app.route('/page/<int:num1>-<int:num2>')
12. def page(num1, num2):
13. print(num1)
14. print(num2)
15. return 'hello world'

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

**8.5 编写转换器**

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](mailto:%60@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
3. from werkzeug.routing import BaseConverter

6. class MyIntConverter(BaseConverter):
8. def \_\_init\_\_(self, url\_map):
9. super(MyIntConverter, self).\_\_init\_\_(url\_map)
11. def to\_python(self, value):
12. return int(value)
14. def to\_url(self, value):
15. return value \* 2

18. app = Flask(\_\_name\_\_)
19. app.url\_map.converters['my\_int'] = MyIntConverter

22. @app.route('/')
23. def hello\_world():
24. return 'hello world'

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'

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

**8.6 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-006>

**8.7 值得读**

[理解RESTful架构](http://www.ruanyifeng.com/blog/2011/09/restful.html)。

**9. 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。

**9.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**9.2 编写代码**

编辑HelloWorld/server.py：

1. from flask import Flask, url\_for
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. pass
9. @app.route('/user/<name>')
10. def user(name):
11. pass
13. @app.route('/page/<int:num>')
14. def page(num):
15. pass
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'
25. if \_\_name\_\_ == '\_\_main\_\_':
26. app.run(debug=True)

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

**9.3 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-007>

**10. 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。

**10.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**10.2 编写代码**

使用redirect时，给它一个字符串类型的参数就行了。

编辑HelloWorld/server.py：

1. from flask import Flask, url\_for, redirect
3. app = Flask(\_\_name\_\_)
5. @app.route('/')
6. def hello\_world():
7. return 'hello world'
9. @app.route('/test1')
10. def test1():
11. print('this is test1')
12. return redirect(url\_for('test2'))
14. @app.route('/test2')
15. def test2():
16. print('this is test2')
17. return 'this is test2'
19. if \_\_name\_\_ == '\_\_main\_\_':
20. app.run(debug=True)

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

**10.3 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-008>

**11. 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。

这个实例中使用了模板继承、if判断、for循环。

**11.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**11.2 创建并编辑HelloWorld/templates/default.html**

内容如下：

1. <html>
2. <head>
3. <title>
4. {% if page\_title %}
5. {{ page\_title }}
6. {% endif %}
7. </title>
8. </head>
10. <body>
11. {% block body %}{% endblock %}

14. ```
15. 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。
16. ``标签中定义了一个名为`body`的block，用来被其他模板文件继承。
17. ### 11.3 创建并编辑HelloWorld/templates/user\_info.html
18. 内容如下：
19. ```
20. {% extends "default.html" %}
22. {% block body %}
23. {% for key in user\_info %}
25. {{ key }}: {{ user\_info[key] }}

28. {% endfor %}
29. {% endblock %}

变量user\_info应该是一个字典，for循环用来循环输出键值对。

**11.4 编辑HelloWorld/server.py**

内容如下：

1. from flask import Flask, render\_template
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

11. @app.route('/user')
12. def user():
13. 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)

22. if \_\_name\_\_ == '\_\_main\_\_':
23. app.run(port=5000, debug=True)

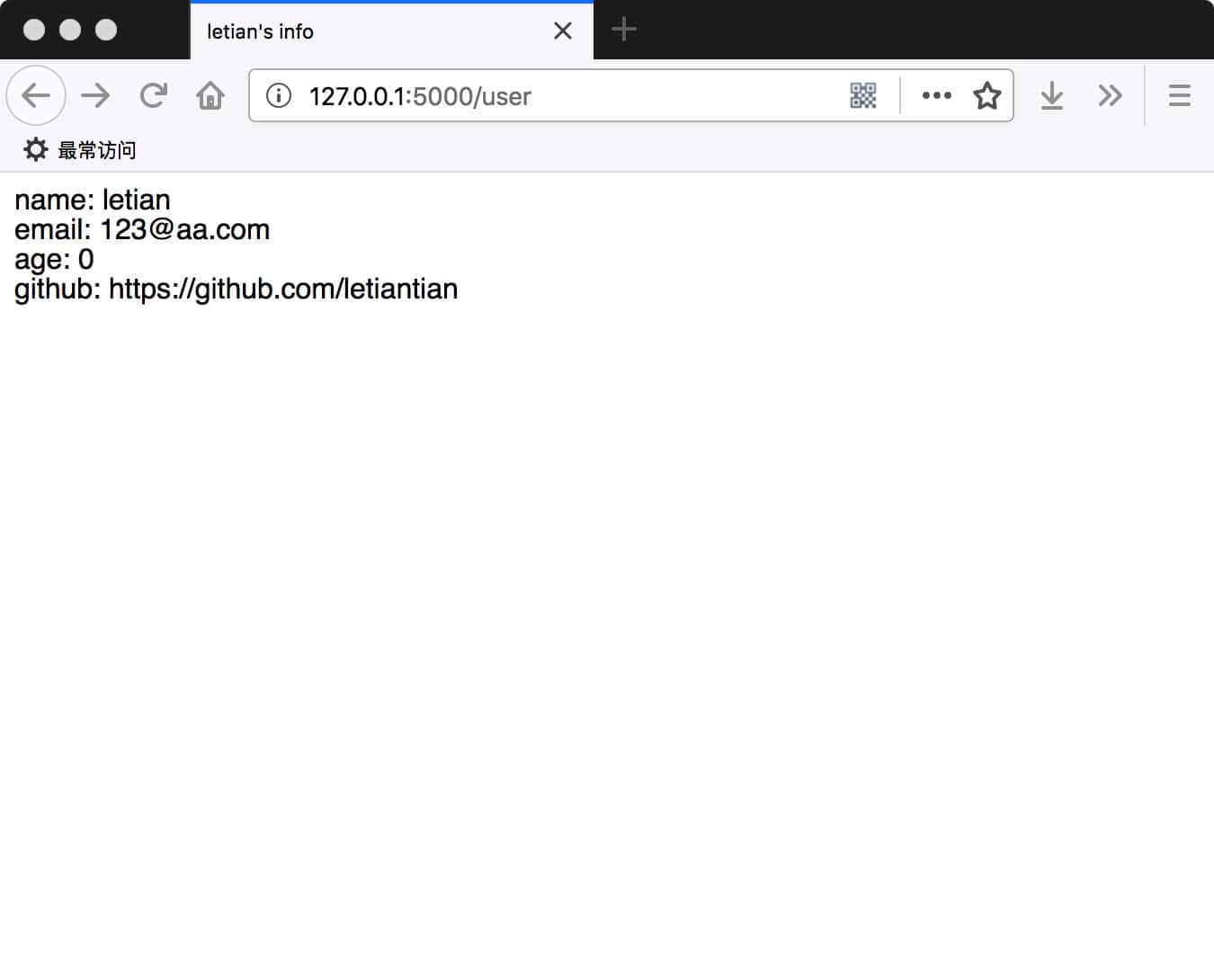
render\_template()函数的第一个参数指定模板文件，后面的参数是要传递的数据。

**11.5 运行与测试**

运行HelloWorld/server.py：

$ python3 HelloWorld/server.py

在浏览器中访问http://127.0.0.1:5000/user，效果图如下：

[](https://www.letiantian.me/content/images/learn-flask/flask-003.jpg)

查看网页源码：

1. <html>
2. <head>
3. <title>
4. letian&#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>

**11.6 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-009>

**12. 自定义404等错误的响应**

要处理HTTP错误，可以使用flask.abort函数。

**12.1 示例1：简单入门**

建立Flask项目

按照以下命令建立Flask项目HelloWorld:

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

代码

编辑HelloWorld/server.py：

1. from flask import Flask, render\_template\_string, abort
3. app = Flask(\_\_name\_\_)

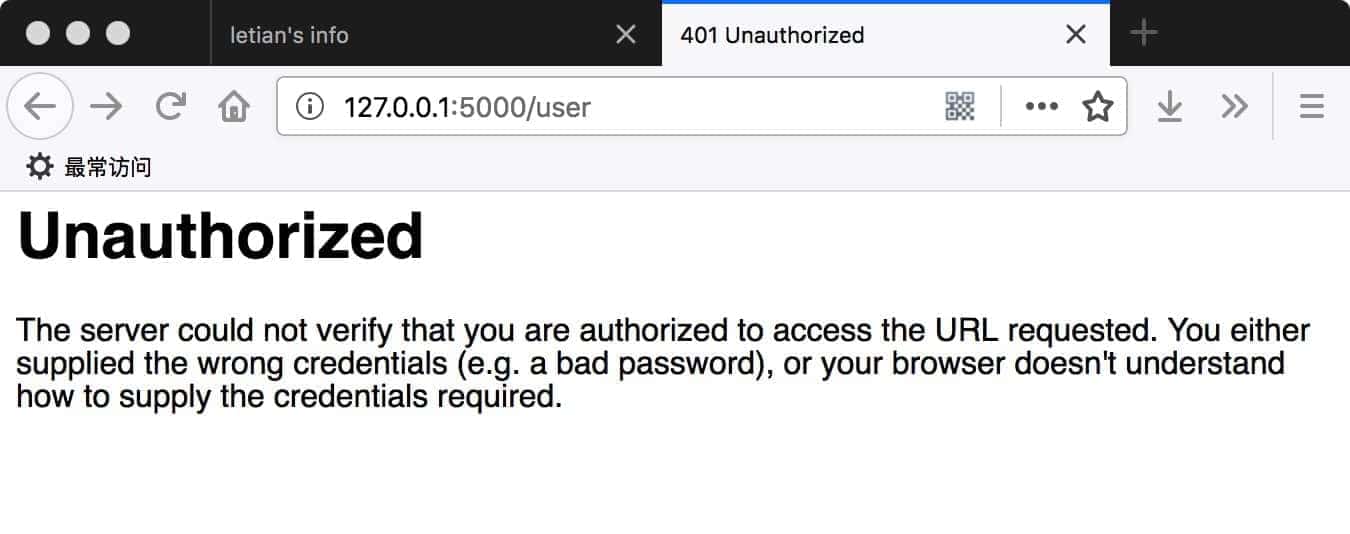
6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

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

17. if \_\_name\_\_ == '\_\_main\_\_':
18. app.run(port=5000, debug=True)

效果

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

[](https://www.letiantian.me/content/images/learn-flask/flask-004.jpg)

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

**12.2 示例2：自定义错误页面**

代码

将服务器代码改为：

1. from flask import Flask, render\_template\_string, abort
3. app = Flask(\_\_name\_\_)

6. @app.route('/')
7. def hello\_world():
8. return 'hello world'

11. @app.route('/user')
12. def user():
13. abort(401) *# Unauthorized*

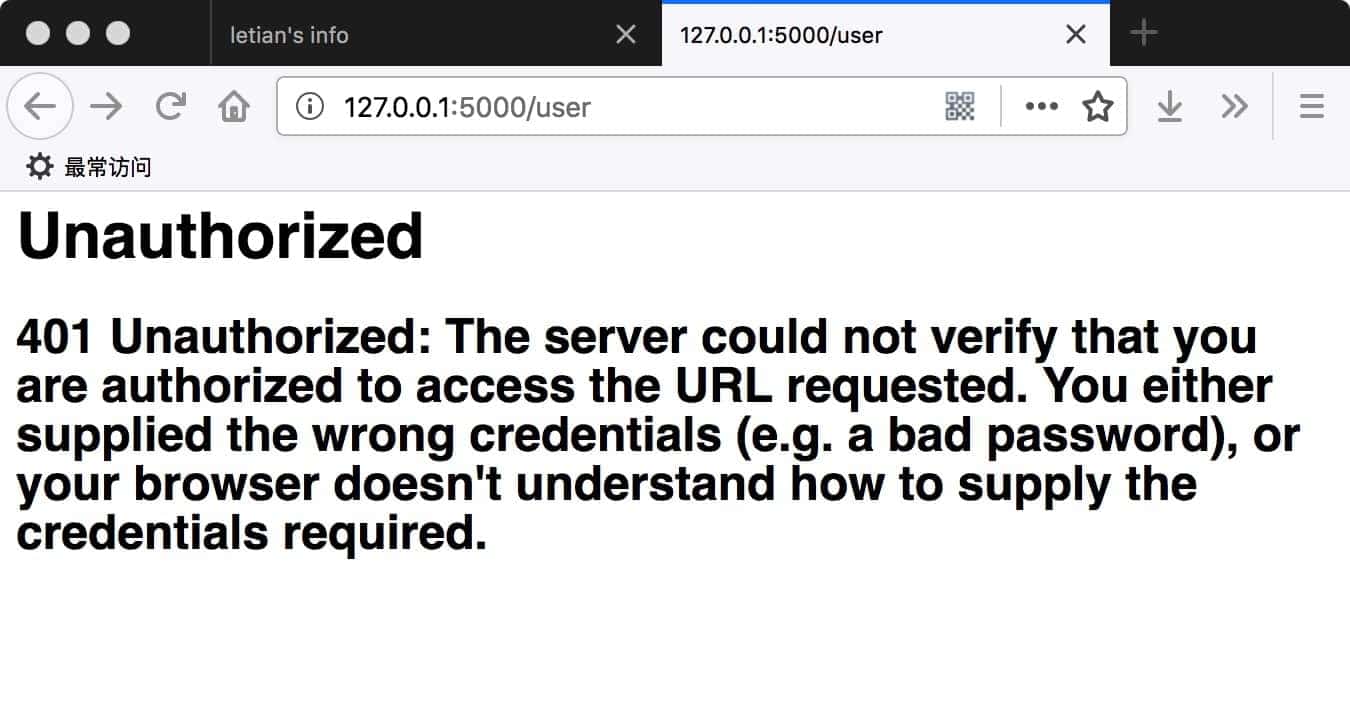
16. @app.errorhandler(401)
17. def page\_unauthorized(error):
18. return render\_template\_string('<h1> Unauthorized </h1><h2>{{ error\_info }}</h2>', error\_info=error), 401

21. if \_\_name\_\_ == '\_\_main\_\_':
22. 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。

效果

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

[](https://www.letiantian.me/content/images/learn-flask/flask-005.jpg)

**12.3 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-010>

**13. 用户会话**

session用来记录用户的登录状态，一般基于cookie实现。

下面是一个简单的示例。

**13.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**13.2 编辑HelloWorld/server.py**

内容如下：

1. from flask import Flask, render\_template\_string, \
2. session, request, redirect, url\_for
4. app = Flask(\_\_name\_\_)
6. app.secret\_key = 'F12Zr47j\3yX R~X@H!jLwf/T'

9. @app.route('/')
10. def hello\_world():
11. return 'hello world'

14. @app.route('/login')
15. def login():
16. page = '''
17. <form action="{{ url\_for('do\_login') }}" method="post">
18. <p>name: <input type="text" name="user\_name" /></p>
19. <input type="submit" value="Submit" />
20. </form>
21. '''
22. return render\_template\_string(page)

25. @app.route('/do\_login', methods=['POST'])
26. def do\_login():
27. name = request.form.get('user\_name')
28. session['user\_name'] = name
29. return 'success'

32. @app.route('/show')
33. def show():
34. return session['user\_name']

37. @app.route('/logout')
38. def logout():
39. session.pop('user\_name', None)
40. return redirect(url\_for('login'))

43. if \_\_name\_\_ == '\_\_main\_\_':
44. app.run(port=5000, debug=True)

**13.3 代码的含义**

app.secret\_key用于给session加密。

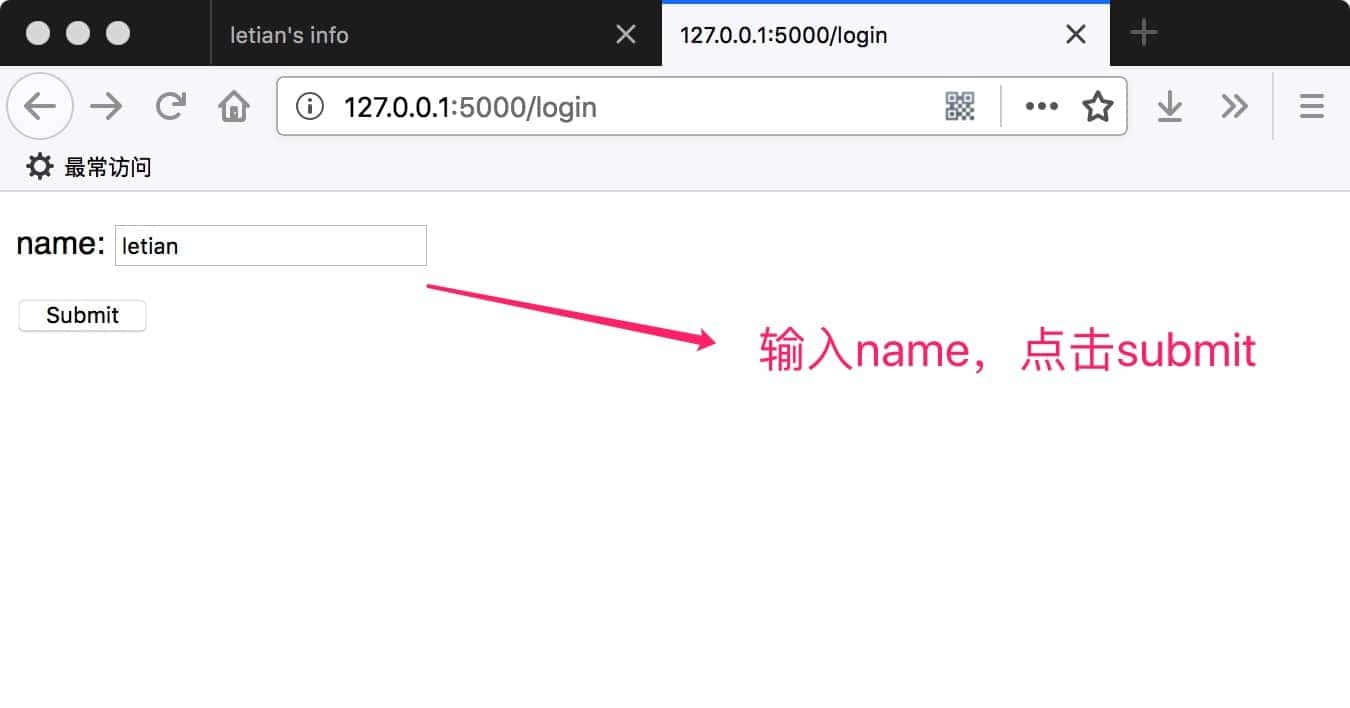
在/login中将向用户展示一个表单，要求输入一个名字，submit后将数据以post的方式传递给/do\_login，/do\_login将名字存放在session中。

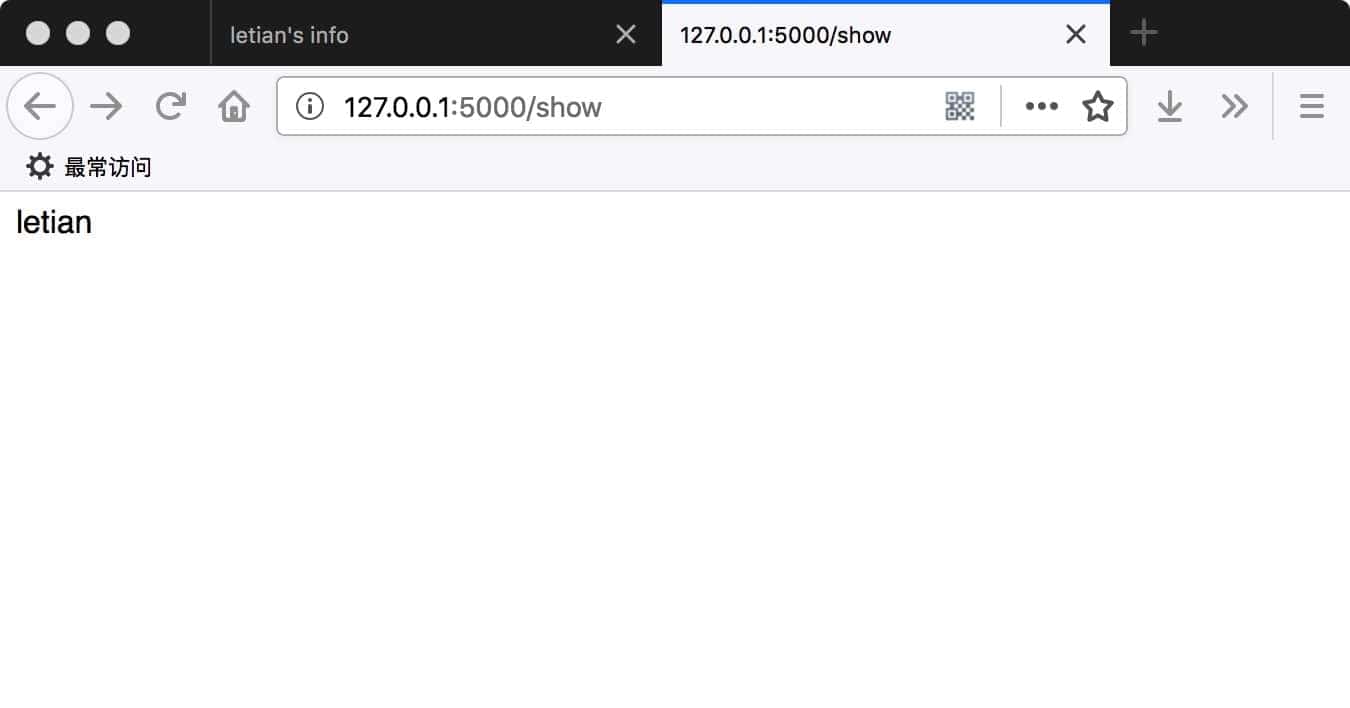
如果用户成功登录，访问/show时会显示用户的名字。此时，打开firebug等调试工具，选择session面板，会看到有一个cookie的名称为session。

/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。

另外，使用redirect()重定向时，一定要在前面加上return。

**13.4 效果**

进入http://127.0.0.1:5000/login，输入name，点击submit：  
[](https://www.letiantian.me/content/images/learn-flask/flask-006.jpg)

进入http://127.0.0.1:5000/show查看session中存储的name：  
[](https://www.letiantian.me/content/images/learn-flask/flask-007.jpg)

**13.5 设置sessin的有效时间**

下面这段代码来自[Is there an easy way to make sessions timeout in flask?](http://stackoverflow.com/questions/11783025/is-there-an-easy-way-to-make-sessions-timeout-in-flask)：

1. from datetime import timedelta
2. from flask import session, app
4. session.permanent = True
5. app.permanent\_session\_lifetime = timedelta(minutes=5)

这段代码将session的有效时间设置为5分钟。

**13.6 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-011>

**14. 使用Cookie**

Cookie是存储在客户端的记录访问者状态的数据。具体原理，请见 <http://zh.wikipedia.org/wiki/Cookie> 。 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。下面是一个示例。

**14.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**14.2 代码**

修改HelloWorld/server.py：

1. from flask import Flask, request, Response, make\_response
2. import time
4. app = Flask(\_\_name\_\_)

7. @app.route('/')
8. def hello\_world():
9. return 'hello world'

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

19. @app.route('/show')
20. def show():
21. return request.cookies.\_\_str\_\_()

24. @app.route('/del')
25. def del\_cookie():
26. res = Response('delete cookies')
27. res.set\_cookie('name', '', expires=0)
28. return res

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](http://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.

**14.3 运行与测试**

运行HelloWorld/server.py：

$ python3 HelloWorld/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

下面查看一下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=/

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'}

**14.4 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-012>

**15. 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。

下面是一个示例。

**15.1 建立Flask项目**

按照以下命令建立Flask项目HelloWorld:

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

**15.2 编写HelloWorld/server.py**

内容如下：

1. from flask import Flask, flash, get\_flashed\_messages
2. import time
4. app = Flask(\_\_name\_\_)
5. app.secret\_key = 'some\_secret'

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

13. @app.route('/gen')
14. def gen():
15. info = 'access at '+ time.time().\_\_str\_\_()
16. flash(info)
17. return info

20. @app.route('/show1')
21. def show1():
22. return get\_flashed\_messages().\_\_str\_\_()

25. @app.route('/show2')
26. def show2():
27. return get\_flashed\_messages().\_\_str\_\_()

30. if \_\_name\_\_ == "\_\_main\_\_":
31. app.run(port=5000, debug=True)

**15.3 效果**

运行服务器：

$ python3 HelloWorld/server.py

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

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

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

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：

[]

**15.4 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：

1. from flask import Flask, flash, get\_flashed\_messages
2. import time
4. app = Flask(\_\_name\_\_)
5. app.secret\_key = 'some\_secret'

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

13. @app.route('/gen')
14. def gen():
15. info = 'access at '+ time.time().\_\_str\_\_()
16. flash('show1 '+info, category='show1')
17. flash('show2 '+info, category='show2')
18. return info

21. @app.route('/show1')
22. def show1():
23. return get\_flashed\_messages(category\_filter='show1').\_\_str\_\_()
25. @app.route('/show2')
26. def show2():
27. return get\_flashed\_messages(category\_filter='show2').\_\_str\_\_()

30. if \_\_name\_\_ == "\_\_main\_\_":
31. 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：

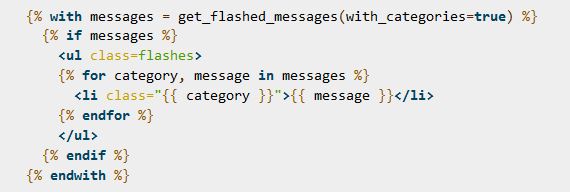
['1 access at 1404022326.39']

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

[]

**15.5 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：

[](https://www.letiantian.me/content/images/learn-flask/flask-008.jpg)

**15.6 本节源码**

<https://github.com/letiantian/Learn-Flask/tree/master/flask-demo-013>

* **本文作者：** letiantian
* 更多Python视频、源码、资料加群683380553免费获取
* **本文链接：** <http://www.letiantian.me/learn-flask/>