

Flask自学笔记

原创

Zerg_Wang

于 2019-06-03 11:55:31 发布 146 收藏 1

编辑 版权

分类专栏： Web Development



Web Development 专栏收录该内容

0 订阅 3 篇文章

开始

Flask，一个Python下的轻量级Web应用框架。使用Flask可实现一些简单的网站服务。

安装：

```
pip install Flask
```

一个最简单的程序

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/123/')
5 def hello_world():
6     return 'Hello World!'
7
8 if __name__ == '__main__':
9     app.run()
```

运行该程序后，在本机浏览器打开127.0.0.1:5000/123/可看到页面中显示的“Hello World!”。

app.run()

app.run()中可填入参数：host、port、debug，也可不填，这时取默认值。

```
app.run(host='0.0.0.0', port='5200', debug=True)
```

host指定IP地址，本机运行，默认为127.0.0.1，若该程序运行在服务器，设置为host='0.0.0.0'，然后在其他机器的浏览器访问服务器IP地址即可。

port为端口，默认为5000。

debug为调试模式，默认关闭。开启后调试模式后运行程序，只要对代码进行改动，保存后无需重新编译运行，即可在浏览器处看到变化。

@app.route()

与其之后的def函数相配对，每个app.route对应一个def函数。用于为def指定一个URL。例如上面的例子，访问网址127.0.0.1:5000/123/，就会执行hello_world函数，在页面上显示“Hello World!”。

每个def对应一个唯一的URL，此外，在为def分配URL时要注意是否需要“/”，例如

```
1 @app.route('/123/')
2 @app.route('/123')
```

这两个是不一样的，如果两者都存在，则会严格按app.route分配的URL执行def函数，若只有前者，则访问不带斜杠的会自动重定向到带斜杠的地址。若只有后者，访问带斜杠的则会404。

与@app.route()相似功能的还有app.add_url_rule，如果要与上面例子一致，则写法为：

```
app.add_url_rule('/123/', 'hello_world', hello_world)
```

一般写在def函数后面，第一个参数为指定的URL，第三个为函数名，第二个为endpoint，一般与函数名一致，具体用法会在url_for提到。
@app.route也可指定endpoint，在URL后面添加endpoint='hello_world'即可。

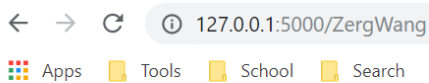
除了以上参数，还可以指定函数的HTTP方法：

```
app.add_url_rule('/upload/', 'upload_page', upload_page, methods=['GET', 'POST'])
```

URL的变量部分（动态路由）

通过@app.route可为def指定固定的URL（app.add_url_rule同样适用），但这个URL是可以变化的：

```
1 | @app.route('/<username>')
2 | def welcome(username):
3 |     return 'Welcome: %s' %username
```



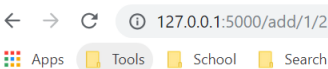
← → ↻ ⓘ 127.0.0.1:5000/ZergWang

Apps Tools School Search

Welcome: ZergWang

可以根据URL而进行变化。这里要注意的是，URL中允许的变量默认为字符串，def中return的也默认为字符串，因此要进行格式化输出。如果要指定其他数据类型，则要进行相应变化，下面的这个例子可以通过URL进行加法运算：

```
1 | @app.route('/add/<int:num1>/<int:num2>')
2 | def add(num1, num2):
3 |     return '%d' %(num1+num2)
```



← → ↻ ⓘ 127.0.0.1:5000/add/1/2

Apps Tools School Search

3

除了int，也可指定为float型。（不过好像不支持负数，可能是负号的原因……而且选择为float，输入一个整数就404了……）

函数返回部分

以上面利用动态路由做加法的例子，其返回值还可以这么写：

```
1 | @app.route('/add/<float:num1>/<float:num2>')
2 | def add(num1, num2):
3 |     return '<h1>'+str(num1+num2)+'</h1>'
```

可以发现，作为返回值的字符串可以作为页面的HTML代码。进一步，返回值可以是事先编辑好的本地HTML文件：

```
1 | @app.route('/')
2 | def welcome():
3 |     return render_template("index.html")
```

使用render_template，要引用：

```
from flask import render_template
```

本地的index.html还要放在与此代码文件同级的名为templates的文件夹中。

url_for函数的使用

用法一：获取函数对应的URL

每个路由对应一个def函数，通过访问URL来执行函数，然而在程序中，我们可以通过url_for函数来通过函数名得到其对应的URL。例如：

```
1 | @app.route('/download/')
2 | def back_to_index():
3 |     print(url_for('index_page'))
4 |     return redirect(url_for('index_page'))
5 |
6 | @app.route('/index/')
7 | def index_page():
8 |     return 'Here is index'
```

url_for('index_page')则返回函数index_page上@app.route括号中的内容，即其对应的URL：/index/，虽然省略了前面的127.0.0.1:5000，然而通过这个简短的URL仍可以正常访问index主页。

假如说127.0.0.1:5000/download/这个页面还未编辑好，用户访问到这个页面时，我们要将页面重定向到index主页，可以通过url_for获得这个主页的地址，然后使用redirect重定向函数即可。使用前：

```
from flask import url_for, redirect
```

实际上，url_for()并不是通过函数名找到URL，而是通过endpoint找到URL，但默认每个函数的endpoint与函数名一致，看起来就是直接通过函数名找的URL……。用好endpoint，会使程序执行效率更高，之后在编辑Blueprint相关功能时更能起到关键的作用。

用法二：为静态文件配置URL

一个网页除了HTML，可能还有CSS、JavaScript等静态文件，以及一些图片、音乐等的媒体文件，通过url_for可以为它们分配URL使用户可以访问。

以上的静态文件、媒体文件放在与代码同级的名为static的文件夹下。

```
1 | @app.route('/photo/')
2 | def back_to_index():
3 |     return redirect(url_for('static', filename='astronaut.jpg'))
```

访问127.0.0.1:5000/photo/，即可重定向到127.0.0.1:5000/static/astronaut.jpg，显示出照片。当然，直接访问后面这个URL也可以。

HTTP方法及请求对象

HTTP方法简单来说就是客户端（浏览器）对网页的操作，例如GET方法，即是获取页面信息，POST方法即上传某些内容使之显示于页面中（）。定义页面中可用什么方法：

```
1 | @app.route('/index/', methods=['GET', 'POST'])
2 | def visit_index():
3 |     if request.method == 'POST':
4 |         ...
5 |     else:
6 |         ...
```

然后在def中详细为各种方法制定执行内容。我们通过URL直接访问页面，执行的是GET方法。这里写一个简单的登录页面：

```
1 | @app.route('/login/', methods=['POST', 'GET'])
2 | def login():
3 |     if request.method == 'POST':
4 |         if request.form['userID'] == 'ZergWang' and request.form['password'] == '123':
5 |             return 'Login Successfully'
6 |         else:
7 |             return 'Wrong ID or Password'
```

```
8 |         else:
9 |             return render_template('login.html')
```

以下是login.html的代码：

```
1 | <!DOCTYPE html>
2 | <html>
3 | <body>
4 |     <form method="POST">
5 |         <fieldset>
6 |             <legend>用户登录</legend>
7 |             <p>请输入用户名: </p>
8 |             <input type="text" name="userID" /><br/>
9 |             <p>请输入密码: </p>
10 |            <input type="password" name="password" />
11 |            <input type="submit" />
12 |        </fieldset>
13 |    </form>
14 </body>
15 </html>
```

运行程序，访问127.0.0.1:5000/login/，执行的是GET方法，加载login.html的页面：



The screenshot shows a web browser window with the title '用户登录'. Inside the browser, there is a form with two input fields. The first field is labeled '请输入用户名:' and the second is labeled '请输入密码:'. Below the password field is a 'Submit' button. The browser's address bar shows 'http://127.0.0.1:5000/login/'.

输入账号密码后，点击提交，因为在login.html中规定了点击提交按钮使用POST方法（method="POST"），则Python代码执行检测账号密码是否匹配，这里通过request.form获取HTTP表单中填入的值（即账号密码，通过<input>中设置的name属性来对应）。

使用request要：

```
from flask import request
```

文件上传

与前面的类似，使用request.file获得用户上传的文件并保存到服务器本地：

```
1 | @app.route('/upload/', methods=['GET', 'POST'])
2 | def upload_page():
3 |     if request.method == 'POST':
4 |         f = request.files['myfile']
5 |         f.save('files/' + f.filename)
6 |         return 'Upload Successful'
7 |     else:
8 |         return render_template('upload.html')
```

```
1 | <form method="POST" enctype="multipart/form-data">
2 | <fieldset>
3 |     <legend>文件上传</legend>
4 |     <input type="file" name="myfile" />
5 |     <input type="submit" />
6 | </fieldset>
7 | </form>
```

这里需要注意的是表单form处要设置属性enctype，这个是设置表单数据的编码方式，multipart/form-data适用于非文本文件的传输。

f.save即把上传文件保存到本地，相对或绝对路径均可，后面的f.filename为用户上传文件的文件名。

此外，函数secure_filename可以安全获取文件名，但不支持中文，例如“文件.doc”，接收后会文件名变成“doc”）。要使用需要：

```
from werkzeug.utils import secure_filename
```

用法：

```
f.save('files/' + secure_filename(f.filename))
```

错误

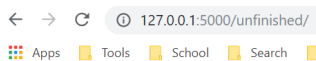
与redirect类似，当某个页面未开发完或者有其他用途而要显示指定的错误信息时，可以使用abort()函数：

```
1 | @app.route('/unfinished/')
2 | def unfinished_page():
3 |     abort(401)
```

使用前：

```
from flask import abort
```

打开页面会看到：



Unauthorized

The server could not verify that you are authorized to access the requested URL. You either supplied the wrong credentials (e.g. password or basic authentication) or your browser doesn't understand how to supply the credentials.

abort()中的参数为错误类型，401为未授权的访问，还有大家熟知的404，一切正常是200，403是禁止访问等等。

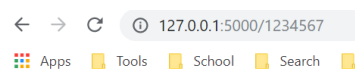
当然，如果为每个页面都写个abort过于麻烦，但不写直接404又不太好，实际上，页面的错误信息可以自定义：

```
1 | @app.errorhandler(404)
2 | def page_not_found(error):
3 |     return render_template('page_not_found.html')
```

page_not_found.html：

```
1 | <!DOCTYPE HTML>
2 | <html>
3 | <body>
4 |     <p>这个页面还未完成编辑</p>
5 |     <p>点击返回首页：<a href="http://127.0.0.1:5000">首页</a></p>
6 | </body>
7 | </html>
```

对于自己URL下所有不存在的页面，都可以通过此定义显示为自己定义的错误页面，例如，我随便输入一个不存在的URL：



这个页面还未完成编辑

点击返回首页：[首页](http://127.0.0.1:5000/)

参考资料

