

Flask 入门

简介

Python Web 框架是我们可以用来开发 Web 应用的 Python 库，目前比较著名是 Flask 和 Django。

Flask 是 Python 社区比较知名的微框架。Flask 本身只维护一个核心，被设计成可以通过插件拓展。如果要和另一个知名的 Python Web 框架 Django 做对比的话，Django 更像一个大品牌出的电脑整机，你不用操心使用什么配件，你需要什么 Django 全家桶都有。而 Flask 可以说是一个组装机了，更准确的说是一个设计精良的 CPU。这给了你很大的灵活性去选择需要的配件（插件）。

知识点

- Flask 简介
- 配置方法
- 路由和视图函数
- 模板渲染
- GET 与 POST
- session
- cookies
- 错误处理
- 插件

安装Flask

Flask 有两个主要依赖：一个是负责路由、调试和 web 服务器网关接口子系统的Werkzeug (<http://werkzeug.pocoo.org/>)，另一个是模板系统jinja2 (<http://jinja.pocoo.org/>)。

在实验环境中默认已经安装 Flask 0.12.2，可以通过如下方式查看：

```
shianlou:~/ $ python3
Python 3.5.3 (default, Apr 22 2017, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import flask
>>> flask.__version__
'0.12.2'
>>>
```

如果大家需要在本地环境中安装 Flask，可以通过 `sudo pip3 install flask==0.12.2` 进行安装。
🔗 楼+之Python实战第10期 (/courses/1190)

最小的应用

一个最简单的例子，创建 `/home/shiyanlou/app.py`，写入：

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello World!' # 浏览器访问主页面会显示一个 'Hello World!'

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

`app = Flask(__name__)`：这是用于初始化。使用时必须创建 Flask 类的实例对象 `app`，服务器使用 wsgi 协议把来自客户端的所有请求都转交给 `app` 对象处理。在生成实例对象时，需要传递 `__name__` 参数，这个参数在 Python 解释器中会被解释为程序主模块或包的名字，这个参数决定了程序的根目录，以便之后从根目录去找到其它资源文件。

`@app.route (mailto: `@app.route`)'('/')``：这是路由。我们在浏览器中输入一个 URL，浏览器发送请求 URL 给 web 服务器，服务器转发请求 URL 给程序实例 `app`，程序实例 `app` 把请求 URL 映射到对应的处理函数进行处理。处理 URL 和函数之间映射关系的程序称为路由。在 Flask 中定义路由会使用 `@app.route()` 装饰器，参数为请求的相对路径。

`index()`：这是视图函数。对客户端请求进行处理的函数被称为视图函数，视图函数处理后会返回处理结果，再由 web 服务器把结果返回给客户端。

Flask 提供了一个管理 Flask 应用的命令行工具，首先要设置应用的环境变量，在终端中执行如下命令：

```
export FLASK_APP=app.py
export FLASK_DEBUG=1
```

环境变量 `FLASK_APP` 是用来指向 `flask run` 执行的 Flask 应用代码的路径，这里是 `app.py` 的路径。`FLASK_DEBUG=1` 表示打开 DEBUG 信息，可以输出访问和出错信息，帮助我们解决代码中出现的的问题，建议后续只要执行 `flask run` 都要打开 `FLASK_DEBUG`，开启 DEBUG 模式后可以自动重载代码。

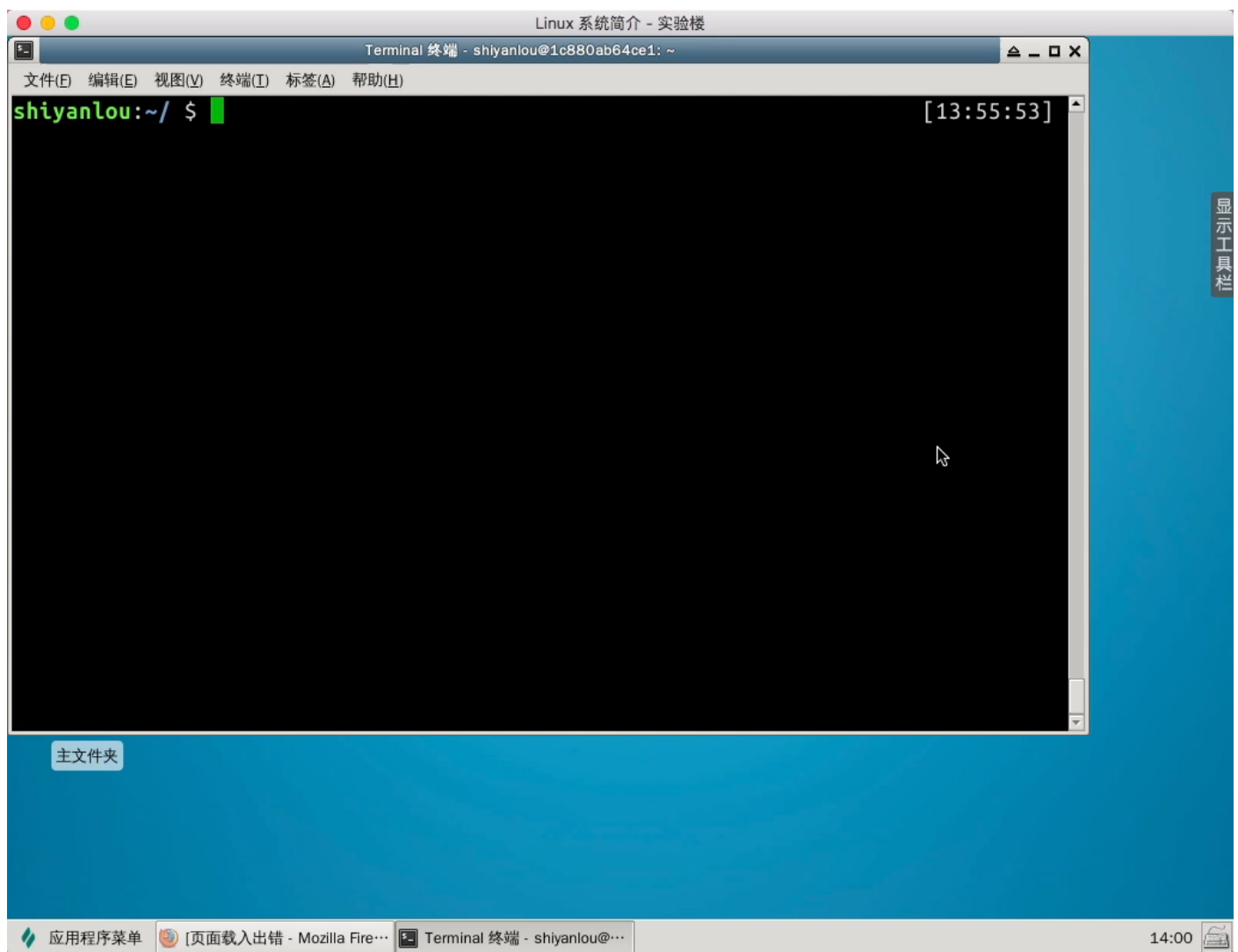
然后就可以这样运行 Flask 应用了：

```
shyanlou@py:~$ flask run
* Serving Flask app "app"
* Forcing debug mode on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 294-454-826
```

默认地，应用运行在 `localhost:5000` 上。打开实验环境中的浏览器，访问这个地址就能看到返回的 `Hello World!` 了。

如果想要停止该应用，按 `Ctrl+C` 即可。

Flask 简单示例开发视频：



练习题：创建 Flask 应用

注意：练习题的目录和学习的目录是分开的，本节实验中的练习题应该在 `/home/shiyanlou/flasktest/` 目录下操作，而操作演示的代码在 `/home/shiyanlou/` 目录下执行

首先使用 `Ctrl + C` 停止前面运行的 `flask run` 命令，然后创建一个新的应用：

```
cd /home/shiyanlou/flasktest
touch app.py
```

在 `/home/shiyanlou/flasktest/app.py` 中完成以下代码：

```
from flask import Flask

app = Flask(__name__)

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

这段代码创建并运行了一个简单的 Flask app，需要你补充一个路由，当应用运行后，我们访问 `http://127.0.0.1:5000/` 的时候，可以看到 `Hello Shiyanlou!` 字符串显示在页面上。

完成代码后，需要在终端按照之前的所学配置 `FLASK_APP` 环境变量并使用 `flask run` 启动应用。

注意本节实验后续的几个题目是连续的，所以 `/home/shiyanlou/flasktest` 目录创建后不要删除。

应用启动后，点击 [下一步](#)，系统将自动检测完成结果。

flask shell

除了 `flask run` 之外，还有一个常用的命令是 `flask shell`，这两个命令都会自动把 `FLASK_APP` 环境变量中指定的代码模块进行加载，不同的是 `flask run` 直接进入运行 `app` 的状态，而 `flask shell` 只加载并进入到一个 Shell 终端，在这个终端中可以执行一些代码，比如后续章节中要用到的初始化数据库，向数据库中插入一些数据等。

```
$ export FLASK_APP=app.py
$ export FLASK_DEBUG=1
$ flask shell
Python 3.5.3 (default, Apr 22 2017, 00:00:00)
[GCC 4.8.4] on linux
App: app [debug]
Instance: /home/shiyanlou/instance
>>>
```

配置

初始化一个 Flask app 后，可以通过 `app.config` 管理配置。`app.config` 存储的配置信息本质上是个字典，所以你可以用字典的方法添加或者更新配置。比如说，初始化 `app` 后，配置一个密钥：

```
app.config.update({
    'SECRET_KEY': 'a random string'
})
```

所有的配置选项需要用大写，多个单词间用下划线 _ 连接。大型项目中，配置通常写在一个单独的 config.py 文件中，这时候就可以用 app.config 提供的特有方法来更新 config，参数是配置文件 config.py 的路径：

```
app.config.from_pyfile('path/to/config.py')
```

其他类似的方法：

- from_envvar(variable_name)：使用一个环境变量指定的配置文件更新配置
- from_object(obj)：使用一个对象更新配置文件，dict 无效
- from_json(filename)：使用 JSON 文件更新配置
- from_mapping(*mapping, **kwargs)：类似前面的 update，不同的是，这个方法不强制使用大写字母

获得一个配置信息的方法是用字典的形式 app.config['SECRET_KEY'] 这样可以获得 SECRET_KEY 的配置值。

路由和视图函数

Flask 使用 `@app.route` (mailto:`@app.route`) 装饰器来将路由映射到对应的视图函数上，形成一一对应的关系，每一个路由都有对应的视图函数进行处理。

前面的例子如下，我们添加了一个 / 路由，当使用 `http://127.0.0.1:5000/` 访问网站主页时，Flask 会用 index 函数来处理。

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

上面演示的是静态固定的路由，也可以在路由中传入变量，格式为 `<variable_name>`，比如每个用户的主页需要不同的路由，可以使用用户名作为路由的变量，向 /home/shiyanlou/app.py 文件中添加如下代码：

```
@app.route('/user/<username>')
def user_index(username):
    # 在函数中指明变量名称 username，就能获取到通过路由传入的变量值 username
    return 'Hello {}'.format(username)
```

还可以指定路由变量的类型，比如说，一个博客应用的每个博文页面可以用这篇博文的 ID 作为路由变量，ID 应该是个 int 类型的值，向 `/home/shiyanlou/app.py` 文件中添加如下代码：

```
@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post {}'.format(post_id)
```

路由 return 的内容会包含在返回给用户的响应中，这两个路由都返回字符串，所以用户使用浏览器访问这两个链接地址的时候看到的就是字符串显示在浏览器页面上。

在这个例子中，使用 `flask run` 启动应用，访问地址 `http://127.0.0.1:5000/user/shixiaolou` 可以看到页面上显示 `Hello shixiaolou`，访问地址 `http://127.0.0.1:5000/post/9` 可以看到页面上显示 `Post 9`。这样我们就获取到了路由中传递的变量并将它显示在了页面上。

练习题：为 Flask 应用增加路由及视图函数

首先使用 `Ctrl + C` 停止前面运行的 flask 应用，然后进入到 `/home/shiyanlou/flasktest` 中为应用增加路由及视图函数：

```
$ cd /home/shiyanlou/flasktest
$ vim app.py
```

在 `/home/shiyanlou/flasktest/app.py` 中增加以下路由及视图函数：

1. 增加路由链接地址为 `/courses/<name>`，并添加对应的视图函数 `courses()`，要求当访问 `http://localhost:5000/courses/linux` 时，页面上显示字符串 `Courses:linux`

完成路由后，需要在终端按照之前的所学配置 `FLASK_APP` 环境变量并使用 `flask run` 启动应用。

注意本节实验后续的几个题目是连续的，所以 `/home/shiyanlou/flasktest` 目录创建后不要删除。

应用启动后，点击 下一步，系统将自动检测完成结果。

url_for

在 Flask 中不仅可以静态匹配 URL，也可以使用 `url_for()` 函数在程序中动态构造 URL。

`url_for()` 函数的第一个参数是视图函数的名字，如果该视图函数对应的 URL 需要传递参数则将参数以 `key=value` 的形式写在后面，未知变量部分会添加到 URL 末尾作为查询参数。`url_for()` 函数最终返回的是视图函数对应的 URL 地址。

使用方法为：`url_for('函数名', 命名参数, 其它参数, _external, _anchor)`

其中：

- external=True: 返回绝对地址，不填的话默认返回相对路径
- _anchor: 用于生成锚点，就是我们经常在浏览器中见到的类似 # 这样的链接地址：http://example.com/home#xxx

把下面的代码写入 /home/shiyanlou/app.py 文件中：

```
from flask import url_for

@app.route('/test')
def test():
    print(url_for('index'))
    print(url_for('user_index', username='shixiaolou'))
    print(url_for('show_post', post_id=1, _external=True))
    print(url_for('show_post', post_id=2, q='python 03'))
    print(url_for('show_post', post_id=2, q='python可以'))
    print(url_for('show_post', post_id=2, _anchor='a'))
    return 'test'
```

保存后执行 flask run 并访问地址 http://127.0.0.1:5000/test ，可以看到页面上显示 test 字符串，并在 Xfce 终端中打印了如下内容：

```
/
/user/shixiaolou
http://127.0.0.1:5000/post/1
/post/2?q=python+03
/post/2?q=python%E5%8F%AF%E4%BB%A5 # 可以看到通过构建的方式可以自动转义特殊字符和Unicode数据
/post/2#a
```

redirect

在 Flask 中可以使用重定向 redirect() 把来自客户端的请求 URL 地址重定向到另外一个 URL 地址，实现访问地址的自动跳转。redirect() 方法通常和 url_for() 方法一起使用。

把下面的代码写入 /home/shiyanlou/app.py 文件中：

```
from flask import redirect

@app.route('/<username>')
def hello(username):
    if username == 'shixiaolou': # 如果访问 /shixiaolou 则显示页面
        return 'hello {}'.format(username)
    else:
        return redirect(url_for('index')) # 否则重定向到首页
```

保存后执行 flask run 并访问 http://127.0.0.1:5000/shixiaolou 可以看到页面显示 hello shixiaolou，然后再访问 http://127.0.0.1:5000/xiaoming 可以看到浏览器自动跳转到了 http://127.0.0.1:5000 地址，同时页面显示 Hello World!。

练习题：为 Flask 应用动态构造 URL 并实现重定向

楼+之Python实战第10期 (/courses/1190)

首先使用 `Ctrl + C` 停止前面运行的 flask 应用，然后进入到 `/home/shiyanlou/flasktest` 中为应用增加路由：

```
cd /home/shiyanlou/flasktest
vim app.py
```

在 `/home/shiyanlou/flasktest/app.py` 中动态构造 URL 并实现重定向：

1. 定义一个路由 `/test`，为这个路由绑定一个 `test()` 视图函数，要求函数实现构造一个 URL，在运行程序后，访问 `http://127.0.0.1:5000/test` 地址，可以在终端中打印出 `http://127.0.0.1:5000/courses/java`，然后把 URL 地址重定向到网站首页，页面上显示 `Hello Shiyanlou!`

完成后，需要在终端按照之前的所学配置 `FLASK_APP` 环境变量并使用 `flask run` 启动应用。

注意本节实验后续的几个题目是连续的，所以 `/home/shiyanlou/flasktest` 目录创建后不要删除。

应用启动后，点击 [下一步](#)，系统将自动检测完成结果。

模板渲染

在上面的例子中，处理函数返回的都是字符串，但是在真正的项目中，需要使用 HTML 编写页面，不可能把所有的内容都写到字符串中。模板引擎的作用就是你用模板引擎规定的语法编写 HTML 页面，在处理函数中指定模板，传入相应的模板变量，Flask 就能调用模板引擎自动渲染出一个完整的 HTML 页面。

Flask 默认的模板引擎是 `jinja2`，理论上你是可以更换其它模板引擎的，但是 `jinja2` 已经足够好用。

Flask 使用 `render_template` 函数渲染模板，指定了一个模板名称后，Flask 会到 `templates` 目录下去找这个模板，然后使用传入的变量渲染模板。

如果我们用模板来改写用户主页的例子，代码放在 `/home/shiyanlou/flask-test-app` 目录，那么处理函数可以这样写，在 `flask-test-app/app.py` 文件中写入如下代码：

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/user/<username>')
def user_index(username):
    return render_template('user_index.html', username=username)
```


然后创建 flask-test-app/templates 目录，向 flask-test-app/templates/user_index.html 页面中写入如下代码：

```
<h1>Hello, {{ username }}!</h1>
```

最终的目录结构变成这样：

```
/flask-test-app
  app.py
  /templates
    user_index.html
```

在 jinja2 中，用 {{ }} 来渲染一个字符串变量。这里的 username 就是在 render_template 的时候传入的关键字参数 username。使用 flask run 运行应用，现在访问一个用户主页，比如说：

```
localhost:5000/user/shiyanlou
```

就能看到用一个 h1 标签包裹的 Hello, shiyanlou! 了。

练习题：为 Flask 应用增加模板

首先使用 Ctrl + C 停止前面运行的 flask 应用，然后进入到 /home/shiyanlou/flasktest 中为已有的路由增加模板渲染：

```
$ cd /home/shiyanlou/flasktest
$ mkdir templates
```

在 flasktest/templates 目录下新建 courses.html 页面，并向其中写入如下代码：

```
<h1>Course: {{ coursename }}!</h1>
```

修改 /home/shiyanlou/flasktest/app.py 文件中的视图函数 courses()，使其返回 courses.html 页面。

最终实现效果为：当访问 http://localhost:5000/courses/linux 时，页面内容为：

```
Course: linux!
```

完成代码后，需要在终端按照之前的所学配置 FLASK_APP 环境变量并使用 flask run 启动应用。

注意本节实验后续的几个题目是连续的，所以 /home/shiyanlou/flasktest 目录创建后不要删除。

应用启动后，点击 下一步，系统将自动检测完成结果。

📍 楼+之Python实战第10期 (/courses/1190)

GET 与 POST

Flask 通过 `request` 对象获取请求相关的数据，要使用它，要从 flask 导入：

```
from flask import request
```

一般常用的两种请求方式就是：GET 请求和 POST 请求。其中：

- GET 请求: 只是从服务器上获取数据，不会对服务器上的数据产生影响（通常只是查询操作）。传参放在 URL 中，通过 `?` 的形式指定 key 和 value。GET 请求获取参数是通过 `request.args.get('key')` 来获取。
- POST 请求: 用于向指定的资源提交待被处理的数据，会对服务器上的数据产生影响（做增加、删除或是修改的操作）。传参不放在 URL 中，通过表单数据的形式发送给服务器。POST 请求获取参数是通过 `request.form.get('key')` 来获取。

GET 请求

从 `request.headers` 获取请求头的数据，可以把它当作字典来使用，比如要获取用户的 `user-agent`：

```
request.headers.get('User-Agent')
```

从 `request.args` 获取请求的参数，修改 `/home/shiyanlou/app.py` 文件中的 `user_index` 函数代码如下所示：

```
from flask import request

@app.route('/user/<username>')
def user_index(username):
    print('User-Agent:', request.headers.get('User-Agent')) # 打印请求头的数据
    print('time:', request.args.get('time')) # 获取请求 URL 中 ? 后面的 time 参数
    print('q:', request.args.get('q')) # 获取请求 URL 中 ? 后面的 q 参数
    print('Q:', request.args.getlist('Q')) # 当参数的值不止一个时使用 getlist 方法
    return 'Hello {}'.format(username)
```

运行 `flask run`，用下面这个 URL 访问用户页面：

```
http://127.0.0.1:5000/user/shixiaolou?time=1&q=2&Q=5&Q=6
```

可以看到页面显示 `Hello shixiaolou`，同时在终端中打印的内容如下所示：

```

$ curl -APost: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:48.0) Gecko/20100101 Firefox/48.0
time: 1
q: 2
Q: ['5', '6'] # 参数的值不止一个，返回值的列表

```

这样就在后端获取到了用户使用 GET 请求传递的参数。

POST 请求

一般而言，POST 请求是通过表单传递数据的，可以通过 `request.form` 获取表单数据，通过 `request.method` 获取当前请求的方法（GET 或 POST）。

为了解释说明这个过程，我们使用 Python 的 requests 库模拟浏览器发送 post 请求，首先需要安装：

```
sudo pip3 install requests
```

在 `/home/shiyanlou` 目录下新建 `client.py` 文件，向文件中写入需要发送的数据：

```

import requests

# 设置需要发送的数据
user_info = {'name': 'shixiaolou', 'password': 'abc123', 'hobbies': ['code', 'swim']}
# 向 URL 发送 post 请求
r = requests.post("http://127.0.0.1:5000/register", data=user_info)
# 打印返回文本
print(r.text)

```

向 `/home/shiyanlou/app.py` 文件中添加一个注册 register 的路由和视图函数：

```

@app.route('/register', methods=['GET', 'POST']) # 如果需要处理 POST 请求，就需要指定 methods
=['GET', 'POST']
def register():
    print('method:', request.method) # 查看本次请求的方式
    print('name:', request.form['name']) # 获取 name 值
    print('password:', request.form.get('password')) # 获取 password 值
    print('hobbies:', request.form.getlist('hobbies')) # 获取 hobbies 值，因为有多项使用 get
list 方法
    print('age:', request.form.get('age', default=18)) # 如果没有传递 age 值，可以在程序中设
置默认值
    return 'register succeeded!'

```

执行 `flask run` 运行程序，然后运行脚本发送请求：

```

$ python3 client.py
register succeeded!

```

查看终端输出，结果如下：

🔗 楼+之Python实战第10期 (/courses/1190)

```
method: POST
name: shixiaolou
password: abc123
hobbies: ['code', 'swim']
age: 18
```

练习题：判断请求方式并打印请求值

首先使用 `Ctrl + C` 停止前面运行的 flask 应用，然后进入到 `/home/shiyanlou/flasktest` 目录中。

新建一个文件 `client.py` 文件，向文件中写入需要发送的数据：

```
import requests

# get 请求
payload = {'t':'1', 'q':'2'}
r1 = requests.get('http://127.0.0.1:5000/httptest', params=payload)
print(r1.text)

# post 请求
user_info = {'Q':['5','6']}
r2 = requests.post("http://127.0.0.1:5000/httptest", data=user_info)
print(r2.text)
```

你需要完成的是在 `app.py` 文件中新加一个 `/httptest` 路由以及对应的 `httptest` 视图处理函数，在这个函数中需要完成如下需求：

- 判断请求方式是 GET 还是 POST
- 如果是 GET，分别打印请求参数对应的值，并返回 `It is a get request!`
- 如果是 POST，打印请求参数对应的值，并返回 `It is a post request!`

完成后，需要在终端按照之前的所学配置 `FLASK_APP` 环境变量并使用 `flask run` 启动应用，然后使用 `python3 client.py` 运行脚本文件。

运行 `client.py` 后结果如下所示：

```
It is a get request!
It is a post request!
```

此时终端打印结果如下：

🔗楼+之Python实战第10期 (/courses/1190)

q: 2

127.0.0.1 - - [07/Dec/2018 13:57:13] "GET /httptest?q=2&t=1 HTTP/1.1" 200 -

Q: ['5', '6']

127.0.0.1 - - [07/Dec/2018 13:57:13] "POST /httptest HTTP/1.1" 200 -

注意本节实验后续的几个题目是连续的，所以 `/home/shiyanlou/flasktest` 目录创建后不要删除。

应用启动后，点击 [下一步](#)，系统将自动检测完成结果。

session

HTTP 协议是无状态的，每一次请求它们的关系都是相互独立的。但是在实际的应用中，我们确实有很多数据需要服务器记住，但又不适合存放在数据库中。比如说，一个登录页面需要在用户密码错误输入 3 次后要求输入验证码，服务器就需要一个计数器纪录错误次数，但把它放到数据库也不太合适。session 就是用来为每个用户独立存放一些数据的地方。存放在 session 里的数据可以在特定用户的多个请求之间共享。因此 session 是在服务端保存的一个数据结构，用来追踪用户的状态。

session 的常见使用方法如下：

```
from flask import session

# 设置 session
@app.route('/set_session')
def set_session():
    session.permanent = True # 设置session的持久化
    app.permanent_session_lifetime = timedelta(minutes=5) # 设置session的存活时间为5分钟
    session['username'] = 'shixiaolou'
    return '成功设置session'

# 获取 session
@app.route('/get_session')
def get_session():
    value = session.get('username')
    return '获取的session值为{}'.format(value)

# 移除 session
@app.route('/del_session')
def del_session():
    value = session.pop('username')
    return '成功移除session, 其值为{}'.format(value)
```

参考链接：COOKIE和SESSION有什么区别? (<https://www.zhihu.com/question/19786827>)

cookies

cookies 与 session 类似，只不过 cookies 是以文本文件的形式存在于客户端的加密信息。在 Flask 中，cookie 使用配置的 SECRET_KEY 作为签名进行加密。

在 flask 中，cookies 设置在响应对象上，使用 make_response() 函数从视图函数的返回值中获取响应对象，使用响应对象的 set_cookie() 函数存储 cookie。

cookie 常见用法设置：

```
# 设置 cookie:
response.set_cookie(
    key,      # 键
    value,    # 值
    max_age,  # 以秒为单位的 cookie 存活时间
    expires,  # 失效时间，这是一个 datetime 的对象
    path,     # 存储的路径
)

# 获取 cookie:
request.cookies.get('key')

# 删除 cookie:
response.delete_cookie('key')
```

比如在访问用户主页的路由中，将用户名设置为一个 cookie，这样用户再次访问时，我们就能知道他是谁了，修改 /home/shiyanlou/app.py 文件中的 user_index 函数：

```
from flask import make_response

@app.route('/user/<username>')
def user_index(username):
    resp = make_response(render_template('user_index.html', username=username))
    resp.set_cookie('username', username)
    return resp
```

设置 cookies 后，用户访问其他页面可以从 request.cookies 获取到我们设置的 username，request.cookies 获取到的是一个由 cookie 变量作为键对应值作为值的字典：

```
from flask import request

@app.route('/')
def index():
    username = request.cookies.get('username')
    return 'Hello {}'.format(username)
```

然后启动 flask，首先访问主页 http://127.0.0.1:5000，可以看到页面显示 Hello None；然后访问用户页面 http://127.0.0.1:5000/user/shixiaolou，用户页面将显示 Hello,shixiaolou！；最后访问主页 http://127.0.0.1:5000，页面会显示 Hello shixiaolou。通过这样我们可以看出主页获取到了保存的 cookies。

练习题：设置 cookie 并获取

楼+之Python实战第10期 (7/courses/1190)

首先使用 `Ctrl + C` 停止前面运行的 `flask run` 命令，然后进入到 `/home/shiyanlou/flasktest` 目录，在这个目录下新建 `cookie.py` 文件：

```
cd /home/shiyanlou/flasktest
touch cookie.py
```

在 `/home/shiyanlou/flasktest/cookie.py` 中写入如下代码并按要求补充相应的代码：

```
from flask import Flask, render_template, request, make_response

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('cookie_index.html')

@app.route('/setcookie', methods = ['POST', 'GET'])
def setcookie():
    if request.method == 'POST':
        user = request.form['name']
        # TODO:
        # 1. 响应返回 readcookie.html 页面
        # 2. 设置一个 cookie 的键为 userID，值为 user

@app.route('/getcookie')
def getcookie():
    // TODO:
    // 1. 获取 userID 的 cookie，并赋值给 name
    return '<h1>welcome, '+name+'</h1>'
```

并在 `/home/shiyanlou/flasktest/templates` 目录下新建 `cookie_index.html`，并向其中写入如下代码：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Flask Cookies</title>
</head>
<body>
    <form action = "/setcookie" method = "POST">
        <p><h3>Enter userID</h3></p>
        <p><input type = 'text' name = 'name' /></p>
        <p><input type = 'submit' value = '提交' /></p>
    </form>
</body>
</html>
```

同时在 `/home/shiyanlou/flasktest/templates` 目录下新建 `readcookie.html`，并向其中写入如下代码：

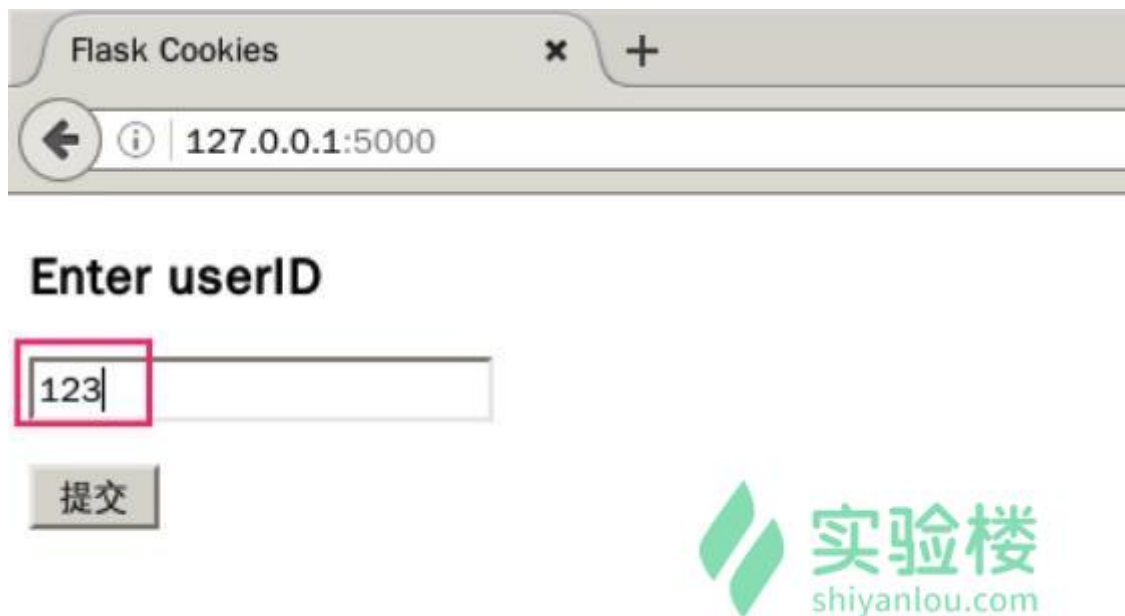
```
<h1> cookie is ok </h1>
```

完成代码后，启动应用：

```
export FLASK_APP=cookie.py
export FLASK_DEBUG=1
flask run
```

实现的效果如下：

当访问 `http://127.0.0.1:5000` 时会显示一个表单，在其中填写当前的用户ID并点击提交：



Enter userID

提交

实验楼
shiyanlou.com

然后链接会跳转到 `http://127.0.0.1:5000/setcookie`，页面会显示 `cookie is ok`：



cookie is ok

实验楼
shiyanlou.com

最后访问 `http://127.0.0.1:5000/getcookie` 就可以获取到之前设置的 cookie :

🔗 楼+之Python实战第10期 (/courses/1190)



注意本节实验后续的几个题目是连续的，所以 `/home/shiyanlou/flasktest` 目录创建后不要删除。

应用启动后，点击 下一步 ，系统将自动检测完成结果。

错误处理

使用 `app.errorhandler` 装饰器可以注册错误处理函数，比如对于最常见的 404 错误，我们返回一个特定的 `404.html` 页面。

```
@app.errorhandler(404)
def not_found(error):
    return render_template('404.html'), 404
```

例子中也展示了使用 `render_template` 的一个小知识点，就是可以在它后面指定本次返回的状态码。

在 flask 中还有一个经常用来处理错误的方法 `abort()`，使用 `abort(404)` 则能够直接进入到页面无法找到（HTTP 状态码404）的处理逻辑中。修改 `/home/shiyanlou/flask-test-app/app.py` 文件内容如下：

```
from flask import Flask, render_template, abort

app = Flask(__name__)

@app.route('/user/<username>')
def user_index(username):
    if username == 'invalid':
        abort(404)
    return render_template('user_index.html', username=username)
```

运行应用，当访问 `http://127.0.0.1:5000/user/haha` 时，可以看到页面显示 `Hello,haha!`，也就是页面能够正常访问。但是当 `username` 为 `invalid` 字符串的时候，即访问 `http://127.0.0.1:5000/user/invalid` 地址的时候，直接返回页面无法找到 `Not Found`。

插件

相比重型 Web 框架 Django，Flask 缺少很多重要的功能，因为 Flask 被设计为可扩展形式，开发者可以自主选择最适合的程序包来实现一些功能。这也是 Flask 晚于前者诞生却能够做到用户量持续高速增长的一个原因--它的设计模式允许用户自主选择很多重要的功能。

社区人员开发了很多不同用途的扩展，下面列出了一些 Flask 开发中常用的插件，这些插件大多在后面的项目中会用到，因为篇幅的原因不能一一介绍。每个插件都可以在 Github 搜到，大家可以先去了解一下。官方推荐的第三方插件都可以在 Flask Extensions (<http://flask.pocoo.org/extensions/>) 中找到。通常第三方插件都可以使用 `pip` 进行安装，一般而言都可以直接使用 `import 模块名` 的方式导入。

- flask-sqlalchemy：ORM，封装了 sqlalchemy，使用更简单
- flask-login：管理用户 session，如登入、登出，session 过期管理等等
- flask-migrate：数据库版本管理
- flask-wtf：封装了 wtforms 表单生成与验证工具，提供了 CSRF 支持
- flask-session：flask 默认 session 基于客户端 cookie 的，这个插件方便在服务端做 session

实际的项目开发中，Flask 会大量应用各种插件，例如和数据库对接使用 `flask_sqlalchemy`，管理登入登出 session 则使用 `flask-login`，这些插件的使用可以极大的提高我们开发 Web 应用的效率，所以当你有任何开发需求的时候，先去搜索是否已经有现成的模块可以给我们使用了，如果有的话就可以参考使用文档直接应用。

总结

本节实验通过一些简单的开发示例学习了 Flask Web 应用开发的基础，本节的内容中包括以下的知识点：

- Flask 简介
- 配置方法
- 路由和视图函数
- 模板渲染
- GET 与 POST
- session
- cookies
- 错误处理
- 插件

关于这些知识点的用法大家先有一个了解即可，因为这些知识点不能够死记硬背，只能在不断的使用过程中才能够得到有效的强化，更加实际的应用主要依靠后面的实战项目。

拓展阅读

- Flask 快速入门官方文档（中文）(<http://docs.jinkan.org/docs/flask/quickstart.html>)

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

上一节：第一周挑战解析与下周知识导学 (</courses/1190/labs/8531/document>)

下一节：HTML 和 CSS (</courses/1190/labs/8533/document>)