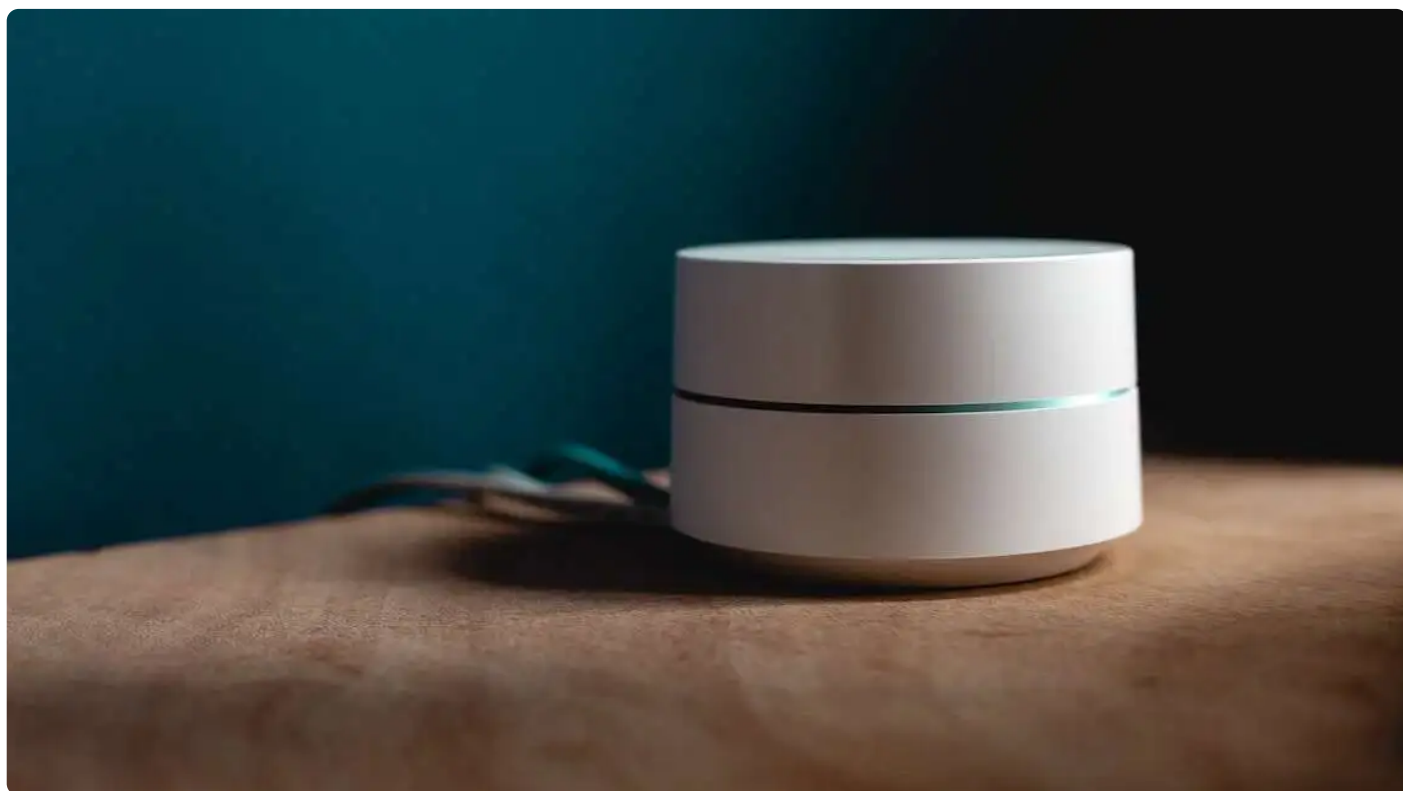


## 19 | 路由定义和视图常用逻辑：路由定义和视图的核心是什么？

2023-06-05 Barry 来自北京

《Python实战 · 从0到1搭建直播视频平台》



你好，我是 Barry。

上节课我们完成了 Flask 框架搭建，同时我们也提到了路由。虽然我们都知道路由是处理 URL 和视图函数之间关系的程序，但在项目中究竟为什么要使用路由，我们又该如何使用，这些问题我们仍然没有解决。

shikey.com转载分享

这节课，我们重点学习静态路由还有视图函数的常用逻辑，了解了这些，你就能能够编写和处理复杂的视图函数，掌握从路由请求中提取有用的信息，并能够生成动态响应做一些接口的开发。

shikey.com转载分享

### 初识路由

路由是 Flask 框架中非常重要的组成部分。为什么说它非常重要呢？因为路由的作用是将 HTTP 请求的 URL 路径映射到相应的函数处理程序。这样我们在开发过程中，就能将不同的

URL 路径与相应的函数处理程序关联起来，从而实现对 Web 应用的灵活控制。

路由可以分为静态路由和动态路由。两者主要是在形式上有一些区别，通常静态路由使用 `@app.route( '/' )` 这样的装饰器形式。

而动态路由会使用变量名 `<name>`，形式通常为 `@app.route( '/' , <name> )`。当然，其中还可以传入其他类型的参数，如整型、浮点型，甚至 Path 路径等。现在你先知道它们有什么区别就可以，动态路由这部分我会在下一节课重点去讲解，这节课我们重点学习静态路由。

## 静态路由详解


路由的装饰器形式是 `@app.route( '/' )`，装饰器中的第一个参数是 `'/'`，括号当中的斜杠表示项目根路径。

静态路由的工作机制很容易理解：在我们的项目开发过程中，都是在根路径的基础上去修改 URL，对于不同的 URL 需要使用装饰器的方式来绑定不同的视图函数。

不难看出静态路由里装饰器的重要性，我们这就来了解一下装饰器函数中的重要参数，它们分别是 `methods`、`endpoint`、`url_for`、`redirect_to`，掌握了这些参数能让你进一步加深对路由的了解。我们这就依次去看一下它们的具体用途。

## methods 参数

我们首先来看 `methods`。`methods` 是当前视图函数支持或者说函数指定的 HTTP 请求方法。这么说有点抽象，我们不妨通过后面这个案例来看看。

 复制代码

```
1 from flask import Flask, request
2 app = Flask(__name__)
3 @app.route('/message', methods=['POST'])
4     def message():
5         name = request.form['name']
6         location = request.form['location']
7         age = request.form['age']
8         return f'Hi, {name} ,you have live in {location} for {age} years!'
9 if __name__ == '__main__':
```

methods 参数主要用来指定函数的请求方法。在上面的案例中，路由装饰器的 methods 参数指定 message() 函数使用 POST 请求方法。在 message() 函数里的参数 name、location、age 使用 request 方法从表单中获取数据，message() 函数最后返回了表单中的名字、地址和年龄。

当用户在提交表单数据的时候，Flask 会自动调用视图函数 message()，并且从请求当中获取提交的数据。最后，我们将提交的数据返回并展示给用户。这就是 methods 核心作用。

## endpoint 函数

接下来我们再来了解一下 endpoint。

Flask 路由装饰器 @app.route 中的 endpoint 参数是一个函数，用于指定要处理的 URL 的名称。这个参数可以是一个字符串，用于匹配要处理的 URL，也可以是一个函数，用于生成 URL。

当我们使用 @app.route 装饰器时，它会自动添加 endpoint 参数。这个参数可以接受一个字符串或一个函数作为值。如果我们采用函数作为 endpoint 参数传递的形式，那么 Flask 将在路由处理函数内部调用该函数。

接下来我们看一个例子，直观感受一下。

shikey.com转载分享

复制代码

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

shikey.com转载分享

在上面的代码中，`/hello` 是路由的 URL，`hello_world` 是处理该请求的视图函数。当用户访问 `/hello` 路径时，Flask 将会调用 `hello_world` 函数来处理请求，并返回一个包含 “Hello, World!” 的响应。

`endpoint` 参数的作用是让 Flask 能够处理相同 URL 的请求。比方说，如果有多个处理相同 URL 的视图函数，但是它们使用不同的 `endpoint` 参数，Flask 就会根据传入的 `endpoint` 参数来选择正确的视图函数。

另外，如果在 Flask 应用程序中有多个路由，但是它们使用相同的 `endpoint` 参数，那么这些路由将共享同一个视图函数。这样可以避免在多个路由中编写相同的视图函数，从而提高代码的复用性和可维护性。

## url\_for 装饰器

`url_for` 是一个用于生成 URL 的装饰器，它可以帮助我们在模板中更方便地生成动态的 URL。该函数的参数是一个字符串或字典，用于指定要生成的 URL 的变量名。

`url_for` 装饰器可以在路由函数的 `path` 参数中使用，也可以在路由函数的 `url_for` 参数中使用。如果在 `path` 参数中使用，它将在生成的 URL 中使用变量名替换路径部分；如果在 `url_for` 参数中使用，它将生成一个新的 URL，其中变量名将作为路径部分。我们还是结合案例来感受一下。

 复制代码

```
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5 @app.route('/hello')
6 def hello():
7     name = 'World'
8     return render_template('hello.html', name=name)
9
10 if __name__ == '__main__':
11     url = url_for(hello)
12     print(url) # 输出: <http://localhost/hello?name=World>
```

在上面的示例中，`/hello` 路由的返回值是一个字符串 `'World'`，这个字符串将被替换为 URL 中的参数值，最终生成的 URL 是 `<http://localhost/hello?name=World>`。


需要注意的是，**`url_for` 装饰器只会生成一个唯一的 URL**，如果在同一个请求中多次使用相同的 `url_for` 装饰器，则会生成相同的 URL。如果需要避免重复的 URL，可以在模板中手动拼接 URL。

## redirect\_to

`redirect_to` 是重定向装饰器。当用户点击路由跳转链接时，浏览器会发送一个 HTTP 重定向请求，Flask 路由装饰器 `redirect_to` 可以捕获这个重定向请求，并将用户重定向到目标 URL。

应用 `redirect_to` 装饰器的重点，就是了解它需要接收一个参数，这个参数就是重定向后要跳转到的目标 URL。当用户点击路由跳转链接时，Flask 会自动发送一个 HTTP 重定向请求到目标 URL。装饰器 `redirect_to` 会拦截这个重定向请求，并且把用户重定向到指定的目标 URL。

我们一起通过后面这段代码来学习一下 `redirect_to` 具体的用法。

 复制代码

```
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5 @app.route('/hello')
6 def hello():
7     name = 'World'
8     return render_template('hello.html', name=name, redirect_to='/home')
9
10 if __name__ == '__main__':
11     url = '/hello?name=World'
12     response = app.run()
```

在上面的示例中，`hello` 视图函数使用 `render_template` 函数返回一个渲染后的模板，模板文件名为 `hello.html`。当用户点击路由跳转链接的时候，然后 Flask 会把用户重定向到

<http://localhost/home>。

到这里，我们已经清楚了路由装饰器中核心的配置项，这是 Flask 框架使用过程中的一个必备知识点，相信掌握之后会让你后面的开发实现更加流畅。

我们已经知道了静态路由会将 URL 路径与特定的函数或视图函数关联起来，以便在接收到相应的 HTTP 请求时调用相应的函数或视图函数来处理请求。那视图函数到底扮演着什么样的角色呢？这就是我们接下来要探索的内容。

## 视图函数

Flask 中的视图函数就是用于处理 HTTP 请求并返回响应的 Python 函数。视图函数通常接受 HTTP 方法（如 GET、POST、PUT、DELETE 等）和 URL 路径作为参数，并根据请求的不同和条件来执行不同的操作。

比方说，在处理 GET 请求时，视图函数可以生成 HTML 响应或从数据库中检索数据；而在处理 POST 请求时，视图函数可以将表单数据写入数据库或执行其他一些特定操作。

和视图函数紧密相关的函数和对象包括 `render_template` 函数和 `request` 对象，我们依次来看看。

## `render_template` 函数

`render_template` 函数主要的作用就是将视图函数中的数据传递给模板，并在模板中使用这些数据来生成 HTML 响应。

由此我们就理清了视图函数和 `render_template` 的关系：**视图函数负责处理请求并生成需要传递给模板的数据，然后使用 `render_template` 函数将数据传递给模板来渲染响应内容。**

在 `render_template` 函数内可以接受两个参数。第一个参数是 `template_name`，代表要渲染的模板文件名称。

第二个参数是 context，可以是一个字典，包含要在模板中使用的变量和数据。context 参数还可以是 Flask 视图函数的返回值，它可以被视图函数中的变量引用。

我们可以在用于生成 HTML 响应的 Jinja2 模板中，使用 {{ }} 语法来引用 context 中的变量。


你可以结合后面这个简单的 render\_template 例子来理解一下。

 复制代码

```
1 from flask import Flask, render_template
2 app = Flask(__name__)
3 @app.route('/')
4 def index():
5     return render_template('index.html')
6 if __name__ == '__main__':
7     app.run()
```

在这个例子中，视图函数返回的就是我们事先编写好的 index.html 网页。这里的渲染的模板 index.html 文件是之前就存在应用程序的 templates 文件夹当中的。


除了使用 render\_template 去渲染模板，我们还可以给模板中传入值，你不妨结合后面的代码案例来看看。

 复制代码

```
1 from flask import Flask, render_template
2 app = Flask(__name__)
3 @app.route('/')
4 def index():
5     data={
6         'name': '列表',
7         'len': '8',
8         'list': {1,2,3,4,5,6,7,8}
9     }
10    return render_template('index.html', data=data)
11 if __name__ == '__main__':
12    app.run()
```

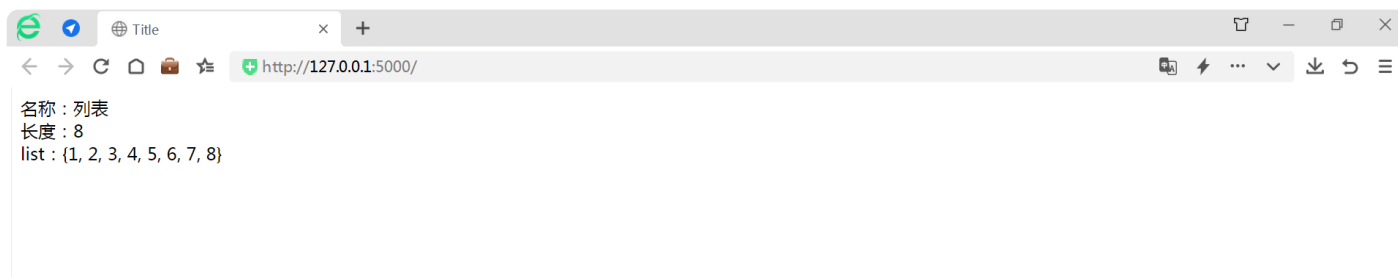


前面的 data 是 HTML 文件中的 data，后一个 data 是我们传入的 data。html 文件的代码我也放在了下面。

 复制代码

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8 名称: {{data['name']}}
9 <br>
10 长度: {{data['len']}}
11 <br>
12 list: {{data['list']}}
13 </body>
14 </html>
```

代码输出的效果是后面这样。



我们需要注意的是，这里传给 render\_template 函数中的 data 参数，**也必须在 HTML 文件中出现同样的形参 data，才能传入成功。**

其实这种传值的方法，还可以使用在我们的在线视频平台项目里。比如说，只要我们能获取到视频的点赞数、观看数和评论数等信息，就可以通过这种方法，把值通过函数 render\_template 传给视频首页，从而实现数据的更新。

以上我们是实现对 render\_template 函数的应用，除了完成页面的展示，在请求过程中我们还需要关注的信息就是请求返回信息。我们需要通过返回信息来做代码逻辑的处理，例如请求



的 URL、HTTP 方法、请求头、请求体等，这就要用到我们下面即将学到的 request 对象了。

## request 对象

Flask 视图函数中的 request 是一个非常有用的对象，它包含了客户端与服务器之间的所有请求和响应数据。

在 Flask 中，视图函数的 request 对象通常是由客户端发送给服务器的第一个请求对象，它包含了客户端发送给服务器的所有数据，比如后面这些信息。

1. 请求头信息：如 Accept、Content-Type 等
2. 请求的 URL 参数 request.url
3. 发送给服务器的请求体数据 request.json
4. 请求方法 request.method
5. 表单数据，即 POST 请求中传递的数据 request.form

在视图函数中，我们可以通过 request 对象来获取这些数据，并对它们进行处理。接下来，我们结合后面简单的案例来学习一下 request 要如何使用。

shikey.com转载分享

shikey.com转载分享

```
1 from flask import Flask, request
2 app = Flask(__name__)
3
4 @app.route('/')
5 def index():
6     url = request.url
7     return 'Hello, World!'
8
9 if __name__ == '__main__':
10     response = app.run()
11     print(response.url) # 输出: <http://localhost:8080/>
12
```

在上面的示例中，我们定义了一个名为 `index` 的视图函数，它返回了一个字符串 “Hello, World!”。在视图函数中，我们通过 `request.url` 获取了请求的 URL，并将其返回给客户端。

除了获取请求 URL，我们还可以通过 `request.method`、`request.path` 和 `request.data` 等属性来获取更多的请求信息。这些属性可以帮助我们更方便地处理和操作客户端发送给服务器的数据。

接下来，我们来了解一下 `request` 的两个核心的属性 `request.form` 和 `request.method`。

### **request.form 属性应用**


我们先来了解 `request.form` 属性，它是一个字典类型对象，可以用来获取表单数据。具体来说，这些数据包含在客户端向服务器发送的 POST 请求里，这就是它的使用场景。

此时需要注意，如果前端传回来的数据格式不是表单，此时我们就不能用 `request.form`。比如 JSON 或者其他格式，那么就要用 `request.data` 来获取。无论是 `request.form` 还是 `request.data`，都是需属于 `request` 对象本身的参数。

但是特殊情况下，如果 request.form 这个字典对象中没有某个我们想获取的键对值，就会引发 keyerror 这个异常。此时我们就通过 request.form.get() 来更安全地获取 request.form 这个字典当中的值。

request.form.get 是获取 request.form 当中特定字段的一个方法，并且我们还可以设置一个默认值，防止如果这个字典当中不存在这个值引起意外错误，这也是你应用过程中需要注意的地方。

比方说，我们想获取前端用户输入的手机号、用户名等信息，就可以这么操作。

 复制代码

```
1 from flask import Flask, request
2 app = Flask(__name__)
3 @app.route('/login', methods=['POST'])
4 def login():
5     name = request.form.get('name', '')
6     mobile = request.form.get('mobile', '')
7     # 这里可以进行用户名和手机号信息验证等其他操作
8     return name + ' ' + mobile
9 if __name__ == '__main__':
10     app.run()
```

从前面这段代码可以看到，request.form 中包含我们的用户和密码两个字段。我们让参数 name 从 request.form 中获取 name 字段对应的值，mobile 从 request.form 中获取 mobile 字段对应的值，并且我们把默认值设置成了空字符串，这样可以避免 keyerror 的异常。如果表单当中没有对应的键对值，我们就会返回默认值，也就是我们这里的空字符串。最后再返回用户的信息。

## request.method 属性

其次就是 request.method，它代表了客户端发起的 HTTP 请求的方法。当然了，我们也可以根据 method 请求方式不同，返回不同的响应界面。

我们把之前讲 request.form 时使用的代码例子稍作修改，就变成了后面这样。

```
1 from flask import Flask, request
2 app = Flask(__name__)
3 @app.route('/', methods=['GET', 'POST'])
4 def login():
5     if request.method == "GET":
6         return render_template('login.html')
7     name = request.form.get('name', '')
8     pwd = request.form.get('pwd', '')
9     return name + ' ' + pwd
10 if __name__ == '__main__':
11     app.run()
```

在这个例子中，路由允许 GET 和 POST 两种方式，但我们限定只能使用 POST 方式提交用户信息。如果用了 GET 方式，则会渲染 login.html，效果就是提示用户在 login.html 页面中提交信息，然后自动返回登录界面。

如果这个 request.method 是 GET 方法，那我们就显示“这是 GET 请求方法”。如果 request.method 是 POST 方法，就表示它会通过前端向服务器请求数据，这时候需要我们在对应的 HTML 文件当中设置这个 POST。这部分内容我们前端实战篇也提到过，你可以再复习巩固一下。

## 案例实战

前面我们已经掌握了很多的储备知识，包括静态路由和视图函数。

接下来我们来把前面所学综合应用一下，一起实现“哨兵检查口令”这个案例。

我们先在 Practice 文件下新建 cipher\_check.py 文件，再新建一个 practice\_2\_index.html 文件，这两个文件你也可以自己命名。我也给你把文件创建的结构截图贴在了后面，供你参考。

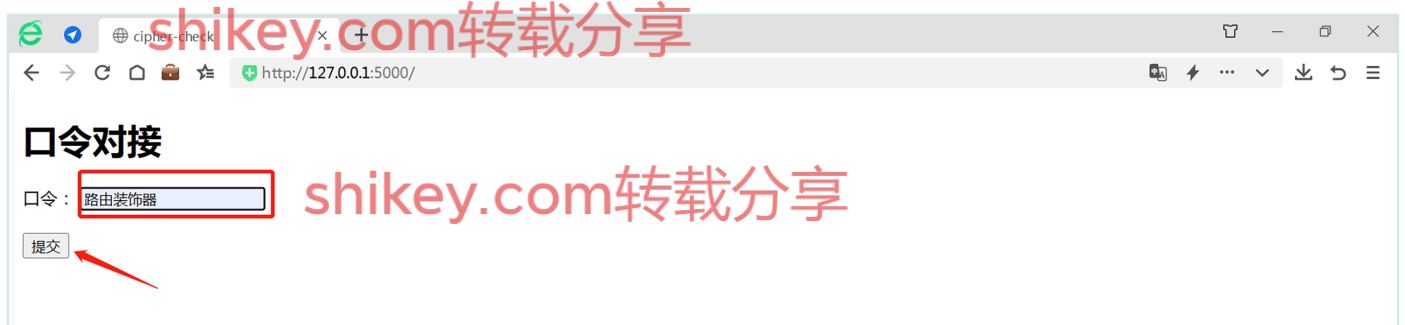


你可以参考后面这段代码，这是我们在 HTML 中需要实现的内容，主要是建立一个表单，并且在里面设置一个带有口令输入的输入框。


复制代码

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>cipher-check</title>
6 </head>
7 <body>
8 <h1>口令对接</h1>
9 <form action="/check" method="post">
10 <label for="cipher">口令: </label>
11 <input type="text" id="cipher" name="cipher"><br><br>
12 <input type="submit" value="提交">
13 </form>
14 </body>
15 </html>
```

我们运行前面这段代码，具体页面展示的效果就是后面这样，你可以参考一下。



在 cipher\_check.py 具体的代码实现，你同样可以参考后面我写的版本，你可以边看文稿边听我讲解。

 复制代码

```
1 from flask import Flask, render_template, request, redirect, url_for
2 app = Flask(__name__)
3 @app.route('/')
4 def index():
5     return render_template('practice_2_index.html')
6 @app.route('/<code>', methods=['GET'])
7 def check_(code):
8     Cipher = '路由装饰器'
9     if request.method == 'GET' and code == Cipher:
10         return '使用GET方法，口令正确'
11     else:
12         return redirect(url_for('failure'))
13 @app.route('/check', methods=['POST'])
14 def check():
15     Cipher = '路由装饰器'
16     cipher = request.form.get('cipher')
17     if request.method == 'POST' and cipher == Cipher:
18         return redirect(url_for('success'))
19     else:
20         return redirect(url_for('failure'))
21 @app.route('/success')
22 def success():
23     return '暗号对接成功！'
24 def failure():
25     return '暗号对接失败！'
26 if __name__ == '__main__':
27     app.run()
```

shikey.com转载分享

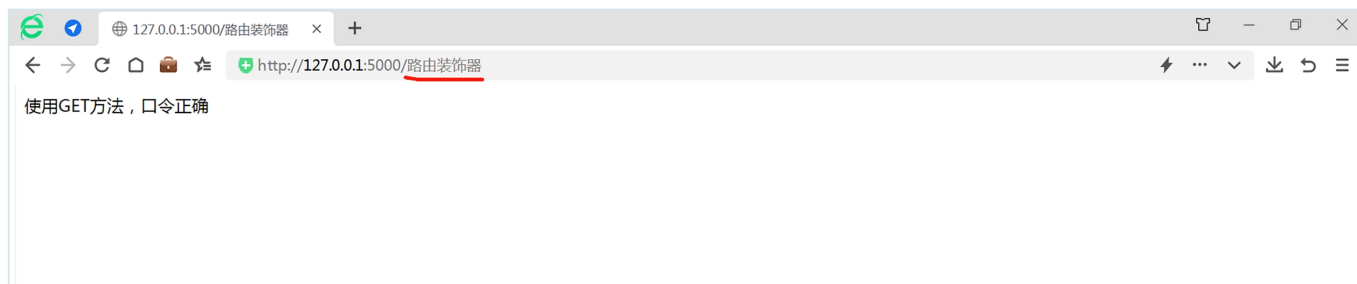
不难发现这段代码中，index() 视图函数渲染了用来给用户输入口令的模板

practice\_2\_index.html。输入口令可以用两种方式来实现，一种在 URL 中直接输入口令，一种通过表单提交。

shikey.com转载分享

我们需要提前把口令 Cipher 的参数设定为“路由装饰器”，同时在口令检查函数 check\_() 内，我们要指定采用 GET 请求方式。如果用户输入内容正确，就直接返回“使用 GET 方法，口令正确”的字样；如果输入错误，则重定向到 failure() 视图函数，提示用户暗号对接失败。

如果口令检查函数 `check()` 指定 POST 请求方式。这时候就会从表单中获取用户提交口令，口令检查对接上后，重定向到 `success()` 视图函数，提示用户“暗号对接成功”。我们一起来看看 GET 方式的运行结果。



极客时间

最后，我们来看看 POST 方式运行后的结果，后面截图里呈现的就是对接成功之后的效果。



极客时间

好，我们成功完成了“哨兵检查口令”这个有趣的案例，通过实践巩固了我们前面学到的路由和视图函数知识。在课后你也要自己尝试一下，这样学习效果会更好。

## 总结

又到了课程的尾声，我们一起来回顾总结一下这节课的内容。

路由在 Flask 框架中非常重要，它能够将 URL 路径与应用程序中的函数关联起来，从而实现对 HTTP 请求的处理。路由又分为静态路由和动态路由，两者主要是在形式上有一些区别，这节课我们重点学习了静态路由。

关于静态路由，你重点要掌握后面这四点。

1.methods 参数，它用于设定当前视图函数支持或者函数指定的 HTTP 请求方法。



2.endpoint 函数，它用于指定要处理的 URL 的名称。endpoint 可以是一个字符串，用于匹配要处理的 URL，也可以是一个函数，用于生成 URL。

3.url\_for 是一个用于生成 URL 的装饰器，它可以帮助我们在模板中更方便地生成动态的 URL。

4. 捕获重定向请求的 redirect\_to，并将用户重定向到目标 URL。

我们一定要明确每个参数的作用，这样我们在实战中更顺畅。

除了静态路由，我们还学习了视图函数，它是用于处理 HTTP 请求并返回响应的 Python 函数。

其中你需要重点掌握 **render\_template 这个函数，它会根据模板文件中的变量和条件动态地生成 HTML 代码，并将其返回给客户端。**以后做视图的时候，别忘了 render\_template 这种实现方式。render\_template 函数可以接受两个参数，分别是 template\_name 和 context。

最后，我们还要关注请求返回信息 request 对象，request 通常是由客户端发送给服务器的第一个请求对象，它包含了客户端发送给服务器的所有数据。**request.form 和 request.method 的用法也是重要知识点**，开发里经常会用到，只有扎实地掌握这两个属性，才能提高你的代码实战能力。

希望你在课后多多实践，巩固今天学到的内容。下节课我们继续学习动态路由，敬请期待。

## 思考题

这节课我们简单讨论过动态路由是什么。那么在路由定义中，为什么要使用动态路由参数？

欢迎你在留言区和我交流互动，如果这节课对你有启发，记得分享给身边的同事、朋友，和他一起交流进步。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (2)



曾泽浩

2023-06-15 来自广东

endpoint这个感觉确实不是很有必要，@app.route下面跟着就是处理函数，一个路由对应一个函数，如果想多个路由复用同一个函数，抽象出来一个方法复用就可以了，一对一的映射比较清晰点，而且通常多个路由都是同一个处理函数的场景比较少吧

作者回复：方法复用确实是一个不错的做法，这样可以将一些公共逻辑提取出来，避免代码冗余，这个在使用上还是有选择性的，非必需使用。当然它也有优势的地方，例如endpoint可以用于在蓝图中管理路由，这个我们后边课有讲。我们可以使用endpoint将路由与蓝图关联起来，这样会让路由的管理和维护更加方便和可复用，也是一种优化的选择。

共 2 条评论 >



peter

2023-06-05 来自北京

请教老师几个问题：

Q1：怎么判断endpoint是字符串还是函数？

文中的例子，函数是def hello\_world()。endpoint='hello\_world'，这里的hello\_world是字符串还是函数？

Q2：“如果在 Flask 应用程序中有多个路由，但是它们使用相同的 endpoint 参数，那么这些路由将共享同一个视图函数”，怎么理解？能否举例说明。

路由装饰器都是加在函数名称前面的，多个路由就是多个函数啊。

Q3：request的数据只有表单和json两种吗？如果是其他种类会怎么处理？

Q4：render\_template函数中的第一个参数，就是文件名，这个文件必须在目录templates下面吗？

Q5：url\_for的例子代码是怎么执行的？

我的理解：程序运行后，1 最下面的“if \_\_name\_\_ == '\_\_main\_\_':”会执行，即执行“url = url\_for(hello)”和“print(url)”；

2 “def hello():”这个函数不会执行，只有用户通过浏览器发请求的时候才执行。

是这样吗？

Q6：url\_for(hello)中的“hello”是指“def hello():”中的hello吗？还是指@app.route('/hello')中的“hello”？

“/hello 路由的返回值是一个字符串 ‘World’”，为什么会返回字符串world？ /hello对应函数hello，而函数hello返回的是一个网页啊（render\_template）

作者回复: 1、hello\_world 是一个字符串, 它被用作端点名称。端点是一个标识符, 用于在 Flask 应用中引用一组视图函数。在这个例子中, hello\_world 端点只包含一个视图函数 hello\_world。

2、当在 Flask 应用程序中有多个路由使用相同的 endpoint 参数时, 它们将共享同一个视图函数。这意味着, 无论你通过哪个路由访问该端点, 都将调用相同的视图函数来处理请求。是这样的关系。下面我给你写个案例, 你看案例就能看清他们的关系:

```
@app.route('/')
@app.route('/home')
@app.route('/index', endpoint='home')
def home():
    return render_template('home.html')
```

3、在HTTP请求中, 数据可以以多种格式进行传输, 包括表单数据、JSON数据、XML数据等。在Flask中, 可以通过请求对象的数据属性来获取请求的数据。

对于其他格式的数据, 例如XML数据, 则可以通过Flask提供的request.get\_data()方法来获取整个请求体数据, 然后使用相应的库来解析XML数据。这个根据类型去选择就可以

4、是的, Flask默认会将模板文件放在templates目录下。当使用render\_template函数时, 需要传入文件名作为第一个参数, 如果该文件在templates目录下, 可以只传入文件名; 如果文件不在templates目录下, 需要指定文件的完整路径

5、1、是的, 理解是正确的

2、在 if \_\_name\_\_ == '\_\_main\_\_': 代码块中, 调用 Flask 提供的 url\_for() 函数, 传入 hello() 函数作为参数。这将返回 /hello 路由对应的 URL。然后, 将 URL 打印到控制台。这个对待吗问题统一解释, 你就明白核心逻辑了。继续加油, 多多实践。

因此, 这段代码会在以下情况下执行:

当你运行这个 Flask 应用程序时, 它将执行 if \_\_name\_\_ == '\_\_main\_\_': 代码块中的代码。

当用户访问 /hello 路由时, Flask 将调用 hello() 函数, 执行其中的代码。

 [shikey.com](https://shikey.com)转载分享 

[shikey.com](https://shikey.com)转载分享