

路由就是根据请求的 `url` 找到对应处理的函数视图的过程。

在请求之前应该建立好一张路由表保存`url`与视图的对应关系，这样有请求过来才能正确找到对应的视图。

Flask中有两种常用方式构建路由规则：

1、`@app.route('url规则')` decorator装饰器方式

2、`app.add_url_rule()`

`add_url_rule(self, rule, endpoint=None, view_func=None, **options)`参数的含义如下：

`rule`: url 规则字符串，可以是静态的 /`path`, 也可以包含 /

`endpoint`: 要注册规则的 `endpoint`(站点)，默认是 `view_func` 的名字

`view_func`: 对应 `url` 的处理函数，也被称为视图函数

`endpoint`这是路线的名称；您将在 `url_for()` 函数 中使用的名称。端点名称是 `View` 的注册键，这是一个符号名称，您可以通过它引用来自应用程序其他部分的路由。

The screenshot shows the PyCharm IDE interface. On the left, the project structure is visible with files like config.cfg, demo.py, and static/templates. The demo.py file is open, showing code for a Flask application. A red box highlights the line `print(url_for('index'))`. A yellow arrow points from this line to the terminal window at the bottom, which displays the output of running the application. The terminal shows the application is running on port 5000, and a URL is provided for access. A circled slash character '/' is also shown in the terminal output.

```
# 装饰器将路由映射到视图index
@app.route('/')
# 定义一个视图
def index():
    return 'ok'

#这两种方法是等价的
#app.add_url_rule('/', 'index', index)
```

```
# 装饰器将路由映射到视图index
@app.route('/')
# 定义一个视图
def index():
    return 'ok'

#这两种方法是等价的
#app.add_url_rule('/', 'index', index)
```

```
def hello_world():
    return 'hello world'
app.add_url_rule('/', 'hello', hello_world)
```

```
# 装饰器将路由映射到视图index
# @app.route('/')
def index():
    # print(app.config.get('A'))
    print(current_app.config.get('B')) # from flask import current_app
    return "ok"
app.add_url_rule('/', 'index', index)
```

1.查看路由信息

在django中url统一配置在URLconf配置文件中，但是Flask直接配置在视图没有统一的配置文件如何查看路由信息呢？

--- app.url_map查看所有路由

```
# -*- encoding: utf-8 -*-

# coding=utf-8
# 导入Flask类
from flask import Flask, current_app

# Flask 接收一个参数 name ,
# 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
app = Flask(__name__)

# 装饰器将路由映射到视图index
@app.route('/')
# 定义一个视图
def index():
    print(app.url_map)
    return 'ok'

if __name__ == '__main__':
    # Flask 应用程序实例的方法run启动web服务器
    app.run(debug=True)
```

The screenshot shows the PyCharm IDE interface. On the left, the project structure is visible with files demo.py, demo2.py, and config.cfg. The demo2.py file is open in the editor, containing the following code:

```

13     # Flask 接收一个参数 name ,
14     # 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
15     app = Flask(__name__)
16
17
18     # 装饰器将路由映射到视图index
19     @app.route('/')
20     # 定义一个视图
21     def index():
22         print(app.url_map)
23         return 'ok'
24
25
26     if __name__ == '__main__':
27

```

A yellow box highlights the line `print(app.url_map)`.

In the bottom terminal window, the application is running and the URL map is printed:

```

Run: demo2 x PRESS CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 109-510-767
127.0.0.1 - - [21/Oct/2022 20:07:32] "GET / HTTP/1.1" 200 -
Map([<Rule '/static/<filename>' (HEAD, OPTIONS, GET) -> static>,
    <Rule '/' (HEAD, OPTIONS, GET) -> index>])
127.0.0.1 - - [21/Oct/2022 20:07:32] "GET /favicon.ico HTTP/1.1" 404 -

```

Map([<Rule '/static/<filename>' (HEAD, OPTIONS, GET) -> static>,
 <Rule '/' (HEAD, OPTIONS, GET) -> index>])

--- url 规则
 --- 支持的请求方式, 默认支持get请求, 不支持post.
 --- 对应的视图

可以在python shell中导入flask项目查看：

The screenshot shows a Python shell session in the PyCharm IDE. The user has imported the Flask module and accessed the `url_map` attribute of the application object:

```

Python Console x
>>> import sys; print('Python %s on %s' % (sys.version, sys.platform))
>>> sys.path.append(['D:\\pythonfile\\pythonProject', 'D:/pythonfile/pythonProject'])
>>> 
+ Ⓜ PyDev console: starting.

Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit (AMD64)]
>>> from demo2 import app
>>> app.url_map
Map([<Rule '/static/<filename>' (HEAD, GET, OPTIONS) -> static>,
    <Rule '/' (HEAD, GET, OPTIONS) -> index>])

>>>

```

2. 同一路由装饰不同视图

同一个路由规则装饰不同函数，只有第一个视图函数会匹配到。

虽然都生成了路由表，但是url匹配中第一个规则之后就会调用视图，不再继续往下匹配。

```
# -*- encoding: utf-8 -*-
# 导入Flask类
from flask import Flask, current_app

# Flask 接收一个参数 name ,
# 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
app = Flask(__name__)

# 同一个url规则装饰不同的视图函数
@app.route('/index')
def index1():
    return 'index1'

@app.route('/index')
def index2():
    return 'index2'

if __name__ == '__main__':
    # Flask 应用程序实例的方法run启动web服务器
    app.run(debug=True)
```

The screenshot shows the PyCharm IDE interface. In the top navigation bar, there are tabs for 'demo.py' and 'demo2.py'. The 'demo2.py' tab is active, displaying the following Python code:

```
# 导入模块的目录，flask以这个目录为基础，寻找静态文件目录static和模板目录templates
app = Flask(__name__)

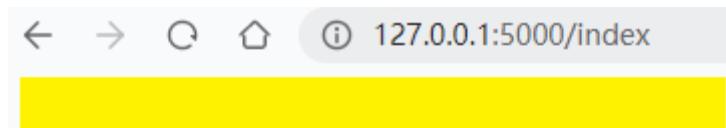
# 同一个url规则装饰不同的视图函数
@app.route('/index')
def index1():
    return 'index1'

@app.route('/index')
def index2():
    return 'index2'

index1()
```

In the bottom right corner of the code editor, there is a language switcher with '英' (English) selected. Below the code editor, the 'Run' tool window is open, showing the application is running on 'http://127.0.0.1:5000'. It also displays the command to quit ('Press CTRL+C to quit') and the message '* Restarting with stat'. A yellow box highlights the output of the application's run log, which shows the mapping of URLs to functions:

```
Map([<Rule '/static/<filename>' (HEAD, GET, OPTIONS) -> static>,
     <Rule '/index' (HEAD, GET, OPTIONS) -> index1>,
     <Rule '/index' (HEAD, GET, OPTIONS) -> index2>])
* Debugger is active!
* Debugger PIN: 109-510-767
```



index1

3.一个视图函数多个路由装饰器

同一个视图函数有多个路由装饰器，会生成多条路由信息，每条对应规则的[url](#)都可以访问到视图函数。

```
# 同一个url规则装饰不同的视图函数
@app.route('/index')
@app.route('/index1')
def index1():
    print(app.url_map)
    return 'index1'
```

```
4
5
6     # Flask 接收一个参数 name ,
7     # 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
8     app = Flask(__name__)
9
10
11    @app.route('/index')
12    @app.route('/index1')
13    def index1():
14        print(app.url_map)
15        return 'index1'
16
17
18    print(app.url_map)
index1
```

Run: demo2.x
Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
Map([<Rule '/static/<filename>' (OPTIONS, HEAD, GET) -> static>,
<Rule '/index1' (OPTIONS, HEAD, GET) -> index1>, <Rule '/index' (OPTIONS, HEAD, GET) -> index1>])
* Debugger is active!
* Debugger PIN: 109-510-767

4.methods 参数

HTTP (与 Web 应用会话的协议) 有许多不同的访问 URL 方法。

默认情况下, 路由只回应 **GET** 请求, 但是通过**route()** 装饰器传递 **methods** 参数可以改变这个行为**methods** 参数接收一个字典, 元素为字符串形式的请求方式名称。

如果不传**methods**参数, 默认支持 **GET**, **HEAD**, **OPTIONS**

OPTIONS 给客户端提供一个敏捷的途径来弄清这个 URL 支持哪些 HTTP 方法。从 **Flask 0.6** 开始, 实现了自动处理。

HEAD 浏览器告诉服务器: 欲获取信息, 但是只关心 消息头 。

应用应像处理 **GET** 请求一样来处理它, 但是不分发实际内容(不返回实际内容)。

在 **Flask** 中你完全无需 人工 干预, 底层的 **Werkzeug** 库已经替你打点好了。

```
@app.route('/login', methods=['GET','POST'])
def login():
    if request.method == 'POST':
        do_the_login()
    else:
        show_the_login_form()
```

5.反向解析

`url_for` 可以通过视图函数名称，反向解析得到视图对应的url.

视图函数名称其实就是我们前面`app.add_url_rule`设置路由详解的`endpoint`站点参数

The screenshot shows a PyCharm IDE interface with the following details:

Project Structure: Shows a project named "pythonProject D:\python" containing files: static, templates, config.cfg, demo.py, and demo2.py. demo2.py is the active file.

Code Editor (demo2.py):

```
1 # -*- encoding: utf-8 -*-
2 # 导入Flask类
3 from flask import Flask, current_app, url_for
4
5 # Flask 接收一个参数 name ,
6 # 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
7 app = Flask(__name__)
8
9
10 @app.route('/index')
11 @app.route('/index1')
12 def index1():
13     print(url_for('index1'))
14     return 'index1'
15
16
17 if __name__ == '__main__':

```

Run Output:

```
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 109-510-767
127.0.0.1 - - [21/Oct/2022 20:49:34] "GET /index HTTP/1.1" 200 -
/index1
```

In the code editor, the line `print(url_for('index1'))` is highlighted with a red box. In the run output, the line `/index1` is also highlighted with a red box.

The screenshot shows the PyCharm IDE interface. On the left is the project structure for 'pythonProject' containing files like 'demo.py', 'demo2.py', and 'config.cfg'. The 'demo2.py' file is open in the editor, showing Flask code. A red box highlights the route definition for '/user' and another red box highlights the print statement in the 'index1()' function. The terminal window at the bottom shows the application starting and a log entry for a GET request to '/index'.

```
4
5     # Flask 接收一个参数 name ,
6     # 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
7     app = Flask(__name__)
8
9     @app.route('/user')
10    def user():
11        return 'user'
12
13     @app.route('/index')
14     @app.route('/index1')
15     def index1():
16         print(url_for('user', name='张三'))
17         return 'index1'
18
19
20 if __name__ == '__main__':
21     user()
```

Run: demo2

```
* Restarting with stat
* Debugger is active!
* Debugger PIN: 109-510-767
/usr?name=%E5%BC%A0%E4%B8%89
127.0.0.1 - - [21/Oct/2022 20:55:15] "GET /index HTTP/1.1" 200 -
```

6. 动态路由

要给 URL 添加变量部分，把这些特殊的字段标记为 `<name>`，这个部分将会作为命名参数传递到你的函数。

变量放在`<>`中

```
@app.route('/param/<name>')
def get_url_param(name):
    return '参数是: %s' % name
```

```
pythonProject D:\python
  static
  templates
  config.cfg
  demo.py
  demo2.py
  External Libraries
  Scratches and Consoles

4
5  # Flask 接收一个参数 name ,
6  # 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
7  app = Flask(__name__)
8
9
10 @app.route('/param/<name>')
11 def get_url_param(name):
12     return '参数是: %s' % name
13
14
15 if __name__ == '__main__':
16
17     # Flask 应用程序实例的方法run启动web服务器
18     app.run(debug=True)
19

get_url_param

Run: demo2 x
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 109-510-767
127.0.0.1 - - [21/Oct/2022 20:57:22] "GET /param/张三" HTTP/1.1" 200 -
```



参数是: 张三

规则可以用 `<converter:variable_name>` 指定一个可选的转换器。

转换器: 默认匹配的是不带/的字符串
`int`: 接受整数
`float`: 接受浮点数
`path`: 和默认的相似, 但也接受斜线,

```
# 默认<>的规则匹配不带/的整数
@app.route('/param_int/<int:id>')
def get_url_param_int(id):
    return '获取的参数是: %s' % id
# 默认<>的规则匹配不带/的 浮点数
@app.route('/param_float/<float:f>')
def get_url_param_float(f):
    return '获取的参数是: %s' % f
# 匹配参数后面带/
@app.route('/param_path/<path:p>')
```

```
def get_url_param_path(p):
    return '获取的参数是: %s' % p
```

7. 自定义正则转换器

Flask路由转换器，没有提供基于正则的，但是我们可以自定义基于正则的路由转换器。

1. 自定义转换器必须继承BaseConverter类，自定义转换器需要重写父类的init方法

2. 注册转换器，url_map中保存了所有的路由转换器，是字典类型

```
# 导入Flask类
from flask import Flask, current_app, url_for
from werkzeug.routing import BaseConverter

# Flask 接收一个参数 name ,
# 导入模块的目录, flask以这个目录为基础, 寻找静态文件目录static和模板目录templates
app = Flask(__name__)

# 正则转换器
class RegexConverter(BaseConverter):
    def __init__(self, url_map, *args):
        # 调用父类的初始化方法
        super(RegexConverter, self).__init__(url_map)
        # 将正则表达式传给转换器对象, flask在解析路径的时候, 会来这里获取regex保存的正则表达式
        self.regex = args[0]

    def to_python(self, value):
        # 对获取到的参数进行处理
        # 默认获取到的是字符串, 可以对获取到的参数进行处理, 比如类型转换
        # 这样就可以在视图中直接使用
        print(type(value))
        print(value)
        return value

    def to_url(self, value):
        # 当使用反解析的时候会调用这个方法, 可以对参数进行处理
        print(value)
        return value

# 注册re 转换器 RegexConverter
app.url_map.converters['re'] = RegexConverter

@app.route("/param_re/<re('\d'):num>/<re('\d+'):num2>")
```

```

def get_param_re(num, num2):
    """url中提取参数"""
    return '自定义正则转换器获取参数1: %s , 参数2: %s' % (num, num2)

@app.route("/param_re/")
def get_param(num, num2):
    """url中提取参数"""
    return '自定义正则转换器获取参数1: %s , 参数2: %s' % (num, num2)

@app.route('/get_param_re_url/')
def get_param_url():
    """在视图函数中获取url"""
    print(url_for('get_param', num='1', num2='2'))
    print(url_for('get_param_re', num='1', num2='2'))
    return '<a href="%s">to_url演示</a>' % (url_for('get_param_re', num='1', num2='2'))

if __name__ == '__main__':
    # Flask 应用程序实例的方法run启动web服务器
    app.run(debug=True)

```

The screenshot shows the PyCharm IDE interface. In the top navigation bar, there are tabs for 'demo.py', 'demo2.py', and 'config.cfg'. Below the tabs, the code editor displays Python code for a Flask application. A red box highlights the section of code where the URL is printed:

```

print(url_for('get_param', num='1', num2='2'))
print(url_for('get_param_re', num='1', num2='2'))

```

In the bottom right corner of the code editor, there is a small preview window showing the resulting HTML output:

```

<a href="/param_re/1/2">to_url演示</a>

```

Below the code editor is the terminal window, titled 'Run: demo2'. It shows the application's startup logs and a successful HTTP request:

```

* Restarting with stat
* Debugger is active!
* Debugger PIN: 109-510-767
127.0.0.1 - - [21/Oct/2022 21:20:39] "GET /get_param_re_url/ HTTP/1.1" 200 -
/param_re/?num=1&num2=2
1
2
/param_re/1/2
1
2

```