这不是魔法：Flask和@app.route（2）

2015/01/02 · 工具与框架, 初级 · 8 评论 · Flask

本文由 伯乐在线 - 木羊同学 翻译，艾凌风 校稿。未经许可，禁止转载！
英文出处：aims.io。欢迎加入翻译组。

在我上一篇文章，我用了一个框架，模板了Flask网站上「@app.route(」了「第一条例子的行为。

如果你错过了那篇「这不是魔法」，请点击这里。

在这篇文章中，我们又要稍微调高点难度，为我们的URL加入可变参数的能力，在本文的最后，我们将支持下代码段将那里达到的行为。

```
app = Flask(__name__)
@app.route("/hello/<username>")
def hello_user(username):
    return "Hello {}!".format(username)
```

这样下面的路径实例（path）：
/hello/alex
将会匹配上面的路径，结构们的输出为
Hello alex!

以正则的形式表达我们的路径。

现在我们将允许我们的URL成为变化，我们不再能够将用先前使用「@app.route(」」注册的路径直接与路径实例匹配。

我们将用什么替代？我们需要用上正则表达式，这样我们就可以将路径作为一种模式进行匹配，而不是一条固定的字符串比较了。

我不打算在本文展开讲论正则表达式的细节，不过如果你需要一些复习资料，可以点击这个网站。

那么，我们的第一步是将我们的路径转化成正则表达式模式，这样我们就能在输入路径实例时进行匹配，我们当前使用这个正则表达式模板我们很小题的变量。

那么，匹配我们的「/hello/」的正则表达式应该长啥样呢？

唯一个简单的正则表达式模式「^/hello/\.+$」将是个好的开始。让我们一起看看它实时识别是怎么一起工作的：

```
import re
route_regex = re.compile(r"^/hello/\.+$")
match = route_regex.match("/hello/alex")
print match.groups()
```

将会输出：
('alex',)

不错，不过，理想情况是我们想要提供我们对匹配上的第一组结果，并且从路径「/hello/」中删除「username」。

命名捕获组

幸运的是，正则表达式也支持命名捕获组，允许我们将匹配组分配一个名字。我们能在读取我们的匹配之后将匹配。

我们可以使用下述符号，给出第一个例子中的「username」的捕获组。

```
r'/hello/(?P<username>.+)$'
```

然后我们可以对我们的正则表达式使用groupdict方法生，将所有捕获组当作一个字典，组的名字对应匹配上的值。

那么我们组出下述代码：

```
route_regex = re.compile(r"^/hello/(?P<username>.+)$")
match = route_regex.match("/hello/alex")
print match.groupdict()
```

将为我们输出以下字典：
{'username': 'alex'}

现在，有了我们新增类型正则表达式的格式，以及如何使用它们去匹配输入的URL或地址，最后剩下的是写一个方法，将我们声明的路径转换成可实现分组正则表达式模式。

要做这个我们将使用另一个正则表达式（接下来将会是正则表达式），为了让我们路径中的变量转换成正则表达式模式。那就是作为示例匹配三种将「 转换成「(?P<」。

听起来太简单了！我们将可以只用一行新代码实现它。

```
def build_route_pattern(route):
    route_regex = re.sub(r'<(\w+)>', r'(?P<\1>.+)', route)
    return re.compile(r"^{}$".format(route_regex))
print build_route_pattern("/hello/<username>")
```

这里我们用一个正则表达式代表所有出现的模式（一个包含在尖括号中的字符串），与它的正则表达式命名组等价。

re.sub的第一个参数 我们将我们的模式匹配进括号，目的是把它分配到第一个匹配组。在我们的第二个参数，我们可以使用第一匹配组的内容，方法是写1（2将是第二匹配组的内容，以此类推……）

那么最后，输入模式

```
/hello/<username>
```

将给我们正则表达式：

```
^/hello/(?P<username>.+)$
```

路线注册

让我们来下一跟上次我们写的简单Flask类。

```
class Nonflask():
    def __init__(self):
        self.routes = {}

    def route(self, route_str):
        def decorator(f):
            self.routes[route_str] = f
            return f

        return decorator

    def serve(self, path):
        view_function = self.routes.get(path)
        if view_function:
            return view_function()
        else:
            raise ValueError("Route '{}' has not been registered".format(path))

app = Nonflask()

@app.route("/")
def hello():
    return "Hello World!"
```

现在我们有一个新的改进方法用来匹配输入的路径，我们才算稍微得到了上一版实现时用到的原生字典。

让我们从改造我们的函数着手。以便于添加路径，这样我们就可以用pattern, view_function元组作为我们新字典保存路径的路径。

这意味着当一个程序员使用@app.route()装饰一个函数，我们将要尝试将他们的路径编译变成一个正则表达式，然后存储它，属于一个在我们新的路径列表里的新项函数。

让我们看看实现代码：

```
class Nonflask():
    def __init__(self):
        self.routes = []

    # Here's our build_route_pattern we made earlier
    @staticmethod
    def build_route_pattern(route):
        route_regex = re.sub(r'<(\w+)>', r'(?P<\1>.+)', route)
        return re.compile(r"^{}$".format(route_regex))

    def route(self, route_str):
        def decorator(f):
            # Instead of adding just the function to our routes dictionary,
            # we'll append the path to our route list
            route_pattern = self.build_route_pattern(route_str)
            self.routes.append((route_pattern, f))

            return f

        return decorator
```

我们创建两个一个新的改进方法用来匹配输入的路径，我们才算稍微得到了上一版实现时用到的原生字典。

让我们从改造我们的函数着手。以便于添加路径，这样我们就可以用pattern, view_function元组作为我们新字典保存路径的路径。

这意味着当一个程序员使用@app.route()装饰一个函数，我们将要尝试将他们的路径编译变成一个正则表达式，然后存储它，属于一个在我们新的路径列表里的新项函数。

让我们看看实现代码：

```
class Nonflask():
    def __init__(self):
        self.routes = []

    # Here's our build_route_pattern we made earlier
    @staticmethod
    def build_route_pattern(route):
        route_regex = re.sub(r'<(\w+)>', r'(?P<\1>.+)', route)
        return re.compile(r"^{}$".format(route_regex))

    def route(self, route_str):
        def decorator(f):
            # Instead of adding just the function to our routes dictionary,
            # we'll append the path to our route list
            route_pattern = self.build_route_pattern(route_str)
            self.routes.append((route_pattern, f))

            return f

        return decorator
```

我们怎打算需要一个get_route_match方法，给它一个路径实例，将会尝试匹对到一个匹配的view_function，或者返回None如果一个也找不到的话。

然而，如果我们不到匹配的话，除了view_function之外，我们还需要返回一个东西。那就是我们在当之前捕获匹配组的字典。我们需要它来为视图函数传递正确的参数。

好了我们的get_route_match大概就长这样：

```
def get_route_match(path):
    for route_pattern, view_function in self.routes:
        m = route_pattern.match(path)
        if m:
            return m.groupdict(), view_function

    return None
```

现在我们快要完成了，最后一步将是向出现向view_function的活动。使用来自正则表达式匹配组字典的正确参数。

调用一个函数的若干种方法

让我们回顾一下不同的方法调用一个python的函数：

比如像这样：

```
def hello_user(username):
    return "Hello {}!".format(username)
```

最简单的（也许正是你所熟悉的）方法是使用正则参数。在这里参数的顺序和匹配我们定义的那些函数的顺序。

```
>>> hello_user("alex")
Hello alex!
```

另一种方法调用一个函数是最使用关键词参数。关键词参数可以通过任何顺序指定，适合有许多可选参数的函数。

```
>>> hello_user(username="alex")
Hello alex!
```

在内，python最后一种调用一个函数的方法是使用关键词参数字典。字典中的关键词对应参数名称。我们在python解包一个字典，并通过使用两个星号 "**"，来把它当作函数的关键词参数。下面的代码段与上面的代码段完全一样，现在我们使用字典参数，我们可以非常好得的创建它。

```
>>> kwargs = {"username": "alex"}
>>> hello_user(**kwargs)
Hello alex!
```

好了，还记得上面的groupdict方法？就是那个将特的在正则表达式完成匹配后返回「'username' : 'alex'」的那个？那么现在我们了解了kwargs，我们能很容易将我们的view_function传递字典匹配，完成Flask routing。

那么让我们把这些都建进我们最得的类中。

```
class Nonflask():
    def __init__(self):
        self.routes = []

    @staticmethod
    def build_route_pattern(route):
        route_regex = re.sub(r'<(\w+)>', r'(?P<\1>.+)', route)
        return re.compile(r"^{}$".format(route_regex))

    def route(self, route_str):
        def decorator(f):
            route_pattern = self.build_route_pattern(route_str)
            self.routes.append((route_pattern, f))

            return f

        return decorator

    def get_route_match(self, path):
        for route_pattern, view_function in self.routes:
            m = route_pattern.match(path)
            if m:
                return m.groupdict(), view_function

        return None

    def serve(self, path):
        route_match = self.get_route_match(path)
        if route_match:
            kwargs, view_function = route_match
            return view_function(**kwargs)
        else:
            raise ValueError("Route '{}' has not been registered".format(path))
```

接下来，就是见证奇迹的时刻，请看下面代码段：

```
app = Nonflask()

@app.route("/hello/<username>")
def hello_user(username):
    return "Hello {}!".format(username)

print app.serve("/hello/alex")
```

我们将得到输出：

Hello alex!

结语

好了这就是「这不是魔法」对Flask的app.route()的深入探讨。它证明我们所需要做的只是一点正则表达式和python使用关键词参数调用函数的力法。用来和我们的URL增加一点动态性。

NotFlask例子的源码，以及配套的测试测试套件已经放在GitHub上请前去查看。

本系列的下一篇我将会深入研究AngularJS，看看它有多新颖主题，以及如何实现下面代码段的声明：

```
angular.module('test', ['$http', function($http) {
    $http.get("http://google.com").success(function(data) {
        console.log(data);
    });
}]);
```


关于作者：木羊同学


hacks.py
个人主页 · 我的文章 · 粉 12

相关文章

- Flask 模板简介
- Flask 应用中的 URL 处理
- Flask 中的蓝图简介
- Flask 中模块化应用的实现
- 一个Flask应用的打包和部署 · 5

可能感兴趣的话题

- 紧张关于神经网络的不当格式和消息
- 写了一个Vim-Markdown预览插件，欢迎使用 · 2
- 一键发微博和支持14个Java测试题，你都对了吗程序员
- 深入浅出MVC之后端免费资源
- 你也能写入公司的前端合作公司吗？ · 4
- Vuex1.0 一个基于Vue的数据流框架

最新评论

bimbim（粉 1）
计算机硕士
2015/07/19

你好，那个正则表达式是对了，应该是
route_regex = re.sub(r'(?P<\1>.+)', route)才对。

还有最后的地方
@app.route("/hello/")
应该是改为@app.route("/hello")

楼主正规文章翻译很不错，支持适用，谢谢！

黑色城（粉 86 · ♀）
Qt
2015/05/20

嗯，谢谢范围，已经更新了。

应该是以前的代码再用到的思路问题

木羊同学（粉 12）
程序员
2015/07/20

抱歉啦，你们那条点了

xiqin（粉 0）
2016/01/18

好文章，作为一个新手准备模仿解释知识点,渐渐的文章，对于新功能的理解又深了一些，期待后面又下一个正则，就是nationmethod的同不知道用啥心看明白，怎样改写build_route_pattern(self, route)这一段就能解