11.5 创建一个简单的REST接口¶

问题¶

你想使用一个简单的REST接口通过网络远程控制或访问你的应用程序，但是你又不想自己去安装一个完整的web框架。

解决方案¶

构建一个REST风格的接口最简单的方法是创建一个基于WSGI标准（PEP 3333）的很小的库，下面是一个例子：

```
# resty.py

import cgi

def notfound_404(environ, start_response):
    start_response('404 Not Found', [ ('Content-type', 'text/plain') ])
    return [b'Not Found']

class PathDispatcher:
    def __init__(self):
        self.pathmap = { }

    def __call__(self, environ, start_response):
        path = environ['PATH_INFO']
        params = cgi.FieldStorage(environ['wsgi.input'],
                                  environ=environ)
        method = environ['REQUEST_METHOD'].lower()
        environ['params'] = { key: params.getvalue(key) for key in params }
        handler = self.pathmap.get((method,path), notfound_404)
        return handler(environ, start_response)

    def register(self, method, path, function):
        self.pathmap[method.lower(), path] = function
        return function
```

为了使用这个调度器，你只需要编写不同的处理器，就像下面这样：

```
import time

_hello_resp = '''\
<html>
  <head>
     <title>Hello {name}</title>
   </head>
   <body>
     <h1>Hello {name}!</h1>
   </body>
</html>'''

def hello_world(environ, start_response):
    start_response('200 OK', [ ('Content-type','text/html')])
    params = environ['params']
    resp = _hello_resp.format(name=params.get('name'))
    yield resp.encode('utf-8')

_localtime_resp = '''\
```

```
_localtime_resp = '''\
<?xml version="1.0"?>
<time>
  <year>{t.tm_year}</year>
  <month>{t.tm_mon}</month>
  <day>{t.tm_mday}</day>
  <hour>{t.tm_hour}</hour>
  <minute>{t.tm_min}</minute>
  <second>{t.tm_sec}</second>
</time>'''

def localtime(environ, start_response):
    start_response('200 OK', [ ('Content-type', 'application/xml') ])
    resp = _localtime_resp.format(t=time.localtime())
    yield resp.encode('utf-8')

if __name__ == '__main__':
    from resty import PathDispatcher
    from wsgiref.simple_server import make_server

    # Create the dispatcher and register functions
    dispatcher = PathDispatcher()
    dispatcher.register('GET', '/hello', hello_world)
    dispatcher.register('GET', '/localtime', localtime)

    # Launch a basic server
    httpd = make_server('', 8080, dispatcher)
    print('Serving on port 8080...')
    httpd.serve_forever()
```

要测试下这个服务器，你可以使用一个浏览器或 `urllib` 和它交互。例如：

```
>>> u = urlopen('http://localhost:8080/hello?name=Guido')
>>> print(u.read().decode('utf-8'))
<html>
  <head>
    <title>Hello Guido</title>
  </head>
  <body>
    <h1>Hello Guido!</h1>
  </body>
</html>

>>> u = urlopen('http://localhost:8080/localtime')
>>> print(u.read().decode('utf-8'))
<?xml version="1.0"?>
<time>
  <year>2012</year>
  <month>11</month>
  <day>24</day>
  <hour>14</hour>
  <minute>49</minute>
  <second>17</second>
</time>
>>>
```

讨论¶

在编写REST接口时，通常都是服务于普通的HTTP请求。但是跟那些功能完整的网站相比，你通常只需要处理数据。这

些数据以各种标准格式编码，比如XML、JSON或CSV。 尽管程序看上去很简单，但是以这种方式提供的API对于很多应用程序来讲是非常有用的。

例如，长期运行的程序可能会使用一个REST API来实现监控或诊断。 大数据应用程序可以使用REST来构建一个数据查询或提取系统。 REST还能用来控制硬件设备比如机器人、传感器、工厂或灯泡。 更重要的是，REST API已经被大量客户端编程环境所支持，比如Javascript, Android, iOS等。 因此，利用这种接口可以让你开发出更加复杂的应用程序。

为了实现一个简单的REST接口，你只需让你的程序代码满足Python的WSGI标准即可。 WSGI被标准库支持，同时也被绝大部分第三方web框架支持。 因此，如果你的代码遵循这个标准，在后面的使用过程中就会更加的灵活！

在WSGI中，你可以像下面这样约定的方式以一个可调用对象形式来实现你的程序。

```
import cgi

def wsgi_app(environ, start_response):
    pass
```

`environ` 属性是一个字典，包含了从web服务器如Apache[参考Internet RFC 3875]提供的CGI接口中获取的值。 要将这些不同的值提取出来，你可以像这么这样写：

```
def wsgi_app(environ, start_response):
    method = environ['REQUEST_METHOD']
    path = environ['PATH_INFO']
    # Parse the query parameters
    params = cgi.FieldStorage(environ['wsgi.input'], environ=environ)
```

我们展示了一些常见的值。 `environ['REQUEST_METHOD']` 代表请求类型如GET、POST、HEAD等。 `environ['PATH_INFO']` 表示被请求资源的路径。 调用 `cgi.FieldStorage()` 可以从请求中提取查询参数并将它们放入一个类字典对象中以便后面使用。

`start_response` 参数是一个为了初始化一个请求对象而必须被调用的函数。 第一个参数是返回的HTTP状态值，第二个参数是一个(名,值)元组列表，用来构建返回的HTTP头。例如：

```
def wsgi_app(environ, start_response):
    pass
    start_response('200 OK', [('Content-type', 'text/plain')])
```

为了返回数据，一个WSGI程序必须返回一个字节字符串序列。可以像下面这样使用一个列表来完成：

```
def wsgi_app(environ, start_response):
    pass
    start_response('200 OK', [('Content-type', 'text/plain')])
    resp = []
    resp.append(b'Hello World\n')
    resp.append(b'Goodbye!\n')
    return resp
```

或者，你还可以使用 `yield` ：

```
def wsgi_app(environ, start_response):
    pass
    start_response('200 OK', [('Content-type', 'text/plain')])
    yield b'Hello World\n'
```

```
    yield b'Goodbye!\n'
```

这里要强调的一点是最后返回的必须是字节字符串。如果返回结果包含文本字符串，必须先将其编码成字节。 当然，并没有要求你返回的一定是文本，你可以很轻松的编写一个生成图片的程序。

尽管WSGI程序通常被定义成一个函数，不过你也可以使用类实例来实现，只要它实现了合适的 `__call__()` 方法。例如：

```
class WSGIApplication:
    def __init__(self):
        ...
    def __call__(self, environ, start_response)
        ...
```

我们已经在上面使用这种技术创建 `PathDispatcher` 类。 这个分发器仅仅只是管理一个字典，将(方法,路径)对映射到处理器函数上面。 当一个请求到来时，它的方法和路径被提取出来，然后被分发到对应的处理器上面去。 另外，任何查询变量会被解析后放到一个字典中，以 `environ['params']` 形式存储。 后面这个步骤太常见，所以建议你在分发器里面完成，这样可以省掉很多重复代码。 使用分发器的时候，你只需简单的创建一个实例，然后通过它注册各种WSGI形式的函数。 编写这些函数应该超级简单了，只要你遵循 `start_response()` 函数的编写规则，并且最后返回字节字符串即可。

当编写这种函数的时候还需注意的一点就是对于字符串模板的使用。 没人愿意写那种到处混合着 `print()` 函数 、XML和大量格式化操作的代码。 我们上面使用了三引号包含的预先定义好的字符串模板。 这种方式的可以让我们很容易的在以后修改输出格式(只需要修改模板本身，而不用动任何使用它的地方)。

最后，使用WSGI还有一个很重要的部分就是没有什么地方是针对特定web服务器的。 因为标准对于服务器和框架是中立的，你可以将你的程序放入任何类型服务器中。 我们使用下面的代码测试测试本节代码：

```
if __name__ == '__main__':
    from wsgiref.simple_server import make_server

    # Create the dispatcher and register functions
    dispatcher = PathDispatcher()
    pass

    # Launch a basic server
    httpd = make_server('', 8080, dispatcher)
    print('Serving on port 8080...')
    httpd.serve_forever()
```

上面代码创建了一个简单的服务器，然后你就可以来测试下你的实现是否能正常工作。 最后，当你准备进一步扩展你的程序的时候，你可以修改这个代码，让它可以为特定服务器工作。

WSGI本身是一个很小的标准。因此它并没有提供一些高级的特性比如认证、cookies、重定向等。 这些你自己实现起来也不难。不过如果你想要更多的支持，可以考虑第三方库，比如 `WebOb` 或者 `Paste`