

从现在开始，我们将真正开始接触REST框架的核心。我们来介绍几个基本的构建模块。

一、请求对象（Request objects）

DRF引入了一个扩展Django常规 `HttpRequest` 对象的 `Request` 对象，并提供了更灵活的请求解析能力。`Request` 对象的核心功能是 `request.data` 属性，它与 `request.POST` 类似，但对于使用Web API更为有用。

```
1 request.POST # 只处理表单数据 只适用于'POST'方法
2 request.data # 处理任意数据 适用于'POST', 'PUT'和'PATCH'等方法
```

在DRF中，请始终使用 `request.data`，不要使用 `request.POST`。

二、响应对象（Response objects）

DRF同时还引入了一个 `Response` 对象，这是一种尚未对内容进行渲染的 `TemplateResponse` 类型，并使用内容协商的结果来确定返回给客户端正确的内容类型。

```
1 return Response(data) # 渲染成客户端请求的内容类型。
```

三、状态码（Status codes）

在前后端分离的RESTful模式中，我们不能简单、随意地返回响应，而是需要使用HTTP规定的，大家都认可的状态码的形式。

然而，在视图中使用纯数字的HTTP状态码并不总是那么容易被理解，很容易被忽略。REST框架为 `status` 模块中的每个状态代码（如 `HTTP_400_BAD_REQUEST`）提供了更明确的标识符。使用它们来代替纯数字的HTTP状态码是个很好的主意。

四、封装API视图

REST框架提供了三种可用于编写API视图的包装器（wrappers）。

1. 基于函数视图的 `@api_view` 装饰器
2. 基于类视图的 `APIView` 类系列


```
23         return Response(serializer.errors,
                           status=status.HTTP_400_BAD_REQUEST)
```

当前的视图比前面的示例有所改进，它稍微简洁一点。在装饰器的参数位置指定视图支持的HTTP方法类型。不需要你额外处理csrf问题，也不使用Django的JSONResponse方法了，而是使用DRF的Response方法。此外，我们还使用了命名状态代码status，这使得响应意义更加明显。你可以对比一下代码前后的变化，加深理解和印象。

同样，我们重写 `views.py` 模块中snippet的detail视图。

```
1  @api_view(['GET', 'PUT', 'DELETE'])
2  def snippet_detail(request, pk):
3      """
4      Retrieve, update or delete a code snippet.
5      """
6      try:
7          snippet = Snippet.objects.get(pk=pk)
8      except Snippet.DoesNotExist:
9          return Response(status=status.HTTP_404_NOT_FOUND)
10
11     if request.method == 'GET':
12         serializer = SnippetSerializer(snippet)
13         return Response(serializer.data)
14
15     elif request.method == 'PUT':
16         serializer = SnippetSerializer(snippet, data=request.data)
17         if serializer.is_valid():
18             serializer.save()
19             return Response(serializer.data)
20         return Response(serializer.errors,
                           status=status.HTTP_400_BAD_REQUEST)
21
22     elif request.method == 'DELETE':
23         snippet.delete()
24         return Response(status=status.HTTP_204_NO_CONTENT)
```

目前为止，我们写的API视图和普通的Django视图没有什么太大区别，还是很好理解的。

注意，我们不再显式地将请求或响应绑定到给定的内容类型。`request.data` 可以处理传入的 `json` 请求，也可以处理其他格式。同样，我们返回带有数据的响应对象，但允许REST框架将响应给渲染成正确内容类型，比如json。

六、为url添加可选的后缀

其实这部分内容不应该出现在这里，它是和主干无关的细节。

在DRF的机制中，响应数据的格式不再与单一内容类型连接，可以同时响应json格式或HTML格式。我们可以为API路径添加对格式后缀的支持。使用格式后缀给我们明确指定了给定格式的URL，这意味着我们的API将能够处理诸如 `http://example.com/api/items/4.json` 之类的URL。

像下面这样在这两个视图中添加一个 `format` 关键字参数。

```
1 def snippet_list(request, format=None):
```

和

```
1 def snippet_detail(request, pk, format=None):
```

仅仅在视图中添加format参数还不够，还需要在路由中进行设置。现在更新 `snippets/urls.py` 文件，为现有的URL后面添加一组 `format_suffix_patterns`。

```
1 from django.urls import path
2 from rest_framework.urlpatterns import format_suffix_patterns
3 from snippets import views
4
5 urlpatterns = [
6     path('snippets/', views.snippet_list),
7     path('snippets/<int:pk>', views.snippet_detail),
8 ]
9
10 urlpatterns = format_suffix_patterns(urlpatterns)
```

注意上面最后一句代码，它的意思是用 `format_suffix_patterns` 来封装urlpatterns，这样每一个带有 `.json` 等后缀的url都能被正确解析。

以上操作都是可选的，我们不一定需要添加这些额外的url模式，但它给了我们一个简单，清晰的方式来引用特定的格式。

七、测试我们的工作

从命令行开始测试API，就像我们在前面所做的那样。

可以像以前一样获取所有snippet的列表。

```
1  执行命令: http http://127.0.0.1:8000/snippets/
2
3
4
5  HTTP/1.1 200 OK
6  Allow: GET, OPTIONS, POST
7  Content-Length: 319
8  Content-Type: application/json
9  Date: Sun, 28 Apr 2019 04:32:47 GMT
10 Server: WSGIServer/0.2 CPython/3.7.3
11 Vary: Accept, Cookie
12 X-Frame-Options: SAMEORIGIN
13
14  [
15      {
16          "code": "foo = \"bar\"\n",
17          "id": 1,
18          "language": "python",
19          "linenos": false,
20          "style": "friendly",
21          "title": ""
22      },
23      {
24          "code": "print(\"hello, world\")\n",
25          "id": 2,
26          "language": "python",
27          "linenos": false,
28          "style": "friendly",
29          "title": ""
30      },
31      {
32          "code": "print(\"hello, world\")",
33          "id": 3,
34          "language": "python",
35          "linenos": false,
36          "style": "friendly",
37          "title": ""
38      }
39  ]
40
```

可以通过使用 **Accept** 标头来控制我们回复的响应格式:

```
1 http http://127.0.0.1:8000/snippets/ Accept:application/json # 请求JSON
2 http http://127.0.0.1:8000/snippets/ Accept:text/html # 请求HTML
```

或者通过附加后缀的方式:

```
1 http http://127.0.0.1:8000/snippets.json # JSON后缀
2 http http://127.0.0.1:8000/snippets.api # 可浏览API后缀
```

类似地, 可以使用 `Content-Type` 头控制我们发送的请求的格式。

```
1 # POST表单数据
2 http --form POST http://127.0.0.1:8000/snippets/ code="print 123"
3
4 HTTP/1.1 201 Created
5 Allow: GET, OPTIONS, POST
6 Content-Length: 93
7 Content-Type: application/json
8 Date: Sun, 28 Apr 2019 04:35:02 GMT
9 Server: WSGIServer/0.2 CPython/3.7.3
10 Vary: Accept, Cookie
11 X-Frame-Options: SAMEORIGIN
12
13 {
14     "code": "print 123",
15     "id": 4,
16     "language": "python",
17     "linenos": false,
18     "style": "friendly",
19     "title": ""
20 }
21
22
23 # POST JSON数据
24 http --json POST http://127.0.0.1:8000/snippets/ code="print 456"
25
26 HTTP/1.1 201 Created
27 Allow: GET, OPTIONS, POST
28 Content-Length: 94
29 Content-Type: application/json
30 Date: Sun, 28 Apr 2019 04:35:39 GMT
31 Server: WSGIServer/0.2 CPython/3.7.3
32 Vary: Accept, Cookie
33 X-Frame-Options: SAMEORIGIN
34
```

```
35  {
36      "code": "print(456)",
37      "id": 5,
38      "language": "python",
39      "linenos": false,
40      "style": "friendly",
41      "title": ""
42  }
```

如果你向上述 `http` 请求添加 `--debug` 参数，则可以在请求标头中查看更详细的内容。

最关键的是：现在可以在浏览器中访问 `http://127.0.0.1:8000/snippets/`，可以看到下面的页面：

The screenshot shows a web browser interface for a Django REST framework API. At the top, there is a dark header with the text "Django REST framework". Below this, the page title is "Snippet List". To the right of the title are two buttons: "OPTIONS" and "GET". Below the title, there is a description: "List all code snippets, or create a new snippet." Below this, there is a section for the GET request, showing the URL "/snippets/" and the response. The response is a JSON array of three code snippets, each with fields for id, title, code, linenos, language, and style. The first snippet has id 1, title "", code "foo = \"bar\\n\"", linenos false, language "python", and style "friendly". The second snippet has id 2, title "", code "print(\"hello, world\")\n", linenos false, language "python", and style "friendly". The third snippet has id 3, title "", and is partially visible.

并且下面还有一个可以创建新sinppet的表单：

Media type: application/json

Content:

POST

同样，访问 <http://127.0.0.1:8000/snippets/1/>，会看到下面的序号为1的snippet的具体内容：

Django REST framework

Snippet List / Snippet Detail

Snippet Detail

Retrieve, update or delete a code snippet.

DELETE OPTIONS GET

GET /snippets/1/

```
HTTP 200 OK
Allow: GET, OPTIONS, PUT, DELETE
Content-Type: application/json
Vary: Accept

{
  "id": 1,
  "title": "",
  "code": "foo = `bar`\n",
  "linenos": false,
  "language": "python",
  "style": "friendly"
}
```

并且也有一个更新内容的表单：

Media type: application/json

Content:

PUT

你可能会疑惑，这么高端大气的页面怎么来的，我们没有编写这个页面HTML啊。这是DRF安利给我们的，内置的。

DRF的API视图会根据客户端请求的响应内容类型返回对应类型的数据，因此当Web浏览器请求snippets时，它实际请求的是HTML格式，而不是json等格式，API会按要求返回HTML格式的表现，这个过程是DRF早就写好了的，在源码中实现了的。

DRF的这个功能，比较类似Django的admin后台的理念，大大降低了开发人员检查和使用API的障碍，可视化后更直接、更清晰、更便捷。