🡪

Django目前发展的比较好的REST框架为Django REST FRAMEWORK.可用Django版本为：1.11, 2.0, 2.1, Python版本的要求为2.7,3.4,3.6,3.7. 并且DRF（实际就是python包）有自己的版本：例如3.11.0.

DRF的一个好处是，在后端中采用DRF，提供API方式的数据支持，在前端便可以采用任意的框架, 框架并不局限于web框架，指的更是手机APP等应用。不同的终端，有不同的渲染技术，但是数据只有一个，所以RESEFUL API这样不局限于前台，返回数据提交数据，达到可以随意切换前台框架。达到前后台完全分离隔绝，使后台稳定。

测试DRF的工具有许多，可以直接浏览器访问，使用curl命令行，使用postman工具等。

🡪

使用DRF的时候，一般和网页处理函数分开，更易于管理。可以在根urls.py中对Restful请求开头的所有url匹配放入对应restful app下的自身的urls.py中，并且匹配时候到对应restful app的单独views.py中。

🡪

主要的媒体格式类型(MIME类型)如下：

* text/html ： HTML格式
* text/plain ：纯文本格式
* text/xml ： XML格式
* image/gif ：gif图片格式
* image/jpeg ：jpg图片格式
* image/png：png图片格式

   以application开头的媒体格式类型：

* application/xhtml+xml ：XHTML格式
* application/xml     ： XML数据格式
* application/atom+xml ：Atom XML聚合格式
* application/json    ： JSON数据格式
* application/pdf       ：pdf格式
* application/msword ： Word文档格式
* application/octet-stream ： 二进制流数据（如常见的文件下载）
* application/x-www-form-urlencoded ： <form encType=””>中默认的encType，form表单数据被编码为key/value格式发送到服务器（表单默认的提交数据的格式）

   另外一种常见的媒体格式是上传文件之时使用的：

* multipart/form-data ： 需要在表单中进行文件上传时，就需要使用该格式

我们可以在发送http请求中(也就是http请求头中):

***Accept字段***：告诉服务器，我们请求端这边需要什么类型的MIME返回类型。

***Content-type字段***：告诉服务器，请求端放在http请求中所携带数据的MIME类型。

当客户端发起请求，并且没有特别指定Accept字段时候，Django Rest Framework默认会返回HTML格式的数据。

🡪

面向资源是REST最明显的特征，对于同一个资源的一组不同的操作。资源是服务器上一个可命名的抽象概念，资源是以名词为核心来组织的，首先关注的是名词。REST要求，必须通过统一的接口来对资源执行各种操作。对于每个资源只能执行一组有限的操作。（7个HTTP方法：GET/POST/PUT/DELETE/PATCH/HEAD/OPTIONS）

并且REST架构风格并不是绑定在HTTP上，只不过目前HTTP是唯一与REST相关的实例

🡪

幂等与非幂等：

等幂性简单点说就是一次请求和多次请求，资源的状态是一样。比如GET和HEAD，不论你请求多少次，资源还是在那里。请注意，DELETE和PUT也是等幂的，因为对同一个资源删除一次或者多次，结果是一样的，就是资源被删除了，不存在了。为什么说PUT也是等幂的？当你PUT一个新资源的时候，资源被创建，再次PUT这个URI的时候，资源还是没变。当你PUT一个存在的资源时，更新了资源，再次PUT的时候，还是更新成这个样子。

我们设计Restful WS的时候，GET，HEAD, PUT, DELETE一定要设计成等幂的。由于网络是不可靠的，安全性和等幂性就显得特别重要。如果一次请求，服务器收到处理以后，客户端没有收到相应，客户端会再次请求，如果没有等幂性保障，就会发生意想不到的问题。

POST是不安全也不等幂的，还是拿weblog的例子，如果两次POST相同的博文，则会产生两个资源，URI可能是这样/weblogs/myweblog/entries/1和/weblogs/myweblog/entries/2，尽管他们的内容是一样的

|  |  |
| --- | --- |
| Verd | 描述 |
| HEAD（SELECT） | 只获取某个资源的头部信息，幂等 |
| GET（SELECT） | 获取资源，幂等 |
| POST（CREATE） | 创建资源，非幂等 |
| PATCH（UPDATE） | 更新资源的部分属性，幂等 |
| PUT（UPDATE） | 更新资源，客户端需要提供新建资源的所有属性，幂等 |
| OPTIONS | 用于url验证，显示接口信息查询服务信息，幂等 |
| DELETE（DELETE） | 删除资源，幂等 |

在HTTP报文构成中，有个字段很重要：status code。它说明请求的大致情况，是否正常处理、出现了什么错误等。状态码都是三位数，大概分为了一下几个区间：

|  |  |
| --- | --- |
| 状态码 | 描述 |
| 2XX | 请求正常处理并返回 |
| 3XX | 重定向，请求的资源位置发生变化 |
| 4XX | 客户端发送的请求有误 |
| 5XX | 服务器端的错误 |

🡪

DRF中的browserable API需要用的JS以及css文件的支撑。这些文件在通过pip安装的DRF中都已经自己包含，不需要用户执行下载。我们要做的是确保我们的服务器可以serve到这些静态文件。

🡪

Coreapi以及django-rest-swagger包可以为我们DRF API提供自动生成的文档。

🡪

Django Rest Framework中Response类是rest\_framework.response.Response. 该类可以同request请求头共同协商返回的数据类型。

构造签名为：

Response(data, status=None, template\_name=None, headers=None, content\_type=None)

其中data为序列化后的data.序列化的data包括字典，列表，序列器实例.data。

而普通Django框架下中Response类是django.http.HttpResponse.虽然在Restful中也可以使用django.http.HttpResponse作为响应返回。但是django.http.HttpResponse没有协商机制，不推荐。

🡪

Django Rest Framework中通过注册不同的Render可以提供不同的功能。Render的作用是，当DRF取出数据以后，数据以什么样的方式返回给请求端。例如JSONRender将数据以JSON字符串的形式返回给请求端，BrowsableAPIRenderer将数据渲染成HTML文件返回给请求端。

**如果确定数据返回给客户端时候使用什么render?**

确定使用什么render要经过一个和收到的请求进行协商的过程(negotiation)，来选取最合适的render.

DRF会检查请求头部中的Accept字段，检查请求端想收到什么样的形式。

另外DRF检查url中是否含有信息，明确的表达了想收到什么格式的信息。例如http://example.com/api/users\_count.json地址明确表示想收到json字符串的结果。服务器端将使用对应的render.（在DRF中通过format\_suffix\_patterns启用）

另外DRF检查url中是否含有指明格式的查询参数，明确的表达了想收到什么格式的信息。例如http://example.com/api/users\_count&format=json中format=json明确表示想收到json字符串的结果。服务器端将使用对应的render. （在DRF中默认启用）

当协商完毕以后，我们可以使用request.accepted\_renderer来获取最终协商的结果将采用什么render实例. 而每个render实例又有.format属性来查看最终的渲染格式。

Class list\_users(APIView):

queryset = Users.objects.filter(active=True)

render\_class = [“TemplateHTMLRenderer”, “JSONRenderer”]

if request.accepted\_renderer.format == 'html': # 根据协商格式不同，执行不同逻辑

# TemplateHTMLRenderer takes a context dict,

# and additionally requires a 'template\_name'.

# It does not require serialization.

data = {'users': queryset}

return Response(data, template\_name='list\_users.html') # 当采用TemplateHTMLRender时候执行模板文件， 数据无需序列化.例如现在data为{'users': queryset},而queryset为Django的ORM对象。我们可以在不序列化的情况下直接传给response.

# JSONRenderer requires serialized data as normal.

serializer = UserSerializer(instance=queryset)

data = serializer.data

return Response(data) # 当采用JSONRenderer时候数据需要序列化,所以在此我们通过序列器serializer.data后再传入给Response对象。

**如何设置使用什么render?**

**在global范围**内以a list of classes的形式设置使用renderer，例如：

REST\_FRAMEWORK = {

'DEFAULT\_RENDERER\_CLASSES': [

'rest\_framework.renderers.JSONRenderer',

'rest\_framework.renderers.BrowsableAPIRenderer',

]

}

上述设置将global范围内设置首要的render为JSONRenderer'， 次要的为BrowsableAPIRenderer

**另外我们还可以在更小的颗粒度上进行设置，例如对某个独立的view**进行设置：

class UserCountView(APIView):

"""

A view that returns the count of active users in JSON.

"""

renderer\_classes = [JSONRenderer] 🡪 对于UserCountView使用JSONRenderer作为renderer

def get(self, request, format=None):

user\_count = User.objects.filter(active=True).count()

content = {'user\_count': user\_count}

return Response(content)

**render的设置顺序：**

**在render中设置的顺序会具有优先级不同的区别，在前面申明的render具有更高的优先级。如果请求头中没有指定，例如请求头中为Accept: \*/\*， 或者请求头中没有包括Accept字段，那么DRF默认选择定义在最前的render来作为渲染器。**

例如在'DEFAULT\_RENDERER\_CLASSES'设置：

REST\_FRAMEWORK = {

'DEFAULT\_RENDERER\_CLASSES': [

'rest\_framework.renderers.JSONRenderer',

'rest\_framework.renderers.BrowsableAPIRenderer',

]

}

例如在view中设置：

renderer\_classes = [JSONRenderer, TemplateHTMLRenderer ]

**Render种类：**

DRF提供一些默认的render例如JSONRenderer, TemplateHTMLRenderer等。除此以外，我们也可以安装额外的renderer或者自行设计一个renderer.

**JSONRenderer**: 将请求数据以JSON格式返回，使用utf-8编码

**TemplateHTMLRenderer**: 使用Django标准的模块渲染机制，通过对response.data渲染成HTML并返回。比较特别的是传给TemplateHTMLRenderer的数据不需要序列化。并且可以在Response（）中以template\_name指定模板名称。

例如：

class UserDetail(generics.RetrieveAPIView):

"""

A view that returns a templated HTML representation of a given user.

"""

queryset = User.objects.all()

renderer\_classes = [TemplateHTMLRenderer]

def get(self, request, \*args, \*\*kwargs):

self.object = self.get\_object()

return Response({'user': self.object}, template\_name='user\_detail.html') # 通过指定template\_name关键字参数，DRF将使用TemplateHTMLRenderer作为渲染器返回，模板中的数据为前面的字典。

**StaticHTMLRenderer**:直接在View中构建出HTML字符串，然后作为HTML文件返回。

例如：

@api\_view(['GET'])

@renderer\_classes([StaticHTMLRenderer]) # 使用StaticHTMLRenderer渲染器

def simple\_html\_view(request):

data = '<html><body><h1>Hello, world</h1></body></html>' # 直接构建HTML字符串

return Response(data) # 将预先构建HTML字符串作为响应进行返回。

**BrowserableAPIRender**:将返回的结果变为browserable的HTML返回。Browserable的HTML意味着，可以看到返回的数据，并且还可以直接在返回的HTML中进行GET, POST表单操作。而在返回的browserable HTML页面中，由于看到数据，所以这些返回的数据也是通过renderer进行渲染过。BrowserableAPIRender默认使用除BrowserableAPIRender以外，优先级最高的renderer去渲染这些数据，然后把渲染完的结果插入到返回的HTML中。