一、解析器

> 解析器是干什么的？因为前后端分离，因为可能采用json、xml、html等各种不同格式的内容，后端必须要有一个解析器来解析前端发送过来的数据，也就是翻译器！否则后端凭什么看懂前端的数据？对应地，后端也有一个渲染器Render，和解析器是相反的方向，将后端的数据翻译成前端能明白的数据格式。

REST框架提供了许多内置的Parser类，用来处理各种媒体类型的请求，比如json，比如xml。还支持自定义解析器，可以灵活地设计API接受的媒体类型。

Django原生的解析器对于post的数据，如果要从request.body中解析出来放到request.POST中，那么必须同时满足两个条件：

1. 请求头部 `Content_type = 'application/x-www-form-urlencoded'`
2. 数据格式必须是： `name=xxx&password=xxx&email=xxx.....`

而对于前端发送过来的例如JSON格式的数据则无法处理（当然你自己处理也是可以的）。DRF则不同，它提供了一些额外的解析器帮我们处理各种格式。

DRF的parsers模块非常简单，只定义了几个解析器类：

- BaseParser：解析器基类，以下四个类都直接继承它
- JSONParser
- FormParser
- MultiPartParser
- FileUploadParser

DRF在运行的时候如何知道该使用哪个解析器呢?

DRF将有效的解析器集定义为类的列表。当 `request.data` 被访问时，REST框架将检查请求头部的 `Content-Type` 属性，以此来确定要使用哪个解析器来解析数据。

所以，**要注意！解析器只有在请求request.data的时候才会被调用！**如果不需要data数据，那么就不用解析。

注意：在开发客户端应用程序时，务必确保在请求头部中包含`Content-Type`属性。如果未设置内容类型，则大多数客户端将默认使用`'application/x-www-form-urlencoded'`类型，但这可能不是你想要的。

例如，如果使用jQuery的ajax方法发送`json`编码的数据，则应在请求头部中设置`contentType: 'application/json'`。

解析器的相关配置参数

可以在Django项目的settings.py文件中，使用`DEFAULT_PARSER_CLASSES`配置项，进行全局的解析器设置。例如，以下设置仅允许解析JSON格式的请求，而不是默认的JSON或表单数据：

```
REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework.parsers.JSONParser',
    )
}
```

当然，也可以同时支持多种解析器，比如下面的配置，**这也是DRF默认的解析器配置**：

```
REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES':(
        'rest_framework.parsers.JSONParser',
        'rest_framework.parsers.FormParser',
        'rest_framework.parsers.MultiPartParser',
    )
}
```

几种解析器的写法没有先后顺序的要求，不像中间件那样的配置有顺序关系。

DRF支持视图级别的解析器，比如为基于APIView的类视图专门指定使用的解析器，核心是指定`parser_classes`属性为某个解析器:

```python
from rest_framework.parsers import JSONParser
from rest_framework.response import Response
from rest_framework.views import APIView

class ExampleView(APIView):
    """
    A view that can accept POST requests with JSON content.
    """
    # 看这里
    parser_classes = (JSONParser,)

    def post(self, request, format=None):
        return Response({'received data': request.data})
```

当然，也可以为使用 `@api_view` 装饰器改造的视图指定专用的解析器，核心是 `@parser_classes((JSONParser,))` 装饰器：

```python
from rest_framework.decorators import api_view
from rest_framework.decorators import parser_classes
from rest_framework.parsers import JSONParser

@api_view(['POST'])
@parser_classes((JSONParser,))  # 看这里
def example_view(request, format=None):
    """
    A view that can accept POST requests with JSON content.
    """
    return Response({'received data': request.data})
```

> 那么，问题来了！如果我在全局配置只允许JSON类型，但又在某个视图指定可以使用form表单类型，结果是什么？视图中的配置具有更高的优先级。

二、API参考

JSONParser

解析 `JSON` 格式的请求内容。

其.media_type属性值为 `application/json`

FormParser

解析HTML表单内容，使用QueryDict的数据填充request.data。这也是Django原生支持的解析方式。

通常我们希望同时支持FormParser和MultiPartParser两种解析器，以便完全支持HTML表单数据。

.media_type: `application/x-www-form-urlencoded`

MultiPartParser

解析多部分的HTML表单内容，支持文件上传。

.media_type: `multipart/form-data`

帮助：HTML的form表单的enctype属性规定了form表单在发送数据到服务器时的编码方式，有三种方式：

- application/x-www-form-urlencoded：默认的编码方式，常用于键值对数据。但是在用文本的传输和MP3等大型文件的时候，使用这种编码效率低下。
- multipart/form-data： 指定传输数据为二进制类型，比如图片、mp3、文件。
- text/plain：纯文体的传输。空格转换为 “+” 加号，但不对特殊字符编码。使用较少。

这种情况下，接收二进制文件的例子：

```
class FileUploadView(views.APIView):

    def post(self,request):
        print(request.body)
        with open(r'd:\temp', 'wb') as f:
            f.write(request.body)
        return Response("200,ok")
```

FileUploadParser

解析原始文件上传内容。此时，`request.data`属性将是一个字典，并且只包含一个键，这个键叫做`'file'`，对应的值包含上传的文件内容。

如果使用FileUploadParser解析器的视图，在被调用的时候URL中携带一个`filename`关键字参数，则该参数将被用作文件名。如果在没有这个关键字参数的情况下调用它，则客户端必须在HTTP头部的`Content-Disposition`中设置文件名。例如`Content-Disposition: attachment; filename=upload.jpg`。

.media_type：`*/*`

注意：

- `FileUploadParser`用于原生的文件上传请求。对于在浏览器中上传或者使用带有分段上传功能的客户端，请用`MultiPartParser`解析器。
- 由于`FileUploadParser`的`media_type`属性值是`*/*`，表示可以接受所有内容类型，所以无需指定别的解析器，指定`FileUploadParser`就足够了。
- `FileUploadParser`遵守Django标准的`FILE_UPLOAD_HANDLERS`设置和`request.upload_handlers`属性。

西面看一个具体的例子：

```
# views.py
class FileUploadView(views.APIView):
    parser_classes = (FileUploadParser,)

    def put(self, request, filename, format=None):
        file_obj = request.data['file']
        # ...
        # 在这里处理你的文件内容
        # ...
        return Response(status=204)

# urls.py
urlpatterns = [
    # ...
    path('upload/<str:filename>/', FileUploadView.as_view())
]
```

三、自定义解析器

要自定义解析器，必须继承 `BaseParser` 类，设置 `.media_type` 属性，并实现 `.parse(self, stream, media_type, parser_context)` 方法，该方法应返回将用于填充 `request.data` 属性的数据。

parse()方法的参数说明：

- stream

类似于流的对象，表示请求的主体。

- media_type

请求内容的媒体类型，可选

根据请求头部的 `Content-Type:` ，这可能比渲染器的 `media_type` 属性更具体，并且可能包括媒体类型参数。例如 `"text/plain; charset=utf-8"` 。

- parser_context

可选，字典格式。如果提供，将包含解析请求内容可能需要的任何其他上下文。

默认情况下，它包括以下的键： `view` ， `request` ， `args` ， `kwargs` 。

下面自定义了一个纯文本解析器，它将字符串形式表示的请求内容，填充到request.data属性。

```
class PlainTextParser(BaseParser):
    """
    纯文本解析器
    """
    media_type = 'text/plain'

    def parse(self, stream, media_type=None, parser_context=None):
        """
        简单的返回一个请求主体内容的字符串形式
        """
        return stream.read()
```

四、第三方模块

下面这些第三方模块为DRF提供了额外的解析器。

YAML解析器

`djangorestframework-yaml` 包为我们提供了解析和渲染yaml格式的能力。它以前直接包含在REST框架包中，现在作为第三方包出现。

直接使用pip安装。

```
$ pip install djangorestframework-yaml
```

可以进行下面的配置：

```
REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework_yaml.parsers.YAMLParser',
    ),
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework_yaml.renderers.YAMLRenderer',
    ),
}
```

XML解析器

以前也是内置，现在也作为第三方包存在：

```
$ pip install djangorestframework-xml
```

配置方法：

```
REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework_xml.parsers.XMLParser',
    ),
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework_xml.renderers.XMLRenderer',
    ),
}
```