一、渲染器

所谓的渲染器（Renderer），其实就是将服务器生成的数据的格式转换为HTTP请求的格式。

REST框架包含许多内置的Renderer类，可以返回各种媒体类型的响应。还支持定义自定义渲染器，灵活地设计自己的媒体类型。

如何确定渲染器

在DRF配置参数中，可用的渲染器依然是作为一个类的列表进行定义。但与解析器不同的是，渲染器的列表是有顺序关系的。REST框架将对传入请求执行内容协商，根据请求的类型确定最合适的渲染器以满足类型要求。

内容协商过程会检查请求头部的 `Accept` 属性，以确定客户期望的媒体类型。可选的，URL上的格式后缀可用于显式地请求特定的内容类型，例如 `http://example.com/api/users_count.json` 表明客户希望服务器返回JSON格式的数据。

配置渲染器的方法

使用 `DEFAULT_RENDERER_CLASSES` 设置全局渲染器集合。例如，以下设置将 `JSON` 用作主要媒体类型，也可以渲染可视化API，这也是**DRF的默认配置**。

```
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
    )
}
```

还可以为使用 `APIView` 基于类的视图，设置视图级别的，单独使用的渲染器。

```
from django.contrib.auth.models import User
from rest_framework.renderers import JSONRenderer
from rest_framework.response import Response
from rest_framework.views import APIView

```

```
class UserCountView(APIView):
    """
    A view that returns the count of active users in JSON.
    """
    # 注意这一行
    renderer_classes = (JSONRenderer, )

    def get(self, request, format=None):
        user_count = User.objects.filter(active=True).count()
        content = {'user_count': user_count}
        return Response(content)
```

或者对使用 `@api_view` 装饰器的视图，进行单独的渲染器指定。

```
@api_view(['GET'])
@renderer_classes((JSONRenderer,))    # 看这里!
def user_count_view(request, format=None):
    """
    A view that returns the count of active users in JSON.
    """
    user_count = User.objects.filter(active=True).count()
    content = {'user_count': user_count}
    return Response(content)
```

排序渲染器类的顺序

在为API指定渲染器类时，要考虑清楚为每种媒体类型分配什么样的优先级，这一点很重要。如果客户端未明确指定它可以接受的内容类型，例如发送 `Accept: */*` 属性值，或者根本没有 `Accept` 属性值，那么REST框架将选择settings列表中的第一个渲染器用于渲染响应内容。

如果你的API包含可根据请求的不同，同时提供常规网页和API响应的视图，请使用 `TemplateHTMLRenderer` 作为默认渲染器。

二、API参考

JSONRenderer

使用utf-8编码将响应内容渲染为json格式的数据。

默认样式包含unicode字符，并使用紧凑样式，没有不必要的空格，例如：

```
{"unicode black star":"★","value":999}
```

如果客户端设置了 `'indent'` 缩进参数，返回的内容将缩进。例如 `Accept: application/json; indent=4` 。

```
{
    "unicode black star": "★",
    "value": 999
}
```

可以使用 `UNICODE_JSON` 和 `COMPACT_JSON` 更改默认的JSON编码格式。

.media_type： `application/json`

.format： `'json'`

.charset： `None`

TemplateHTMLRenderer

使用Django的标准模板渲染器将数据渲染为HTML格式。与其他渲染器不同，此时传递给 `Response` 的数据不需要序列化。此外，可能还要使用 `template_name` 参数，指定你要渲染的HTML模板。

TemplateHTMLRenderer使用 `response.data` 作为上下文字典创建 `RequestContext` ，并确定要使用哪个模板。

模板的查询规则（按优先顺序）：

1. 显式指定的 `template_name` 参数。
2. 在TemplateHTMLRenderer上显式设置的 `.template_name` 属性。
3. `view.get_template_names()` 方法调用的返回结果。

下面是一个使用 `TemplateHTMLRenderer` 渲染器的示例：

```
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')
```

可以使用 `TemplateHTMLRenderer` 返回常规HTML页面，也可以返回API类型的响应。

如果你是与其他渲染器类一起混用的网站，则应将 `TemplateHTMLRenderer` 作为 `renderer_classes` 设置列表中的第一个类。

.media_type： `text/html`

.format： `'html'`

.charset： `utf-8`

StaticHTMLRenderer

一个简单的渲染器，只返回渲染好的HTML内容数据。与其他渲染器不同，传递给响应对象的数据应该是要返回内容的字符串形式。

例如：

```
@api_view(('GET',))
@renderer_classes((StaticHTMLRenderer,))
def simple_html_view(request):
    # 注意data变量是个字符串，虽然看起来像HTML代码
    data = '<html><body><h1>Hello, world</h1></body></html>'
    return Response(data)
```

既可以使用 `StaticHTMLRenderer` 返回常规的HTML页面，也可以返回API响应。

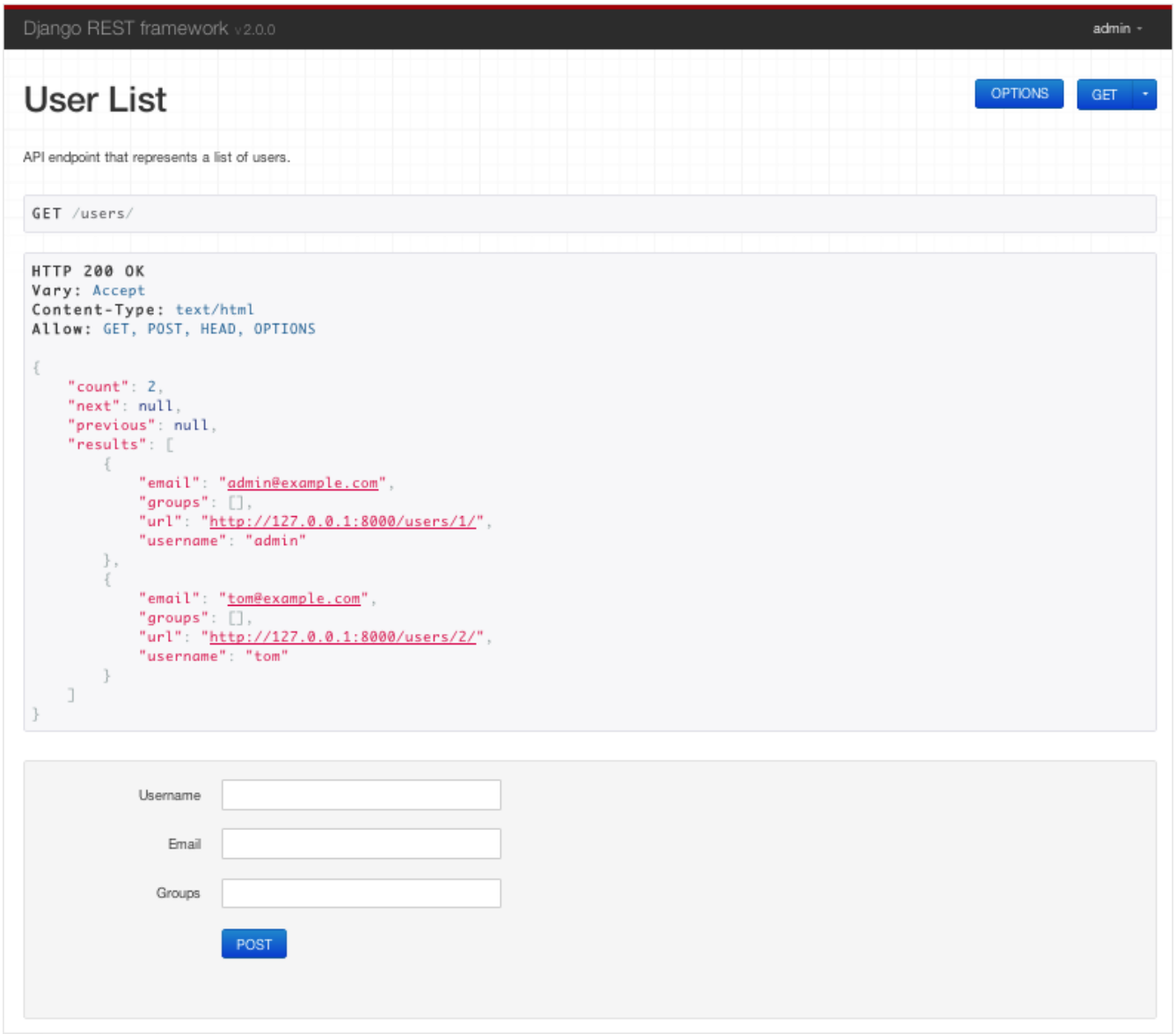
.media_type： `text/html`

.format： `'html'`

.charset： `utf-8`

BrowsableAPIRenderer

通过这个渲染器可以将数据渲染为HTML页面，提供可浏览的API页面:



BrowsableAPIRenderer会探测哪个其他渲染器被赋予了最高优先级，并使用该渲染器在HTML页面中显示API样式。

.media_type： `text/html`

.format： `'api'`

.charset： `utf-8`

.template： `'rest_framework/api.html'`

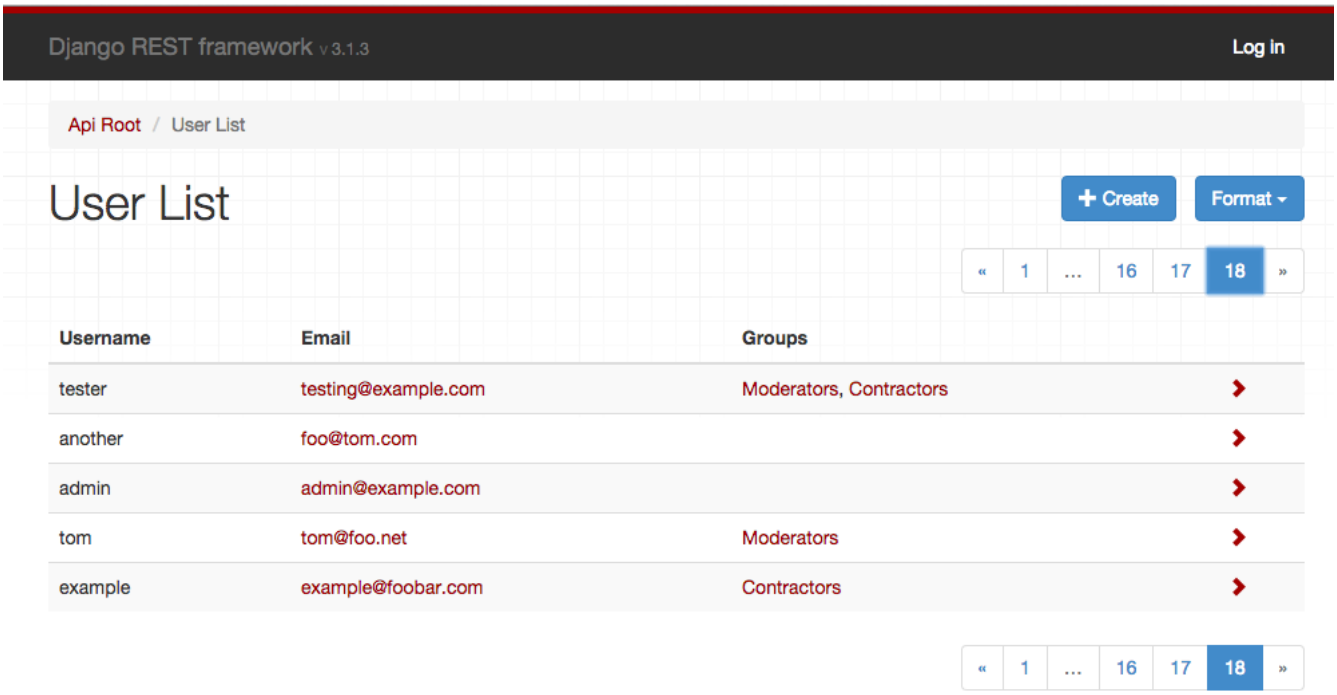
默认情况下，响应内容将使用除 `BrowsableAPIRenderer` 外，最高优先级的渲染器进行渲染。如果您需要自定义此行为，例如使用HTML作为默认返回格式，但在可浏览API中使用JSON。

可以采用下面的解决方法：覆盖 `get_default_renderer()` 方法。例如：

```
class CustomBrowsableAPIRenderer(BrowsableAPIRenderer):
    def get_default_renderer(self, view):
        return JSONRenderer()
```

AdminRenderer

将数据渲染为类似管理员后台界面的HTML页面，如下图所示：



请注意，具有嵌套序列化器或序列化器列表的视图将无法正常使用 `AdminRenderer` ，因为HTML表单无法正确支持它们。

注意： `AdminRenderer` 只有在数据中正确配置 `URL_FIELD_NAME` （ `url` 默认情况下）属性时，才能包含指向详细信息页面的链接。对于 `HyperlinkedModelSerializer` 无需担心，但是对于 `ModelSerializer` 或普通 `Serializer` 类，需要明确包含该字段。例如，我们使用模型 `get_absolute_url` 方法：

```
class AccountSerializer(serializers.ModelSerializer):
    url = serializers.CharField(source='get_absolute_url', read_only=True)

    class Meta:
        model = Account
```

.media_type： `text/html`

.format： `'admin'`

.charset： `utf-8`

.template： `'rest_framework/admin.html'`

HTMLFormRenderer

将序列化程序返回的数据呈现为HTML表单。此渲染器的输出不包括封闭的 `<form></form>` 标签，隐藏的CSRF标签或任何提交按钮，这些都要你自己写。

此渲染器不建议直接使用，而是通过将序列化器的实例传递给 `render_form` 模板标记来在模板中使用。

```
{% load rest_framework %}

<form action="/submit-report/" method="post">
    {% csrf_token %}
    {% render_form serializer %}
    <input type="submit" value="Save" />
</form>
```

.media_type： `text/html`

.format： `'form'`

.charset： `utf-8`

.template： `'rest_framework/horizontal/form.html'`

MultiPartRenderer

此渲染器用于渲染HTML分成多个部分的表单数据。 **它不适合作为响应的渲染器**，而是用于创建测试请求，使用REST框架的测试客户端和测试请求工厂方法。了解即可，绝大多数场景下用不着。

.media_type： `multipart/form-data; boundary=BoUnDaRyStRiNg`

.format： `'multipart'`

.charset： `utf-8`

三、自定义渲染器

要自定义渲染器，首先要继承 `BaseRenderer` 类，然后设置 `.media_type` 和 `.format` 属性，最后实现 `.render(self, data, media_type=None, renderer_context=None)` 方法。

render()方法应返回一个bytestring对象，它将用作HTTP响应的主体。

`.render()` 方法的参数是：

- `data`

请求数据，由 `Response()` 实例化。

- `media_type=None`

可选参数。如果提供，则是内容协商的媒体类型。

- `renderer_context=None`

可选参数。如果提供，则是由视图提供的上下文信息的字典。

默认情况下，这个字典会包括以下的键： `view` ， `request` ， `response` ， `args` ， `kwargs` 。

范例：

自定义一个纯文本渲染器，它将返回一个将 `data` 参数作为内容的响应。

```
from django.utils.encoding import smart_unicode
from rest_framework import renderers

class PlainTextRenderer(renderers.BaseRenderer):
    media_type = 'text/plain'
    format = 'txt'

    def render(self, data, media_type=None, renderer_context=None):
        return data.encode(self.charset)
```

设置字符集

默认情况下，渲染器类使用 `UTF-8` 编码。要使用其他编码，请在渲染器上设置 `charset` 属性。

```
class PlainTextRenderer(renderers.BaseRenderer):
    media_type = 'text/plain'
    format = 'txt'
    charset = 'iso-8859-1'

    def render(self, data, media_type=None, renderer_context=None):
        return data.encode(self.charset)
```

请注意，如果渲染器返回unicode字符串，则响应内容将被强制转换为字节字符串 `Response` ，并在渲染器上设置 `charset` 属性以确定编码。

如果渲染器返回表示原始二进制内容的字节字符串，则应将字符集值设置为 `None` ，这将确保 `Content-Type` 响应的标头不会 `charset` 设置值。

在某些情况下，可能需要将 `render_style` 属性设置为 `'binary'` 。这样做将确保可浏览的API不会尝试将二进制内容显示为字符串。

```
class JPEGRenderer(renderers.BaseRenderer):
    media_type = 'image/jpeg'
    format = 'jpg'
    charset = None
    render_style = 'binary'

    def render(self, data, media_type=None, renderer_context=None):
        return data
```

四、渲染器高级用法

可以使用REST框架的渲染器执行一些非常灵活的操作。比如：

- 根据请求的媒体类型，提供来自同一API的平坦或嵌套表示。
- 同时提供常规HTML网页和基于JSON的API响应。
- 为要使用的API客户端指定多种类型的HTML表示。
- 取消指定渲染器的媒体类型，例如使用 `media_type = 'image/*'` ，并使用 `Accept` 标头来改变响应的编码。

媒体类型的不同行为

在某些情况下，您可能希望视图使用不同的序列化器，具体取决于接受的媒体类型。如果需要这样做，你可以访问 `request.accepted_renderer` 以确定将用于响应的渲染器。

例如：

```
@api_view(('GET',))
@renderer_classes((TemplateHTMLRenderer, JSONRenderer))
def list_users(request):
    """
    A view that can return JSON or HTML representations
    of the users in the system.
    """
    queryset = Users.objects.filter(active=True)

    if request.accepted_renderer.format == 'html':
        # 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')

    # JSONRenderer requires serialized data as normal.
    serializer = UserSerializer(instance=queryset)
    data = serializer.data
    return Response(data)
```

未指定媒体类型

在某些情况下，您可能希望渲染器同时支持一系列媒体类型。在这种情况下，可以将 `media_type` 设置为 `*/*` 。

如果未指定渲染器的媒体类型，则应确保在使用该 `content_type` 属性返回响应时明确指定媒体类型。例如：

```
1   return Response(data, content_type='image/png')
```

HTML错误视图

通常情况下，渲染器的行为都是相同的，无论它是处理常规响应，还是由异常引起的响应（如 `Http404` 、 `PermissionDenied` 或 `APIException` 的子类）。

引发和处理的异常的HTML页面将尝试按优先顺序使用以下方法之一进行选择。

- 加载并渲染名为 `{status_code}.html` 的模板。
- 加载并渲染名为 `api_exception.html` 的模板。
- 返回HTTP状态代码和文本，例如“404 Not Found”。

模板将使用 `RequestContext` 包含 `status_code` 和 `details` 键的内容进行渲染。

注意：如果 `DEBUG=True` ，将显示Django的标准回溯错误页面，而不是呈现HTTP状态代码和文本。

五、第三方模块

DRF中，通过以下的第三方模块实现功能更强大的渲染器扩展：

YAML

它以前直接包含在REST框架中，现在作为第三方模块。

使用pip安装。

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

修改REST框架设置。

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

XML

一种简单的非正式XML格式。它以前直接包含在REST框架中，现在作为第三方模块。

使用pip安装。

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

修改REST框架设置。

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

JSONP

提供JSONP渲染支持。它以前直接包含在REST框架中，现在作为第三方模块。

警告：如果您需要跨域AJAX请求，通常应该使用更现代的CORS方法作为 `JSONP` 的替代方案。有关更多详细信息，请参阅**CORS文档**。

使用pip安装。

```
$ pip install djangorestframework-jsonp
```

修改REST框架设置。

```
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework_jsonp.renderers.JSONPRenderer',
    ),
}
```

XLSX

Microsoft Office Excle的数据格式。

使用pip安装。

```
$ pip install drf-renderer-xlsx
```

修改REST框架设置。

```
REST_FRAMEWORK = {
    ...

    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
        'drf_renderer_xlsx.renderers.XLSXRenderer',
    ),
}
```

为了避免文件流没有文件名（浏览器的默认文件名为“download”，没有扩展名），我们需要使用mixin来覆盖 `Content-Disposition` 标题。如果没有提供文件名，则默认为 `export.xlsx` 。例如：

```
from rest_framework.viewsets import ReadOnlyModelViewSet
from drf_renderer_xlsx.mixins import XLSXFileMixin
from drf_renderer_xlsx.renderers import XLSXRenderer

from .models import MyExampleModel
from .serializers import MyExampleSerializer

class MyExampleViewSet(XLSXFileMixin, ReadOnlyModelViewSet):
    queryset = MyExampleModel.objects.all()
    serializer_class = MyExampleSerializer
    renderer_classes = (XLSXRenderer,)
    filename = 'my_export.xlsx'
```

Pandas（CSV，Excel，PNG）

Django REST Pandas提供了一个序列化器和渲染器，支持使用Pandas DataFrame API进行额外的数据处理和输出。Django REST Pandas这个模块包括用于Pandas风格的CSV文件、Excel工作簿（`.xls`和`.xlsx`）以及许多其他格式的渲染器。

此外，通过第三方模块，还可以支持CSV、LaTeX、UltraJSON、camel case JSON和MessagePack等格式。