源代码位于：response.py

REST framework 提供一个 `Response` 类来支持 HTTP内容协商，该类允许返回可以呈现为多种类型的内容，具体取决于客户端的请求。

这个 `Response` 类是 Django中 `SimpleTemplateResponse` 类的一个子类。 `Response` 对象使用Python原生的数据类型进行初始化。 然后REST framework 使用标准的HTTP内容协商来确定如何呈现最终的响应内容。

我们并不一定非要使用DRF的 `Response` 类进行响应，也可以返回常规的 `HttpResponse` 或者 `StreamingHttpResponse` 对象，但是使用 `Response` 类可以提供一个多种格式的更漂亮的界面。

除非由于某种原因你要对 REST framework 做大量的自定义，否则你应该始终使用 `APIView` 类或者 `@api_view` 函数这些DRF提供的视图。这样做可以确保视图在返回之前能够执行 内容协商并且为响应选择适当的渲染器。

---

# 创建 responses

## Response()

---

**签名:** `Response(data, status=None, template_name=None, headers=None, content_type=None)`

与常规的 `HttpResponse` 对象不同，我们不使用渲染过的内容来实例化一个 `Response` 对象，而是传递未渲染的数据，这些数据可以是任何Python基本数据类型。

`Response` 类使用的渲染器无法自行处理像 Django模型实例这样的复杂数据类型，因此你需要在创建 `Response` 对象之前将数据序列化为基本数据类型，比如json。

1 注：这段话我们要深刻理解一下！由于前后端分离了，前端页面的代码不是后端人员写的，前端不知道django模型（甚至不知道后端是什么语言、什么框架、什么服务、什么数据库），不知道数据库的表结构，没有Python中model的代码签名，无法自己反序列化一个Django的模型的对象，就像Python的内置json模块无法序列化一个自定义的Python类一样。

2 然而，由于API形式的前后端分离，通过HTTP传输的内容格式通常都是json或者xml这些类型。为了将Python，也就是Django中的模型中的具体数据内容可以json化，我们需要自己写serializer，也就是序列化器，进行模型model对象的序列化和反序列化。如果不这么做，那前端看着后端发来的数据会懵逼的，后端拿着前端post过来的数据也不知道该怎么存到数据库里去。

3 这完全不同于Django本身的全栈式开发模式，因为在前后端都是一个人编写的情况下，编写者掌握全局，无论前端页面的渲染还是后端数据库的CRUD都由此人掌控，原理上随便怎么定义数据传输接口都可以。

你可以使用 REST framework的 `Serializer` 类来执行此类数据的序列化，或者使用你自定义的序列化器。

参数说明:

- `data` : 要响应的已经序列化了的数据
- `status` : 响应的状态码。默认是200。
- `template_name` : 当选择 `HTMLRenderer` 渲染器时，指定要使用的模板的名称。
- `headers` : 一个字典，包含响应的HTTP头部信息。
- `content_type` : 响应的内容类型。通常由渲染器自行设置，由协商内容确定，但是在某些情况下，你需要明确指定内容类型。

# 属性

## .data

未渲染的，已经序列化的要响应的数据

## .status_code

HTTP 响应的数字状态码。

## .template_name

`template_name` 只有在使用 `HTMLRenderer` 或者其他自定义模板作为响应的渲染器时才具有该属性。

---

# 标准的HttpResponse 属性

DRF的 `Response` 类扩展了 Django原生的 `SimpleTemplateResponse` ，所有原生的属性和方法都是提供的。比如你可以使用标准的方法设置响应的header信息：

```
1   response = Response()
2
3   response['Cache-Control'] = 'no-cache'
```

## .render()

和其他的 `TemplateResponse` 一样，调用该方法将序列化的数据渲染为最终的响应内容。当 `.render()` 被调用时， 响应的内容将被设置成在 `accepted_renderer` 实例上调用 `.render(data, accepted_media_type, renderer_context)` 方法返回的结果。

我通常并不需要自己调用 `.render()` ，因为它是由Django的标准响应过程来处理的。

**我们只要记住下面几个知识点就可以了：**

- 扩展了Django原生的SimpleTemplateResponse
- 可以使用标准的方法设置响应的头部信息
- 会根据内容协商的结果自动渲染成指定的类型
- 执行render()将序列化的数据渲染为最终响应的内容
- 始终使用DRF提供的视图系统，并调用DRF提供的Response