源代码位于：request.py

对于DRF的request对象，有下面的主要属性：

- .data
- .query_params
- .parsers
- .accepted_renderer
- .accepted_media_type
- .user
- .auth
- .authenticators
- .method
- .content_type
- .stream

---

> 如果你正在做基于REST的Web服务...你最好忽略request.POST。
>
> — Malcom Tredinnick, **Django 开发组成员**

REST framework的 `Request` 类扩展了Django标准的 `HttpRequest` ，添加了对REST framework请求解析和身份验证的支持。

源代码片段：

```
class Request(object):
    """
    Wrapper allowing to enhance a standard `HttpRequest` instance.

    Kwargs:
        - request(HttpRequest). The original request instance.
        - parsers_classes(list/tuple). The parsers to use for parsing the
          request content.
        - authentication_classes(list/tuple). The authentications used to try
          authenticating the request's user.
    """

    def __init__(self, request, parsers=None, authenticators=None,
                 negotiator=None, parser_context=None):
```

```
15          assert isinstance(request, HttpRequest), (
16              'The `request` argument must be an instance of '
17              '`django.http.HttpRequest`, not `{}.{}`.'
18              .format(request.__class__.__module__,
    request.__class__.__name__)
19          )
20
21          self._request = request
22          self.parsers = parsers or ()
23          self.authenticators = authenticators or ()
24          self.negotiator = negotiator or self._default_negotiator()
25          self.parser_context = parser_context
26          self._data = Empty
27          self._files = Empty
28          self._full_data = Empty
29          self._content_type = Empty
30          self._stream = Empty
```

# 请求解析相关

REST framework的request请求对象以通常处理表单数据相同的方式使用JSON数据或其他媒体类型处理请求。下面的属性是request中数据主体的部分。

## .data

`request.data` 返回请求正文的解析内容，这与标准的 `request.POST` 和 `request.FILES` 属性类似，除了下面的区别：

- `request.data` 包含所有解析的内容,包括 文件或非文件输入。
- `request.data` 支持除 `POST` 之外的HTTP方法，这意味着你可以访问 `PUT` 和 `PATCH` 请求的内容。
- `request.data` 支持更灵活的请求解析，而不仅仅是表单数据。 例如，你可以与处理表单数据相同的方式处理传入的JSON数据。

```
1       @property
2       def data(self):
3           if not _hasattr(self, '_full_data'):
4               self._load_data_and_files()
5           return self._full_data
```

## .query_params

`request.query_params` 是 `request.GET` 的一个更准确的同义词，也就是在DRF中的替代品。

为了让你的代码清晰明了，我们建议使用 `request.query_params` 而不是Django标准的 `request.GET` 。这样做有助于保持代码库更加正确和明了。通俗地说，就是不要把DRF的属性/方法名和Django原生的属性/方法名混用。

```
    def query_params(self):
        """
        More semantically correct name for request.GET.
        """
        return self._request.GET
```

## .parsers

`APIView` 类或 `@api_view` 装饰器将根据view中设置的 `parser_classes` 值或 `DEFAULT_PARSER_CLASSES` 配置参数进行设置，确保此属性自动设置为 `Parser` 实例列表。通常并不需要访问这个属性。

---

**Note:** 如果客户端发送格式错误的内容，则访问 `request.data` 可能会引发 `ParseError` 。默认情况下REST framework的 `APIView` 类或 `@api_view` 装饰器将捕获错误并返回 `400 Bad Request` 响应。

如果客户端发送具有无法解析的内容类型（content-type）的请求，则会引发 `UnsupportedMediaType` 异常，默认情况下会捕获该异常并返回 `415 Unsupported Media Type` 响应。

---

# 内容协商相关

request请求对象还提供了一些属性允许你确定内容协商阶段的结果。这允许你实现具体的行为，例如为不同的媒体类型选择不同的序列化方案。

## .accepted_renderer

内容协商阶段选择的renderer实例。

## .accepted_media_type

内容协商阶段接受的媒体类型的字符串。

# 认证相关

requset对象提供了灵活的，每次请求都进行验证的认证功能，并支持下面的特性：

- 对API的不同部分使用不同的身份验证策略。
- 支持同时使用多种身份验证策略
- 提供与传入请求相关联的用户和令牌信息

## .user

`request.user` 默认情况下使用的是Django自带的auth框架的用户模型，返回一个 `django.contrib.auth.models.User` 实例。但该行为取决于你所使用的的认证策略，很显然当你使用第三方认证模块时，就不一样了。

如果请求未认证则 `request.user` 的默认值为 `django.contrib.auth.models.AnonymousUser` 的一个实例。

## .auth

`request.auth` 返回所有附加的身份验证上下文。 其实际行为取决于所使用的具体认证策略。如果请求未认证或者没有其他上下文，则 `request.auth` 的默认值为 `None` 。

## .authenticators

使用的认证策略。通常并不需要访问此属性。

```
    def _authenticate(self):
        """
        Attempt to authenticate the request using each authentication instance
        in turn.
        """
        for authenticator in self.authenticators:
            try:
                user_auth_tuple = authenticator.authenticate(self)
            except exceptions.APIException:
                self._not_authenticated()
                raise

            if user_auth_tuple is not None:
                self._authenticator = authenticator
                self.user, self.auth = user_auth_tuple
                return

        self._not_authenticated()
```

# 浏览器相关

REST framework 支持一些浏览器增强功能，例如基于浏览器的 `PUT`，`PATCH` 和 `DELETE` 表单功能。

下面是request对象中关于浏览器增强相关的一些属性：

## .method

`request.method` 返回请求的HTTP方法的 **大写** 字符串表示形式。

隐含支持基于浏览器的 `PUT`，`PATCH` 和 `DELETE` 方法。

## .content_type

`request.content_type` 返回HTTP请求正文的媒体类型的字符串对象，如果未提供媒体类型，则返回空字符串。也就是正文的格式。

一般我们不使用这个属性，但如果真要在DRF中访问请求的内容类型，请务必使用 `.content_type` 属性，而不是使用 `request.META.get('HTTP_CONTENT_TYPE')`，因为前者为基于浏览器的非表单内容提供了隐含的支持。

## .stream

`request.stream` 返回请求主体内容的流形表示。

我们通常不需要直接访问请求的内容主体，而是依赖REST framework默认的请求解析行为。

# request._request

**如果你强烈需要使用Django原生的request对象，请这么调用：** `request._request`

由于 REST framework 的 `Request` 对象扩展了 Django的 `HttpRequest` 对象，所以所有Django原生的标准属性和方法也是可用的。例如 `request.META` 和 `request.session` 字典正常可用。

请注意， DRF的 `Request` 类并不是直接继承Django原生的 `HttpRequest` 类，而是使用合成扩展出来的。