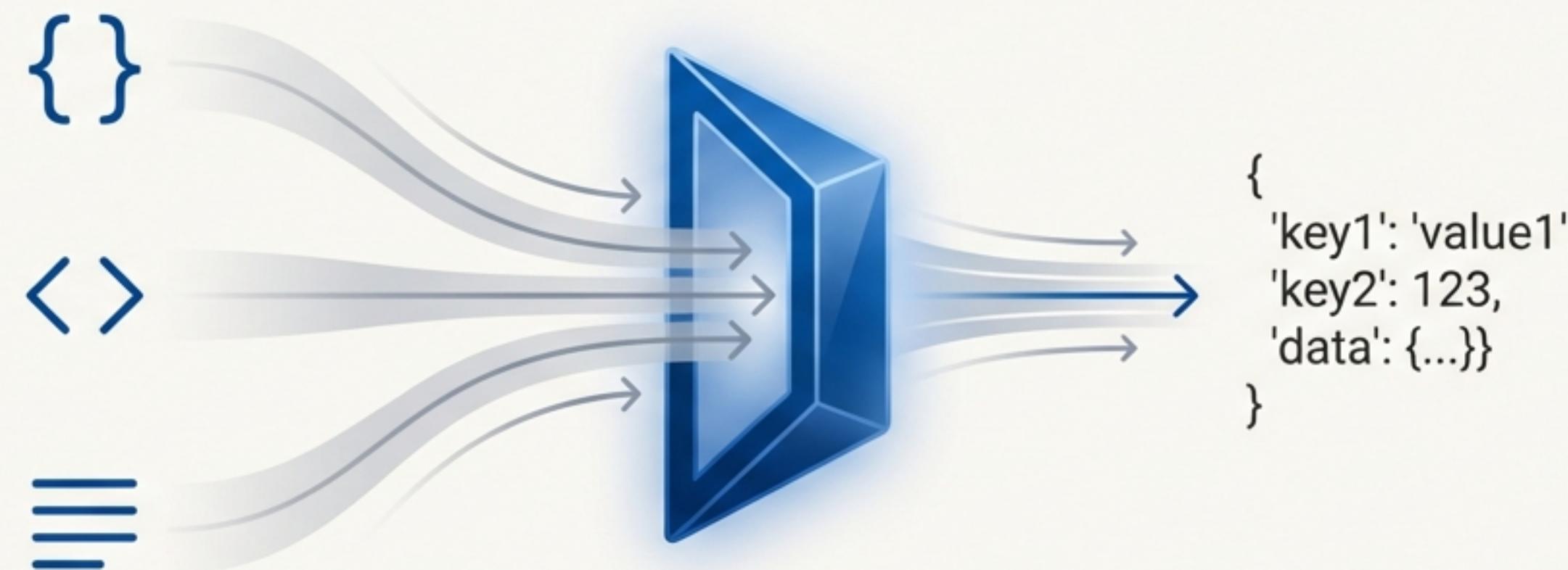


# پارسِرها در Django REST Framework دروازه ورودی داده‌های شما

راهنمای کامل برای درک، پیکربندی و ساخت پارسِرهای سفارشی برای API شما.



## چرا به پارسِرها نیاز داریم؟ فراتر از فرم‌های HTML

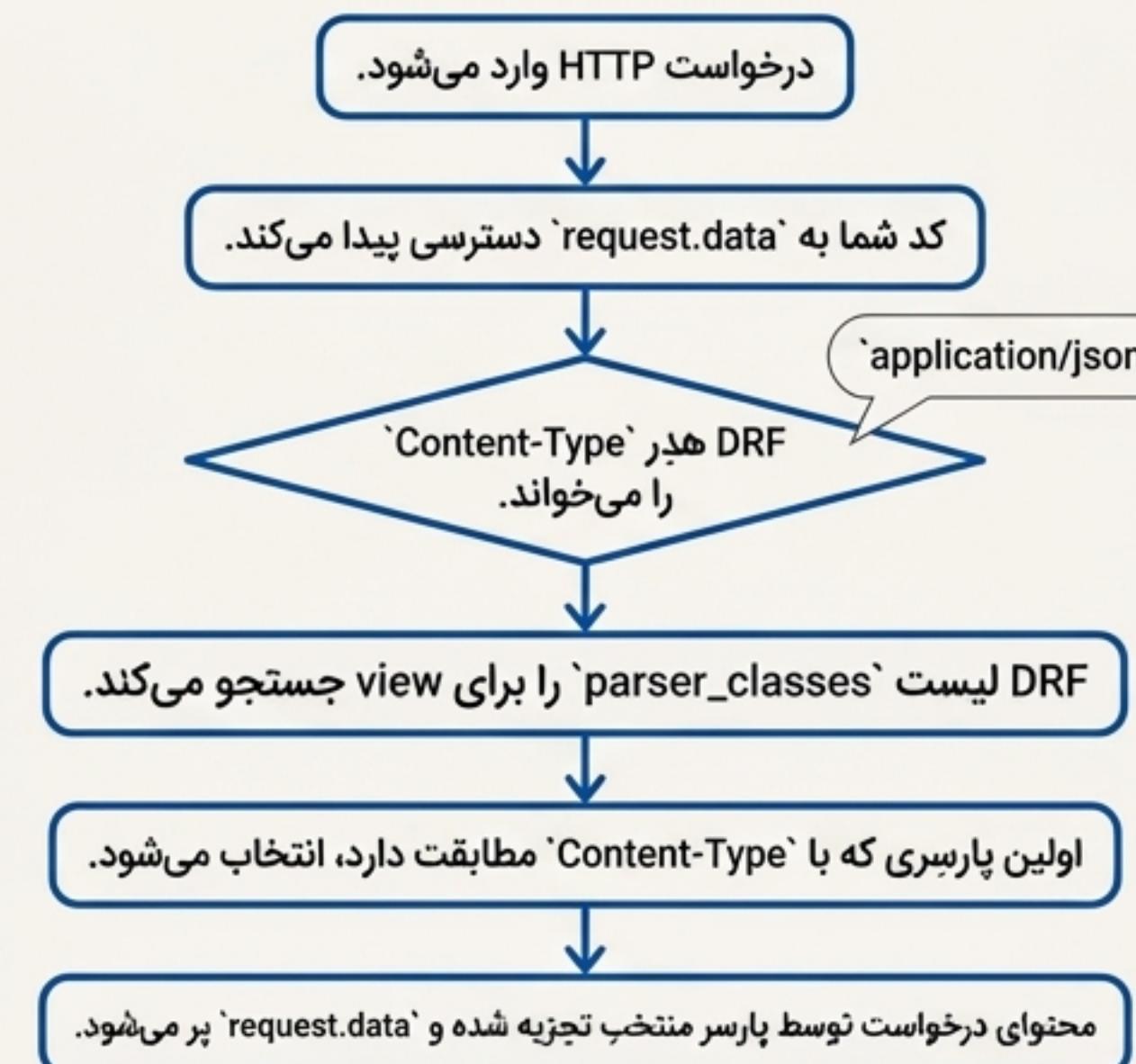
سرویس‌های وب که ماشین‌ها با آن‌ها تعامل دارند، تمایل دارند از فرم‌ت‌های ساختاری‌افته‌تری برای ارسال داده‌ها نسبت به فرم‌های کدگذاری‌شده استفاده کنند، زیرا داده‌های پیچیده‌تری نسبت به فرم‌های ساده ارسال می‌کنند.

– گروه توسعه‌دهندگان جنگو –

- Parser داخلی است که به شما امکان می‌دهد Django REST Framework (DRF) شامل مجموعه‌ای از کلاس‌های درخواست‌ها را با انواع مختلف رسانه (media types) بپذیرید.
- این چارچوب همچنین از تعریف پارسِرهای سفارشی پشتیبانی می‌کند و به شما انعطاف‌پذیری لازم برای طراحی انواع رسانه‌ای که API شما می‌پذیرد را می‌دهد.

# فرآیند انتخاب پارسِر: مذاکره بر اساس Content-Type

مجموعه پارسِرهای معتبر برای یک view همیشه به صورت لیستی از کلاس‌ها تعریف می‌شود. هنگامی که به `request.data` دسترسی پیدا می‌کنید، DRF ۱ درخواست ورودی را بررسی کرده و تصمیم می‌گیرد از کدام پارسِر برای تجزیه محتوای درخواست استفاده کند.



# یک نکته کلیدی برای توسعه‌دهندگان کلاینت



## همیشه هدر `Content-Type` را تنظیم کنید!

هنگام ارسال داده در یک درخواست HTTP، حتماً هدر `Content-Type` را تنظیم کنید. اگر این کار را نکنید، نکنید، اکثر کلاینت‌ها به طور پیش‌فرض از 'application/x-www-form-urlencoded' استفاده می‌کنند، که ممکن است مد نظر شما نباشد.

برای ارسال داده‌های کدگذاری شده به صورت json در jQuery `\$.ajax()` با استفاده از متده استفاده از متد `json` با صورت `contentType: 'application/json'` را لحاظ کنید.

```
$.ajax({
  url: '/api/data/',
  type: 'POST',
  contentType: 'application/json', // This is crucial!
  data: JSON.stringify({ key: 'value' }),
  success: function(response) { ... }
});
```

## پیکربندی پارسِرها (بخش ۱): رویکرد سراسری

شما می‌توانید مجموعه پیش‌فرض پارسِرها را به صورت سراسری با استفاده از تنظیمات `DEFAULT\_PARSER\_CLASSES` در فایل `settings.py` خود تعیین کنید. کد زیر، API شما را طوری پیکربندی می‌کند که فقط درخواست‌های با محتوای `JSON` را محتوای `JSON` را بپذیرد و جایگزین تنظیمات پیش‌فرض (که شامل JSON و داده‌های فرم است) می‌شود.

```
# settings.py

REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES': [
        'rest_framework.parsers.JSONParser',
    ]
}
```

نکته\*\* این روش برای اعمال یک سیاست یکسان در تمام نقاط پایانی (endpoints) API شما ایده‌آل است.

# View پیکربندی پارسِرها (بخش ۲): رویکرد مبتنی بر

برای انعطاف‌پذیری بیشتر، می‌توانید پارسِرها را برای یک view خاص با استفاده از خصوصیت `parser\_classes` تنظیم کنید.

## Class-Based Views (`APIView`)

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

class ExampleView(APIView):
    """
    یک view که می‌تواند درخواست‌های POST با محتوای JSON را بپذیرد.
    """

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

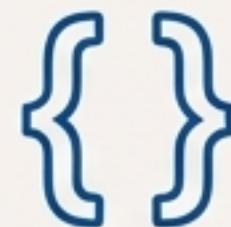
## Function-Based Views (`@api\_view`)

```
from rest_framework.decorators import api_view, parser_classes
from rest_framework.parsers import JSONParser
from rest_framework.response import Response

@api_view(['POST'])
@parser_classes([JSONParser])
def example_view(request, format=None):
    """
    یک view که می‌تواند درخواست‌های POST با محتوای JSON را بپذیرد.
    """

    return Response({'received data': request.data})
```

# جعبه ابزار داخلی DRF: آشنایی با پارسرهای پیشفرض



• برای داده‌های API‌ها. رایج‌ترین فرمت application/json.



• برای داده‌های فرم HTML سنتی (application/x-www-form-urlencoded) (برای داده‌های فرم .form-urlencoded)



• برای داده‌های فرم چندبخشی (multipart/form-data) که شامل آپلود فایل نیز می‌شود.



• برای آپلود فایل به صورت خام و بدون فرم.

# پارسراصلی API های مدرن: `JSONParser`

- محتوای درخواست JSON را تجزیه می کند.
- با یک دیکشنری از داده ها پر می شود.

**media\_type**

application/json

# `FormParser` و `MultiParser` : HTML فرم‌هایی کامل

## `MultiPartParser`



برای پشتیبانی کامل از داده‌های فرم HTML، معمولاً باید از هر دو `FormParser` و `MultiPartParser` با هم استفاده کنید.

### `FormParser`

- **وظیفه:** محتوای فرم HTML را تجزیه می‌کند.
- `request.data` از `QueryDict` با یک داده‌ها پر می‌شود.
- `media_type`: "application/x-www-form-urlencoded"

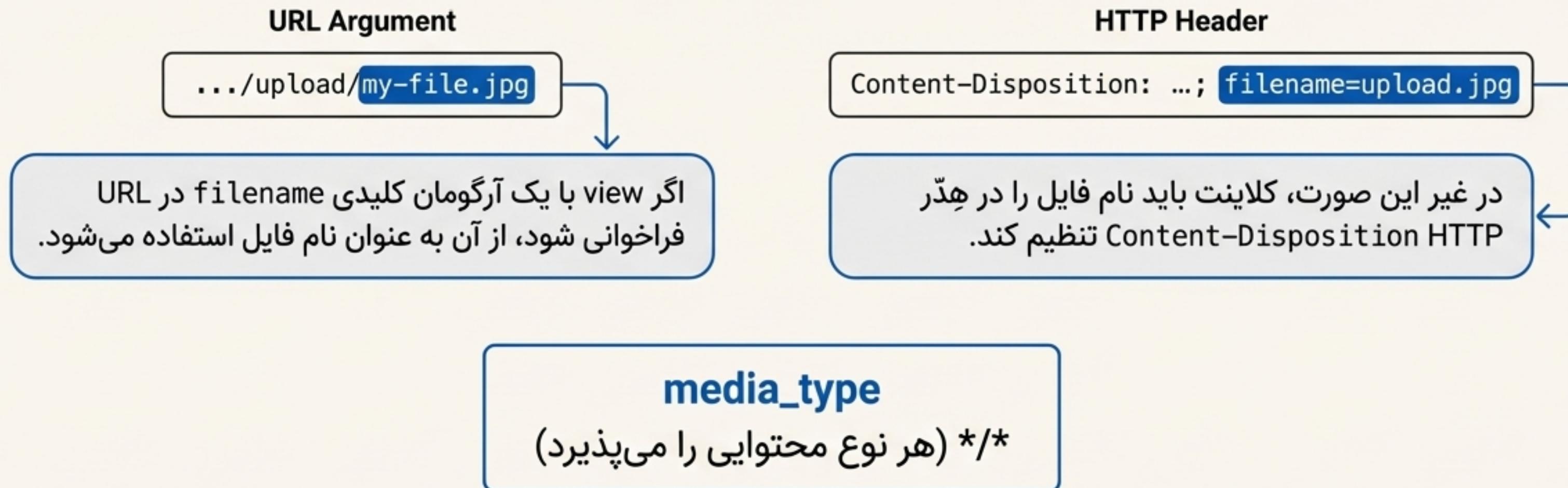
### `MultiPartParser`

- **وظیفه:** محتوای فرم چندبخشی HTML را تجزیه می‌کند که از آپلود فایل پشتیبانی می‌کند.
- `request.data` و `request.FILES` به ترتیب با یک `MultiValueDict` و `QueryDict` پر می‌شوند.
- `media_type`: "multipart/form-data"

# آپلود فایل خام: آشنایی با `FileUploadParser`

- محتوای آپلود فایل خام را تجزیه می‌کند.
- خصوصیت `request.data` یک دیکشنری با یک کلید واحد به نام 'file' خواهد بود که حاوی فایل آپلود شده است.

نحوه تعیین نام فایل:



# نکات مهم و مثال کاربردی `FileUploadParser`



- این پارسر برای کلاینت‌های نیتیو (native) است که می‌توانند فایل را به عنوان یک درخواست داده خام آپلود کنند. برای آپلودهای مبتنی بر وب، از `MultiPartParser` استفاده کنید.
- از آنجایی که `media\_type` این پارسر با هر نوع محتوایی مطابقت دارد، معمولاً باید تنها پارسر تنظیم شده روی یک API view باشد.
- این پارسر از تنظیمات استاندارد جنگو `FILE\_UPLOAD\_HANDLERS` همان‌ادر' پیروی می‌کند.

## مثال کد کامل

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

    def put(self, request, filename, format=None):
        file_obj = request.data['file']
        # ... do some stuff with uploaded file ...
        return Response(status=204)

# urls.py
urlpatterns = [
    # ...
    re_path(r'^upload/(?P<filename>[^/]+)/$', FileUploadView.as_view())
]
```

# فراتر از پیشفرضها: ساخت پارسِر سفارشی خودتان

برای پیادهسازی یک پارسِر سفارشی، باید از `BaseParser` ارث بری کنید، خصوصیت `media\_type` را تنظیم کنید، و متد `parse()` را پیادهسازی نمایید.

1.

ارث بری

```
class MyCustomParser(Base  
Parser):
```

2.

Media Type تعیین

```
media_type =  
'application/my-custom-  
type'
```

3.

پیادهسازی منطق

```
def parse(self, stream,  
media_type,  
parser_context):
```

**نتیجه:** متد `parse()` باید داده‌هایی را برگرداند که برای پر کردن خصوصیت `request.data` استفاده خواهد شد.

# `parse()` آناتومی یک پارسِر سفارشی: آرگومان‌های متد()

```
def parse(self, stream, media_type=None, parser_context=None):
```

## stream

یک شیء شبه-جريان (stream-like object)، که بدنه درخواست را نشان می‌دهد.

## media\_type (اختیاری)

نوع رسانه محتوای درخواست ورودی. بسته به هدر Content-Type ممکن است جزئیات بیشتری نسبت به خصوصیت `media\_type` پارسِر داشته باشد (مثال: `text/plain; charset=utf-8`).

## parser\_context (اختیاری)

یک دیکشنری حاوی هرگونه زمینه اضافی مورد نیاز برای تجزیه محتوا. به طور پیش‌فرض شامل کلیدهای زیر است: `view`, `request`, `args`, `kwargs`.

## مثال عملی: ساخت یک `PlainTextParser` ساده

در ادامه، یک پارسِر متن ساده (plaintext) را با یک رشته که خصوصیت `request.data` را مشاهده می‌کنید که نمایانگر بدن درخواست است، پر می‌کند.

```
from rest_framework.parsers import BaseParser

class PlainTextParser(BaseParser):
    """
    Plain text parser.
    """
    media_type = 'text/plain'

    def parse(self, stream, media_type=None, parser_context=None):
        """
        Simply return a string representing the body of the request.
        """
        return stream.read()
```

**کاربرد:** این پارسِر برای API‌هایی که نیاز به دریافت داده‌های متنی خام دارند، مانند لاغ‌ها یا پیام‌های ساده، ایده‌آل است.

# گسترش قابلیت‌ها: اکوسیستم پکیج‌های شخص ثالث

پکیج‌های شخص ثالث زیر برای پشتیبانی از فرمات‌های داده بیشتر در دسترس هستند.

## YAML

`djangorestframework-yaml`

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

```
'DEFAULT_PARSER_CLASSES':  
    ['rest_framework_yaml.parsers.YAMLPParser'],
```

## XML

`djangorestframework-xml`

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

```
'DEFAULT_PARSER_CLASSES':  
    ['rest_framework_xml.parsers.XMLParser'],
```

## MessagePack

`djangorestframework-msgpack`

یک فرمات سریال‌سازی باینری سریع و کارآمد.

کنید، اما آن‌ها را به صورت camelCase در API نمایش دهید.

## CamelCase JSON

`djangorestframework-camel-case`

به شما امکان می‌دهد از نام‌های فیلد underscored در پایتون استفاده کنید، اما آن‌ها را به صورت camelCase در API نمایش دهید.