一、分页Pagination

DRF框架允许自定制分页样式，也就是说你可以设置每页显示的对象数量。

分页API支持以下操作：

- 将分页链接作为响应内容的一部分。
- 响应头中包含分页链接，如 `Content-Range` 或 `Link` 。

只有在使用通用视图或视图集时才会自动执行分页。如果您使用的是常规APIView，则需要自己调用分页API，以确保响应中包含分页数据。

可以通过将分页类设置为None来关闭分页功能。

设置分页风格

可以使用 `DEFAULT_PAGINATION_CLASS` 和 `PAGE_SIZE` 进行全局性的分页风格设置。例如下面的例子:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.LimitOffsetPagination',
    'PAGE_SIZE': 100
}
```

注意， `DEFAULT_PAGINATION_CLASS` 和 `PAGE_SIZE` 必须同时设置。**默认情况下，它们都是 `None` ，也就是不分页。**

也可以为单个的视图添加 `pagination_class` 属性，设置视图级别的分页风格。

修改分页风格

一般我们通过继承现有分页类，然后设置类属性的方式来修改分页风格，其实也就是相当于自定义分页类了。

```python
class LargeResultsSetPagination(PageNumberPagination): # 继承
    page_size = 1000   # 记住这几个可以配置的属性的名字
    page_size_query_param = 'page_size'
    max_page_size = 10000

class StandardResultsSetPagination(PageNumberPagination):
    page_size = 100
    page_size_query_param = 'page_size'
    max_page_size = 1000
```

然后你就可以在视图中添加 `pagination_class` 属性，并指定为你刚才自定义的分页类了，如下所示：

```python
class BillingRecordsView(generics.ListAPIView):
    queryset = Billing.objects.all()
    serializer_class = BillingRecordsSerializer
    pagination_class = LargeResultsSetPagination # 看这里
```

或者在 `DEFAULT_PAGINATION_CLASS` 中进行全局性配置：

```python
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS':
'apps.core.pagination.StandardResultsSetPagination'
}
```

二、API参考

在DRF的pagination模块中，定义了四个分页类：

- BasePagination：基类
- PageNumberPagination：继承了BasePagination
- LimitOffsetPagination：继承了BasePagination
- CursorPagination：继承了BasePagination

没事可以读一读它的源码，看看别人是怎么写分页功能的。

PageNumberPagination

这种分页风格接收一个url中的页码参数。

请求:

```
GET https://api.example.org/accounts/?page=4
```

响应:

```
HTTP 200 OK
{
    "count": 1023  # 数量
    "next": "https://api.example.org/accounts/?page=5",  #上一页url
    "previous": "https://api.example.org/accounts/?page=3", # 下一页url
    "results": [  # 具体的结果
       …
    ]
}
```

配置

要使用 `PageNumberPagination` 分割，需要进行下面的配置：

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 100
}
```

对于继承了 `GenericAPIView` 的类视图，你也可以通过设置 `pagination_class` 属性选择 `PageNumberPagination` 作为视图级别的分页功能。

属性

`PageNumberPagination` 类包含一系列属性用于设置分页风格：

要启用这些属性，你需要继承 `PageNumberPagination` 类，然后激活你自定义的子类，并添加类属性：

- `django_paginator_class` - 使用Django的分页类。默认是 `django.core.paginator.Paginator`，它适用于大多数场景。
- `page_size` - 每页包含对象的数量。
- `page_query_param` - 一个字符串，指示查询参数的名字。也就是 `https://api.example.org/accounts/?page=5` 中的那个page字符串。
- `page_size_query_param` - 如果设置了，这将是一个字符串值，在url中设置每页对象数量的参数的名字，也就是 `https://api.example.org/accounts/?page=5&page_size=100` 中的 `page_size`。默认为None，表示客户端可能无法控制请求

页面的对象数量。注意，`page_size_query_param`指示的是那个参数的名字，而不是参数的值。

- `max_page_size` - 如果设置了，限制每页最大允许显示的对象数量。只有当`page_size_query_param`设置了的时候才有效。
- `last_page_strings` - 一个字符串的列表或元组。用于表示请求最后一页，默认是`('last',)`，比如`https://api.example.org/accounts/?page=last`
- `template` - 当使用可浏览API功能时，用于渲染分页控制的模板。设置为None，将关闭这一功能。默认是`"rest_framework/pagination/numbers.html"`。

LimitOffsetPagination

这种分页样式类似查找多个数据库记录时使用的语法。客户端使用一个“limit”和一个“offset”查询参数。limit参数指示要返回的最大对象数，相当于其他样式中的`page_size`属性。offset参数指示相对于完整的未分页集合的起始偏移位置。也就是相对于完整的查询集，从第一个对象开始offset往后数指定数目的位置开始，取出limit个对象。

请求:

```
GET https://api.example.org/accounts/?limit=100&offset=400
```

响应:

```
HTTP 200 OK
{
    "count": 1023
    "next": "https://api.example.org/accounts/?limit=100&offset=500",
    "previous": "https://api.example.org/accounts/?limit=100&offset=300",
    "results": [
       …
    ]
}
```

配置

要使用`LimitOffsetPagination`风格的分页器，需要如下配置：

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.LimitOffsetPagination'
}
```

你还可以添加一个 `PAGE_SIZE` 配置，相当于设置默认的 `limit` 值，客户端可以不再指定limit。

对于继承了 `GenericAPIView` 的类视图，你也可以通过设置 `pagination_class` 属性选择 `LimitOffsetPagination` 作为视图级别的分页功能。

属性

`LimitOffsetPagination` 类包括下面的属性。要使用这些属性，你首先要继承 `LimitOffsetPagination` 类，并按上面的方式激活子类。

- `default_limit` - 一个数值，表示默认的limit数量。
- `limit_query_param` - url中指示limit的参数名，一个字符串，默认是 `'limit'` 。
- `offset_query_param` - 类似上面，默认为 `'offset'` 。
- `max_limit` - 如果设置了，表示最大允许的limit数量。默认为None。
- `template` - 同前。默认为 `"rest_framework/pagination/numbers.html"` 。

CursorPagination

基于光标的分页显示了一个不透明的“光标”指示器，客户端可以使用它在结果集中分页。这种分页样式只显示正向和反向控件，不允许客户端导航到任意位置。

基于光标的分页要求结果集中有一个唯一的、不变的排序方式。这种排序依据通常是记录的创建时间戳，因为这提供了一种一致的排序方式，以便进行分页。

基于光标的分页比其他方案更复杂，但是，它也有以下好处：

- 提供一致的分页视图。如果使用得当， `CursorPagination` 可以确保客户端在翻页时不会看到同一对象两次，即使在分页过程中其他客户端正在插入新对象。
- 支持超大数据集。对于非常大的数据集，使用基于偏移量的分页样式进行分页可能会变得效率低下或不可用。基于光标的分页方案具有固定的时间属性，并且不会随着数据集的增大而减慢。

一些细节和说明

恰当地使用基于光标的分页需要注意一些细节。你需要考虑使用什么排序方式。默认是按 `"-created"` 排序，也就是创建时间的逆序。这需要一个前提，就是在模型上必须有一个 `created` 时间戳字段，并且将呈现一个“时间线”样式的分页视图，首先显示最近添加的项。

您可以通过覆盖 `pagination` 类上的 `ordering` 属性，或使用 `OrderingFilter` 过滤器类和 `CursorPagination` 来修改排序。

要正确使用光标分页应该有一个符合以下条件的字段：

- 应该是一个不变的值，例如时间戳或其他在创建时只设置一次的字段。
- 应该是唯一的，或者几乎是唯一的。毫秒精度的时间戳就是一个很好的例子。光标分页的这种实现使用了一种智能的“位置加偏移”样式，允许它正确地支持排序时不严格唯一的值。
- 应该是可以强制为字符串的不可为空的值。
- 不应该是浮点数。精度误差容易导致不正确的结果。
- 该字段应具有数据库索引。

使用不满足这些约束的字段进行排序也许仍然有效，但是将失去光标分页的一些优点。

配置

要使用 `CursorPagination` ，需要进行如下的全局配置：

```
1  REST_FRAMEWORK = {
2      'DEFAULT_PAGINATION_CLASS':
   'rest_framework.pagination.CursorPagination',
3      'PAGE_SIZE': 100
4  }
```

对于继承了 `GenericAPIView` 的类视图，你也可以通过设置 `pagination_class` 属性选择 `CursorPagination` 作为视图级别的分页功能。

属性

`CursorPagination` 类包括下面的属性。要使用这些属性，你首先要继承 `CursorPagination` 类，并按上面的方式激活子类。

- `page_size` :指示每页显示的数量。
- `cursor_query_param` ： url中表示‘cursor’分页的参数名。 默认为 `'cursor'` 。
- `ordering` ： 一个字符串，或者字符串的列表。表示用于排序的字段。默认是 `-created` 。
- `template` ： 同前，默认为 `"rest_framework/pagination/previous_and_next.html"` .

三、自定义分页风格

自定义分页类的步骤：

1. 继承 `pagination.BasePagination`

2. 重写 `paginate_queryset(self, queryset, request, view=None)` 和 `get_paginated_response(self, data)` 方法。

- `paginate_queryset` 方法接收初始的查询集，并返回一个可迭代的对象，这个对象只包含当前请求页面的数据。
- `get_paginated_response` 方法接收分页的数据，返回渲染过的实例。用这个方法替代DRF默认的Response方法，可以在返回的响应内容中，添加一些额外分页相关的数据和链接。

看一个例子，假设我们想用一个修改过的格式替换默认的分页输出样式，该格式在嵌套的“links”键中包含下一个和上一个链接。我们可以这样自定义分页类：

```
class CustomPagination(pagination.PageNumberPagination):
    def get_paginated_response(self, data):
        return Response({
            'links': {
                'next': self.get_next_link(),
                'previous': self.get_previous_link()
            },
            'count': self.page.paginator.count,
            'results': data
        })
```

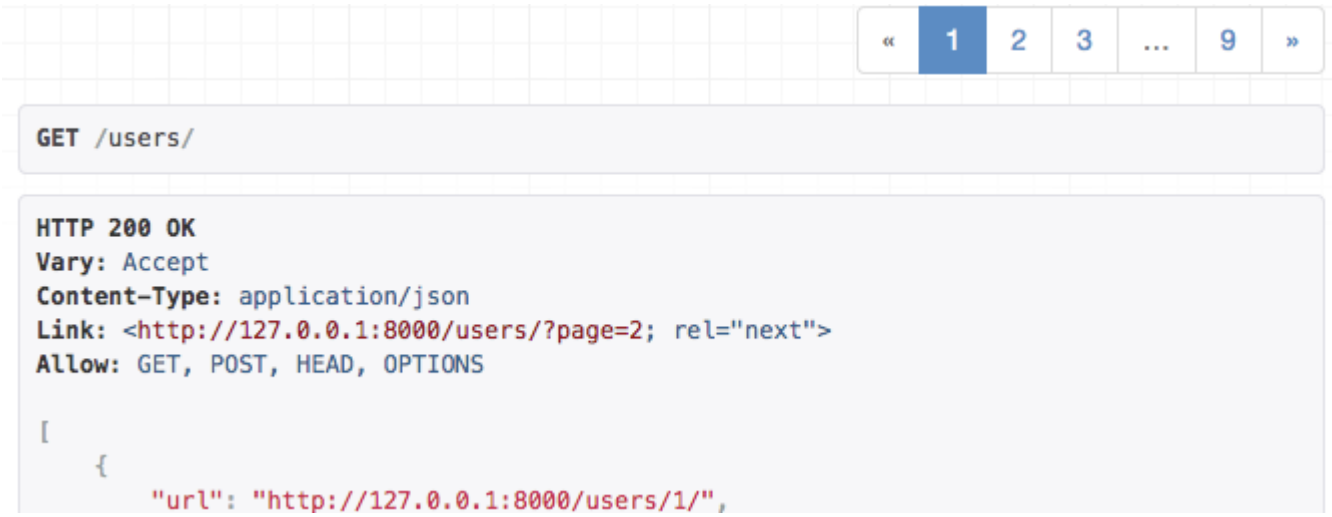
不要忘了配置我们自定义的分页类，否则它是不起作用的：

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS':
'my_project.apps.core.pagination.CustomPagination',
    'PAGE_SIZE': 100
}
```

你也可以在DRF框架提供的自动生成模式中使用分页控制，这需要你实现 `get_schema_fields()` 方法。该方法的签名如下：

```
get_schema_fields(self, view)
```

这个方法必须返回 `coreapi.Field` 实例的列表。



四、第三方模块

- DRF-extensions
- drf-proxy-pagination
- link-header-pagination