一、概要Schemas

API概要是一个有用的工具，通过它，你可以对服务器提供的所有API有一个整体上的浏览或查询，包括生成参考文档，或者驱动与API交互的动态客户端。

安装 Core API 和PyYAML

我们需要安装 `coreapi` 工具，为DRF提供概要支持。也有可能需要安装 `pyyaml` 模块，这样你就可以将概要渲染成常用的基于YAML的OpenAPI格式。通过pip一起安装它们吧：

```
pip install coreapi pyyaml
```

生成概要

有两种方式为你的API提供概要描述。

- 使用 `generateschema` 命令生成概要

具体命令如下：

```
$ python manage.py generateschema > schema.yml
```

- 使用 `get_schema_view` 添加一个视图

`get_schema_view` 方法可以为你添加一个动态生成的概要视图。

```
from rest_framework.schemas import get_schema_view

schema_view = get_schema_view(title="Example API")  # 看这里,生成了一个视图

urlpatterns = [
    path('schema/', schema_view),  #为视图添加url
    ...
]
```

概要的内部形式

当使用coreapi时，概要的形式为Document，它是有关API信息的顶级容器对象。可用的API交互使用“link”对象表示。每个链接都包含一个URL、HTTP方法，并且可能包含一个“field”实例列表，这些实例描述了API端点可以接受的参数。“link”和“field”实例也可能包含说明，这些说明允许将API架构渲染到用户文档中。

下面是一个API描述的例子，包含一个 `search` 端点：

```
coreapi.Document(
    title='Flight Search API',
    url='https://api.example.org/',
    content={
        'search': coreapi.Link(
            url='/search/',
            action='get',
            fields=[
                coreapi.Field(
                    name='from',
                    required=True,
                    location='query',
                    description='City name or airport code.'
                ),
                coreapi.Field(
                    name='to',
                    required=True,
                    location='query',
                    description='City name or airport code.'
                ),
                coreapi.Field(
                    name='date',
                    required=True,
                    location='query',
                    description='Flight date in "YYYY-MM-DD" format.'
                )
            ],
            description='Return flight availability and prices.'
        )
    }
)
```

概要的输出格式

为了在HTTP响应中呈现概要，内部表示必须渲染为响应中使用的实际内容。

REST框架提供几个不同的渲染器，可以使用它们来编码API概要：

- `renders.openapienderer` -渲染为基于YAML的OpenAPI，这是最广泛使用的API概要格式。
- `renders.jsonopenapinederer` -渲染到基于json的OpenAPI。
- `renders.corejsonrenderer` -渲染成Core JSON格式。这是一种用于coreapi客户端的格式。

二、创建概要

手动生成

使用 Core API 的 `Document` 方法手动生成概要，如下：

```
schema = coreapi.Document(
    title='Flight Search API',
    content={
        ...
    }
)  # 这里省略了coreapi的import
```

自动生成概要

使用 `SchemaGenerator` 类来自动生成概要.

最基本的用法就是提供一个title来初始化一个 `SchemaGenerator` 的实例，然后调用实例的 `get_schema()` 方法:

```
generator = schemas.SchemaGenerator(title='Flight Search API')
schema = generator.get_schema()
```

三、添加概要视图

有以下几种方法将概要视图添加到你的API中：

get_schema_view快捷方式

最简单的方法就是使用 `get_schema_view()` 函数，前面已经介绍过：

```
from rest_framework.schemas import get_schema_view

schema_view = get_schema_view(title="Server Monitoring API")

urlpatterns = [
    path('', schema_view),
    ...
]
```

然后你就可以在客户端中请求概要url，查看概要的详细内容了：

```
 $ http http://127.0.0.1:8000/ Accept:application/coreapi+json
 HTTP/1.0 200 OK
 Allow: GET, HEAD, OPTIONS
 Content-Type: application/vnd.coreapi+json

 {
     "_meta": {
         "title": "Server Monitoring API"
     },
     "_type": "document",
     ...
 }
```

`get_schema_view()` 的参数说明：

- `title`

概要的标题

- `url`

可用于传递一个架构规范的url。

```
schema_view = get_schema_view(
    title='Server Monitoring API',
    url='https://www.example.org/api/'
)
```

- `urlconf`

一个字符串，表示要为其生成API概要的URL conf的导入路径。默认为Django的 `ROOT_URLCONF` 设置的值。

```
schema_view = get_schema_view(
    title='Server Monitoring API',
    url='https://www.example.org/api/',
    urlconf='myproject.urls'
)
```

- `renderer_classes`

用于渲染的渲染器类的列表：

```
from rest_framework.schemas import get_schema_view
from rest_framework.renderers import JSONOpenAPIRenderer

schema_view = get_schema_view(
    title='Server Monitoring API',
    url='https://www.example.org/api/',
    renderer_classes=[JSONOpenAPIRenderer]  # 看这里
)
```

- `patterns`

url路径的列表，限制概要的范围。如果你只想暴露 `myproject.api` 的url，那么可以这么做：

```
schema_url_patterns = [
    url(r'^api/', include('myproject.api.urls')),
]

schema_view = get_schema_view(
    title='Server Monitoring API',
    url='https://www.example.org/api/',
    patterns=schema_url_patterns, # 看这里
)
```

- `generator_class`

用于指定 `SchemaGenerator` 的子类，并传递给 `SchemaView` 。

- `authentication_classes`

指定认证类的列表，用于概要端点的身份验证。默认使用 `settings.DEFAULT_AUTHENTICATION_CLASSES` 的值。

- `permission_classes`

权限验证的类列表，用于控制概要的访问权限。默认使用 `settings.DEFAULT_PERMISSION_CLASSES` 的值。

显示指定概要视图

如果你需要比 `get_schema_view()` 这个快捷函数多一点控制能力，那么你可以直接使用 `SchemaGenerator` 类，自动生成 `Document` 实例，并从视图中返回它。

这让你可以更灵活的定制纲要，比如实施不同的权限、限流或者认证机制。

下面是一个例子：

views.py:

```
from rest_framework.decorators import api_view, renderer_classes
from rest_framework import renderers, response, schemas

generator = schemas.SchemaGenerator(title='Bookings API') # 看这里

@api_view()
@renderer_classes([renderers.OpenAPIRenderer])  # 注意这里
def schema_view(request):  # 专门编写的概要视图
    schema = generator.get_schema(request)  # 生成概要
    return response.Response(schema) # 返回概要
```

urls.py:

```
urlpatterns = [
    path('/', schema_view),
    ...
]
```

还可以为不同的用户提供不同的概要，这取决于他们拥有的权限。此方法可用于确保未经身份验证的请求与经过身份验证的请求具有不同的概要，或者确保不同用户根据其角色可以看到API的不同部分。

为了根据用户权限的不同过滤概要的内容，你需要在 `get_schema()` 方法中传递一个request参数。

```
@api_view()
@renderer_classes([renderers.OpenAPIRenderer])
def schema_view(request):
    generator = schemas.SchemaGenerator(title='Bookings API')
    return response.Response(generator.get_schema(request=request))
```

显示地定义概要的详细信息

自动生成概要的另外一种方法是通过声明一个 `Document` 对象来显式地指定API纲要。这样做需要做更多的工作，但可以确保你完全控制概要的表示形式。

```
import coreapi
from rest_framework.decorators import api_view, renderer_classes
from rest_framework import renderers, response

schema = coreapi.Document(
    title='Bookings API',
    content={
        ...
    }
)  # 看这里

@api_view()
@renderer_classes([renderers.OpenAPIRenderer])
def schema_view(request):
    return response.Response(schema)
```

四、将概要作为文档

API概要的一个常见用途是使用它们来构建文档页面。

REST框架生成概要过程中使用docstrings自动填充文档中的描述信息。

这些描述文字基于：

- 方法的docstring（如果存在）。
- -类docstring中的命名部分，可以是单行或多行。
- -类的docstring。

看下面的例子：

一个带有docstring的APIView视图：

```
class ListUsernames(APIView):
    def get(self, request):
        """
        Return a list of all user names in the system.
        """
        usernames = [user.username for user in User.objects.all()]
        return Response(usernames)
```

一个带有docstring的ViewSet视图：

```
class ListUsernames(ViewSet):
    def list(self, request):
        """
        Return a list of all user names in the system.
        """
        usernames = [user.username for user in User.objects.all()]
        return Response(usernames)
```

一个带有docstring的通用视图，单行风格：

```
class UserList(generics.ListCreateAPIView):
    """
    get: List all the users.
    post: Create a new user.
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = (IsAdminUser,)
```

一个带有docstring的通用ViewSet视图，多行风格：

```
class UserViewSet(viewsets.ModelViewSet):
    """
    API endpoint that allows users to be viewed or edited.

    retrieve:
    Return a user instance.

    list:
    Return all users, ordered by most recently joined.
    """
    queryset = User.objects.all().order_by('-date_joined')
    serializer_class = UserSerializer
```

五、API参考

1. SchemaGenerator

一个类，它遍历路由的URL模式列表，为每个视图生成概要，并生成coreapi文档。

简单实例化方法如下：

```
generator = SchemaGenerator(title='Stock Prices API')
```

参数:

- `title` 必填！ API的名字。
- `url` - API纲要的根路由。可选。
- `patterns` -一个路由的列表。默认是项目的 URL conf。
- `urlconf` - 一个URL conf 模块，用于生成概要，默认是 `settings.ROOT_URLCONF` 。

get_schema(self, request)

返回一个 `coreapi.Document` 实例，表示API概要。

```
@api_view
@renderer_classes([renderers.OpenAPIRenderer])
def schema_view(request):
    generator = schemas.SchemaGenerator(title='Bookings API')
    return Response(generator.get_schema())
```

get_links(self, request)

返回一个嵌套字典，其中包含API概要中应包含的所有链接。

2. AutoSchema

在 `APIView` 视图中通过 `schema` 属性添加 `AutoSchema` 类。

`AutoSchema` 最主要的是关键字参数 `manual_fields` 。

manual_fields: 一个 `coreapi.Field` 实例的列表，用于生成字段。

```
class CustomView(APIView):
    schema = AutoSchema(manual_fields=[
        coreapi.Field(
            "my_extra_field",
            required=True,
            location="path",
            schema=coreschema.String()
        ),
    ])
```

也可以继承 `AutoSchema` 类，实现更多的自定制：

```
class CustomViewSchema(AutoSchema):
    """
    Overrides `get_link()` to provide Custom Behavior X
    """

    def get_link(self, path, method, base_url):
        link = super().get_link(path, method, base_url)
        # Do something to customize link here...
        return link

class MyView(APIView):
  schema = CustomViewSchema()
```

`AutoSchema` 类有下面的方法，可以使用或者重写：

- get_link(self, path, method, base_url)
- get_description(self, path, method)
- get_encoding(self, path, method)
- get_path_fields(self, path, method)

- get_serializer_fields(self, path, method)
- get_pagination_fields(self, path, method)
- get_filter_fields(self, path, method)
- get_manual_fields(self, path, method)
- update_fields(fields, update_with)

3. ManualSchema

DRF允许使用ManualSchema，手动为概要提供 `coreapi.Field` 实例的列表，以添加一些额外的信息：

```
class MyView(APIView):
  schema = ManualSchema(fields=[
        coreapi.Field(
            "first_field",
            required=True,
            location="path",
            schema=coreschema.String()
        ),
        coreapi.Field(
            "second_field",
            required=True,
            location="path",
            schema=coreschema.String()
        ),
    ]
  )
```

`ManualSchema` 有三个参数：

fields: 一个 `coreapi.Field` 实例的列表，必填参数。

description: 一个字符串描述，可选。

encoding: 编码方式，默认为 `None` ，例如 `application/json` ，可选。

4. Core API

下面是 `coreapi` 模块的一些简单用法。它们要使用 `from coreapi import ....` 的方式导入，而不是从 `rest_framework` 导入。

Document

代表一个API概要的容器。简单地理解为概要就行了。

- `title`： API的名字
- `url`： API使用的url
- `content`

字典类型，包含概要中的 `Link` 对象。有可能嵌套。例如：

```
content={
    "bookings": {
        "list": Link(...),
        "create": Link(...),
        ...
    },
    "venues": {
        "list": Link(...),
        ...
    },
    ...
}
```

Link

代表每个独立的API端点。

- `url`：端点的url。可能是一个URL模式，比如 `/users/{username}/` 。
- `action`：使用的HTTP方法。
- `fields`：一个字段实例的列表，描述输入的参数。
- `description`：API的简单描述

Field

表示一个输入参数

- `name`：输入的名字
- `required`：布尔值，表示是否必填
- `location`：决定信息如何编码。其取值只能是"path"、"query"、"form"、"body"之一
- `encoding`：编码方式，主要有**"application/json"**、**"multipart/form-data"**、**"application/x-www-form-urlencoded"**、**"application/octet-stream"**。
- `description`：简短的描述