一、概述

本教程将介绍如何创建一个简单的对代码片段进行高亮展示的Web API。这个过程中, 将会介绍组成DRF框架的各个组件，并让你大概了解各个组件是如何一起工作的。

这个教程是相当深入的，可能需要结合后面的API，反复揣摩。

注意：本教程的代码可以在Github的 tomchristie/rest-framework-tutorial 库中找到。

二、创建虚拟环境

我们先用virtualenv创建一个新的虚拟环境。这样就能确保与我们正在开展的任何其他项目保持良好的隔离。

以下为linux环境。windows或Pycharm环境，参考 www.liujiangblog.com 网站相关教程。

```
virtualenv env
source env/bin/activate
```

进入virtualenv环境后，安装我们需要的包。

```
pip install django  #当前为2.2版本
pip install djangorestframework   #当然为3.9版本
pip install pygments  # 代码高亮插件
```

DRF在导入时的名字为 `rest_framework` ，不要搞错了。

注意： 要随时退出virtualenv环境，只需输入 `deactivate` 。

三、 创建项目

好了，我们现在要开始写代码了。 首先，创建一个新的项目。

```
cd ~
django-admin.py startproject tutorial
cd tutorial
```

再创建一个app，名字叫做snippets，这个单词的意思是‘片段’，理解为代码片段。

```
python manage.py startapp snippets
```

我们需要将新建的 `snippets` app和 `rest_framework` 本身添加到 `INSTALLED_APPS` 。编辑 `tutorial/settings.py` 文件：

```
INSTALLED_APPS = (
    ...
    'rest_framework',
    'snippets.apps.SnippetsConfig',
)
```

好了，我们的准备工作做完了。

四、编写model模型

为了实现本教程的目的，我们将开始创建一个用于存储代码片段的简单的 `Snippet` model模型。打开 `snippets/models.py` 文件，并写入下面的代码：

```
from django.db import models
from pygments.lexers import get_all_lexers
from pygments.styles import get_all_styles

# 下面的几行代码是处理代码高亮的，不好理解，但没关系，它不重要。
LEXERS = [item for item in get_all_lexers() if item[1]]
LANGUAGE_CHOICES = sorted([(item[1][0], item[0]) for item in LEXERS])
STYLE_CHOICES = sorted((item, item) for item in get_all_styles())


class Snippet(models.Model):
    created = models.DateTimeField(auto_now_add=True)
    title = models.CharField(max_length=100, blank=True, default='')
    code = models.TextField()
    linenos = models.BooleanField(default=False)
    language = models.CharField(choices=LANGUAGE_CHOICES, default='python', max_length=100)
    style = models.CharField(choices=STYLE_CHOICES, default='friendly', max_length=100)

    class Meta:
        ordering = ('created',)
```

然后，使用下面的命令创建数据表：

```
python manage.py makemigrations snippets
python manage.py migrate
```

五、创建序列化类

开发Web API的第一件事是为我们的代码片段对象创建一种序列化和反序列方法，将其与诸如 `json` 格式进行互相转换。具体方法是声明与Django forms非常相似的序列化器（serializers）来实现。 在 `snippets` 的目录下创建一个名为 `serializers.py` 文件，并添加以下内容。

> 为每一个你需要序列化的model创建一个对应的序列化类。

```
from rest_framework import serializers
from snippets.models import Snippet, LANGUAGE_CHOICES, STYLE_CHOICES


class SnippetSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True) # 序列化时使用，反序列化时不用
    title = serializers.CharField(required=False, allow_blank=True, max_length=100)
    code = serializers.CharField(style={'base_template': 'textarea.html'})
    linenos = serializers.BooleanField(required=False)
    language = serializers.ChoiceField(choices=LANGUAGE_CHOICES, default='python')
    style = serializers.ChoiceField(choices=STYLE_CHOICES, default='friendly')
    # 注意，没有为model的created创建对应的序列化字段
    def create(self, validated_data):
        """
        使用验证后的数据，创建一个代码片段对象。使用的是Django的ORM的语法。
        """
        return Snippet.objects.create(**validated_data)

    def update(self, instance, validated_data):
        """
        使用验证过的数据，更新并返回一个已经存在的‘代码片段’对象。依然使用的是Django的ORM的语法。
        """
        instance.title = validated_data.get('title', instance.title)
```

```
24          instance.code = validated_data.get('code', instance.code)
25          instance.linenos = validated_data.get('linenos', instance.linenos)
26          instance.language = validated_data.get('language',
    instance.language)
27          instance.style = validated_data.get('style', instance.style)
28          instance.save()
29          return instance
```

Serializer是DRF提供的序列化基本类，供我们继承使用，它位于DRF的serializers包中。使用这个类，你需要自己编写所有的字段以及create和update方法，比较底层，抽象度较低，接近Django的form表单类的层次。

让我们看一下上面的代码。SnippetSerializer类的第一部分定义了序列化/反序列化过程中需要的字段。 `create()` 和 `update()` 方法定义了在调用 `serializer.save()` 时如何创建和修改实例。

序列化类与Django 的 `Form` 类非常相似，并在各种字段中包含类似的验证标志，例如 `required` ， `max_length` 和 `default` 。这里暂时不讲解各种字段的含义，以及它们包含的参数的用法，详见API。

字段参数可以控制serializer在某些情况下如何显示，比如渲染HTML的时候。上面的 `style={'base_template': 'textarea.html'}` 等同于在Django 的 `Form` 类中使用 `widget=widgets.Textarea` ，也就是使用文本输入框标签。这对于控制如何显示可浏览的API特别有用。

实际上也可以通过使用 `ModelSerializer` 类来节省一些编写代码的时间，就像教程后面会用到的那样。但是现在还继续使用我们刚才定义的serializer。

六、序列化器的基本使用

我们先来熟悉一下Serializer类的基本使用方法。输入下面的命令进入Django shell。

```
1   python manage.py shell
```

在命令行中，像下面一样导入几个模块，然后创建一些代码片段：

```
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
from rest_framework.renderers import JSONRenderer
from rest_framework.parsers import JSONParser

snippet = Snippet(code='foo = "bar"\n')
snippet.save()

snippet = Snippet(code='print("hello, world")\n')
snippet.save()
```

我们现在已经有2个代码片段实例了，让我们将第二个实例序列化：

```
serializer = SnippetSerializer(snippet)
serializer.data
# {'id': 2, 'title': '', 'code': 'print("hello, world")\n', 'linenos':
False, 'language': 'python', 'style': 'friendly'}
```

此时，我们将模型实例对象转换为了Python的原生数据类型。

```
>>> type(serializer)
<class 'snippets.serializers.SnippetSerializer'>
>>> type(serializer.data)
<class 'rest_framework.utils.serializer_helpers.ReturnDict'>
>>> issubclass(type(serializer.data), dict)
True
```

但是，要完成最终的序列化过程，我们还需要将数据转换成 `json` 格式，这样的话，客户端才可以理解。

```
content = JSONRenderer().render(serializer.data)
content
# b'{"id": 2, "title": "", "code": "print(\\"hello, world\\")\\n",
"linenos": false, "language": "python", "style": "friendly"}'

type(content)
#<class 'bytes'>
```

以上是序列化过程，也就是使用ORM从数据库中读取对象，然后序列化为DRF的某种格式，再转换为json格式（为什么上面是bytes类型，这是和Python语言相关的），最后将json数据通过HTTP发送给客户端。

反序列化则是上面过程的逆向。首先我们使用Python内置的io模块将我们前面生成的content转换为一个流（stream）对象，模拟从前端发送过来的json格式的请求数据，然后将数据解析为Python原生数据类型：

```
import io

stream = io.BytesIO(content)
data = JSONParser().parse(stream)
type(data)
# <class 'dict'>
```

然后我们要将数据类型转换成模型对象实例并保存。

```
serializer = SnippetSerializer(data=data)
serializer.is_valid()
# True
serializer.validated_data
# OrderedDict([('title', ''), ('code', 'print("hello, world")\n'),
('linenos', False), ('language', 'python'), ('style', 'friendly')])
serializer.save()
# <Snippet: Snippet object (3)>
```

上面的操作都是在shell中进行的，实际中我们不会这么麻烦。

可以看到序列化器的API和Django的表单(forms)是多么相似。

也可以序列化查询结果集（querysets）而不是单个模型实例，也就是同时序列化多个对象。只需要为serializer添加一个 `many=True` 标志。（这个功能是比较重要的）

```
serializer = SnippetSerializer(Snippet.objects.all(), many=True)
serializer.data
# [OrderedDict([('id', 1), ('title', ''), ('code', 'foo = "bar"\n'),
('linenos', False), ('language', 'python'), ('style', 'friendly')]),
OrderedDict([('id', 2), ('title', ''), ('code', 'print("hello, world")\n'),
('linenos', False), ('language', 'python'), ('style', 'friendly')]),
OrderedDict([('id', 3), ('title', ''), ('code', 'print("hello, world")'),
('linenos', False), ('language', 'python'), ('style', 'friendly')])]
```

七、使用ModelSerializers类

除了前面的Serializer类，DRF还给我们提供了几种别的可以继承的序列化类，ModelSerializers就是常用的一个。

前面我们写的的 `SnippetSerializer` 类中重复了很多包含在 `Snippet` 模型类（model）中的信息。如果能自动生成这些内容，像Django的 `ModelForm` 那样就更好了。事实上REST framework的 `ModelSerializer` 类就是这么一个类，它会根据指向的model，自动生成默认的字段和简单的create及update方法。

让我们来看看如何使用 `ModelSerializer` 类重构我们的序列化类。再次打开 `snippets/serializers.py` 文件，并将 `SnippetSerializer` 类替换为以下内容。

```
1   class SnippetSerializer(serializers.ModelSerializer):
2       class Meta:
3           model = Snippet
4           fields = ('id', 'title', 'code', 'linenos', 'language', 'style')
```

额外的提示一下，DRF的序列化类有一个repr属性可以通过打印序列化器类实例的结构(representation)查看它的所有字段。以下操作在命令行中进行：

```
1   from snippets.serializers import SnippetSerializer
2   serializer = SnippetSerializer()
3   print(repr(serializer))
4   # SnippetSerializer():
5   #     id = IntegerField(label='ID', read_only=True)
6   #     title = CharField(allow_blank=True, max_length=100, required=False)
7   #     code = CharField(style={'base_template': 'textarea.html'})
8   #     linenos = BooleanField(required=False)
9   #     language = ChoiceField(choices=[('Clipper', 'FoxPro'), ('Cucumber',
    'Gherkin'), ('RobotFramework', 'RobotFramework'), ('abap', 'ABAP'), ('ada',
    'Ada')...
10  #     style = ChoiceField(choices=[('autumn', 'autumn'), ('borland',
    'borland'), ('bw', 'bw'), ('colorful', 'colorful')...
```

注意： `ModelSerializer` 类并不会做任何特别神奇的事情，它们只是创建序列化器类的快捷方式：

- 一组自动确定的字段。
- 默认简单实现的 `create()` 和 `update()` 方法。

这个 `ModelSerializer` 类帮我们节省了很多代码，但同时，又降低了可定制性，如何取舍，取决于你的业务逻辑。

没有谁规定必须用 `ModelSerializer` 类，不能用前面的更基础的Serializer类，实际上在复杂的业务逻辑中，定制性更高的Serializer类，反而是更实用的。 `ModelSerializer` 类感觉比较鸡肋。

八、编写常规的Django视图

让我们看看如何使用我们新的Serializer类编写一些API视图。目前我们不会使用任何REST框架的其他功能，我们只需将视图作为常规Django视图编写。

编辑 `snippets/views.py` 文件，并且添加以下内容：

```
from django.http import HttpResponse, JsonResponse
from django.views.decorators.csrf import csrf_exempt
from rest_framework.renderers import JSONRenderer
from rest_framework.parsers import JSONParser
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
```

我们API的根视图的功能是列出所有的snippets或创建一个新的snippet。

```
@csrf_exempt    # 防止403
def snippet_list(request):
    """
    列出所有的代码片段或者创建新的。
    """
    if request.method == 'GET':
        snippets = Snippet.objects.all()
        serializer = SnippetSerializer(snippets, many=True) #注意many参数
        # 实用Django自带方法，响应json格式的数据
        return JsonResponse(serializer.data, safe=False)

    elif request.method == 'POST':
        data = JSONParser().parse(request)
        serializer = SnippetSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return JsonResponse(serializer.data, status=201)
        return JsonResponse(serializer.errors, status=400)
```

请注意，因为我们后面会使用没有CSRF令牌的客户端对此视图进行POST测试，因此我们需要为视图增加 `csrf_exempt` 装饰器，跳过csrf的检测，避免403。事实上DRF有专门应对的策略。

另外，我们还需要写一个与单个snippet对象相应的detail视图，用于获取，更新和删除这个snippet。

```
@csrf_exempt
def snippet_detail(request, pk):
```

```
    """
    获取、更新和删除指定的某个代码片段。
    """
    try:
        snippet = Snippet.objects.get(pk=pk)
    except Snippet.DoesNotExist:
        return HttpResponse(status=404)

    if request.method == 'GET':
        serializer = SnippetSerializer(snippet)
        return JsonResponse(serializer.data)

    elif request.method == 'PUT':
        data = JSONParser().parse(request)
        serializer = SnippetSerializer(snippet, data=data)
        if serializer.is_valid():
            serializer.save()
            return JsonResponse(serializer.data)
        return JsonResponse(serializer.errors, status=400)

    elif request.method == 'DELETE':
        snippet.delete()
        return HttpResponse(status=204)
```

注意视图函数的名称！注意两个视图函数各自支持的HTTP操作！注意区分POST和PUT方法！注意，同时只能创建或更新一个对象，暂时不支持批量更新或创建！但是读取可以批量!

视图有了，序列化器有了，模型有了，我们还差编写路由把请求和视图链接起来。创建一个 `snippets/urls.py` 文件：

```
from django.urls import path
from snippets import views

urlpatterns = [
    path('snippets/', views.snippet_list),
    path('snippets/<int:pk>/', views.snippet_detail),
]
```

上面是snippets这个app自己的二级路由文件，我们还需要在项目根URL配置 `tutorial/urls.py` 文件中，添加我们的snippet应用的include语句。

```
from django.urls import path, include

urlpatterns = [
    path('', include('snippets.urls')),
]
```

注：原来的admin路由，保留与否，随意。

这样，`127.0.0.1:8000/snippets/`将访问`snippet_list`视图。

值得注意的是，目前我们还没有正确处理好几种特殊情况。比如假设我们发送格式错误的`json`数据，或者使用视图不处理的HTTP方法发出请求，那么我们最终会出现一个500“服务器错误”响应。不过，暂时没有关系。

九、测试前面的工作

现在退出所有的shell...，启动Django开发服务器：

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 28, 2019 - 10:51:57
Django version 2.2, using settings 'tutorial.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

打开一个终端窗口，我们在命令行下测试服务器。

我们可以使用curl或httpie测试我们的服务器。Httpie是用Python编写的用户友好的http客户端，我们安装它。

可以使用pip来安装httpie：

```
pip install httpie
```

访问下面的url可以得到所有snippet的列表：

```
输入命令：http http://127.0.0.1:8000/snippets/

结果如下：
HTTP/1.1 200 OK
Content-Length: 354
```

```
Content-Type: application/json
Date: Sun, 28 Apr 2019 02:53:42 GMT
Server: WSGIServer/0.2 CPython/3.7.3
X-Frame-Options: SAMEORIGIN

[
    {
        "code": "foo = \"bar\"\n",
        "id": 1,
        "language": "python",
        "linenos": false,
        "style": "friendly",
        "title": ""
    },
    {
        "code": "print(\"hello, world\")\n",
        "id": 2,
        "language": "python",
        "linenos": false,
        "style": "friendly",
        "title": ""
    },
    {
        "code": "print(\"hello, world\")",
        "id": 3,
        "language": "python",
        "linenos": false,
        "style": "friendly",
        "title": ""
    }
]
```

或者我们可以指定id来获取特定snippet的detail信息：

```
http http://127.0.0.1:8000/snippets/2/

HTTP/1.1 200 OK
Content-Length: 120
Content-Type: application/json
Date: Sun, 28 Apr 2019 02:54:21 GMT
Server: WSGIServer/0.2 CPython/3.7.3
X-Frame-Options: SAMEORIGIN

{
```

```
        "code": "print(\"hello, world\")\n",
        "id": 2,
        "language": "python",
        "linenos": false,
        "style": "friendly",
        "title": ""
    }
```

当然，也可以在浏览器中访问这些URL来显示相同的json。