https://blog.csdn.net/qq\_43401941/article/details/108962767

Django REST Framework入门学习

初学爱好者 于 2020-10-08 14:27:27 发布 344 收藏 5

分类专栏： Python 文章标签： python django restful sqlite3

版权

Python

专栏收录该内容

13 篇文章 0 订阅

订阅专栏

Django REST Framework

导学

学习目标：使用DRF开发RESTful API接口

学习内容：序列化(serializers)、视图集(viewsets)、路由(routers)、认证(authentication)、认证(permission)

学习效果：DRF的多种视图实现课程信息的增删改查

image.png

一、创建项目

1.1 创建Django项目

image.png

1.2 修改 settings.py配置文件

ALLOWED\_HOSTS = ["\*"]

LANGUAGE\_CODE = 'zh-hans'

TIME\_ZONE = 'Asia/Shanghai'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = False

// 在settings.py文件的最下面添加

STATIC\_ROOT = os.path.join(BASE\_DIR, "static")

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, "staticfiles")

]

1

2

3

4

5

6

7

8

9

10

11

12

13

14

然后在 drf\_tutonrial项目下新建一个static文件夹

1.3 生成数据表

python manage.py makemigrations

python manage.py migrate

创建管理员用户，填写用户名、邮箱地址（非必填）和密码

python manage.py createsuperuser

1.4 运行并访问

运行命令：python manage.py runserver

image.png

image.png

二、安装 Django REST Framework

1）点击Settings进行安装

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-Xc5crBWn-1602137911055)(F:\Android\android案例开发笔记\直播案例开发图片\image-20201007142127980.png)]

2）在pycharm控制台中通过命令进行安装

pip install djangorestframework

2.1 在 settings.py中添加配置

INSTALLED\_APPS = [

......,

'rest\_framework', # RESTful API

'rest\_framework.authtoken', # DRF自带的 Token认证

]

1

2

3

4

5

在settings.py文件最下面添加DRF的全局配置

# DRF的全局配置

REST\_FRAMEWORK = {

# 设置分页

'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.PageNumberPagination',

'PAGE\_SIZE': 10,

# 设置日期格式化

'DATETIME\_FORMAT': '%Y-%m-%d %H:%M:%S',

# 当 DRF返回 response对象时，要使用的那个 rander类

'DEFAULT\_RENDER\_CLASSES': [

'rest\_framework.renders.JSONRenderer',

'rest\_framework.renders.BrowsableAPIRenderer',

],

# 解析器，如何解析 request请求中的 request.data

'DEFAULT\_PARSER\_CLASSES': [

'rest\_framework.parsers.JSONParser',

'rest\_framework.parsers.FormParser',

'rest\_framework.parsers.MultiPartParser',

],

# 权限以及认证相关的配置

'DEFAULT\_PERMISSION\_CLASSES': [

'rest\_framework.permissions.IsAuthenticated',

],

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.BasicAuthentication',

'rest\_framework.authentication.SessionAuthentication',

'rest\_framework.authentication.TokenAuthentication',

]

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

2.2 生成数据表

python manage.py makemigrations

python manage.py migrate

2.3 添加DRF登录以及退出的路由

在urls.py文件中

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

# DRF的登录退出

path('api-auth/', include('rest\_framework.urls')),

]

1

2

3

4

5

6

7

8

2.4 运行并访问

运行命令：python manage.py runserver

image.png

这里跳转到了这个404页面。因为我们还有开发这个accounts/profile/路由，实际上已经登录成功了

image-20201007143911287.png

三、编写课程信息模型

3.1 在course/models.py中编写课程信息模型

from django.db import models

from django.conf import settings

# Create your models here.

class Course(models.Model):

name = models.CharField(max\_length=255, unique=True, help\_text='课程名称', verbose\_name='名称')

introduction = models.CharField(max\_length=255, unique=True, help\_text='课程介绍', verbose\_name='介绍')

teacher = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE, help\_text='课程讲师', verbose\_name='讲师')

price = models.DecimalField(max\_digits=6, decimal\_places=2, help\_text='课程价格', verbose\_name='价格')

create\_at = models.DateTimeField(auto\_now\_add=True, verbose\_name='创建时间')

update\_at = models.DateTimeField(auto\_now=True, verbose\_name='更新时间')

class Meta:

verbose\_name = '课程信息'

verbose\_name\_plural = verbose\_name

# 使用价格排序

ordering = ('price',)

def \_\_str\_\_(self):

return self.name

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

3.2 将上面的字段注册到admin.py中

方便添加数据时可以从Admin后台直接加

from django.contrib import admin

from .models import Course

# Register your models here.

@admin.register(Course)

class CourseAdmin(admin.ModelAdmin):

list\_display = ('name', 'introduction', 'teacher', 'price')

# 可搜索字段

search\_fields = list\_display

# 过滤字段

list\_filter = list\_display

1

2

3

4

5

6

7

8

9

10

11

12

13

14

3.3 生成数据表

python manage.py makemigrations

python manage.py migrate

image.png

image.png

四、序列化（Serialization）

4.1 什么时序列化

我们可以将序列化理解为：将程序中的一个数据结构类型转换为其他格式（字典、JSON、XML等），例如将Django中的模型类对象转换为JSON字符串，这个转换过程我们称为序列化。

4.2 继承ModelSerializer序列化模型类

from django import forms

from rest\_framework import serializers

from .models import Course

from django.contrib.auth.models import User

# class CourseForm(forms.ModelForm):

# """ form表单类的写法，这里不用 """

# class Meta:

# model = Course

# fields = ('name', 'introduction', 'teacher', 'price')

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_'

class CourseSerializer(serializers.ModelSerializer):

""" 课程序列化模型类 """

# 外键字段 只读

teacher = serializers.ReadOnlyField(source='teacher.username')

class Meta:

model = Course

# fields = ('id', 'name', 'introduction', 'teacher', 'price', 'create\_at', 'update\_at')

# 如果将所有字段都包含进来可以写 \_\_all\_\_

fields = '\_\_all\_\_'

# 在后面视图集的时候使用

# class CourseSerializer(serializers.HyperlinkedModelSerializer):

# """ 课程序列化模型类 [带超链接的API] """

# # 外键字段 只读

# teacher = serializers.ReadOnlyField(source='teacher.username')

#

# class Meta:

# model = Course

# # url是默认值。可在settings.py中设置 URL\_FIELD\_NAME使全局生效

# fields = ('id', 'url', 'name', 'introduction', 'teacher', 'price', 'create\_at', 'update\_at')

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

五、Django的views开发RESTful API接口

5.1 DRF视图开发RESTful API接口

函数式编程 Function Based View

类视图 Classed Based View

通用类视图 Generic Classed Based View

DRF的视图集 viewsets

5.2 Django 原生开发API接口

from django.shortcuts import render

import json

from django.http import JsonResponse, HttpResponse

from django.views.decorators.csrf import csrf\_exempt

from django.utils.decorators import method\_decorator

from django.views import View

# Create your views here.

course\_dict = {

'name': '课程名称',

'introduction': '课程介绍',

'price': 0.11

}

# 原生 Django FBV 编写 API接口

@csrf\_exempt

def course\_list(request):

if request.method == 'GET':

# return HttpResponse(json.dumps(course\_dict), content\_type='application/json')

# 与上面的写法效果是一样的

return JsonResponse(course\_dict)

if request.method == 'POST':

course = json.loads(request.body.decode('utf-8'))

# return HttpResponse(json.dumps(course), content\_type='application/json')

return JsonResponse(course, safe=False)

# 原生 Django CBV 编写 API接口

@method\_decorator(csrf\_exempt, name='dispatch')

class CourseList(View):

def get(self, request):

return JsonResponse(course\_dict)

def post(self, request):

course = json.loads(request.body.decode('utf-8'))

return JsonResponse(course, safe=False)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

5.3 DRF中的装饰器api\_view

5.3.1 查看和添加课程信息

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from .models import Course

from .serializers import CourseSerializer

""" 一、函数式编程 Function Based View """

@api\_view(["GET", "POST"])

def course\_list(request):

"""

获取所有课程信息或新增一个课程

:param request:

:return:

"""

if request.method == 'GET':

s = CourseSerializer(instance=Course.objects.all(), many=True)

return Response(data=s.data, status=status.HTTP\_200\_OK)

elif request.method == 'POST':

# 部分更新用 partial=True属性

# s = CourseSerializer(data=request.data, partial=True)

s = CourseSerializer(data=request.data)

if s.is\_valid():

s.save(teacher=request.user)

return Response(data=s.data, status=status.HTTP\_201\_CREATED)

return Response(s.errors, status=status.HTTP\_400\_BAD\_REQUEST)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

在 course下新建 urls.py文件

from django.urls import path, include

from course import views

urlpatterns = [

# Function Based View

path('fbv/list/', views.course\_list, name='fbv-list')

]

1

2

3

4

5

6

7

然后在总的路由中添加课程的路由

urlpatterns = [

path('admin/', admin.site.urls),

# DRF的登录退出

path('api-auth/', include('rest\_framework.urls')),

path('course/', include('course.urls'))

]

1

2

3

4

5

6

7

运行并访问

这里我们可以先在Admin后台管理中添加一条课程信息

image.png

在访问course/fbv/list接口

image.png

添加一条课程信息—[POST]，再刷新

image.png

image.png

image.png

5.3.2 获取、更新、删除一条课程信息

views.py

@api\_view(["GET", "PUT", "DELETE"])

def course\_detail(request, pk):

"""

获取、更新、删除一条课程

:param request:

:param pk:

:return:

"""

try:

course = Course.objects.get(pk=pk)

except Course.DoesNotExist:

return Response(data={"message": "没有此课程信息"}, status=status.HTTP\_404\_NOT\_FOUND)

else:

if request.method == "GET":

s = CourseSerializer(instance=course)

return Response(data=s.data, status=status.HTTP\_200\_OK)

elif request.method == "PUT":

s = CourseSerializer(instance=course, data=request.data)

if s.is\_valid():

s.save()

return Response(s.data, status=status.HTTP\_200\_OK)

elif request.method == "DELETE":

course.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

运行并访问

获取一条课程信息：

image.png

image.png

5.4 使用Postman工具测试API接口

Postman工具下载地址：https://www.postman.com/downloads/

第一次请求需要进行认证，选择用户名和密码认证

image.png

1）查询所有课程信息

image-20201007185507187.png

2）添加一条课程信息

image.png

响应我们添加的数据

image.png

3）获取一条课程信息

image.png

4）更新一条课程信息，比如说更新第二条

更新前：

image-20201007191035610.png

更新后：

image.png

5）删除一条课程信息，比如说删除id为3的

image.png

六、DRF中的视图APIView

6.1 获取课程信息和添加课程信息

from rest\_framework.views import APIView

"""二、 类视图 Class Based View"""

class CourseList(APIView):

def get(self, request):

"""

查询所有课程信息

:param request:

:return:

"""

queryset = Course.objects.all()

s = CourseSerializer(instance=queryset, many=True) # instance是查询集

return Response(s.data, status=status.HTTP\_200\_OK)

def post(self, request):

"""

添加课程信息

:param request:

:return:

"""

s = CourseSerializer(data=request.data) # 这里的 data是前端传递过来的数据

if s.is\_valid():

s.save(teacher=self.request.user)

return Response(data=s.data, status=status.HTTP\_201\_CREATED)

return Response(s.errors, status=status.HTTP\_400\_BAD\_REQUEST)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

6.2 在urls.py文件中添加路由

urlpatterns = [

# Function Based View

path('fbv/list/', views.course\_list, name='fbv-list'),

path('fbv/detail/<int:pk>/', views.course\_detail, name='fbv-detail'),

# Class Based View

path('cbv/list/', views.CourseList.as\_view(), name='cbv-list')

]

1

2

3

4

5

6

7

8

6.3 使用Postman测试接口

1）查询课程信息

image.png

2）添加课程信息

image.png

6.4 编写课程的详情信息接口

class CourseDetail(APIView):

@staticmethod

def get\_object(pk):

"""

处理通用的返回对象

:param pk:

:return:

"""

try:

return Course.objects.get(pk=pk)

except Course.DoesNotExist:

return

def get(self, request, pk):

"""

获取一条课程信息

:param request:

:param pk:

:return:

"""

obj = self.get\_object(pk=pk)

# 如果没有获取到该课程信息

if not obj:

return Response(data={"message": "没有此课程信息"}, status=status.HTTP\_404\_NOT\_FOUND)

# 有课程信息，进行序列化

s = CourseSerializer(instance=obj)

return Response(s.data, status=status.HTTP\_200\_OK)

def put(self, request, pk):

"""

更新一条课程信息

:param request:

:param pl:

:return:

"""

obj = self.get\_object(pk=pk)

# 如果没有获取到该课程信息

if not obj:

return Response(data={"message": "没有此课程信息"}, status=status.HTTP\_404\_NOT\_FOUND)

# 反序列化

s = CourseSerializer(instance=obj, data=request.data)

if s.is\_valid():

s.save()

return Response(data=s.data, status=status.HTTP\_201\_CREATED)

return Response(s.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, pk):

"""

删除一条课程信息

:param request:

:param pk:

:return:

"""

obj = self.get\_object(pk=pk)

# 如果没有获取到该课程信息

if not obj:

return Response(data={"message": "没有此课程信息"}, status=status.HTTP\_404\_NOT\_FOUND)

obj.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

6.6 使用Postman测试接口

1）获取一条课程信息

GET http://127.0.0.1:8000/course/cbv/detail/4/

2）更新一条课程信息

PUT http://127.0.0.1:8000/course/cbv/detail/4/

3）删除一条课程信息

DELETE http://127.0.0.1:8000/course/cbv/detail/2/

7、DRF中的通用类视图GenericAPIView

7.1 增删改查一次搞定

from rest\_framework import generics

""" 三、通用类视图 Generic Class Based View """

class GCourserList(generics.ListCreateAPIView):

queryset = Course.objects.all()

serializer\_class = CourseSerializer

# 因为在保存这个实例的时候没有把当前请求的用户赋值给我们自定义的 teacher，

# 所以我们要重写这个方法

def perform\_create(self, serializer):

serializer.save(teacher=self.request.user)

class GCourseDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = Course.objects.all()

serializer\_class = CourseSerializer

# 这里因为不需要更新 teacher字段，所以不需要重新上面的方法

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

7.2 添加路由

# Generic Class Based View

path('gcbv/list/', views.GCourserList.as\_view(), name='gcbv-list'),

path('gcbv/detail/<int:pk>/', views.GCourseDetail.as\_view(), name='gcbv-list')

1

2

3

7.4 使用Postman工具测试API接口

1）获取所有课程信息

GET http://127.0.0.1:8000/course/gcbv/list/

2）添加一条课程信息

POST http://127.0.0.1:8000/course/gcbv/list/

3）获取一条课程信息

GET http://127.0.0.1:8000/course/gcbv/detail/6/

4）修改一条课程信息

PUT http://127.0.0.1:8000/course/gcbv/detail/6/

5）删除一条课程信息

DELETE http://127.0.0.1:8000/course/gcbv/detail/6/

八、DRF的viewsets视图集的增删改查接口

8.1 只需一个接口实现增删改查

from rest\_framework import viewsets

""" 四、DRF的视图集viewsets """

class CourseViewSet(viewsets.ModelViewSet):

queryset = Course.objects.all()

serializer\_class = CourseSerializer

def perform\_create(self, serializer):

serializer.save(teacher=self.request.user)

1

2

3

4

5

6

7

8

9

10

11

8.2 设置路由

1）写法一

# viewsets

path('viewset/', views.CourseViewSet.as\_view({

"get": "list", "post": "create"

}), name='viewsets-list'),

path('viewset/<int:pk>/', views.CourseViewSet.as\_view({

"get": "retrieve", "put": "update", "patch": "partial\_update", "delete": "destroy"

}), name='viewsets-detail')

1

2

3

4

5

6

7

不同的请求对应不同的方法

image.png

2）写法二

from rest\_framework.routers import DefaultRouter

# 实例化

router = DefaultRouter()

# 把实例集注册到这个路由里面

router.register(prefix='viewsets', viewset=views.CourseViewSet)

urlpatterns = [

......

path('', include(router.urls))

]

1

2

3

4

5

6

7

8

9

10

11

8.3 使用Postman工具测试API接口

1）获取全部课程信息

GET http://127.0.0.1:8000/course/viewsets

2）获取一条课程信息

GET http://127.0.0.1:8000/course/viewsets/7

3）添加一条课程信息

POST http://127.0.0.1:8000/course/viewsets/

4）修改一条课程信息

PUT http://127.0.0.1:8000/course/viewsets/8/

5）删除一条课程信息

DELETE http://127.0.0.1:8000/course/viewsets/8/

————————————————

版权声明：本文为CSDN博主「初学爱好者」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_43401941/article/details/108962767