1

2022年8月19日

11:35

1

1.        drf框架简介

DRF(django rest framework)是基于django的一个框架，由Tom Christie大神二次开发的开源项目，它可以帮助我们快速开发restful规范的接口框架

什么是restful。

REST与技术⽆关，代表的是⼀种软件架构⻛格（REST是Representational state transfer的简称，中⽂翻译为 表征状态转移）

特点：

•        提供了定义序列化器Serializer的方法，可以快速根据 Django ORM 或者其它库自动序列化/反序列化；

•        提供了丰富的类视图、Mixin扩展类，简化视图的编写；

•        多种身份认证和权限认证方式的支持；

•        直观的 API web 界面；

2.        restful简介

restful 在前后端分离的应用模式中，我们通常将后端开发的每个视图都称为一个接口，或者API，前端通过访问接口来对数据进行增删改查。

restful是一种后端API接口规范

•        http动词

对于资源的具体操作类型，由HTTP动词表示。

常用的HTTP动词有下面四个（括号里是对应的SQL命令）：

•        GET（SELECT）：从服务器取出资源（一项或多项）。

•        POST（CREATE）：在服务器新建一个资源。

•        PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）。

•        DELETE（DELETE）：从服务器删除资源。

三个不常用的HTTP动词：

•        PATCH（UPDATE）：在服务器更新(更新)资源（客户端提供改变的属性）。

•        HEAD：获取资源的元数据。

•        OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的

•        状态码

•        网络状态码

•        200 ok : 服务器成功返回用户请求的数据

•        201 created : 用户新建或修改数据成功

•        202 accepted : 表示一个请求已经进入后台排队(异步任务)

•        204 no content : 用户删除数据成功

•        400 invalid request : 用户发出的请求有误,服务器没有新建或修改数据的操作

•        401 unauthorized : 表示用户咩有权限(令牌,用户名,密码错误)

•        403 forbidden : 表示用户得到授权,但是访问是被禁止的

•        404 not found : 用户发出的请求针对的是不存在的记录,服务器没有进行操作

•        406 not acceptable : 用户请求的格式不对(比如用户请求json格式,但是只有xml格式)

•        410 gone : 用户请求的资源被永久删除

•        422 unprocesable entity : 当创建一个对象时,发生一个验证错误

•        500 internal server error : 服务器发生错误,用户无法判断发出的请求是否成功

•        数据状态码

•        0：成功（+msg：具体信息）

•        -1：失败（+msg：具体信息）

•        错误处理

return Response({'code': -1, 'res': "error", 'msg': repr(e)}, status=status.HTTP\_400\_BAD\_REQUEST)

•        正确处理：

return Response({'code': 0, 'res': "success", 'msg': "删除成功"}, status=status.HTTP\_200\_OK)

•        返回结果

•        GET /collection：返回资源对象的列表（数组）

•        GET /collection/resource：返回单个资源对象

•        POST /collection：返回新生成的资源对象

•        PUT /collection/resource：返回完整的资源对象

•        PATCH /collection/resource：返回完整的资源对象

•        DELETE /collection/resource：返回一个空文档

•        流程

•        安装与注册

•        drf是django框架的一个插件，所以要先安装django；

•        安装命令：pip3 install djangorestframework

•        使用drf时，要在settings中注册。因为drf相当于一个别人写好的app，所以我们拿来用时要先在settings中注册。

•        序列化

序列化和反序列化

api接口开发，最核心最常见的一个过程就是序列化，所谓序列化就是把数据转换格式，序列化可以分两个阶段：

序列化： 把我们识别的数据转换成指定的格式提供给别人，可选择序列化的模型表字段。（后端处理数据返回给前端）

ex：我们在django中获取到的数据默认是模型对象，但是模型对象数据无法直接提供给前端或别的平台使用，所以我们需要把数据进行序列化，变成字符串或者json数据，提供给别人。

反序列化：把别人提供的数据转换/还原成我们需要的格式，可选择对数据验证的规则。(处理前端传过来的数据)

ex：前端js提供过来的json数据，对于python而言就是字符串，我们需要进行反序列化换成模型类对象，这样我们才能把数据保存到数据库中。

3.        视图

lookup\_field = "id" or filter\_fields = ('dept')

4.        路由

drf中重写了 as\_view 和dispatch方法，其实就是在原来django的功能基础上添加了一些功能

•        as\_view，免除了csrf 验证，一般前后端分离不会使用csrf token认证（后期会使用jwt认证）。

•        dispatch，内部添加了 版本处理、认证、权限、访问频率限制等诸多功能。

•        视图

DRF

2022年8月19日

11:35

* 1. drf框架简介

DRF(django rest framework)是基于django的一个框架，由Tom Christie大神二次开发的开源项目，它可以帮助我们快速开发restful规范的接口框架

* 1. 特点：
     + 提供了定义序列化器Serializer的方法，可以快速根据 Django ORM 或者其它库自动序列化/反序列化；
     + 提供了丰富的类视图、Mixin扩展类，简化视图的编写；
     + 多种身份认证和权限认证方式的支持；
     + 直观的 API web 界面；

* 1. restful简介

REST与技术⽆关，代表的是⼀种软件架构⻛格（REST是Representational state transfer的简称，中⽂翻译为 表征状态转移）。

restful 在前后端分离的应用模式中，我们通常将后端开发的每个视图都称为一个接口，或者API，前端通过访问接口来对数据进行增删改查。

restful是一种后端API接口规范

* + RESTful API规范（ <https://blog.csdn.net/songhaixing2/article/details/115937002>）

* + 数据的安全保障, 通常使用 https 进行传输
  + 接口特征表现 : 域名中有 api 标识
  + 多版本 : 请求地址中带版本, 或者在请求头中

機器產生的替代文字:
1 
2 
3 
https://127.e.@.e:8@8@/api/vl/ 
https://127.e.@.e:8@8@/api/v2/ 

* + 数据即资源, 均使用名词表示 (可以复数尽量不要用动词)
  + 通过请求方式的不同来执行不同的操作

機器產生的替代文字:
1 
get 
2 
post . 
3 
put 
patch : 
4 
delete : 
5 
6 
7 
https. 
• // api . baidu.com/books 
8 
https. 
• // api . baidu.com/books/l 
9 
https. 
• // api . baidu.com/books 
https. 
• // api . baidu.com/books/l 
11 
https. 
• // api . baidu.com/books/l 
12 
https. 
• // api . baidu.com/books/l 
13 
getiå*_ : 
ge : 
postiå*_ : 
putiå*_ : 
patchiå*_ : 
de leteiå*_ : 

* + 请求路径中带过滤, 在 url 后面带参数
  + 返回数据中携带响应状态码
  + 返回数据中携带错误信息
  + 针对不同操作, 返回数据符合如下规范(这只是规范

機器產生的替代文字:
1 
2 
3 
4 
5 
6 
GET 'books: 
GET 'books/ 1: 
POST 'books: 
PUT 'books/ 1: 
PATCH 'books/l: 
DELETE 'books/ 1: 

* + 返回结果中携带资源连接

機器產生的替代文字:
'name 
ur11' : " https : / cnblogs.com/songhaixing/" 

* + 状态码
    - 网络状态码
      * 200  ok : 服务器成功返回用户请求的数据
      * 201  created  : 用户新建或修改数据成功
      * 202  accepted  : 表示一个请求已经进入后台排队(异步任务)
      * 204  no content : 用户删除数据成功
      * 400  invalid request : 用户发出的请求有误,服务器没有新建或修改数据的操作
      * 401  unauthorized : 表示用户咩有权限(令牌,用户名,密码错误)
      * 403  forbidden : 表示用户得到授权,但是访问是被禁止的
      * 404  not found : 用户发出的请求针对的是不存在的记录,服务器没有进行操作
      * 406  not acceptable : 用户请求的格式不对(比如用户请求json格式,但是只有xml格式)
      * 410  gone : 用户请求的资源被永久删除
      * 422  unprocesable entity : 当创建一个对象时,发生一个验证错误
      * 500  internal server error : 服务器发生错误,用户无法判断发出的请求是否成功

* + 数据状态码
    - 0：成功（+msg：具体信息）
    - -1：失败（+msg：具体信息）

* + 错误处理

return Response({'code': -1, 'res': "error", 'msg': repr(e)}, status=status.HTTP\_400\_BAD\_REQUEST)

* + 正确处理：

return Response({'code': 0, 'res': "success", 'msg': "删除成功"}, status=status.HTTP\_200\_OK)

* + 流程
    - 安装与注册
      * drf是django框架的一个插件，所以要先安装django；
      * 安装命令：pip3 install djangorestframework
      * 使用drf时，要在settings中注册。因为drf相当于一个别人写好的app，所以我们拿来用时要先在settings中注册。

機器產生的替代文字:
INSTALLED APPS 
' django . contrib. admin' 
' django . contrib. auth ' 
' django . contrib. contenttypes , 
' django . contrib. sessions ' 
' django . contrib . messages 
' django . contrib. staticfiles ' 
' rest framework 
'django filters' , 
' django_apscheduler ' 
# 'rest_framework . authtoken ' 
' database', 
user' 
'form', 
' public

* + 路由

<https://zhuanlan.zhihu.com/p/425798131>

drf中重写了 as\_view 和dispatch方法，其实就是在原来django的功能基础上添加了一些功能

* + CBV传统路由

機器產生的替代文字:
from django.urls import path 
from . 
import views 
urlpatterns 
path(" apply/" , views . Apply.as_view(), ) , 
views. ModifyApp1y.as_view(), 
path("excel/", views . form _ excel"), 
path("signCenter/", views . SignCenter.as view(), 
path (" , 
views . Signlnfo.as view(), ) , 
path("equipmentTimeCenter/", views. EquipmentTimeCenter.as view(), name:" Equipment Time"), 
views. FullPage. as_view(), name:" Full Page"), 
path , 
path ( " scheduler/" , 
views . Scheduler. as view(),

* + 继承ViewSetMixin

機器產生的替代文字:
1 
2 
3 
4 
5 
6 
eg: 
getTxä*iP 
-r Vi ewSetMi x i n 
(list) 

機器產生的替代文字:
Fit,2 
url(r' Aapi /books/v2/' , views .BookMode1ViewSet1 get' : 
'list' D) 

機器產生的替代文字:


* + 自动生成路由

機器產生的替代文字:
# h it3 Model ViewSet 
from rest _ framework import routers 
1. SimpleRouter 
2. DefaultR0uter 
SimpleRouter -5 DefaultRouter 
register( 

機器產生的替代文字:
from rest_framework import routers 
router 
routers . SimpleRouter ( ) 
router . regi ster(' api /books/v3' , 
vi ews. BookMode IV i ewSet2) 
pri nt(router.urls) 
3RegexlJRLPattern book-list *api / books/v3/$', 
CRegexLlRLPattern book-detail Aapi /books/v3/ ( ? P (plo CA/ . ) + ) 
urlpatterns +- router . urls 

* + 视图

機器產生的替代文字:
class ModifyApp1y(RetrievelJpdateDestroyAPIView 
GET : 
PATCH 
queryset Application. objects. all 
serializer_class ApplySeria1izer 
filter backends 
DjangoFi1terBackend, 
lookup _ field 
" id" 
pagination_class 
StandardPageNumberPagination 

lookup\_field = "id" （获取单个对象）

filter\_fields = ('dept') （过滤）

* + 使用这些方法的重要前提必须要
    - queryset
    - serializer\_class
    - 如果我们要使用get\_object()必须前端传给我们一个pk

* + 属性

|  |  |
| --- | --- |
| **属性** | **作用** |
| **queryset** (常用) | 需要序列化的查询集 |
| **serializer\_class** (常用) | 指定使用的序列化器 |
| pagination\_class | 分页控制类 |
| filter\_backends | 过滤控制后端 |
| lookup\_field | 查询单一数据库对象时使用的条件字段，默认为\*\*’pk’\*\* |

* + 方法

機器產生的替代文字:
get_serializer_class(selt) 
"get serializer(self, args, 
*kwargs) 
get_queryset(self) 
get object(selt) 
E: request. tormat. View, 
request 
view 
format 

* + self.get\_queryset() :获取所有
  + self.get\_object() :获取根据pk获取单个

常用的几个参数

* + data:用于反序列化传参
  + instance:如果有后面save执行更新方法,没有执行创建方法
  + content：额外添加数据给序列化器
  + many:获取多个序列化对象
  + partial:为True可以进行局部字段修改,默认为False为所有字段修改

content ex：

传值：

機器產生的替代文字:
views . py 
serilizer TestSeriIizer 
i nstancezi nstance 
data-request . data 
context: 
'arg' 
value 

获取：

機器產生的替代文字:
def update(self, instance, validate data 
self. context. get 
('arg') 
arg 
pass 
return instance 

* + 过滤器

from rest\_framework.filters import SearchFilter

* + 排序过滤器

機器產生的替代文字:
frcn franewczk. 
fran . models, 
frcn franewczk. 
class 
class 
filter backend- 
f felds 
. crder_by ( 
) .21L() 

* + 群查搜索过滤器

機器產生的替代文字:
frcn franewczk. generics 
fran . models, 
frcn framework. SearchEf2ter 
class 
mcde2s.Ccuzse.2btects.f2LZer (2 s 
serialize: class 
filter backend- 
. crder_by ( 
) .21L() 
search fields 
search python 
'name' I 

* 1. 自定义

機器產生的替代文字:
frcn franewczk. 
class : 
def (self, Eau—st, gueryset, view) 
try: 
return 
except : 

* 1. 分页器

from rest\_framework.pagination import PageNumberPagination

機器產生的替代文字:
from rest framework. pagination import PageNumberPagination 
class StandardPageNumberPagination(PageNumberPagination 
page_slze 
size' 
page_slze_query_param 
20 
max_page_slze 
" page" 
page_query_param 

偏移条数分页器

機器產生的替代文字:
frcn L2mftOffseCVagfnaZfcn 
class C2uzseL2rn2ZOffsetPaginaZ2cn (L2n2ZOffE2ZVag2naZf2n) : 
de fault 
# ,kkcff 
guery_param 
offset Zuezy_pazar. 
max 2 
' offset ' 

* 1. 筛选器
     1. 普通筛选器

from django\_filters.rest\_framework import DjangoFilterBackend（常用）

機器產生的替代文字:
class LoginCenter(ListCreateAPIView 
POST : 
GET 
queryset User. objects. all 
serializer class 
UserSeria1ize 
filter backends 
DjangoFi1terBackend, 
( employee_id ' 
filter fields 
' state ' 
' employee _ name ' , 
pagination_class 
StandardPageNumberPagination 
' role ' 
'dept') 

* 1. 自定义

機器產生的替代文字:
class Apply ListCreateAPIView 
queryset Application. objects. all 
. order_by 
-id') 
serializer class 
filter backends 
filterset class 
pagination _ class 
ApplySeria1izer 
DjangoFi1terBackend, 
FormFi1ter 
StandardPageNumberPagination 

機器產生的替代文字:
from 
django_filters import FilterSet 
from 
django_filters . rest framework import DateFi1ter, NumberFi1ter 
from 
database. models import Application 
class FormFi1ter(Fi1terSet 
lookup_expr— ' icontains ' ) 
time apply time' , 
dept', 'status ' 
status It 
class Meta: 
model 
fields 
status ' , lookup expr- 'It' ) 
Application 
'mandator', 
' payment 

过滤器中常用的字段类型，这些类型要输模型中对应字段类型兼容

* + CharFilter 字符串类型
  + BooleanFilter 布尔类型
  + DateTimeFilter 日期时间类型
  + DateFilter 日期类型
  + DateRangeFilter 日期范围
  + TimeFilter 时间类型
  + NumberFilter 数值类型，对应模型中IntegerField, FloatField, DecimalField

参数说明：

* + field\_name: 过滤字段名，一般应该对应模型中字段名
  + lookup\_expr: 查询时所要进行的操作，和ORM中运算符一致

Meta字段说明

* + model： 引用的模型，不是字符串
  + fields：指明过滤字段，可以是列表，列表中字典可以过滤，默认是判等；也可以字典，字典可以自定义操作
  + exclude = ['password'] 排除字段，不允许使用列表中字典进行过滤

\_\_gt 大于

\_\_gte 大于等于

\_\_lt 小于

\_\_lte 小于等于

* + 序列化器
    - 序列化和反序列化

api接口开发，最核心最常见的一个过程就是序列化，所谓序列化就是把数据转换格式，序列化可以分两个阶段：

機器產生的替代文字:
anøs 
server 
Models 

序列化： 把我们识别的数据转换成指定的格式提供给别人，可选择序列化的模型表字段。（后端处理数据返回给前端）

ex：我们在django中获取到的数据默认是模型对象，但是模型对象数据无法直接提供给前端或别的平台使用，所以我们需要把数据进行序列化，变成字符串或者json数据，提供给别人。

反序列化：把别人提供的数据转换/还原成我们需要的格式，可选择对数据验证的规则。(处理前端传过来的数据)

ex：前端js提供过来的json数据，对于python而言就是字符串，我们需要进行反序列化换成模型类对象，这样我们才能把数据保存到数据库中。

* + 一个完整序列化器中应该包含四方面的内容：
    - 转换的字段声明

客户端字段 = serializer.字段类型（字段校验项=字段值）

* + 如果当前的序列化器为ModelSerializer，则需要声明调用的模型信息

class Meta:

model = 模型

fields = 字段列表

* 1. 验证代码的对象方法
     1. 多字段校验

機器產生的替代文字:
class TestSeriaIizer serializers. Serializer 
def validate (self,attrs 
pass 
return 
attrs 

* 1. 单字段校验

機器產生的替代文字:
class TestSeriaIizer serializers. Serializer 
def 
self, value 
pass 
return value 

* 1. 模型操作的方法，两个方法create和update方法

機器產生的替代文字:
def data): 
pass 
def update(self, instance, validate data) : 
pass 

说明：

* + 用于序列化时，将模型类对象传入 instance 参数；
  + 用于反序列化时，将要被反序列化的数据传入 data 参数；
  + 除了 instance 和 data 参数外，在构造 Serializer 对象时，还可以通过 context 参数额外添加数据，通过 context 添加的数据，可以通过 Serializer 对象的 context 属性获取；
  + 模型序列化器：

機器產生的替代文字:
# serializers.py 
class TeacherMode1SeriaIizer serializers. Model Seriali zer 
class Meta 
# Django €4, 
model Teacher 
"nickname") 
# fields : 
all 
'age 
sex" 
fields 
C"id" , 
'name" 
"classmate 
read only _ fields 
C'id', 
date_j oi ned 
# tÉ5JL'ÅtE%exc 
# exc lude 
: Cid' j 
extra kwargs 
name 
required' : 
'age 
"max value" 
"min value" 
True 
"must younger than 28" , 
"must older than , 
'error_messages 
"max value" . 
"min value": 

注：

* + extra\_kwargs = { # 类似于这种形式name=serializers.CharField(max\_length=16,min\_length=4)

‘max\_length’：16，

'min\_length'：4

'price': {'write\_only': True},

}

* + write\_only\_fields=('id',) #弃用了，使用extra\_kwargs
  + 需要返回模型中没有的字段时

ex：外键的写法（缺点：只能用来序列化，默认只读）

機器產生的替代文字:
class UserSeria1ize(seria1izers . ModelSeria1izer 
You, | 1 author (You) 
class Meta: 
model 
fields 
User 
all 
extra_kwargs 
( 'password 'J) 
# exclude 
role name 
source— 
dept _ name 
source— 
serializers. CharFie1d 
' role. name', required—False, read only-True) 
serializers. CharFie1d 
' dept . name', required—False, read only-True) 
obj): 
def get role_name(self, 
return obj.role. name 
def get dept_name(self, obj): 
return obj . dept. name 

选项参数：

|  |  |
| --- | --- |
| **参数名称** | **作用** |
| **max\_length** | 最大长度 |
| **min\_lenght** | 最小长度 |
| **allow\_blank** | 是否允许为空 |
| **trim\_whitespace** | 是否截断空白字符 |
| **max\_value** | 最小值 |
| **min\_value** | 最大值 |

通用参数：

機器產生的替代文字:
read_only 
write_only 
required 
default 
allow null 
validators 
error messages 
label 
help text 
*iAFalse 

备注：高级用法sourse

* + 反序列化：

ex：

機器產生的替代文字:
self data—I 
serializer 
"last login time": timezone.now().strftime( '%Y-%m-%d ) ) , 
serializer.is_valid(raise exception: True) 
serializer. save() 
partial—True) 

機器產生的替代文字:
serializer.is valid 
serializer . is valid raise exception: True) 

校验多个对象时，需要加 many=True

ex:

機器產生的替代文字:
student list 
Student. obj ects 
. all C) 
seri al i zer 
Studentserializer(instancezstudent list 
many—True) 

补充说明：

* + 在对序列化器进行 save() 保存时，可以额外传递数据，这些数据可以在 create() 和 update() 中的 validated\_data 参数中获取到

機器產生的替代文字:
1 serializer. save owner-request. user 

* + 默认序列化器必须传递所有的 required 字段，否则会抛出校验异常，但是可以使用 partial 参数来允许部分字段更新

更新 name，不需要校验其他字段，可以设置 partial=True

機器產生的替代文字:
self. Jet data—I 
serializer 
timezone.now().strftime( '%Y-%m-%d ' ) ) , partial—True) 
"last login time" 

tesy

2022年9月19日

9:18

from rest\_framework.filters import OrderingFilter,SearchFilter

*# filter\_backends = [SearchFilter]*

*# search\_fields = ['name']*

*# filter\_backends = [OrderingFilter,]*

*# ordering\_fields = ['id'*