day81 drf

内容回顾和补充

1. 什么是restful规范

```
是一套规则，用于程序之间进行数据交换的约定。
他规定了一些协议，对我们感受最直接的的是，以前写增删改查需要写4个接口，restful规范的就是1个接口，根据method的不同做不同的操作，比如：get/post/delete/put/patch/delete.
初次之外，resetful规范还规定了：
    - 数据传输通过json

扩展：前后端分离、app开发、程序之间（与编程语言无关）
```

```
JSON:
    {
        name:'alex',
        age:18,
        gender:'男'
    }
```

```
以前用webservice，数据传输格式xml。
XML
    <name>alex</name>
    <age>alex</age>
    <gender>男</gender>
```

2. 什么是drf?

```
drf是一个基于django开发的组件，本质是一个django的app。
drf可以办我们快速开发出一个遵循restful规范的程序。
```

3. drf如何帮助我们快速开发的？drf提供了那些功能?

```
- 视图，APIView用处还不知道。
- 解析器，根据用户请求体格式不同进行数据解析，解析之后放在request.data中。
        在进行解析时候，drf会读取http请求头 content-type.
        如果content-type:x-www-urlencoded，那么drf会根据 & 符号分割的形式去处理请求体。
            user=wang&age=19

        如果content-type:application/json,那么drf会根据 json 形式去处理请求体。
         {"user":"wang","age":19}
- 序列化，可以对QuerySet进行序列化，也可以对用户提交的数据进行校验。
- 渲染器，可以帮我们把json数据渲染到页面上进行友好的展示。（内部会根据请求设备不同做不同的展示）
```

4. 序列化：many=True or False
5. 序列化：展示特殊的数据（choices、FK、M2M）可使用

```
depth
source，无需加括号，在源码内部会去判断是否可执行，如果可执行自动加括号。【fk/choice】
SerializerMethodField，定义钩子方法。【m2m】
```

6. 写程序的潜规则：约束

```
# 约束子类中必须实现f1
class Base(object):
    def f1(self):
        raise NotImplementedError('asdfasdfasdfasdf')

class Foo(Base):

    def f1(self):
        print(123)


obj = Foo()
obj.f1()
```

7. 面向对象的继承

```
class Base(object):
    def f1(self):
        print('base.f1')
        self.f2()
    def f2(self):
        print('base.f2')
class Foo(Base):
    def f2(self):
        print('foo.f2')

obj = Foo()
obj.f1()
```

```
class Base(object):
    x1 = 123

    def f1(self):
        print(self.x1)

class Foo(Base):
    x1 = 456

obj = Foo()
obj.f1()
```

```python
class APIView(object):
    version_class = 123

    def get_version(self):
        print(self.version_class)

class UserView(APIView):
    version_class = 666

obj = UserView()
obj.get_version()
```

```python
class APIView(object):
    version_class = 123

    def dispatch(self,method):
        self.initial()
        getattr(self,method)()

    def initial(self):
        print(self.version_class)

class UserView(APIView):
    version_class = 666

    def get(self):
        print('userview.get')

obj = UserView()
obj.dispatch('get')
```

```python
class URLPathVersion(object):

    def determin_version(self):
        return 'v1'



class APIView(object):
    version_class = None

    def dispatch(self,method):
        version = self.initial()
        print(version)
        getattr(self,method)()

    def initial(self):
        self.process_version()

    def process_version():
        obj = self.version_class()
        return obj.determine_version()

class UserView(APIView):
    version_class = URLPathVersion
```

```
    def get(self):
        print('userview.get')

obj = UserView()
obj.dispatch('get')
```

今日概要

1. 上节作业
2. 分页
3. 筛选
4. 视图

今日详细

1.上节作业

```
    url(r'^new/article/$', views.NewArticleView.as_view()),
    url(r'^new/article/(?P<pk>\d+)/$', views.NewArticleView.as_view()),
```

```
class NewArticleView(APIView):

    def get(self,request,*args,**kwargs):
        pk = kwargs.get('pk')
        if not pk:
            queryset = models.Article.objects.all()
            ser = serializer.NewArticleSerializer(instance=queryset,many=True)
            return Response(ser.data)
        article_object = models.Article.objects.filter(id=pk).first()
        ser = serializer.NewArticleSerializer(instance=article_object,
many=False)
        return Response(ser.data)

    def post(self,request,*args,**kwargs):
        ser = serializer.FormNewArticleSerializer(data=request.data)
        if ser.is_valid():
            ser.save()
            return Response(ser.data)
        return Response(ser.errors)

    def put(self, request, *args, **kwargs):
        """全部更新"""
        pk = kwargs.get('pk')
        article_object = models.Article.objects.filter(id=pk).first()
        ser = serializer.FormNewArticleSerializer(instance=article_object,
data=request.data)
        if ser.is_valid():
```

```
            ser.save()
            return Response(ser.data)
        return Response(ser.errors)

    def patch(self,request,*args,**kwargs):
        """局部"""
        pk = kwargs.get('pk')
        article_object = models.Article.objects.filter(id=pk).first()
        ser = serializer.FormNewArticleSerializer(instance=article_object,
data=request.data,partial=True)
        if ser.is_valid():
            ser.save()
            return Response(ser.data)
        return Response(ser.errors)

    def delete(self,request,*args,**kwargs):
        pk = kwargs.get('pk')
        models.Article.objects.filter(id=pk).delete()
        return Response('删除成功')
```

```
class NewArticleSerializer(serializers.ModelSerializer):
    tag_info = serializers.SerializerMethodField()
    class Meta:
        model = models.Article
        fields = ['title','summary','tag_info']

    def get_tag_info(self,obj):
        return [row for row in obj.tag.all().values('id','title')]

class FormNewArticleSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.Article
        fields = '__all__'
```

2.分页

2.1 PageNumberPagination

- 配置 settings.py

```
REST_FRAMEWORK = {
    "PAGE_SIZE":2
}
```

- 在视图的列表页面

```
from rest_framework.pagination import PageNumberPagination
from rest_framework import serializers

class PageArticleSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.Article
        fields = "__all__"

class PageArticleView(APIView):
```

```
    def get(self,request,*args,**kwargs):
        queryset = models.Article.objects.all()

        # 方式一：仅数据
        """
        # 分页对象
        page_object = PageNumberPagination()
        # 调用 分页对象.paginate_queryset方法进行分页，得到的结果是分页之后的数据
        # result就是分完页的一部分数据
        result = page_object.paginate_queryset(queryset,request,self)
        # 序列化分页之后的数据
        ser = PageArticleSerializer(instance=result,many=True)
        return Response(ser.data)
        """
        # 方式二：数据 + 分页信息
        """
        page_object = PageNumberPagination()
        result = page_object.paginate_queryset(queryset, request, self)
        ser = PageArticleSerializer(instance=result, many=True)
        return page_object.get_paginated_response(ser.data)
        """
        # 方式三：数据 + 部分分页信息

        page_object = PageNumberPagination()
        result = page_object.paginate_queryset(queryset, request, self)
        ser = PageArticleSerializer(instance=result, many=True)
        return
Response({'count':page_object.page.paginator.count,'result':ser.data})
```

2.2 LimitOffsetPagination

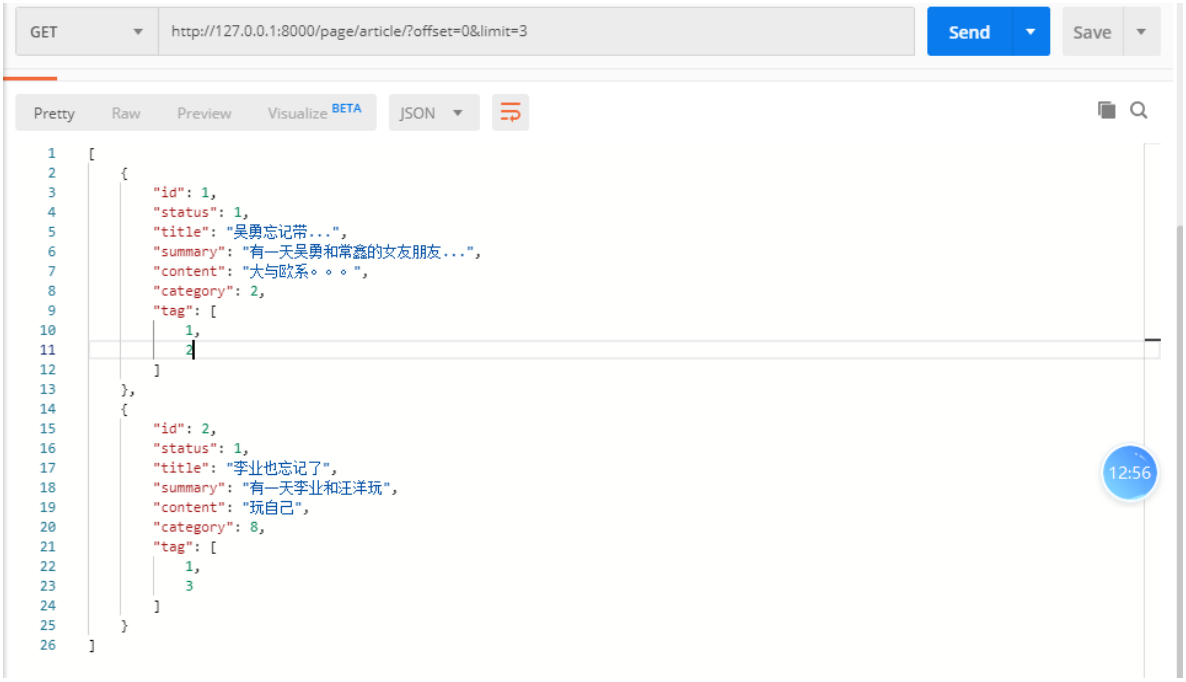
```
from rest_framework.pagination import PageNumberPagination
from rest_framework.pagination import LimitOffsetPagination
from rest_framework import serializers

class PageArticleSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.Article
        fields = "__all__"

class HulaLimitOffsetPagination(LimitOffsetPagination):
    max_limit = 2

class PageArticleView(APIView):
    def get(self,request,*args,**kwargs):
        queryset = models.Article.objects.all()

        page_object = HulaLimitOffsetPagination()
        result = page_object.paginate_queryset(queryset, request, self)
        ser = PageArticleSerializer(instance=result, many=True)
        return Response(ser.data)
```



扩展：

```
url(r'^page/view/article/$', views.PageViewArticleView.as_view()),
```

```
from rest_framework.generics import ListAPIView

class PageViewArticleSerializer(serializers.ModelSerializer):
    class Meta:
        model = models.Article
        fields = "__all__"

class PageViewArticleView(ListAPIView):
    queryset = models.Article.objects.all()
    serializer_class = PageViewArticleSerializer
```

```
REST_FRAMEWORK = {
    "PAGE_SIZE":2,
    "DEFAULT_PAGINATION_CLASS":"rest_framework.pagination.PageNumberPagination"
}
```

作业：实现呼啦圈【小组实现】

- 设计表结构
- 文章列表接口 + 分页
- 文章详细页面
  - 文章详细信息
  - 评论信息
- 提交评论