项目开发实战：

Django 前后端分离开发新闻系统

导师：Andy

2018-06

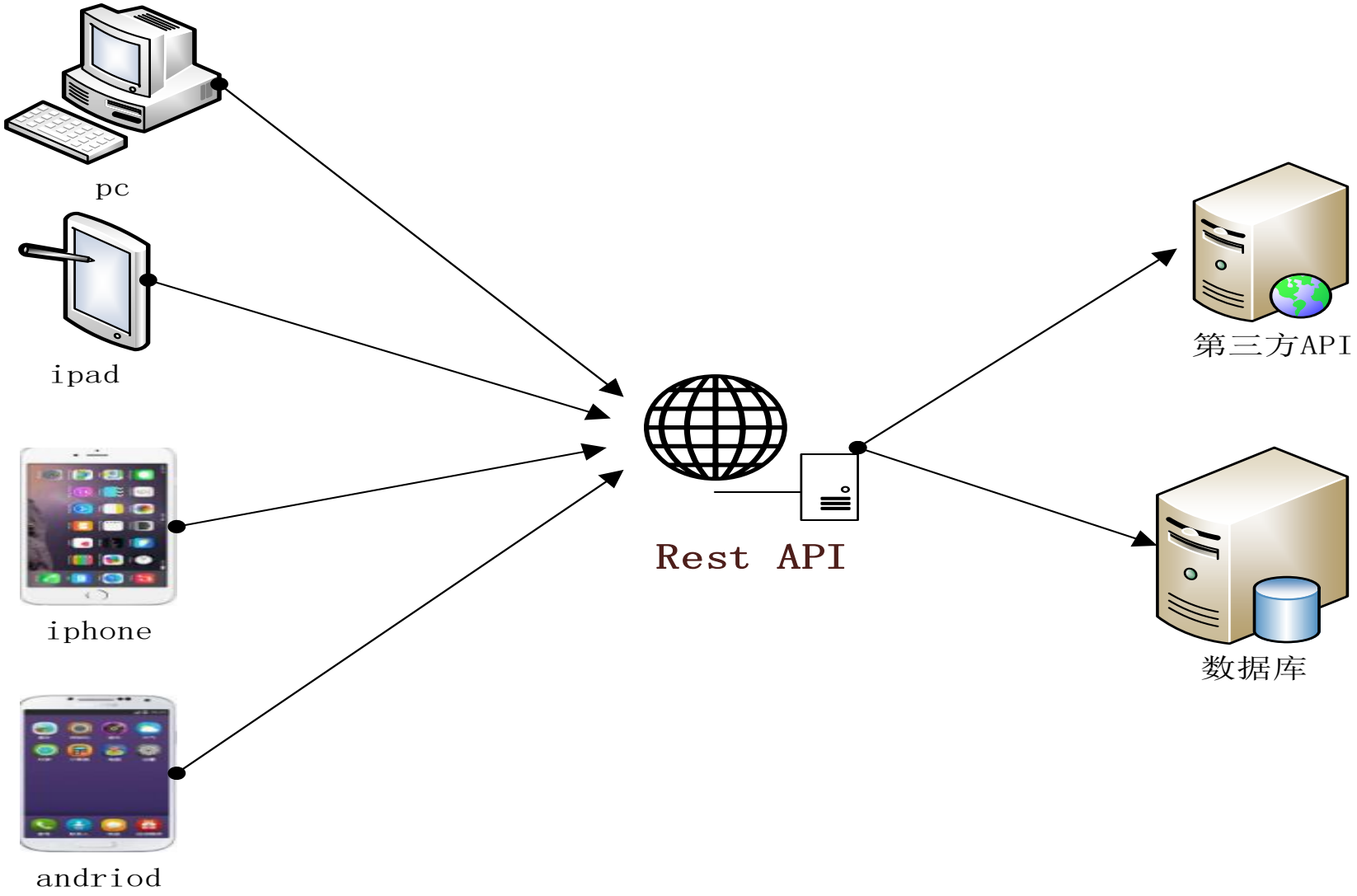
Django Rest Framework介绍

Django REST Framework（简称DRF），是一个用于构建Web API的强大且灵活的工具包。

主要实现前后端分离，方便提供接口API开发,并遵从了REST API的标准规范

REST是一种Web API设计标准，是目前比较成熟的一套互联网应用程序的API设计理论。

前后端分离框架

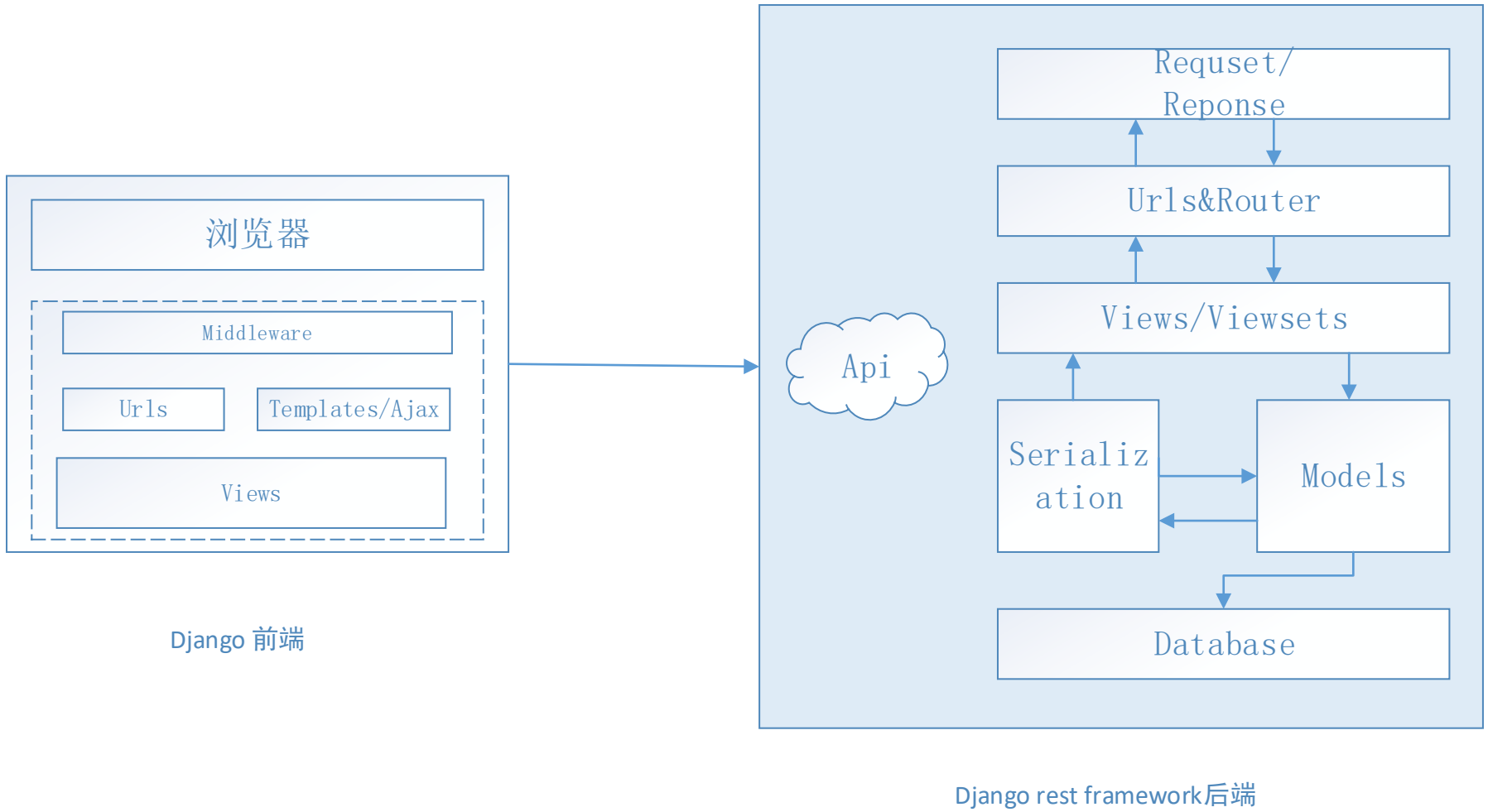


为什么要前后端分离

前后端职责清晰了

后端	前端
• 提供数据	• 接收数据，返回数据
• 处理业务逻辑	• 处理渲染逻辑
• Server-side MVC架构	• Client-side MV* 架构
• 代码跑在服务器上	• 代码跑在浏览器上

Django 实现新闻系统前后端分离框架



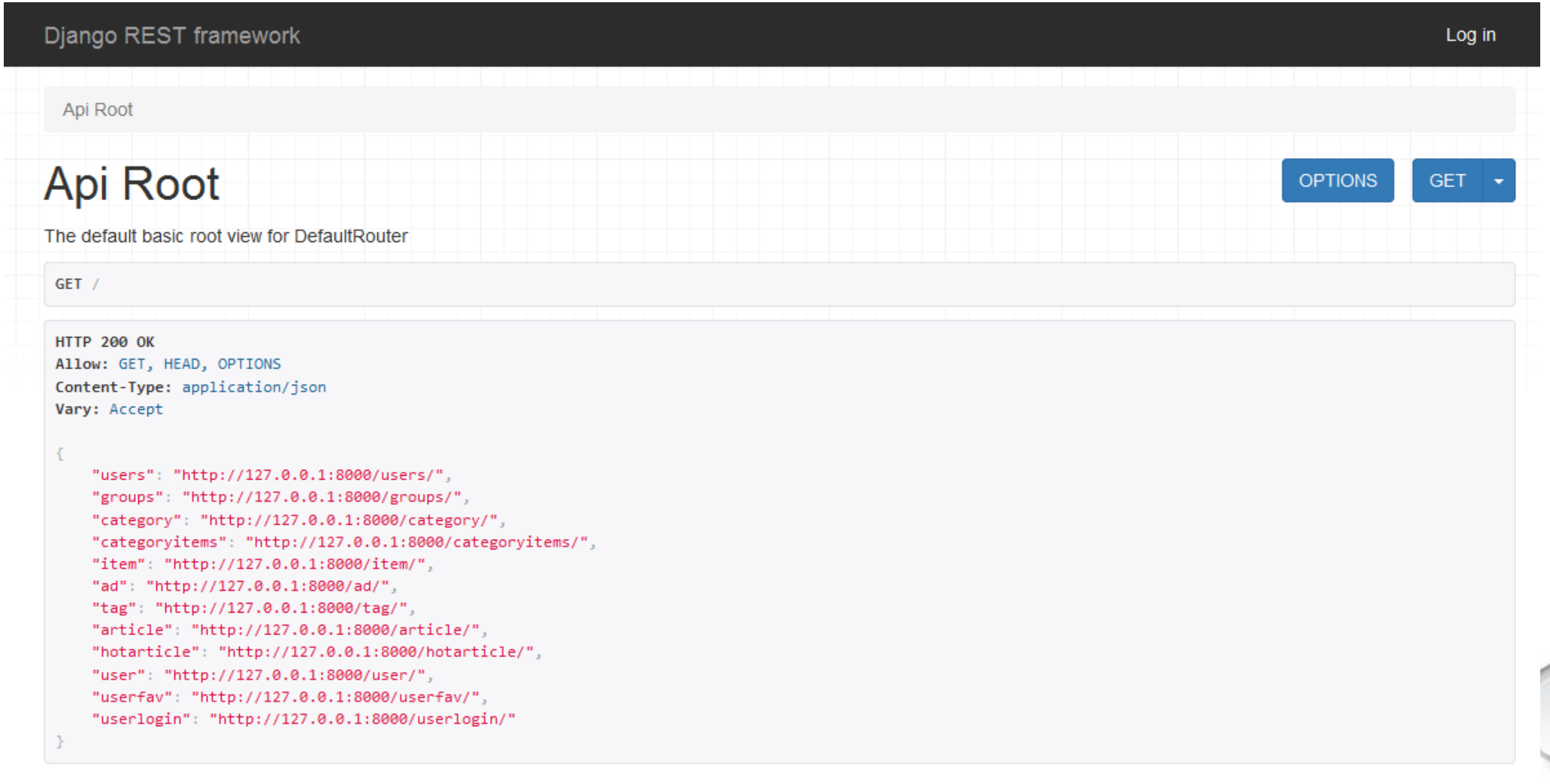
Django 前端

Django rest framework后端

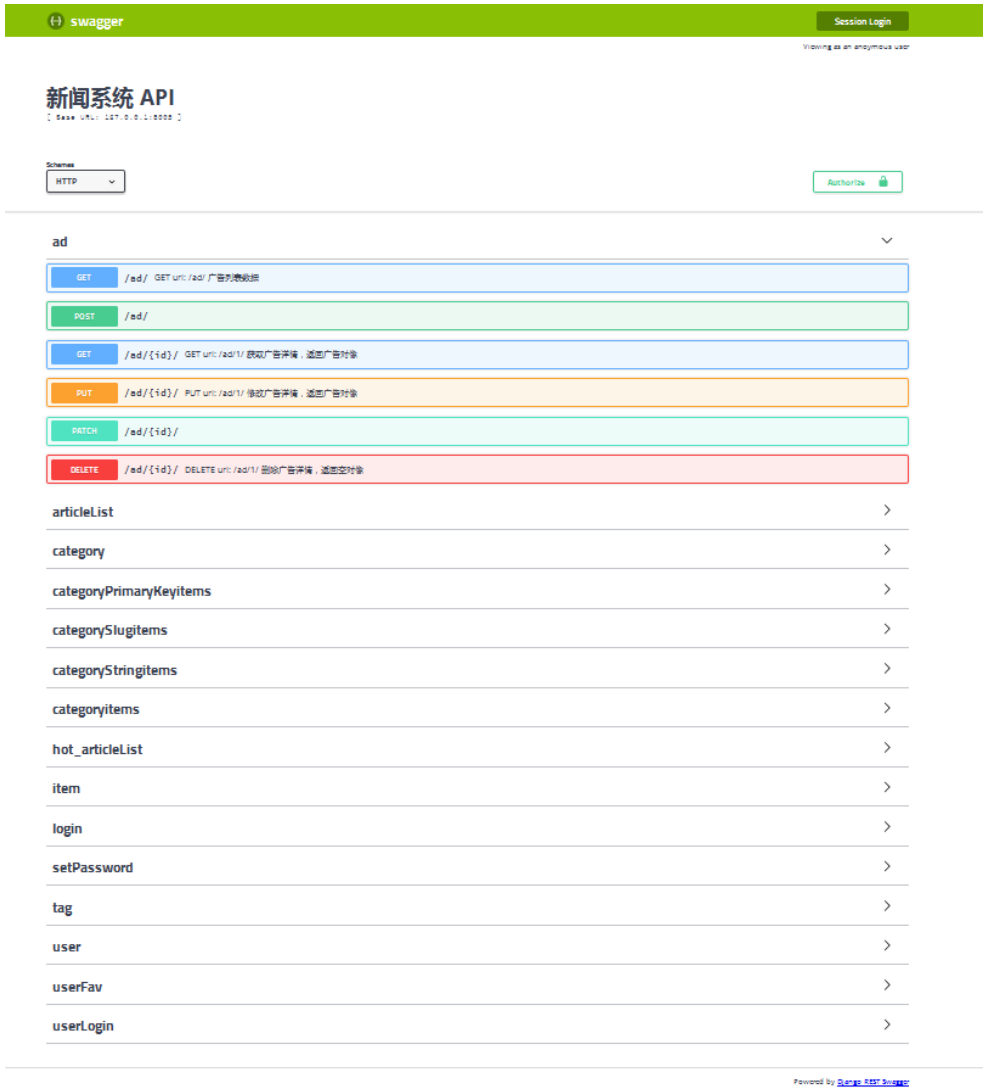
如何用DRF实现api编程

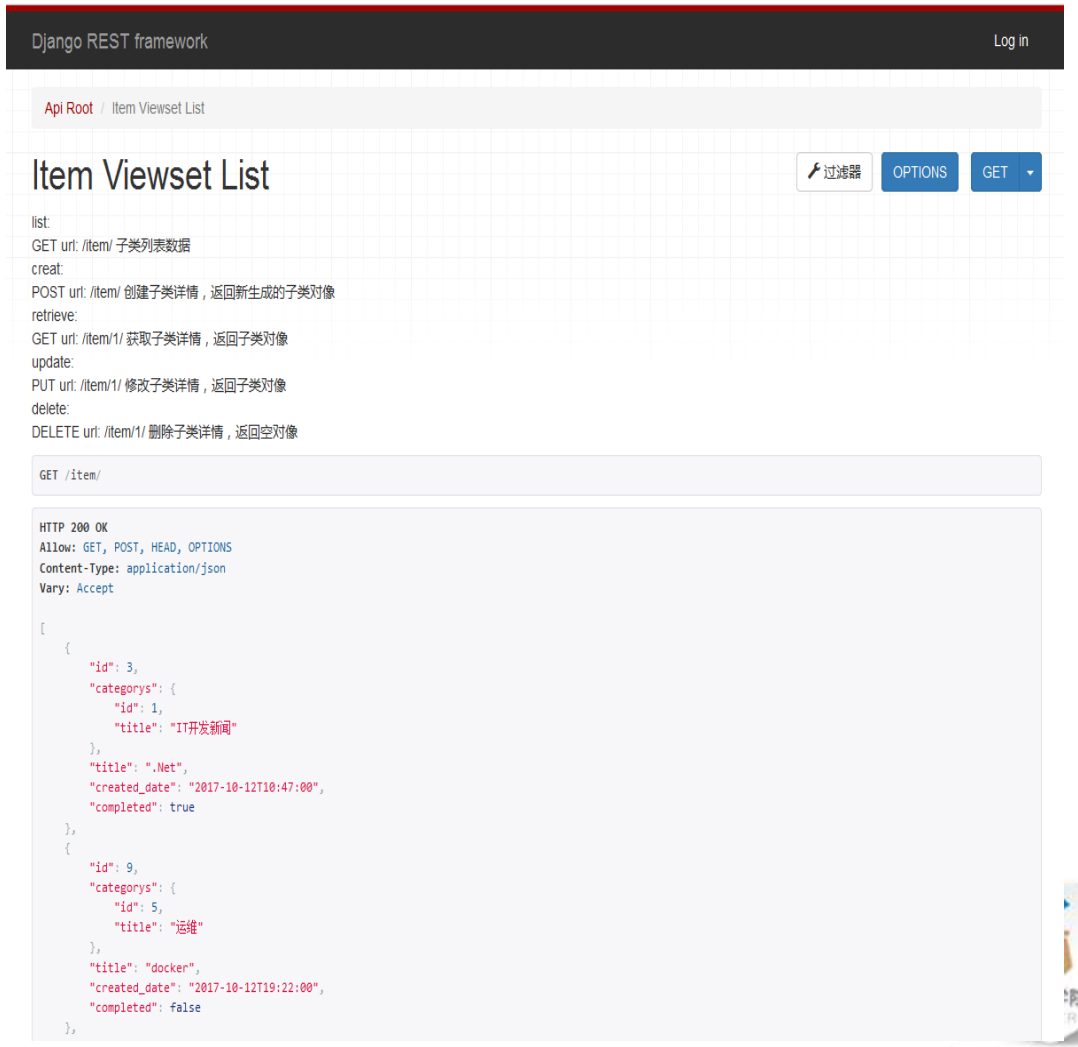
- 接口设计
- 序列化
- Views的快速实现
- 统一的参数校验
- 灵活的查询过滤
- 异常处理
- api权限、token认证
- 分页
- 限流
- URL与路由设置
- 缓存如何可以做的更简单统一

实用API Root


Django REST framework
Log in
Api Root
Api Root
OPTIONS
GET
The default basic root view for DefaultRouter
GET /
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"users": "http://127.0.0.1:8000/users/",
"groups": "http://127.0.0.1:8000/groups/",
"category": "http://127.0.0.1:8000/category/",
"categoryitems": "http://127.0.0.1:8000/categoryitems/",
"item": "http://127.0.0.1:8000/item/",
"ad": "http://127.0.0.1:8000/ad/",
"tag": "http://127.0.0.1:8000/tag/",
"article": "http://127.0.0.1:8000/article/",
"hotarticle": "http://127.0.0.1:8000/hotarticle/",
"user": "http://127.0.0.1:8000/user/",
"userfav": "http://127.0.0.1:8000/userfav/",
"userlogin": "http://127.0.0.1:8000/userlogin/"
}

实用API Root





前后端分离适用场景

- 公司多平台应用统一后端
- 对外提供接口
- 微信应用
- 小程序应用

加入课程学习干货

加入课程，学习干货，让我们一起变好！！！

源码下载和咨询学习群

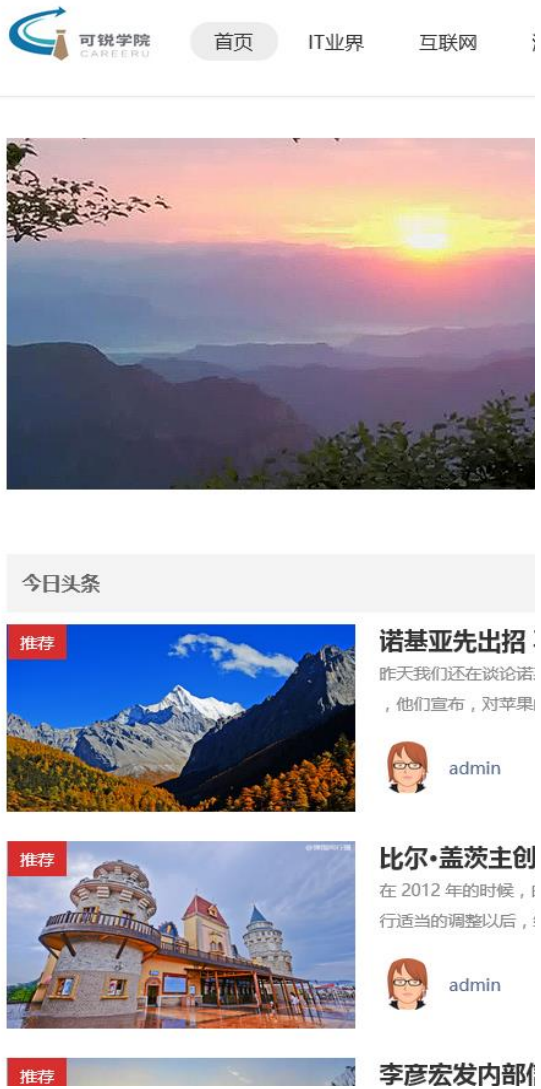
QQ服务群:631575625

可锐职场—传承职场经验，助您提升职场竞争力！

目 录

新闻管理系统功能需求

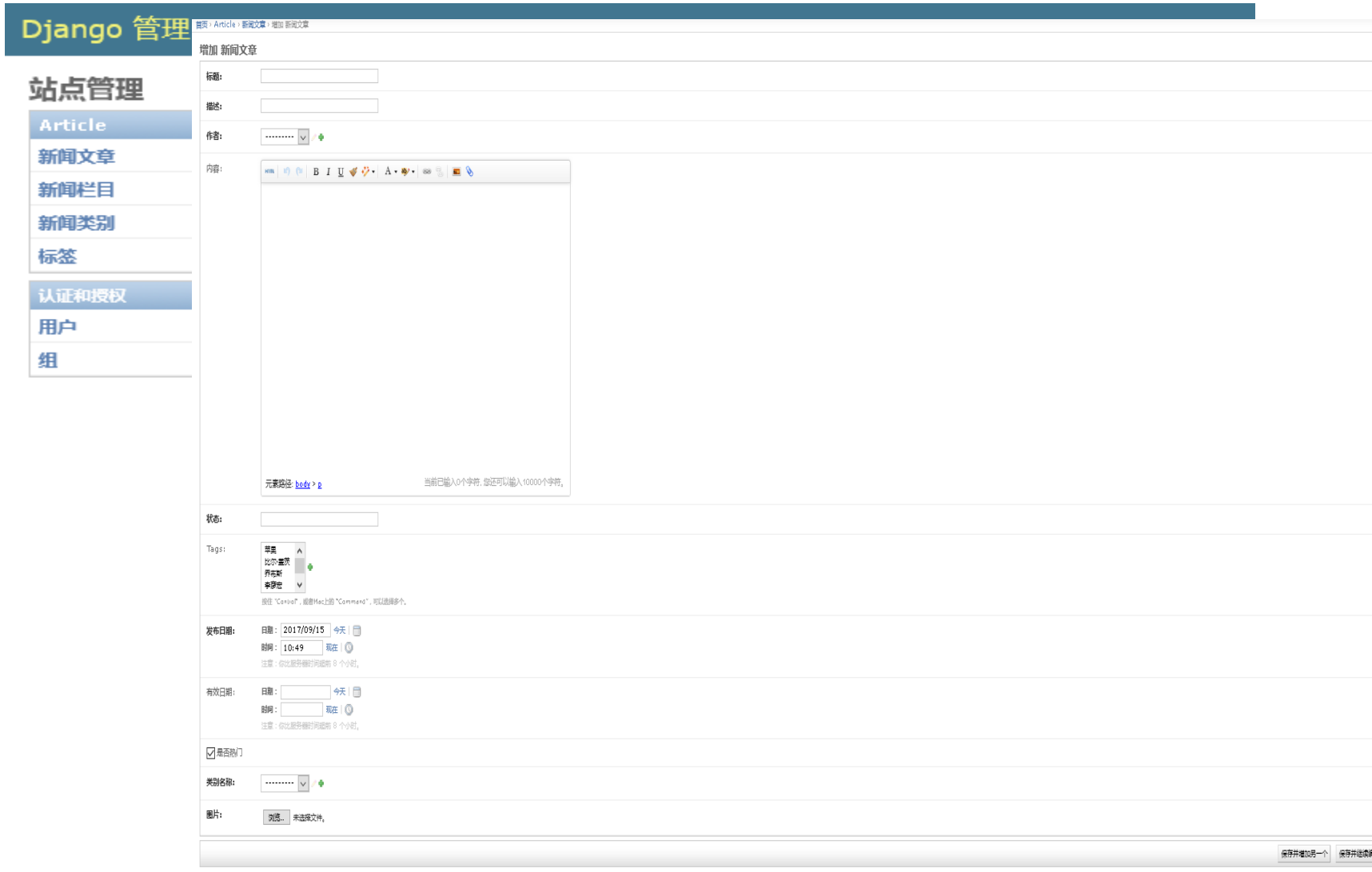
- 新闻WEB站点
  - 首页
  - 按主栏目显示新闻列表
  - 按子栏目显示新闻列表
  - 按标签显示新闻列表
  - 热门新闻列表
  - 新闻详情页面
  - 主广告轮播
  - 广告图片显示
  - 查询新闻
  - 新闻列表分页



可锐职场—传承职场经验，助您提升职场竞争力！

新闻管理系统功能需求

- 后台管理部分
  - 栏目类别管理
  - 子栏目管理
  - 标签管理
  - 发布新闻
  - 新闻管理
  - 作者管理



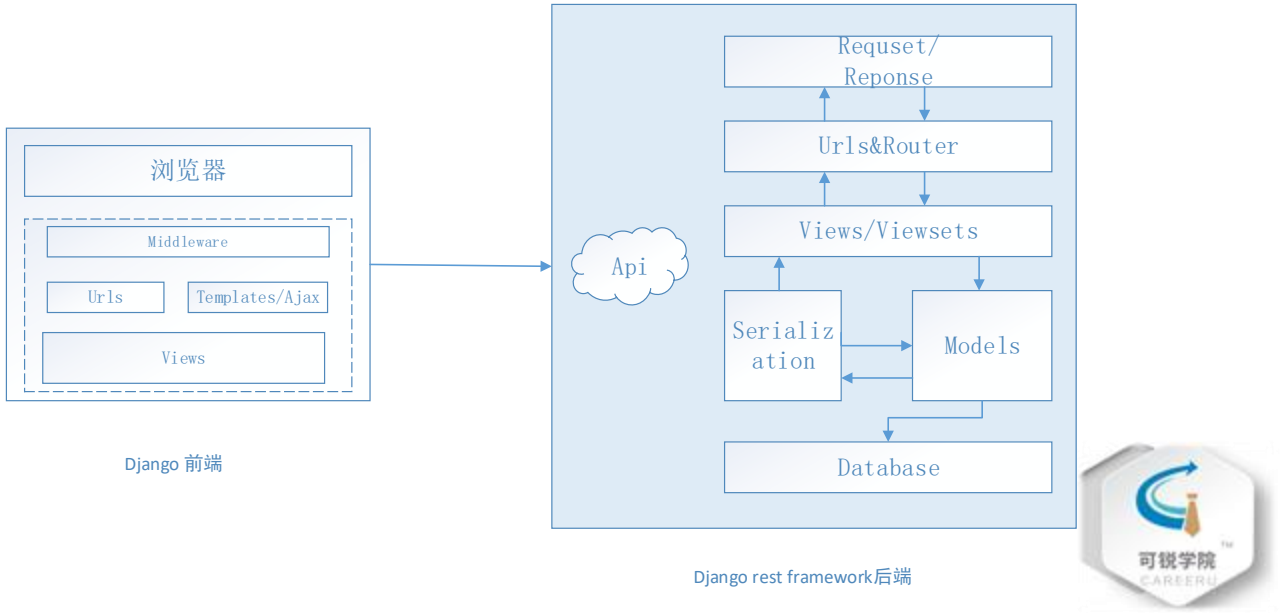
如何实现前后端分离？

- 后端专注于：

  后端控制层（Restful API） & 服务层 & 数据访问层；

- 前端专注于：

  前端控制层& 视图层（展现）



如何实现前后端分离？

- 四个阶段

1. 项目设计阶段，前后端架构负责人将项目整体进行分析，共同讨论并确定API功能、标准、返回数据内容、数据格式等；设计确定后，前后端人员共同制定开发接口。
2. 项目开发阶段，前后端分离是各自分工，协同敏捷开发，后端提供Restful API，并给出详细文档说明，前端人员进行页面渲染，前台的任务是发送API请求(GET,PUT,POST,DELETE等)获取数据（json，xml）后渲染页面。
   - 当然并不是所有的接口都可以提前定义完整，有一些是须在开发过程中进行调整的。
3. 项目测试阶段，API完成之前，前端人员会使用mock server进行模拟测试，后端人员采用junit进行API单元测试，不用互相等待；API完成之后，前后端再对接测试，联调到所制定要求就可以了。
4. 项目部署阶段，利用nginx 做反向代理，即apache2+ nginx 方式进行,分别部署前、后端应用。

目录

本课程所需环境

- 开发工具Pycharm
- 开发环境为

  Windows10、Python3.5、Django2.0.3、djangorestframework(3.8.2)

- 数据库： Mysql5.6
- 前端： Div+Css 、ajax 、 Django2.0.3
- 后端： Django2.0.3、djangorestframework(3.8.2)

软件安装和环境准备

- windows环境
  - 安装python环境（3.5）
    - 建议使用3.5下载地址： https://www.python.org/downloads/
    - windows下安装比较简单，只需一直下一步，最后导出环境变量就好了：
    - D:\Program Files (x86)\python35; D:\Program Files (x86)\python35\scripts;
  - **安装Django-2.0.3**

    pip install Django==2.0.3

    或源码安装

    下载地址：https://www.djangoproject.com/download/
    - 直接将下载的Django-2.0.3.tar.gz解压python的安装目录下，在cmd中进入该目录，输入：
      python setup.py install
    - 最后，把D:\Program Files (x86)\python27\scripts;加入环境变量PATH中
    - 验证安装

      如果想验证是否成功安装了 Django，可以在终端输入 python。然后在 Python 提示符下，尝试导入 Django：

      ```
      >>> import django
      >>> print(django.get_version())
      ```

      2.0.3

软件安装和环境准备

- **创建虚拟环境及安装相关依赖包**

//创建虚拟环境文件夹

pychamr 创建工程时创建

//创建虚拟环境

env\Scripts\activate

//安装Django和restframework框架 virtualenv

pip install django **==2.0.3**

pip install djangorestframework==3.8.2

pip install pymsql

pip install Markdown

pip install django-filter

pip install django-crispy-forms

软件安装和环境准备

- 安装mysql5.6

安装步骤：https://blog.csdn.net/zhouzezhou/article/details/52446608

创建数据库

mysql>CREATE DATABASE IF NOT EXISTS newstwo default charset utf8 COLLATE utf8_general_ci;

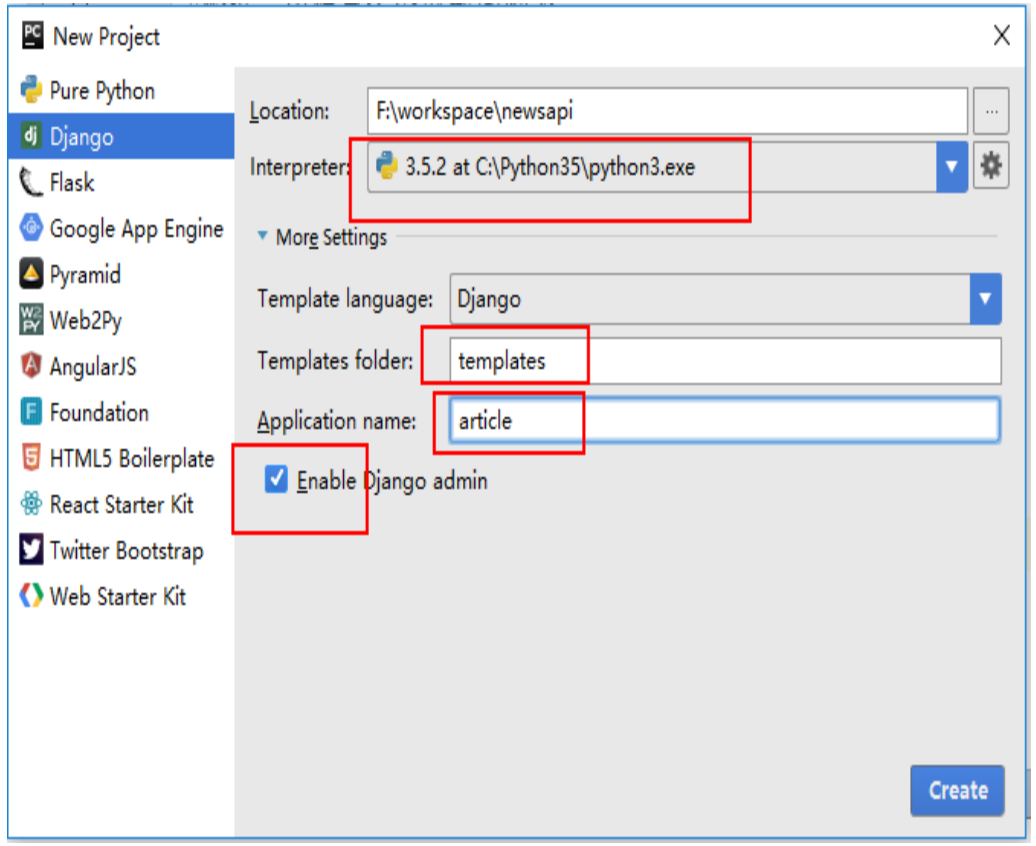
- 开发工具Pycharm安装

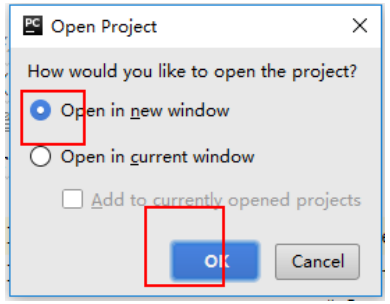
https://www.jetbrains.com/pycharm/download/#section=windows

下载后，点击安装，点击next,直到安装完成。

1、创建工程

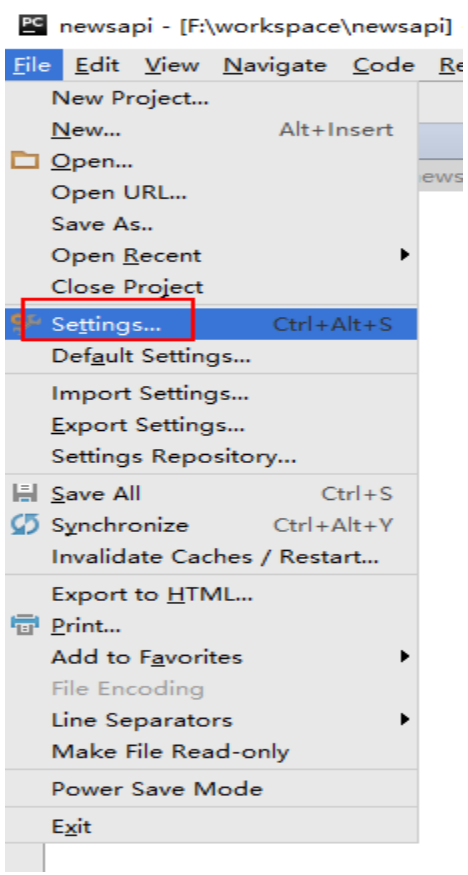
创建django工程， 如下图所示：

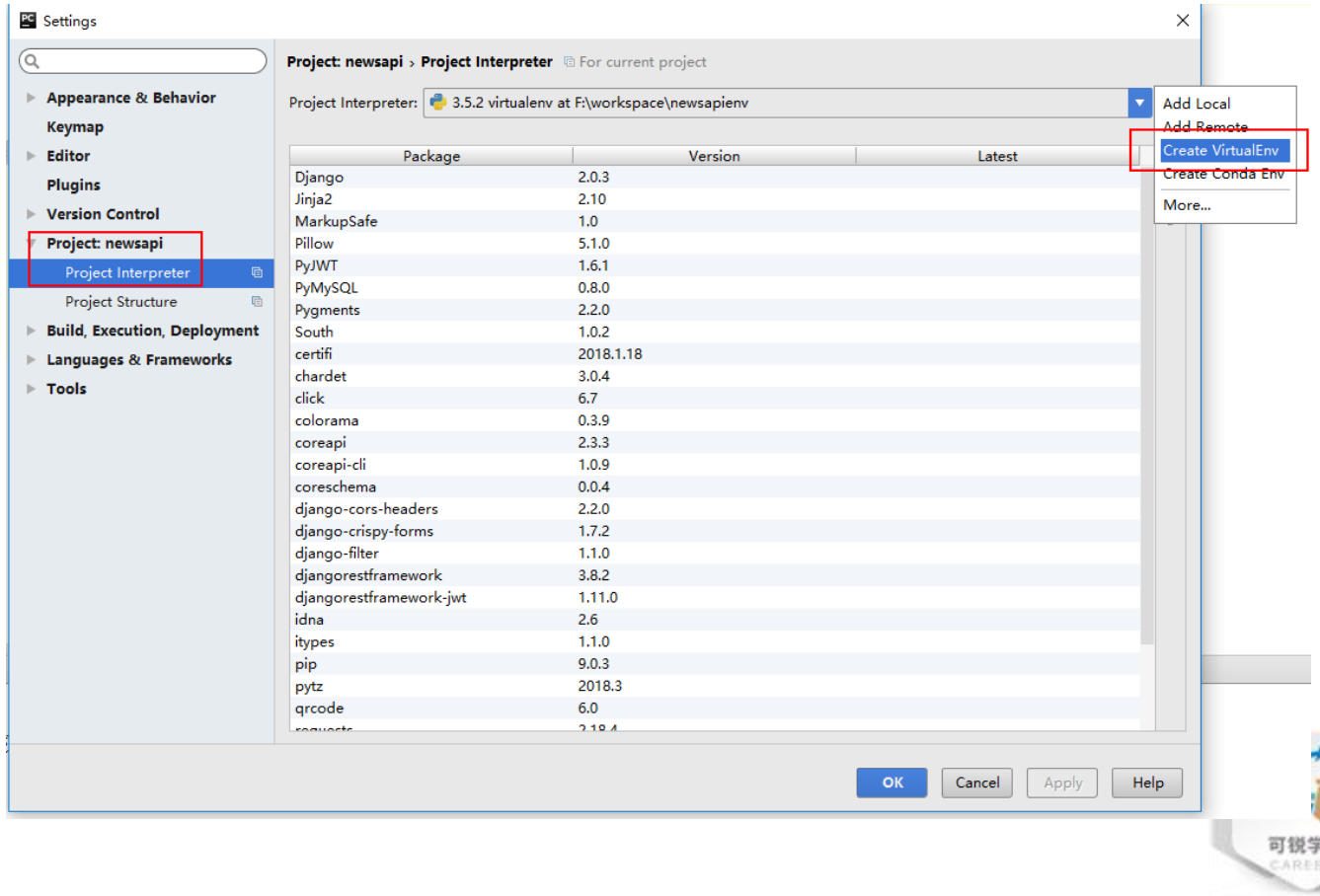




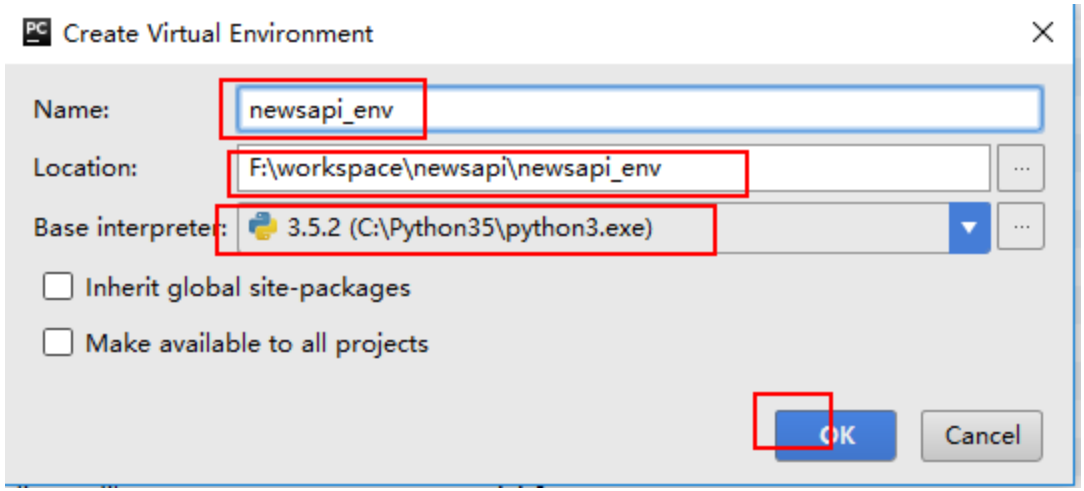
创建虚拟环境

在settings中创建虚拟环境， 如下图所示：

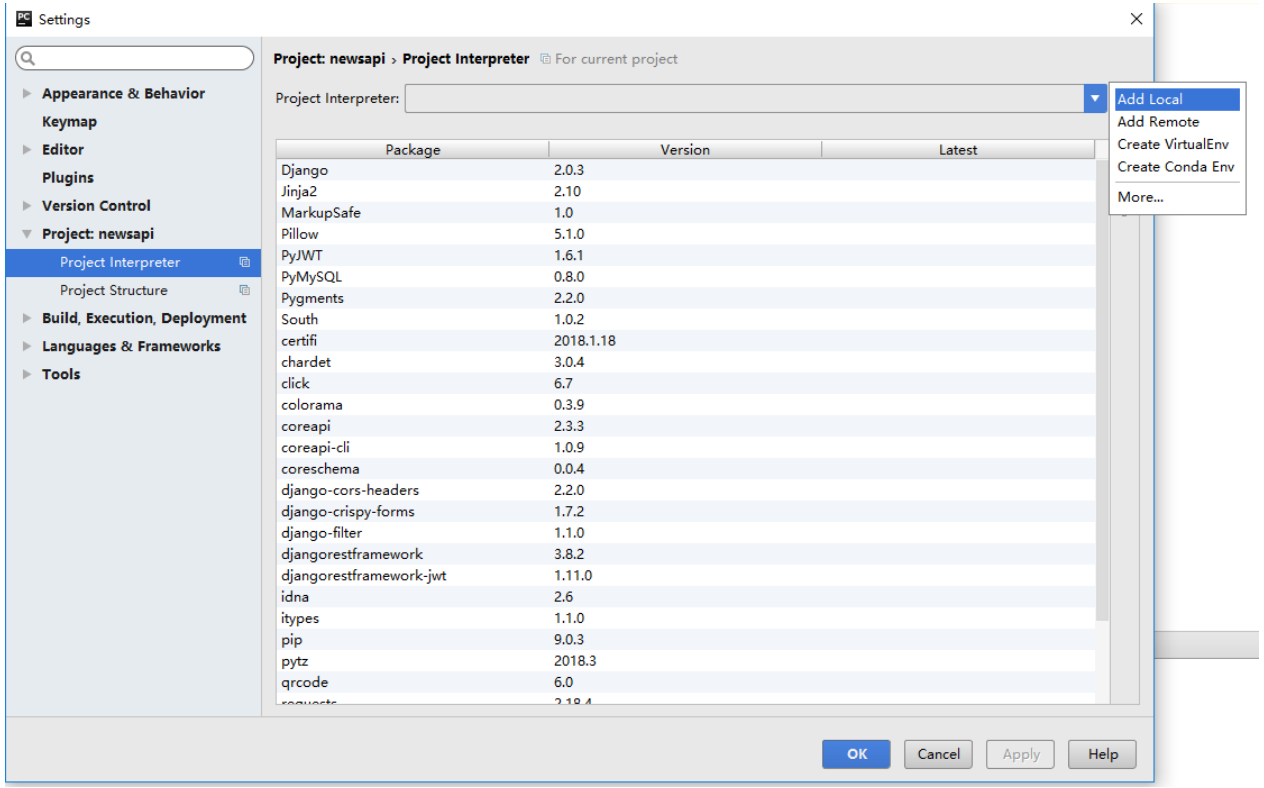


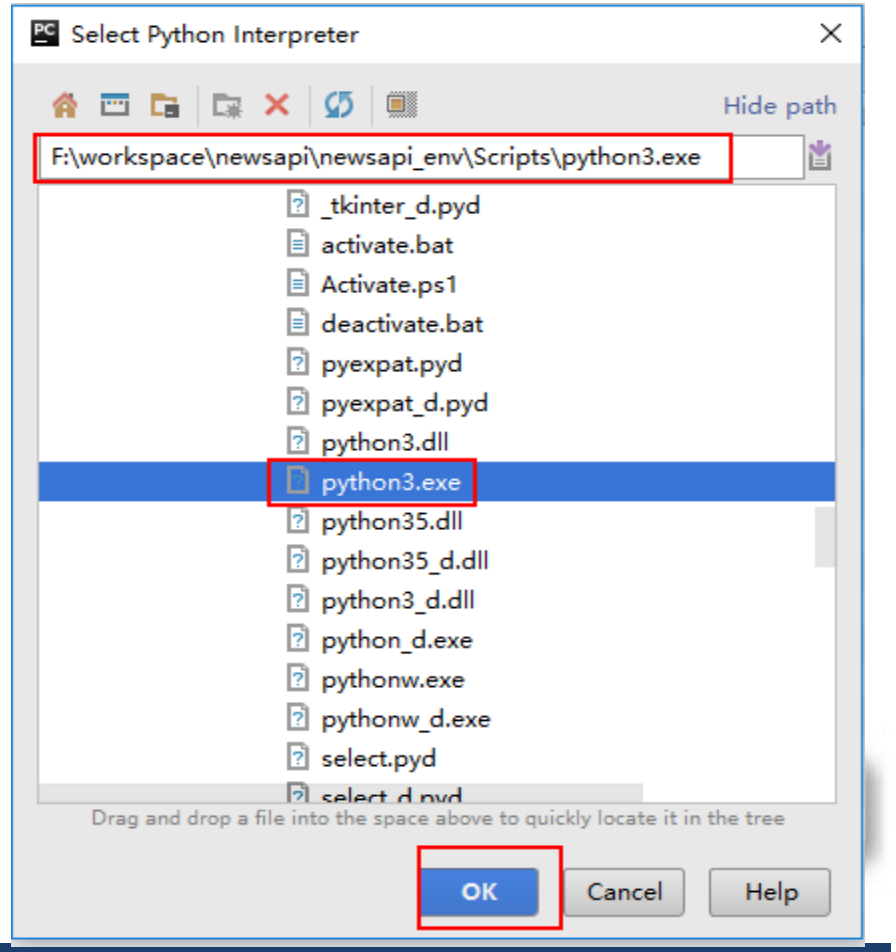


指定虚拟环境名称、路径， 如下图所示：

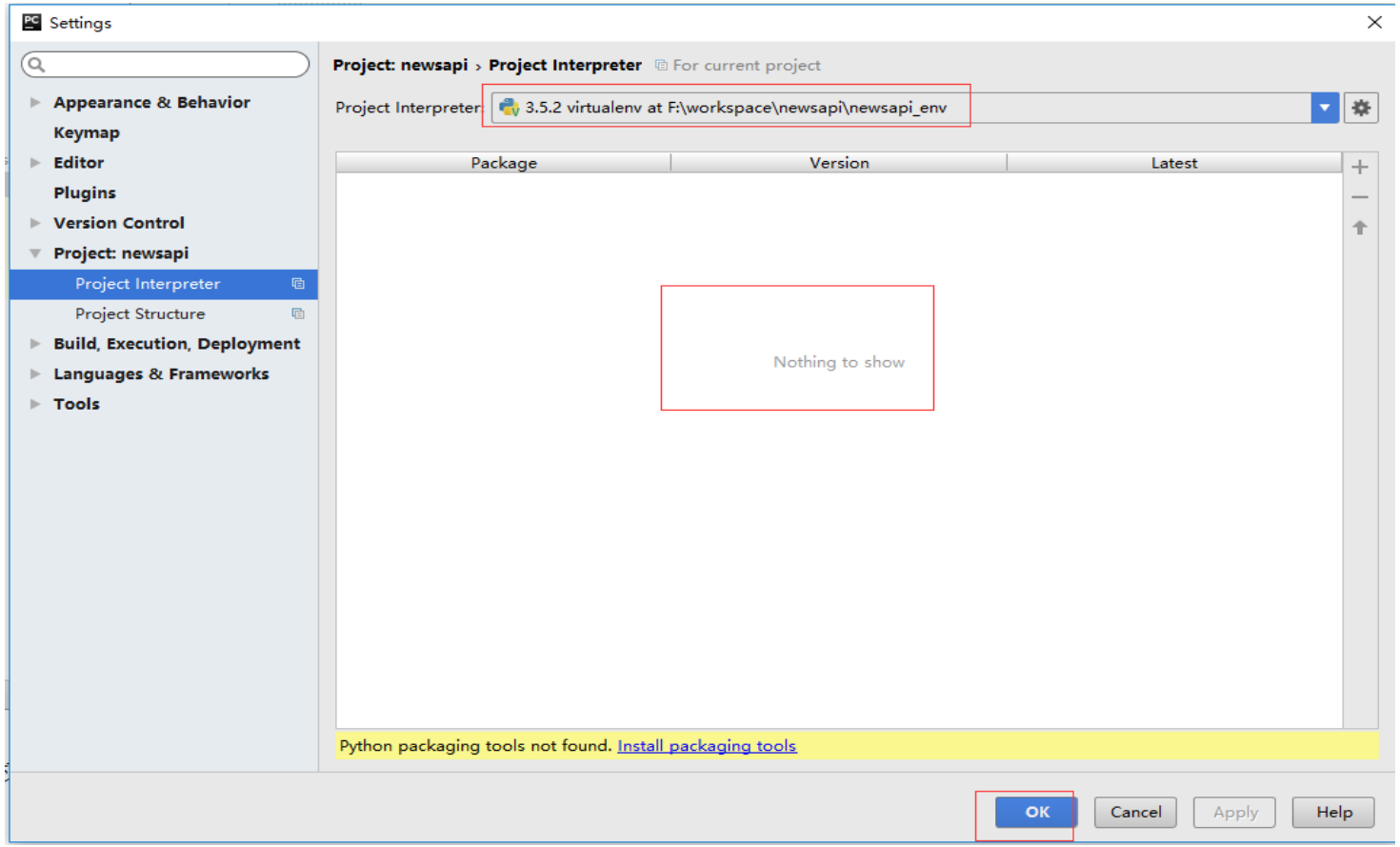


将解释器指向我们创建环境目录下scripts的python3.exe， 如下图所示：

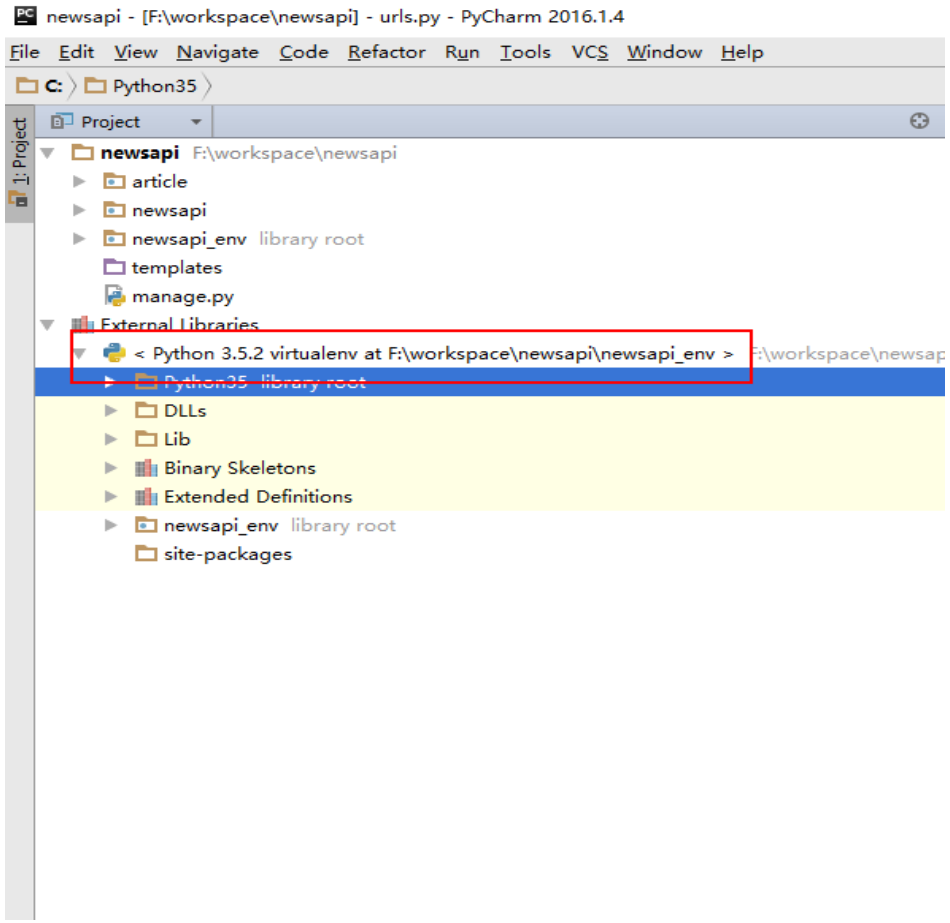




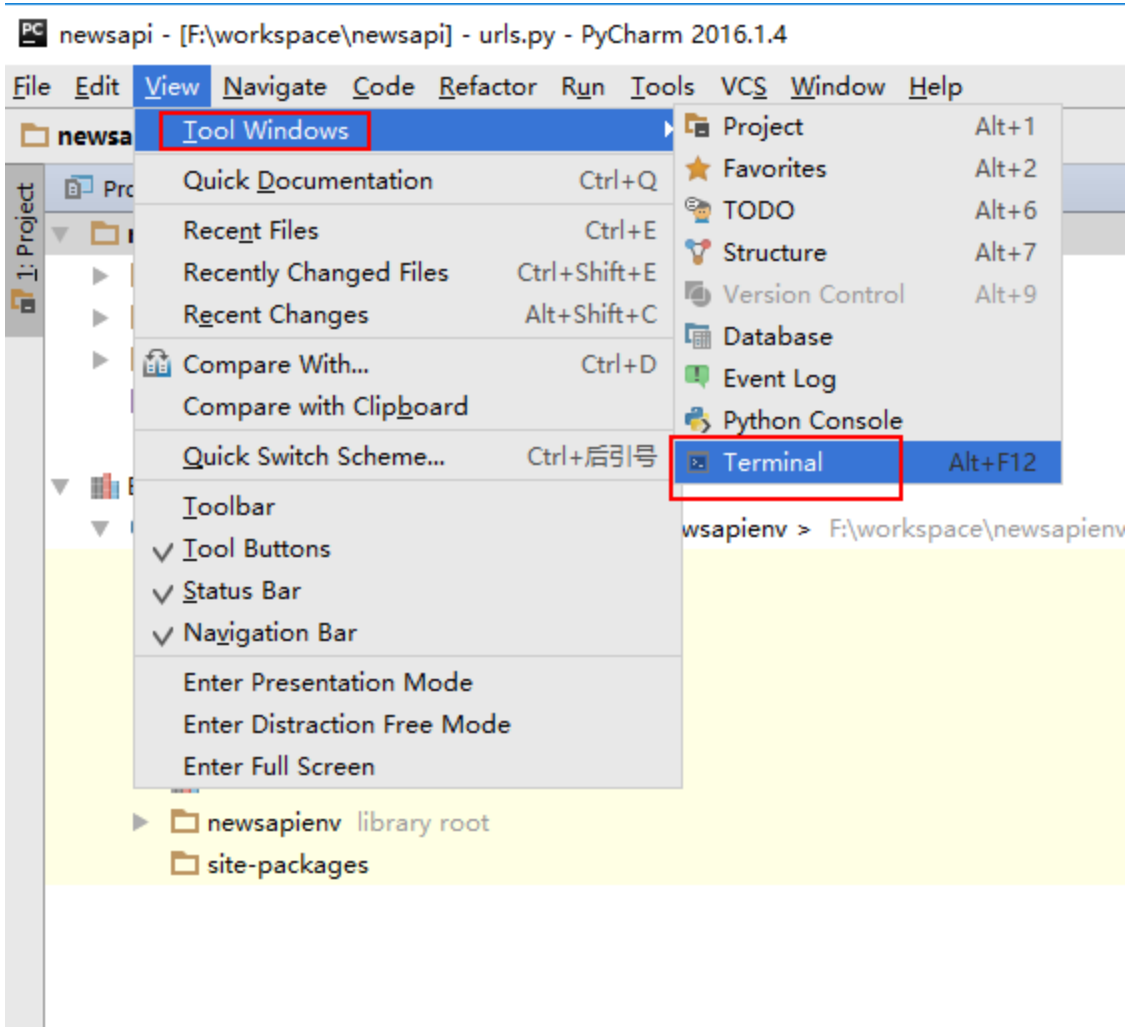
可以看到我们的虚拟环境是空的， 如下图所示：



我们工程的解释器是虚拟环境中的了， 如下图所示：

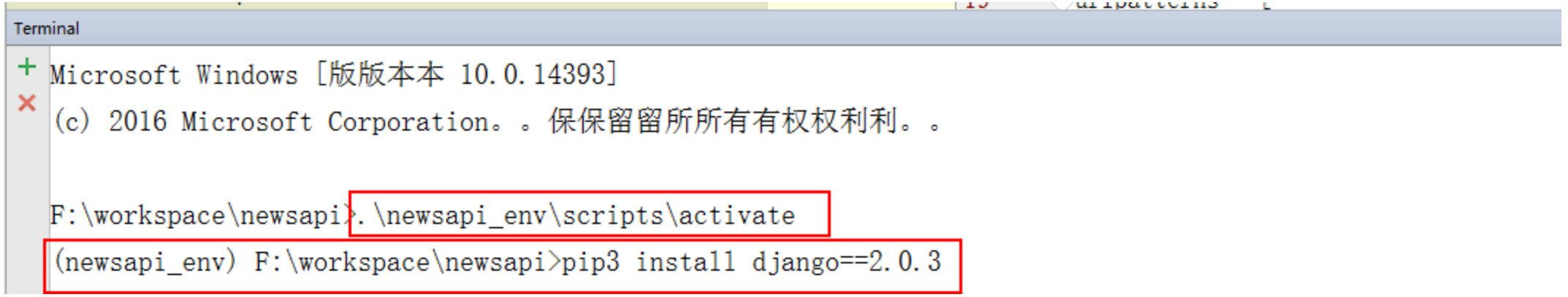


启动Termial终端安装相关依赖包



启动Termial终端安装相关依赖包

激活虚拟环境，安装依赖包



(newsapi_env) F:\workspace\newsapi>pip install -r requirement.txt

目 录

Django Rest Framework API设计指南

Django REST Framework（简称DRF），是一个用于构建Web API的强大且灵活的工具包。

REST是一种Web API设计标准，是目前比较成熟的一套互联网应用程序的API设计理论。

URL格式	HTTP请求方式	操作	功能
{prefix}/	GET	list	查询列表
{prefix}/	POST	create	提交数据
{prefix}/{pk}/	GET	retrieve	查询某条详情
{prefix}/{pk}/	PUT	update	更新某条数据所有字段
{prefix}/{pk}/	PATCH	partial_update	更新某条数据部分字段
{prefix}/{pk}/	DELETE	destroy	删除某条数据

Django Rest Framework API设计指南

如对Item对象设计API

list:

GET url: /item/ 子类列表数据

creat:

POST url: /item/ 创建子类详情，返回新生成的子类对像

retrieve:

GET url: /item/1/ 获取子类详情，返回子类对像

update:

PUT url: /item/1/ 修改子类详情，返回子类对像

delete:

DELETE url: /item/1/ 删除子类详情，返回空对像

Django Rest Framework API设计指南

- **查询过滤api，通过URL上传参的形式传递搜索条件**
  - http://127.0.0.1:8000/article/?limit=10：指定返回记录的数量
  - http://127.0.0.1:8000/article/?offset=10：指定返回记录的开始位置
  - http://127.0.0.1:8000/article/?page=2&per_page=10：指定第几页，以及每页的记录数
  - http://127.0.0.1:8000/article/?sortby=publish_date&order=asc：指定返回结果按照哪个属性排序，以及排序顺序
  - http://127.0.0.1:8000/article/? title=1：指定筛选条件

Django Rest Framework介绍

- **状态码**

200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。

201 CREATED - [POST/PUT/PATCH]：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 - [DELETE]：用户删除数据成功。

400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。

401 Unauthorized - [*]：表示用户没有权限（令牌、用户名、密码错误）。

403 Forbidden - [*] 表示用户得到授权（与401错误相对），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，该操作是幂等的。

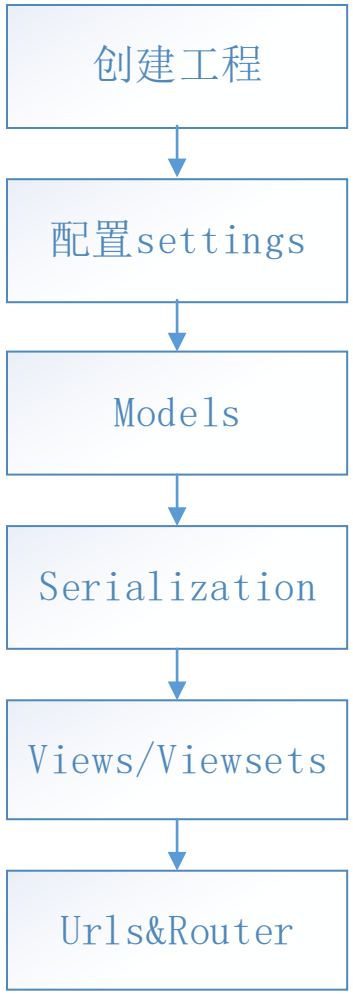
406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。

410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。

422 Unprocesable entity - [POST/PUT/PATCH] 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，用户将无法判断发出的请求是否成功。

DRF开发步骤



1. **创建工程**
2. **配置settings**
3. **创建模型和同步数据库**
4. **创建序列化器**
   - 依靠 Serialiers 将数据库取出的数据 Parse 为 API 的数据（可用于返回给客户端，也可用于浏览器显示）
5. **编写视图**
   - ViewSet 是一个 views 的集合，根据客户端的请求（GET、POST等），返回 Serialiers 处理的数据
   - 加入queryset属性
   - 加入serializer_class属性
   - 设置过滤器模板：filter_backends
   - 设置分页模板：pagination_class
   - 认证Authentication、权限 Premissions、分流throttle_classes
6. **配置urls& Routers**

2、配置settings

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'article',
    'rest_framework',
    'crispy_forms',
    'django_filters',
    'rest_framework.authtoken'
]
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'newstwo',
        'HOST': 'localhost',
        'PORT': '3306',
        'USER': 'root',
        'PASSWORD': 'root',
        'CHARSET':'utf8',##设置字符集，不然会出现中文乱码
    }
}
```

3、创建模型和同步数据库

- 指定mysql数据库驱动

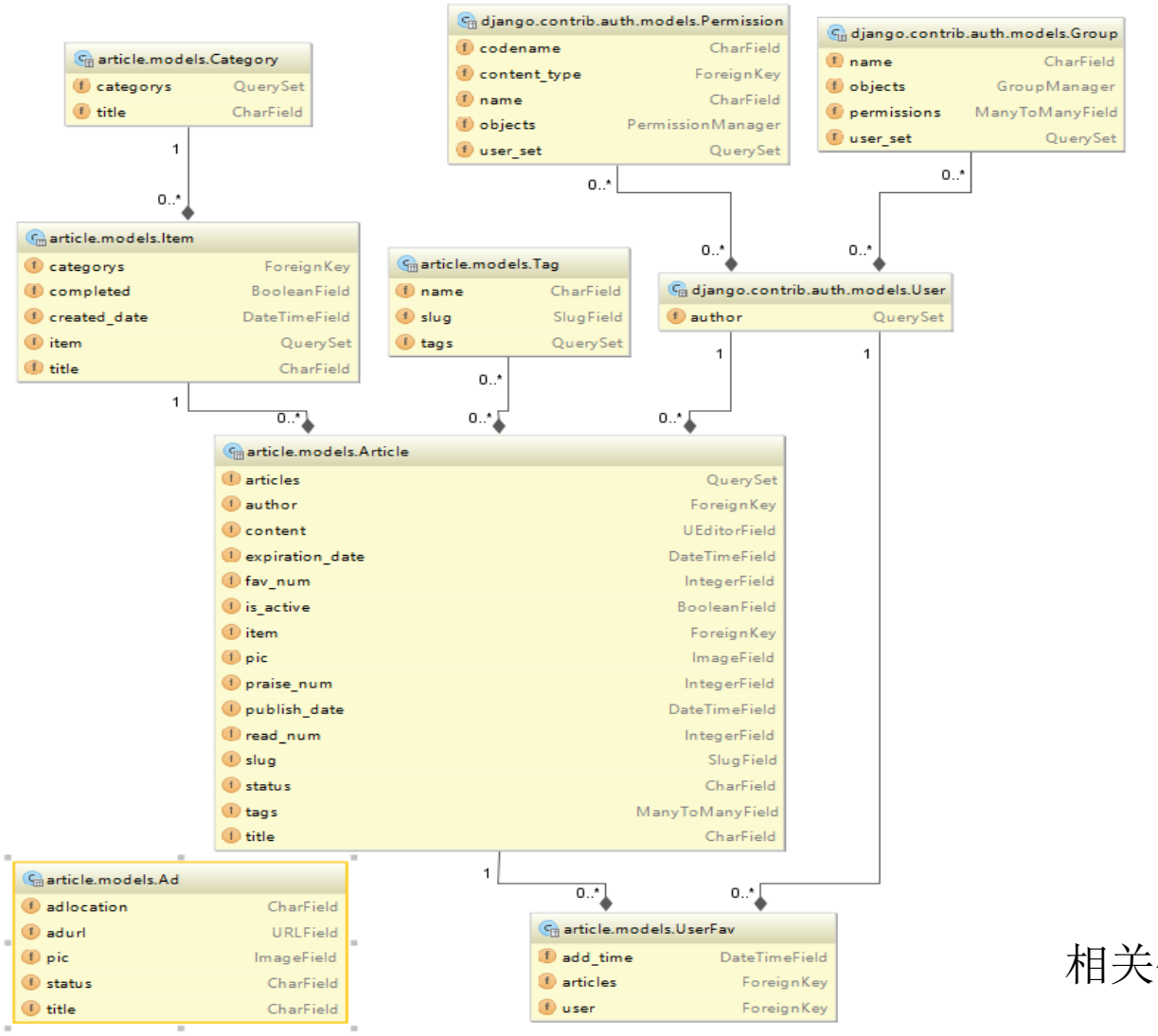
  与settings同目录下的__init__.py 文件中加下面代码：

  import pymysql

  pymysql.install_as_MySQLdb()

- 设计模型

新闻系统相关模型



相关代码参见源码

3、创建模型和同步数据库

- 同步数据库

  Python3 manage.py makemigrations

  Python manage.py migrate

  Python manage.py createsupuser

- 配置admin

- 初始化数据

4、配置ADMIN管理后台

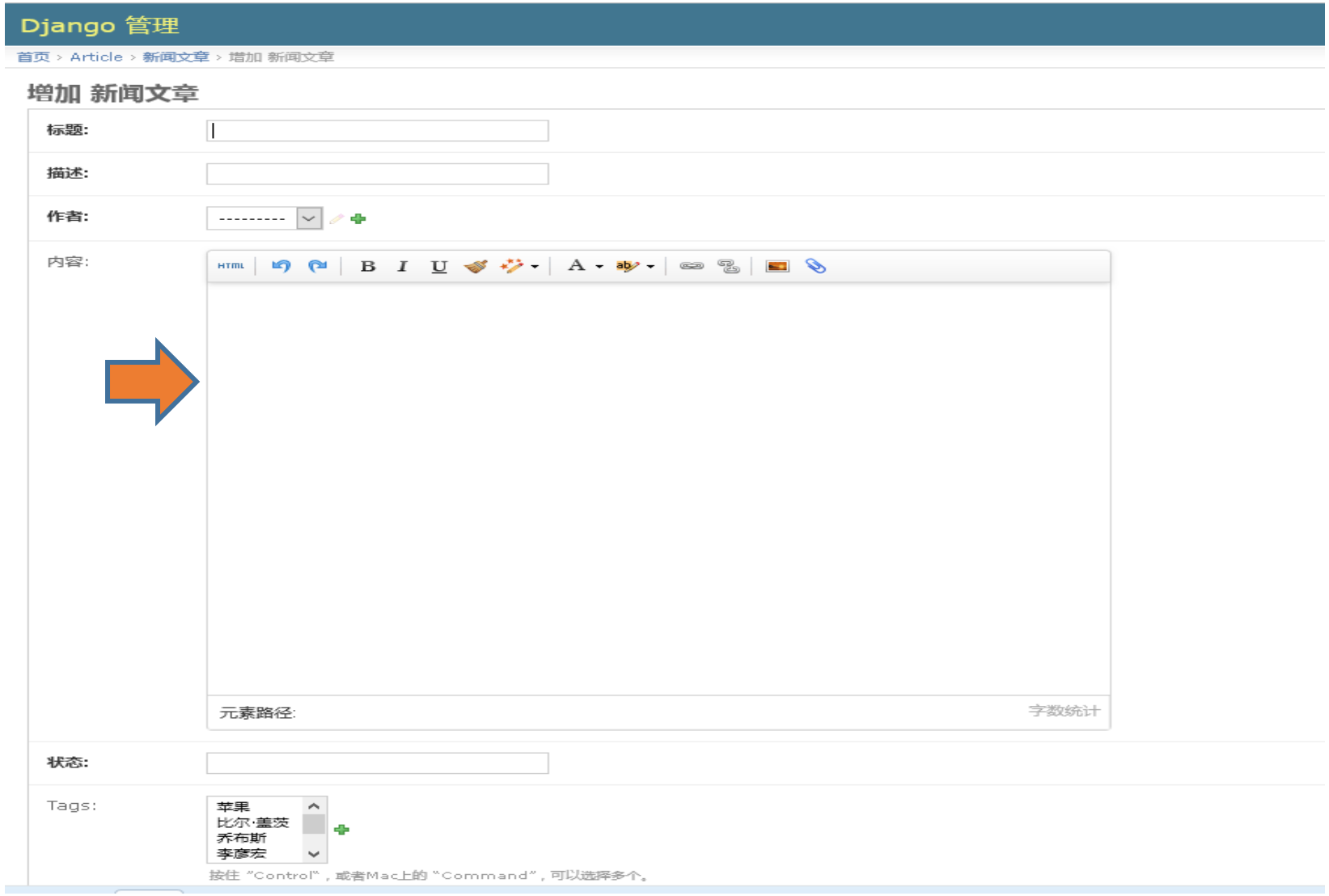
主要技术点：

admin.py 中配置的指定

- 列表展示字段： list_display
- 搜索字段: search_fields
- 筛选字段: list_filter
- 配置第三方js,css
- 注册admin

```python
# -*- coding: utf-8 -*-
from django.contrib import admin
from models import Tag, Article,Category,Item

# Register your models here.
class ArticleAdmin(admin.ModelAdmin):
    #列表显示字段
    list_display = ('title', 'item', 'status', 'author', 'publish_date',
                    'expiration_date', 'is_active')

    list_filter = ('author', 'status', 'is_active', 'publish_date',
                   'expiration_date')
    #每页记录数
    list_per_page = 25
    #查询字段
    search_fields = ('title', 'tags', 'slug', 'content')

    class Media:
        js = ('/static/ueditor/ueditor.config.js', '/static/ueditor/ueditor.all.min.js',)

class TagAdmin(admin.ModelAdmin):
    list_display = ('name', 'article_count')

    def article_count(self, obj):
        return obj.article_set.count()
```

➢运行后台管理界面



5、创建序列化器

- serializers是一个数据转换器
  将复杂的数据结构与json或者xml这个格式互相转换

  实例对象 <=> 原生数据类型 <=> JSON（XML）

  User <=> b'{"name": 1}' <=> {"name": 1}

- serializers有以下几个作用：
  - 将queryset与model实例等进行序列化，转化成json格式，返回给用户(api接口)。
  - 将post与patch/put的上来的数据进行验证。
  - 对post与patch/put数据进行处理。

- 实现序列化二个类：Serializer与ModelSerializer

Serializer与ModelSerializer

- 实现序列化二个类：Serializer与ModelSerializer

主要内容	小标题	Serializer	ModelSerializer
field	常用的field	CharField、BooleanField、IntegerField、DateTimeField	
	Core arguments	read_only、write_only、required、allow_null / allow_blank、label、help_text、style	
	HiddenField	HiddenField的值不需要用户自己post数据过来，也不会显式返回给用户。配合CurrentUserDefault()可以实现获取到请求的用户	
save instance		serializer.save()，它会调用了serializer的create或update方法	
		当有post请求，需要重写serializer.create方法	ModelSerializer已经封装了这两个方法，如果额外自定义，也可以进行重载
		当有patch请求，需要重写serializer.update方法	
Validation自定义验证逻辑	单独的validate	对某个字段进行自定义验证逻辑，重载validate_ + 字段名	
	联合validate	重写validate()方法，对多个字段进行验证或者对read_only的字段进行操作	
	Validators	1.单独作用于某个field，作用于重载validate_ + 字段名相似。 2. UniqueValidator: 指定某一个对象是唯一的，直接作用于某个field。 3. UniqueTogetherValidator: 联合唯一，需要在Meta属性中设置	
ModelSerializer专属	validate		重载validate可以实现删除用户提交的字段，但该字段不存在指定model，避免save()出错
	SerializerMethodField		SerializerMethodField实现将model不存在字段或者获取不到的数据序列化返回给用户
外键的serializers	正向	PrimaryKeyRelatedField，不关心外键具体内容	通过field已经映射，不关心外键具体内容
		嵌套serializer，获取外键具体内容	
	反向	Model设置related_name，通过related_name嵌套serializer	

序列化(输出)与反序列化(输入）

```
# user 为实例对象
serializer = UserSerializer(user)
serializer.data
# {"name": 1}


# data 为{"name": 1}
serializer = UserSerializer(data=data)
serializer.is_valid()
# True
serializer.save()
```

使用Serializer

序列化一个对象

```
serializer = UserSerializer(user)
```

序列化一个集合

```
serializer = UserSerializer(queryset, many=True)
```

当然，在实际的使用中，有时我们所要的序列化结果是要随请求中带有的参数来决定的，这个时候，你就需要将request传给序列化类，然后再做具体处理

```
serializer = UserSerializer(queryset, many=True, context={'request': request})
```

使用ModelSerializer

- ModelSerializer继承了Serializer的相关功能，是对model实现序列化的封装

```
class ItemSerializer(serializers.ModelSerializer):
    categorys = CategorySerializer()

    class Meta:
        model = Item
        fields = "__all__"

class TagSerializer(serializers.Serializer):
    #id = serializers.Field()
    name = serializers.CharField(required=True, max_length=100)
    slug = serializers.CharField(required=True, max_length=100)
    class Meta:
        model = Tag
        fields = ('id','name', 'slug')
```

使用ModelSerializer

- ModelSerializer已经重载了create与update方法

  它能够满足将post或patch上来的数据进行进行直接地创建与更新，除非有额外需求，那么就可以重载create与update方法。

- ModelSerializer在Meta中设置fields字段，系统会自动进行映射，省去每个字段再写一个field。

详细内容介绍：http://www.careeru.cn/blog/article/?id=85

序列化时嵌套显示外键关联字段

- 正向嵌套

```
#新闻分类
class CategorySerializer(serializers.ModelSerializer):

    class Meta:
        model = Category
        fields = “__all__”

#分类下的栏目
class ItemSerializer(serializers.ModelSerializer):
    categorys = CategorySerializer()

    class Meta:
        model = Item
        fields = ”__all__”
```

在序列化对象里添加自定义内容

```
from django.contrib.auth.models import User
from django.utils.timezone import now
from rest_framework import serializers


class UserSerializer(serializers.ModelSerializer):
    days_since_joined = serializers.SerializerMethodField()

    class Meta:
        model = User
    def get_days_since_joined(self, obj):
        return (now() - obj.date_joined).days
```

DRF基础：请求（Request）

1、请求

.data: 获取请求的主体，相当于request.POST和request.FILES

.query_params: request.GET的重命名

.parsers: APIView类或@api_view装饰将确保这个属性将自动设置为一个解析器实例列表

2、内容协商

.accepted_render: 接受渲染一个对象内容协商

.accepted_media_type: 接受的媒体类型

3、身份验证

.user: 通常返回django.contrib.auth.models.user的对象。虽然行为取决于所使用的身份验证策略。

.auth: 被认证的后才会被使用，返回任何额外身份的环境。

.authenticators: APIView类或@api_view装饰将确保这个属性将自动设置为一个认证实例列表

4、浏览器增强方法

django rest framework提供一些附加的表单提交方法:PUT,PATCH,DELETE

.method: 返回一个大写的HTTP请求方法字符串，包含了PUT,PATCH,DELETE等。

.content_type: 返回一个字符串对象代表HTTP请求的媒体类型的身体,或如果没有媒体类型提供一个空字符串。

同样可以使用：request.META.get('HTTP_CONTENT_TYPE')

.stream: 返回一个请求主体类容的流。

5、 支持其他标准的HttpResquest属性

.META/.session/等等

DRF基础： Response（响应）

你需要使用一个APIView类或@api_view函数返回response对象的views

1、创建Response

from rest_framework.response import Response

例: Response(data, status=None, template_name=None, headers=None, content_type=None)

data: 响应serialized(序列化)后的数据

status: 定义响应状态码，默认200

template_name: 如果HTMLRender被选择，返回一个模板

headers: 是关于HTTP请求头的一个字典

content_type: 响应内容类型，一般会根据内容自动协商。

2、其他标准的HttpResponse属性

response = Response()

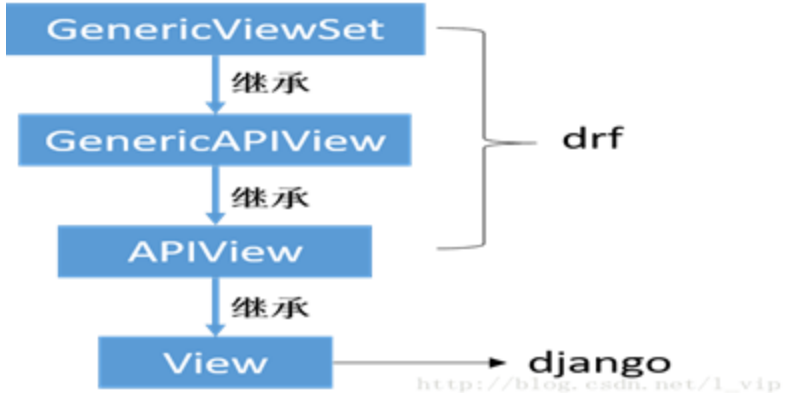
response['Cache-Control'] = 'no-cache'

.render(): 响应序列化的数据。

6、编写视图

drf为我们提供强大的通用view的功能

主要的几种view以及他们之间的关系



这其中，还涉及了mixins，主要也分为5类：

mixins	作用	对应http的请求方法
mixins.ListModelMixin	定义list方法，返回一个queryset的列表	Get
mixins.CreateModelMixin	定义create方法，创建一个实例	Post
mixins.RetrieveModelMixin	定义retrieve方法，返回一个具体的实例	Get
mixins.UpdateModelMixin	定义update方法，对某个实例进行更新	Put/Patch
mixins.DestroyModelMixin	定义delete方法，删除某个实例	delete

强烈建议在做drf的时候，使用ViewSet与mixins方法结合进行开发

- 路由系统是用于接收合适的请求，然后返回相应的响应
- REST框架允许将一系列相关的业务逻辑函数写在一起，称为ViewSet.在其他框架中，相似的实现也被称为’Resources’或者’Controllers’.
- 一个ViewSet类也是一个基础的View类，不提供任何请求处理函数，如get或post,只提供类似list和created这样的方法
- 一个ViewSet类，通过.as_view() method绑定到相应的url上
- 一般不需要在路由管理器中另外注册视图函数，只需要注册这个viewset类就够了，然后就会自动帮你处理

ViewSet

- **ViewSet**

  ```
  class ViewSet(ViewSetMixin, views.APIView):
  ```

  继承自APIView，利用permission_classes, authentication_classes等来控制API策略 。一般都需要重写此类中的方法

- **GenericViewSet**

  class GenericViewSet(ViewSetMixin, generics.GenericAPIView):

  GenericViewSet继承自GenericAPIView，提供了一系列如get_object, get_queryset方法，如果使用此类，需要重写此类或者与mixin类进行组合

- **ModelViewSet**

  class ModelViewSet(mixins.CreateModelMixin,
  mixins.RetrieveModelMixin,
  mixins.UpdateModelMixin,
  mixins.DestroyModelMixin,
  mixins.ListModelMixin,
  GenericViewSet):

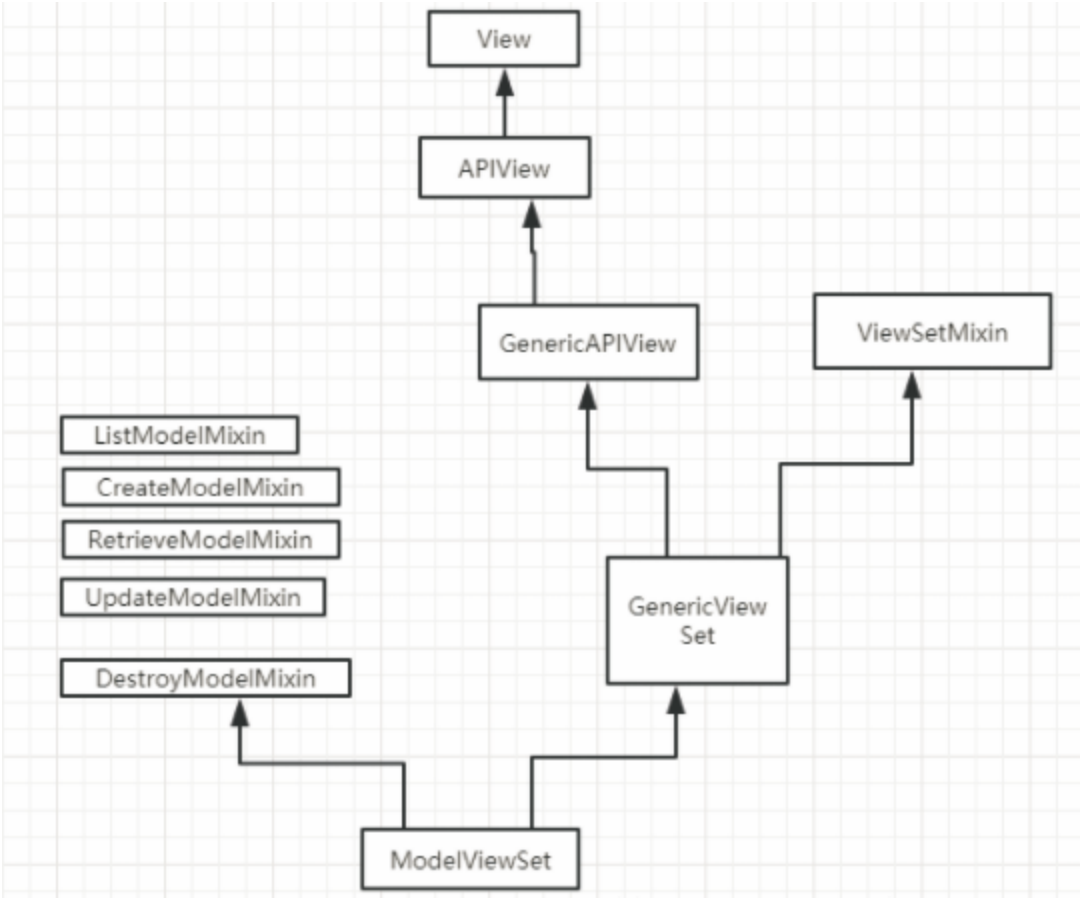
  也是继承自GenericAPIView，通过mixin类，实现了一些具体的操作，.list(), .retrieve(), .create(), .update(), .partial_update(), 和 .destroy()。

- **ReadOnlyModelViewSet**

  ```
  class ReadOnlyModelViewSet(mixins.RetrieveModelMixin,
                             mixins.ListModelMixin,
                             GenericViewSet):
  ```

  也是继承自GenericAPIView，仅提供只读操作，.list()和.retrieve()

ModelViewSet关系图



集成自动生成文档工具--swagger

```
pip install coreapi
pip install django-rest-swagger
```

```
# settings.py 中
INSTALLED_APPS = [
    ...
'rest_framework_swagger', ... ]
```

```
#urls.py
from rest_framework_swagger.views import get_swagger_view

schema_view = get_swagger_view(title='新闻系统 API')

urlpatterns = [
    re_path('^$', schema_view),
       ...
     ]
```

三个外键关联字段显示

```
class CategoryStringSerializer(serializers.ModelSerializer):
    # 用unicode方法表示只读关系 items 为外键关系中的related_name
    items = serializers.StringRelatedField(many=True)
    class Meta:
        model = Category
        #fields = ('id', 'title','items')
        fields = "__all__ "

class CategoryPrimaryKeySerializer(serializers.ModelSerializer):
    # 用主键表示关系
    items = serializers.PrimaryKeyRelatedField(many=True, read_only=True)
    class Meta:
        model = Category
        fields = "__all__"

class CategorySlugSerializer(serializers.ModelSerializer):
    # 选取关系对象中任意一个字段（唯一标识）表示关系
    items = serializers.SlugRelatedField(
        many=True,
        read_only=True,
        slug_field='title'
    )
    class Meta:
        model = Category
        fields = "__all__"
```

ViewSet使用

```
# APIView 类—类级别
class TagViewset(mixins.ListModelMixin,
mixins.RetrieveModelMixin,
viewsets.GenericViewSet):
    """
    list:
        标签列表数据
    retrieve:
        标签广告详情
    """
    queryset = Tag.objects.all()
    serializer_class = TagSerializer
    filter_backends =
(DjangoFilterBackend,)
    #filter_fields = ('name',)
    lookup_field = "id"
```

```
class ArticleListViewSet(mixins.ListModelMixin, mixins.RetrieveModelMixin,
viewsets.GenericViewSet):
    """
    商品列表页，分页， 搜索， 过滤， 排序
    可根据'author','status','publish_date','is_active','item','tags'查询
    """
    # throttle_classes = (UserRateThrottle,)
    # 查询对象集
    queryset = Article.objects.all()
    # 序列化的类名
    serializer_class = ArticleSerializer
    #分页
    pagination_class = ArticlePagination
    # authentication_classes = (TokenAuthentication,)
    #过滤、查询类
    filter_backends = (DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter)
    #DjangoFilterBackend对应filter_fields属性，做相等查询
    # 过滤字段类
    filter_class = ArticleFilter
    #SearchFilter对应search_fields，对应模糊查询
    search_fields = ('title','item__title','tags__name')
    #排序
    ordering_fields = ('id','publish_date')
    lookup_field = "id"
    def retrieve(self, request, *args, **kwargs):
        instance = self.get_object()
        instance.read_num += 1
        instance.save()
        serializer = self.get_serializer(instance)
        return Response(serializer.data)
```

ViewSet使用

- 加入queryset属性，可以直接设置这个属性，不必再将实例化的courses，再次传给seriliazer,系统会自动检测到。除此之外，可以重载get_queryset()，这样就不必设置’queryset=*’，这样就变得更加灵活，可以进行完全的自定义。
- 加入serializer_class属性与实现get_serializer_class()方法。两者的存在一个即可，通过这个，在返回时，不必去指定某个serilizer
- 序列化的类名
- 设置过滤器模板：filter_backends

```
DjangoFilterBackend对应filter_fields属性，做相等查询
#自定义过滤字段类
filter_class = ArticleFilterSearchFilter对应search_fields，对应模糊查询
```

- 设置分页模板：pagination_class

ViewSet使用

- 排序ordering_fields
- authentication_classes：用户登录认证方式，session或者token等
- permission_classes：权限设置，是否需要登录等
- throttle_classes：限速设置，对用户进行一定的访问次数限制等等。
- 加入 lookup_field=”pk”，以及实现了get_object方法：这个用得场景不多，但十分重要。

  它们两者的关系同1，要么设置属性，要么重载方法。它们的功能在于获取某一个实例时，指定传进来的后缀是什么。

  举个例子，获取具体的某个课程，假设传进来的ulr为：http://127.0.0.1:8000/course/1/，系统会默认这个1指的是course的id。那么，

  现在面临一个问题，假设我定义了一个用户收藏的model，我想要知道我id为1的课程是否收藏了，我传进来的url为：

  http://127.0.0.1:8000/userfav/1/,系统会默认获取userfav的id=1的实例，这个逻辑明显是错的，我们需要获取course的id=1的收藏记录，

  所以我们就需要用到这个属性或者重载这个方法lookup_field=”course_id”.

DRF Filtering（过滤器）

```
pip install django-filter

# settings.py 中
REST_FRAMEWORK = { 'DEFAULT_FILTER_BACKENDS':
('django_filters.rest_framework.DjangoFilterBackend',) }

#views.py
#过滤、查询类
filter_backends = (DjangoFilterBackend, filters.SearchFilter, filters.OrderingFilter)
#DjangoFilterBackend对应filter_fields属性，做相等查询
#filter_fields = ()
# 自定义过滤字段类
filter_class = ArticleFilter
#SearchFilter对应search_fields，对应模糊查询
search_fields = ('title','item__title', 'tags__name')
#排序
ordering_fields = ('id', 'publish_date')
```

```
class ArticleFilter(django_filters.rest_framework.FilterSet):
    """
    文章的过滤类
    """
    author = django_filters.CharFilter(name='author', help_text="作者")
    status = django_filters.ChoiceFilter(name='status', help_text="状态")
    publish_date = django_filters.DateTimeFilter(name='publish_date', help_text="发布时间")
    item = django_filters.CharFilter(name='item', help_text="分类")
    tags = django_filters.CharFilter(name='tags', help_text="标签")
    # 其中method指向自己定义的过滤函数，label用于标识在测试API界面中的过滤界面字段，item_categorys控制查询字段
    item_categorys  = django_filters.NumberFilter(method='item_categorys_filter', help_text="大类")
#   top_category = django_filters.NumberFilter(method='item_categorys_filter')

    PEP 8: indentation is not a multiple of four (comment)
    def item_categorys_filter(self, queryset, name, value):
        return queryset.filter(item__categorys=value)


    class Meta:
        model = Article
        fields = ['author', 'status', 'publish_date', 'is_active', 'item','item__categorys', 'tags']
```

搜索行为可以为search_fields字段设置不同的字符进行限制

- ‘^’ Starts-with search(以…开始查询).
- ‘=’ Exact matches（准确查询）.
- ‘@’ Full-text search.（全文检索） (Currently only supported Django’s MySQL backend.)
- ‘$’ Regex search（正则查询）.

```
search_fields = ('=username', '=email')
```

ordering_fields指定哪些字段用来排序

UserViewset重写mixins中CreateModelMixin 中的Create方法

```python
class CreateModelMixin(object):
    """
    Create a model instance.
    """
    def create(self, request, *args, **kwargs):
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        self.perform_create(serializer)
        headers = self.get_success_headers(serializer.data)
        return Response(serializer.data, status=status.HTTP_201_CREATED, headers=headers)

    def perform_create(self, serializer):
        serializer.save()
```

```python
#重写create方法,给密码加密，并查询和创建token
def create(self, request, *args, **kwargs):
    serializer = self.get_serializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    passwd = request.data['password']
    user = self.perform_create(serializer)
    #给密码加密
    user.set_password(passwd)
    user.save()
    re_dict = serializer.data
    #查询和创建token
    token = Token.objects.get_or_create(user=user)

    serializer = UserRegSerializer({'id': user.id, 'username': user.username})
    serializer.data["status"] = HTTP_201_CREATED
    #headers = self.get_success_headers(serializer.data)
    return Response(serializer.data)
```

```python
def perform_create(self, serializer):
    return serializer.save()
```

Django rest framework views知识

Vies详细内容介绍：

Django rest framework APIView介紹
http://www.careeru.cn/blog/article/?id=93

Django-restframework 视图函数基类ViewSets和路由类Router
http://www.careeru.cn/blog/article/?id=92

django rest framework views知识
http://www.careeru.cn/blog/article/?id=86

7、配置urls& Routers

通过一个 Router 类会自动处理 视图Viewset和 url 连接资源。

我们所需要做的就是通过路由注册一个适当的视图集。

```
# 建立一个路由器对象
router = DefaultRouter()
# 将我们的路由注册到url里
router.register(r'category', views.CategoryViewset, base_name="category")
```

8、分页Pagination

- 1、设置分页

  *可以在settings中设置分页样式

  REST_FRAMEWORK = { ‘DEFAULT_PAGINATION_CLASS’: ‘rest_framework.pagination.LimitOffsetPagination’ }

  *修改分页样式

```
class ArticlePagination(PageNumberPagination):
    page_size = 5
    page_size_query_param = 'page_size'
    page_query_param = "page"
    max_page_size = 20
```

  *应用到视图中ArticleListViewSet

```
    ...
    #分页
    pagination_class = ArticlePagination
```

- 2 API引用

  *PageNumberPagination

  设置全局的分页大小：

```
REST_FRAMEWORK = { ‘DEFAULT_PAGINATION_CLASS’: ‘rest_framework.pagination.PageNumberPagination’, ‘PAGE_SIZE’: 100 }
```

9、认证与权限Authentication & Permissions

- 认证
  - 类：authenticate/authenticate_header
  - 返回值：None,（user,auth）,异常
  - 配置：
    - 视图：

```
class UserViewset(...):
    authentication_classes  = [MyAuthentication,]
```

    - 全局：

```
REST_FRAMEWORK = {
    'UNAUTHENTICATED_USER':  None,
    'UNAUTHENTICATED_TOKEN':  None,
    "DEFAULT_AUTHENTICATION_CLASSES":  [
     "article.auth.MyTokenAuthentication",  #定义自己的认证
    ],
}
```

- 权限
  - 类：has_permission/has_object_permission
  - 返回值： True、False、exceptions.PermissionDenied(detail="错误信息")
  - 配置：
    - 视图：

```
class UserViewset(...):
    permission_classes = [MyPermission,]
```

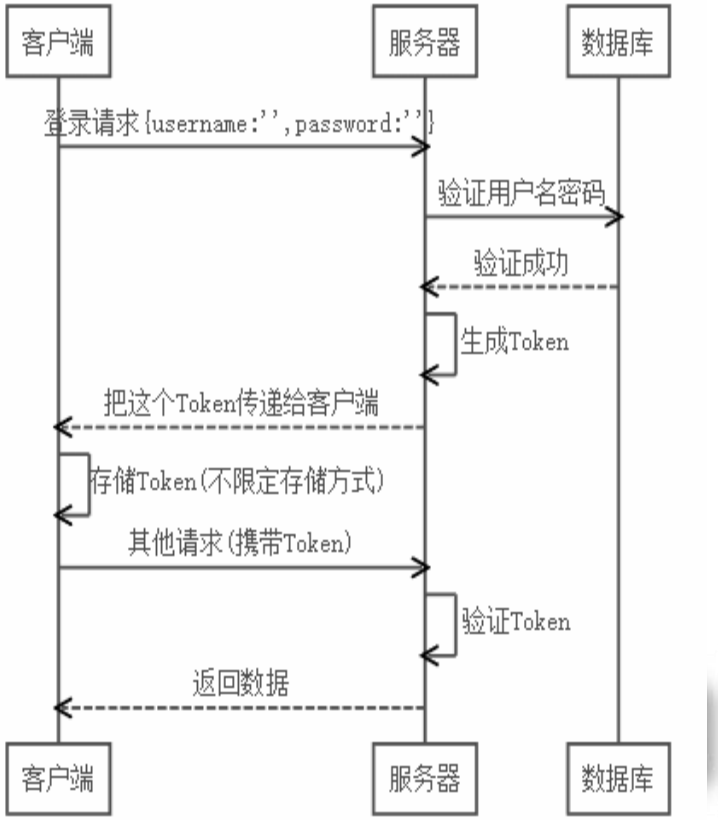
    - 全局：

```
REST_FRAMEWORK = {
      "DEFAULT_PERMISSION_CLASSES":  [
    # "xxx.utils.MyAuthentication",
      ],
}
```

基于Token的身份验证

整个基于Token的验证流程如下:

1. 客户端使用用户名跟密码请求登录
2. 服务器收到请求,去验证用户名和密码
3. 验证成功后,服务端会签发一个Token,再把这个Token发送到客户端
4. 客户端收到的Token以后可以把它存储起来,比如放在Cookie里
5. 客户端每次向服务器发送其他请求的时候都要带着服务器签发的Token
6. 服务器收到请求,去验证客户端请求里面带着的Token,如果验证成功,就像客户端返回请求的数据



10、节流Throttling

- **设置节流方案**
  *可以在settings文件中配置DEFAULT_THROTTLE_CLASSES 和 DEFAULT_THROTTLE_RATES

```
'DEFAULT_THROTTLE_CLASSES':
  ('rest_framework.throttling.AnonRateThrottle',
'rest_framework.throttling.UserRateThrottle'),
'DEFAULT_THROTTLE_RATES':{
'anon':'100/day',
'user':'1000/day'
}
```

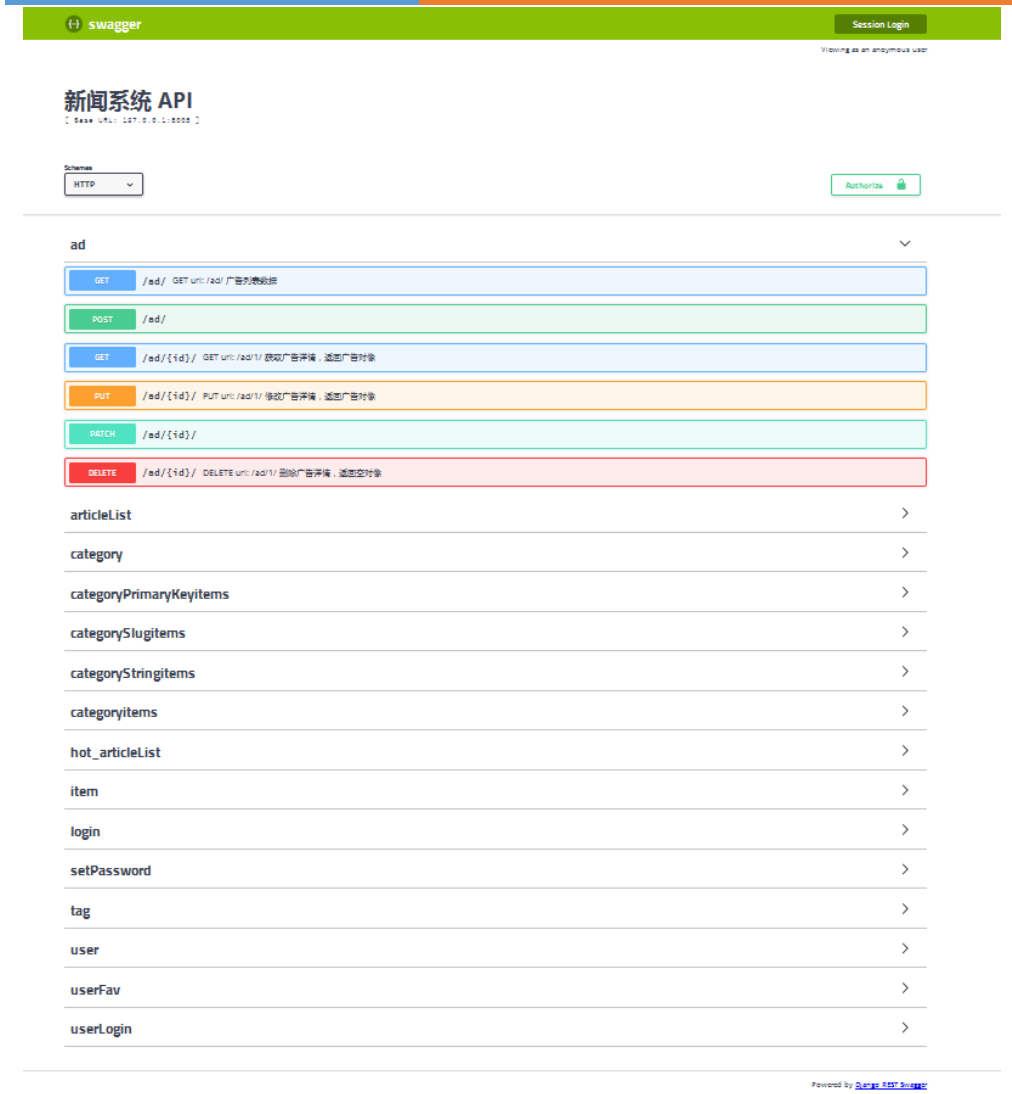
*可以在视图中设置：

```
（APIView）
throttle_classes = (UserRateThrottle,)
（@api_view）
@throttle_classes([UserRateThrottle])
```

配置urls& Routers

```
schema_view = get_swagger_view(title='新闻系统 API')
# 建立一个路由器对象
router = DefaultRouter()
# 将我们的路由注册到url里
router.register(r'category', views.CategoryViewset, base_name="category")
router.register(r'categoryitems', views.CategoryitemsViewset, base_name="categoryitems")
router.register(r'categoryStringitems', views.CategoryStringitemsViewset, base_name="categoryStringitems")
router.register(r'categoryPrimaryKeyitems', views.CategoryPrimaryKeyitemsViewset, base_name="categoryPrimaryKeyitems")
router.register(r'categorySlugitems', views.CategorySlugitemsViewset, base_name="categorySlugitems")
router.register(r'item', views.ItemViewset, base_name="item")
router.register(r'tag', views.TagViewset, base_name="tag")
router.register(r'ad', views.AdViewset, base_name="ad")
router.register(r'articleList', views.ArticleListViewSet, base_name="articleList")
router.register(r'hot_articleList', views.Hot_articleListViewSet, base_name="hot_articleList")
router.register(r'user', views.UserViewset, base_name="user")
router.register(r'userFav', views.UserFavViewset, base_name="userFav")
router.register(r'userLogin', views.UserLoginViewset, base_name="userLogin")
router.register(r'setPassword', views.UserSetPasswordViewset, base_name="setPassword")
urlpatterns = [
    re_path('^$', schema_view),
    path('admin/', admin.site.urls),
    path(r'^api-auth/$', include('rest_framework.urls',
                                 namespace='rest_framework')),
    re_path(r'^login/$', views.UserLogin.as_view(), name="login"),
    #path(r'docs/', include_docs_urls(title="新闻系统API")),
    # re_path(r"^docs/$", schema_view),
    # path('ListUsers/', views0.ListUsers.as_view(),name="ListUsers"),
    # path('UserList/', views0.UserList.as_view(),name="UserList"),
    # path('UserListCreate/', views0.UserListCreate.as_view(),name="UserListCreate"),
    # path('user_list/', views0.user_list),
    # path('user_detail/(?P<pk>[0-9]+)/', views0.user_detail),
]
urlpatterns += router.urls
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

新闻系统web API列表



Django Rest Framework生命周期

Django生命周期:

前端发送请求-->Django的wsgi

-->中间件

-->路由系统

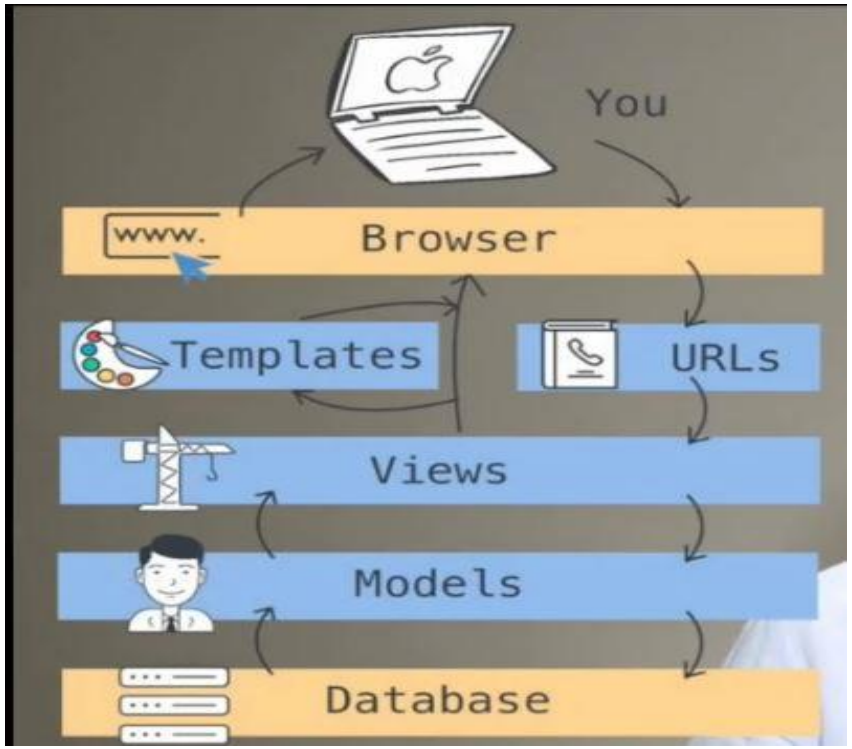
-->视图

-->ORM数据库操作

-->中间件

-->模板

-->返回数据给用户



Django Rest Framework生命周期

rest framework生命周期：

前端发送请求-->Django的wsgi

-->中间件

-->路由系统_执行CBV的as_view()，就是执行内部的dispath方法

-->在执行dispath之前，有版本分析和渲染器

-->在dispath内，对request封装

-->版本

-->认证

-->权限

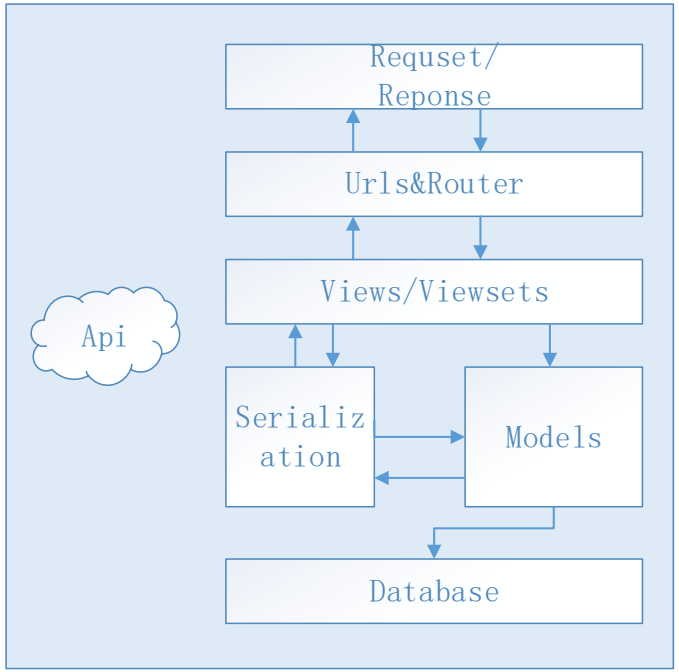
-->限流

-->视图

-->如果视图用到缓存(request.data or request.query_params)就用到了解析器

-->视图处理数据，用到了序列化(对数据进行序列化或验证)

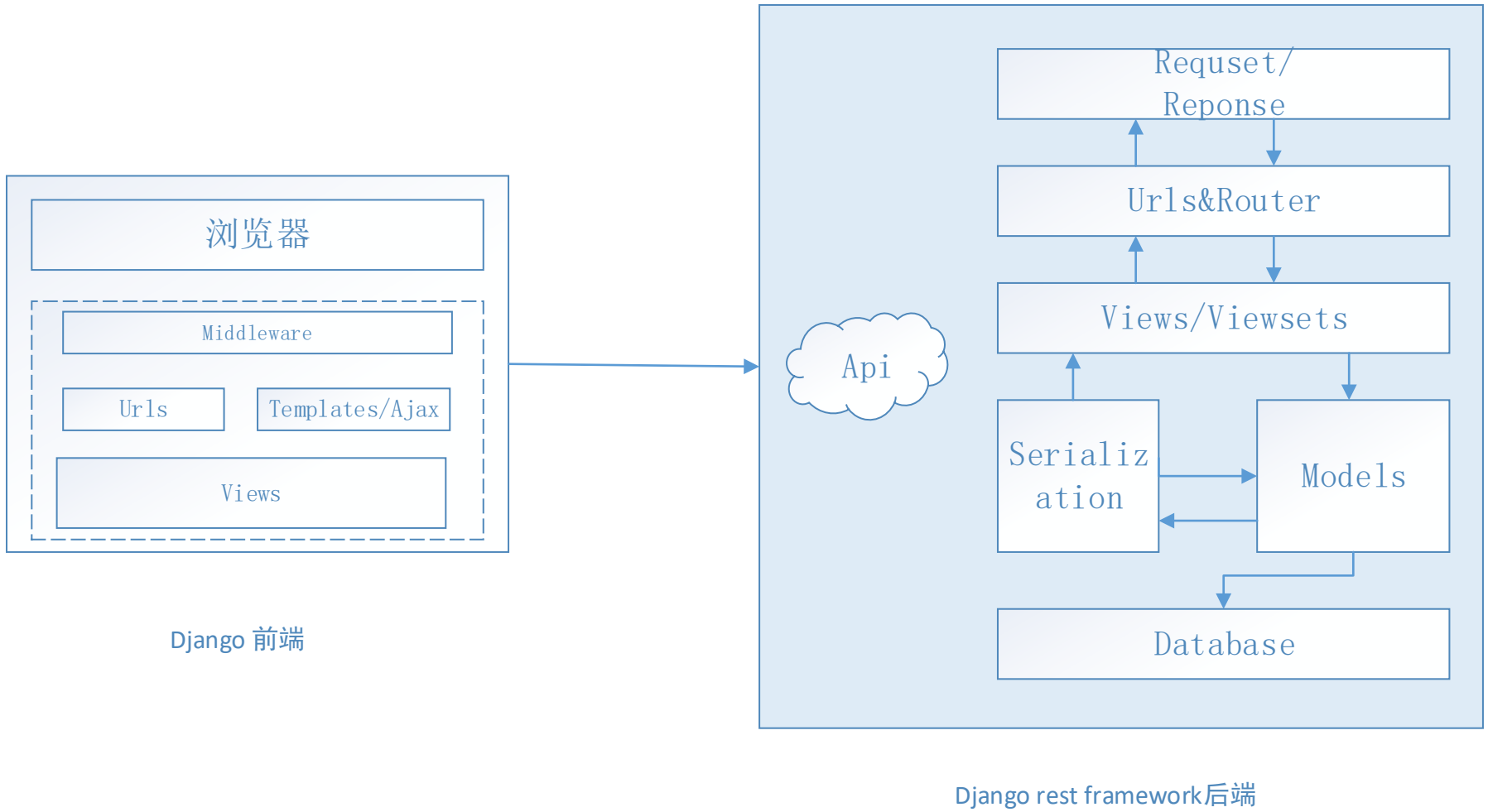
-->视图返回数据（可以用到分页）



Django rest framework后端

目 录

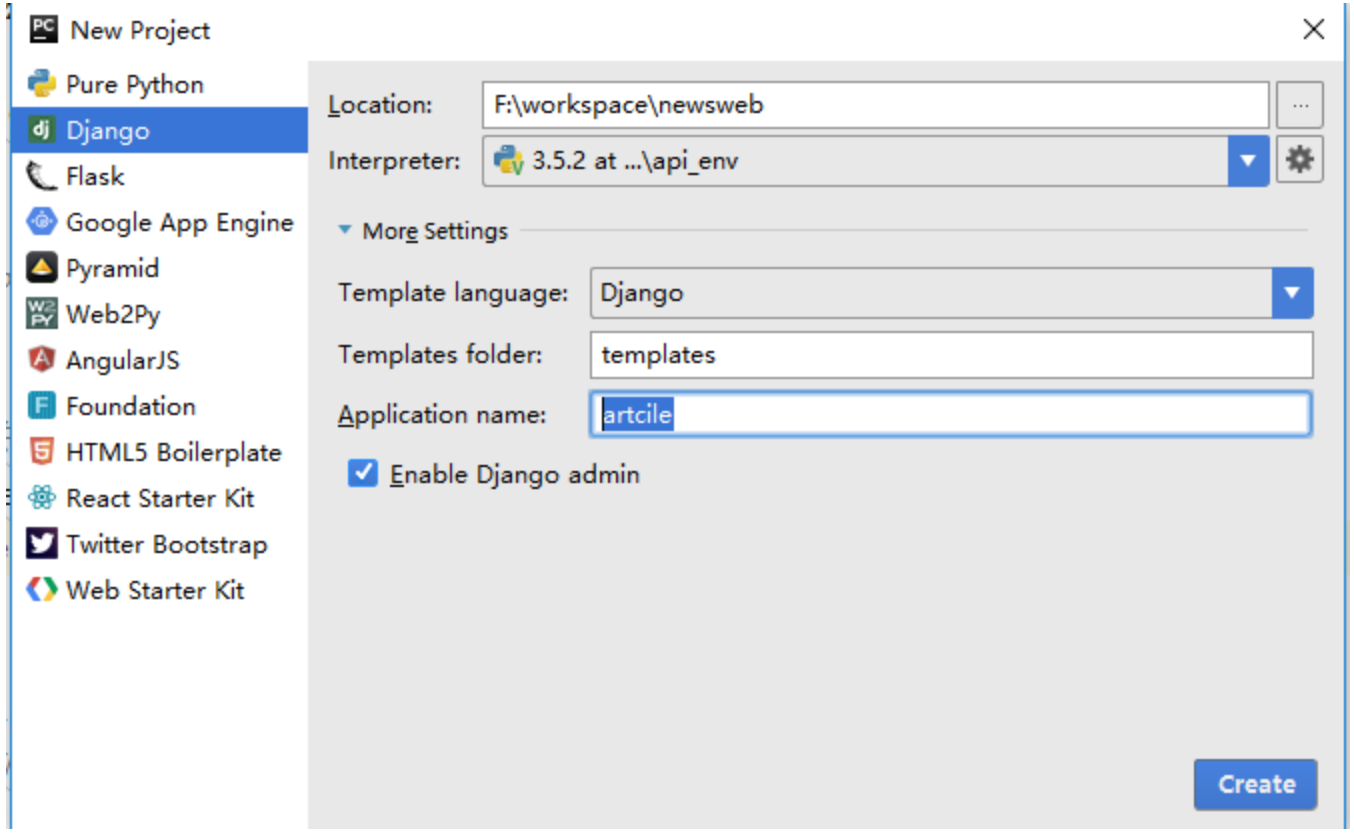
Django 实现新闻系统前后端分离框架



Django 前端

Django rest framework后端

创建新闻系统前端项目工程



Urllib模块的使用

1.基本方法-urlopen

urllib.request.urlopen(*url, data=None, [timeout,]*, cafile=None, capath=None, cadefault=False, context=None*)

- url: 需要打开的网址
- data：Post提交的数据
- timeout：设置网站的访问超时时间

直接用urllib.request模块的urlopen（ ）获取页面，page的数据格式为bytes类型，需要decode（ ）解码，转换成str类型。

```
from urllib import request
response = request.urlopen(r'http://python.org/')  # <http.client.HTTPResponse  object at 0x00000000048BC908>  HTTPResponse类型
page = response.read()
page = page.decode('utf-8')
```

urlopen返回对象提供方法：

- read()，readline() ,readlines()，fileno()，close() ：对HTTPResponse类型数据进行操作
- info()：返回HTTPMessage对象，表示远程服务器返回的头信息
- getcode()：返回Http状态码。如果是http请求，200请求成功完成;404网址未找到
- geturl()：返回请求的url

Urllib模块的使用

2.使用Request

urllib.request.Request(*url, data=None, headers={}, method=None*)

使用request（）来包装请求，再通过urlopen（）获取页面。

```
url = r'http://www.lagou.com/zhaopin/Python/?labelWords=label'
 headers = {
         'User-Agent': r'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) '
                       r'Chrome/45.0.2454.85 Safari/537.36 115Browser/6.0.3',
        'Referer': r'http://www.lagou.com/zhaopin/Python/?labelWords=label',
        'Connection': 'keep-alive' }
req = request.Request(url, headers=headers)
page = request.urlopen(req).read()
page = page.decode('utf-8')
```

用来包装头部的数据：

- User-Agent ：这个头部可以携带如下几条信息：浏览器名和版本号、操作系统名和版本号、默认语言
- Referer：可以用来防止盗链，有一些网站图片显示来源http://***.com，就是检查Referer来鉴定的
- Connection：表示连接状态，记录Session的状态。

读取web API的方法

```
def getdata(url,data=None):
      headers = {…}

      try:
          if data:
              #data = parse.urlencode(data).encode('utf8')
              #data 参数如果要传必须传 bytes （字节流）类型的，如果是一个字典，可以先用 urllib.parse.urlencode() 编码。
              #data = bytes(parse.urlencode(data), encoding="utf8")
              data = '?'+parse.urlencode(data)
              #使用request（）来包装请求，再通过urlopen（）获取页面。
              url = urljoin(url,data)
              req = request.Request(url=url, headers=headers, method='GET')
          else:
              req = request.Request(url=url, headers=headers)

          reponsedata = request.urlopen(req,timeout=10).read()
          reponsedata = reponsedata.decode('utf-8')
          returndata = json.loads(reponsedata)
      except Exception as e:
          print(e.code)
          returndata = {'result':e.reason,'code':e.code(),'msg':'请求api数据错误！','data':'{}','redirect_url':''}
      return returndata
```

Web站点的开发

- 实现Template的全局变量

主要技术点：

```python
# 实现全局变量
def globl_init(request):
    # 取新闻分类
    category_list = getdata(category_url)
    # 取热门新闻
    hot_articles = getdata(hotarticles_url)
    # 取广告数据
    ad_list = getdata(ad_url)
    user = request.user
    return locals()
```

```python
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'article.views.globl_init'
            ],
        },
    },
]
```

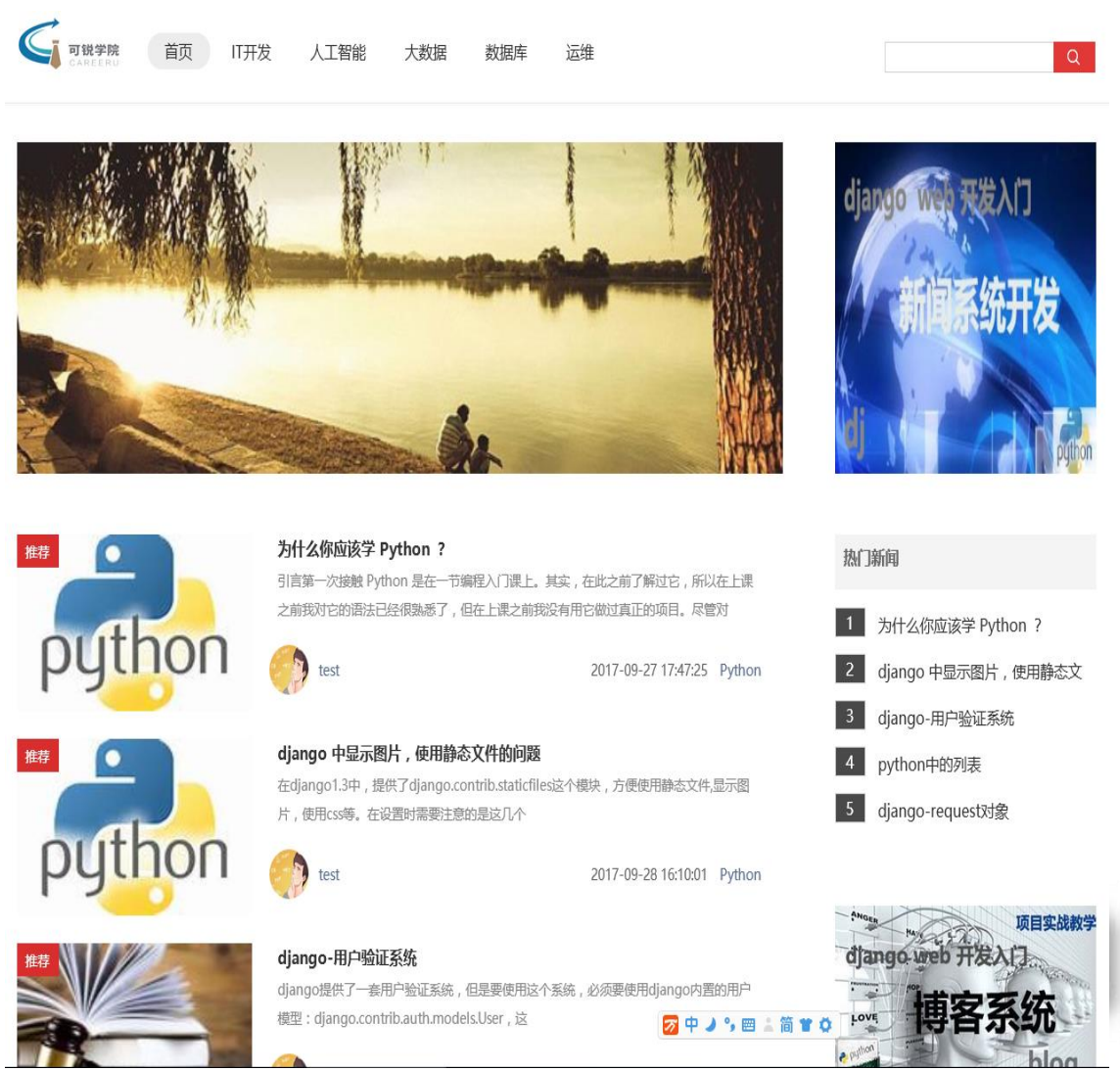
Web站点的开发

- 首页
  - 新闻类别导航栏的实现
  - 新闻列表的实现
  - 热门新闻的实现

主要技术点：

```
#首页
def index(request):

    data = {
        "ordering":'-id',
    }
    article_data = getdata(articles_url,data)
    article_list = article_data["results"]

    user = []
    return render(request, 'index.html', locals())
```

```
url:re_path(r'^$', index, name='index'),
```



Web站点的开发

- 首页
  - 广告的实现

Web站点的开发

- 首页
  - 广告的实现

主要技术点：

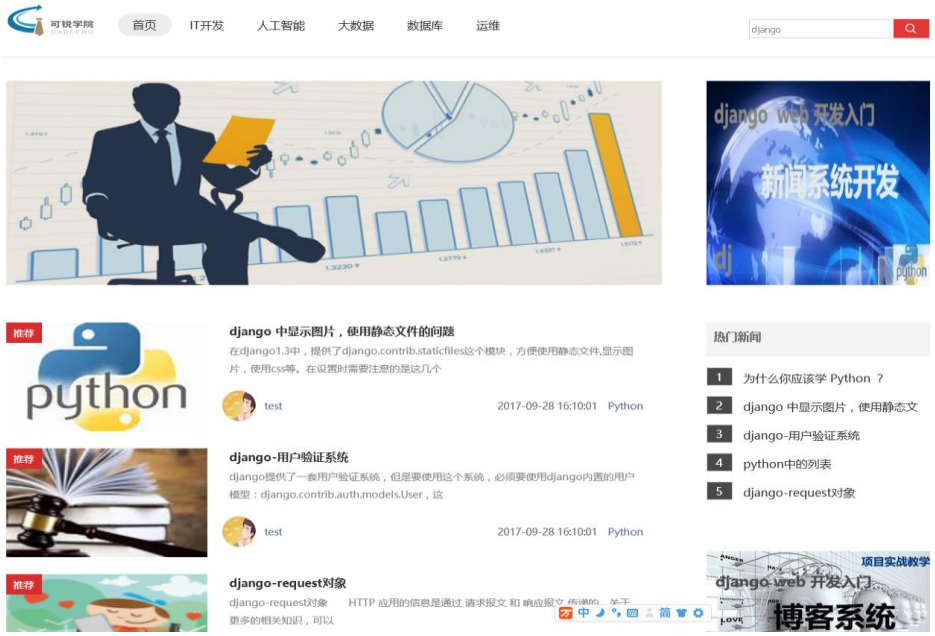
views:

```
ad_url = 'http://127.0.0.1:8000/ad/'

# 取广告数据
ad_list = getdata(ad_url)
```

url:`re_path(r'^$', index, name='index'),`

Web站点的开发

- 首页
  - 查询新闻



url:path ('search', search, name='search'),

template:

主要技术点：

```
#查询
def search(request):
    strquery = request.GET.get('query')
    page = int(request.GET.get('page', 1))
    data = {
        "search": strquery,
        "page": page,
        "ordering": '-id',
    }
    article_data = getdata(articles_url, data)
    article_list = article_data["results"]
    # 总记录数
    count = article_data["count"]
    # 下一页
    next = article_data["next"]
    nextpage = page + 1
    # 上一页
    previous = article_data["previous"]
    previouspage = page - 1
    # 总页数
    num_pages = int(count / PAGESIZE)
    curr_url = request.get_full_path()
    nPos = curr_url.find('&page')
    if nPos>0 :
        curr_url = request.get_full_path()[0:nPos]
    else :
         curr_url = request.get_full_path()

    return render(request, 'searchlist.html', locals())
```

Web站点的开发

- 新闻详情页面实现

主要技术点：

```
#文章详情页
def article(request):
    id = request.GET.get('id')
    articles_url = 'http://127.0.0.1:8000/article/'
    #构造url http://127.0.0.1:8000/article/id/
    articles_url = urljoin(articles_url, id)
    article = getdata(articles_url)
    return render(request, 'article.html', locals())
```

```
url:path('article/', article, name='article'),
```

```
template:article.html
```

Web站点的开发

- 新闻类别列表页
  - 子栏目、分页
  - 列表页（按子栏目）

url:

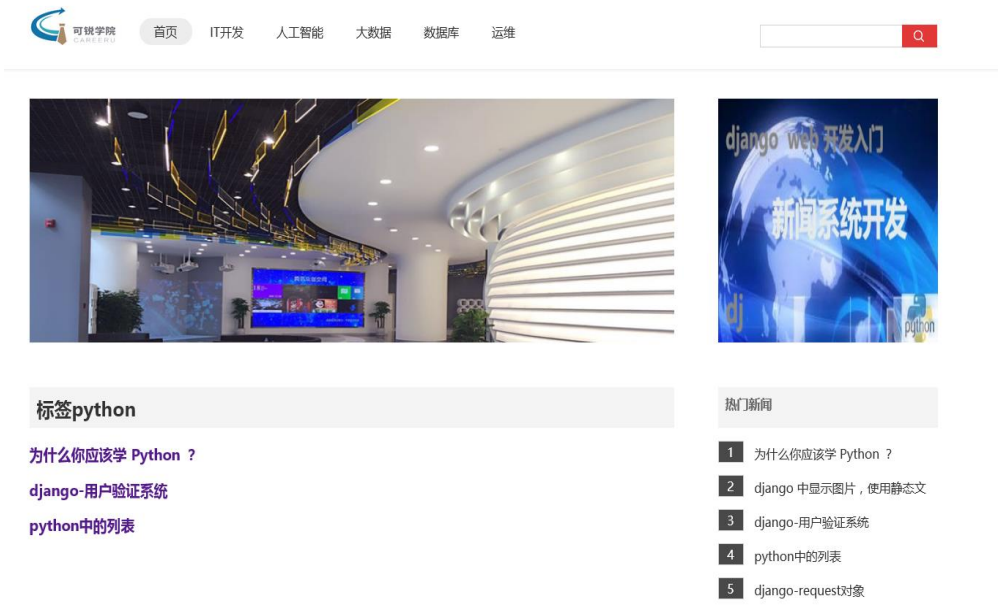
```
path(' category ', category, name=' category '),
path('item', item, name='item'),
```

template: category.html

```
#分类页
def category(request):

    categoryid = request.GET.get('cid')
    page = int(request.GET.get('page',1))
    # 取二级栏目
    data = {
        "categorys": categoryid,
    }
    items_list =  getdata(items_url,data)
    # 取新闻
    data = {
        "item__categorys": categoryid,
        "page": page,
        "ordering": '-id',
    }
    article_data = getdata(articles_url, data)
    article_list = article_data["results"]
    # 总记录数
    count = article_data["count"]
    # 下一页
    next = article_data["next"]
    nextpage = page + 1
    # 上一页
    previous = article_data["previous"]
    previouspage = page - 1
    # 总页数
    num_pages = int(count / PAGESIZE)
    curr_url = request.get_full_path()
    nPos = curr_url.find('&page')
    if nPos > 0:
        curr_url = request.get_full_path()[0:nPos]
    else:
        curr_url = request.get_full_path()
    return render(request, 'category.html', locals())
```

Web站点的开发

- 按标签显示新闻列表列表页
  - 列表页（按标签）
  - 分页



url:path('tag', tag, name= 'tag'),

template: tag.html

```python
# 按标签查询对应的文章列表
def tag(request):
    tagid = request.GET.get('tagid')
    page = int(request.GET.get('page', 1))
    data = {
        "tags": tagid,
        "page": page,
        "ordering": '-id',
    }
    article_data = getdata(articles_url, data)
    article_list = article_data["results"]
    # 总记录数
    count = article_data["count"]
    # 下一页
    next = article_data["next"]
    nextpage = page + 1
    # 上一页
    previous = article_data["previous"]
    previouspage = page - 1
    # 总页数
    num_pages = int(count / PAGESIZE)
    curr_url = request.get_full_path()
    nPos = curr_url.find('&page')
    if nPos > 0:
        curr_url = request.get_full_path()[0:nPos]
    else:
        curr_url = request.get_full_path()
    return render(request,'tag.html',locals())
```

登录--ajax实现

- 登录
- 注册


```
$('#loginsubmit').click(function(){
      var username=$('#login-form input[id="username"]').val();
      var password=$('#login-form input[id="password"]').val();
      var ErrorInfo = $("#ErrorInfo");
      if(post_flag) return;
         //标记当前状态为正在提交状态
      post_flag = true;
      var var_data = {
            "username": username,
            "password": password,
      }
      var json_str = JSON.stringify(var_data);
       $.ajax({
                url: "http://127.0.0.1:8005/userLogin/",
                type: "POST",
                data: json_str,
                contentType: "application/json",
                dataType: "json",
                success: function (result) {
                       if(result){
                            $.cookie('api_token', result.token);
                            $.cookie('user', result.username);
                            $.cookie('user_id', result.id);
                            $('#nologin').css('display', 'none');
                            $('#logined').css('display', 'block');
                            $('#navprofile').html(result.username);
                            $form_modal.removeClass('is-visible');
                            post_flag =false; //在提交成功之后将标志标记为可提交状态
                      }
                      else {
                        $('#login-form input[id="email"]').focus();
                         post_flag =false; //在提交成功之后将标志标记为可提交状态
                         return false;
                      }          },
                error:function(XMLHttpRequest, textStatus, errorThrown){
                   ....       post_flag =false; //在提交成功之后将标志标记为可提交状态
                 }
       })
      return false;

})
```

Drf web API 跨域问题

- **后台应用安装 cors-headers**

  pip install django-cors-headers

- **Setting配置**

```
INSTALLED_APPS = [ # ... 'corsheaders', # ... ]
MIDDLEWARE = [ # ... 具体前后位置参照文档，在SessionMiddleware 下面
'corsheaders.middleware.CorsMiddleware',
# ... 在CommonMiddleware 上面]
# 注意，这个，切记不能够`Allow-Origin: *`，因为返回的`Allow-Origin`必须跟原域匹配才可获取发送cookie 的权限
CORS_ORIGIN_REGEX_WHITELIST = r'.*'
# 必须有这个才接受前端跨域发送cookie
CORS_ALLOW_CREDENTIALS = True
```

- 还有参看setting文件

文章收藏—ajax实现

```
$('#articlefav').click(function(){
    userid = $.cookie('user_id');
    var isfav =  $('#articlefav').html();
    if (isfav == '已收藏'){
        return;
    }
    if (userid == null){
         $('.cd-user-modal').addClass('is-visible');
         $('#cd-login').addClass('is-selected');
         $('.cd-switcher').children('li').eq(0).children('a').addClass('selected');
         return;
    }
    var var_data = {
                    "articles": {{ article.id }},
                    "user":  userid,
                    "token ": $.cookie('api_token'),
        }
        var json_str = JSON.stringify(var_data);
        $.ajax({
                    url: "http://127.0.0.1:8005/userFav/",
                    type: "POST",
                    data: json_str,
                    contentType: "application/json",
                    dataType: "json",
                    success: function (data) {
                         setTimeout(function(){
                                }, 1000);
                       if (data) {
                            $('#articlefav').html('已收藏');
                            $form_modal.removeClass('is-visible');
                        }
                        else {
                         signup_selected();
                           ErrorInfo.removeClass().addClass("inputterror").html(data.msg);
                           $('#register-form input[id="email"]').focus();
                           return false;
                        }                    },
                    error:function(XMLHttpRequest, textStatus, errorThrown){
}           })
return false;
})
```

url配置知识点

- urls.py: 映射请求的URL到要执行的代码
- 使用正则表达式匹配
- 每一个应用在各自的module下面可以有自己的urls.py文件

```
from django.contrib import admin
from django.urls import path,re_path,include
from django.conf import settings
from article.views import index,category,article,search,item,tag
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    re_path(r'^$', index),
    path('category/', category, name='category'),
    path('article/', article, name='article'),
    path('search/', search, name='search'),
    path('item/', item, name='item'),
    path('tag/', tag, name='tag'),
    path('ueditor/', include('DjangoUeditor.urls')),

]
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Django Rest Framework介绍

RESTful API设计规范

- **URL**

  https://api.example.org/　　API很简单

- **url名词**

  https://api.example.com/v1/zoos　　因为是资源，最好起一个英文名词 zooms、animals、employees等

- **版本**

  https://api.example.com/v1/zoos?limit=10　　如果有版本最好可以显示在url　v1 可以代表第一版

- **提交方式**

  GET　　：从服务器取出资源（一项或多项）

  POST　　：在服务器新建一个资源

  PUT　　：在服务器更新资源（客户端提供改变后的完整资源）

  PATCH　　：在服务器更新资源（客户端提供改变的属性）

  DELETE　　：从服务器删除资源

Django Rest Framework介绍

- **status　　状态码**

  200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。

  201 CREATED - [POST/PUT/PATCH]：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 - [DELETE]：用户删除数据成功。

  400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。

  401 Unauthorized - [*]：表示用户没有权限（令牌、用户名、密码错误）。

  403 Forbidden - [*] 表示用户得到授权（与401错误相对），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，该操作是幂等的。

  406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。

  410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。

  422 Unprocesable entity - [POST/PUT/PATCH] 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，用户将无法判断发出的请求是否成功。

- **错误详细**

  错误处理，状态码是4xx时，应返回错误信息，error当做key。

  ```
  { error: "Invalid API key", }
  ```

目 录

软件安装和环境准备

- **虚拟机Ubuntu 16.04 环境**

  - **安装VMware12虚拟机管理软件及Ubuntu 16.04虚拟机**

    https://jingyan.baidu.com/article/c275f6ba07e269e33d756714.html

    Ubuntu 16.04 下载： http://cn.ubuntu.com/download/

  - **远程登录服务器--ssh的安装和配置**

    服务器端： https://jingyan.baidu.com/article/9c69d48fb9fd7b13c8024e6b.html

    windows: https://jingyan.baidu.com/article/fa4125acd320e228ac709200.html

    ssh无法连接解决方法： https://blog.csdn.net/anxpp/article/details/54620097

软件安装和环境准备

- **虚拟机Ubuntu 16.04**
  - 安装python 3.5

    sudo add-apt-repository ppa:jonathonf/python-3.5
    sudo apt-get update
    sudo apt-get install python3.5

    sudo apt-get install python3-pip
    pip3 install --upgrade pip （升级pip到最新版本）

  - **创建一个独立的Python环境**
    - 安装virtualenv：

      pip3 install virtualenv
    - 创建一个独立的环境

      virtualenv newsenv

      以上命令会创建一个包含你的Python环境的newsenv/目录。
    - 激活你的virtualenv

      source newsenv/bin/activate
    - deactivate命令随时停用你的virtualenv

软件安装和环境准备

- 安装Django-2.0.3

  pip3 install Django==2.0.3

- 检查Django是否成功安装

```
(newsenv) andy@ubuntu:~$ pip list
DEPRECATION: The default format will switch to columns in the future. You can u
e --format=(legacy|columns) (or define a format=(legacy|columns) in your pip.co
f under the [list] section) to disable this warning.
Django (2.0.3)
pip (9.0.3)
pytz (2018.4)
setuptools (39.0.1)
wheel (0.30.0)
You are using pip version 9.0.3, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
(newsenv) andy@ubuntu:~$ python
Python 3.5.3+ (default, Nov 29 2017, 08:55:08)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object is not callable
>>> django.VERSION
(2, 0, 3, 'final', 0)
>>>
```

部署

●安装依赖包

➢ Mysql

sudo apt-get install mysql-server
sudo apt-get install mysql-client
sudo apt-get install libmysqlclient-dev

➢ MySQLdb模块

sudo apt-get install python-dev
sudo apt-get install python-mysqldb

➢ 初始化数据库

运行mysql,执行下面脚本：

```
mysql> CREATE DATABASE IF NOT EXISTS news2 DEFAULT CHARSET utf8 COLLATE utf8_gen
eral_ci
    -> ;
Query OK, 1 row affected (0.00 sec)

mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| myblog             |
| mysql              |
| news2              |
| performance_schema |
| seckillshop        |
| sys                |
+--------------------+
7 rows in set (0.04 sec)
```

andy@ubuntu:~$ mysql -uroot -proot

```
CREATE DATABASE IF NOT EXISTS news2 DEFAULT CHARSET utf8 COLLATE utf8_general_ci
create user andy@'localhost' identified by 'andy';
grant all privileges on newtwo.* to andy@'localhost';
grant all privileges on newtwo.* to root@'localhost';
```

部署

●获取最新代码

- 远程连接服务器。
- ftp 将代码上传生产服务器（或进入项目根目录，从远程仓库拉取最新的代码)。
- 如果项目引入了新的依赖，需要执行 pip3 install -r requirement.txt 安装最新依赖。
  - source newsenv/bin/activate
  - pip3 install -r requirement.txt
- 修改settings中的数据库配置

  修改ALLOWED_HOSTS = ['*']

- 如果修改或新增了项目静态文件，需要执行 python3 manage.py collectstatic 收集静态文件。
- 执行 python3 manage.py migrate 迁移数据库。

部署

python3 manage.py createsuperuser (admin/admin)

试运行 python3 manage.py runserver 0.0.0.0:8010

```
(newsenv) andy@ubuntu:~/newsdjango2$ python3 manage.py runserver 0.0.0.0:8010
Performing system checks...

System check identified some issues:

WARNINGS:
article.Article.publish_date: (fields.W161) Fixed default value provided.
        HINT: It seems you set a fixed date / time / datetime value as default f
or this field. This may not be what you want. If you want to have the current da
te as default, use `django.utils.timezone.now`

System check identified 1 issue (0 silenced).
May 08, 2018 - 16:00:02
Django version 2.0.3, using settings 'newsdjango2.settings'
Starting development server at http://0.0.0.0:8010/
Quit the server with CONTROL-C.
```

部署： Django web框架开发招数

●安装 apache2 和 mod_wsgi

sudo apt-get install apache2
# Python 3.5
sudo apt-get install libapache2-mod-wsgi-py3

apache2的配置：

在/etc/apache2/sites-enabled/中新建一个文件news.conf

```
<VirtualHost *:80>
   ServerName  127.0.0.1:80
   DocumentRoot  /home/andy/newsdjango2
   Alias /media/ /home/andy/newsdjango2/media/
   Alias /static/ /home/andy/newsdjango2/static/
   <Directory /home/andy/newsdjango2/media>
    Require all granted
   </Directory>
   <Directory /home/andy/newsdjango2/static>
     Require all granted
   </Directory>
  WSGIDaemonProcess http://127.0.0.1:80  python-
path=/home/andy/newsdjango2:/home/andy/newsdjango2/
newsenv/lib/python3.5/site-packages
  WSGIProcessGroup http://127.0.0.1:80
   WSGIScriptAlias / /home/andy/newsdjango2/newsdjango2/wsgi.py
   <Directory /home/andy/newsdjango2/newsdjango2>
            <Files wsgi.py>
                  Require all granted
            </Files>
   </Directory>
</VirtualHost>
```

/etc/apache2/apache2.conf 相关配置

```
WSGIScriptAlias /   /home/andy/newsdjango2/newsdjango2/wsgi.py
Alias /static/  /home/andy/newsdjango2/static/
Alias /media/   /home/andy/newsdjango2/media/
```

●启动、停步apache2

sudo service apache2 start

sudo service apache2 stop 停止服务

部署： Django web框架开发招数

- **修改wsgi的配置**

```
import os
from os.path import join,dirname,abspath
from django.core.wsgi import get_wsgi_application

PROJECT_DIR = dirname(dirname(abspath(__file__)))
import sys
sys.path.insert(0,PROJECT_DIR)
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "newsdjango2.settings")
application = get_wsgi_application()
```

- **修改setting的配置**

  ALLOWED_HOSTS = ['*']

- 用户上传目录还要设置给 www-data 用户的写权限

cd media/ # 进入media文件夹

sudo chgrp -R www-data media

sudo chmod -R g+w media

- **重启apache2**

sudo a2ensite news.conf （使配置文件生效）
sudo service apache2 reload

部署： Django web框架开发招数

●部署经验总结

- 出错的话，一定要多看看apache2的err log。
  命令行输入： tail /var/log/apache2/error.log

● **可在代码目录中先执行 python manage.py runserver 0.0.0.0:8080**

检查代码及相关依赖包是否完备

● **检查/etc/apache2/apache2.conf 相关配置**

● **在ubuntu上测试： curl localhost:80**

感谢观看！

THANKS

这不仅仅是一次学习，而是一次改变！