十次方项目遇到的问题解决

1. django-haystack装不上

2.关于登录

3.drf中使用haystack对接es

bug说明：

4. 关于后台管理，直接使用xadmin

1. 安装

2. 使用

站点的全局配置

站点Model管理

站点保存对象数据方法重写

自定义用户管理

5. 欠缺的技术点补充

6. 关于MongoDB数据库，建议直接用mysql数据库

7. 关于协同开发git的使用

8. 关于时间的问题

9. 关于分组的问题

10. 关于接口文档

11. 关于项目汇报

12. 关于代码

1. django-haystack装不上

解决方法

```
pip3 install setuptools_scm
```

2.关于登录

Django REST framework JWT提供了登录签发JWT的视图，可以直接使用

```
from rest_framework_jwt.views import obtain_jwt_token
urlpatterns = [
    url(r'^authorizations/$', obtain_jwt_token),
]
```

但是默认的返回值仅有token，我们还需在返回值中增加username和user_id。

通过修改该视图的返回值可以完成我们的需求。

在users/utils.py 中，创建

```
def jwt_response_payload_handler(token, user=None, request=None):
    """
    自定义jwt认证成功返回数据,需要什么写什么
    """
    return {
        'token': token,
        'user_id': user.id,
        'username': user.username
    }
```

修改配置文件

```
# JWT
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
    'JWT_RESPONSE_PAYLOAD_HANDLER': 'users.utils.jwt_response_payload_handler',
}
```

3.drf中使用haystack对接es

1）安装

```
pip install drf-haystack
pip install elasticsearch==2.4.1
```

drf-haystack是为了在REST framework中使用haystack而进行的封装（如果在Django中使用haystack，则安装django-haystack即可）。

2）注册应用

```
INSTALLED_APPS = [
    ...
    'haystack',
    ...
]
```

3）配置

在配置文件中配置haystack使用的搜索引擎后端

```
# Haystack
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.elasticsearch_backend.ElasticsearchSearchEngine',
        'URL': 'http://10.211.55.5:9200/',  # 此处为elasticsearch运行的服务器ip地址，端口号固定为9200
        'INDEX_NAME': 'meiduo',  # 指定elasticsearch建立的索引库的名称
    },
}
# 当添加、修改、删除数据时，自动生成索引
HAYSTACK_SIGNAL_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'
```

注意：

HAYSTACK_SIGNAL_PROCESSOR 的配置保证了在Django运行起来后，有新的数据产生时，haystack仍然可以让Elasticsearch实时生成新数据的索引

4）创建索引类

通过创建索引类，来指明让搜索引擎对哪些字段建立索引，也就是可以通过哪些字段的关键字来检索数据。

在goods应用中新建search_indexes.py文件，用于存放索引类

```
from haystack import indexes
from .models import SKU
class SKUIndex(indexes.SearchIndex, indexes.Indexable):
    """
    SKU索引数据模型类
    """
    text = indexes.CharField(document=True, use_template=True)
    id = indexes.IntegerField(model_attr='id')
```

```
    name = indexes.CharField(model_attr='name')
    price = indexes.DecimalField(model_attr='price')
    default_image_url = indexes.CharField(model_attr='default_image_
url')
    comments = indexes.IntegerField(model_attr='comments')
    def get_model(self):
        """返回建立索引的模型类"""
        return SKU
    def index_queryset(self, using=None):
        """返回要建立索引的数据查询集"""
        return self.get_model().objects.filter(is_launched=True)
```

在SKUIndex建立的字段，都可以借助haystack由elasticsearch搜索引擎查询。

其中text字段我们声明为document=True，表名该字段是主要进行关键字查询的字段， 该字段的索引值可以由多个数据库模型类字段组成，具体由哪些模型类字段组成，我们用use_template=True表示后续通过模板来指明。其他字段都是通过model_attr选项指明引用数据库模型类的特定字段。

在REST framework中，索引类的字段会作为查询结果返回数据的来源。

6）在templates目录中创建text字段使用的模板文件

具体在templates/search/indexes/goods/sku_text.txt文件中定义

```
{{ object.name }}
{{ object.caption }}
{{ object.id }}
```

此模板指明当将关键词通过text参数名传递时，可以通过sku的name、caption、id来进行关键字索引查询。

7）手动生成初始索引

```
python manage.py rebuild_index
```

8）创建序列化器

在goods/serializers.py中创建haystack序列化器

```
from drf_haystack.serializers import HaystackSerializer
class SKUIndexSerializer(HaystackSerializer):
    """
    SKU索引结果数据序列化器
    """
    class Meta:
        index_classes = [SKUIndex]
        fields = ('text', 'id', 'name', 'price', 'default_image_url',
```

```
    'comments')
```

注意fields属性的字段名与SKUIndex类的字段对应。

9）创建视图

在goods/views.py中创建视图

```
from drf_haystack.viewsets import HaystackViewSet
class SKUSearchViewSet(HaystackViewSet):
    """
    SKU搜索
    """
    index_models = [SKU]
    serializer_class = SKUIndexSerializer
```

注意：

- 该视图会返回搜索结果的列表数据，所以如果可以为视图增加REST framework的分页功能。
- 我们在实现商品列表页面时已经定义了全局的分页配置，所以此搜索视图会使用全局的分页配置。

返回的数据举例如下：

```
{
    "count": 10,
    "next": "http://api.meiduo.site:8000/skus/search/?page=2&text=%E5%8D%8E",
    "previous": null,
    "results": [
        {
            "text": "华为 HUAWEI P10 Plus 6GB+64GB 钻雕金 移动联通电信4G手机 双卡双待\nwifi双天线设计！徕卡人像摄影！P10徕卡双摄拍照，低至2988元！\n9",
            "id": 9,
            "name": "华为 HUAWEI P10 Plus 6GB+64GB 钻雕金 移动联通电信4G手机 双卡双待",
            "price": "3388.00",
            "default_image_url": "http://10.211.55.5:8888/group1/M00/00/02/CtM3BVrRcUeAHp9pAARfIK95am88523545",
            "comments": 0
        },
        {
            "text": "华为 HUAWEI P10 Plus 6GB+128GB 钻雕金 移动联通电信4G手机 双卡双待\nwifi双天线设计！徕卡人像摄影！P10徕卡双摄拍照，低至2988元！\n10",
            "id": 10,
```

```
            "name": "华为 HUAWEI P10 Plus 6GB+128GB 钻雕金 移动联通电信4G手机 双卡双待",
            "price": "3788.00",
            "default_image_url": "http://10.211.55.5:8888/group1/M00/00/02/CtM3BVrRchWAMc8rAARfIK95am88158618",
            "comments": 5
        }
    ]
}
```

10）定义路由

通过REST framework的router来定义路由

```
router = DefaultRouter()
router.register('skus/search', views.SKUSearchViewSet, base_name='skus_search')
...
urlpatterns += router.urls
```

11）测试

我们可以GET方法访问如下链接进行测试

```
http://api.meiduo.site:8000/skus/search/?text=wifi
http://api.meiduo.site:8000/skus/search/?id=1
http://api.meiduo.site:8000/skus/search/?name=iphone
```

bug说明：

如果在配置完haystack并启动程序后，出现如下异常，是因为drf-haystack还没有适配最新版本的REST framework框架

```
    urlconf_module = import_module(urlconf_module)
  File "/Users/delron/.virtualenv/meiduo_27/lib/python3.6/importlib/__init__.py", line 126, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 978, in _gcd_import
  File "<frozen importlib._bootstrap>", line 961, in _find_and_load
  File "<frozen importlib._bootstrap>", line 950, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 655, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 678, in exec_module
  File "<frozen importlib._bootstrap>", line 205, in _call_with_frames_removed
  File "/Users/delron/Desktop/meiduo/meiduo_mall/meiduo_mall/apps/users/urls.py", line 5, in <module>
    from . import views
  File "/Users/delron/Desktop/meiduo/meiduo_mall/meiduo_mall/apps/users/views.py", line 19, in <module>
    from goods.serializers import SKUSerializer
  File "/Users/delron/Desktop/meiduo/meiduo_mall/meiduo_mall/apps/goods/serializers.py", line 2, in <module>
    from drf_haystack.serializers import HaystackSerializer
  File "/Users/delron/.virtualenv/meiduo_27/lib/python3.6/site-packages/drf_haystack/serializers.py", line 24, in <module>
    from rest_framework.pagination import _get_count
ImportError: cannot import name '_get_count'
```

可以通过修改REST framework框架代码，补充`_get_count`函数定义即可

文件路径 虚拟环境下的 `lib/python3.6/site-packages/rest_framework/pagination.py`

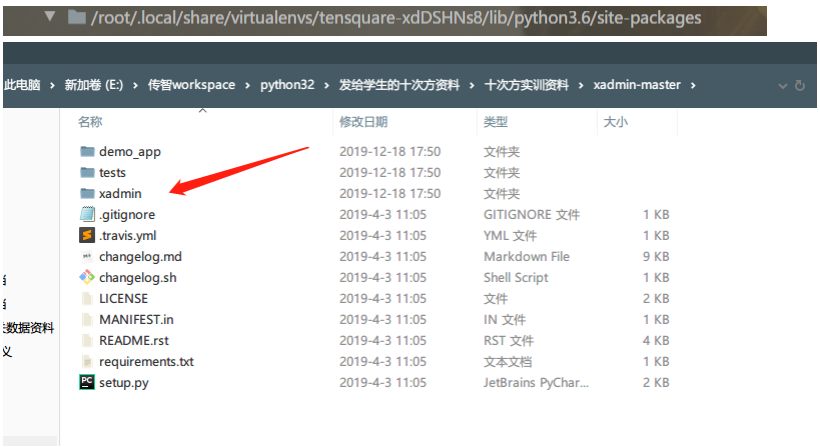
```
def _get_count(queryset):
```

```
    """
    Determine an object count, supporting either querysets or regular lists.
    """
    try:
        return queryset.count()
    except (AttributeError, TypeError):
        return len(queryset)
```

4. 关于后台管理，直接使用xadmin

1. 安装

Django给的xadmin-master中的xadmin文件夹直接拷贝到你虚拟环境下的lib/site-package下



在配置文件中注册如下应用

```
INSTALLED_APPS = [
    ...
    'xadmin',
    'crispy_forms',
    'reversion',
    ...
]
```

xadmin有建立自己的数据库模型类，需要进行数据库迁移

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

在总路由中添加xadmin的路由信息

```
import xadmin
urlpatterns = [
    # url(r'^admin/', admin.site.urls),
    url(r'xadmin/', include(xadmin.site.urls)),
    ...
]
```

2. 使用

- xadmin不再使用Django的admin.py，而是需要编写代码在adminx.py文件中。
- xadmin的站点管理类不用继承`admin.ModelAdmin`，而是直接继承`object`即可。

在goods应用中创建adminx.py文件。

站点的全局配置

```
import xadmin
from xadmin import views
from . import models
class BaseSetting(object):
    """xadmin的基本配置"""
    enable_themes = True  # 开启主题切换功能
    use_bootswatch = True
xadmin.site.register(views.BaseAdminView, BaseSetting)
class GlobalSettings(object):
    """xadmin的全局配置"""
    site_title = "美多商城运营管理系统"  # 设置站点标题
    site_footer = "美多商城集团有限公司"  # 设置站点的页脚
    menu_style = "accordion"  # 设置菜单折叠
xadmin.site.register(views.CommAdminView, GlobalSettings)
```

站点Model管理

xadmin可以使用的页面样式控制基本与Django原生的admin一直。

- **list_display**控制列表展示的字段
- **search_fields**控制可以通过搜索框搜索的字段名称，xadmin使用的是模糊查询
- **list_filter**可以进行过滤操作的列
- **ordering**默认排序的字段
- **readonly_fields**在编辑页面的只读字段
- **exclude**在编辑页面隐藏的字段
- **list_editable**在列表页可以快速直接编辑的字段
- **show_detail_fileds**在列表页提供快速显示详情信息
- **refresh_times**指定列表页的定时刷新
- **list_export**控制列表页导出数据的可选格式
- **show_bookmarks**控制是否显示书签功能
- **data_charts**控制显示图标的样式
- **model_icon**控制菜单的图标

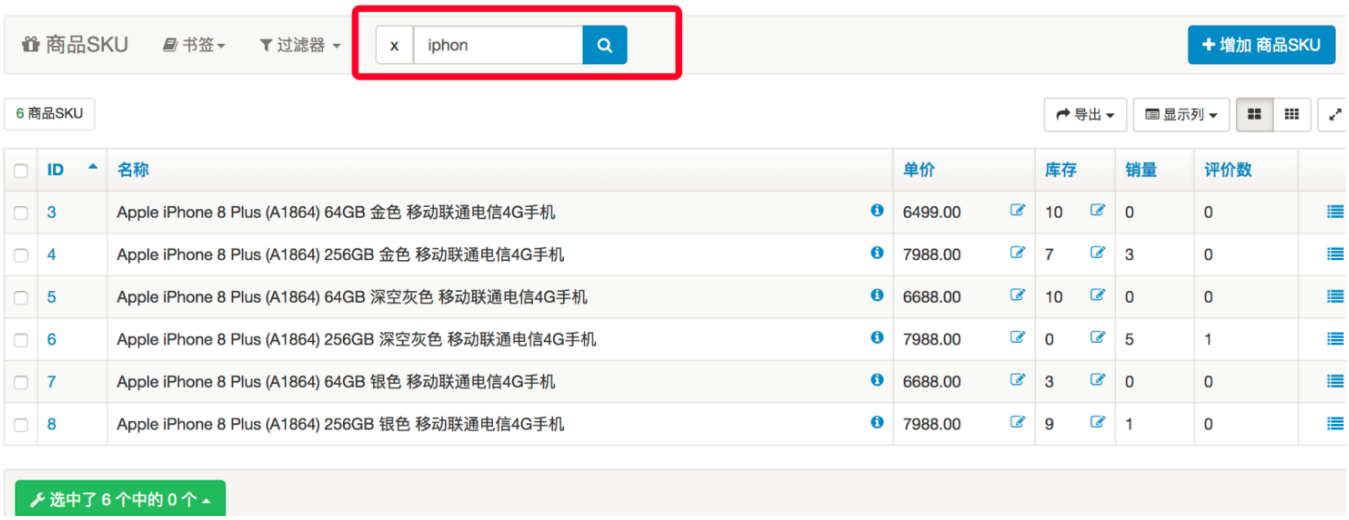
1） model_icon

```
class SKUAdmin(object):
    model_icon = 'fa fa-gift'
xadmin.site.register(models.SKU, SKUAdmin)
```

可选的图标样式参考http://fontawesome.dashgame.com/

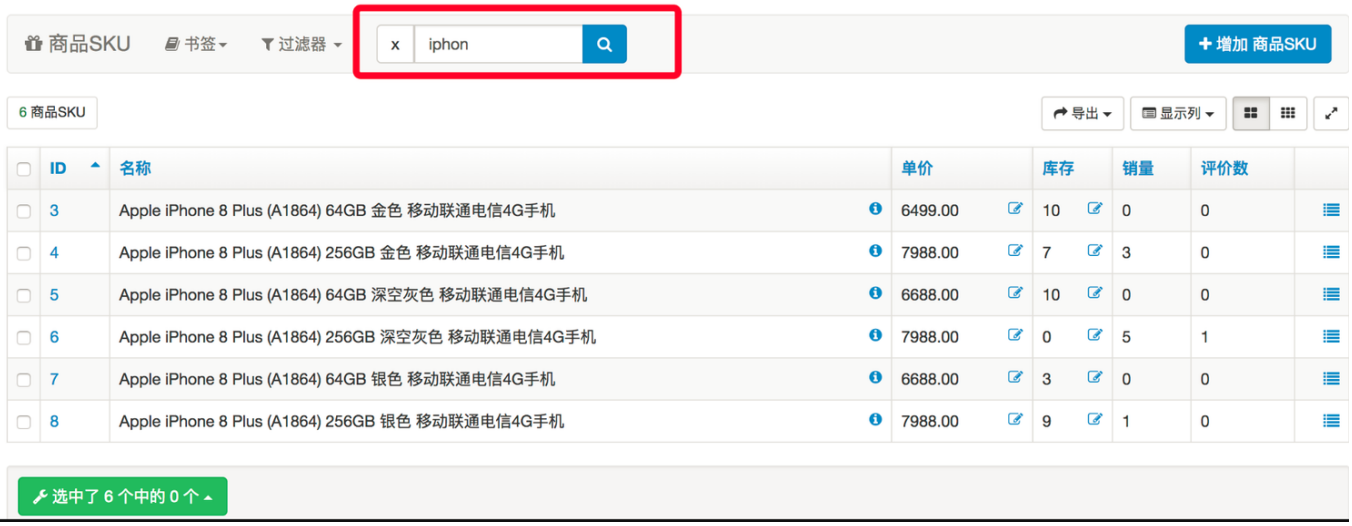
2） list_display

```
1 list_display = ['id', 'name', 'price', 'stock', 'sales', 'comments']
```



3）search_fields

```
1 search_fields = ['id','name']
```



4）list_filter

```
1 list_filter = ['category']
```

5）list_editable

```
list_editable = ['price', 'stock']
```

6）show_detail_fields

```
show_detail_fields = ['name']
```

7）show_bookmarks

```
show_bookmarks = True
```

8）list_export

```
list_export = ['xls', 'csv', 'xml']
```

注意，导出到xls（excel) 需要安装**xlwt**扩展

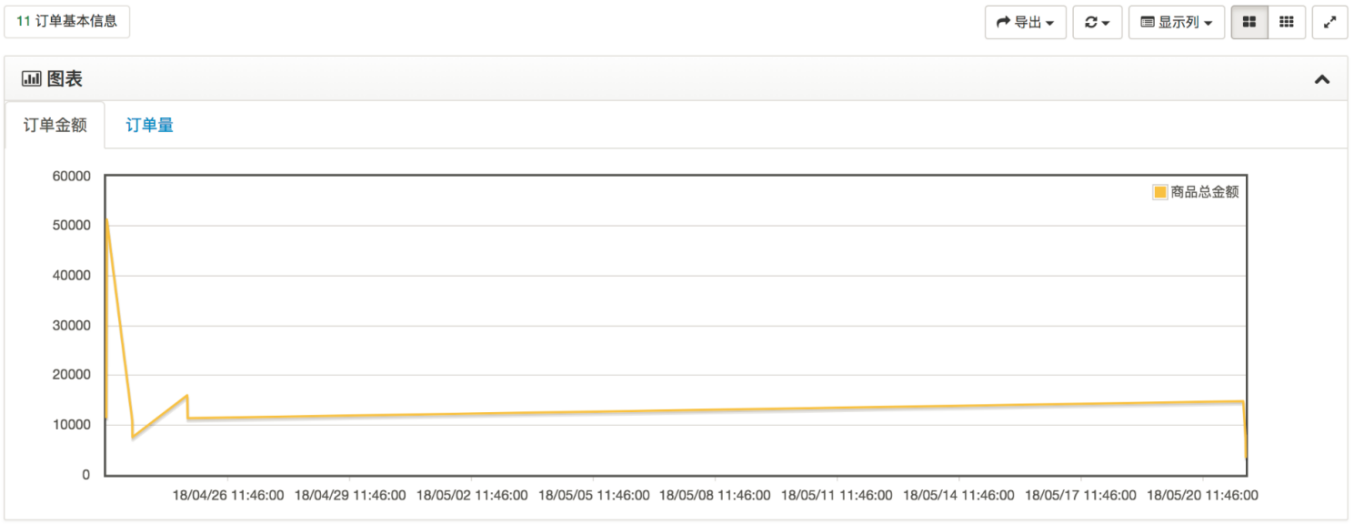
9）refresh_times

```
class OrderAdmin(object):
    list_display = ['order_id', 'create_time', 'total_amount', 'pay_method', 'status']
    refresh_times = [3, 5]  # 可选以支持按多长时间(秒)刷新页面
```

10）data_charts

```
data_charts = {
        "order_amount": {'title': '订单金额', "x-field": "create_time", "y-field": ('total_amount',),
                         "order": ('create_time',)},
        "order_count": {'title': '订单量', "x-field": "create_time", "y-field": ('total_count',),
                         "order": ('create_time',)},
    }
```

- title 控制图标名称
- x-field 控制x轴字段
- y-field 控制y轴字段，可以是多个值
- order 控制默认排序



11）readonly_fields

```
class SKUAdmin(object):
    ...
    readonly_fields = ['sales', 'comments']
```

销量	6
评价数	1

站点保存对象数据方法重写

在Django的原生admin站点中，如果想要在站点保存或删除数据时，补充自定义行为，可以重写如下方法：

- `save_model(self, request, obj, form, change)`
- `delete_model(self, request, obj)`

而在xadmin中，需要重写如下方法：

- `save_models(self)`
- `delete_model(self)`

在方法中，如果需要用到当前处理的模型类对象，需要通过`self.obj`来获取，如

```
class SKUSpecificationAdmin(object):
    def save_models(self):
        # 保存数据对象
        obj = self.new_obj
        obj.save()
        # 补充自定义行为
        from celery_tasks.html.tasks import generate_static_sku_detail_html
```

```
        generate_static_sku_detail_html.delay(obj.sku.id)
    def delete_model(self):
        # 删除数据对象
        obj = self.obj
        sku_id = obj.sku.id
        obj.delete()
        # 补充自定义行为
        from celery_tasks.html.tasks import generate_static_sku_detail_html
        generate_static_sku_detail_html.delay(sku_id)
```

自定义用户管理

xadmin会自动为admin站点添加用户User的管理配置
xadmin使用xadmin.plugins.auth.UserAdmin来配置
如果需要自定义User配置的话，需要先unregister(User)，在添加自己的User配置并注册

```
import xadmin
# Register your models here.
from .models import User
from xadmin.plugins import auth
class UserAdmin(auth.UserAdmin):
    list_display = ['id', 'username', 'mobile', 'email', 'date_joined']
    readonly_fields = ['last_login', 'date_joined']
    search_fields = ('username', 'first_name', 'last_name', 'email', 'mobile')
    style_fields = {'user_permissions': 'm2m_transfer', 'groups': 'm2m_transfer'}
    def get_model_form(self, **kwargs):
        if self.org_obj is None:
            self.fields = ['username', 'mobile', 'is_staff']
        return super().get_model_form(**kwargs)
xadmin.site.unregister(User)
xadmin.site.register(User, UserAdmin)
```

5. 欠缺的技术点补充

我们在序列化文章列表的时候希望序列化先出出来文章的评论数量来，但是数据库中没有存储评论的数量，此时在定义序列化器添加article_comments_count，由SerializerMethodField生成。
定义方法get_字段名，返回值就是序列化每个对象时这个字段的值。

案例：

```
from rest_framework import serializers
from .models import Article, Comment

class ArticleSerializer(serializers.ModelSerializer):
    article_comments_count = serializers.SerializerMethodField()
    class Meta:
        model = Article
        fields = ('id', 'title', 'summary', 'content', 'create_user', 'create_time', 'article_comments_count')

    def get_article_comments_count(self, obj):
        return obj.article_comments.all().count()

class CommentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Comment
        fields = ('id', 'article', 'comment', 'create_user_id', 'create_user_name', 'create_time')
```

6. 关于MongoDB数据库，建议直接用mysql数据库

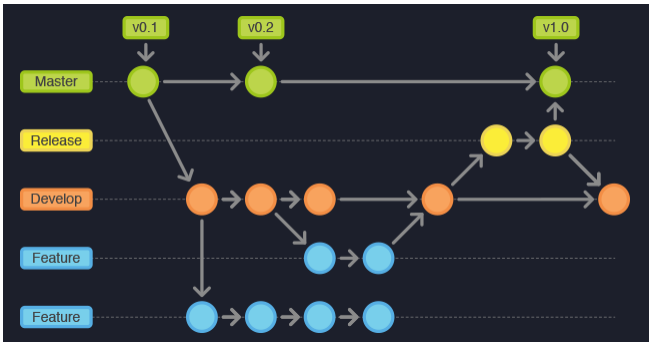
吐槽部分的点赞和收藏是存在redis中的，建议提前告知学员。

```
class Spit(models.Model):
    content = models.TextField(max_length=10000, verbose_name="吐槽内容")  # 吐槽内容
    publishtime = models.DateTimeField(default=datetime.utcnow)  # 发布日期
    userid = models.CharField(max_length=100, verbose_name="发布人id")  # 发布人ID
    nickname = models.CharField(max_length=100, verbose_name="发布人昵称")  # 发布人昵称
    visits = models.IntegerField(default=0, verbose_name="浏览量")  # 浏览量
    thumbup = models.IntegerField(default=0, verbose_name="点赞数")
```

```
    # 点赞数
8       comment = models.IntegerField(default=0, verbose_name="回复数")
    # 回复数
9       avatar = models.CharField(max_length=100, verbose_name="用户的头
    像")  # 用户的头像
10      parent = models.ForeignKey("self", on_delete=models.SET_NULL, re
    lated_name='subs', null=True, blank=True,verbose_name='被吐槽的吐槽')
    # 上级ID
11      collected = models.BooleanField(default=False)  # 是否收藏
12      hasthumbup = models.BooleanField(default=False)  # 是否点赞
```

7. 关于协同开发git的使用

1. 创建项目小组拉成员，分配角色
2. 创建项目、创建分支，采用gitfow流程开发（需要辅导老师提前讲解）



3. 建议采用云服务器集中管理数据库和docker

8. 关于时间的问题

建议延长实训时间，从3天改为5天，学生普遍代码敲不完，敲代码的时候需要反复看前面的课件

9. 关于分组的问题

需要重新进行分组，有些不走web的同学不参与项目实训的。学得很差的同学需要特别关注，有出现因为实在不会而放弃的现象。

10. 关于接口文档

接口文档由于是从java改过来的，有些的东西与Python类型不是很匹配，建议学生用chrom抓包去看部署在外网的项目的接口返回的数据。

11. 关于项目汇报

31期采用的是总体功能展示 + 随便模块讲解 + 经验总结汇报的的模式
建议后边的班级提前调好共屏， 投影仪在进行代码讲解的时候后边的同学看不清楚。建议采用远程控制 + 屏幕共享
的形式，mac的可以考虑QQ共屏 + 屏幕共享的形式。

12. 关于代码

建议小组开会指定一套代码规范，按照规范写代码，对变量和方法进行明白，便于整合后维护。