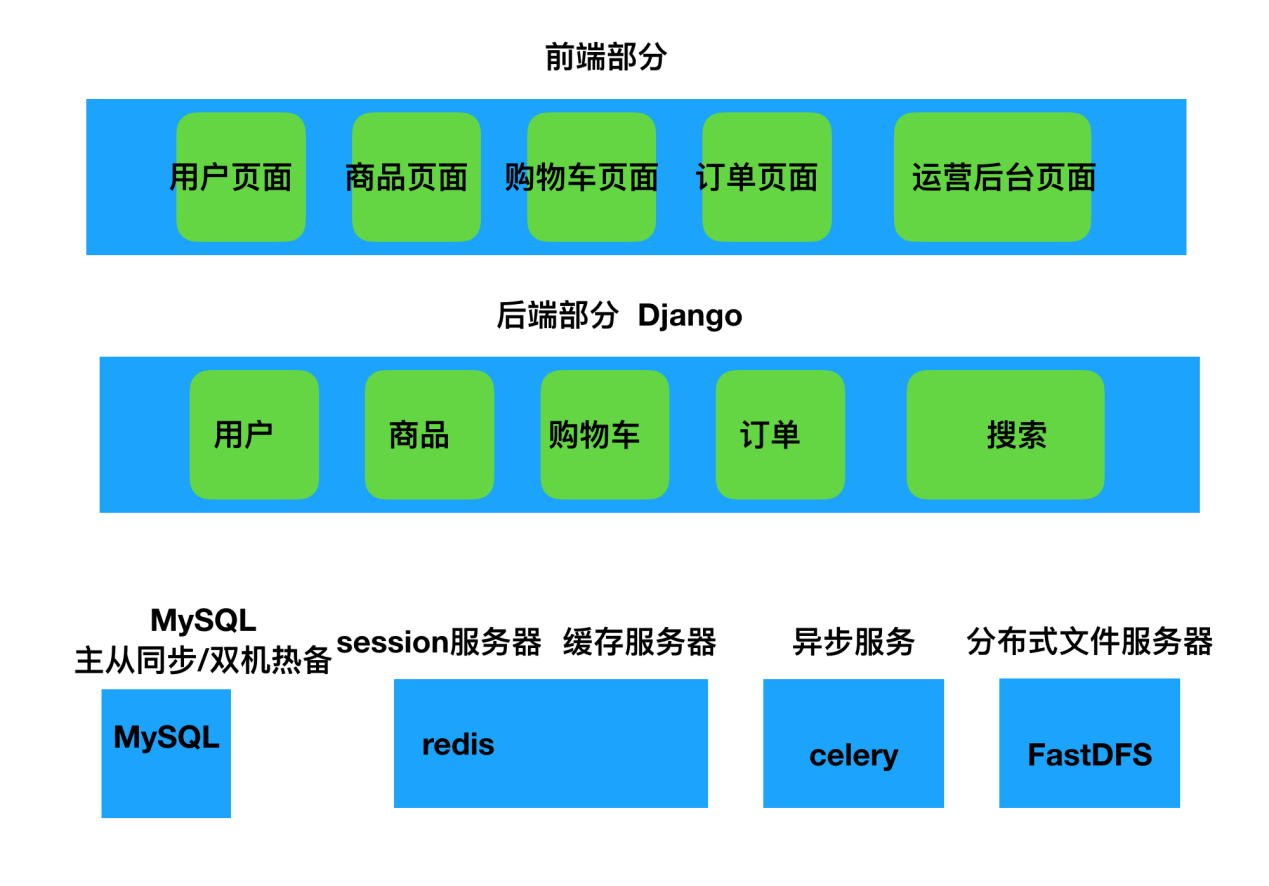
# 项目架构



# 数据库表结构



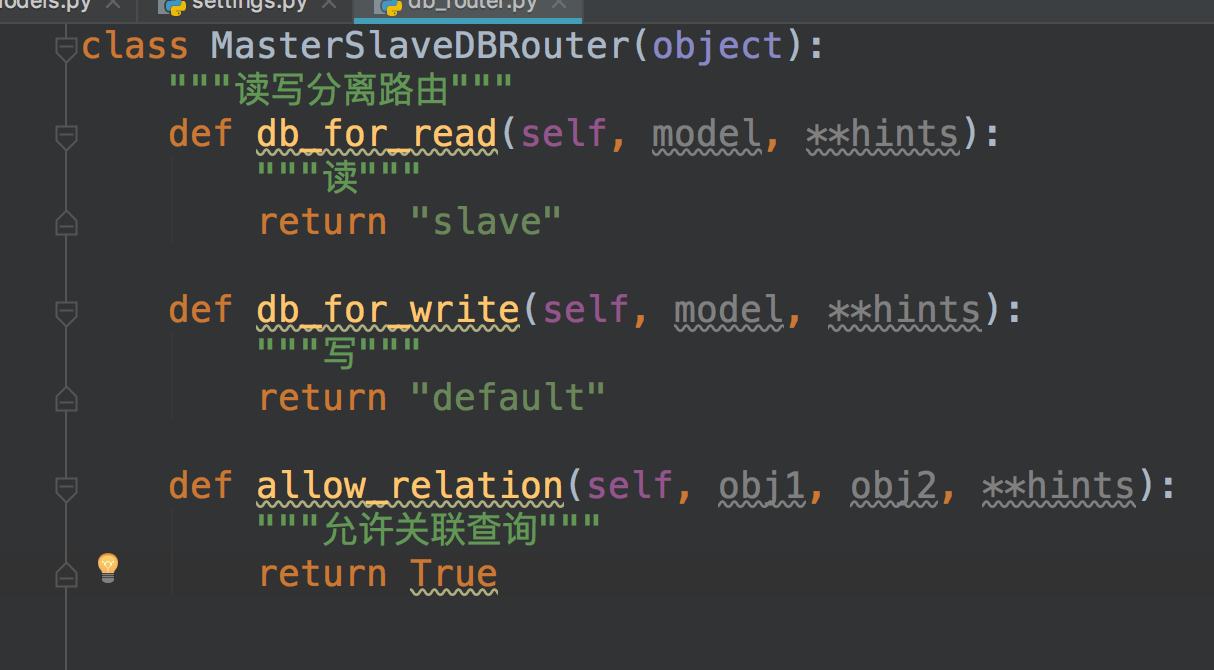
# 数据库读写分离



# Django读写分离配置



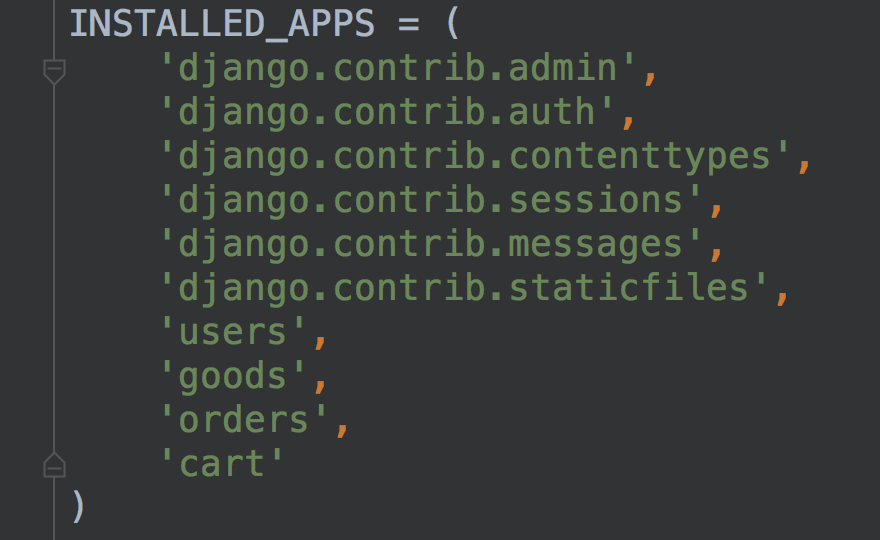
新建utils/db\_router.py

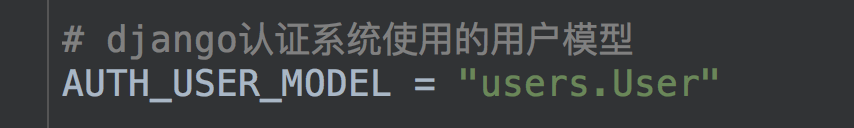


课后阅读资料

<http://python.usyiyi.cn/documents/django_182/topics/db/multi-db.html>

# 5. **用户认证模型**





注意： AUTH\_USER\_MODEL配置参数要在第一次迁移数据库之前配置，否则可能django的认证系统工作不正常

6.类视图

将视图view以类的形式定义 必须继承django.views.generic.View

通用类视图基类：

django.views.generic.View ( 与django.views.generic.base.View 是同一个)

urls.py中配置路由使用类视图的as\_view()方法

由dispatch()方法具体将请求request分发至对应请求方式的处理方法中（get、post等）

扩展阅读：

<http://python.usyiyi.cn/translate/django_182/topics/class-based-views/intro.html>

# 用户模块开发

Django认证系统文档

<http://python.usyiyi.cn/documents/django_182/topics/auth/default.html>

用户注册

1. 保存用户数据到数据库 生成token
2. 发送邮件给用户

3.用户点击邮件的激活链接 <http://127.0.0.1:8000/user/active>/fdsjflkjglkjj548502jrio23fdsffdsjt92ji2398jgfkdj9t439

3.1用户点击链接 访问了我们的服务器 访问了激活的视图

<http://127.0.0.1:8000/user/active>/fdsjflkjglkjj548502jrio23fdsffdsjt92ji2398jgfkdj9t439

用户id -> 签名 > token -> 令牌(口令)

4.发送邮件 （用户点击邮件中的链接地址---激活的地址）

激活：获取token 设置用户的激活状态

把用户的id放到token中 签名

itsdangerous pip install itsdangerous

文档 <http://itsdangerous.readthedocs.io/en/latest/>

celery pip install celery



Celery 使用

1

# 创建客户端

# 参1 指定任务的文件 参2 指定 存放任务队列的redis 语法 redis://:密码@ip/服务器号

app = Celery('celerytask.tasks', broker='redis://192.168.1.135/3')

2

#装饰器 成为一个异步任务函数

@app.task

def send\_active\_email(title,address,token):

'''异步任务的函数'''

# 这里处理发送邮件的功能

3

启动worker: celery -A celerytask.tasks worker -l info

Woker端的task文件代码里 要添加如下代码 用来加载django环境 导入setttings

os.environ["DJANGO\_SETTINGS\_MODULE"] = "dailyfresh.settings"

import django

django.setup()

用户激活

用户登录

使用redis作为session服务器

pip install django-redis

django-redis文档：

[http://django-redis-chs.readthedocs.io/zh\_CN/latest/#](http://django-redis-chs.readthedocs.io/zh_CN/latest/)

django的session使用文档：

<http://python.usyiyi.cn/documents/django_182/topics/http/sessions.html>

登录-> 保存用户的session数据

只要浏览器中cookie能够在访问的时候带上session\_id,

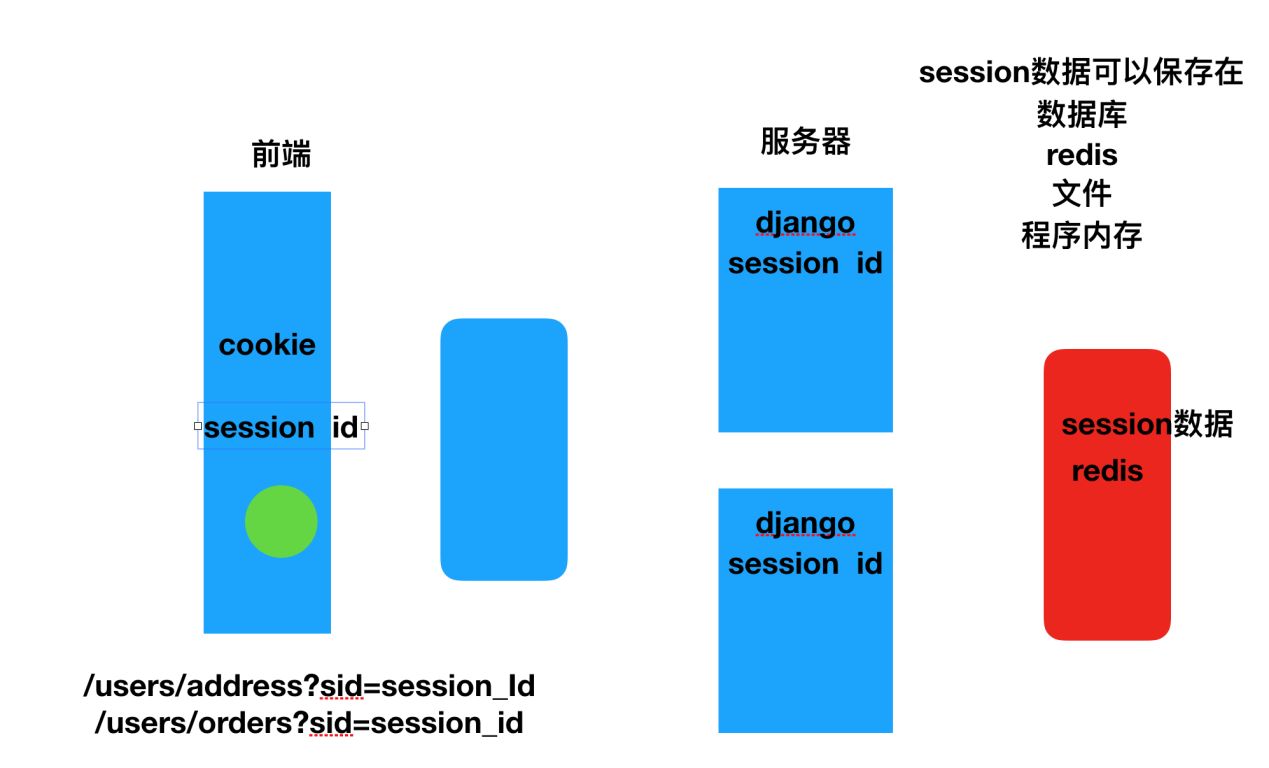
用户访问其他页面，不需要再次登录

session保存在redis里

Django-redis中文文档

<http://django-redis-chs.readthedocs.io/zh_CN/latest/> http://django-redis-chs.readthedocs.io/zh\_CN/latest/

如果cookie中没有session\_id，访问的时候后端无法识别用户，要求用户从新登录



## 用户退出

## 用户地址

## 用户个人信息

浏览历史记录：

产生: 用户访问商品详情页面的时候记录 次数频繁 ，所以放到内存型数据库中 保存sku\_id

不放到session数据中，方便用户退出再登陆的时候还能查看到数据

手动直接操作redis，存放历史记录信息

查询： 只在用户的个人信息页面中看到

“key”： 字符串、列表、哈希、set zset

用一条记录保存所有人的信息, 维护起来不方便，不采纳

“history”: {“user\_1”: “1,2,3,4,5,6” , “user\_2”: “”}

conn.hget(“history”, “user\_1”)

每个用户一条数据，单独维护

“history\_1”: [1,2,2,3,4,4]

conn.lrange(“history\_1”,0, 4)

### django查询集queryset使用方法

1. Address.objects.all().order\_by("-create\_time")  
   Address.objects.filter().order\_by("-create\_time")  
     
   select \* from tbl\_addr where .... order by create\_time desc

Address.objects.filter().order\_by("-create\_time")

select \* from tbl where .... order by create\_time desc

Address.objects.all().order\_by(“-create\_time”)[0]

Address.objects.latest(“-create\_time”)

关于Django查询方法的文档:

<http://python.usyiyi.cn/documents/django_182/ref/models/querysets.html>

### 7.7.2 使用redis保存用户的浏览历史

通过django\_redis 的 get\_redis\_connection(连接名称) 在程序中使用redis

### 7.7.3 redis中的用户浏览历史数据类型选择

<http://redisdoc.com/index.html>

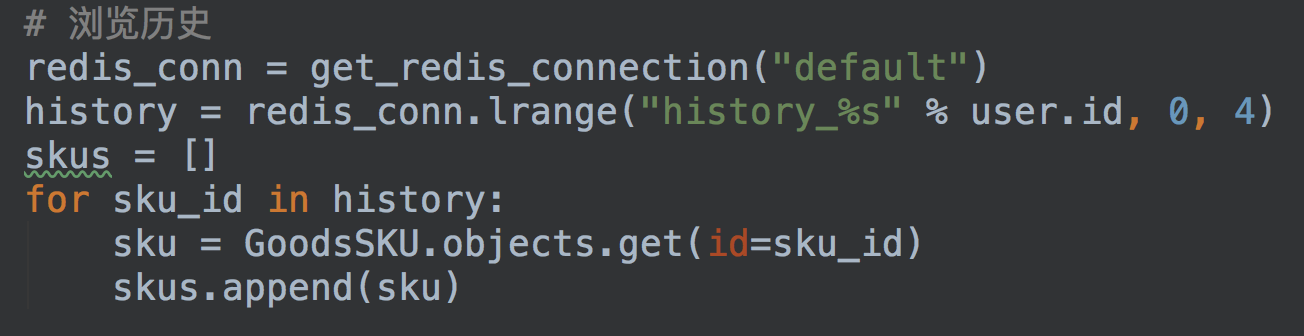
[http://redis-py.readthedocs.io/en/latest/#indices-and-tables](http://redis-py.readthedocs.io/en/latest/" \l "indices-and-tables)

浏览历史记录 redis

字符串 列表 哈希 set

sku\_id

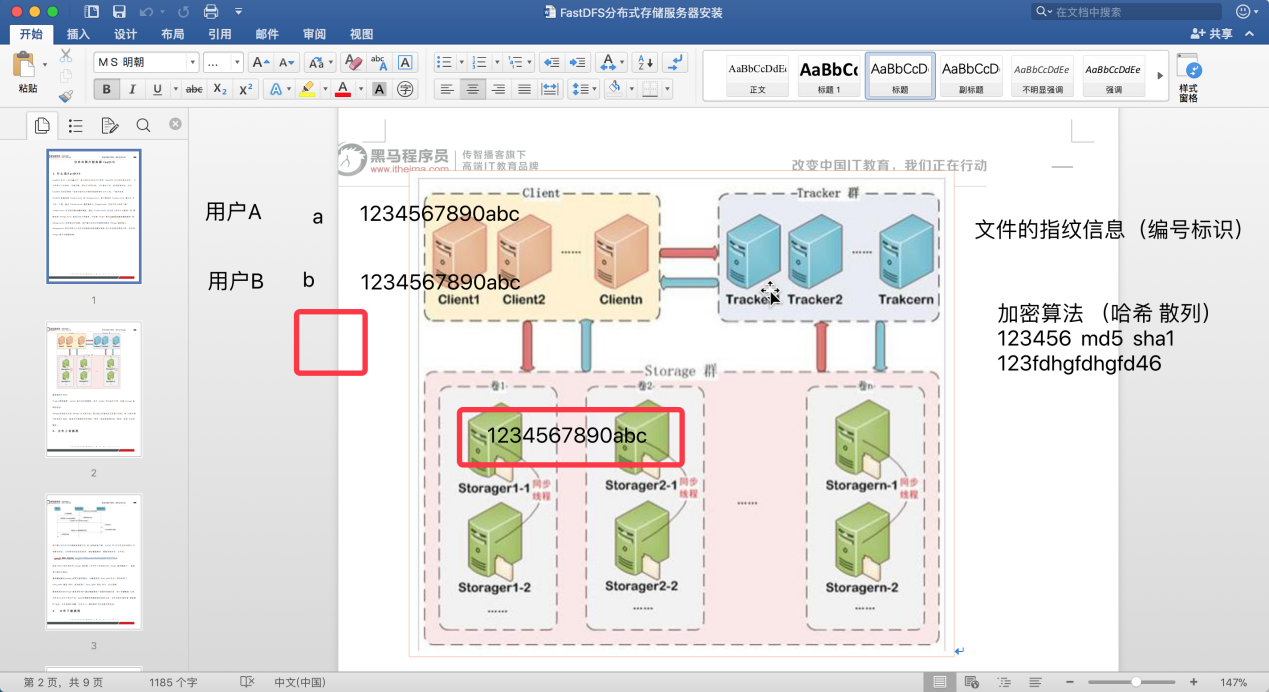
“user\_1\_history”： “1,2,3,4,5”



# 商品模块开发

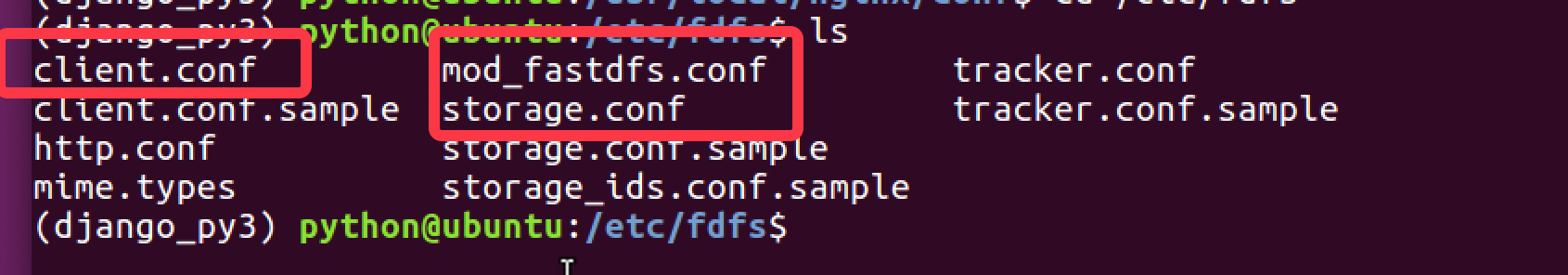
## 了解FastDFS分布式文件系统

集群



启动FastDFS的方法，需要的操作：

1. 修改如下的配置文件 （在/etc/fdfs目录中）



tracker\_server=自己的ip地址:22122

1. 启动tracker、storage、nginx服务：

sudo service fdfs\_trackerd start

sudo service fdfs\_storaged start

sudo /usr/local/nginx/sbin/nginx

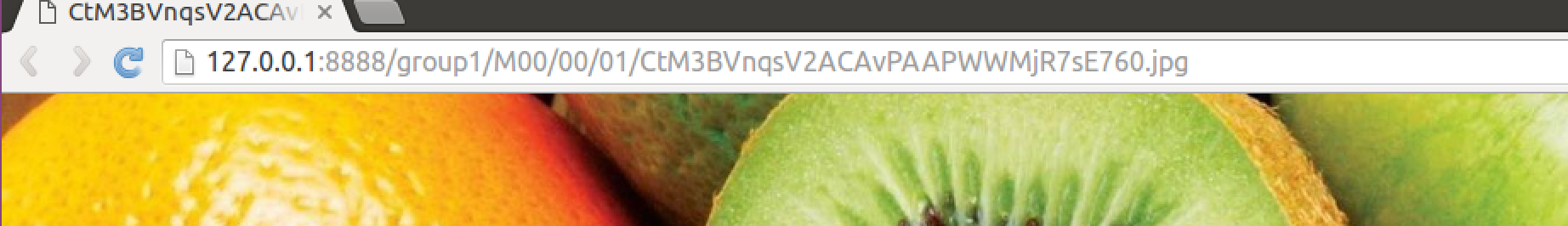
1. 执行如下命令测试是否成功

**fdfs\_upload\_file /etc/fdfs/client.conf 要上传的图片文件**

如果返回类似group1/M00/00/00/rBIK6VcaP0aARXXvAAHrUgHEviQ394.jpg的文件id则说明文件上传成功

在浏览器中可以用 127.0.0.1:8888/返回的文件id

访问图片



python对接fastdfs

1. workon django\_py3

2. 进入fdfs\_client-py-master.zip所在目录

3. pip install fdfs\_client-py-master.zip

>>> from fdfs\_client.client import Fdfs\_client

>>> client = Fdfs\_client('/etc/fdfs/client.conf')

>>> ret = client.upload\_by\_filename('test')

>>> ret

{'Group name':'group1','Status':'Upload successed.', 'Remote file\_id':'group1/M00/00/00/

wKjzh0\_xaR63RExnAAAaDqbNk5E1398.py','Uploaded size':'6.0KB','Local file name':'test'

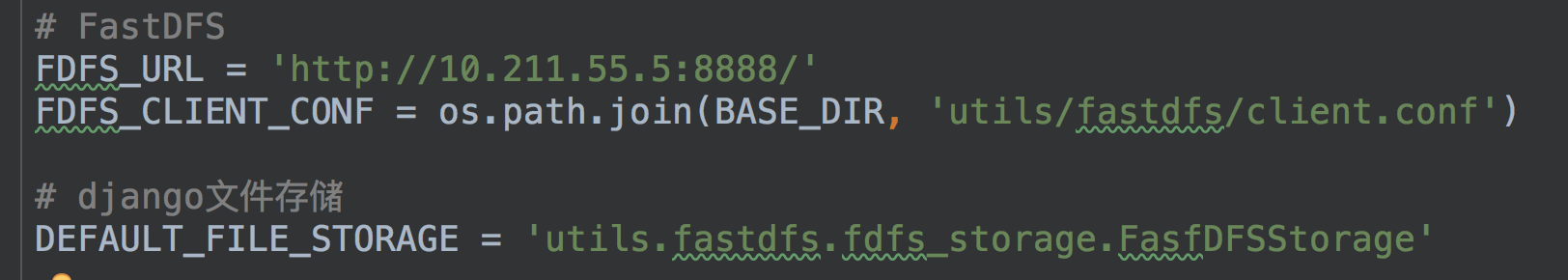
, 'Storage IP':'192.168.243.133'}

## Django二次开发对接FastDFS

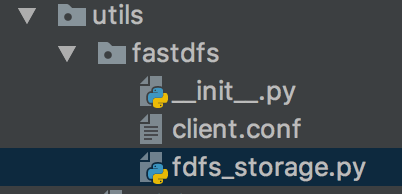
Django文档：

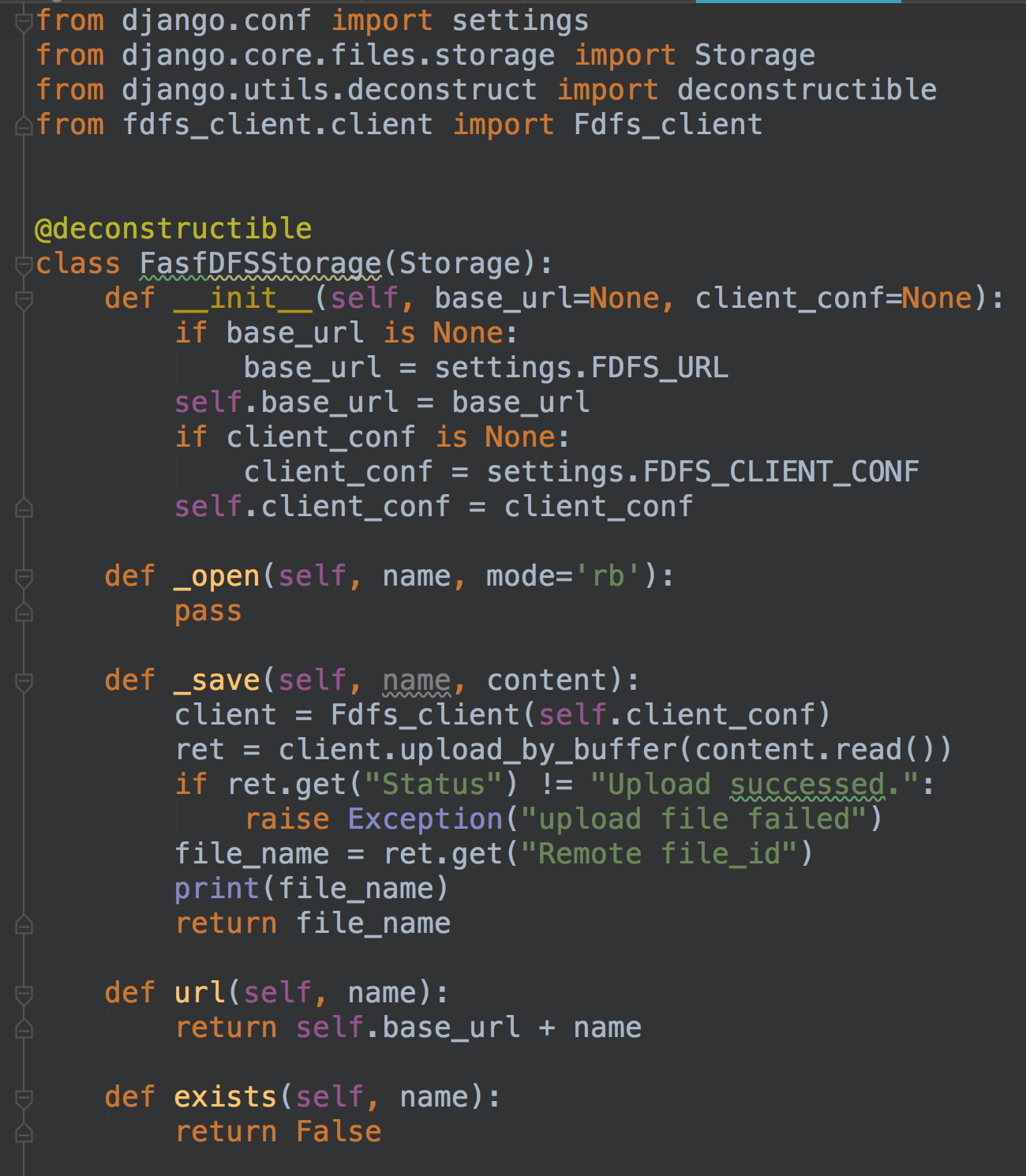
<http://python.usyiyi.cn/translate/django_182/ref/files/storage.html>

<http://python.usyiyi.cn/translate/django_182/howto/custom-file-storage.html>



创建utils/fastdfs 目录



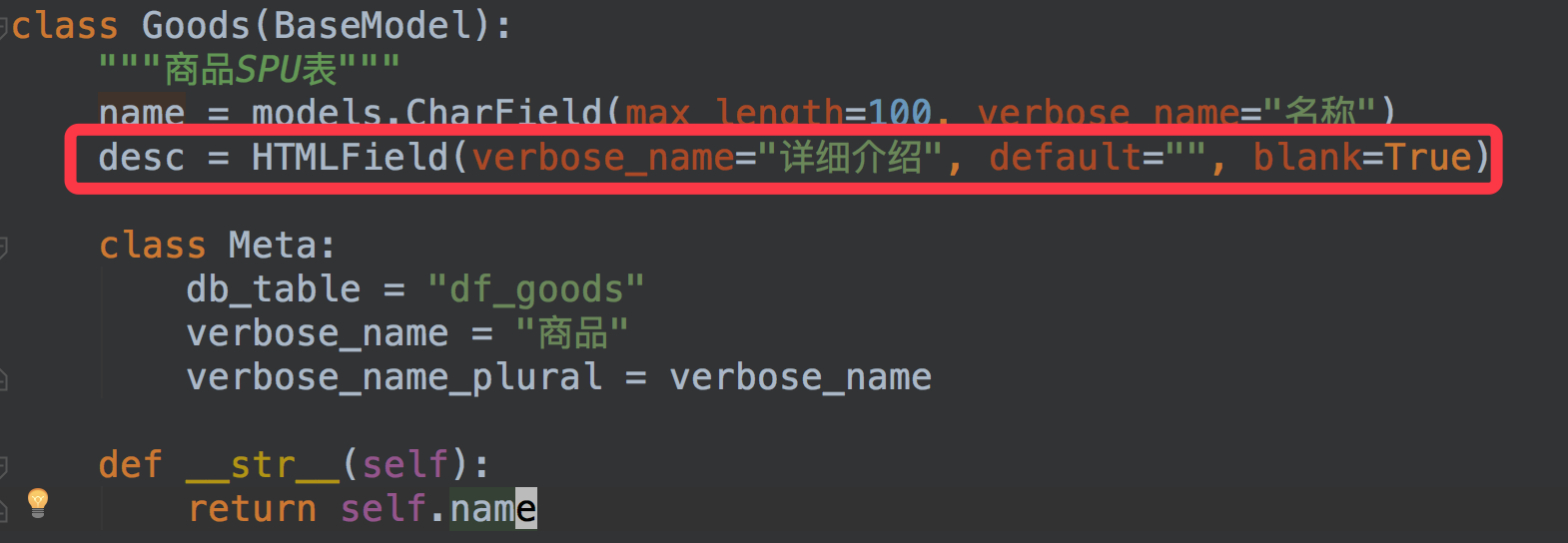


在goods/admin.py中注册商品相关模型，然后登陆admin站点进行测试



## 富文本编辑器

参考课件 为Goods模型添加详细信息字段



借助富文本编辑器，网站的编辑人员能够像使用offfice一样编写出漂亮的、所见即所得的页面。此处以tinymce为例，其它富文本编辑器的使用也是类似的。

在虚拟环境中安装包。

pip install django-tinymce==2.6.0

安装完成后，可以使用在Admin管理中，也可以自定义表单使用。

示例

1）在test6/settings.py中为INSTALLED\_APPS添加编辑器应用。

INSTALLED\_APPS = (

...

'tinymce',

)

2）在test6/settings.py中添加编辑器配置。

TINYMCE\_DEFAULT\_CONFIG = {

'theme': 'advanced',

'width': 600,

'height': 400,

}

3.在dailyfresh006/urls.py中配置编辑器url。

urlpatterns = [

...

url(r'^tinymce/', include('tinymce.urls')),

]

接下来介绍在Admin页面、自定义表单页面的使用方式。

## 商品首页

### 8.4.1 基本页面处理

需要查询哪些数据

填充到模板显示(模板语言)

### 8.4.2 保存购物车信息的数据设计

保存在redis中，每个人维护一条购物车数据， 选择哈希类型

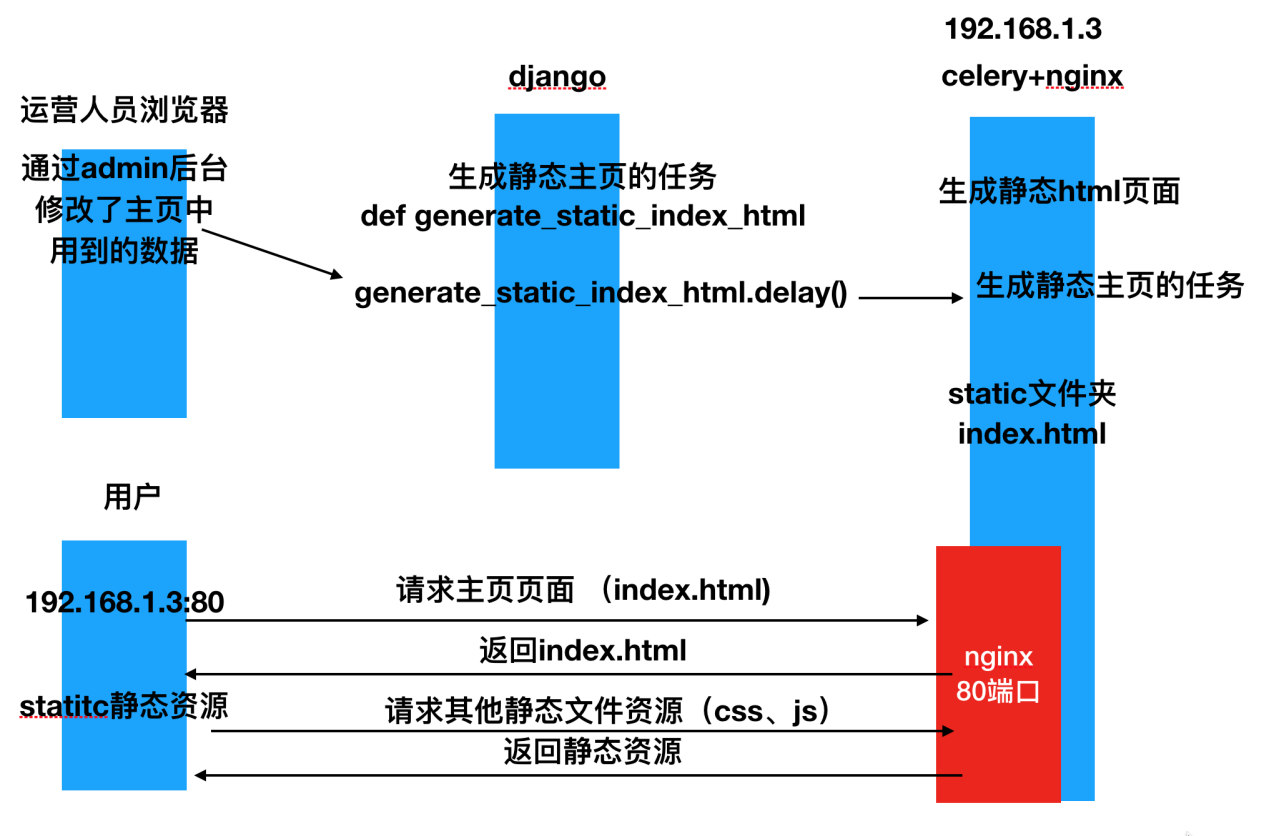
“cart\_用户id”: { “sku\_1”:“10”, ” sku\_2”: “11”}

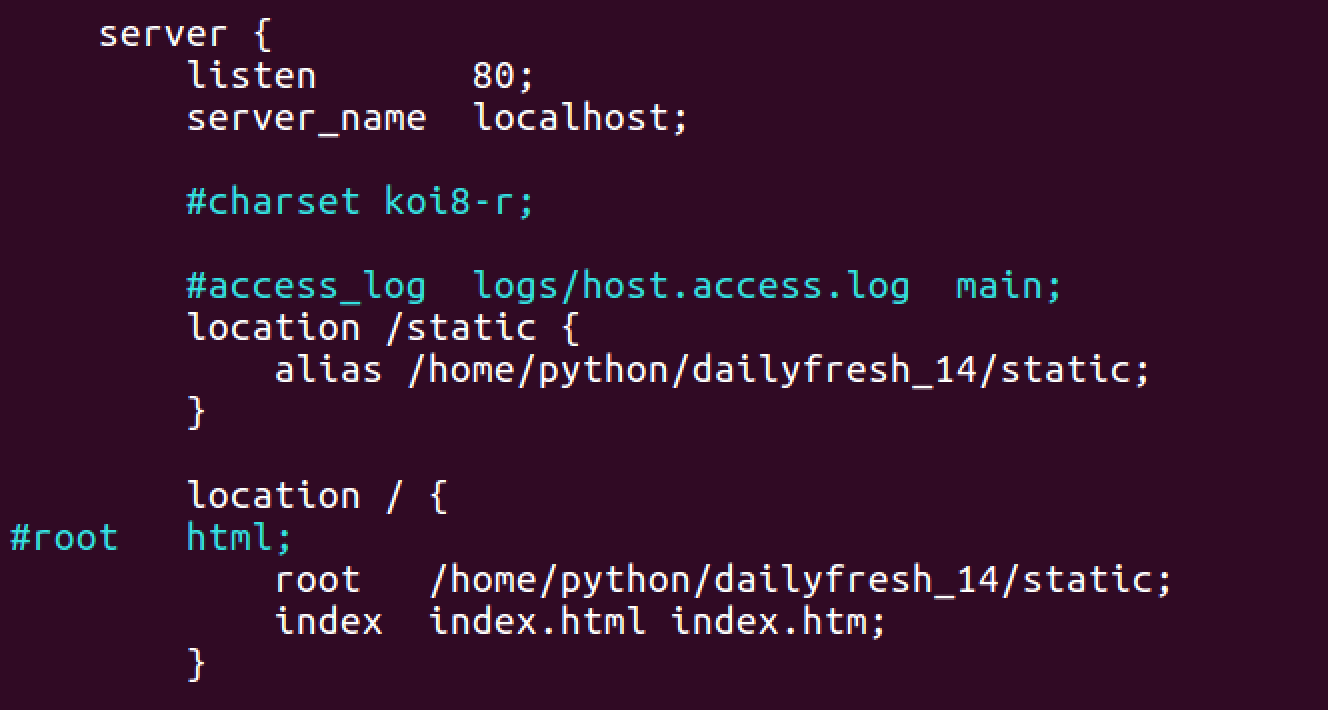
hlen() 是获取字典的数量大小, 但我们需要的是value值的总和 需要遍历

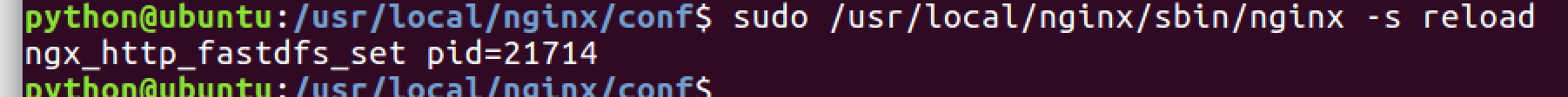


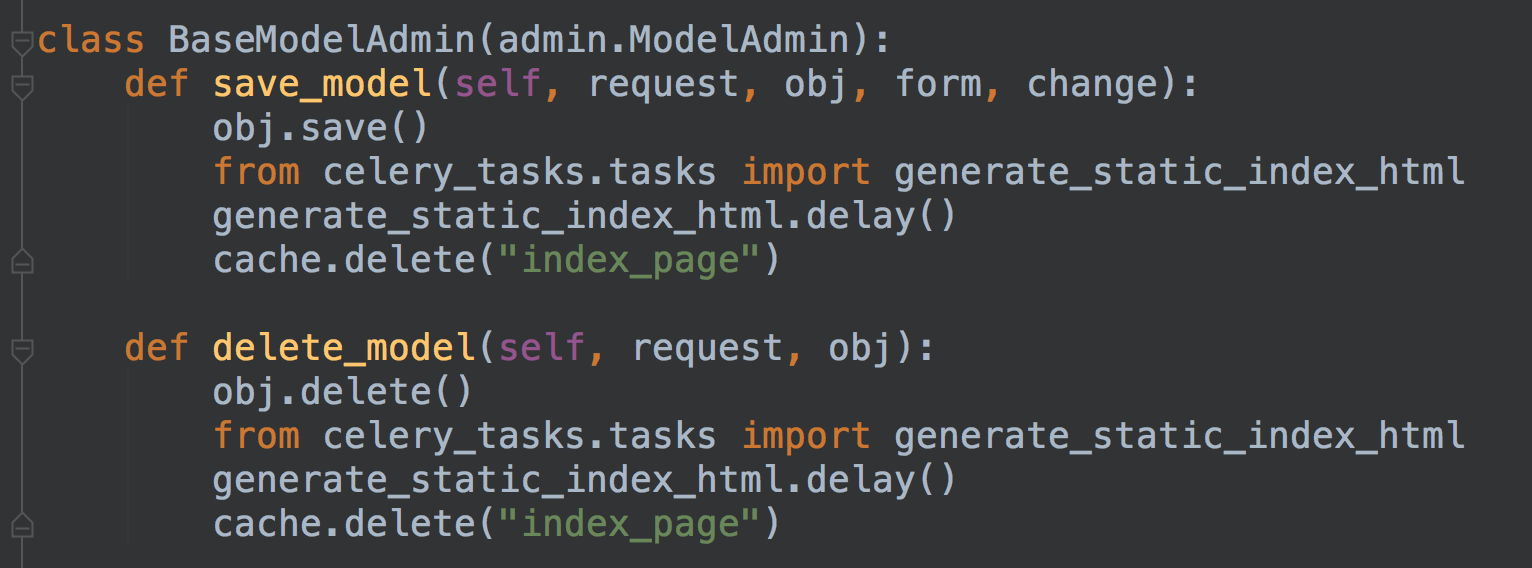
### 8.4.3 页面静态化

把原本动态的页面处理结果保存成html文件，让用户直接访问这个生成出来的静态的html页面









### 8.4.4 使用缓存

将处理计算的结果先临时保存起来，下次使用的时候可以先直接使用，如果没有这个备份的数据，重新进行计算处理

将缓存数据保存在内容中 （本项目中保存在redis中）

django提供了缓存的功能 见下面文档

cache

Pickle(序列化字符串) dumps loads

修改了数据库的数据，直接删除缓存

缓存要设置有效期

django文档：

<http://python.usyiyi.cn/translate/django_182/topics/cache.html>

## 商品详情页

Url 统一资源定位符UniformResourceLocator

detail/1

RESTFul

1 根据skuid 获取sku商品数据

2 获取分类信息

3 查询sku的订单 目的是用里面的评论信息 排序获取最近的30个

4 获取最新的当前类别的推荐商品 最新的两个

5 获取当前的sku对应的spu里的其它sku

6 数据缓存

7 获取购物车数量 cart\_num 和主页逻辑一样

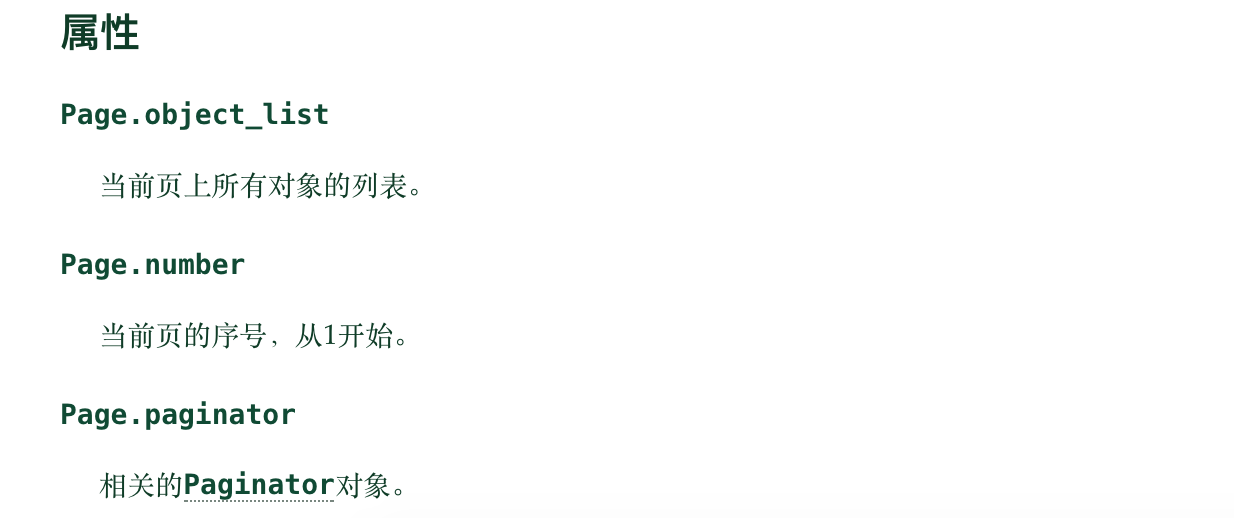
8 记录商品浏览历史记录

## 商品列表页

**分页**

<http://python.usyiyi.cn/documents/django_182/topics/pagination.html>







## 搜索页面

**草莓 GoodsSKU df\_sku**

**select \* from df\_sku where name like ‘%草莓%’ or title like ‘%草莓%’**

**草莓 500g 盒装草莓 大草莓**

**北京 草莓**

**搜索引擎**

1. **建立数据的索引表 草莓 sku= 1 2 3 基围虾 sku =100**
2. **进行分词操作**

**whoosh**

**搜索框架**

**haystack 搭建了用户和搜索引擎之间的沟通问题**

**配置：**

1. 安装python包。

pip install django-haystack

pip install whoosh

1. 在settings.py文件中注册应用haystack并做如下配置。





3）在goods应用目录下新建一个search\_indexes.py文件(文件名字固定)，在其中定义一个商品索引类。

from haystack import indexes

from goods.models import GoodsSKU

# 指定对于哪个模型类创建索引 类名是 模型类名字+Index

class GoodsSKUIndex(indexes.SearchIndex, indexes.Indexable):

# 索引字段 使用模板来指定那些数据字段创建索引

text = indexes.CharField(document=True, use\_template=True)

# 返回要创建索引的模型类

def get\_model(self):

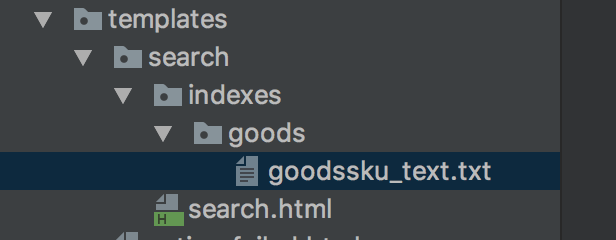
return GoodsSKU

# 返回可以查询的查询集

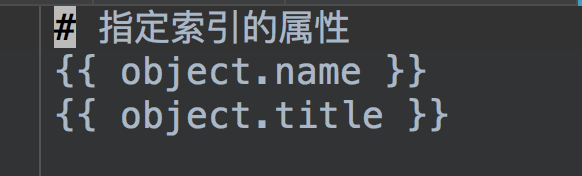
def index\_queryset(self, using=None):

return self.get\_model().objects.all()

4）在templates下面新建目录search/indexes/goods。



在此目录下面新建一个文件goodssku\_text.txt并编辑内容如下：

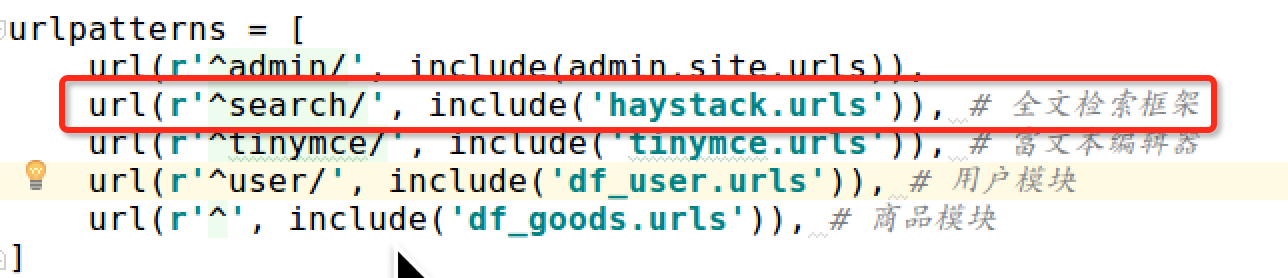


5）使用命令生成索引文件。

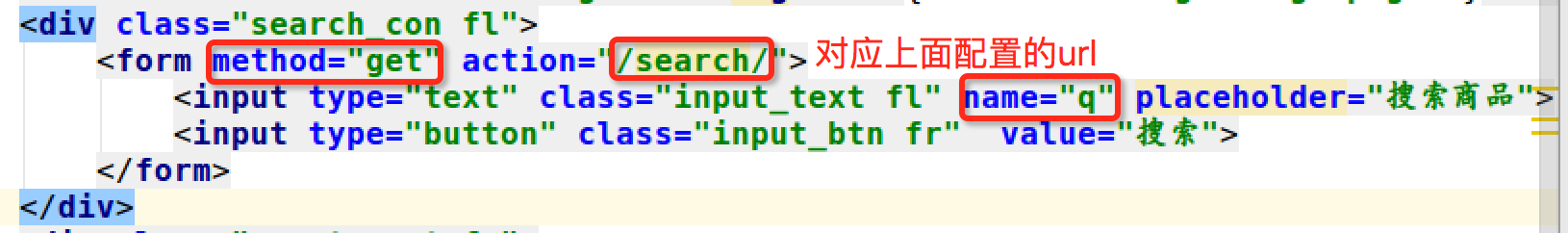
python manage.py rebuild\_index

**全文检索的使用：**

1）配置url。



2）表单搜索时设置表单内容如下。



点击标题进行提交时，会通过haystack搜索数据。

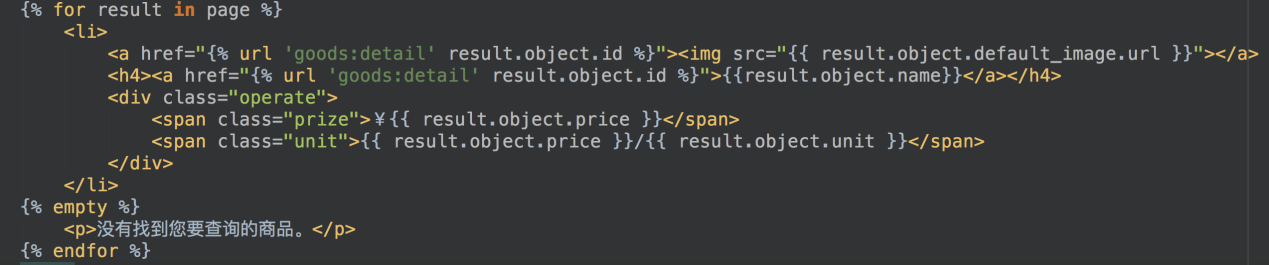
**全文检索结果：**

搜索出结果后，haystack会把搜索出的结果传递给templates/search目录下的search.html，传递的上下文包括：

* query：搜索关键字
* page：当前页的page对象
* paginator：分页paginator对象

search/?q=草莓&page=1

通过**HAYSTACK\_SEARCH\_RESULTS\_PER\_PAGE** 可以控制每页显示数量



**改变分词方式：**

1. 安装jieba分词模块。

pip install jieba

1. 找到虚拟环境py\_django下的haystack目录。

/home/python/.virtualenvs/bj10\_py3/lib/python3.5/site-packages/haystack/backends/

1. 在上面的目录中创建ChineseAnalyzer.py文件。

import jieba

from whoosh.analysis import Tokenizer, Token

class ChineseTokenizer(Tokenizer):

def \_\_call\_\_(self, value, positions=False, chars=False,

keeporiginal=False, removestops=True,

start\_pos=0, start\_char=0, mode='', \*\*kwargs):

t = Token(positions, chars, removestops=removestops, mode=mode, \*\*kwargs)

seglist = jieba.cut(value, cut\_all=True)

for w in seglist:

t.original = t.text = w

t.boost = 1.0

if positions:

t.pos = start\_pos + value.find(w)

if chars:

t.startchar = start\_char + value.find(w)

t.endchar = start\_char + value.find(w) + len(w)

yield t

def ChineseAnalyzer():

return ChineseTokenizer()

1. 复制whoosh\_backend.py文件，改为如下名称：

whoosh\_cn\_backend.py

1. 打开复制出来的新文件，引入中文分析类，内部采用jieba分词。

from .ChineseAnalyzer import ChineseAnalyzer

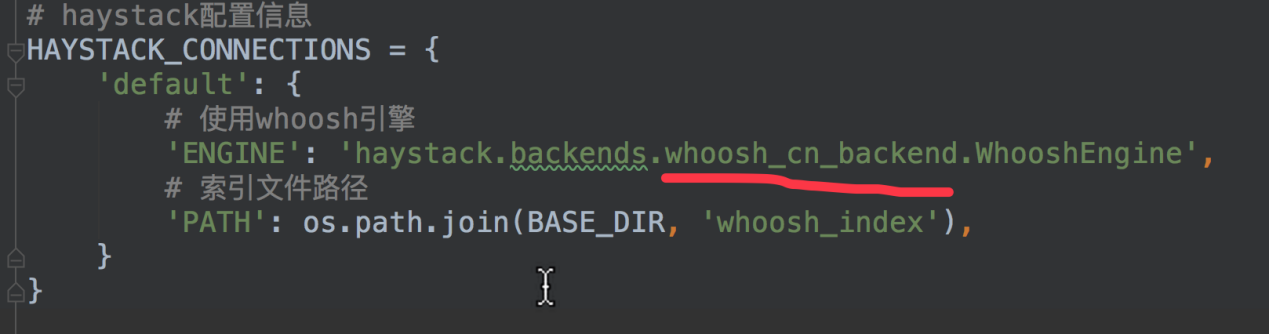
1. 更改词语分析类。

查找

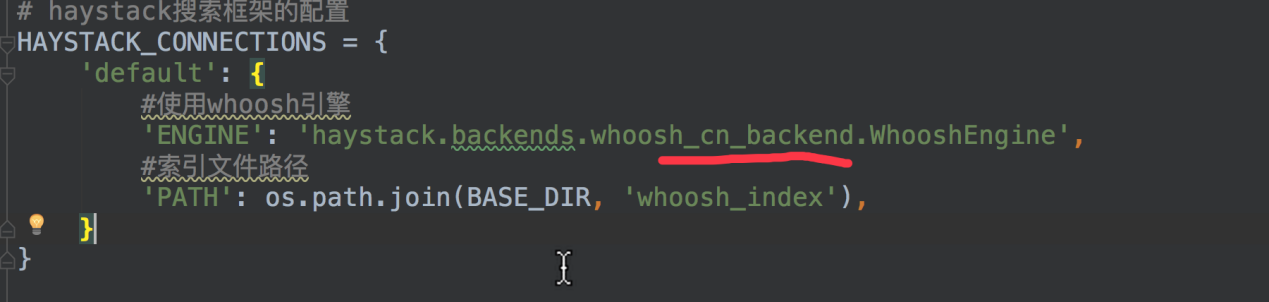
analyzer=StemmingAnalyzer()

改为

analyzer=ChineseAnalyzer()

1. 
2. 重新创建索引数据

python manage.py rebuild\_index



# 购物车模块

如果用户未登录，则将购物车数据保存到用户浏览器的cookie中

将购物车的数据转换为json字符串存在cookie

“cart”： ‘{ “sku\_1”:10, “sku\_2”: 20 }’

如果用户已登录，则将购物车数据保存到后端的redis中

在用户登陆的时候，将cookie中的购物车信息与redis中的购物车进行合并

json模块

将python中的字典转换为json字符串 json.dumps( python字典数据 )

将json字符串转换为python中的字典 json.loads( json字符串 )

## 添加到购物车

确定前端是否传递数据，传递什么数据，什么格式

确定前端访问的方式（get post）

确定返回给前端的什么数据，什么格式

## 购物车页面

decimal python中精度比较高的小数点数字

from decimal import Decimal

## 更新购物车

非幂等 /cart/add?sku\_id=1&num=2

幂等 /cart/update?sku\_id=1&finally\_num=18

对于同一种行为，如果执行不论多少次，最终的结果都是一致相同的，就称这种行为是幂等的

对于同一种行为，如果最终的结果与执行的次数有关，每次执行后结果都不相同，就称这种行为为非幂等

# 订单模块

## 确认订单页面（place order）

从商品详情页【立即购买】跳转过来----把商品存入购物车

从购物车页面【去结算】跳转过来

## 提交订单（下单）

python 的两个时间模块 datetime time

django中提供了一个支持时区的时间 django.utils.timezone

strftime 将时间类型转换为字符串类型

strptime 将字符串类型转换为时间类型

### 10.2.1 验证登录的装饰器（自定义，返回json数据）

utils/views.py中



### 10.2.2 事务的使用：

<http://python.usyiyi.cn/translate/django_182/topics/db/transactions.html>

from django.db import transaction

save\_id = transaction.savepoint() # 创建保存点

transaction.savepoint\_rollback(save\_id) # 回退（回滚）到保存点

transaction.savepoint\_commit(save\_id) # 提交保存点

### 10.2.3 并发访问控制

并发

悲观锁

SQL语句： select …. for update

Select \* from tble where id fro update

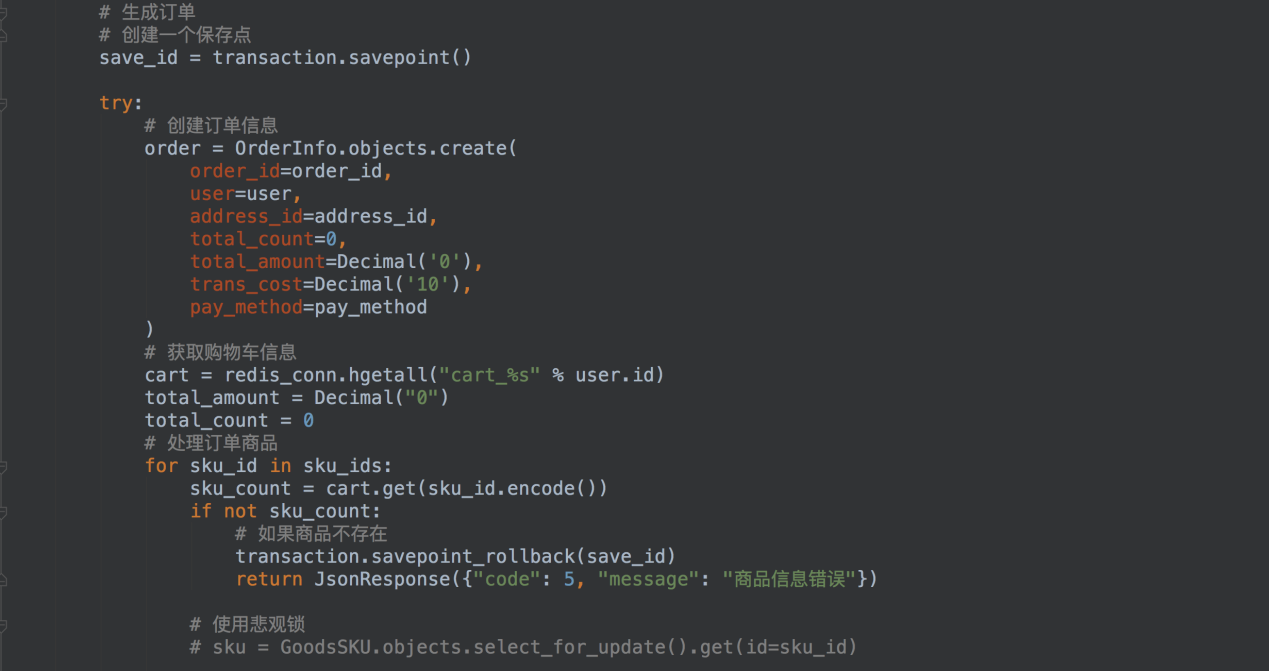


乐观锁

查询时不锁数据，提交更改时进行判断

队列



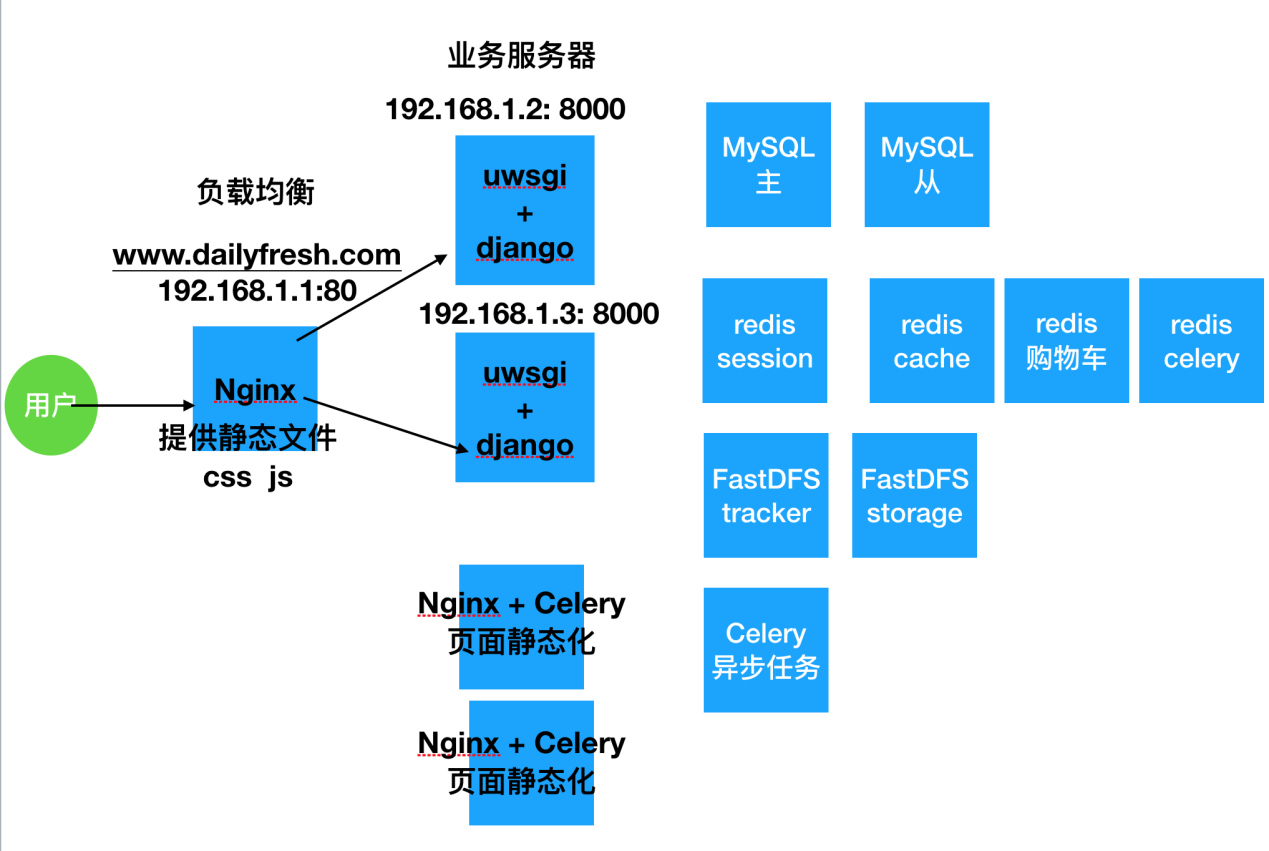
   

## 用户订单页面

## 订单支付

## 检查订单支付状态

## 订单评论



WSGI 并发

服务器 wsgi 框架部分（动态的业务逻辑处理程序）

总结

1. 生鲜类产品 B2C PC电脑端网页
2. 功能模块：用户模块 商品模块（首页、 搜索、商品） 购物车模块 订单模块（下单、 支付）
3. 用户模块：**注册、登录、激活、退出**、个人中心、地址
4. 商品模块：**首页**、详情、列表、**搜索**
5. 购物车： **增加**、删除、修改、**查询**
6. 订单模块：确认订单页面、**提交订单（下单）、请求支付、查询支付结果、**评论
7. django默认的认证系统 AbstractUser
8. itsdangerous 生成签名的token （序列化工具 dumps loads）
9. 邮件 （django提供邮件支持 配置参数 send\_mail）
10. celery (重点 整体认识 异步任务)
11. 页面静态化 （缓解压力 celery nginx）
12. 缓存（缓解压力， 保存的位置、有效期、与数据库的一致性问题）
13. FastDFS (分布式的图片存储服务， 修改了django的默认文件存储系统)
14. 搜索（ whoosh 索引 分词）
15. 购物车登录与未登录的结局方案 cookie redis 哈希
16. ajax 前端用ajax请求后端接口
17. 事务
18. 高并发的库存问题 （悲观锁、乐观锁、队列）
19. 支付的使用流程
20. nginx （负载均衡 提供静态文件）
21. 数据库主从 （读写分离）