# 项目介绍

* 1. 项目描述简介

通过完成一个仿京东购物平台，让大家掌握企业开发特点，以及解决问题的能力。

该项目会涉及以后工作中的一些非常有代表性的功能。

用户端采用前后端分离开发模式，前端通过调用后端API接口来获取和提交数据。掌握项目中前后端分离的开发模式。

我们使用git作为项目的版本控制软件。

在项目中会使用很多其他的知识，比如系统架构、维护、优化等等。

* 1. 主要功能模块

用户模块,商品模块,购物车模块,订单模块

* 1. 开发环境和技术

|  |  |
| --- | --- |
| 开发环境 | Window |
| 开发工具 |  |
| 相关技术 |  |

* 1. 项目人员组成周期成本
     1. 人员组成

|  |  |  |
| --- | --- | --- |
| 职位 | 人数 | 备注 |
| 项目经理和组长 | 1 | 一般小公司由项目经理负责管理，中大型公司项目由项目经理或组长负责管理 |
| 开发人员 | 2~3 |  |
| UI设计人员 | 1 |  |
| 前端开发人员 | 1 | 专业前端不是必须的，所以前端开发和UI设计人员可以同一个人 |
| 测试人员 | 1~3 | 有些公司并未有专门的测试人员，测试人员可能由开发人员完成测试。  公司有测试部，测试部负责所有项目的测试。  项目测试由产品经理进行业务测试。 |

* + 1. 项目周期成本

|  |  |  |
| --- | --- | --- |
| 人数 | 周期 | 备注 |
| 3 | 四个月  第一个月需求设计  第二三个月编码实现  第四个月编码实现、测试 |  |

1. 用户功能模块
   1. 用户账号注册模块
      1. 需求

基本信息:手机号,密码.确认手机号的格式,密码不能少于6位,验证第二次密码是否与第一次密码相同.完成后点击注册

* + 1. 流程

1.用户填写手机号和密码点击注册,数据提交到form验证层

2.form验证层验证手机号码的格式和长度是否正确,有无重复,密码长度是不是在规定 范围内.出错时提示手机号格式不对或已经被注册,密码最少要有6个字符.

3.如果填写没有问题那么就将填写好的数据保存到数据库,并跳转到登录页面

* + 1. 设计要点

1.数据库的基本字段:

**ID** 主键

phone 手机号码 CharField

user\_name 昵称 CharField null=True

Password 密码 CharField

Gender 性别 SmallIntegerField ((1,男),(2,女),(3,保密))

default=3

School 学校 CharField null=True

homtown 家乡 CharField null=True

is\_delete 删除 假删除 default=False

create\_time 添加时间 auto\_now\_add=True

update\_time 修改时间 auto\_now=True

2. 前端页面对数据的校验

(1)对文本框进行空字段校验，如果为空，点击注册按钮后会提示不能为空；

(2)对必填字段进行字数的校验，如果用户填写的字数少于或多余规定的字符长度， 提示长度不对；

(3)对手机号码进行正则判断，用户如果输入了格式不正确的手机号，提示用户输入 正确的手机号；

* + 1. 要点难点及解决方案

1.注册表单没有验证403

解决方法:验证跳转的页面写错

* 1. 用户账号登录模块
     1. 需求

1. 用户账号(手机号),密码,点击登录后数据提交
2. 2.对用户提交过来的数据进行校验，如果数据与数据库内的信息一致，登录成功，跳转到商城主页；
3. 用户填写的密码为明文密码，如何与哈希过的密码进行比较

解决方案：将用户填写的基本校验成功后的密码进行哈希化，将哈希后的密码与数据库内的密码 进行比较是否一致

* + 1. 流程

1.用户填写手机号和密码点击登录,数据提交到form验证层

2.form验证层验证手机号码的格式和长度是否正确,验证手机号码是否存在,如果不存在就提示需要注册.密码长度是否正确,若不正确提示密码错误

3.如果填写没有问题那么提示登录成功并跳转到首页

* + 1. 设计要点

1.数据库的基本字段:

**ID** 主键

phone 手机号码 CharField

user\_name 昵称 CharField null=True

Password 密码 CharField

Gender 性别 SmallIntegerField ((1,男),(2,女),(3,保密))default=3

School 学校 CharField null=True

homtown 家乡 CharField null=True

is\_delete 删除(假删除) default=False

create\_time 添加时间 auto\_now\_add=True

update\_time 修改时间 auto\_now=True

2. 交互设计

(1)对文本框进行空字段校验，如果为空，登录失败，提示用户填写基本信息；

(2)对填写的字段进行基本判断，判断用户输入的字符长度是否符合规范；

(3)为文本框添加placeorder属性，提示用户需要填写的信息。

* + 1. 要点难点及解决方案
       1. 难点:

1.当用户登录后，如何将用户的信息渲染到个人信息页面？

解决方案：使用session来获取用户的唯一信息，再使用唯一信息进行数据查询，将查询到 的数据渲染到页面上进行显示。

2.用户状态的保存

解决方案：用session将用户的唯一信息进行保存，当用户登录后，session会被传 递到浏览器，用户的信息也将被保存。

要点:

将session保存到redis数据库

1. 安装   
   pip install django-redis
2. 作为 cache backend 使用配置

CACHES = {

"default": {

"BACKEND": "django\_redis.cache.RedisCache",

"LOCATION": "redis://127.0.0.1:6379/1",

"OPTIONS": {

"CLIENT\_CLASS": "django\_redis.client.DefaultClient",

}

}

}

URL 格式举例

redis://[:password]@localhost:6379/0

rediss://[:password]@localhost:6379/0

unix://[:password]@/path/to/socket.sock?db=0

配置session的存储引擎

SESSION\_ENGINE = "django.contrib.sessions.backends.cache"

SESSION\_CACHE\_ALIAS = "default"

1）存储在数据库中，如下设置可以写，也可以不写，这是默认存储方式。

SESSION\_ENGINE='django.contrib.sessions.backends.db'

2）存储在缓存中：存储在本机内存中，如果丢失则不能找回，比数据库的方式读写更快。

SESSION\_ENGINE='django.contrib.sessions.backends.cache'

3）混合存储：优先从本机内存中存取，如果没有则从数据库中存取。

SESSION\_ENGINE='django.contrib.sessions.backends.cached\_db'

* 1. 用户个人信息修改模块
     1. 需求

1.用户登录商城；

2.用户的信息渲染到个人资料页面；

3.用户到个人资料页面修改个人资料页面内的信息；

4.点击确认按钮，保存成功。

* + 1. 流程

1.填写好基本信息后，用户点击修改按钮将数据进行提交；

2.Form验证层对提交过来的数据进行校验；

3.如果校验通过了，用户的数据将被修改成功。

* + 1. 设计要点

交互:

1.为文本框添加placeorder属性，提示用户需要填写的信息；

2.对填写的修改信息进行校验，数据是否合法，长度是否符合规范；

3.上传图片时判断用户是否有上传图片；

4.对必填字段进行空字段校验，如果为空提示用户输入必填项。

* + 1. 要点难点及解决方案
  1. 用户地址管理模块
     1. 需求

1.用户选择商品后，进入到确认订单模块；

2.用户点击地址，如果没有地址，用户添加地址；

3.用户添加地址后，点击提交，添加的信息提交到form层进行校验；

4.如果数据没有问题，数据添加成功；

5.如果用户有地址，点击地址进行选择，将地址设置为默认地址。

* + 1. 流程

1.用户添加地址

(1)用户点击添加地址，进入到添加 页面；

(2)填写收件地址等信息，填写完成后点击提交；

(3)form对提交信息进行校验，如果符合要求，提交数据到数据库，如果数据不符合要求，提示用 户添加地址失败，显示错误信息。

2.用户选择地址

(1)用户选择某地址为默认地址；

(2)该地址被修改为默认地址，之前的默认地址被修改为普通地址；

(3)默认地址被渲染到订单页面。

* + 1. 设计要点

1.数据库的基本字段:

Id 主键

Create\_time 添加时间 datetime

Update\_time 修改时间 datetime

Is\_delete 是否删除（假删除） Boolean

Name 收货人姓名 varchar

Telephone 电话号码 varchar

Hcity 省份 varchar

Hproper 城市 varchar

Harea 地区 varchar

Brief 详细地址 varchar

isDefault 是否是默认地址 Boolean

user\_id 所属用户 ForegnKey

2. 交互设计

(1)用户添加地址的时候，对填写的信息进行空字段校验，如果为空，显示响应的提示；

(2)对填写的信息进行字数的校验，如果少于或多于规定的字符数，显示相应的提示信息；

(3)如果用户选择或者添加了默认地址信息，之前的默认地址则被修改为普通地址。

* + 1. 要点难点及解决方案

难点：用户选择某个地址作为默认地址时，该地址的信息渲染到订单页面的地址信息处，其他的默认地址被修改为普通地址

1. 商品功能模块
   1. **商品的增删改查**
      1. 需求

基本信息:手机号,密码.确认手机号的格式,密码不能少于6位,验证第二次密码是否与第一次密码相同.完成后点击注册

* + 1. 流程

1.设计好数据库,并创建模块

2.将数据库内供商家控制的模型添加到admin控制模块中；

3.创建超级用户来管理后端数据；

4.通过后台账户进入到后台；

5.对数据进行增删查改。

* + 1. 设计要点

1.数据库的设计

商品分类表:GoodsClassifyModel

Class\_name 分类名 varchar

Class\_intro 分类简介 varchar

Is\_delete 是否删除 Boolean

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

商品单位表:GoodsUnitModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Unit\_name 单位名称 varchar

商品相册表:GoodsAlbumModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Album\_image 商品图片地址 ImageField

Album\_sku 商品SKUID ForeignKey

商品SPU表:SpuModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Spu\_name 商品SPU名称 varchar

Spu\_desc 商品spu描述 RichTextUploadingField

商品SKU表:GoodsSkuModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Goods\_name 商品信息 char

Goods\_intro 商品介绍 text

Price 商品价格 decimal

Unitinfo 商品单位 foreignkey

Num 库存 positiveinteger

Sell\_num 销售量 positiveinteger

Logo 图片 ImageField

Is\_putaway 上架 Boolean

Goodcate 商品分类 foreignkey

Goodsspu 商品spu分类 foreignkey

* + 1. 要点难点及解决方案

要点:

1. 以上5张表都有3个共同的参数可以将这三个参数提出来设计成一个基础的表,让后面的表可以继承.可以减少代码量

2. 设计基础表时要添加一个class Meta: abstract=True 设置为抽象类,不会被迁移

3. 富文本编辑器的使用

1.在您的python路径中安装或添加django-ckeditor

pip install django-ckeditor

2.将ckeditor添加到您的INSTALLED\_APPS设置中

INSTALLED\_APPS = ['ckeditor', # 添加ckeditor富文本编辑器]

3.配置静态文件路径

删除之前配置的STATICFILES\_DIRS（本来上线就要删除），修改为STATIC\_ROOT

1. #STATICFILES\_DIRS = [
2. # os.path.join(BASE\_DIR, "static"),
3. #]
4. # 设置静态文件根目录 上线的时候使用
5. STATIC\_ROOT = os.path.join(BASE\_DIR, "static")

使用以下命令，将django中安装在INSTALLED\_APPS的其他应用所有的静态文件收集到STATIC\_ROOT指定目录下

1. python.exe manage.py collectstatic

这个时候 我们在 static 目录下就能看的 ckeditor 目录，里面就是ckeditor使用到的所有的静态文件（css,js,image）。

这个时候又将之前的修改改回去，现在还没有上线，还得用之前的配置。

1. STATICFILES\_DIRS = [
2. os.path.join(BASE\_DIR, "static"),
3. ]
4. # 设置静态文件根目录 上线的时候使用
5. # STATIC\_ROOT = os.path.join(BASE\_DIR, "static")

4.设置ckeditor的上传目录   
这个目录是相对目录，相对与 MEDIA\_ROOT

1. CKEDITOR\_UPLOAD\_PATH = "uploads/"

将ckeditor\_uploader添加到您的INSTALLED\_APPS设置中。

1. INSTALLED\_APPS = [
2. 'django.contrib.admin',
3. 'django.contrib.auth',
4. 'django.contrib.contenttypes',
5. 'django.contrib.sessions',
6. 'django.contrib.messages',
7. 'django.contrib.staticfiles',
8. 'sp\_user.apps.SpUserConfig', # 用户模块
9. 'sp\_goods.apps.SpGoodsConfig', # 商品模块
10. 'ckeditor', # 添加ckeditor富文本编辑器
11. 'ckeditor\_uploader', # 添加ckeditor富文本编辑器文件上传部件
12. ]

5.在主路由中配置 ckeditor 上传文件使用到的url地址   
ckeditor上传文件内部机制使用 ajax 异步上传文件，需要配置上传访问地址才能被访问到。

1. urlpatterns = [
2. url(r'^admin/', admin.site.urls),
3. # 上传部件自动调用的上传地址
4. url(r'^ckeditor/', include("ckeditor\_uploader.urls")),
5. # 用户模块
6. url(r'^user/', include("sp\_user.urls", namespace="sp\_user")),
7. # 其他模块
8. ]

6.安装pillow(已经安装的不用重复安装)   
ckeditor上传文件使用pillow

7.设置ckeditor在后台显示的样式（显示效果）   
使用默认配置就行，具体看文档修改

1. # 编辑器样式配置
2. CKEDITOR\_CONFIGS = {
3. 'default': {
4. 'toolbar': 'full',
5. },
6. }

8.将商品SPU表的详情字段修改RichTextUploadingField，使用ckeditor为们单独提供的字段类型，修改完后，后台自动显示富文本编辑器样式，可以上传图片

1. # 导入ckeditor上富文本编辑器自带字段
2. from ckeditor\_uploader.fields import RichTextUploadingField
3. class GoodsSPU(BaseModel):
4. """
5. 商品SPU表
6. """
7. spu\_name = models.CharField(verbose\_name='商品SPU名称',
8. max\_length=20,
9. )
10. # 使用ckeditor为我们提供的字段，不用重新迁移就可以
11. content = RichTextUploadingField(verbose\_name="商品详情")
12. def \_\_str\_\_(self):
13. return self.spu\_name
14. class Meta:
15. verbose\_name = "商品SPU"
16. verbose\_name\_plural = verbose\_name

9.前台模板显示   
{{content|safe}}

排序: 1. 页面刚加载的时候 显示的商品只 显示 排序 排第一的分类下的商品

2. 点击哪个分类 就显示 对应分类下的商品

3. 可以按照 销量,价格(降,升),添加时间,综合(pk) 排序 并且 是对应分类下的商品

添加一个参数order:

0: 综合

1: 销量降

2: 价格升

3: 价格降

4: 添加时间降

order\_rule = ['pk', '-sale\_num', 'price', '-price', '-create\_time']

点击时价格排序的图片改变,html类:

.list-price{  
 background: url({% if order == 2 %}{% static 'images/order-bot.png' %}{% elif order == 3 %}{% static 'images/order-top.png' %}{% else %}{% static 'images/order.png' %}{% endif %}) no-repeat right center;  
 padding-right: 20px;  
 font-style: normal;  
}

JS控制价格升降的JS:

$(**function** () {  
 //修改 标签上的地址  
 {% if order == 2 %}  
 $("#price").attr('href', '{% url "com:超市" cate\_id 3 %}');  
 {% elif order == 3 %}  
 $("#price").attr('href', '{% url "com:超市" cate\_id 2 %}');  
 {% endif %}  
})

* 1. 首页模块
     1. 需求
        1. 渲染首页的banner部分；
        2. 查找出商品活动信息，渲染出活动以及该活动下的商品信息。
     2. 设计要点
        1. 渲染banner部分

查询出所有banner信息，将banner信息传递到首页上，首页遍历出所有的banner图片，渲染到首页的banner框内；

* + - 1. 渲染首页的活动信息

在视图函数内查询出所有的活动表；

遍历表内的信息，得到所有的活动；

通过活动信息，利用外键查询，得到所有的商品信息；

遍历商品信息，得到所有的商品。

* + 1. 要点难点及解决方案

1. 数据库

首页轮播商品表:BannerModel

Banner\_name 名称 char

Banner\_sku 商品sku信息 foreignkey

Banner\_image 图片地址 image

Order 排序 smallint

Create\_time 创建时间 datetime

Updatetime 更新时间 datetime

Is\_delete 是否删除 Boolean

商品活动表:ActivityModel

Title 名称 char

Activity\_image 商品图片 Image

url 商品地址 URL

Create\_time 创建时间 datetime

Updatetime 更新时间 datetime

Is\_delete 是否删除 Boolean

首页专区活动和商品:ActivityZoneModel

Create\_time 创建时间 datetime

Updatetime 更新时间 datetime

Is\_delete 是否删除 Boolean

Area 首页活动 foreignkey

Goods 商品 foreignkey

* 1. 商品分类模块
     1. 需求
        1. 得到所有的分类信息；
        2. 遍历分类信息，将所有分类显示到页面上；
        3. 通过分类查询出该分类下的商品；
        4. 将查询到的商品信息渲染到对应的分类目录下。
     2. 流程
        1. 查询出所有的分类信息；
        2. 在静态页面中遍历并渲染分类信息；
        3. 利用分类信息查询到该目录下的所有商品；

遍历出所有商品，将商品渲染到相应的分类下。

* + 1. 设计要点
       1. 数据库

商品分类表:GoodsCategoriesModel

Classname 分类名 varchar

Classintro 分类简介 varchar

Is\_delete 是否删除 Boolean

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

商品单位表:GoodsUnitModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Unitname 单位名称 varchar

商品SPU:SpuModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Name 商品SPU名称 varchar

Desc 商品spu描述 RichTextUploadingField

商品SKU表:GoodsSkuModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Goodsname 商品信息 char

Goodsintro 商品介绍 text

Price 商品价格 decimal

Unitinfo 商品单位 foreignkey

Num 库存 positiveinteger

Sellnum 销售量 positiveinteger

Logo 图片 ImageField

Is\_putaway 上架 Boolean

Goodcate 商品分类 foreignkey

Goodsspu 商品spu分类 foreignkey

2. 页面交互

如果分类下有商品信息，则显示出商品的信息，如果不存在商品，显示出没有商品的提示信息。

* + 1. 要点难点及解决方案
       1. 难点1：关于参数问题

解决方案：

为了方便排序，url地址都使用两个参数，一个是商品的分类的id信息，一个是排序的序号信息；

难点2： 点击每个分类时，如何查询渲染才能展示出当前分类下的商品;

解决方案：

查询出分类下所有商品信息，用商品的主键pk来区分不同的商品信息。

全文检索:

1.在虚拟环境中依次安装需要的包。

1. pip install django-haystack
2. pip install whoosh
3. pip install jieba

2.在配置文件settings.py中添加应用 全文搜索框架haystack   
添加到所有自己安装应用的前面

1. INSTALLED\_APPS = [
2. 'django.contrib.admin',
3. 'django.contrib.auth',
4. 'django.contrib.contenttypes',
5. 'django.contrib.sessions',
6. 'django.contrib.messages',
7. 'django.contrib.staticfiles',
8. 'haystack', # 全文检索框架
9. 'sp\_user.apps.SpUserConfig', # 用户模块
10. 'sp\_goods.apps.SpGoodsConfig', # 商品模块
11. 'ckeditor', # 添加ckeditor富文本编辑器
12. 'ckeditor\_uploader', # 添加ckeditor富文本编辑器上传部件
13. ]

3.配置全文检索框架haystack支持whoosh搜索引擎

1. # 全文检索框架的配置
2. HAYSTACK\_CONNECTIONS = {
3. 'default': {
4. # 配置搜索引擎
5. 'ENGINE': 'haystack.backends.whoosh\_backend.WhooshEngine',
6. # 配置索引文件目录
7. 'PATH': os.path.join(BASE\_DIR, 'whoosh\_index'),
8. },
9. }
10. #当添加、修改、删除数据时，自动生成索引
11. HAYSTACK\_SIGNAL\_PROCESSOR = 'haystack.signals.RealtimeSignalProcessor'

4.在应用目录（goods）下，创建一个文件search\_indexes.py文件   
在该文件中定义一个索引类，全文检索框架根据该类生成索引

1. # 导入全文检索框架索引类
2. from haystack import indexes
3. from sp\_goods.models import GoodsSKU
4. class GoodsSKUSearchIndex(indexes.SearchIndex, indexes.Indexable):
5. # 设置需要检索的主要字段内容 use\_template表示字段内容在模板中
6. text = indexes.CharField(document=True, use\_template=True)
7. # 获取检索对应对的模型
8. def get\_model(self):
9. return GoodsSKU
10. # 设置检索需要使用的查询集
11. def index\_queryset(self, using=None):
12. """Used when the entire index for model is updated."""
13. return self.get\_model().objects.all()

5.创建模板文件，4 中检索的字段就定义在模板中   
创建文件：templates/search/indexes/应用名/模型名小写\_text.txt   
例如：templates/search/indexes/sp\_goods/goodssku\_text.txt

在文件中定义需要检索的字段,object就代表get\_model()方法返回的对象

1. # 指定搜索的字段
2. {{object.sku\_name}}
3. {{object.brief}}
4. {{object.goods\_spu.content}}

6.使用命令生成索引文件

1. python manage.py rebuild\_index

7.在主路由中配置全文检索框架使用的子路由

1. urlpatterns = [
2. url(r'^admin/', admin.site.urls),
3. #全文搜索框架
4. url(r'^search/', include('haystack.urls')),
5. # 添加自己的应用的子路由
6. ]

8.创建搜索表单   
创建一个form表单，用于搜索，关键点在于:   
-- 搜索引擎默认接收一个GET方式传递q参数作为搜索关键字   
-- action 请求地址就是在主路由配置的地址 /search/

1. <form action="/search/" method="get">
2. <input type="text" name="q" value="" class="cate-input" placeholder="请输入您要的搜索的产品关键词"/>
3. <input type="submit" class="cate-btn" value=""/>
4. </form>

点击进行提交时，会通过haystack搜索数据

9.搜索的结果处理   
搜索结果会自动传递给 templates/search/search.html模板页面，并且传递以下关键变量   
-- query 搜索的关键字   
-- page 当前页检索结果集----> 遍历后是一个一个的SearchResult对象--->SearchResult.object才是检索的模型（商品）对象   
-- paginator： 分页paginator对象（自己研究）

1. {% if query %}
2. <ul class="list-pro" style="margin-top: 20px">
3. {% for result in page %}
4. <li>
5. <a href="{% url 'sp\_goods:detail' result.object.pk %}"><img
6. src="{{ MEDIA\_URL }}{{ result.object.logo }}" class="list-pic1"/></a>
7. <div class="shop-list-mid" style="width: 65%;">
8. <div class="tit">
9. <a href="{% url 'sp\_goods:detail' result.object.pk %}">
10. {{ result.object.goods\_spu.spu\_name }}{{ result.object.sku\_name }}
11. </a>
12. </div>
13. <div class="am-gallery-desc">￥{{ result.object.price }}</div>
14. <p>销量：{{ result.object.sale\_num }}件</p>
15. </div>
16. </li>
17. {% empty %}
18. <li>没有找到您搜索的产品！</li>
19. {% endfor %}
20. </ul>
21. {% endif %}

10.whoosh自带的分词对中文不友好，使用免费的中文分词jieba   
-- 打开虚拟环境/Lib/site-packages/haystack/backends/   
-- 在上面的目录下创建ChineseAnalyzer.py文件   
-- 复制 whoosh\_backend.py 改名为 whoosh\_cn\_backend.py   
-- 打开复制出来的新文件，引入中文分析类，内部采用jieba分词。

1. from jieba.analyse import ChineseAnalyzer

-- 更改词语分析类

1. 查找
2. analyzer=StemmingAnalyzer()
3. 改为
4. analyzer=ChineseAnalyzer()

-- 修改settings.py文件中的配置项

1. # 全文检索框架的配置
2. HAYSTACK\_CONNECTIONS = {
3. 'default': {
4. # 配置搜索引擎
5. # 'ENGINE': 'haystack.backends.whoosh\_backend.WhooshEngine',
6. # 中文分词 使用jieba的whoosh引擎
7. 'ENGINE': 'haystack.backends.whoosh\_cn\_backend.WhooshEngine',
8. # 配置索引文件目录
9. 'PATH': os.path.join(BASE\_DIR, 'whoosh\_index'),
10. },
11. }

-- 重新创建索引数据

1. python manage.py rebuild\_index
   1. 商品排序模块
      1. 需求
         1. 进入超市页面,点击不同方式的排序按钮能够使商品信息按照相应的规律进行排序;;
         2. 点击综合排序按照综合信息进行排序;
         3. 点击销量,商品以销量进行降序排序;
         4. 点击价格:

价格按升序排序

按价格降序排序

* + - 1. 点击新品,按照商品发布时间进行升序排序。
    1. 流程
       1. 需要实现的效果：当点击某一项方式进行排序时，所有的商品按照该方式进行排序；
       2. 视图函数中传递两个参数，用来方便排序以及显示分类下的商品信息；第一个参数是分类的id,第二个参数是排序方式的id.
       3. 先用第一个参数作为主键来查询商品分类，通过分类得到分类下的所有商品；
       4. 设置一个排序列表，列表中的字段是用于排序的字段（来自model中）；
       5. 商品排序，将排序列表作为参数传递给排序方法，将第二个参数order传递给作为参数传递给排序列表。
       6. 在前端渲染页面上，将排序的参数传递到相应的位置，进行排序。
    2. 设计要点（数据库和页面交互）
       1. 数据库

商品分类表:GoodsCategoriesModel

Classname 分类名 varchar

Classintro 分类简介 varchar

Is\_delete 是否删除 Boolean

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

商品SKU表:GoodsSku

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Goodsname 商品信息 char

Goodsintro 商品介绍 text

Price 商品价格 decimal

Unitinfo 商品单位 foreignkey

Num 库存 positiveinteger

Sellnum 销售量 positiveinteger

Logo 图片 ImageField

Is\_putaway 上架 Boolean

Goodcate 商品分类 foreignkey

Goodsspu 商品spu分类 foreignkey

* + 1. 要点难点及解决方案
       1. 难点1： 对价格进行排序

价格既可以升序排列，又可以降序排列，两个排序方式都需要提供给用户，如何才能实现在用户点击升序或者降序排列的按钮时能实现相应的排序。

解决方案：

利用ajax进行判断；默认的排序方式是升序，为排序添加一个类，用该类来作为判断标识；当再次点击时，判断排序的序号为降序或升序，如果为降序，则将他的排序方式修改为升序，反之，则修改为升序。

1. 购物车功能模块
   1. 购物车商品添加
      1. 需求

在商品详情页面，点击商品下的+号，选择加入购物车的数量， 点击加入购物车，将该数量的商品加入购物车。

在商品分两类页面，点击商品信息下的+号，将商品加入购物车，点击一次加一个商品。

* + 1. 流程

1. 前端通过ajax post请求方式，添加sku\_id（sku商品id）和count（商品的数量）  
3. 验证是否登录，如果没有登录，高数ajax中的js代码实现跳转到登录页面location.href=url  
4. 验证数据的合法性，必须都为整数  
5. 验证山沟是否存在  
6. 验证库存是否足够  
7. 保存到redis

* + 1. 设计要点（数据库和页面交互）
       1. 数据库设计：

使用hash对象 保存

hset key 属性 值

hset cart\_user\_id 商品id 数量

key: user\_telephone

filed: sku\_id

value sku\_count

* + - 1. 页面交互

用ajax进行数据交互以及页面跳转。

* + 1. 要点难点及解决方案

难点1：用Ajax实现购物车数据的添加；

解决方案：查文档

难点2：用redis数据库保存购物车内的信息；

解决方案：使用redis的对象方法进行保存，该方法有三个参数，正好解决数据的存储问题，第一个值存储用户的唯一信息，电话或者id; 第二个值保存商品的sku信息，作为商品的唯一标识； 第三个值作为商品的数量。

* 1. 购物车商品删除
     1. 需求

在商品详情页面，添加商品到购物车的时候，可以点击-减去刚刚添加到购物车内的商品；

在购物车页面，点击商品下的-号可以删除购物车内的该商品信息，点击一次删除一个商品。

* + 1. 流程
       1. 前端通过ajax post请求方式，添加sku\_id（sku商品id）和count（商品的数量）  
          3. 验证是否登录，如果没有登录，高数ajax中的js代码实现跳转到登录页面location.href=url  
          4. 验证请求删除的数据是否存在；

5. 当商品存在时，判断商品的数量是否大于1；

6. 如果商品数量大于1时，删除数据。

* + 1. 设计要点（数据库和页面交互）
       1. 数据库设计：

使用hash对象 保存

hset key 属性 值

hset cart\_user\_id 商品id 数量

key: user\_telephone

filed: sku\_id

value sku\_count

* + - 1. 页面交互

用ajax进行数据交互以及页面跳转。

* + 1. 要点难点及解决方案

难点1：用Ajax实现购物车数据的添加；

解决方案：查文档

难点2：用redis数据库保存购物车内的信息；

解决方案：使用redis的对象方法进行保存，该方法有三个参数，正好解决数据的存储问题，第一个值存储用户的唯一信息，电话或者id; 第二个值保存商品的sku信息，作为商品的唯一标识； 第三个值作为商品的数量。

* 1. 购物车商品显示
     1. 需求

1.将商品加入购物车；

2. 进入购物车页面，显示出加入购物车的商品；

* + 1. 流程
       1. 从数据库中取出sku\_id, count；
       2. 从数据库取出完整的商品信息；
       3. 计算总金额和总商品数量；  
          4. 在购物车页面渲染出商品的信息， 总价， 总数量。
    2. 设计要点（数据库和页面交互）
       1. 数据库设计：

使用redis的对象进行保存数据

key: user\_telephone

filed: sku\_id

value sku\_count

* + - 1. 页面交互

用ajax进行数据交互以及页面跳转。

* + 1. 要点难点及解决方案

难点1：勾选上全选，所有的商品信息被选中，显示所有商品的价格的综合；

解决方案：利用ajax判断商品是否全选，如果全选就调用视图函数中计算得到的总金额。

难点2：选中不同的商品，如何精确的计算出价格；

解决方案：在商品列表中，添加一个类来监测被选中的商品信息，用ajax得到商品的选中状态，用find方法查找出选中状态商品的价格，将价格进行累加，将结果用toFixed过滤一下得到商品价格精确结果。

django中redis使用:

1. 安装 django-redis(session保存到redis中已经安装)
2. pip install django-redis
3. 配置redis数据库连接(已经配置的不用再配置)   
   只要在django配置好数据库的链接信息，django就可以自动帮我们连接到redis数据库。
4. # 缓存配置
5. CACHES = {
6. "default": {
7. "BACKEND": "django\_redis.cache.RedisCache",
8. "LOCATION": "redis://127.0.0.1:6379/1",
9. "OPTIONS": {
10. "CLIENT\_CLASS": "django\_redis.client.DefaultClient",
11. }
12. }
13. }
14. 获取redis连接
15. # 导入redis连接方法
16. from django\_redis import get\_redis\_connection
17. # 使用默认配置连接到redis
18. cnn = get\_redis\_connection('default')
19. # 使用连接上的方法操作redis
20. cnn.hset('对象名'，'属性' ，'值')