# 项目介绍

* 1. 项目描述简介

通过完成一个仿京东的电子商城项目，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。

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. 创建项目

2.1. 问题

创建项目之后,项目在目录下不见了,但是能打开

重新删除建立了一次,能看见了

2.2. 渲染页面

渲染页面时,静态文件太多了,包括每个页面的图片等的加载,如何才能更加快速地添加静态资源?

2.3 创建父模板

开始时,不知道创建多少个,在早上讲了之后会创建

* 1. 用户账号注册功能模块
     1. 需求

1. 用户填写基本信息：正确的手机号，6个以上的密码，和密码相同的确认密码，填写完毕后点击获取验证码；

2. 获取验证码以后，填写验证码，点击立即注册。

* + 1. 流程

创建用户表

**class Users(**models.Model**):** gender\_choices **= (  
 (**1,**'男')**,  
 **(**2,**'女')**,  
 **(**3,**'保密')**,  
 **)** phoneNum **=** models.CharField**(**max\_length**=**11,  
 verbose\_name**='手机号'**,  
 validators**=[** RegexValidator**(r'^1[3-9]\d{9}$'**,**'手机号码格式错误')  
 ])** nickName **=** models.CharField**(**max\_length**=**16,null**=True**,  
 verbose\_name**='昵称'**,  
 validators**=[** MinLengthValidator**(**2,**'昵称至少为2个字符')  
 ])** password **=** models.CharField**(**max\_length**=**32**)** gender **=** models.SmallIntegerField**(**choices**=**gender\_choices,default**=**3**)** school **=** models.CharField**(**max\_length**=**200,null**=True**,verbose\_name**="学校名称")** home\_address **=** models.CharField**(**max\_length**=**200,null**=True**,verbose\_name**="老家地址")** detail\_address **=** models.CharField**(**max\_length**=**200,null**=True**,verbose\_name**='现住地址')** birthday **=** models.DateField**(**null**=True**,verbose\_name**='生日日期')** add\_time **=** models.DateTimeField**(**auto\_now\_add**=True**,verbose\_name**='添加时间')** change\_time **=** models.DateTimeField**(**auto\_now**=True**,verbose\_name**='更新时间')** is\_delete **=** models.BooleanField**(**default**=False**,verbose\_name**='是否删除')** head **=** models.ImageField**(**upload\_to**='head/%Y%m'**,default**='head/hlwtx.jpg'**,verbose\_name**='用户头像')  
  
 def \_\_str\_\_(**self**):  
 return** self.phoneNum  
 **class Meta:** db\_table **= 'users'**

1. 用户填写基本信息，填写完毕后数据点击获取验证码，如果手机号码格式没有问题，则向阿里云发起请求验证码。

2. 阿里云响应并返回验证码，用户填写获得的验证码后点击立即注册，数据被丢到了form验证层；

3. form验证数据是否合法，手机号码是否重复，如果密码长度不够，手机号码格式不正确或手机号码已被注册，则返回错误信息，提示用户注册失败；

4.如果用户填写的信息没有问题，则注册成功，跳转到用户登录界面。

5.手机验证还未做,先使用原始的注册方法:

(1)在用户users子应用下的models.py中建立Users模型,以建立表存储用户注册时的手机号和密码信息，验证手机号时加入正则方式验证

(2)在视图的Register类中进行操作,验证参数合法性

(3)为验证合法性,在users子应用下创建forms.py建立验证信息注册表单类模型,在模型类中验证账号和密码的合法性:

A.密码的位数及不能为空

B.手机号是否存在,手机号的位数限制,手机号不能为空

C.密码和确认密码不能相同

6.登录:

(1)在视图的Login类中进行操作,验证参数合法性

(2)在forms.py建立验证登录表单类模型,在模型类中验证登录信息合法性:

A.验证手机号不能填空,手机号的位数,手机号是否已经注册

B.验证密码位数,密码不能为空,以及手机号存在的情况下密码不正确

7.个人资料设置:

(1)最开始用的函数,决定修改为类,解决问题方便一点

(2)在类中定义get和post方法:

A.get解决用户点击进入界面,后台通过request调用用户session信息得到用户id,可以查询出用户的所有信息,然后回显在界面的表单所有信息

(3)在个人资料增加提交按钮,用户修改资料后点提交可以直接修改数据库中数据,然后回显当前界面

8.忘记密码:

(1)通过session获得用户ID,然后通过ID回显手机号在form表单中

(2)修改密码跟个人资料的修改差不多,只是需要在forms.py中重新建立验证类,需要验证用户是否已经修改了密码,还有密码的合法性

9.短信验证

# 登录验证装饰器使用

用户访问用户个人中心，有两种方式，第一种方式就是登陆后访问，第二种就是用户可以能直接拷贝链接直接访问。   
显然第二种方式就不允许用户访问的，所有我们应该做的就是无论用户以哪种方式访问个人中心（需要验证登录后才能访问的页面）都要验证sessin中是否有等表示（例如，用户ID,用户名。。。）

例如：

1. # 个人中心视图函数
2. def center(request):
3. # 如果session中没有用户ID,说明没有登录，就跳转到登录页面
4. if request.session.get("ID") is None:
5. return redirect("/user/login/")
6. return render(request, "sp\_user/member.html")

但是我们的网站有很多的页面都有可能需要验证用户是否登录，也就是很多的视图函数都要加上以上类似的验证代码，这样太麻烦了，所以我们可以借助于装饰器来帮我们完成。

### 创建装饰器

1.在配置文件settings中 添加一个配置，用于代表登录的地址

1. # 配置登录URL地址
2. LOGIN\_URL = "/user/login/"

2.在应用下创建一个helper.py,在该文件中创建装饰器函数

1. # 登录验证装饰器
2. def verify\_login\_required(func):
3. """
4. :param func: 传入的函数
5. :return:
6. """ # 登陆验证器
7. def verify(request, \*args, \*\*kwargs):
8. # 判断session中是否有ID,如果没有说明没有登录，请登录
9. if request.session.get("ID") is None:
10. # 配置文件中获取登录的URL地址
11. login\_url = settings.LOGIN\_URL
12. return redirect(login\_url)
13. else:
14. # 返回被调用函数
15. return func(request, \*args, \*\*kwargs)
16. return verify

### 视图函数装饰器使用

1. # 个人中心视图函数
2. @verify\_login\_required
3. def center(request):
4. # 如果session中没有用户ID,说明没有登录，就跳转到登录页面
5. # if request.session.get("ID") is None:
6. # return redirect("/user/login/")
7. return render(request, "sp\_user/member.html")

### 视图类装饰器使用

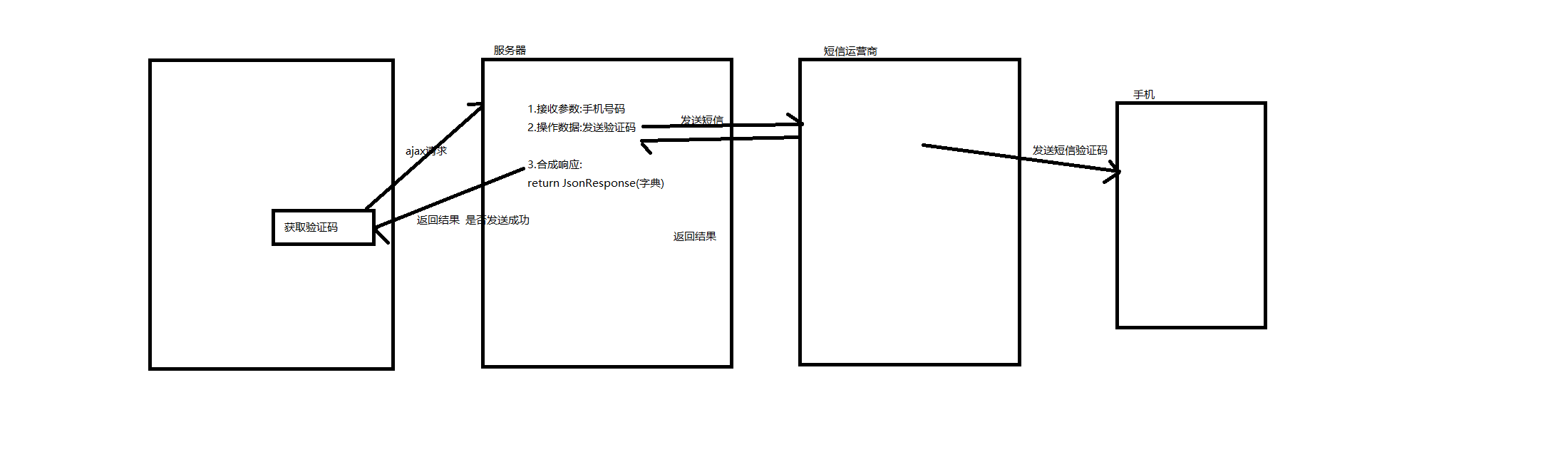
[https://docs.djangoproject.com/en/1.11/topics/class-based-views/intro/#decorating-the-class](https://docs.djangoproject.com/en/1.11/topics/class-based-views/intro/" \l "decorating-the-class" \t "https://www.zybuluo.com/yqcdwg/note/_blank)   
1.由于很多的视图类都可能需要使用装饰验证登录，所以，同样创建一个基础视图类，该基础视图类专门帮我们验证登录   
在项目根目录下，创建一个db包文件夹，在里面新建一个base\_view.py文件   
里面添加以下内容

1. from django.utils.decorators import method\_decorator
2. from django.views import View
3. from sp\_user.helper import verify\_login\_required
4. class BaseVerifyView(View):
5. """
6. 基础类视图，用于验证是否登录
7. """
8. @method\_decorator(verify\_login\_required)
9. def dispatch(self, request, \*args, \*\*kwargs):
10. return super(BaseVerifyView, self).dispatch(request, \*args, \*\*kwargs)

2.所有需要验证登录的视图类基础该基础视图类即可

1. # 个人中心 继承基础视图
2. class CenterView(BaseVerifyView):
3. def get(self, request):
4. return render(request, "sp\_user/member.html")
5. def post(self, request):
6. pass
7. # 修改个人资料 继承基础视图
8. class InfoView(BaseVerifyView):
9. def get(self, request):
10. return render(request, "sp\_user/infor.html")
11. def post(self, request):
12. pass

(1)发送短信流程图



<**script type="application/javascript"**>  
 **$(function()**{  
 //1.绑定点击事件  
 **$('.yzm-hq')**.on**('click'**, **function ()** {  
 //获取手机号码,手机号码上添加一个id  
 **var** phoneNum = **$('#phoneNum')**.val**()**;  
 **var** reg = /^1[3-9]\d{9}$/;  
 // 验证手机号  
 **if (**!reg.test**(**phoneNum**))** {  
 //验证失败  
 *alert***('手机号码格式错误')  
 return**;  
 }  
 **var** self = **this**;  
 //发送ajax请求  
 **$**.**ajax(**{  
 **type**:**'post'**,  
 **url**:**'**{% **url 'users:发送验证码'** %}**'**,  
 **data**:{**'phoneNum'**:phoneNum,**'csrfmiddlewaretoken'**:**'**{{ **csrf\_token** }}**'**},  
 **dataType**:**'json'**,  
 success:**function (**data**)** {  
 **if (**data.**error** == 0**)** {  
 //1.按钮变灰  
 **$(**self**)**.css**('background-color'**,**'gray')**;  
 //2.按钮不能再点击  
 **$(**self**)**.prop**('disabled'**,**true)**;  
 //3.开始倒计时  
 **var** time = 60;  
 **var** msg = time + **'秒后重新发送'**;  
 **$(**self**)**.val**(**msg**)**;  
  
 **var** timer = window.setInterval**(function ()** {  
 time -= 1;  
 **if (**time == 0**)**{  
 //1.按钮变灰  
 **$(**self**)**.css**('background-color'**,**'#76bb2a')**;  
 //2.按钮不能再点击  
 **$(**self**)**.prop**('disabled'**,**false)**;  
 //3.返回原来的文字  
 **$(**self**)**.val**("获取验证码")**;  
 window.clearInterval**(**timer**)**;  
 **return**;  
 }  
 msg = time + **'秒后重新发送'**;  
 **$(**self**)**.val**(**msg**)**;  
 },1000**)**;  
 }**else** {  
 *alert***(**data.errMsg**)**;  
 }  
 }  
 }**)** }**)** }**)**</**script**>

1. 安全退出:

直接在该函数中清除session数据,用request.session.flush()方法

然后跳转到登录界面重新登录

1. 在后台管理中添加数据

手动添加,无技巧

1. 头像修改

涉及到头像的地址保存问题,在Users模型中创建 head=models.ImageField**(**upload\_to**='head/%Y%m'**,default**='head/hlwtx.jpg'**,verbose\_name**='用户头像')**

字段用来保存头像的地址

在Terminal中输入pip install pillow 安装上三方依赖库

在settings文件里，定义一个变量叫MEDIA\_URL。（目的是分配一个资源URL）

MEDIA\_URL = "/static/media/"

配置该URL对应的物理目录存储地址

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'static/media')

在HTML渲染中，增加上传URL前缀   
1. 模型中的ImageField实际在数据库中就是varchar结构，在渲染时，就是一个字符串   
Django提供了一个{{ MEDIA\_URL }}渲染引擎变量   
2. 根据   
[https://docs.djangoproject.com/en/1.11/ref/settings/#media-url](https://docs.djangoproject.com/en/1.11/ref/settings/" \l "media-url" \t "https://www.zybuluo.com/yqcdwg/note/_blank)   
需要配置一个渲染处理器就可以使用这个变量了。

1. TEMPLATES = [
2. {
3. ...
4. 'OPTIONS': {
5. 'context\_processors': [
6. ... ...
7. 'django.template.context\_processors.media',
8. ],
9. },
10. },
11. ]

然后就可以在后台上传图片

# 将session保存到redis数据库(重要)

django默认将session信息存储到mysql数据库，如果用户访问量增大，会给mysql数据库服务造成非常大的压力。

redis缓存数据库，将数据存储到内存，存取速度快。   
django

参考文档：[http://django-redis-chs.readthedocs.io/zh\_CN/latest/#](http://django-redis-chs.readthedocs.io/zh_CN/latest/" \t "https://www.zybuluo.com/yqcdwg/note/_blank)

1. 安装   
   pip install django-redis
2. 作为 cache backend 使用配置
3. CACHES = {
4. "default": {
5. "BACKEND": "django\_redis.cache.RedisCache",
6. "LOCATION": "redis://127.0.0.1:6379/1",
7. "OPTIONS": {
8. "CLIENT\_CLASS": "django\_redis.client.DefaultClient",
9. }
10. }
11. }

URL 格式举例

1. redis://[:password]@localhost:6379/0
2. rediss://[:password]@localhost:6379/0
3. unix://[:password]@/path/to/socket.sock?db=0

配置session的存储引擎

* 1. SESSION\_ENGINE = "django.contrib.sessions.backends.cache"
  2. SESSION\_CACHE\_ALIAS = "default"

了解   
设置SESSION\_ENGINE项指定Session数据存储的方式，可以存储在数据库、缓存、Redis等。

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。

1. 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。

1. 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。

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

首页:

添加模型

商品分类表

# 商品分类表  
**class GoodsClass(**BaseModel**):** class\_name **=** models.CharField**(**max\_length**=**50,verbose\_name**='分类名')** class\_intro **=** models.CharField**(**max\_length**=**250,verbose\_name**='分类介绍'**,null**=True)  
  
 def \_\_str\_\_(**self**):  
 return** self.class\_name  
  
 **class Meta:** db\_table **= 'goods\_class'** verbose\_name **= '商品分类'** verbose\_name\_plural **=** verbose\_name

商品SPU表

**class GoodsSpu(**BaseModel**):** name **=** models.CharField**(**max\_length**=**50,verbose\_name**='商品spu名称')** desc **=** RichTextUploadingField**(**verbose\_name**='商品spu详情')  
  
 def \_\_str\_\_(**self**):  
 return** self.name  
  
 **class Meta:** db\_table **= 'goods\_spu'** verbose\_name **= '商品Spu表'** verbose\_name\_plural **=** verbose\_name

商品单位表

# 商品单位表  
**class GoodsUnit(**BaseModel**):** unitName **=** models.CharField**(**max\_length**=**20,verbose\_name**='单位名称')  
  
 def \_\_str\_\_(**self**):  
 return** self.unitName  
  
 **class Meta:** db\_table **= 'goods\_unit'** verbose\_name **= '商品单位表'** verbose\_name\_plural **=** verbose\_name

商品相册表

# 商品相册表  
**class GoodsPhotos(**BaseModel**):** image **=** models.ImageField**(**upload\_to**='goods/%Y%m'**,verbose\_name**='图片地址')** goods\_sku **=** models.ForeignKey**(**to**='GoodsSku'**,verbose\_name**='商品sku\_ID')  
  
 def \_\_str\_\_(**self**):  
 return "商品相册"  
  
 class Meta:** db\_table **= 'goods\_photos'** verbose\_name **= '商品相册表'** verbose\_name\_plural **=** verbose\_name

商品SKU表

# 商品Sku表  
**class GoodsSku(**BaseModel**):** goods\_name **=** models.CharField**(**max\_length**=**100,verbose\_name**='商品名')** goods\_intro **=** models.TextField**(**null**=True**,verbose\_name**='商品介绍')** price **=** models.DecimalField**(**decimal\_places**=**2,max\_digits**=**10,verbose\_name**='商品价格')** unit **=** models.ForeignKey**(**to**='GoodsUnit'**,verbose\_name**='商品单位')** num **=** models.PositiveIntegerField**(**verbose\_name**='库存')** sellNum **=** models.PositiveIntegerField**(**verbose\_name**='销量')** logo **=** models.ImageField**(**upload\_to**='goods/%Y%m'**,verbose\_name**='商品图片')** is\_putaway **=** models.BooleanField**(**default**=False**,verbose\_name**='是否上架')** goods\_cate **=** models.ForeignKey**(**to**='GoodsClass'**,verbose\_name**='商品分类')** goods\_spu **=** models.ForeignKey**(**to**='GoodsSpu'**,verbose\_name**='商品spu')  
  
 def \_\_str\_\_(**self**):  
 return** self.goods\_name  
  
 **class Meta:** db\_table **= 'goods\_sku'** verbose\_name **= '商品sku表'** verbose\_name\_plural **=** verbose\_name

轮播表

# 轮播  
**class Carousel(**BaseModel**):** goods\_name **=** models.CharField**(**max\_length**=**150,verbose\_name**='轮播活动名称')** goodsSku\_id **=** models.ForeignKey**(**to**='GoodsSku'**,max\_length**=**50,verbose\_name**='商品id')** image **=** models.ImageField**(**upload\_to**='banner/%Y%m/%d'**,verbose\_name**='图片地址')** order **=** models.SmallIntegerField**(**verbose\_name**='排序'**,default**=**0**)  
  
  
 def \_\_str\_\_(**self**):  
 return** self.goods\_name  
  
 **class Meta:** db\_table **= 'Carousel'** verbose\_name **= '轮播表'** verbose\_name\_plural **=** verbose\_name

首页活动

**class Activity(**BaseModel**):** title **=** models.CharField**(**verbose\_name**='活动名称'**,max\_length**=**150**)** img\_url **=** models.ImageField**(**verbose\_name**='活动图片地址'**,upload\_to**='activity/%Y%m/%d')** activity\_url **=** models.URLField**(**verbose\_name**='活动的url地址'**,max\_length**=**200**)  
  
 def \_\_str\_\_(**self**):  
 return** self.title  
  
 **class Meta:** verbose\_name **= '活动管理'** verbose\_name\_plural **=** verbose\_name

特色专区

# 特色专区  
**class Activity\_zone(**BaseModel**):** title **=** models.CharField**(**verbose\_name**='活动专区名称'**,max\_length**=**150**)** title\_intro **=** models.CharField**(**verbose\_name**='活动专区简介'**,  
 max\_length**=**200,  
 null**=True**,  
 blank**=True)** order **=** models.SmallIntegerField**(**verbose\_name**='排序'**,  
 default**=**0**)** is\_putaway **=** models.BooleanField**(**verbose\_name**='是否上线'**,  
 choices**=((False**,**'下架')**,**(True**,**'上架')**,**)**,  
 default**=**0**)** goods\_sku **=** models.ManyToManyField**(**to**='GoodsSku'**,verbose\_name**="商品")  
  
 def \_\_str\_\_(**self**):  
 return** self.title  
  
 **class Meta:** verbose\_name **= '活动专区管理'** verbose\_name\_plural **=** verbose\_name

1. 轮播图

有轮播模型,后台中在轮播模型中添加图片,然后渲染出来即可

1. 定位

在高德地图中注册账号,创建应用得到key值,然后修改示例

到首页代码中修改

1. 超市-->进入商品分类

分类的渲染

先将商品分类中的所有类查询出来,合成上下文渲染页面

在html中可直接通过for in 展示出来

分类的列表

将商品的sku表全部商品查询出来,合成上下文渲染

在html中可以直接查询数据

1. 将spu的商品详情设置为富文本

1.在您的python路径中安装或添加django-ckeditor。

1. pip install django-ckeditor

2.将ckeditor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. ]

3.配置静态文件路径   
看文档：[https://docs.djangoproject.com/en/dev/howto/static-files/](https://docs.djangoproject.com/en/dev/howto/static-files/" \t "https://www.zybuluo.com/yqcdwg/note/_blank)

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
17. 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']**

排序规则列表是以下判断转化为的简单模式

# if order == 0:  
# goods\_skus = goods\_skus.order\_by("pk")  
# elif order == 1:  
# goods\_skus = goods\_skus.order\_by("-sale\_num")  
# elif order == 2:  
# goods\_skus = goods\_skus.order\_by("price")  
# elif order == 3:  
# goods\_skus = goods\_skus.order\_by("-price")  
# elif order == 4:  
# goods\_skus = goods\_skus.order\_by("-create\_time")

需要在html中加上

.**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**: 20**px**;  
 **font-style**: **normal**;  
}

并在js中修改

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

2.4. 全文搜索

2.4.1 需求

使用关键字即可搜索

## 开始使用

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

-- 重写搜索就可以对中文进行分词了

购物车功能:

1. 需求:

(1)在商品列表页，商品详情页面，用户点击可以将该商品加入到购物车

(2)在商品列表页面，用户每次点击+ 或者 - 都在购物车中添加或者减少一个商品数量

(3)在商品详情页面，用户可以指定添加多少商品到购物车

(4)通过ajax提交实现，无刷新效果，提高用户体验度

(5)在购物车管理页面，用户可以管理购物车功能，可以选中某些商品，可以全选，全不选，可以添加商品数量和减少商品数量。当商品数量为0上删除该商品数据。

(6)由于购物车数据用户可以频繁操作，如果保存到数据库会增加数据库的负担，造成数据库宕机，所以将购物车数据添加到redis中保存，大大减少了数据库服务器的压力。

2.

问题及难点:

1.短信验证的难点是如何完成ajax的传参设计

2.分类的current选中问题

3.写的过程中,经常出现少写某些参数的问题,导致莫名其妙的报错

总结应该是自己写代码过少的问题

1. 建立商品模型多表时的外键的确立
2. 连表查询的技巧问题,可以省掉很多的时间
3. 排序时遇到找不到位置参数的错误,发现index页面中没有修改跳转
4. 搜索时,page是当前检索结果,遍历page后是一个个searchResult对象,searchResult才是检索的模型对象
5. 安装redis到windows服务中

在管理员命令模式下

安装命令:

redis-server.exe --service-install redis.windows.conf --loglevel verbose

服务器操作:

1. net start redis 开启服务
2. net stop redis 停止服务
3. sc delete redis 删除服务

技巧: 1.前期未建基础模型,发现每次都要多写三个字段,后来在根目录下创建了db包,里面创建了基础模型BaseModel,后面的所有模型都继承自BaseModel