# 项目介绍

* 1. 项目描述简介

通过完成一个仿饿了么的在线订餐平台，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
  2. 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. XXX功能模块
   1. 需求

1.用户输入11位手机号码，再输入6-16位的密码，确认密码要与密码相同，填写后，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

11.用户填写信息过后，通过表单验证手机号码是否被注册和格式是否正确，然后发送验证码，然后发送验证码，向服务器发送ajxs请求，在ajxs中验证手机号是否正确，发送短信

2.服务器响应返回验证码，用户点击立即注册，数据库保存数据，注册成功跳转到登录页面

* 1. 设计要点（数据库和页面交互）

11.model所需的字段

Create\_time 创建时间 datetime（用户注册的时间）

Update\_time 更新时间 datetime （用户修改信息的时间）

Is\_delete 是否删除 boolean （假删除）

Nickname 用户名 varchar

Tel 手机号 varchar

Password 密码 varchar (哈希加密)

Gender 性别 choice (男， 女， 保密)

Birthday 生日 datetime

School 学校 varchar

Location 地址 varchar

Hometown 地址 varchar

2.前端页面用表单校验

1）.对文本框中的字段进行校验，如果为空提示必须填写信息

2）.对必填的字段进行校验，校验（长度，格式等）

3）. 对验证码用ajax进行校验，验证码是从第三方获取到的，用户必须填写与第三方发送过来的验证码一致的数据，否则提示用户验证码不正确。

* 1. 要点难点及解决方案

1问题1.验证码的获取

解决方法;注册阿里云平台，申请一个模板和签名，当点击获取验证码时，阿里云发送验证码到用户手机

问题2.前端代码js代码遗忘

回顾视屏关于js内容，上课视屏回顾

问题3.函数的封装

功能的重复，就是代码重复，每次用到该功能会重复代码加重自己的工作量。

解决：通过文档定义封装函数，

问题4.对验证码校验

解决方案：通过ajax将用户从第三方平台得到的校验码获取到，用户填写校验码后将用户填写的和从第三方平台获取的进行比较，如果一致那么用户填写的校验码没有问题，如果不一致，则用户填写的校验码有错误，注册失败并提示用户。

* 1. 用户登录需求
     + 1. 用户登录成功
       2. 跳转到商城主页
  2. 流程
     + 1. 修改前端页面，添加from跟令牌
       2. Form验证用户数据
  3. 设计要点（数据库和页面交互）

1. 1. 数据库的基本字段：

Id 主键

Create\_time 创建时间 datetime（用户注册的时间）

Update\_time 更新时间 datetime （用户修改信息的时间）

Is\_delete 是否删除 boolean （假删除）

Nickname 用户名 varchar

Telephone 手机号 varchar

Password 密码 varchar (哈希加密)

Gender 性别 choice (男， 女， 保密)

Birthday 生日 datetime

School 学校 varchar

Location 地址 varchar

Hometown 地址 varchar

2.，交互设计

1.对输入的内容校验，如果为空提示填写信息

2.对填写的字段进行判断，判断用户输入是否符合规范

3.将错误提示合成渲染到html页面来提示用户

* 1. 要点难点及解决方案
     + 1. 如何对比被哈希加密的密码
       2. 将用户输入的密码转为哈希，然后在跟数据库密码对比
  2. 用户修改资料需求
     + 1. 用户登录到商城；
       2. 用户信息被渲染到个人资料
       3. 用户到个人信息修改
       4. 点击保存
  3. 流程
     + 1. 用户填写好信息点击保存
       2. Form表单验证输入字段是否合法
       3. 校验通过保存数据库或修改成功
  4. 设计要点
     + 1. 数据库
       2. 交互

2.1对修改信息验证是否合法

2.2上传图片时判断是否为空

* 1. 要点难点及解决方案

难点1：用户登录状态

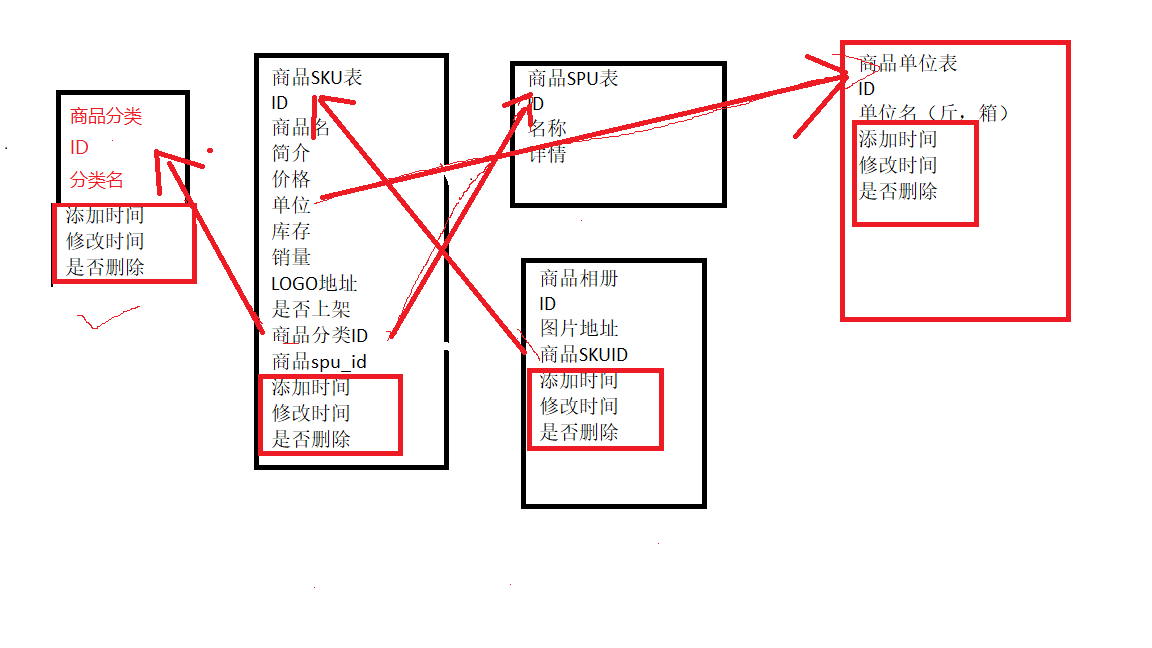
解决：利用session验证是否登录，在登录时把登录信息用session存入redis中

难点2：用户的信息渲染到页面中

解决：利用存入的seesion信息查询出用户，在合成渲染到页面

难点3：头像的上传

解决：查看文档，安装pillow

1. 商城功能模块
   1. 增删改查需求
      * 1. 商家登录后台系统；
        2. 添加+删除+修改+查看商品信息
        3. 将商品信息渲染到详情页面
   2. 流程
      * 1. 画模型联系图
        2. 设计模型数据库
        3. 将数据库中的模型名添加到admin控制模块中
        4. 创建超级用户管理数据
        5. 登录到后台
        6. 对数据进行操作
   3. 设计要点（数据库和页面交互）
      * 1. 
        2. 数据库

from ckeditor\_uploader.fields import RichTextUploadingField  
from django.db import models  
  
# Create your models here.  
from db.model\_base import BaseModel  
  
is\_on\_sale\_choices = (  
 (False, "下架"),  
 (True, "上架"),  
)  
  
class Category(BaseModel):  
 *"""商品分类表"""* cate\_name=models.CharField(verbose\_name='分类名称',  
 max\_length=20  
 )  
 brief = models.CharField(verbose\_name='描述',  
 max\_length=200,  
 null=True,  
 blank=True  
 )  
  
  
 def \_\_str\_\_(self):  
 return self.cate\_name  
  
 class Meta:  
 verbose\_name = "商品分类管理"  
 verbose\_name\_plural = verbose\_name  
  
  
class Goods\_Spu(BaseModel):  
 *"""商品Spu表"""* spu\_name = models.CharField(verbose\_name='商品SPU名称',  
 max\_length=20, )  
 #将之前详情修改使用富文本  
 spu\_detail = RichTextUploadingField(verbose\_name='商品详情')  
  
 def \_\_str\_\_(self):  
 return self.spu\_name  
  
 class Meta:  
 verbose\_name = "商品SPU"  
 verbose\_name\_plural = verbose\_name  
  
  
class Unit(BaseModel):  
 *"""商品单位表"""* unit\_name=models.CharField(max\_length=20,  
 verbose\_name='单位名')  
  
 def \_\_str\_\_(self):  
 return self.unit\_name  
  
 class Meta:  
 verbose\_name="商品单位管理"  
 verbose\_name\_plural=verbose\_name  
  
class Goods\_Sku(BaseModel):  
 *"""商品SKU表"""* sku\_name=models.CharField(max\_length=100,  
 verbose\_name='商品sku名称')  
  
 brief=models.CharField(max\_length=200,  
 null=True,  
 blank=True,  
 verbose\_name='商品简介')  
  
 price=models.DecimalField(max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 verbose\_name='价格')  
  
 unit=models.ForeignKey(to="Unit",verbose\_name="单位")  
  
 stock=models.IntegerField(default=0,  
 verbose\_name='库存')  
  
 salesvolume=models.IntegerField(default=0,  
 verbose\_name='销量')  
  
 #默认相册第一张作为封面图片  
 logo=models.ImageField(upload\_to='goods/%Y%m/%d',  
 verbose\_name='封面图片')  
  
 is\_on\_sale=models.BooleanField(choices=is\_on\_sale\_choices,  
 default=False,  
 verbose\_name='是否上架')  
 category=models.ForeignKey(to="Category" ,  
 verbose\_name='商品分类id')  
  
 goods\_spu=models.ForeignKey(to="Goods\_Spu",  
 verbose\_name='商品Spuid')  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name  
  
  
class Album(BaseModel):  
 *"""商品相册表"""* img\_url=models.ImageField(upload\_to='goods\_gallery/%Y%m/%d',  
 verbose\_name='相册地址'  
 )  
  
 goods\_sku=models.ForeignKey(to='Goods\_Sku')  
  
 class Meta:  
 verbose\_name = "商品相册管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return "商品相册:{}".format(self.img\_url.name)  
  
  
class Banner(BaseModel):  
 *"""首页轮播表"""* name= models.CharField(verbose\_name="轮播活动名",  
 max\_length=150,  
 )  
  
 img\_url = models.ImageField(verbose\_name='轮播图片地址',  
 upload\_to='banner/%Y%m/%d'  
 )  
  
 order = models.SmallIntegerField(verbose\_name="排序",  
 default=0,  
 )  
  
 goods\_sku = models.ForeignKey(to="Goods\_Sku", verbose\_name="商品SKU")  
  
 class Meta:  
 verbose\_name = "轮播管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class Activity(BaseModel):  
 *"""首页活动表"""* 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'  
 )  
 url\_site = 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  
  
  
class ActivityZone(BaseModel):  
 *"""  
 首页活动专区  
 """* title = models.CharField(verbose\_name='活动专区名称', max\_length=150)  
 brief = models.CharField(verbose\_name="活动专区的简介",  
 max\_length=200,  
 null=True,  
 blank=True,  
 )  
  
 order = models.SmallIntegerField(verbose\_name="排序",  
 default=0,  
 )  
  
 is\_on\_sale = models.BooleanField(verbose\_name="是否上线",  
 choices=is\_on\_sale\_choices,  
 default=0,  
 )  
  
 goods\_sku = models.ManyToManyField(to="Goods\_Sku",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

页面交互

用户添加商品的功能能在后台显示出来

* 1. 要点难点及解决方案
     + 1. 如何将图片在后台管理中显示出来

# 解决：富文本编辑器的使用

在django模型中，如果将商品详情字段类型设置为TextFiled(),当将该模型加入到后台管理的时候，只会呈现出普通的文本框。但是，我们实际工作中，编辑商品详情需要进行格式的编辑、上传图片、排版、修改文字颜色，这些普通的TextField都是不能实现的。

怎么解决？使用富文本编辑器   
富文本编辑器有很多种类，我们项目中采用ckeditor   
文档地址：[https://pypi.org/project/django-ckeditor/](https://pypi.org/project/django-ckeditor/" \t "_blank)   
文档中非常详细，具体参考文档，以下是简单入门步骤：

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 "_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

9.前台模板显示   
{{content|safe}}

查看文档配置。

* 1. 商品详情显示功能
     1. 需求
        1. 点击页面中的商品跳转到详情页面
        2. 渲染出点击商品的基本信息
     2. 流程

1.实现添加商品信息的渲染

* + 1. 设计要点（数据库和页面交互）

1 class Goods\_Sku(BaseModel):  
 *"""商品SKU表"""* sku\_name=models.CharField(max\_length=100,  
 verbose\_name='商品sku名称')  
  
 brief=models.CharField(max\_length=200,  
 null=True,  
 blank=True,  
 verbose\_name='商品简介')  
  
 price=models.DecimalField(max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 verbose\_name='价格')  
  
 unit=models.ForeignKey(to="Unit",verbose\_name="单位")  
  
 stock=models.IntegerField(default=0,  
 verbose\_name='库存')  
  
 salesvolume=models.IntegerField(default=0,  
 verbose\_name='销量')  
  
 #默认相册第一张作为封面图片  
 logo=models.ImageField(upload\_to='goods/%Y%m/%d',  
 verbose\_name='封面图片')  
  
 is\_on\_sale=models.BooleanField(choices=is\_on\_sale\_choices,  
 default=False,  
 verbose\_name='是否上架')  
 category=models.ForeignKey(to="Category" ,  
 verbose\_name='商品分类id')  
  
 goods\_spu=models.ForeignKey(to="Goods\_Spu",  
 verbose\_name='商品Spuid')  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name  
  
  
class Album(BaseModel):  
 *"""商品相册表"""* img\_url=models.ImageField(upload\_to='goods\_gallery/%Y%m/%d',  
 verbose\_name='相册地址'  
 )  
  
 goods\_sku=models.ForeignKey(to='Goods\_Sku')  
  
 class Meta:  
 verbose\_name = "商品相册管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return "商品相册:{}".format(self.img\_url.name)

* + 1. 要点难点及解决方案

难点：表的查询一对多 ，多对1

解决方法：word文档

实体间关系及处理方式

一对一: 使用相同的主键id进行关联

一对多: 在多的一方的表中创建一个字段 保存 一的一方的主键id

多对多: 创建一张中间表保存两个表的主键id

2. django中 实体间关系及处理方式

一对一: 使用的 models.OneToOneField(to="应用名.模型类名") 进行处理一对一关系, 设置在任意一方

一对多: 使用的 models.ForeigenKey(to="应用名.模型类名") 进行处理一对多关系, 设置在多的一方

多对多: 使用的 models.ManyToManyField(to="应用名.模型类名") 进行处理多对多关系处理, 设置在任意一方

重要: 一对多

3. 分开讲解

一对多关系处理:

添加:

原始: 按照之前的添加方式, 先添加一的一方,再添加多的一方

关联字段赋值 赋予的 是对象

快捷添加方式:

一对象.模型类名\_set.create(属性名=值)

查询: (重点)

逆向:

一对象.模型类名\_set.all()

正向:

多对象.关联字段

跨模型的过滤查询:

和之前的查询一样, 只是条件 : 模型类名\_\_属性....

模型类a.objects.filter(模型类b\_\_属性=值)

例如:

班级与学生的关系

# 班级模型

class ClassInfo(models.Model):

class\_name = models.CharField(max\_length=50)

class\_room = models.CharField(max\_length=50)

# 学生模型

class Student(models.Model):

gender\_choices = (

(1,'男'),

(2,'女'),

(3,'保密'),

)

stu\_name = models.CharField(max\_length=50)

age = models.SmallIntegerField()

gender = models.SmallIntegerField(choices=gender\_choices,default=3)

classinfo = models.ForeignKey(to="ClassInfo")

一对一关系:

学生基本信息与学生详细信息地址

多对多关系:

班级与老师的关系

操作方式和一对多一样.

* 1. 商品分类列表
     1. 需求
        1. 得到所有分类的信息
        2. 遍历分类信息渲染到页面
        3. 通过分类查询所有商品
        4. 将查询的商品渲染到对应
     2. 流程
        1. 查询所有分类信息
        2. 在静态页面遍历渲染分类信息
        3. 分类信息查询类的所有商品
        4. 遍历商品到自己所属类的下面
     3. 设计要点

数据库

class Category(BaseModel):  
 *"""商品分类表"""* cate\_name=models.CharField(verbose\_name='分类名称',  
 max\_length=20  
 )  
 brief = models.CharField(verbose\_name='描述',  
 max\_length=200,  
 null=True,  
 blank=True  
 )

class Unit(BaseModel):  
 *"""商品单位表"""* unit\_name=models.CharField(max\_length=20,  
 verbose\_name='单位名')

def \_\_str\_\_(self):  
 return self.unit\_name  
  
class Meta:  
 verbose\_name="商品单位管理"  
 verbose\_name\_plural=verbose\_name

class Goods\_Spu(BaseModel):  
 *"""商品Spu表"""* spu\_name = models.CharField(verbose\_name='商品SPU名称',  
 max\_length=20, )  
 #将之前详情修改使用富文本  
 spu\_detail = RichTextUploadingField(verbose\_name='商品详情')  
  
 def \_\_str\_\_(self):  
 return self.spu\_name  
  
 class Meta:  
 verbose\_name = "商品SPU"  
 verbose\_name\_plural = verbose\_name

class Goods\_Sku(BaseModel):  
 *"""商品SKU表"""* sku\_name=models.CharField(max\_length=100,  
 verbose\_name='商品sku名称')  
  
 brief=models.CharField(max\_length=200,  
 null=True,  
 blank=True,  
 verbose\_name='商品简介')  
  
 price=models.DecimalField(max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 verbose\_name='价格')  
  
 unit=models.ForeignKey(to="Unit",verbose\_name="单位")  
  
 stock=models.IntegerField(default=0,  
 verbose\_name='库存')  
  
 salesvolume=models.IntegerField(default=0,  
 verbose\_name='销量')  
  
 #默认相册第一张作为封面图片  
 logo=models.ImageField(upload\_to='goods/%Y%m/%d',  
 verbose\_name='封面图片')  
  
 is\_on\_sale=models.BooleanField(choices=is\_on\_sale\_choices,  
 default=False,  
 verbose\_name='是否上架')  
 category=models.ForeignKey(to="Category" ,  
 verbose\_name='商品分类id')  
  
 goods\_spu=models.ForeignKey(to="Goods\_Spu",  
 verbose\_name='商品Spuid')  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name  
  
  
class Album(BaseModel):  
 *"""商品相册表"""* img\_url=models.ImageField(upload\_to='goods\_gallery/%Y%m/%d',  
 verbose\_name='相册地址'  
 )  
  
 goods\_sku=models.ForeignKey(to='Goods\_Sku')  
  
 class Meta:  
 verbose\_name = "商品相册管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return "商品相册:{}".format(self.img\_url.name)

1. 页面交互

显示出有类别的商品跟类

* + 1. 要点难点及解决方案