# 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

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 | pycharm,navicat |
| 相关技术 | django,mysql,session,cookie,sqlite3,redis, ckeditor 等 |

* 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

2.1用户账号注册功能模块

* 1. 1需求

1.注册页面:手机号的正则验证,6位数以上的密码,两次输入密码的正确性, 协议同意的勾选与否的验证,填写完毕后点击获取验证码.

2.以上流程完成后获取到验证码,验证码的验证,点击注册后,加密密码,写入数据库.

2.1.2流程

1.用户填写手机,密码等信息,填写后点击获取验证码,判断手机格式是否有问题,没有就向验证码平台发起验证码请求

2. 验证码平台返回并响应验证码,用户填写验证码,勾选协议,点击注册,form表单返回数据

3.form表单验证数据是否合法:手机是否已注册,手机格式是否合法,密码长度是否足够,两次密码是否一致,协议是否勾选同意,如果有问题,返回错误提示

4.如果用户信息无误,则注册成功,写入数据库并加密密码,跳转登录页面.

2.1.3设计要点（数据库和页面交互）

1.数据库的字段

Id 主键

mobile 手机号 varchar (必填)

password 密码 varchar (必填,哈希加密)

sex 性别 chioce

birthday 生日 datatime

school 学校 varchar

location 地址 varchar

hometown 家庭地址 varchar

is\_delete 是否删除 boolean(假删)

add\_time 创建时间 datetime （用户注册时间）

update\_time 更新时间 datetime （用户修改信息时间）

2. 前端页面对数据的校验

* 1. 对每一个文本框添加placeorder说明字段；
  2. 文本框进行空字段校验，如果为空，点击注册按钮后会提示该项为必填项；
  3. 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，提示用户；
  4. 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；

2.1.4要点难点及解决方案

难点1：验证码的获取

解决方案：通过阿里云发送，申请一个模板和签名，当用户点击获取验证码时，阿里云向该手机号发送验证码

难点2：项目根目录的判断

解决方案：找到有包含子目录的一级

难点3：验证form表单的数据

解决方案：在子运用到的目录下创建一个forms.py文件定义,创建验证的类来规定验证

2.2用户登录功能模块

* 1. 1需求

1.手机号的正则验证,6位数以上的密码,输入密码的正确性.

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；

2.2.2流程

1.用户填写手机,密码,判断手机格式是否有问题,没有form表单返回数据

2.form表单验证数据是否合法:手机是否已注册,手机格式是否合法,密码长度是否足够,密码不能为空,如果有问题,返回错误提示

4. 将加密好的密码和手机号码与数据库内的数据进行比较，如果数据一致，用户登录成功。

2.2.3设计要点（数据库和页面交互）

1.数据库的字段

Id 主键

mobile 手机号 varchar (必填)

password 密码 varchar (必填,哈希加密)

sex 性别 chioce

birthday 生日 datatime

school 学校 varchar

location 地址 varchar

hometown 家庭地址 varchar

is\_delete 是否删除 boolean(假删)

add\_time 创建时间 datetime （用户注册时间）

update\_time 更新时间 datetime （用户修改信息时间）

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。
4. 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；

2.2.4要点难点及解决方案

难点1：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并且循环加盐，将加密后的密码与数据库内的密码进行比较是否一致。

难点2：验证form表单的数据

解决方案：

在子运用到的目录下创建一个forms.py文件定义,创建验证的类来规定验证

* 1. 用户修改功能模块
     1. 需求
        1. 用户的信息渲染到个人资料页面；
        2. 用户到个人资料页面修改个人资料页面内的信息；
        3. 忘记密码,修改密码必须在用户登录状态下完成
     2. 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。

用户登录状态下完成忘记密码,修改密码,通过提交form信息进行验证,

通过修改,否则提示用户信息

* + 1. 设计要点（数据库和页面交互）
       1. 数据库同注册数据库一致
       2. 交互
    2. 为文本框添加placeorder属性，提示用户需要填写的信息；
    3. 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；
    4. 上传图片时判断用户是否有上传图片；
    5. 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。
    6. 如果校验通过了，用户的数据将被修改成功。

用户登录状态下完成忘记密码,修改密码,通过提交form信息进行验证,

通过修改,否则提示用户信息

* + 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.难点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,使用django-redis保存session减少数据库压力,做到优化效果。

3. 难点3：session保存后的使用

解决方案：

创建一个保存session的方法,方便在程序中调用

4.用户头像的上传

解决方案：

在infor的html中的form表单中加入enctype="multipart/form-data"属性,通过session查询数据库,找到用户id,安装并使用pillow(专门处理图片)包,创建图片存储地址,通过id调用地址,并且在用户提交后刷新session使用户能马上看到更改后的头像,注意:判断用户头像是否更改,没有则不更新

5.用户上传时间问题

解决方案：

在infor的html中将时间的input框加入style=’date’,就不用转换数据格式

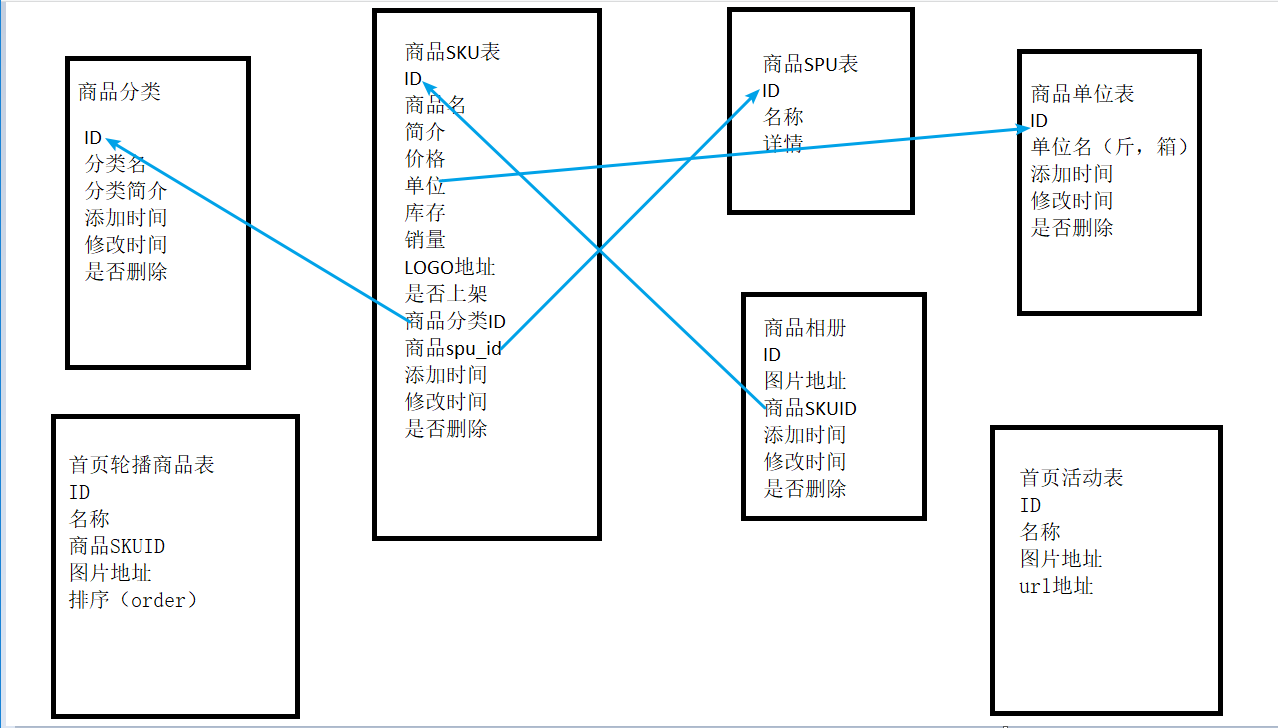
’

1. 商品功能模块

3.1商品增删改查模块

* + 1. 需求
       1. 商家登录后台.
       2. 商家可以增删查改或者看商品信息
       3. 完成添加的信息能写入数据库并且渲染到前台
    2. 流程
       1. 设计好数据库；
       2. 将数据库内供商家控制的模型添加到admin模块中方便更新管理；
       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。
    3. 设计要点（数据库和页面交互）

数据库:

分析表:

* + - 1. 商品分类

# 商品模型一：  
# 商品分类  
# ID  
# 分类名称  
# 描述  
# 排序  
class Category(BaseModel):  
 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  
 )  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
  
 def \_\_str\_\_(self):  
 return self.cate\_name  
  
 class Meta:  
 verbose\_name = "商品分类管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品SKU单位

# 商品模型二:  
# 商品SKU单位  
# ID  
# 单位（斤，箱）  
class Unit(BaseModel):  
 name = models.CharField(max\_length=20,  
 verbose\_name="单位")  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Meta:  
 verbose\_name = "商品单位管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品SPU表

# 商品模型三：  
# 商品SPU表  
# ID  
# 商品SPU名称  
# 商品详情  
class GoodsSPU(BaseModel):  
 spu\_name = models.CharField(verbose\_name='商品SPU名称',  
 max\_length=20  
 )  
 content = models.TextField(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

* + - 1. 商品SKU表

# 模型四：  
# 商品SKU表  
# ID  
# 商品SKU名称  
# 商品的简介  
# 价格  
# 单位  
# 库存  
# 销量  
# 封面图片  
# 是否上架  
# 商品分类  
# 商品SPU  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(verbose\_name='商品SKU名称',  
 max\_length=100,  
 )  
 brief = models.CharField(verbose\_name="商品的简介",  
 max\_length=200,  
 null=True,  
 blank=True,  
 )  
 price = models.DecimalField(verbose\_name='价格',  
 max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 )  
 unit = models.ForeignKey(to="Unit", verbose\_name="单位")  
  
 stock = models.IntegerField(verbose\_name='库存',  
 default=0)  
  
 sale\_num = models.IntegerField(verbose\_name='销量',  
 default=0)  
  
 # 默认相册中的第一张图片作为封面图片  
 logo = models.ImageField(verbose\_name='封面图片',  
 upload\_to='goods/%Y%m/%d'  
 )  
  
 is\_on\_sale = models.BooleanField(verbose\_name="是否上架",  
 choices=is\_on\_sale\_choices,  
 default=False)  
  
 category = models.ForeignKey(to="Category",  
 verbose\_name='商品分类'  
 )  
  
 goods\_spu = models.ForeignKey(to="GoodsSPU", verbose\_name="商品SPU")  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品相册管理

# 模型五：  
# 商品相册管理  
# ID  
# 商品相册  
# 相册图片地址  
# 商品SKU  
class Gallery(BaseModel):  
 img\_url = models.ImageField(verbose\_name='相册图片地址',  
 upload\_to='goods\_gallery/%Y%m/%d'  
 )  
  
 goods\_sku = models.ForeignKey(to="GoodsSKU", verbose\_name="商品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. 首页轮播

# 模型六：  
# 首页轮播  
# ID  
# 轮播活动名  
# 轮播图片地址  
# 排序  
# 商品SKU  
class Banner(BaseModel):  
 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="GoodsSKU", verbose\_name="商品SKU")  
  
 class Meta:  
 verbose\_name = "轮播管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name

* + - 1. 首页活动

# 模型七：  
# 首页活动  
# ID  
# 活动名称  
# 活动图片地址  
# 活动的url地址  
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'  
 )  
 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

* + - 1. 首页活动专区

# 模型八：  
# 首页活动专区  
# ID  
# 活动专区名称  
# 排序  
# 上否上线  
# 商品  
class ActivityZone(BaseModel):  
 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="上否上线",  
 choices=is\_on\_sale\_choices,  
 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

页面交互

1)在创建时注意要在模型里创建图片地址,方便后台上传图片

2)每张表的对应关系要分析清楚再创建,以免以后修改麻烦

* + 1. 要点难点及解决方案
       1. 后台admin创建orm操作比较吃力

解决方案：

查看Django文档,网上找资料

* + - 1. 在视图中写入查询数据库显示到网址时,index网页报错

解决方案：

由于我先渲染了index页面才发现detail页面中需要定义每个商品的id,通过id查询商品,定义后index中html的detail反向路由便缺少id,所以报错,将商品的SKU在主页的视图中查询出来,并在index的html中detail反向路由后加上for循环的id

* + - 1. 富文本编辑器ckeditor的使用,使文本丰富,便于后台操作

解决方案：

首先安装django-ckeditor, 将ckeditor添加到您的INSTALLED\_APPS设置中,注意配置静态文件STATIC\_ROOT = os.path.join(BASE\_DIR, "static"),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, 配置好后注意注释掉静态文件, 设置ckeditor的上传目录 ,在主路由中配置 ckeditor 上传文件使用到的url地址 ,安装pillow(已经安装不用再安装), 设置ckeditor在后台显示的样式,这样admin后台就能很好的编辑文本了

* + - 1. 商品模块的前台显示的设计在,数据库的模型关系问题

解决方案：

在视图设计中,打开models,看到每个表之间的设计关系,多对多,一对多,还是一对一,在调用自己想要的数据时可以通过Ctrl+鼠标右键跳转到上一级关系上,多看就不会头晕,一步一步的来,不容易出错,要是出错不报错没数据显示,可以尝试断点+debug,调试一下,大部分问题都能解决

* 1. 商品首页模块
     1. 需求
        1. 渲染首页banner轮播部分
        2. 渲染首页不规则展示界面
        3. 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. 流程
        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. 渲染不规则推荐信息部分

查询出不规则展示信息的，将所有的信息以字典形式传递到主页，在主页上渲染出所有不规则信息。

* + - 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. 设计要点（数据库和页面交互）
       1. 数据库

首页轮播

# 模型六：  
# 首页轮播  
# ID  
# 轮播活动名  
# 轮播图片地址  
# 排序  
# 商品SKU  
class Banner(BaseModel):  
 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="GoodsSKU", verbose\_name="商品SKU")  
  
 class Meta:  
 verbose\_name = "轮播管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name

首页活动

# 模型七：  
# 首页活动  
# ID  
# 活动名称  
# 活动图片地址  
# 活动的url地址  
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'  
 )  
 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

首页活动专区

# 模型八：  
# 首页活动专区  
# ID  
# 活动专区名称  
# 排序  
# 上否上线  
# 商品  
class ActivityZone(BaseModel):  
 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="上否上线",  
 choices=is\_on\_sale\_choices,  
 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. 界面交互

无

* + 1. 要点难点及解决方案

难点1：

查询活动以及活动下的商品信息

解决方案：

先查询出所有活动信息，遍历活动信息，利用，每一个活动来查询出活动模块下的所有商品信息，遍历这些信息，利用遍历得到的信息渲染出活动模块。

* 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；
        4. 遍历出所有商品，将商品渲染到相应的分类下。
     3. 设计要点（数据库和页面交互）
        1. 数据库

商品分类

# 商品模型一：  
# 商品分类  
# ID  
# 分类名称  
# 描述  
# 排序  
class Category(BaseModel):  
 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  
 )  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
  
 def \_\_str\_\_(self):  
 return self.cate\_name  
  
 class Meta:  
 verbose\_name = "商品分类管理"  
 verbose\_name\_plural = verbose\_name

商品SKU单位

# 商品模型二:  
# 商品SKU单位  
# ID  
# 单位（斤，箱）  
class Unit(BaseModel):  
 name = models.CharField(max\_length=20,  
 verbose\_name="单位")  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Meta:  
 verbose\_name = "商品单位管理"  
 verbose\_name\_plural = verbose\_name

商品SPU表

# 商品模型三：  
# 商品SPU表  
# ID  
# 商品SPU名称  
# 商品详情  
class GoodsSPU(BaseModel):  
 spu\_name = models.CharField(verbose\_name='商品SPU名称',  
 max\_length=20  
 )  
 content = models.TextField(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

商品SKU表

# 模型四：  
# 商品SKU表  
# ID  
# 商品SKU名称  
# 商品的简介  
# 价格  
# 单位  
# 库存  
# 销量  
# 封面图片  
# 是否上架  
# 商品分类  
# 商品SPU  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(verbose\_name='商品SKU名称',  
 max\_length=100,  
 )  
 brief = models.CharField(verbose\_name="商品的简介",  
 max\_length=200,  
 null=True,  
 blank=True,  
 )  
 price = models.DecimalField(verbose\_name='价格',  
 max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 )  
 unit = models.ForeignKey(to="Unit", verbose\_name="单位")  
  
 stock = models.IntegerField(verbose\_name='库存',  
 default=0)  
  
 sale\_num = models.IntegerField(verbose\_name='销量',  
 default=0)  
  
 # 默认相册中的第一张图片作为封面图片  
 logo = models.ImageField(verbose\_name='封面图片',  
 upload\_to='goods/%Y%m/%d'  
 )  
  
 is\_on\_sale = models.BooleanField(verbose\_name="是否上架",  
 choices=is\_on\_sale\_choices,  
 default=False)  
  
 category = models.ForeignKey(to="Category",  
 verbose\_name='商品分类'  
 )  
  
 goods\_spu = models.ForeignKey(to="GoodsSPU", verbose\_name="商品SPU")  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name

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。

* 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

# 商品模型一：  
# 商品分类  
# ID  
# 分类名称  
# 描述  
# 排序  
class Category(BaseModel):  
 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  
 )  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
  
 def \_\_str\_\_(self):  
 return self.cate\_name  
  
 class Meta:  
 verbose\_name = "商品分类管理"  
 verbose\_name\_plural = verbose\_name

商品SKU表

# 模型四：  
# 商品SKU表  
# ID  
# 商品SKU名称  
# 商品的简介  
# 价格  
# 单位  
# 库存  
# 销量  
# 封面图片  
# 是否上架  
# 商品分类  
# 商品SPU  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(verbose\_name='商品SKU名称',  
 max\_length=100,  
 )  
 brief = models.CharField(verbose\_name="商品的简介",  
 max\_length=200,  
 null=True,  
 blank=True,  
 )  
 price = models.DecimalField(verbose\_name='价格',  
 max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 )  
 unit = models.ForeignKey(to="Unit", verbose\_name="单位")  
  
 stock = models.IntegerField(verbose\_name='库存',  
 default=0)  
  
 sale\_num = models.IntegerField(verbose\_name='销量',  
 default=0)  
  
 # 默认相册中的第一张图片作为封面图片  
 logo = models.ImageField(verbose\_name='封面图片',  
 upload\_to='goods/%Y%m/%d'  
 )  
  
 is\_on\_sale = models.BooleanField(verbose\_name="是否上架",  
 choices=is\_on\_sale\_choices,  
 default=False)  
  
 category = models.ForeignKey(to="Category",  
 verbose\_name='商品分类'  
 )  
  
 goods\_spu = models.ForeignKey(to="GoodsSPU", verbose\_name="商品SPU")  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name

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

价格是多变的,可升可降,两种需求都要满足,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. 商品详情展示模块
     1. 需求
        1. 点击某个商品后，会跳转到该商品的详情页面；
        2. 该详情页面会渲染出商品的基本信息，商品图片，商品名称，价格，单位，详情。
     2. 流程
        1. 实现效果：详情页能够轮播该商品的图片信息；渲染出该商品的基本信息。
        2. 视图函数有一个参数gid, 使用gid查询出商品信息，再利用商品信息查询出商品的轮播图片。
     3. 设计要点（数据库和页面交互）
        1. 数据库

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

商品相册表：Goods\_Photography

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Img 图片 imagefield

Goods 商品id foreignkey

* + 1. 要点难点及解决方案
       1. 无

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。

1. 订单功能模块
   1. 确认订单
      1. 需求
         1. 在购物车页面，选择好要购买的商品后，点击去结算按钮，进入到确认订单页面；
         2. 在订单页面内显示出用户在购物车内选择好的商品；
         3. 用户选择或添加地址；
         4. 点击确认订单，进入下单页面。
      2. 流程
         1. 登录状态的用户选择要结算的商品点击提交订单；
         2. 在redis数据库中查询出从购物车提交过来的商品id和商品数量；
         3. 根据商品的数量和价格得到总价，在加上用户选择的运输方式的价格得到应付总金额；
         4. 选择或添加地址信息，当选择某个地址为默认地址时，之前的默认地址会被修改为普通地址，默认地址被渲染到页面上。
      3. 设计要点（数据库和页面交互）
         1. 数据库设计：

Redis数据库：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

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

运输方式表：AdressModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

User 所属用户 foreignkey

Name 收货人姓名 char

Telephone 收货人电话 char

Hcity 省份 char

Hproper 市区 char

Harea 地区 char

Brief 详细地址 char

Isdefault 是否默认 Boolean

订单信息表：OrderInfoModel

Order\_number 订单编号

order\_number 订单编号 char

total\_money 总金额 decimal

user 所属用户 foreignkey

username 收货人 char

telephone 收货人人电话 char

address 订单地址 char

status 状态 choice

transport 运输方式 foreignkey

transport\_price 运费 decimal

payway 付款方式 char

paymoney 实付金额 decimal

description 备注说明 char

* + - 1. 页面交互

将配送方式以下拉框形式显示，便于选择。

* + 1. 要点难点及解决方案

难点1：服务器高并发库存问题

解决方案：使用select\_for\_update()乐观锁，当出现库存问题时，订单提交失败，事务回滚。

* 1. 提交订单
     1. 需求
        1. 提交订单后，产生订单编号；
        2. 使用订单编号查询出订单信息；
        3. 查询出订单内的商品信息；
        4. 将成功创建的订单显示出来。
     2. 流程
        1. 登录状态的用户在确认订单页面点击提交订单按钮；
        2. 创建订单成功，订单编号以参数的形式传递到url内，使用ajax进行页面跳转；
        3. 将创建好的订单内的商品和价格信息渲染出来。
     3. 设计要点（数据库和页面交互）
        1. 数据库设计：

Redis数据库：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

用户表：UserModel

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

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

运输方式表：AdressModel

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

User 所属用户 foreignkey

Name 收货人姓名 char

Telephone 收货人电话 char

Hcity 省份 char

Hproper 市区 char

Harea 地区 char

Brief 详细地址 char

Isdefault 是否默认 Boolean

订单信息表：OrderInfoModel

order\_number 订单编号 char

total\_money 总金额 decimal

user 所属用户 foreignkey

username 收货人 char

telephone 收货人人电话 char

address 订单地址 char

status 状态 choice

transport 运输方式 foreignkey

transport\_price 运费 decimal

payway 付款方式 char

paymoney 实付金额 decimal

description 备注说明 char

商品订单表：OrderGoods

Create\_time 添加时间 datetime

Update\_time 更新时间 datetime

Is\_delete 是否删除 Boolean

Orderinfo 订单信息 foreignkey

Skuinfo 商品信息 foreignkey

Price 价格 decimal

Count 商品数量 int

* + - 1. 页面交互

确认订单页面的信息不允许修改。

* + 1. 要点难点及解决方案

难点1：服务器高并发库存问题

解决方案：使用select\_for\_update()悲观锁，当出现库存问题时，订单提交失败，事务回滚。

难点2：当出现错误时，订单也会创建成功

解决方案：使用事务机制，当出现错误时进行事务回滚，当创建出现错误时，不会再创建出订单。

1. 常见面试问题

① http请求的执行流程：

在接受一个Http请求之前的准备,启动一个支持WSGI网关协议的服务器监听端口等待外界的Http请求，比如Django自带的开发者服务器或者uWSGI服务器。

服务器根据WSGI协议指定相应的Handler来处理Http请求，并且初始化该Handler，在Django框架中由框架自身负责实现这一个Handler。 此时服务器已处于监听状态，可以接受外界的Http请求 当一个http请求到达服务器的时候

服务器根据WSGI协议从Http请求中提取出必要的参数组成一个字典（environ）并传入Handler中进行处理。

在Handler中对已经符合WSGI协议标准规定的http请求进行分析，比如加载Django提供的中间件，路由分配，调用路由匹配的视图等。 返回一个可以被浏览器解析的符合Http协议的HttpResponse。

② 查询集返回的列表过滤器：

all()：返回所有数据

filter()：返回满足条件的数据

exclude()：返回满足条件之外的数据，相当于sql语句中where部分的not关键字

der\_by()：排序

③ WSGI / uwsgi/ uWSGI区分

orWSGI：WSGI的全称是Web Server Gateway Interface（Web服务器网关接口），它不是服务器、python模块、框架、API或者任何软件，只是一种描述web服务器（如nginx，uWSGI等服务器）如何与web应用程序（如用Django、Flask框架写的程序）通信的规范。

server和application的规范在PEP3333中有具体描述，要实现WSGI协议，必须同时实现web server和web application，当前运行在WSGI协议之上的web框架有Bottle, Flask, Django。

uWSGI：uWSGI是一个全功能的HTTP服务器，实现了WSGI协议、uwsgi协议、http协议等。它要做的就是把HTTP协议转化成语言支持的网络协议。比如把HTTP协议转化成WSGI协议，让Python可以直接使用。

uwsgi：与WSGI一样，是uWSGI服务器的独占通信协议，用于定义传输信息的类型(type of information)。每一个uwsgi packet前4byte为传输信息类型的描述，与WSGI协议是两种东西，据说该协议是fcgi【FCGI：fast common gateway interface 快速通用网关接口协议的10倍快。

④ blank和null

blank:设置为True时，字段可以为空。设置为False时，字段是必须填写的。字符型字段CharField和TextField是用空字符串来存储空值的。如果为True，字段允许为空，默认不允许。

null:设置为True时，django用Null来存储空值。日期型、时间型和数字型字段不接受空字符串。所以设置IntegerField，DateTimeField型字段可以为空时，需要将blank，null均设为True。

如果为True，空值将会被存储为NULL，默认为False。如果想设置BooleanField为空时可以选用NullBooleanField型字段。

null 是针对数据库而言，如果 null=True, 表示数据库的该字段可以为空。NULL represents non-existent data.

blank 是针对表单的，如果 blank=True，表示你的表单填写该字段的时候可以不填。比如 admin 界面下增加 model 一条记录的时候。直观的看到就是该字段不是粗体

1. 技术点
   * + 1. 阿里短信验证
       2. ckeditor富文本编辑器使用
       3. session保存到redis缓存中,缓解数据库压力
       4. 页面数据的缓存（缓解数据库压力）
       5. 搜索（ whoosh 索引 分词）
       6. 购物车存放 redis 哈希，解决频繁数据库操作问题
       7. 前端使用ajax请求后端接口(完成购物车添加,减少,删除)
       8. 下单的时候使用事务，保证数据的安全性
       9. 高并发的库存问题的解决 （悲观锁）(select\_for\_update())
       10. 父模板的提取
       11. 图片上传到七牛云
       12. 头像上传 使用webuploader