# 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. 用户功能模块
   1. 用户账号注册功能模块
      1. 需求
2. 用户输入正确的手机号进行注册，6-16位的密码，然后再次输入密码，后台确认两次密码输入相同，然后点击获取验证码
3. 后台验证用户输入的验证码是否正确，最后点击注册
   * 1. 流程
4. 用户输入手机号，form验证手机号输入的格式，以及是否已经被注册
5. form校验输入的密码是否是6-16位，并且两次输入的密码是否一致
6. 阿里云请求发送验证码，将验证码保存到redis数据库中，生命周期为2分钟，判断用户输入的验证码是否和redis数据库中保存的验证码一致
   * 1. 难点及解决方案

1.难点一：用户表的设计

gender\_choices = (  
 (1, "男"),  
 (2, "女"),  
)  
phone = models.CharField(max\_length=11, verbose\_name="手机号")  
nick\_name = models.CharField(max\_length=50, null=True, blank=True, verbose\_name="昵称")  
password = models.CharField(max\_length=32, verbose\_name="密码")  
gender = models.SmallIntegerField(choices=gender\_choices, null=True, blank=True, verbose\_name="性别")  
birthday = models.DateField(null=True, blank=True, verbose\_name="出生日期")  
school = models.CharField(max\_length=50, null=True, blank=True, verbose\_name="学校")  
hometown = models.CharField(max\_length=50, null=True, blank=True, verbose\_name="家乡")  
address = models.CharField(max\_length=50, null=True, blank=True, verbose\_name="收货地址")

2.难点二：模型继承

通常都会在我们的模型中第一三个字段   
create\_time 创建时间   
upodate\_time 修改时间   
is\_delete 删除时间

如果每个模型都去定义，非常浪费时间，我们可以定义一个基础模型类，让所有模型都继承该模型

在项目根目录下创建一个db包文件夹，在里面创建一个文件 base\_model.py

1. from django.db import models
2. class BaseModel(models.Model):
3. create\_time = models.DateTimeField(verbose\_name="创建时间",
4. auto\_now\_add=True,
5. )
6. update\_time = models.DateTimeField(verbose\_name="更新时间",
7. auto\_now=True,
8. )
9. is\_delete = models.BooleanField(verbose\_name="是否删除",
10. default=False,
11. )
12. class Meta:
13. # 说明是一个抽象模型类，不会被迁移
14. abstract = True

Meta 中abstract用于声明该类为抽象类只能用于继承

之后所有的模型都继承BaseModel类

3.难点三：验证手机号码是否被注册和两次输入的密码是否一致

class UserRegisterForm(forms.ModelForm):  
 password1 = forms.CharField(max\_length=16,  
 min\_length=6,  
 error\_messages={  
 "required": "密码必填，请填写密码",  
 "max\_length": "密码长度不能大于16位，请重新输入",  
 "min\_length": "密码长度不能小于6位，请重新输入"  
 })  
 password2 = forms.CharField(error\_messages={  
 "required": "确认密码必填，请填写密码"  
 })  
  
 class Meta:  
 model = UserModel  
 fields = ['phone']  
 error\_messages = {  
 "phone": {  
 "required": "手机号码必须填写，请输入手机号"  
 }  
 }  
  
 # 验证两次输入的密码是否一致  
 def clean\_password2(self):  
 pwd1 = self.cleaned\_data.get("password1")  
 pwd2 = self.cleaned\_data.get("password2")  
 if pwd1 and pwd2 and pwd1 != pwd2:  
 raise forms.ValidationError("两次输入的密码不一致，请重新输入")  
 else:  
 return pwd2  
  
 # 验证手机号是否已经注册  
 def clean\_phone(self):  
 phone = self.cleaned\_data.get("phone")  
 # 判断手机号是否存在  
 res = UserModel.objects.filter(phone=phone).exists()  
 if res:  
 raise forms.ValidationError("该手机号码已经注册")  
 else:  
 return phone

1. 难点四：安全的加密方法

def set\_password(password):  
 *"""  
 定义新的加密方法，返回加密后的密码  
 """* new\_password = "{}{}".format(password, settings.SECRET\_KEY)  
 h = hashlib.md5(new\_password.encode("utf-8"))  
 new\_password = h.hexdigest()  
 return new\_password

1. 难点五：验证码的发送和验证

阿里云下载插件安装，并在后台定义方法来生成随机验证码，存放在redis中，使用阿里云封装好的方法，发送验证码。在前端写点击时间，并使用ajax触发事件调用后端写的发送验证码的方法。然后在后台写form验证验证码的合法性。Ajax代码如下：

$(**function** () {  
 {#绑定获取验证码的点击事件#}  
 $(".yzm-hq").on("click", **function** () {  
 {#console.debug(111)#}  
 {#获取手机号码的值#}  
 **var** phone = $("input[name='phone']").val();  
 {#验证手机号码的格式#}  
 **var** reg = /^1[3-9]\d{9}$/;  
 **if** (!reg.test(phone)) {  
 alert("手机号码格式错误");  
 **return**;  
 }  
 **var** that = **this**;  
 {#发送ajax到后台进行验证#}  
 $.ajax({  
 type: "post",  
 url: "{% url 'user:sendMsg' %}",  
 data: {"phone": phone, 'csrfmiddlewaretoken': "{{ csrf\_token }}"},  
 dataType: "json",  
 success: **function** (data) {  
 **if**(data.err == 0){  
 //发送成功，开始倒计时  
 **var** count = 60;  
 **var** timer = window.setInterval(**function** () {  
 //周期执行函数  
 --count;  
 {#颜色变灰#}  
 $(that).css("background-color", "gray");  
 {#禁止点击#}  
 $(that).attr("disabled","disabled");  
 **var** mes = count + "秒后重新发送";  
 $(that).val(mes);  
 **if** (count == 0){  
 {#背景颜色变回原来的#}  
 $(that).css("background-color", "#76bb2a");  
 {#可以点击#}  
 $(that).removeAttr("disabled")  
 {#内容变回原来的#}  
 $(that).val("获取验证码")  
 {#清除定时#}  
 window.clearInterval(timer)  
 }  
 },1000)  
 }  
 }  
 })  
 })  
 })  
</script>

* 1. 用户登陆功能模块
     1. 需求
        1. 用户填写用户名和密码，然后登陆
        2. 通过form验证用户名是否存在以及密码是否正确
        3. 如果正确跳转到个人中心，并保存session
     2. 流程
        1. 用户分别输入用户名和密码
        2. 通过form分别判断手机号是否存在，再判断密码是否正确，不正确把错误信息渲染到页面
        3. 在登陆的视图函数上获取数据和创建form对象，验证数据的合法性
        4. 如果不合法把错误信息渲染到页面，如果合法则跳转到个人中心，并且保存session
     3. 难点及解决方案
        1. 难点一：验证密码是否正确

class UserLoginForm(forms.ModelForm):  
 class Meta:  
 model = UserModel  
 fields = ["phone", "password"]  
 error\_messages = {  
 "phone": {  
 "required": "手机号必须填写，请输入手机号"  
 },  
 "password": {  
 "required": "密码必须填写，请输入密码"  
 }  
 }  
  
 # 验证用户名和密码  
 def clean(self):  
 data = self.cleaned\_data  
 phone = data.get("phone")  
 password = data.get("password")  
 # 判断用户是否都输入用户名和密码  
 if all([phone, password]):  
 # 根据用户输入的phone查找数据库中的用户user  
 try:  
 user = UserModel.objects.get(phone=phone)  
 except UserModel.DoesNotExist:  
 raise forms.ValidationError({"phone": "手机号/用户名不存在！"})  
 # 判断密码是否正确  
 if set\_password(password) != user.password:  
 raise forms.ValidationError({"password": "密码输入错误，登陆失败"})  
 else:  
 # 正确就将信息保存到data中  
 data["user"] = user  
 return data  
 else:  
 return data

* + - 1. 难点二：判断是否有session的装饰器

def verify\_session(old):  
 *"""  
 定义获取session的装饰器  
  
 """* def check\_login(request, \*args, \*\*kwargs):  
 if request.session.get("id") is None:  
 return redirect("user:login")  
 else:  
 return old(request, \*args, \*\*kwargs)  
  
 return check\_login

* 1. 个人资料页面
     1. 需求
        1. 回显用户的信息（如果存在）
        2. 用户可以补全个人资料，要验证输入数据的合法性
        3. 用户提交之后回显用户信息
     2. 流程
        1. 回显数据库中存在的用户信息
        2. 修改前端页面相应的form标签，type，name等属性
        3. form验证用户填写的信息的合法性
     3. 难点及解决方案
        1. 难点一：上传文件头像

# 设置静态文件的上传目录  
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')

前端页面的form

<form action="" method="post" enctype="multipart/form-data">

* + - 1. 难点二：日期回显格式

<input type="date" name="birthday" value="{{ user.birthday|date:'Y-m-d' }}" class="infor-tele" placeholder="出生日期"/>

* 1. 收获地址模块
     1. 需求
        1. 用户可以在个人中心进入收货地址管理，展示用户添加的用户地址，并且可以对其进行编辑，设为默认以及删除
        2. 用户可以新增收获地址，但是收获地址不能超过6条
        3. 在编辑页面要回显当前的收货地址信息
     2. 流程
        1. 渲染三个页面，分别是管理收获地址页面，增加收货地址页面，编辑收货地址页面
        2. 建立模型，以及form验证

class UserAddress(BaseModel):  
 *"""用户收货地址管理"""* user = models.ForeignKey(to="UserModel", verbose\_name="所属用户")  
 username = models.CharField(verbose\_name="收货人", max\_length=100)  
 phone = models.CharField(verbose\_name="收货人电话",  
 max\_length=11,  
 validators=[  
 RegexValidator(r'^1[3-9]\d{9}$', "手机号码格式错误，请输入正确的手机号")  
 ])  
 hcity = models.CharField(verbose\_name="省", max\_length=100, null=True, blank=True)  
 hproper = models.CharField(verbose\_name="市", max\_length=100, null=True, blank=True)  
 harea = models.CharField(verbose\_name="区", max\_length=100)  
 detail = models.CharField(verbose\_name="详细地址", max\_length=255)  
 isDefault = models.BooleanField(verbose\_name="是否设置为默认地址", default=False)

收货地址不能超过6条

def clean(self):  
 *"""  
 用户的收获地址不能超过6条* ***:return****:  
 """* # 获取用户id  
 user\_id = self.data.get("user\_id")  
 # 根据用户id查询他的收货地址条数  
 counts = UserAddress.objects.filter(user\_id=user\_id, isDelete=False).count()  
 if counts >= 6:  
 raise forms.ValidationError("收货地址不能超过6条")  
 # 默认地址只能有一个，判断当前isDefult是否为True 如果为真，将其他改为false  
 isDefault = self.cleaned\_data.get("isDefault")  
 if isDefault:  
 UserAddress.objects.filter(user\_id=user\_id, isDelete=False).update(isDefault=False)  
 return self.cleaned\_data

* + - 1. 在新增收货地址页面完成数据清洗，把地址写入数据库，跳转到管理收货地址页面，回显所有添加的收货地址
      2. 完成编辑收货地址，首先回显数据，同时也要对form进行合法性验证。验证通过就更新到数据库
      3. 完成设为默认和删除功能（假删除）

1. 商品功能模块
   1. 商品增删改查模块
      1. 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. 添加完成后，添加的信息能够写入数据库，并且能够通过前端渲染到页面上
      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。
         6. 把管理后台添加的数据通过后端代码渲染到前端页面上
      3. 设计要点（数据库和页面交互）

class ShopCategory(BaseModel):  
 *"""  
  
 分类名  
 分类简介  
 添加时间  
 修改时间  
 是否删除  
 """* name = models.CharField(max\_length=50, verbose\_name="分类名")  
 intro = models.CharField(max\_length=255, null=True, blank=True, verbose\_name="分类简介")  
  
 class Meta:  
 db\_table = "ShopCategory"  
 verbose\_name = "商品分类管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class ShopSpu(BaseModel):  
 *"""  
 商品SPU表  
 ID  
 名称  
 详情  
 """* name = models.CharField(max\_length=50, verbose\_name="名称")  
 detail = models.CharField(max\_length=255, null=True, blank=True, verbose\_name="详情")  
  
 class Meta:  
 db\_table = "ShopSpu"  
 verbose\_name = "商品SPU管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class ShopUnit(BaseModel):  
 *"""  
 商品单位表  
 ID  
 单位名（斤，箱）  
 添加时间  
 修改时间  
 是否删除  
 """* name = models.CharField(max\_length=50, verbose\_name="单位名（斤，箱）")  
  
 class Meta:  
 db\_table = "ShopUnit"  
 verbose\_name = "商品单位管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class ShopSku(BaseModel):  
 *"""  
 商品SKU表  
 ID  
 商品名  
 简介  
 价格  
 单位  
 库存  
 销量  
 LOGO地址  
 是否上架  
 商品分类ID  
 商品spu\_id  
 """* name = models.CharField(max\_length=50, verbose\_name="商品名称")  
 intro = models.CharField(max\_length=255, null=True, blank=True, verbose\_name="简介")  
 price = models.DecimalField(max\_digits=9, decimal\_places=2, verbose\_name="价格", default=0)  
 unit = models.ForeignKey(to=ShopUnit, verbose\_name="单位")  
 stock = models.IntegerField(verbose\_name="库存", default=0)  
 sales = models.IntegerField(verbose\_name="销量", default=0)  
 url = models.ImageField(upload\_to="shop\_sku/%Y%m/%d", verbose\_name="LOGO地址")  
 isAdded = models.BooleanField(default=True, verbose\_name="是否上架")  
 category = models.ForeignKey(to=ShopCategory, verbose\_name="商品分类")  
 spu = models.ForeignKey(to=ShopSpu, verbose\_name="商品SPU")  
  
 class Meta:  
 db\_table = "ShopSku"  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class ShopPicture(BaseModel):  
 *"""  
 图片地址  
 商品ID  
 """* url = models.ImageField(upload\_to="shop\_picture/%Y%m/%d", verbose\_name="图片地址")  
 shop\_sku = models.ForeignKey(to=ShopSku, verbose\_name="商品SKU")  
  
 class Meta:  
 db\_table = "shop\_picture"  
 verbose\_name = "商品相册"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return "图片地址"  
  
  
class LunBoModel(BaseModel):  
 *"""  
 首页轮播商品  
 ID  
 名称  
 商品SKUID  
 图片  
 排序（order）  
 添加时间  
 修改时间  
 是否删除  
 """* name = models.CharField(max\_length=50, verbose\_name="名称")  
 shop\_sku = models.ForeignKey(to=ShopSku, verbose\_name="商品SKUID")  
 picture = models.ImageField(upload\_to="lunbo/%Y%m/%d", verbose\_name="图片")  
 order = models.SmallIntegerField(verbose\_name="排序", default=0)  
  
 class Meta:  
 db\_table = "lunbo"  
 verbose\_name = "轮播图管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class Activity(BaseModel):  
 *"""  
 首页活动表  
 ID  
 名称  
 图片地址  
 url地址  
  
 """* name = models.CharField(max\_length=50, verbose\_name="活动名称")  
 picture = models.ImageField(upload\_to="activity/%Y%m/%d", verbose\_name="图片地址")  
 url = models.URLField(verbose\_name="url地址")  
  
 class Meta:  
 db\_table = "activity"  
 verbose\_name = "首页活动管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class ActivityZone(BaseModel):  
 *"""  
 首页活动专区  
 ID  
 名称  
 描述  
 排序  
 是否上架  
 """* name = models.CharField(max\_length=50, verbose\_name="活动专区名称")  
 desc = models.CharField(max\_length=255, null=True, blank=True, verbose\_name="描述")  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
 isAdded = models.BooleanField(default=True, verbose\_name="是否上架")  
  
 class Meta:  
 db\_table = "activity\_zone"  
 verbose\_name = "活动专区管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class ActivityGoods(BaseModel):  
 *"""  
 首页专区活动商品表  
 ID  
 专区ID  
 商品SKU ID  
 添加时间  
 修改时间  
 是否删除  
  
 """* zone = models.ForeignKey(to=ActivityZone, verbose\_name="活动专区ID")  
 shop\_sku = models.ForeignKey(to=ShopSku, verbose\_name="商品SKU\_ID")  
  
 class Meta:  
 db\_table = "activity\_goods"  
 verbose\_name = "首页专区活动商品管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return "首页专区活动商品"

* + 1. 难点及解决方案
       1. 在管理后台添加sku商品的时候把商品相册也添加上。活动专区里面也可以把商品添加到活动专区里面。

在admin.py里面多的一方创建一个类xxxInline，继承了admin.TubularInline,在里面定义两个属性，分别是model和extra，前者表示这个类关联到哪个模型，后者表示有几个增加的框。最后在少的一方类中增加属性inlines = [之前创建的类xxxInline]

* + - 1. 在首页活动管理中，正常情况下图片地址的属性显示出来的就是图片地址，但是希望不显示图片地址，而是图片



需要在对应模型当中定义一个方法：

def show\_picture(self):  
 return "<img style='width:80px' src='{}{}' />".format(MEDIA\_URL, self.picture)  
  
show\_picture.allow\_tags = True # 不允许转义  
show\_picture.short\_description = "LOGO" # 自定义属性名称

最后在ActivityAdmin中list\_diapaly改为：（方法名）

list\_display = ["id", "name", "show\_picture"] # 显示图片的方法名

* 1. 首页模块
     1. 需求
        1. 把数据库中的轮播商品渲染出来，点击图片可以跳转到对应的商品详情页面
        2. 把数据库中的首页活动渲染出来，点击图片可以跳转到对应的活动页面
        3. 查找出数据库中的活动专区，并把活动专区的所有商品渲染到页面上
     2. 流程
        1. 定义视图，查询得到轮播对象，将轮播对象通过字典传到前端
        2. 查询得到所有的活动，把首页活动对象通过字典传到前端渲染出来
        3. 查询得到所有的活动专区，传到前端后通过遍历得到每一个活动专区，再通过遍历逆向查询得到每一个商品SKU，把相应的sku内容渲染出来
  2. 商品详情模块
     1. 需求
        1. 得到所有的sku商品信息
        2. 每个sku商品都有一个唯一的id，作为参数传递到前端页面
        3. 将查询到的商品信息渲染到页面上
     2. 流程
        1. 通过url把sku商品的id绑定到视图
        2. 视图根据id查询出商品
        3. 把商品对象传到前端，通过前端渲染出详情页面
     3. 难点及解决方案
        1. 商品详情页的商品详情，使用传统的多行文本域编辑上传数据，页面会很难看，通过富文本编辑器，可以更好的将商品详情渲染出来
  3. 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。
        5. 根据综合，销量，价格，上新对商品进行排序
        6. 展示的sku商品太多，需要分页展示
     2. 流程
        1. 查询出所有的分类信息
        2. 在前端把所有的商品分类数据遍历展示出来
        3. 利用分类id查询到当前分类下所有的商品
        4. 遍历出所有商品，将商品渲染到相应的分类下。
        5. 对sku商品根据综合，销量，价格，最新进行排序
        6. 对价格进行排序时需要写css代码改变背景图片，以及js代码改变参数order
        7. 分页展示商品信息
     3. 难点及解决方案
        1. 对分类页面的商品进行综合排序（不排序）、销量排序、价格排序、添加时间排序。

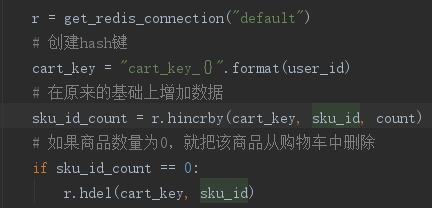
后台定义好排序规则，并把order通过参数传过来。默认order为0，即综合排序，1代表销量降序，2代表价格升序，3代表价格降序，4代表添加时间倒序。通过字典方式把order传到静态页面的a标签上使用。通过判断的方式来实现css效果

* + - 1. 点击价格排序，要对排序规则进行切换，并且改变css效果

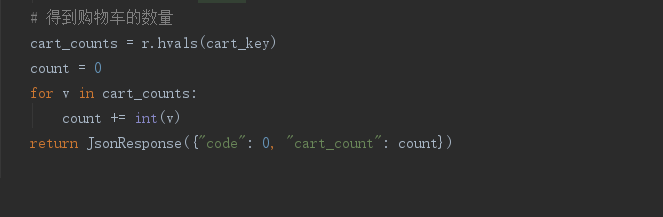
css效果可以通过判断参数order改变相应的背景图片，编写js代码来改变排序规则，即参数。如当order为2的时候，把a标签上的href属性的地址改为排序规则为3

**if** ("{{ order }}" == 2) {  
 $(".list-price").parent("a").attr("href", "{% url 'goods:cate' cate\_id 3 %}")  
}

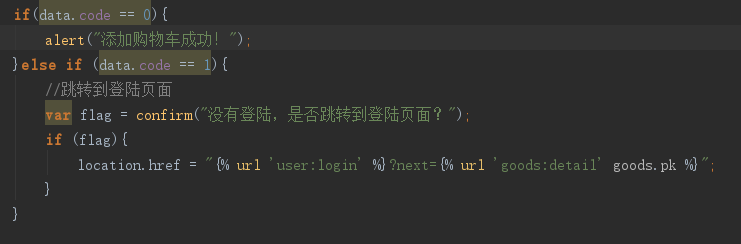
1. 购物车模块
   * 1. 需求
2. 在商品列表页，商品详情页面，用户点击可以将该商品加入到购物车
3. 在商品列表页面，用户每次点击+ 或者 - 都在购物车中添加或者减少一个商品数量
4. 在商品详情页面，用户可以指定添加多少商品到购物车
5. 通过ajax把商品的sku\_id,商品数量count提交到浏览器，后台获取数据。无刷新效果，提高用户体验度
6. 在购物车管理页面，用户可以管理购物车功能，可以选中某些商品，可以全选，全不选，可以添加商品数量和减少商品数量。当商品数量为0上删除该商品数据。
7. 由于购物车数据用户可以频繁操作，如果保存到sql数据库会增加数据库的负担，造成数据库压力，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。
   * 1. 流程
        1. 在购物车应用里面定义购物车增加的视图类，从session里面获取当前用户的id，如果没有就先登陆。然后接收前端传过来的当前商品id，以及数量。通过一系列的判断之后将数据存在redis中。



* + - 1. 从redis数据库中查询购物车的数据，并将购物车商品的总数使用JsonResponse传到前端



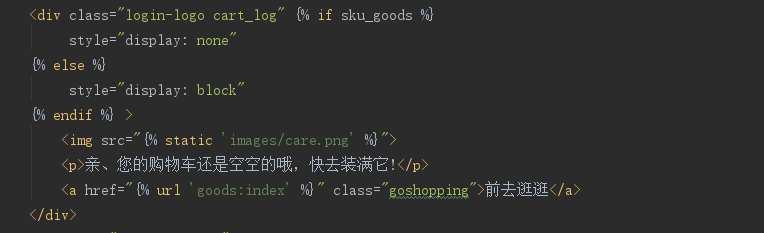
* + - 1. 前端编写加入购物车点击事件，使用ajax把后端需要的商品id，商品数量传到后端，并进行一些判断和优化（在登陆页面也要补充一段代码，判断是否带参数next，然后决定登陆之后跳转的地址）



* + - 1. 之前已经在后端编写购物车加减的程序，所以商品分类页面主要处理前端的功能。主要会用到点击事件，以及ajax，然后加上跳转到购物车的链接
      2. 最后是购物车页面。首先把页面渲染出来，然后在视图里面根据用户的id查询出购物车的数据，然后将购物车的商品传到前端



* + - 1. 在前端接收后端传过来的商品列表，通过遍历把所有购物车的信息渲染出来，如果购物车的商品列表为空，就要友好的显示购物车为空，否则不显示



* + - 1. 在前端页面编写ajax对购物车的数据进行发送，然后进行完善。定义获取总价的方法，在每次对购物车加减的时候调用。当当前商品数量为1时，再次点击将当前的li删除，后端将当前商品id删除。完善全选功能。



1. 订单模块
   * 1. 需求
        1. 用户加入购物车之后可以对订单进行结算
        2. 在结算页面回显用户的订单商品信息以及收货信息，收货信息可以点击进去进行编辑
        3. 提交订单之后进入确认支付页面，展示订单信息，不能进行编辑。点击确认支付跳转到支付平台
        4. 确认支付成功之后跳转到支付成功页面
     2. 流程
        1. 在购物车页面绑定点击事件，通过form提交商品的sku\_id
        2. 使用GET请求方式获得sku\_id，同时获得用户的id，对sku\_id遍历和判断之后，分别从数据库中查询中所有订单商品信息，以及商品数量。再根据用户的id查询出第一条收货地址信息，通过字典将商品，地址，配送方式传递到静态页面，静态页面渲染出相应数据
        3. 前端绑定点击事件，通过ajax提交数据到后台，后台使POST请求方式得到参数，然后对参数进行一系列的判断，如果参数错误就返回Json响应。参数判断完毕之后保存订单信息，创建订单，在创建订单之前要建一个保存点，后面判断库存和商品时如果出现异常就回滚到保存点。成功创建订单后删除购物车中的数据，提交事务，跳转到确认支付页面。
        4. 通过GET请求方式获得传过来的订单编号，通过用户id和订单编号查找出订单信息，通过静态页面把订单信息渲染出来
        5. 点击确认支付的同时把订单编号传到后台，后台根据订单编号查询出订单信息，使用阿里支付接口跳转到阿里支付页面，如果支付成功跳转到订单页面
     3. 难点及解决方案：
        1. 支付功能

@verify\_session  
def pay(request):  
 *"""  
 发起支付  
 """* # 接收参数  
 order\_sn = request.GET.get("order\_sn")  
 if order\_sn is None:  
 return redirect('goods:index')  
 user\_id = request.session.get("id")  
 try:  
 # 获取订单信息  
 order = Order.objects.get(order\_sn=order\_sn, user\_id=user\_id, status=0)  
 except Order.DoesNotExist:  
 return redirect('cart:index')  
 # 得到订单总金额  
 money = order.order\_money  
 # 订单描述  
 brief = "新华超市支付"  
 # 开始发起支付  
 app\_private\_key\_string = open(os.path.join(settings.BASE\_DIR, "apps/order/private\_key.txt")).read()  
 alipay\_public\_key\_string = open(os.path.join(settings.BASE\_DIR, "apps/order/ali\_public\_key.txt")).read()  
 # 创建对象  
 alipay = AliPay(  
 appid="2016092300577273",  
 app\_notify\_url=None, # 默认回调url  
 app\_private\_key\_string=app\_private\_key\_string,  
 # 支付宝的公钥，验证支付宝回传消息使用，不是你自己的公钥,  
 alipay\_public\_key\_string=alipay\_public\_key\_string,  
 sign\_type="RSA2", # RSA 或者 RSA2  
 debug=True # 默认False  
 )  
  
 # 发起支付  
 # 手机网站支付，需要跳转到https://openapi.alipay.com/gateway.do? + order\_string  
 order\_string = alipay.api\_alipay\_trade\_wap\_pay(  
 out\_trade\_no=order.order\_sn, # 订单号  
 total\_amount=str(money), # 总金额  
 subject=brief,  
 return\_url="http://127.0.0.1:8005/order/success/",  
 notify\_url=None # 可选, 不填则使用默认notify url  
 )  
 # 成功就跳转到支付链接  
 return redirect("https://openapi.alipaydev.com/gateway.do?{}".format(order\_string))

@verify\_session  
def success(request):  
 # 发起一次支付查询,查看是否支付成功  
 # 发起支付, 生成了一个地址, 跳转到支付宝地址上  
 app\_private\_key\_string = open(os.path.join(settings.BASE\_DIR, "apps/order/private\_key.txt")).read()  
 alipay\_public\_key\_string = open(os.path.join(settings.BASE\_DIR, "apps/order/ali\_public\_key.txt")).read()  
  
 # 创建对象  
 alipay = AliPay(  
 appid="2016092300577273",  
 app\_notify\_url=None, # 默认回调url  
 app\_private\_key\_string=app\_private\_key\_string,  
 # 支付宝的公钥，验证支付宝回传消息使用，不是你自己的公钥,  
 alipay\_public\_key\_string=alipay\_public\_key\_string,  
 sign\_type="RSA2", # RSA 或者 RSA2  
 debug=True # 默认False  
 )  
  
 # 获取参数  
 out\_trade\_no = request.GET.get('out\_trade\_no')  
  
 # check order status  
 paid = False  
 for i in range(4):  
 result = alipay.api\_alipay\_trade\_query(out\_trade\_no=out\_trade\_no)  
 if result.get("trade\_status", "") == "TRADE\_SUCCESS":  
 paid = True  
 break  
 else:  
 time.sleep(3)  
  
 if paid is False:  
 context = {  
 "message": "支付失败"  
 }  
 else:  
 # 支付成功  
 # 修改订单状态  
 user\_id = request.session.get("ID")  
 Order.objects.filter(order\_sn=out\_trade\_no, user\_id=user\_id, status=0).update(status=1)  
  
 # 渲染数据  
 context = {  
 "message": "支付成功"  
 }  
  
 # 支付成功之后返回的页面  
 return render(request, 'order/pay.html', context)