# 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 | Python,Django |
| 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

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

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。

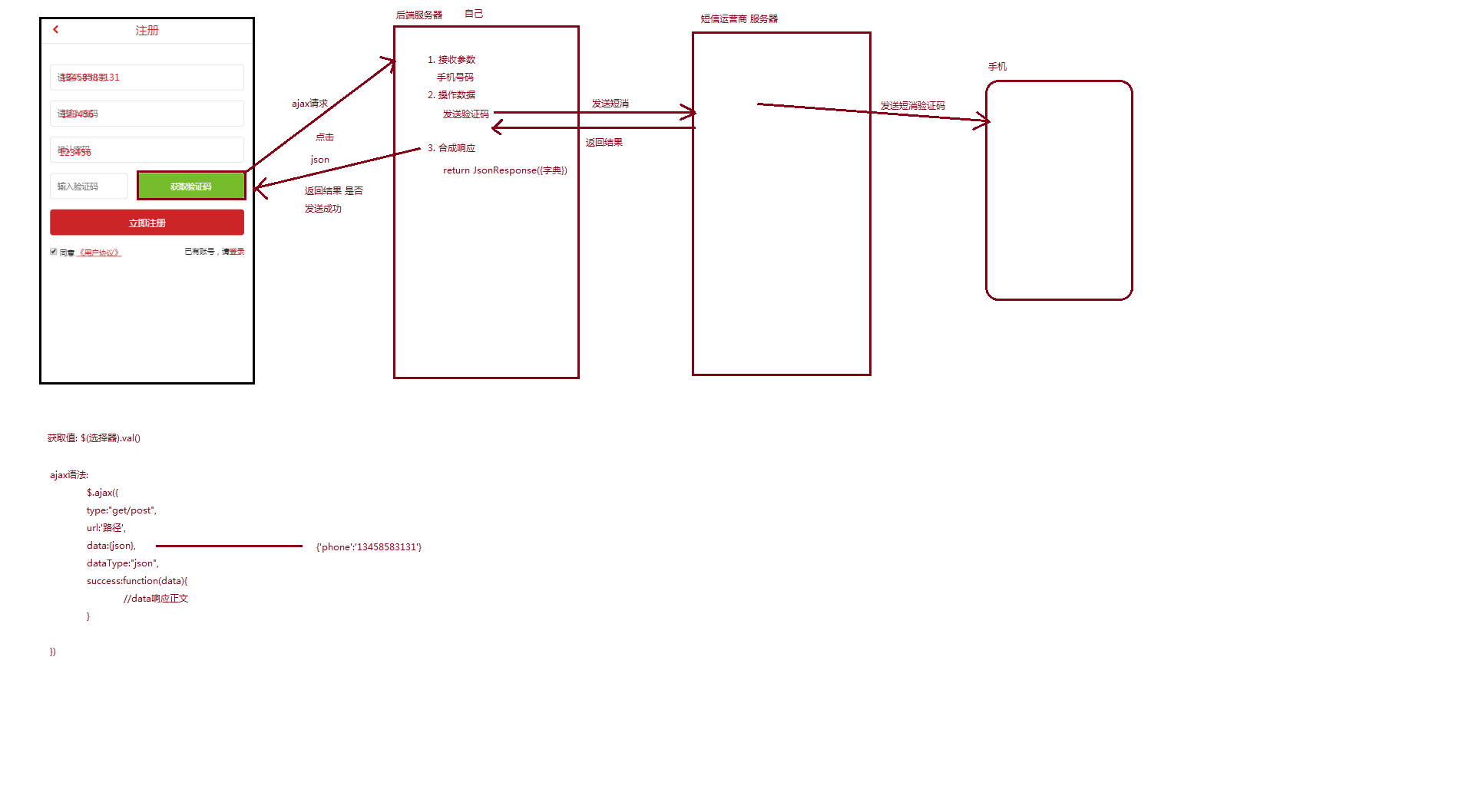
2. 阿里云响应并返回验证码，用户填写获得的验证码后点击立即注册，数据被丢到了form验证层；

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。

码已被注册，则返回错误信息，提示用户注册失败；

1. 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。
2. 流程图



* + 1. 设计要点（数据库和页面交互）

1. 数据库的基本字段：

# 用户表  
**class User(**models.Model**):** gender\_choices **= (  
 (**1, **'男')**,  
 **(**2, **'女')**,  
 **(**3, **'保密')  
 )** create\_time **=** models.DateTimeField**(**auto\_now\_add**=True**, verbose\_name**='注册时间')** update\_time **=** models.DateTimeField**(**auto\_now**=True**, verbose\_name**='更新时间')** is\_delete **=** models.BooleanField**(**default**=False**, verbose\_name**='是否删除')** username **=** models.CharField**(**max\_length**=**20, null**=True**, default**=""**, verbose\_name**='用户名')** telephone **=** models.CharField**(**max\_length**=**11,  
 verbose\_name**='手机号码'  
 )** password **=** models.CharField**(**max\_length**=**32, verbose\_name**='密码')** gender **=** models.SmallIntegerField**(**choices**=**gender\_choices, default**=**3, verbose\_name**='性别')** birthday **=** models.DateField**(**null**=True**, default**="2000-01-01"**, verbose\_name**='生日')** school **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**50, verbose\_name**='学校')** location **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**100, verbose\_name**='地址')** hometown **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**100, verbose\_name**='故乡')** head **=** models.ImageField**(**upload\_to**='head/%Y%m/%d'**, default**='head/contactqq.png'**, verbose\_name**='头像')  
  
 class Meta:** # 设置表名  
 db\_table **= "user"** # 后台管理  
 verbose\_name **= "用户表"  
  
 def \_\_str\_\_(**self**):  
 return** self.telephone

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；
  5. 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

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：对用户填写的验证码进行校验

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. 用户登录功能模块
     1. 需求
        1. 用户填写手机号和密码，点击注册按钮后，数据提交；
        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. 流程

用户填写数据；

填写的数据提交到form验证层进行基本的校验；

如果数据手机号不为空，字数没有问题，符合格式要求，并且密码字数没有问题，将数据提交给view视图函数， 对密码进行hash256加密；

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。

* + 1. 设计要点（数据库和页面交互）

1. 数据库的基本字段：

# 用户表  
**class User(**models.Model**):** gender\_choices **= (  
 (**1, **'男')**,  
 **(**2, **'女')**,  
 **(**3, **'保密')  
 )** create\_time **=** models.DateTimeField**(**auto\_now\_add**=True**, verbose\_name**='注册时间')** update\_time **=** models.DateTimeField**(**auto\_now**=True**, verbose\_name**='更新时间')** is\_delete **=** models.BooleanField**(**default**=False**, verbose\_name**='是否删除')** username **=** models.CharField**(**max\_length**=**20, null**=True**, default**=""**, verbose\_name**='用户名')** telephone **=** models.CharField**(**max\_length**=**11,  
 verbose\_name**='手机号码'  
 )** password **=** models.CharField**(**max\_length**=**32, verbose\_name**='密码')** gender **=** models.SmallIntegerField**(**choices**=**gender\_choices, default**=**3, verbose\_name**='性别')** birthday **=** models.DateField**(**null**=True**, default**="2000-01-01"**, verbose\_name**='生日')** school **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**50, verbose\_name**='学校')** location **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**100, verbose\_name**='地址')** hometown **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**100, verbose\_name**='故乡')** head **=** models.ImageField**(**upload\_to**='head/%Y%m/%d'**, default**='head/contactqq.png'**, verbose\_name**='头像')  
  
 class Meta:** # 设置表名  
 db\_table **= "user"** # 后台管理  
 verbose\_name **= "用户表"  
  
 def \_\_str\_\_(**self**):  
 return** self.telephone

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

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，将哈希后的密码与数据库内的密码进行比较是否一致。

* 1. 用户资料修改功能模块
     1. 需求

1. 用户填写昵称,性别,生日,学校,详细地址,老家地址和手机号码,点击提交保存至数据库
   * 1. 流程

用户填写数据；

填写的数据提交到form验证层进行基本的校验

* + 1. 设计要求

1. 数据库的基本字段：

# 用户表  
**class User(**models.Model**):** gender\_choices **= (  
 (**1, **'男')**,  
 **(**2, **'女')**,  
 **(**3, **'保密')  
 )** create\_time **=** models.DateTimeField**(**auto\_now\_add**=True**, verbose\_name**='注册时间')** update\_time **=** models.DateTimeField**(**auto\_now**=True**, verbose\_name**='更新时间')** is\_delete **=** models.BooleanField**(**default**=False**, verbose\_name**='是否删除')** username **=** models.CharField**(**max\_length**=**20, null**=True**, default**=""**, verbose\_name**='用户名')** telephone **=** models.CharField**(**max\_length**=**11,  
 verbose\_name**='手机号码'  
 )** password **=** models.CharField**(**max\_length**=**32, verbose\_name**='密码')** gender **=** models.SmallIntegerField**(**choices**=**gender\_choices, default**=**3, verbose\_name**='性别')** birthday **=** models.DateField**(**null**=True**, default**="2000-01-01"**, verbose\_name**='生日')** school **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**50, verbose\_name**='学校')** location **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**100, verbose\_name**='地址')** hometown **=** models.CharField**(**null**=True**, default**=""**, max\_length**=**100, verbose\_name**='故乡')** head **=** models.ImageField**(**upload\_to**='head/%Y%m/%d'**, default**='head/contactqq.png'**, verbose\_name**='头像')  
  
 class Meta:** # 设置表名  
 db\_table **= "user"** # 后台管理  
 verbose\_name **= "用户表"  
  
 def \_\_str\_\_(**self**):  
 return** self.telephone

2. 交互设计

1) 对文本框进行空字段校验，如果为空，保存失败，提示用户填写基本信息；

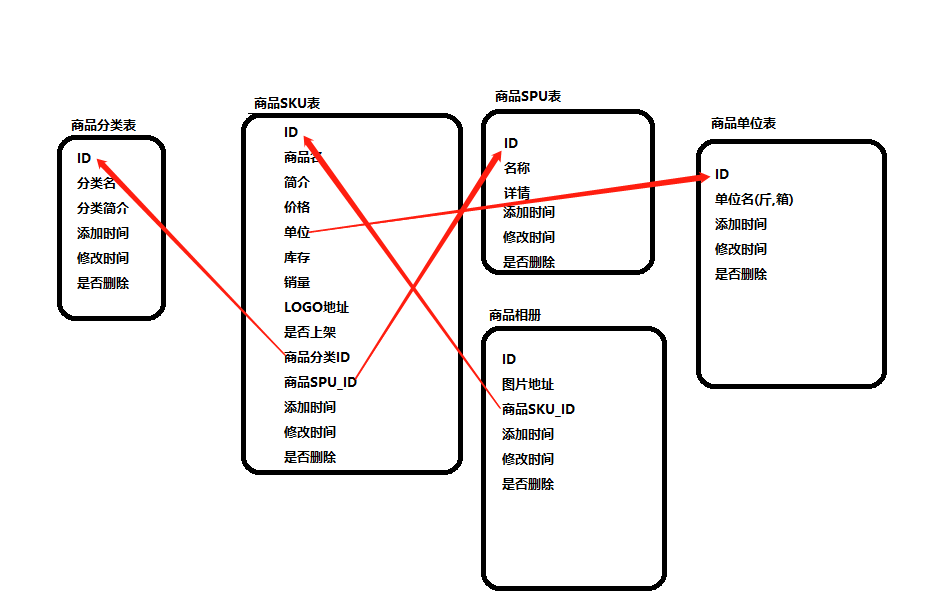
2) 对填写的字段进行基本判断，判断用户输入的字符长度是否符合规范；

3) 为文本框添加placeorder属性，提示用户需要填写的信息。

4) 为html文件中添加form标签和submit框

* + 1. 要点难点及解决方案

1. 填写生日格式必须为xxxx-xx-xx ,目前未解决
2. 因手机号是登陆的时候的凭证, 不能随便就能修改, 所有本次个人资料中的手机号只做展示,并为完成修改手机号的功能, 修改手机号的功能将在后续其他模块中实现, 需用到验证码
3. 商品功能模块
   1. 商品分类及详情展示模块
      1. 需求
4. 得到所有的分类信息
5. 遍历分类信息,将所有分类显示到页面上
6. 通过分类查询出该分类下的商品
7. 将查询到的商品信息渲染到对应的分类目录下
8. 点击某个商品后，会跳转到该商品的详情页面
9. 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
   * 1. 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。
        5. 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
        6. 视图函数有一个参数sku, 使用sku查询出商品信息，再利用商品信息查询出商品的轮播图片
     2. 设计要点（数据库和页面交互）



* + - 1. 数据库

# 商品分类表  
**class CommodityClassModel(**models.Model**):** classname **=** models.CharField**(**verbose\_name**='分类名'**, max\_length**=**20**)** classintro **=** models.TextField**(**verbose\_name**='分类简介')** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 class Meta:** # 设置表名  
 db\_table **= "CommodityClassModel"** # 后台管理  
 verbose\_name\_plural **= "商品分类管理"  
  
 def \_\_str\_\_(**self**):  
 return** self.classname

# 商品单位表  
**class CommodityUnitModel(**models.Model**):** unitname **=** models.CharField**(**verbose\_name**='单位名称'**, max\_length**=**100**)** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 class Meta:** # 设置表名  
 db\_table **= "CommodityUnitModel"** # 后台管理  
 verbose\_name\_plural **= "商品单位管理"  
  
 def \_\_str\_\_(**self**):  
 return** self.unitname

# 商品SPU表  
**class CommoditySpuModel(**models.Model**):** name **=** models.CharField**(**verbose\_name**='名称'**, max\_length**=**100**)** detail **=** RichTextUploadingField**(**verbose\_name**='详情')** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 class Meta:** # 设置表名  
 db\_table **= "CommoditySpuModel"** # 后台管理  
 verbose\_name\_plural **= "商品SPU管理"  
  
 def \_\_str\_\_(**self**):  
 return** self.name

# 商品SKU表  
**class CommoditySkuModel(**models.Model**):** name **=** models.CharField**(**verbose\_name**='名称'**, max\_length**=**50**)** intro **=** models.TextField**(**verbose\_name**='简介')** price **=** models.DecimalField**(**verbose\_name**='价格'**, decimal\_places**=**2, max\_digits**=**10**)** unit **=** models.ForeignKey**(**to**='commodity.CommodityUnitModel'**,  
 verbose\_name**='单位')** num **=** models.PositiveIntegerField**(**verbose\_name**='库存')** sellnum **=** models.PositiveIntegerField**(**verbose\_name**='销量')** logo **=** models.ImageField**(**upload\_to**='head/%Y%m/%d'**, verbose\_name**='logo地址')** is\_putaway **=** models.BooleanField**(**verbose\_name**='是否上架')** class\_id **=** models.ForeignKey**(**to**='commodity.CommodityClassModel'**,  
 verbose\_name**='商品分类ID')** spu\_id **=** models.ForeignKey**(**to**='commodity.CommoditySpuModel'**,  
 verbose\_name**='商品SPU\_ID')** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 class Meta:** # 设置表名  
 db\_table **= "CommoditySkuModel"** # 后台管理  
 verbose\_name\_plural **= "商品SKU管理"  
  
 def \_\_str\_\_(**self**):  
 return** self.name

# 商品相册表  
**class CommodityPictureModel(**models.Model**):** picture **=** models.ImageField**(**upload\_to**='head/%Y%m/%d'**, verbose\_name**='图片地址')** sku\_id **=** models.ForeignKey**(**to**='commodity.CommoditySkuModel'**,  
 verbose\_name**='商品SKU\_ID')** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 class Meta:** # 设置表名  
 db\_table **= "CommodityPictureModel"** # 后台管理  
 verbose\_name\_plural **= "商品相册管理"**

* + 1. 要点难点及解决方案
       1. 表的设计,要特别注意是一对多还是多对多,外键的配置位置是在多个一方的表中
       2. 展示页面需全部从数据库读取,需熟练使用连表查询
       3. 显示在界面上的内容需要过滤,并且数据都不止一条,需要用到模型类名.objects.filter(is\_delete=False),之后直接在html 文件中使用表的查询显示数据,要注意区别查询出来的数据时列表还是对象
       4. 商品列表页跳转商品详情页需要传递参数,这里使用商品数据库的ID作为唯一标识传递,并且在应用urls里要使用正则+小括号截取传入的ID,在视图函数中接收ID,操作数据库显示商品详情
  1. 商品首页模块
     1. 需求
        1. 渲染首页的轮播部分.
        2. 渲染首页不规则展示推荐信息
        3. 查找出商品活动信息,渲染出活动及活动下的商品信息
     2. 流程
        1. 渲染轮播部分

查询出所有轮播信息，将轮播信息传递到首页上，首页遍历出所有的轮播图片，渲染到首页的轮播框内

* + - 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. 设计要点(数据库和页面交互)
       1. 数据库

# 首页轮播  
**class BannerModel(**models.Model**):** 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ID'**, default**=**0**)** Commodity\_sku **=** models.ForeignKey**(**to**='commodity.CommoditySkuModel'**, verbose\_name**='商品SKU')** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 class Meta:** verbose\_name **= "首页轮播管理"** verbose\_name\_plural **=** verbose\_name  
  
 **def \_\_str\_\_(**self**):  
 return** self.name

# 首页活动  
**class ActivityModel(**models.Model**):** 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**)** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 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  
**class ActivityZoneModel(**models.Model**):** 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"**, default**=False)** goods\_sku **=** models.ManyToManyField**(**to**="commodity.CommoditySkuModel"**, verbose\_name**="商品")** addtime **=** models.DateTimeField**(**verbose\_name**='添加时间'**, auto\_now\_add**=True)** updatetime **=** models.DateTimeField**(**verbose\_name**='修改时间'**, auto\_now**=True)** is\_delete **=** models.BooleanField**(**verbose\_name**='是否删除'**, default**=False)  
  
 def \_\_str\_\_(**self**):  
 return** self.title  
  
 **class Meta:** verbose\_name **= "首页活动管理"** verbose\_name\_plural **=** verbose\_name

* + 1. 要点难点及解决方案

难点1:首页活动专区图片显示

解决方案:这里是一个多对多的连表查询,要特别注意多对多查询的时候是跟一对多的查询方式一样,这里需要使用的正向查询,首先在view中得到活动专区model所有对象,在html中遍历该对象,展示所有的活动专区,然后通过遍历得到的子对象查询该子对象下对应的所有SKU商品信息,此处需要使用到循环内套循环,代码如下

<!-- 特色专区-->  
{% **for act** in **activity** %}  
 <**div class="sq-title"**>  
 <**img src="**{% **static 'images/ts.png'** %}**" width="26"**/>  
 {{ **act**.**title** }}  
 </**div**>  
 <**ul data-am-widget="gallery" class="am-gallery pro-list am-avg-sm-2 am-avg-md-2 am-avg-lg-4 am-gallery-default"**>  
 {% **for foo** in **act**.**goods\_sku**.**all** %}  
 <**li**>  
 <**div class="am-gallery-item"**>  
 <**a href="**{% **url 'commodity:商品详情' foo**.**id** %}**" class=""**>  
 <**img src="**{{ **MEDIA\_URL** }}{{ **foo**.**logo** }}**"**/>  
 <**h3 class="am-gallery-title"**>{{ **foo**.**name** }}</**h3**>  
 <**div class="am-gallery-desc"**>￥52</**div**>  
 </**a**>  
 </**div**>  
 </**li**>  
  
 {% **endfor** %}  
 </**ul**>  
{% **endfor** %}

* 1. 商品搜索模块
     1. 需求
        1. 点击首页最上方搜索框跳转到搜索界面
        2. 点击商品列表页上方搜索框可直接搜索
        3. 采用whoosh全文搜索引擎,haystack全文搜索框架,jieba中文分词包实现需求
     2. 流程
        1. 在虚拟环境中依次安装需要的包

pip install django-haystack

pip install whoosh

pip install jieba

* + - 1. 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

INSTALLED\_APPS **= [  
 'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'haystack'**, # 全文检索框架  
 # 添加子应用  
 **'user.apps.UserConfig'**,  
 **'order.apps.OrderConfig'**,  
 **'shopping\_cart.apps.ShoppingCartConfig'**,  
 **'commodity.apps.CommodityConfig'**,  
 **'ckeditor'**, # 添加ckeditor富文本编辑器  
 **'ckeditor\_uploader'**, # 添加ckeditor富文本编辑器文件上传部件  
**]**

* + - 1. 配置全文检索框架haystack支持whoosh搜索引擎

# 全文检索框架的配置  
HAYSTACK\_CONNECTIONS **= {  
 'default': {** # 配置搜索引擎 修改成自己配置的搜索引擎  
 **'ENGINE': 'utils.whoosh\_cn\_backend.WhooshEngine'**,  
 # 配置索引文件目录  
 **'PATH':** os.path.join**(**BASE\_DIR, **'whoosh\_index')**,  
 **}**,  
**}**# 当添加、修改、删除数据时，自动生成索引  
HAYSTACK\_SIGNAL\_PROCESSOR **= 'haystack.signals.RealtimeSignalProcessor'**

* + - 1. .在应用目录（commodity）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

# 导入全文检索框架索引类  
**from** haystack **import** indexes  
**from** commodity.models **import** CommoditySkuModel  
  
**class CommoditySkuModelSearchIndex(**indexes.SearchIndex, indexes.Indexable**):** # 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  
 text **=** indexes.CharField**(**document**=True**, use\_template**=True)** # 获取检索对应对的模型  
 **def get\_model(**self**):  
 return** CommoditySkuModel  
  
 # 设置检索需要使用的查询集  
 **def index\_queryset(**self, using**=None):  
 """Used when the entire index for model is updated."""  
 return** self.get\_model**()**.objects.filter**(**is\_delete**=False)**

* + - 1. 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/commodity/commodityskumodel\_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

# 指定搜索的字段  
{{object.name}}  
{{object.intro}}  
{{object.price}}  
{{object.spu\_id.name}}  
{{object.spu\_id.detail}}

* + - 1. 使用命令生成索引文件

python manage.py rebuild\_index

* + - 1. 在主路由中配置全文检索框架使用的子路由

urlpatterns **= [** url**(r'^admin/'**, admin.site.urls**)**,  
 # 添加第三方插件  
 url**(r'^ckeditor/'**, include**("ckeditor\_uploader.urls"))**,  
 # 全文搜索框架  
 url**(r'^search/'**, include**('haystack.urls'**, namespace**='search'))**,  
 # 添加子路由  
 url**(r'^commodity/'**, include**('commodity.urls'**, namespace**='commodity'))**,  
 url**(r'^order/'**, include**('order.urls'**, namespace**='order'))**,  
 url**(r'^shopping\_cart/'**, include**('shopping\_cart.urls'**, namespace**='shopping\_cart'))**,  
 url**(r'^user/'**, include**('user.urls'**, namespace**='user'))**,  
 url**(r'^$'**, index**)**,  
**]**

* + - 1. 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/

<**form action="**{% **url 'search:haystack\_search'** %}**" method="get"**>  
 <**div class="cate-search"**>  
 <**input type="text" name="q" class="cate-input" value="**{{ **query** }}**"  
 placeholder="**{{ **query**|**default**:**'请输入您要的搜索的产品关键词'** }}**"**/>  
 <**input type="submit" class="cate-btn" value=""**/>  
 </**div**>  
</**form**>

点击进行提交时，会通过haystack搜索数据

* + - 1. 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

{% **if query** %}  
 <**ul class="list-pro"**>  
 {% **for result** in **page** %}  
 <**li**>  
 <**a href="**{% **url 'commodity:商品详情' result**.**object**.**pk** %}**"**><**img  
 src="**{{ **MEDIA\_URL** }}{{ **result**.**object**.**logo** }}**"  
 class="list-pic"**/></**a**>  
 <**div class="shop-list-mid" style="width**: 65%;**"**>  
 <**div class="tit"**><**a  
 href="**{% **url 'commodity:商品详情' result**.**object**.**pk** %}**"**>{{ **result**.**object**.**name** }}</**a**></**div**>  
 <**div class="am-gallery-desc"**>{{ **result**.**object**.**price** }}</**div**>  
 <**p**>销量：{{ **result**.**object**.**sellnum** }}{{ **result**.**object**.**unit** }}</**p**>  
 </**div**>  
 </**li**>  
 {% **empty** %}  
 <**li**>没有找到您搜索的产品!</**li**>  
 {% **endfor** %}  
 </**ul**>  
{% **endif** %}

* + - 1. 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/

-- 复制 whoosh\_backend.py 改名为 whoosh\_cn\_backend.py

-- 根目录新建一个utils包保存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。

-- 更改词语分析类

# 使用结巴  
**from** jieba.analyse **import** ChineseAnalyzer

查找:

analyzer**=stemmingAnalyzer()**

改为:

analyzer**=**ChineseAnalyzer**()**

-- 修改settings.py文件中的配置项

# 全文检索框架的配置  
HAYSTACK\_CONNECTIONS **= {  
 'default': {** # 配置搜索引擎 修改成自己配置的搜索引擎  
 **'ENGINE': 'utils.whoosh\_cn\_backend.WhooshEngine'**,  
 # 配置索引文件目录  
 **'PATH':** os.path.join**(**BASE\_DIR, **'whoosh\_index')**,  
 **}**,  
**}**

-- 重新创建索引数据

python manage.py rebuild\_index

-- 重写搜索就可以对中文进行分词了

* + 1. 设计要点
    2. 要点难点及解决方案

1. 常见面试问题