# 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. 用户注册功能模块

2.1.1 需求

1. 用户填写基本信息：正确的手机号，11位的密码（手机号），确认密码。

2.填写完毕后点击获取验证码；

3. 获取验证码以后，填写验证码，点击立即注册。

* 1. 流程

1.用户填写手机号码-----》密码----》确认密码---》验证手机号/验证密码长度/两次输入密码是否一致----》格式正确且手机号码有效就获取验证码，验证通过就跳转到登录界面——》否则 返回相应的错误信息

* 1. 设计要点（数据库和页面交互）

1.Create\_time 创建时间 datetimeField auto\_now\_add（注册的时间）

Update\_time 更新时间 datetimeField auto\_now （修改信息的时间）

Is\_delete 是否删除 booleanField default=Flase（假删除）

以上三个字段 每个表都需要。

Username 昵称 CharField

Tel 手机号 CharField 用于注册

Password 密码 CharField (哈希加密)

sex 性别 booleanField default=1 (1:男， 2:女， 3:保密)

Birth\_time dateField 生日

School charfield 学校

Location charfield 地址

Hometown charfield 家乡

注：除了手机号，密码和确认密码、验证码 不能为空意外，其他都可以为空（null=true,blank=true)，性别默认为1.

2.手机号、密码、确认密码、验证码的错误信息提示

手机号的格式 、密码的位数，确认密码的一致性，验证码是否正确，且都不能为空

* 1. 要点难点及解决方案

问题点：

1.注册时提交方式为post，需要表单验证，而给我们的模板中没有form，需要自己添加上

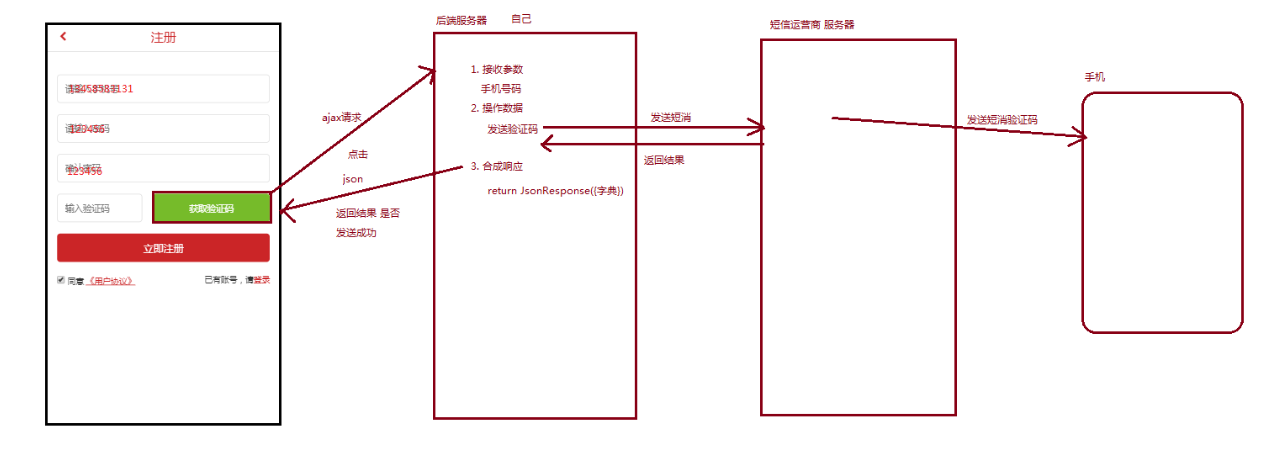
1. <form action="" method="post" class="form-horizontal">...</form>
2. {% csrf\_token %}

2.哈希加密（循环加密+加盐）

import hashlib  
from supermarket.settings import SECRET\_KEY  
  
def set\_password(password):  
 # 循环加密+加盐  
 for v in range(1000):  
 pwd\_str='{},{}'.format(password,SECRET\_KEY)  
 h = hashlib.md5(pwd\_str.encode('utf-8'))  
 return h.hexdigest()

1. 验证码的获取
2. 用户在项目前端，输入手机号，然后点击【获取验证码】，将手机号发到post到后台。 2.后台验证手机号是否合法，是否已被占用，如果通过验证，则生成验证码，并通过运行脚本，让短信运营商向该手机号，发送该验证码，如果没通过验证，则返回错误信息 3.用户收到短信验证码以后，再次将所有信息post到后台。 4.后台验证各个数据，通过验证则完成实名制认证，如果没通过则返回错误信息

5.手机短信验证：流程图



1. ajax与form

Ajax：

'csrfmiddlewaretoken': '{{ csrf\_token }}'



Form：{% csrf\_token %}

1. 验证手机格式：

^1[3-9]\d{9}

1. 生成随机字符串验证码时注意转换：

yz\_cade=''.join(str(random.randint(0,9))for r in range(1,6))

9.单独验证密码和确认密码是否一致：定义了一个clean方法

1. 用户登录功能模块
   1. 需求
2. 用户填写账号和密码，点击登录，跳转到个人中心
3. 用户输入的密码和数据库中的密码会进行验证，如，正确则跳转到个人中心，不对，则提示用户名或者密码错误
   1. 流程

用户填写账号和密码，用form表单格式提交到后台进行验证：用户名直接跟数据库中的用户名进行对比，密码则经MD5加密和再和数据库中的密码进行比较

* 1. 设计要点（数据库和页面交互）

1. 与注册功能使用同一个表
2. 账号名和密码的错误信息提示:如不能为空，账号或密码输入不正确
   1. 要点难点及解决方案
3. 用户输入的为明文密码，而数据库的密码时经过哈希加密的

再验证密码时候正确的时候，先加密用户输入的密码，再和数据库的密码进行比较

1. **商品增删改查模块**

5.1需求

* + - 1. 商家登录后台系统；
      2. 添加||删除||修改||查看 商店或商品信息；
      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。
    1. 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。
    2. **设计要点**

数据库

建表

# 商品类型表  
class GoodsType(BaseModel):  
 type\_name = models.CharField(max\_length=50, verbose\_name='商品类名')  
 sort\_detail = models.TextField(verbose\_name='分类详情')  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
  
 def \_\_str\_\_(self):  
 return self.type\_name  
  
 class Meta:  
 db\_table = 'goodstype'  
 verbose\_name = '商品类型表管理'  
 verbose\_name\_plural = verbose\_name

# 单位表  
class Unit(BaseModel):  
 unit\_name = models.CharField(max\_length=50, verbose\_name='单位名')  
  
 def \_\_str\_\_(self):  
 return self.unit\_name  
  
 class Meta:  
 db\_table = 'unit'  
 verbose\_name = '单位表管理'  
 verbose\_name\_plural = verbose\_name

# 商品SPU表  
class GoodsSPU(BaseModel):  
 spu\_name = models.CharField(max\_length=50, verbose\_name='SPU名')  
 spu\_detail = RichTextUploadingField(verbose\_name='SPU简介')  
  
 def \_\_str\_\_(self):  
 return self.spu\_name  
  
 class Meta:  
 db\_table = 'goodsspu'  
 verbose\_name = '商品SPU表管理'  
 verbose\_name\_plural = verbose\_name

# 商品SKU表  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(max\_length=50, verbose\_name='SKU')  
 brief = models.TextField(verbose\_name='SKU简介')  
 price = models.DecimalField(verbose\_name='价格',  
 max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 )  
 unit = models.CharField(max\_length=10, verbose\_name='单位')  
 stock = models.IntegerField(verbose\_name='库存')  
 sales\_val = models.IntegerField(verbose\_name='销量')  
 logo = models.ImageField(verbose\_name='封面图片', upload\_to='goods/%Y%m/%d')  
  
 res = ((False, "下架"), (True, "上架"))  
 is\_on\_sale = models.BooleanField(verbose\_name='是否上架', choices=res, default=False)  
 goods\_type = models.ForeignKey(to='GoodsType', verbose\_name='商品分类id')  
 goods\_spu = models.ForeignKey(to='GoodsSPU', verbose\_name='商品SPU\_id')  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 db\_table = 'goodssku'  
 verbose\_name = '商品SKU表管理'  
 verbose\_name\_plural = verbose\_name

* + 1. **难点及问题点**

1. 商品详情功能模块
   1. 需求
2. 用户点击商品时，自动跳转到商品详情页面：
3. 会出现商品相册并且又轮播效果
4. 显示商品得名字价格
5. 能选择商品的购买数量
6. 点击收藏，图片变红
7. 此页面下有2个子模块：商品详情和评论
   1. 流程

创建表注意点：

1. 先创建一的一方，再创建多的一方
2. 每个表所需要的字段必须考虑清楚。

将每一个表添加到后台管理，方便增删该查

渲染页面

* 1. 设计要点（数据库和页面交互）

1. 商品类别表----

# 商品类型表  
class GoodsType(BaseModel):  
 type\_name = models.CharField(max\_length=50, verbose\_name='商品类名')  
 sort\_detail = models.TextField(verbose\_name='分类详情')  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")

2.商品的SKU表---

# 商品SKU表  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(max\_length=50, verbose\_name='SKU')  
 brief = models.TextField(verbose\_name='SKU简介')  
 price = models.DecimalField(verbose\_name='价格', max\_digits=9,  
 decimal\_places=2,  
 default=0,)  
 unit = models.CharField(max\_length=10, verbose\_name='单位')  
 stock = models.IntegerField(verbose\_name='库存')  
 sales\_val = models.IntegerField(verbose\_name='销量')  
 logo = models.ImageField(verbose\_name='封面图片', upload\_to='goods/%Y%m/%d')  
  
 res = ((False, "下架"), (True, "上架"))  
 is\_on\_sale = models.BooleanField(verbose\_name='是否上架', choices=res, default=False)  
 goods\_type = models.ForeignKey(to='GoodsType', verbose\_name='商品分类id')  
 goods\_spu = models.ForeignKey(to='GoodsSPU', verbose\_name='商品SPU\_id')

1. 商品的SPU表---设计包括 1.ID 2.名称 3.详情

# 商品SPU表  
 class GoodsSPU(BaseModel):  
 spu\_name = models.CharField(max\_length=50, verbose\_name='SPU名')  
 spu\_detail = RichTextUploadingField(verbose\_name='SPU简介')

商品的相册表-

# 商品相册  
class GoodsPhoto(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")

# 单位表  
class Unit(BaseModel):  
 unit\_name = models.CharField(max\_length=50, verbose\_name='单位名')

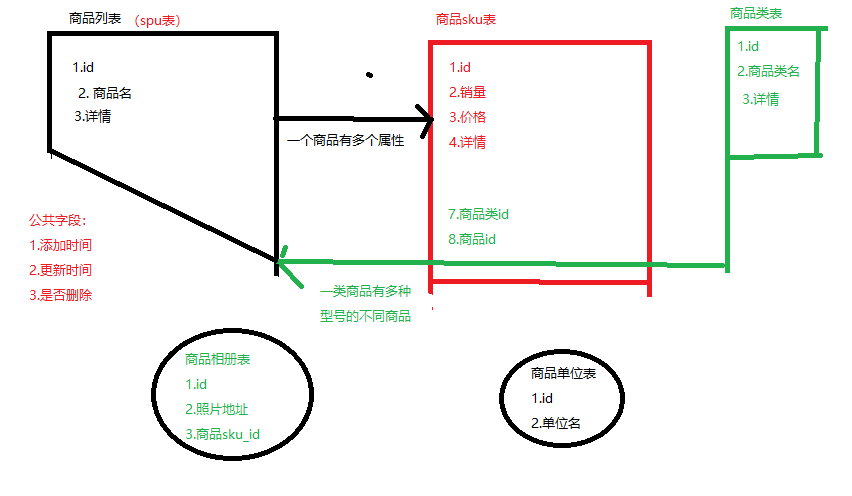
# 商品轮播表  
class Carousel(BaseModel):  
 name = models.CharField(max\_length=200, verbose\_name='轮播活动名')  
 img\_url = models.ImageField(verbose\_name='轮播图片地址', upload\_to='carousel/%Y%m/%d')  
 order = models.SmallIntegerField(verbose\_name="排序",  
 default=0,  
 )  
  
 goods\_sku = models.ForeignKey(to="GoodsSKU", verbose\_name="商品SKU")

# 首页活动表  
class Activity(BaseModel):  
 act\_name = models.CharField(max\_length=50, verbose\_name='活动名')  
 img\_url = models.ImageField(verbose\_name='活动图片地址', upload\_to='activity/%Y%m/%d')  
 url\_address = models.URLField(verbose\_name='活动的url地址', max\_length=200)

# 首页活动专区  
class Activity\_Area(BaseModel):  
 name = models.CharField(verbose\_name='活动专区名称', max\_length=100)  
 describe = models.TextField(verbose\_name='专区的描述')  
  
 res = ((False, "下架"), (True, "上架"))  
 is\_on\_sale = models.BooleanField(verbose\_name='是否上架', choices=res, default=False)  
 goods\_sku = models.ManyToManyField(to="GoodsSKU", verbose\_name="商品")

4.

如图：



* 1. 要点难点及解决方案

1. 一对多查询

多————》一：正向查询：多对象.关联字段

一————》多：逆向查询：一对象.模型类名\_set.all()

1. 在渲染模板的时候，注意哪些时父模板没有的，也要复制到父模板中，不然在页面中显示不出来
2. 在遍历查询结果时，注意关联查询中最后的all，没有括号。  
   如：good\_sku.goodsphoto\_set.all
3. 在需要显示轮播图片的时候必须先将之前在设置文件中修改好的路径（{{MEDIA\_URL}}）写上
4. 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

1. 分析清楚各个表之间的对应关系

**（**1）分析商品类别与商品SKU表的关系（1对多关系）

（2）商品SPU表与商品SKU表的关系（1对多）

（3）商品SKU与商品相册的关系（1对多）

（4）商品单位与商品SKU表（1对多）

1. 商品列表功能模块
   1. 需求
2. 页面刚加载的时候，显示的商品只显示排序第一的分类下的商品
3. 点击哪个类型就显示相应的商品
4. 可以按照 综合（pk） 销量 价格的升降 添加时间排序 并且是对应分类下的商品
   1. 流程

查询所有的分类信息————》将分类信息展示到页面上————》跟据分类的不同查询出每个分类下的商品信息——————》查询出所有商品——————》渲染页面

* 1. 设计要点（数据库和页面交互）

数据库：同商品详情模块下的

页面交互：点击哪个类型就可以查询出该类型下的所有商品，并且可以在该类型下按销量 综合

价格的升降 创建时间查询；当某个类型下没有商品时，显示没有商品

* 1. 要点难点及解决方案

1. 排序
2. 明确需要传入的参数：

类型id：cate\_id

综合（pk）

销量（sale\_val）

价格（price）

创建时间（create\_time）

1. 先做分类列表
2. 先查询出所有的类型
3. 再传入的id查询出对应的类型
4. 再根据查询出来的类型查询出该类型下的所有商品

3）再传入第二参数order，查询出某类型下 按照综合 销量 价格 创建时间排序的商品列表

1.自定义一个排序规则order\_rule = ['pk', '-sales\_val', 'price', '-price', '-create\_time']

刚好order\_rule的索引为0.1.2.3.4

注意：传参

url(r'^type/(?P<cate\_id>\d\*)\_{1}(?P<order>\d?)\.html$',TypeView .as\_view(), name='商品类型'),

再写html代码的时候也要注意 反向解析时也需要传入对应的id，不能少。如：

{% url '商品首页:商品类型' category.pk order %}"

在html中进行if判断时，注意符号的位置。错误示例

<li {% if cate\_id == category.pk %} class="current">{% endif %}

4）

1. **商品排序功能模块**

7.1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。

7.2流程

* + - 1. 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. 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. 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. 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. 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. 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。

7.3设计要点

建表

# 商品类型表  
class GoodsType(BaseModel):  
 type\_name = models.CharField(max\_length=50, verbose\_name='商品类名')  
 sort\_detail = models.TextField(verbose\_name='分类详情')  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
  
 def \_\_str\_\_(self):  
 return self.type\_name  
  
 class Meta:  
 db\_table = 'goodstype'  
 verbose\_name = '商品类型表管理'  
 verbose\_name\_plural = verbose\_name

# 商品SKU表  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(max\_length=50, verbose\_name='SKU')  
 brief = models.TextField(verbose\_name='SKU简介')  
 price = models.DecimalField(verbose\_name='价格',  
 max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 )  
 unit = models.CharField(max\_length=10, verbose\_name='单位')  
 stock = models.IntegerField(verbose\_name='库存')  
 sales\_val = models.IntegerField(verbose\_name='销量')  
 logo = models.ImageField(verbose\_name='封面图片', upload\_to='goods/%Y%m/%d')  
  
 res = ((False, "下架"), (True, "上架"))  
 is\_on\_sale = models.BooleanField(verbose\_name='是否上架', choices=res, default=False)  
 goods\_type = models.ForeignKey(to='GoodsType', verbose\_name='商品分类id')  
 goods\_spu = models.ForeignKey(to='GoodsSPU', verbose\_name='商品SPU\_id')  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 db\_table = 'goodssku'  
 verbose\_name = '商品SKU表管理'  
 verbose\_name\_plural = verbose\_name

7.4要点及难点

* + - 1. 1： 对价格进行排序

价格既可以升序排列，又可以降序排列，两个排序方式都需要提供给用户，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. 购物车功能模块
   1. 购物车商品添加

1.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。

2.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。

3.在未登录得情况下添加购物车需要验证登录

* 1. 流程

1. 在html上用ajax post请求得方式，自定义一个变量，sku\_id，data

sku\_id表示商品id，并且在input标签上自定义一个属性sku\_id;data理由ajax的token，商品数量count，sku\_id。

1. 在添加时需要验证登录
2. 判断添加数量是否为整数
3. 判断商品存在与否
4. 判断库存是否充足
5. 连接redis，保存到redis中，减小服务器压力
   1. 设计要点
6. 获取用户id，商品id，添加到购物车的数量
7. 判断添加数量是否为整数
8. 判断商品是否存在
9. 判断库存是否充足
10. 连接redis
11. 使用hash对象保存数量
12. 在html上验证登录
    1. 要点及难点解决方案
13. 验证数据的合法性-3个

解决方案：

1. 验证添加数量、库存、商品，每一个都要考虑到，

2.验证登录，登录完成后，需要跳转回之前验证登录的页面

解决方案

1. 在验证登录时，保存上一个请求地址
2. 如果地址存在就跳回原来的网址

3.保存数据用redis，提高用户的体验

1.用redis对象进行保存数据，但是要注意hget得到的数据是二进制数据需要转换

而hincrby这个方法刚刚好有三个参数：hincrby(cart\_key,sku\_id,count)

* 1. **购物车显示**

7.5.1.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；

7.5.2.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。

7.5.3设计要点

1. 从seeion中获取当前用户得id
2. 从redis中获取商品得信息
3. 准备空列表，保存商品
4. 根据购物车中商品id查出商品得详细信息
5. 删除redis过器得数据

7.5.4 问题点及解决方案

1.全选相关问题：勾选全部才是全选，勾选多少则总价显示多少

利用ajax判断是否全选，再计算总价

2. 选中不同的商品，如何精确的计算出价格；

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. 购物车商品删除
     1. 需求

1.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。

2.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. 在未登录得情况下添加购物车需要验证登录
   * 1. 流程

1.在html上用ajax post请求得方式，自定义一个变量，sku\_id，data

sku\_id表示商品id，并且在input标签上自定义一个属性sku\_id;data理由ajax的token，商品数量count，sku\_id。

2.在添加时需要验证登录

3.判断添加数量是否为整数

4.判断商品存在与否

5.判断库存是否充足

6.连接redis，保存到redis中，减小服务器压力

* + 1. 设计要点

1.获取用户id，商品id，添加到购物车的数量

2.判断添加数量是否为整数

3.判断商品是否存在

4.判断库存是否充足

5.连接redis

6.使用hash对象保存数量

7.在html上验证登录

* + 1. 问题点及解决方案

1.验证数据的合法性-3个

解决方案：

1.验证添加数量、库存、商品，每一个都要考虑到，

2.验证登录，登录完成后，需要跳转回之前验证登录的页面

解决方案

1.在验证登录时，保存上一个请求地址

2.如果地址存在就跳回原来的网址

3.保存数据用redis，提高用户的体验

1.用redis对象进行保存数据，但是要注意hget得到的数据是二进制数据需要转换

而hincrby这个方法刚刚好有三个参数：hincrby(cart\_key,sku\_id,count)

1. **订单模块**

10.1 确认订单

10.1.1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。
    1. 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。
    2. **设计要点**

数据库

建表

class Transport(BaseModel):  
 *"""运算方式"""* name = models.CharField(max\_length=50, verbose\_name="运输方式")  
 price = models.DecimalField(max\_digits=9, decimal\_places=2, default=0, verbose\_name='运费')  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Meta:  
 verbose\_name = "运输方式管理"  
 verbose\_name\_plural = verbose\_name

class Order(BaseModel):  
 *"""订单基本信息表"""* order\_status\_choices = (  
 (0, "未支付"),  
 (1, "已支付"),  
 (2, "已发货"),  
 (3, "未评价"),  
 (4, "已完成"),  
 (5, "退发货"),  
 (6, "取消订单"),  
 )  
 user = models.ForeignKey(to="users.Users", verbose\_name="用户")  
 order\_sn = models.CharField(max\_length=64, verbose\_name="订单编号")  
 goods\_total\_price = models.DecimalField(max\_digits=9, decimal\_places=2, default=0, verbose\_name="商品总金额")  
 transport\_price = models.DecimalField(max\_digits=9, decimal\_places=2, verbose\_name="运费")  
 transport = models.CharField(max\_length=50, verbose\_name="运输方式")  
 username = models.CharField(max\_length=50, verbose\_name="收货人姓名")  
 phone = models.CharField(max\_length=11, verbose\_name="收货人电话号码")  
 address = models.CharField(max\_length=250, verbose\_name="收货人地址")  
 order\_price = models.DecimalField(max\_digits=9, decimal\_places=2, default=0, verbose\_name="订单总金额")  
 order\_status = models.SmallIntegerField(choices=order\_status\_choices, default=0, verbose\_name="订单状态")  
 payment = models.ForeignKey(to="Payment", null=True, blank=True, verbose\_name="支付方式")  
  
 pay\_time = models.DateTimeField(verbose\_name="支付时间", null=True, blank=True)  
 deliver\_time = models.DateTimeField(verbose\_name="发货时间", null=True, blank=True)  
 finish\_time = models.DateTimeField(verbose\_name="完成时间", null=True, blank=True)  
  
 def \_\_str\_\_(self):  
 return self.order\_sn  
  
 class Meta:  
 verbose\_name = "订单管理"  
 verbose\_name\_plural = verbose\_name

class OrderGoods(BaseModel):  
 *"""订单商品详情表"""* order = models.ForeignKey(to="Order", verbose\_name="订单ID")  
 goods\_sku = models.ForeignKey(to="commodity.GoodsSKU", verbose\_name="订单商品ID")  
 price = models.DecimalField(max\_digits=9, decimal\_places=2, verbose\_name="商品价格")  
 count = models.SmallIntegerField(verbose\_name="订单商品数量")  
  
 def \_\_str\_\_(self):  
 return "{}:{}".format(self.order.order\_sn, self.goods\_sku.sku\_name)  
  
 class Meta:  
 verbose\_name = "订单商品管理"  
 verbose\_name\_plural = verbose\_name

class Payment(BaseModel):  
 name = models.CharField(max\_length=50, verbose\_name="支付方式")  
 brief = models.CharField(max\_length=200, verbose\_name="说明")  
 logo = models.ImageField(upload\_to="payment/%Y", verbose\_name="支付LOGO")  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Meta:  
 verbose\_name = "支付方式管理"  
 verbose\_name\_plural = verbose\_name

2.快递方式以下拉框得形式出现

* + 1. **难点及问题点**

1. 常见面试问题