课前：

接下来由monkey老师带着大家继续学习我们电商项目的实战，我主要会给大家分享电商商品模块的技术难点和业务以及订单秒杀的业务与技术难点。具体课程安排：

商品模块-秒杀：

秒杀系统-商品详细页多级缓存实战一

秒杀系统-商品详细页多级缓存实战二

秒杀系统-商品详细页多级缓存实战三

订单模块-秒杀：

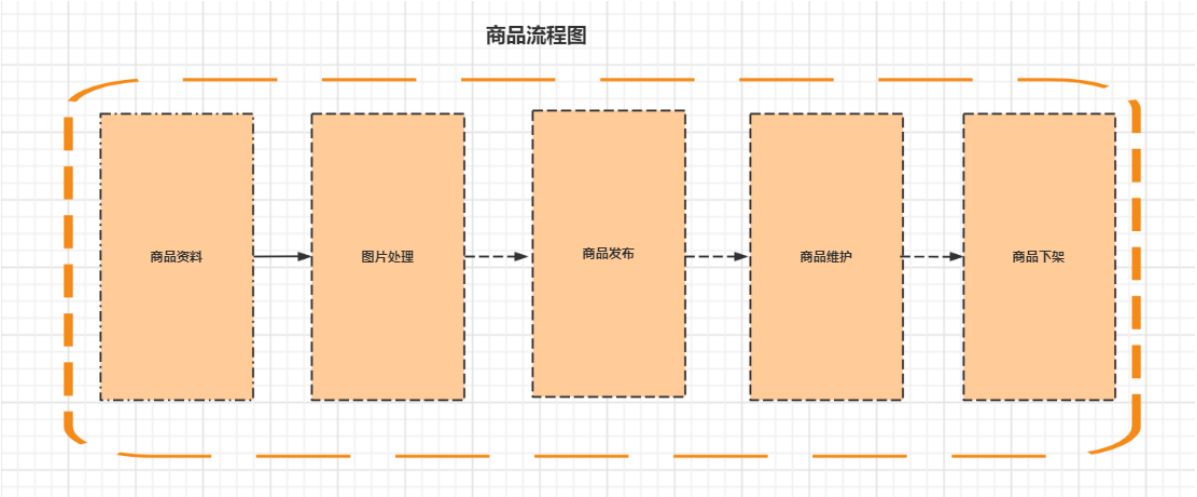
秒杀系统-订单交易全链路优化实战一

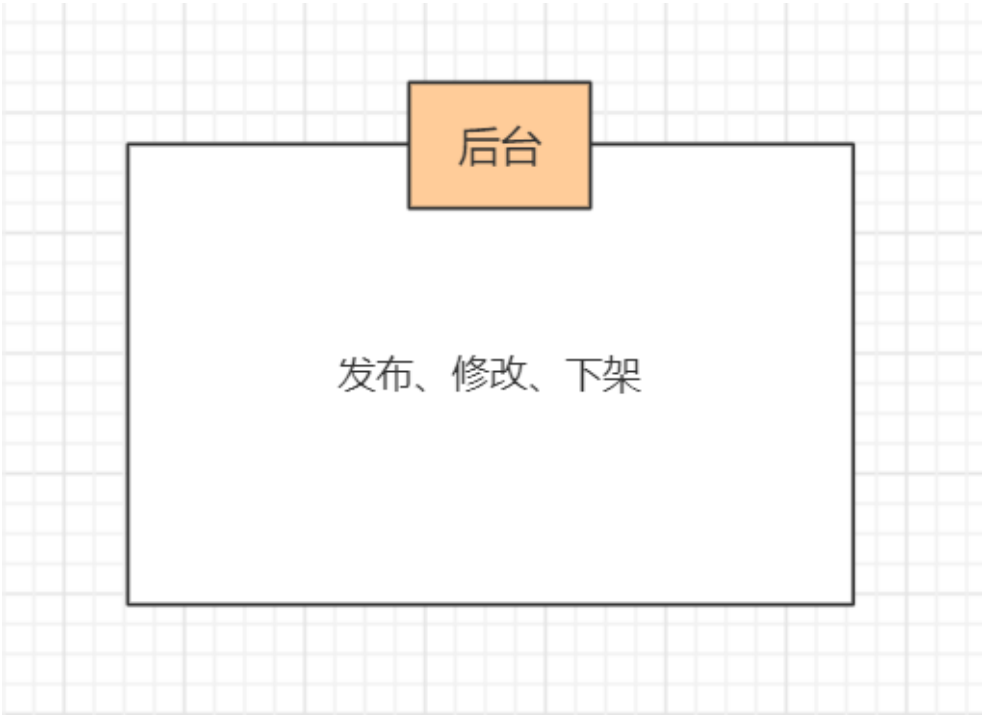
秒杀系统-订单交易全链路优化实战二

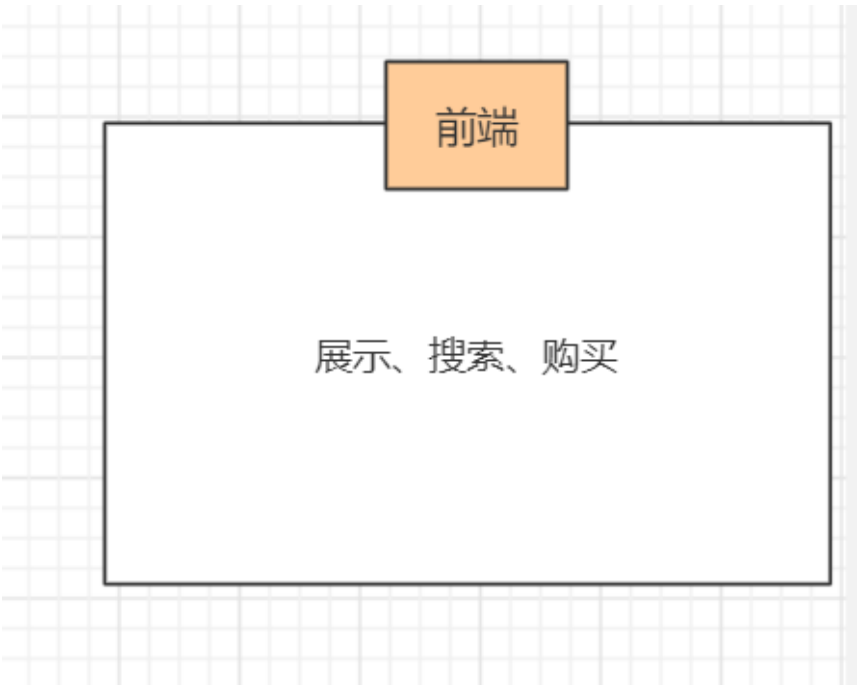
秒杀系统-订单交易全链路优化实战三-楼兰

商品模块业务场景介绍：

商品模块业务详解







一个商品属于哪个类目？属于哪个品牌？属于什么型号？参数有哪些？

二、 商品模块技术难点

表的设计：打开游览器访问京东详细页

问题：

商品这块的数据库如何更好的设计，商品详细页显示这么多信息，是一张表还是多张表更好了？

这个问题到底是一张表还是多张表，我们判断依据是什么？我们判断商品详细页里面显示的这些信息他们的关系。通过他们的关系，我们才能知道到底是设计一张表还是多张表。

一张表：

如果是一张表存储所有数据的话，那么查询是非常方便的，这是其优点，但是你会发现存储的时候是不是很麻烦。不通类型不同大小不通商品等等都不一样，那这样的一张表设计起来实在是太复杂了。

多张表：

如果是多张表的话业务更加清晰，维护起来也更加方便，但是你会发现查询好像会非常的复杂，一个商品页面我们需要查很多的表和数据。

我们来分析下刚才我们看到了一个商品有很多的名词、比如分类、商品属性、商品评价
那他们的关系是?

分析：

分类：一个商品属于某个分类

商品属性：一个商品有N个属性，不同的商品有不同的属性

商品品牌：一个商品属于一个品牌，一个品牌下面有多个商品

商品参数：不同的商品参数不一样

商品活动：一个商品可以有多个活动。

评价：一个商品有多条评论

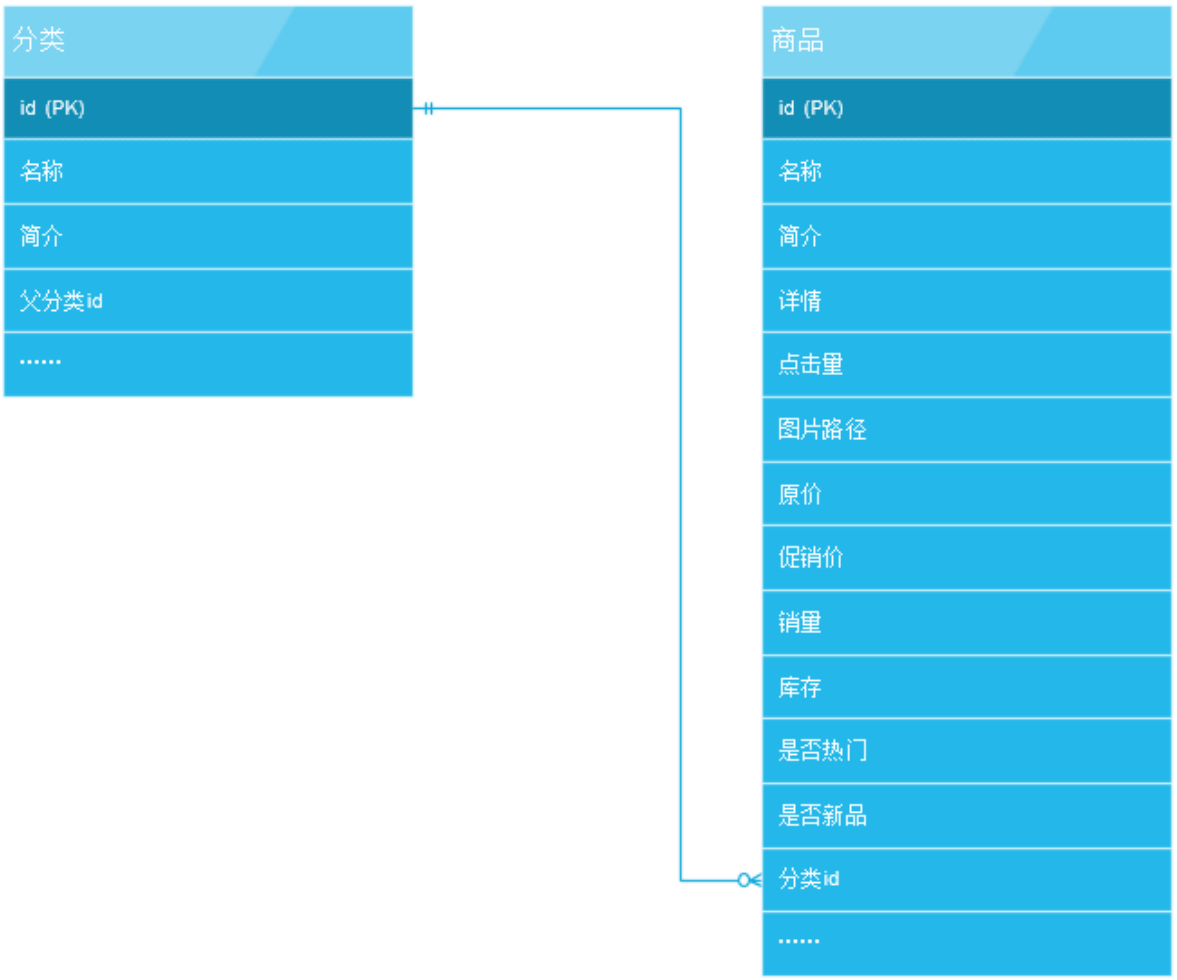
所以综合上述所述，根据商品详细页的显示，我们正确的方式是根据不通的数据类型按不通的表进行存储。

我们现在已经看到商品详细页需要显示这么多内容，那这些内容是的作用是什么，以及他为什么需要这么多数据。带着这样的问题，Monkey老师给大家讲下商品的发展这块的演化。

为什么商品需要分类?

我们知道商品是有不通类型的，比如有吃的 比如有穿的比如还有其他的用的。不通的商品用途不一样。我们一开始就可以按分类来进行划分我们的商品，这个就有点像我们去看论坛的分类是一样的。

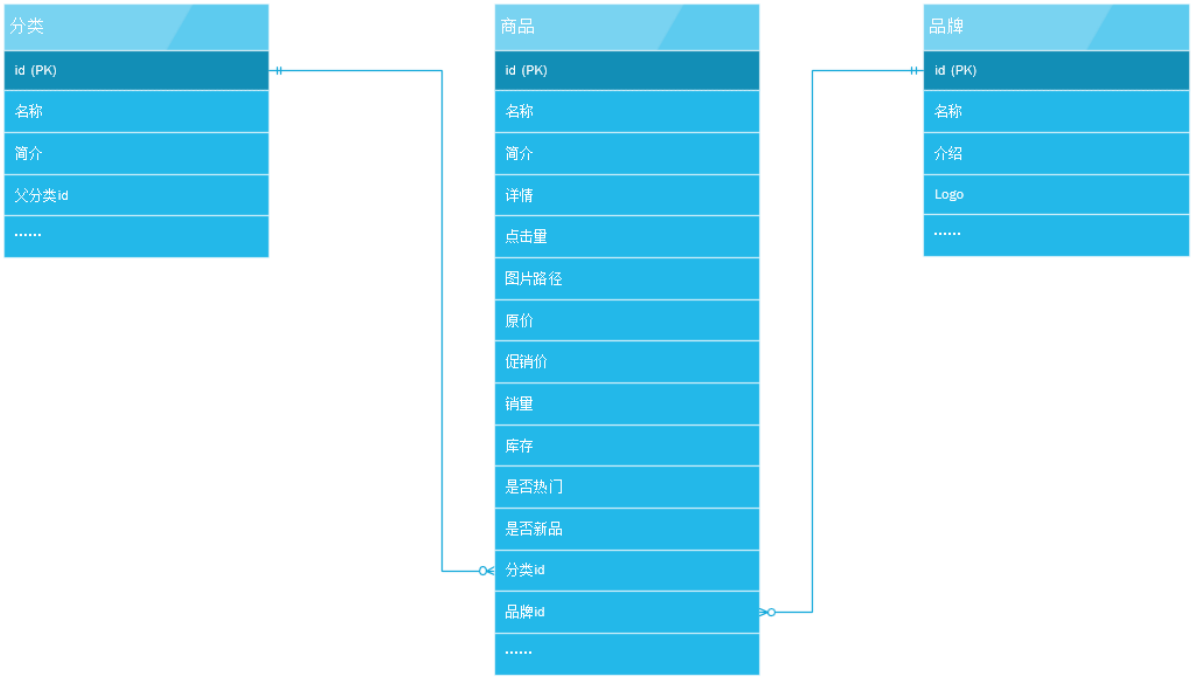
第一个版本：商品+分类



问题：此时有什么问题？：

目前这个方案有什么问题了？我们慢慢发现一个问题，只有分类并不能适应所有的需求，比如nike鞋和nikeT恤，用户可能希望先看nike的所有商品，这个模型就不能满足。我们想在这个关系中，加入“品牌”概念

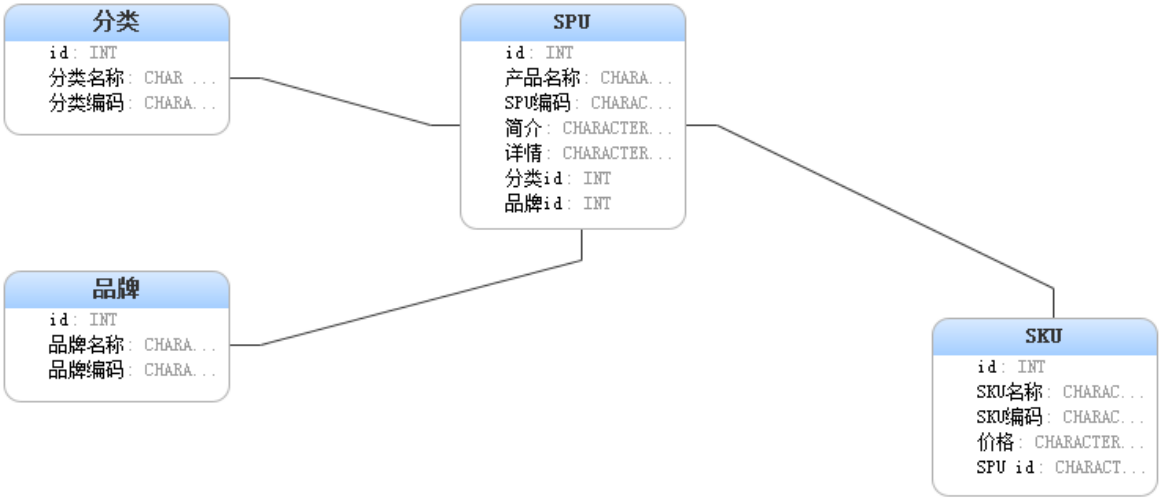
第二个版本：商品+分类+品牌



这样基本用户可以在首页上通过分类或者品牌找到自己想要的商品，也可以直接查看热门的商品和新上架的商品。

问题：此时有什么问题?

但是问题也来了，用户在进入分类后，展示在用户面前的是很多很多商品，用户希望再通过筛选查询出更接近他目标的商品?

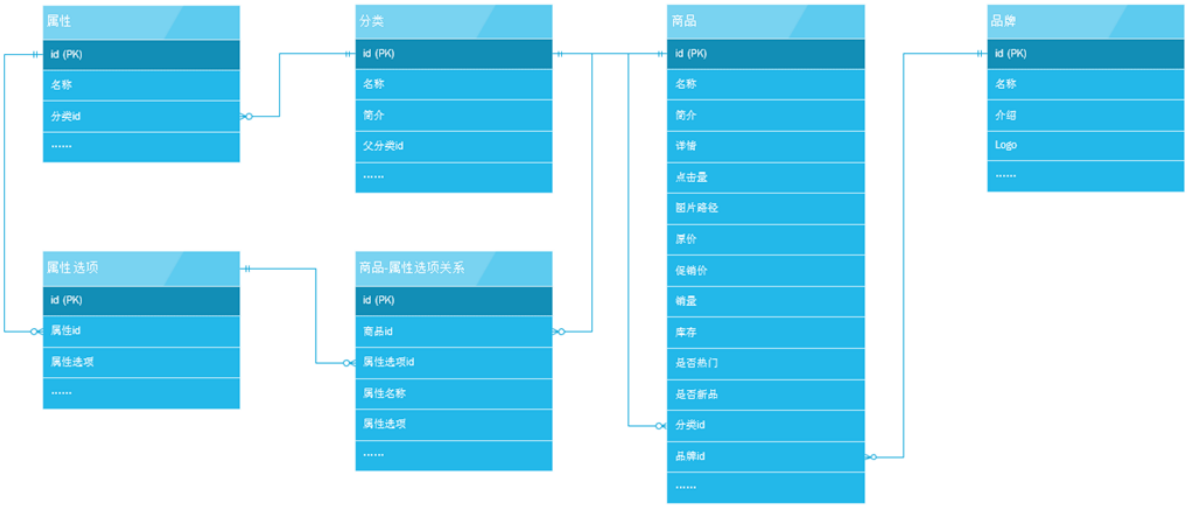


加入属性：

于是优秀的产品设计师，设计出了类似这样的UI：

第三个版本：商品+分类+品牌+属性

用户可以通过这些筛选条件进一步缩小自己的目标范围，那么问题又来了，这样的产品需求排在程序员面前，怎么去实现它？经过分析，我们找出了一个方法，我们知道商品之间的属性可能存在着较大的差别，比如牛仔裤它有版型、腰型、裤长等属性；而电脑它有CPU、显卡等属性，各类商品的属性是不同的。再进一步想，休闲裤也版型、腰型、裤长等属性；台式电脑或者笔记本电脑都有CPU、显卡等属性。所以我们得出：**一个分类对应若干属性，而一个属性，对应若干属性选项**，而一个具体商品又对应若干属性选项（例如具体一条牛仔裤，他的裤长：7分，裤型：直筒）。有点绕，仔细品味一下。

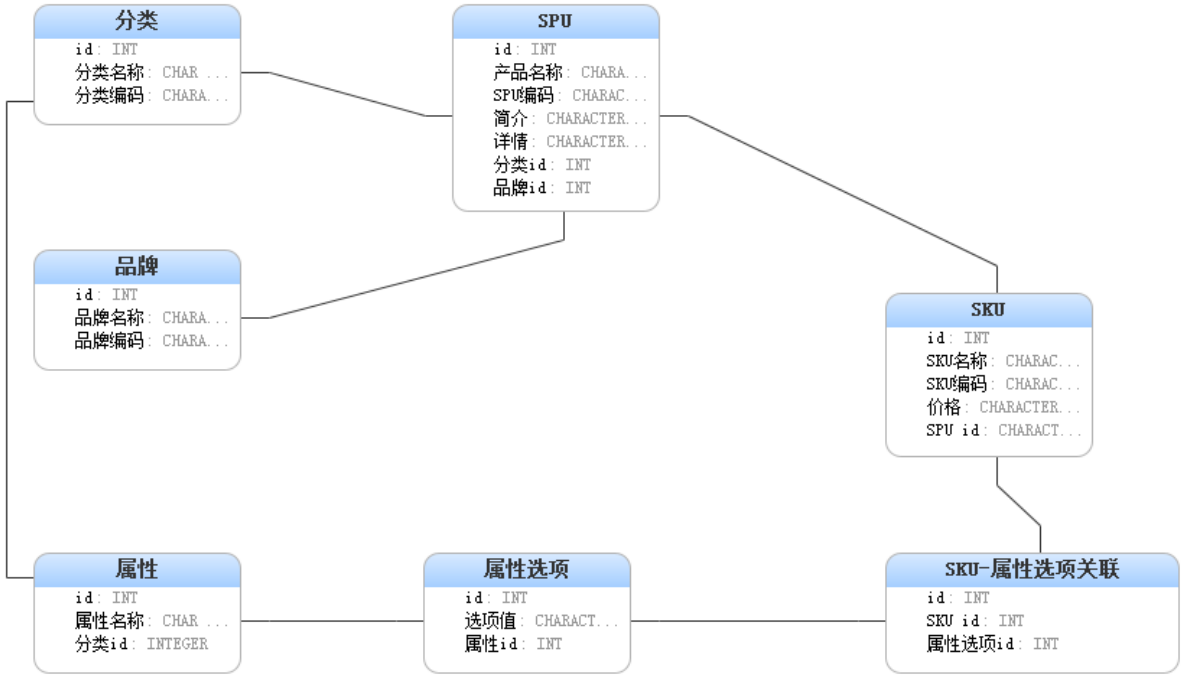


从图上可以看出，分类和属性的关系（例如：“牛仔裤”分类下有裤型、裤长、版型等属性）、属性和属性选项的关系（例如：裤长属性有长款、九分裤、七分裤的选项）、商品和属性选项的关系（例如某条牛仔裤的裤长是7分裤）。至此，我们知道一个商品的分类、品牌以及它有什么属性和对应的属性值。那么通过筛选条件，自然就可以查询出指定的商品。这里特别说一句，价格也是属性，不要设想用商品表中的价格字段去做计算。这不利于查询也增加了复杂度，让商家编辑人员用属性来设置并保证他的正确性。

品牌： 魅族（MEIZU）

商品名称：魅族17 Pro	商品编号：100012881850	商品毛重：0.59kg	商品产地：中国大陆
CPU型号：骁龙865	运行内存：12GB	机身存储：256GB	存储卡：不支持存储卡
摄像头数量：后置四摄	后摄主摄像素：6400万像素	前摄主摄像素：2000万像素	主屏幕尺寸（英寸）：6.6英寸
分辨率：全高清FHD+	屏幕比例：19.1~19.5:9	屏幕前摄组合：开孔屏	充电器：10V/3A；20V/1.8A；5V/3A
热点：无线充电，NFC，5G，超高...	游戏性能：发烧级	屏占比：≥92%	操作系统：Android(安卓)
游戏配置：游戏性能模式			

这个页面展示商品的所有信息，按照之前的设计好像都可以满足。但是我们似乎感觉错过了什么，在图上右边我们发现该商品当前的颜色和尺寸，并且允许用户可以选择其他的颜色和尺寸。这给我们带来了疑惑，这里的“颜色”和“尺寸”是什么，一件商品的不同颜色不同尺寸是算一个商品还是多个商品。



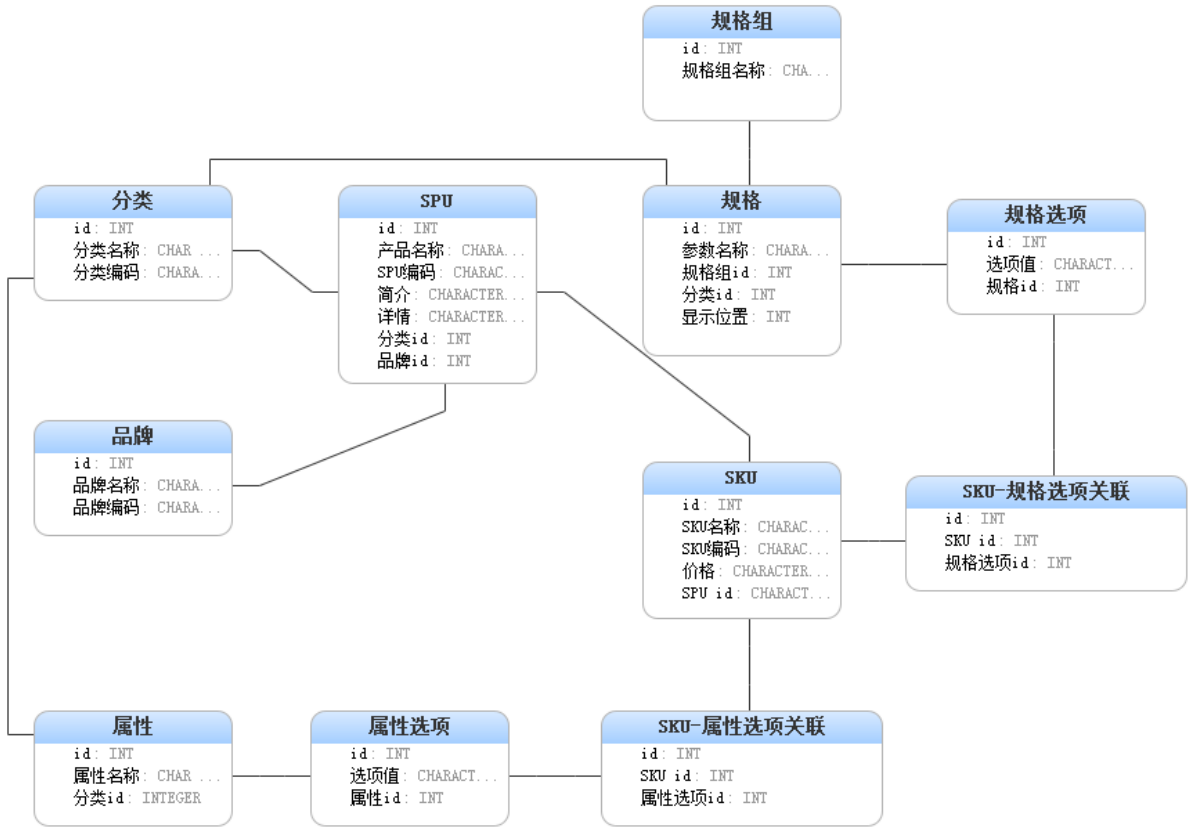
为什么要加入规格：

第四个版本：商品+分类+品牌+属性+规格

经过思考后，我们发现我们混淆了两个概念——“商品”和“货品”。不同规格的货品作为独立的商品。比如一条裤子的有L尺寸、M尺寸、一个U盘有16G还是32G的，都是同样的货品，不同规格的商品。可以认为货品和商品是一对多的关系。弄清了这个概念，处理这个需求就容易多了，这里的“颜色”、“尺寸”我们就作为“规格”来处理，而红色、黑色；L号、M号我们视为规格的选项或者说规格值。一件货品对应若干规格，而具有某一规格值的货品就是商品。

spu：iphone12　sku：金色64 iphone12

好了，现在好像差不多了。基于这个模型可以满足基本的商品搜索、展示的需求。搜索引擎也可以根据这个模型数据生成对应的商品索引，达到准确搜索的目的。商品模块还会和其他模块一起协作，比如用户系统、订单系统、支付系统等。一般情况下我们会把商品业务独立出来做成“商品中心”的服务，集中处理商品查询、更新、发布等业务，支撑其他业务。



商品的搜索：

搜索引擎elasticsearch

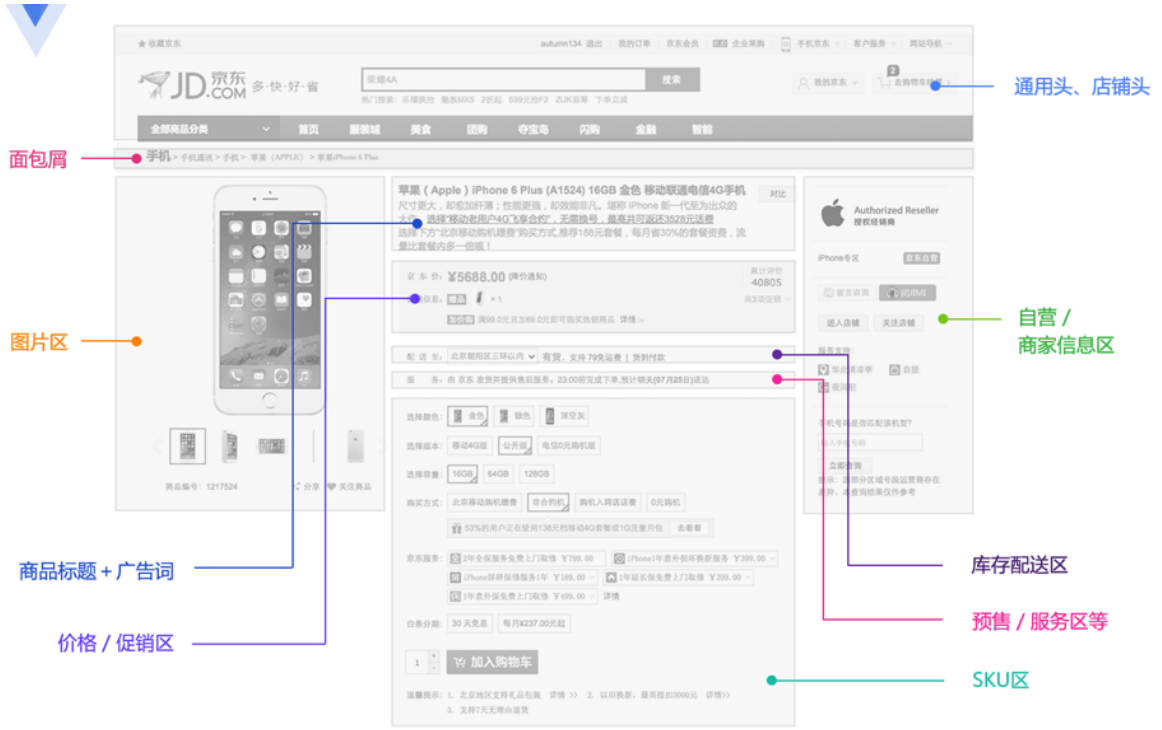
商品的展示：

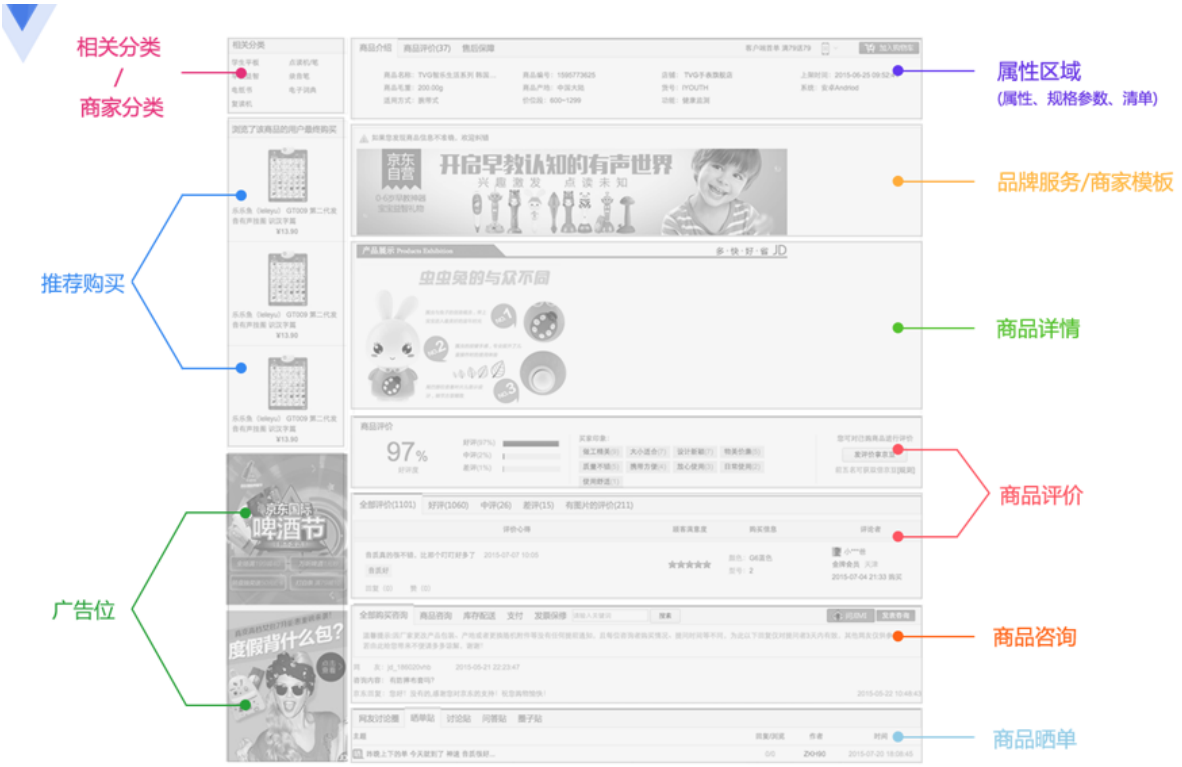
三、商品模块展示技术难点

商品详情页是展示商品详细信息的一个页面，承载在网站的大部分流量和订单的入口。京东商城目前有通用版、全球购、闪购、易车、惠买车、服装、拼购、今日抄底等许多套模板。各套模板的元数据是一样的，只是展示方式不一样。目前商品详情页个性化需求非常多，数据来源也是非常多的，而且许多基础服务做不了的都放我们这，因此我们需要一种架构能快速响应和优雅的解决这些需求问题。因此我们重新设计了商品详情页的架构，主要包括三部分：商品详情页系统、商品详情页统一服务系统和商品详情页动态服务系统；商品详情页系统负责静的部分，而统一服务负责动的部分，而动态服务负责给内网其他系统提供一些数据服务。

商品详情页前端结构

前端展示可以分为这么几个维度：商品维度(标题、图片、属性等)、主商品维度（商品介绍、规格参数）、分类维度、商家维度、店铺维度等；另外还有一些实时性要求比较高的如实时价格、实时促销、广告词、配送至、预售等是通过异步加载。





SPU： Standard Product Unit （标准化产品单元）,SPU是商品信息聚合的最小单位，是一组可复用、易检索的标准化信息的集合，该集合描述了一个产品的特性。

SKU： Stock keeping unit(库存量单位) SKU即库存进出计量的单位（买家购买、商家进货、供应商备货、工厂生产都是依据SKU进行的），在服装、鞋类商品中使用最多最普遍。 例如纺织品中一个SKU通常表示：规格、颜色、款式。SKU是物理上不可分割的最小存货单元。

单品页流量特点

热点少，各种爬虫、比价软件抓取。

3.1、压测测试

jmeter模板

图灵微商城-Jmeter压测.jmx
32.01KB

1、换数据库

2、分库分表

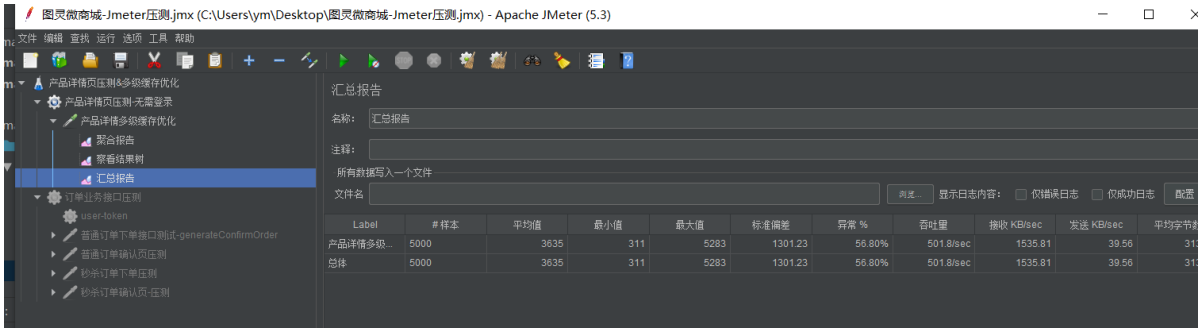
3.2、后台

```
/**
 * 获取商品详情信息
 *
 * @param id 产品ID
```

```
 */
public PmsProductParam getProductInfo(Long id) {
 PmsProductParam productInfo = portalProductDao.getProductInfo(id);
 if (null == productInfo) {
 return null;
 }
 FlashPromotionParam promotion = flashPromotionProductDao.getFlashPromotion(id);
 if (!ObjectUtils.isEmpty(promotion)) {
 productInfo.setFlashPromotionCount(promotion.getRelation().get(0).getFlashPromotionCount());
 productInfo.setFlashPromotionLimit(promotion.getRelation().get(0).getFlashPromotionLimit());
 productInfo.setFlashPromotionPrice(promotion.getRelation().get(0).getFlashPromotionPrice());
 productInfo.setFlashPromotionRelationId(promotion.getRelation().get(0).getId());
 productInfo.setFlashPromotionEndDate(promotion.getEndDate());
 productInfo.setFlashPromotionStartDate(promotion.getStartDate());
 productInfo.setFlashPromotionStatus(promotion.getStatus());
 }
 return productInfo;
}
```

压测结果：

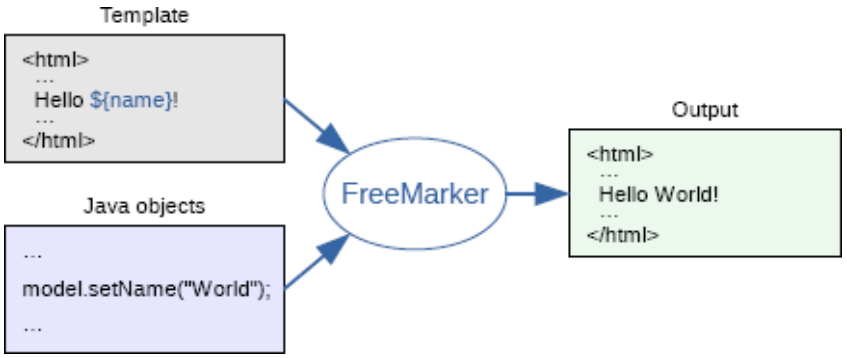
5000并发



四、静态化处理

FreeMarker 是一款模板引擎：即基于模板和数据源生成输出文本（html网页，配置文件，电子邮件，源代码）的通用工具。它是一个 java 类库，最初被设计用来在MVC模式的Web开发

框架中生成HTML页面，它没有被绑定到Servlet或HTML或任意Web相关的东西上。也可以用于非Web应用环境中。
模板编写使用FreeMarker Template Language(FTL)。使用方式类似JSP的EL表达式。模板中专注于如何展示数据，模板之外可以专注于要展示什么数据。



pom引入：

```
<dependency>
 <groupId>org.freemarker</groupId>
 <artifactId>freemarker</artifactId>
 <version>2.3.23</version>
</dependency>
```

来一个demo：
使用步骤：
第一步：创建一个Configuration对象，直接new一个对象。构造方法的参数就是freemarker对于的版本号。
第二步：设置模板文件所在的路径。
第三步：设置模板文件使用的字符集。一般就是utf-8.
第四步：加载一个模板，创建一个模板对象。
第五步：创建一个模板使用的数据集，可以是pojo也可以是map。一般是Map。
第六步：创建一个Writer对象，一般创建一FileWriter对象，指定生成的文件名。
第七步：调用模板对象的process方法输出文件。
第八步：关闭流。

```

public class FreeMarkTest {

 public static void main(String[] args) throws Exception {
 // 第一步：创建一个Configuration对象，直接new一个对象。构造方法的参数就是freemarker对于的版本号。
 Configuration configuration = new Configuration(Configuration.getVersion());
 // 第二步：设置模板文件所在的路径。
```

```
8   configuration.setDirectoryForTemplateLoading(new
File("D:\\ProgramData\\ftl"));
9   // 第三步：设置模板文件使用的字符集。一般就是utf-8.
10   configuration.setDefaultEncoding("utf-8");
11   // 第四步：加载一个模板，创建一个模板对象。
12   Template template = configuration.getTemplate("test.ftl");
13   // 第五步：创建一个模板使用的数据集，可以是pojo也可以是map。一般是Map。
14   Map dataModel = new HashMap<>();
15   //向数据集中添加数据
16   dataModel.put("hello", "我们来测试下数据看可以显示出来嘛");
17   // 第六步：创建一个Writer对象，一般创建一FileWriter对象，指定生成的文件名。
18   Writer out = new FileWriter(new File("D:\\ProgramData\\ftl\\test.html"));
19   // 第七步：调用模板对象的process方法输出文件。
20   template.process(dataModel, out);
21   // 第八步：关闭流。
22   out.close();
23
24   }
```

```
1  <h1>
2  ${hello}
3  </h1>
```

list标签：

```
1
2  <#list studentList as student>
3  ${student.id}/${studnet.name}
4  </#list>
```

if条件标签：

```
1  <#if student_index % 2 == 0>
2  <#else>
3  </#if>
```

Null值的处理:

```
1  <#if a??>
2  a不为空时。。
3  <#else>
4  a为空时###
5  </#if>
```

日期标签：

```
当前日期: ${date?date}
当前时间:${date?time}
当前日期和时间:${date?datetime}
自定义日期格式:${date?string("yyyyMM/dd HH:mm: ss")}
```

包含标签:

```
<#include "hello.ftl"/>
```

实战:

ItemController

```

@RestController
@Api(description = "商品列表信息")
@RequestMapping("/item")
public class ItemController {
 @Autowired
 ItemService itemService;

 @RequestMapping(value = "/static/{id}",method = RequestMethod.GET)
  @ApiOperation(value = "静态化商品")
  public CommonResult<String> buildStatic(@PathVariable Long id){

  String path = itemService.toStatic(id);
  if(StringUtils.isEmpty(path)){
  return CommonResult.failed("静态化商品页面出现异常");
  }
  return CommonResult.success(path);
  }

}
```

接口:

```
public interface ItemService {

 /**
 * 静态化商品详情页
 * @param id
 * @return
 */
 String toStatic(Long id);
}
```

静态化核心代码： `ItemServiceImpl`

```
1
2  @Override
3  public String toStatic(Long id) {
4   //查询商品信息
5   PmsProduct pmsProduct=productMapper.selectByPrimaryKey(id);
6   if (pmsProduct==null){
7   return null;
8   }
9   String outPath="";
10   try {
11   String userHome = System.getProperty("user.home");
12   // 第一步：创建一个Configuration对象，直接new一个对象。构造方法的参数就是freemar
ker对于的版本号。
13   Configuration configuration = new
Configuration(Configuration.getVersion());
14
15   // 第二步：设置模板文件所在的路径。
16   configuration.setDirectoryForTemplateLoading(new File(userHome+"/template/f
tl"));
17
18   // 第三步：设置模板文件使用的字符集。一般就是utf-8.
19   configuration.setDefaultEncoding("utf-8");
20
21   // 第四步：加载一个模板，创建一个模板对象。
22   Template template = null;
23
24   template = configuration.getTemplate("report.ftl");
25   // 第五步：创建一个模板使用的数据集，可以是pojo也可以是map。一般是Map。
26   Map dataModel = new HashMap();
27   // 向数据集中添加数据
28   dataModel.put("item", pmsProduct);
29
30   String images= pmsProduct.getPic();
31   if(StringUtils.isNotEmpty(images)){
32   String[] split = images.split(",");
33   List<String> imageList= Arrays.asList(split);
34   dataModel.put("imageList", imageList);
35   }
36
37   // 第六步：创建一个Writer对象，一般创建一FileWriter对象，指定生成的文件名。
38   outPath=userHome+"/template/report/1000"+pmsProduct.getId()+".html";
39   Writer out = new FileWriter(new File(outPath));
```

```
// 第七步：调用模板对象的process方法输出文件。
template.process(dataModel, out);
// 第八步：关闭流。
out.close();
} catch (IOException e) {
e.printStackTrace();
} catch (TemplateException te) {
te.printStackTrace();
}
return outPath;
}
```

前端：pms/index.vue

```
<el-button
 size="mini"
 @click="product_static(scope.$index, scope.row)">静
</el-button>
定义vue的product_static方法的js代码
```

script:

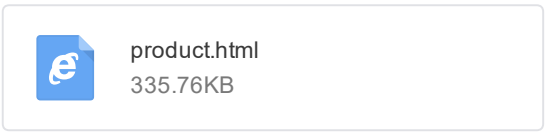
```
product_static(index,obj){
console.log(index,obj.id)
 this.$confirm('确认要静态化', '提示', {
 confirmButtonText: '确定',
 cancelButtonText: '取消',
 type: 'warning'
 }).then(()=>{
 productStatic(obj.id).then(response=>{
 this.$message({
 message: '静态化成功',
 type: 'success',
 duration: 1000
 });
 this.editSkuInfo.dialogVisible=false;
 });
 });
 }
```

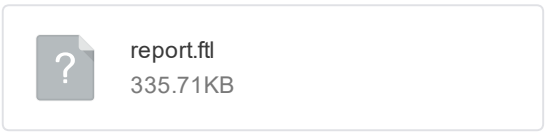
product.js:

```

export function productStatic(id) {
```

```
    return request({
    url:'/item/static/'+id,
    method:'get',
    })
}
```





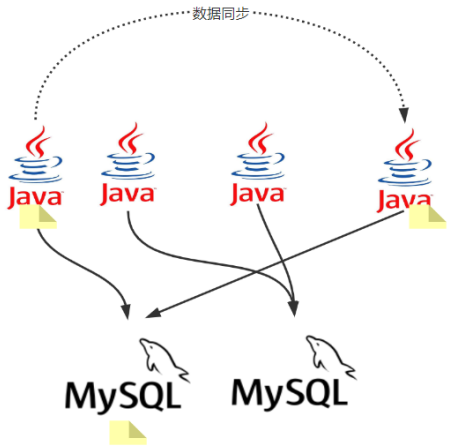
优化：价格改变、秒杀 倒计时、下单？？js没有生效（js、css、图片url）

小流量架构：https://www.processon.com/view/link/5e5774dae4b0cb56daac5a80

1000个 1个模板，1000个静态商品页面*机房（服务）数量
小米：1000个 12台 12000个静态化数据 CDN
京东：10000000 50台 上亿级别静态化页面
插入、修改、数据调整
1个模板改了所以的静态化 页面跟着改
scp

架构方案的问题：

问题一：
我们知道数据新增分:增量和全量数据
如果后台的小二新增了很多的商品，那我们都要对这些商品进行静态化，但是现在有个问题。那这些数据如何同步了？这是一个新增商品同步的问题，那这个问题怎么解决比较好了？。



不同应用部署在不同服务器甚至在不同的机房不同的国家。

1、通过网络同步的方式 就是其中一台服务器静态化之后，然后把文件同步到其他应用服务器上去。比如我们的linux命令scp方式。这种方式虽然可行，但是我们发现问题还是蛮多的，有多少个节点就需要同步多少份，等于是商品的数量*服务器的应用数数。很显然这种办法不是最优的解决办法

如果上述办法无法解决，那我们就用另外的方案，同学们你们觉得还有其他的方案没有?

2、定时任务:可以在某个应用用一个定时任务，然后分别去执行数据库需要静态化的数据即可，可以解决上述1数据同步的问题，因为所有的任务都是在本机运行，就不需要数据同步了。但是也有一个问题。就是如何避免不通的机器跑的数据不要重复，也就是A和B定时任务都跑了一份商品。这个是这种方案需要解决的。（比较直观的就是上锁）

3、消息中间件：还有一种办法就是通过消息中间件来解决。订阅topic然后生成当前服务器静态化的页面。

问题二：

我们的freemark它是数据要事先按我这个模板生产好的，那就是说一定你改了模板，如果要生效的话，需要重新在把数据取出来和我们这个模板进行匹配生产更多的的静态html文件。那这是一个比较大的问题

如果后台数据有变更呢?如何及时同步到其它服务端?

如果页面静态化了，我们搜索打开一个商品详细页，怎么知道要我需要的访问的静态页面?

万一我们模板需要修改了怎么办?

牵一发动全身。

后台优化：

redis缓存：

redis设置：RedisConifg》RedisOpsUtil

```
/**
 * 获取商品详情信息
 *
 * @param id 产品ID
 */
public PmsProductParam getProductInfo(Long id) {
 PmsProductParam productInfo = null;
 //从缓存Redis里找
 productInfo = redisOpsUtil.get(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE + id, PmsProductParam.class);
  if(null!=productInfo){
  return productInfo;
  }
  productInfo = portalProductDao.getProductInfo(id);
  if (null==productInfo) {
  log.warn("没有查询到商品信息,id:"+id);
  return null;
  }
  FlashPromotionParam promotion = flashPromotionProductDao.getFlashPromotion(id);
  if (!ObjectUtils.isEmpty(promotion)) {
  productInfo.setFlashPromotionCount(promotion.getRelation().get(0).getFlashPromotionCount());
  productInfo.setFlashPromotionLimit(promotion.getRelation().get(0).getFlashPromotionLimit());
  productInfo.setFlashPromotionPrice(promotion.getRelation().get(0).getFlashPromotionPrice());
  productInfo.setFlashPromotionRelationId(promotion.getRelation().get(0).getId());
  productInfo.setFlashPromotionEndDate(promotion.getEndDate());
  productInfo.setFlashPromotionStartDate(promotion.getStartDate());
  productInfo.setFlashPromotionStatus(promotion.getStatus());
  }
  redisOpsUtil.set(RedisKeyPrefixConst.PRODUCT_DETAIL_CACHE + id, productInfo, 3600, TimeUnit.SECONDS);
  return productInfo;
 }

```

好处：

加入redis之后我们发现提高了可以把之前请求 数据库查询的商品都缓存到redis中，通过对redis的访问来减少对数据里的依赖，减少了依赖本质就是减少了磁盘IO。

问题：

提高请求的吞吐量，除了减少磁盘IO，还有网络IO，我们可以发现，请求redis其实也会涉及到网络IO，我们所有的请求都要走xxx端口号。那有没有更好的优化思路了，来同学们你们鲜花在哪儿?

压力测试：

我们发现吞吐量有一定的提高。但是问题还是有的。

笔记地址：

文档：02 秒杀系统-商品详细页多级缓存实战一...

链接：http://note.youdao.com/noteshare?id=8395bb526e185797776f1e2a45cb73ff&sub=098910CD87884E1BA9BCF9E715323B37