# 品优购项目话术

## 项目描述

1，品优购网上商城是一个综合性的 B2B2C 平台。网站采用商家入驻的模式，商家入驻平台提交申请，有平台进行资质审核，审核通过后，商家拥有独立的管理后台录入商品信息。商品经过平台审核后即可发布

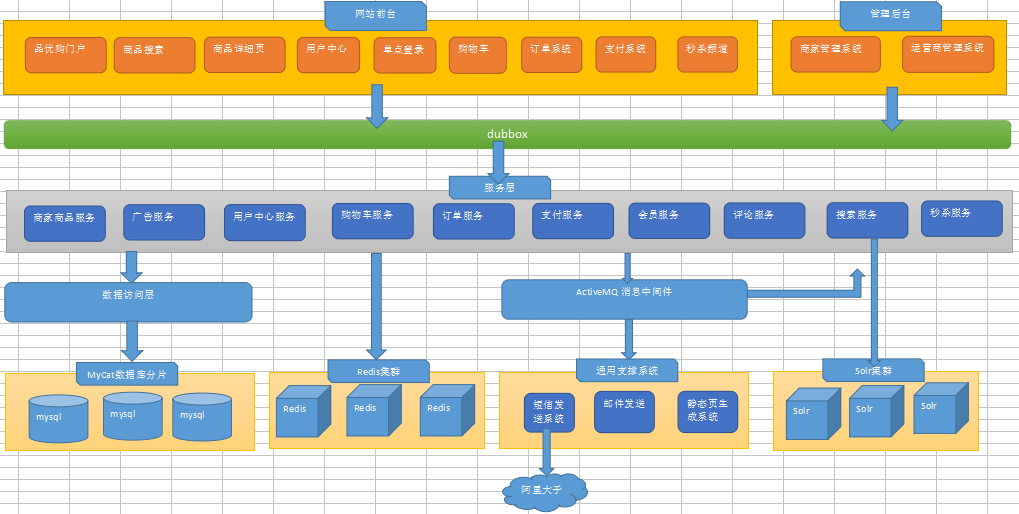
## 三大板块

1. 网站前台主要包括网站首页、商家首页、商品详细页、、搜索页、会员中心、订单与支付相关页面、秒杀频道等
2. 运营商后台是运营商的运营人员的管理后台。 主要包括商家审核、品牌管理、规格管理、模板管理、商品分类管理、商品审核、广告类型管理、广告管理、订单查询、商家结算等

入驻的商家进行管理的后台，主要功能是对商品的管理以及订单查询统计、资金结算等功能。

## 项目架构

为什么互联网项目会采用SOA架构呢？随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，迫切需一个治理系统确保架构有条不紊的演进



## 表结构

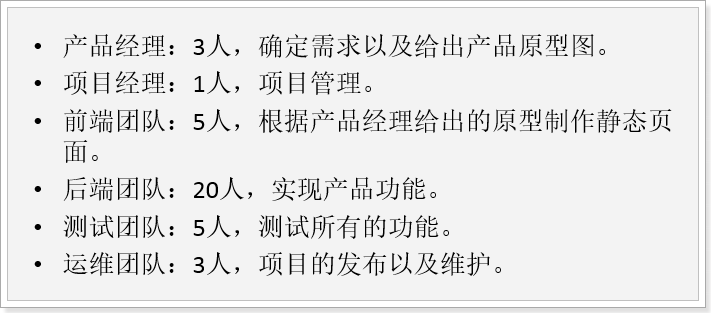
|  |  |
| --- | --- |
| **表名称** | **含义** |
| tb\_brand | 品牌 |
| tb\_specification | 规格 |
| tb\_specification\_option | 规格选项 |
| tb\_type\_template | 类型模板：用于关联品牌和规格 |
| tb\_item\_cat | 商品分类 |
| tb\_seller | 商家 |
| tb\_goods | 商品 |
| tb\_goods\_desc | 商品详情 |
| tb\_item | 商品明细 |
| tb\_content | 内容（广告） |
| tb\_content\_category | 内容（广告）类型 |
| tb\_user | 用户 |
| tb\_order | 订单 |
| tb\_order\_item | 订单明细 |
| tb\_pay\_log | 支付日志 |

## 框架组合

品优购采用当前流行的前后端编程架构。

后端框架采用Spring +SpringMVC+mybatis +Dubbox 。前端采用angularJS + Bootstrap。

## 人员配置



# 常见问题

## 常见问题回答思路

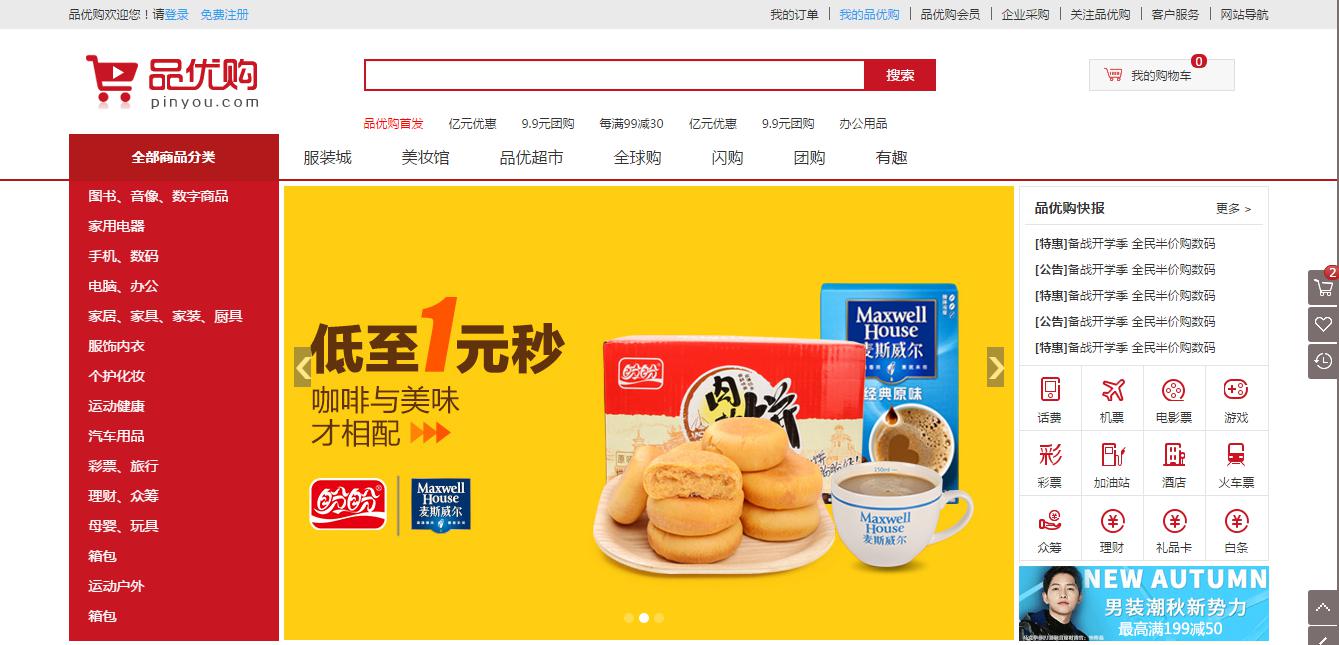
1，简单说下你最近做过的或者熟悉的项目

* 1. 一般情况下，这个问题是在面试中出现频率最高的问题，面试官旨在想了解你曾经做过的项目，此时只需要简单介绍一下项目开发的背景，项目中所使用的技术，已经目前项目的进展（是否已经上线等）

1. 你主要负责哪些模块
   1. 回答此问题面试官旨在考察你的技术能力以及表达能力，首先，从技术角度出发，面试官能够了解到面试者到技术功底，从语言角度出发，面试官能够判断出面试者是否具有独立逻辑思维能力以及良好的语言组织能力，能否将自己曾经做过的项目较为完整的呈现给面试官
2. 具体到模块中的技术点
   1. 回答此类问题的前提，面试者在简历中所写的责任描述以及涉及到的技术点，面试者必须要有所准备，切记不要给你自己挖坑
   2. 回答思路按照是什么问什么怎么样的思路来回答
3. 你在平时开发中遇到过哪些棘手的问题？你是如何解决的？
   1. 对于大部分学生来说，回答此类问题的接本思路就是自己在学习的阶段遇到的bug或者自己在搭建系统环境的时候出现的问题。如何解决，侧重点是考察面试者的问题解决能力。

## 问题汇总

### 网站前台



功能介绍：

主要包括网站首页、商家首页、商品详细页、、搜索页、会员中心、订单与支付相关页面、秒杀频道等

#### 主要模块

1. 前端部分你们使用的是什么框架？简单说下angular.js的特征已经常用的指令？

前端部分主要使用angular.js+bootstrap。Angular的特征主要有四个，分别是MVC模式，双向绑定，依赖注入，模块化设计。常用的指令ng-app，ng-init，ng-model，$scope，$http(相当于jq中的ajax请求)。

1. 网站首页的图片你们是怎么处理的
   1. 网站首页图片我们在开发阶段使用的是分布式文件存储fastDFS。现实环境我们使用了第三方图片服务器，七牛图片服务器，便于后期维护，相对于阿里云的图片服务，七牛图片服务器在成本上更好控制
2. 搜索页中关于搜索的处理你们是怎么做的？solr与Elasticsearch有什么却别？

搜索我们主要了solr来开发的，目前市面上也有诸多的搜索解决策略，比如elasticsearch等。由于solr相较于elasticsearch来说，有更加强大的开发和贡献者社区，便于后期维护，故我们采用了solr作为我们网站的搜索引擎。我们部署solr采用了集群部署的方式。也是惯用的方式，单台solr弊端就是一旦solr挂掉，那么网站的搜索系统就会迅速瘫痪，为了避免此类问题的产生，通常我们都会部署成集群。关于solr索引库的更新，我们使用了activityMQ消息中间件来维护索引库与数据库数据的同步。

solr与Elasticsearch的区别：

solr：

1. Solr有一个更大、更成熟的用户、开发和贡献者社区。
2. 支持添加多种格式的索引，如：HTML、PDF、微软 Office 系列软件格式以及 JSON、XML、CSV 等纯文本格式。
3. Solr比较成熟、稳定。
4. 不考虑建索引的同时进行搜索，速度更快

Elasticsearch

1. Elasticsearch是分布式的。不需要其他组件，分发是实时的，被叫做”Push replication”。
2. Elasticsearch 完全支持 Apache Lucene 的接近实时的搜索。
3. 处理多租户（[multitenancy](http://en.wikipedia.org/wiki/Multitenancy" \t "_blank)）不需要特殊配置，而Solr则需要更多的高级设置。
4. Elasticsearch 采用 Gateway 的概念，使得完备份更加简单。
5. 各节点组成对等的网络结构，某些节点出现故障时会自动分配其他节点代替其进行工作。
6. 对于秒杀频道的设计你了解多少？

要点总结：

1.架构：扩容，业务分离，数据分离

2.产品：下单按钮控制，秒杀答题削峰，简化页面设计

3.前端：限流（反作弊） 静态化

4.后端：内存 队列

一、秒杀一般会带来2个问题：

1、高并发

比较火热的秒杀在线人数都是10w起的，如此之高的在线人数对于网站架构从前到后都是一种考验。

2、超卖

任何商品都会有数量上限，如何避免成功下订单买到商品的人数不超过商品数量的上限，这是每个抢购活动都要面临的难题。

实际上超卖问题是高并发带来的一个子问题，但是因为这个问题太过致命，所以我们把他的解决方案单独拿出来说。

二、如何解决？

1.架构层面：

秒杀架构设计原则：

尽量将请求拦截在系统上游

读多写少的常用多使用缓存

扩容

说白了加机器

系统隔离

为了避免短时间内的大访问量对现有网站业务造成的冲击，可以将秒杀系统独立部署。系统隔离更多是运行时的隔离，可以通过分组部署的方式和另外99%分开。秒杀还申请了单独的域名，目的也是让请求落到不同的集群中。即使秒杀系统崩溃了，也不会对网站造成影响。

数据隔离

将即将被秒杀的热数据维护到redis。秒杀所调用的数据大部分都是热数据，比如会启用单独cache集群或MySQL数据库来放热点数据，目前也是不想0.01%的数据影响另外99.99%。

减库存操作

一种是拍下减库存 另外一种是付款减库存；目前采用的“拍下减库存”的方式，拍下就是一瞬间的事，对用户体验会好些。

2.产品层面：

1.控制秒杀商品页面抢购按钮的可用/禁用。

购买按钮只有在秒杀开始的时候才能点亮，在此之前是灰色的，显示活动未开始。

2.增加了秒杀答题，基于时间分片削峰

秒杀答题一个很重要的目的是为了防止秒杀器。还有一个重要的功能，就是把峰值的下单请求给拉长了，从以前的1s之内延长到2~10s左右，请求峰值基于时间分片了，这个时间的分片对服务端处理并发非常重要，会减轻很大压力，另外由于请求的先后，靠后的请求自然也没有库存了，也根本到不了最后的下单步骤，所以真正的并发写就非常有限了。其实这种设计思路目前也非常普遍，如支付宝的“咻一咻”已及微信的摇一摇。

3.秒杀页面设计简化：

秒杀场景业务需求与一般购物不同，用户更在意的是能够抢到商品而不是用户体验。所以秒杀商品页面应尽可能简单并且拍下后地址等个人信息应该使用默认信息，减轻秒杀进行时系统负载，若有更改可以在秒杀结束后进行更改。

3.前端层面

静态化以及页面缓存

将页面能够静态的部分都静态化，并将静态页面缓存于CDN，以及反向代理服务器，可能还要临时租借服务器。

利用 页面静态化、数据静态化，反向代理 等方法可以避免 带宽和sql压力 ，但是随之而来一个问题，页面抢单按钮也不会刷新了，可以把 js 文件单独放在js服务器上，由另外一台服务器写 定时任务 来控制js 推送。 另外还有一个问题，js文件会被大部分浏览器缓存，我们可以使用xxx.js?v=随机数 的方式来避免js被缓存。

限流（反作弊）

1.针对同一个用户id来实现，前端js控制一个客户端几秒之内只能发送同一个请求，后端校验同一个uid在几秒之内返回同一个页面

2.针对同一个ip来实现，进行ip检测，同一个ip几秒之内不发送请求或者只返回同一个页面

3.针对多用户多ip来实现，依靠数据分析

4.为了避免用户直接访问下单页面URL，需要将改URL动态化，即使秒杀系统的开发者也无法在秒杀开始前访问下单页面的URL。办法是在下单页面URL加入由服务器端生成的随机数作为参数，在秒杀开始的时候才能得到。

4.后端层面：

1.加入缓存redis：

因为秒杀是典型的读多写少的场景，适合操作内存而非操作硬盘；缓存工具redis本身的操作是保证原子性的，所以可以保证请求了redis的写的操作的线程安全性。

2.加入消息队列，利用队列进行削峰：

将用户请求放置于一个或多个队列中，队列中元素总和等于该商品库存总和，未进入队列的请求均失败。利用多线程轮询分别从一个或多个队列中取出用户请求。操作redis进行减库存操作，成功减库存之后返回成功，并将用户信息与商品信息存入另一个队列当中，进行生成订单的操作。利用两个队列异步处理业务减轻秒杀高峰时期服务器负载。

3.程序计数器：

队列与缓存为了保证请求redis的次数不超过总的库存量，利用一个程序计数器来这一点。程序计数器用JUC包下原子类可以实现。

4.分布式锁

分布式情况下可以利用分布式锁来解决任务每次只能由一次服务来执行且不能重复执行。

分布式锁的实现：zk、redis

分布式锁的优化：先考虑是否可以去锁，然后考虑尽可能多用乐观锁，少用悲观锁。这里有一个问题，乐观锁如果每一次都会有并发冲突的话性能反而不如悲观锁，那么难道真的多用乐观锁性能会比悲观锁高吗？选举考虑ha，比如心跳检测。

5.分布式去锁 方案

利用集群并发加入队列，选举队列处理服务单点执行，这样可以保证并发实现和加锁一样的并发量但不会影响性能。

1. 网站使用了什么安全框架？对于密码的校验你们是怎么设计的？
   1. 我们网站在安全方面做了比较充足的准备，在运营商系统的登陆模块中使用了目前最为流行的安全框架Spring Security。Spring Security 是一个能够为基于 Spring 的企业应用系统提供声明式的安全访问控制解 决方案的安全框架。它提供了一组可以在 Spring 应用上下文中配置的 Bean，充分利用了 Spring IoC，DI(控制反转 Inversion of Control ,DI:Dependency Injection 依赖注入)和 AOP(面 向切面编程)功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控 制编写大量重复代码的工作。



* 1. 密码的校验，我们在商家登陆时，对密码做了加密处理，使用到了BCrypt 算法 。理由：用户表的密码通常使用 MD5 等不可逆算法加密后存储，为防止彩虹表破解更会先使用 一个特定的字符串(如域名)加密，然后再使用一个随机的 salt(盐值)加密。 特定字符 串是程序代码中固定的，salt 是每个密码单独随机，一般给用户表加一个字段单独存储，比 较麻烦。 BCrypt 算法将 salt 随机并混入最终加密后的密码，验证时也无需单独提供之前的 salt，从而无需单独处理 salt 问题。



1. redis在你们开发的时候都有哪些地方用到了？

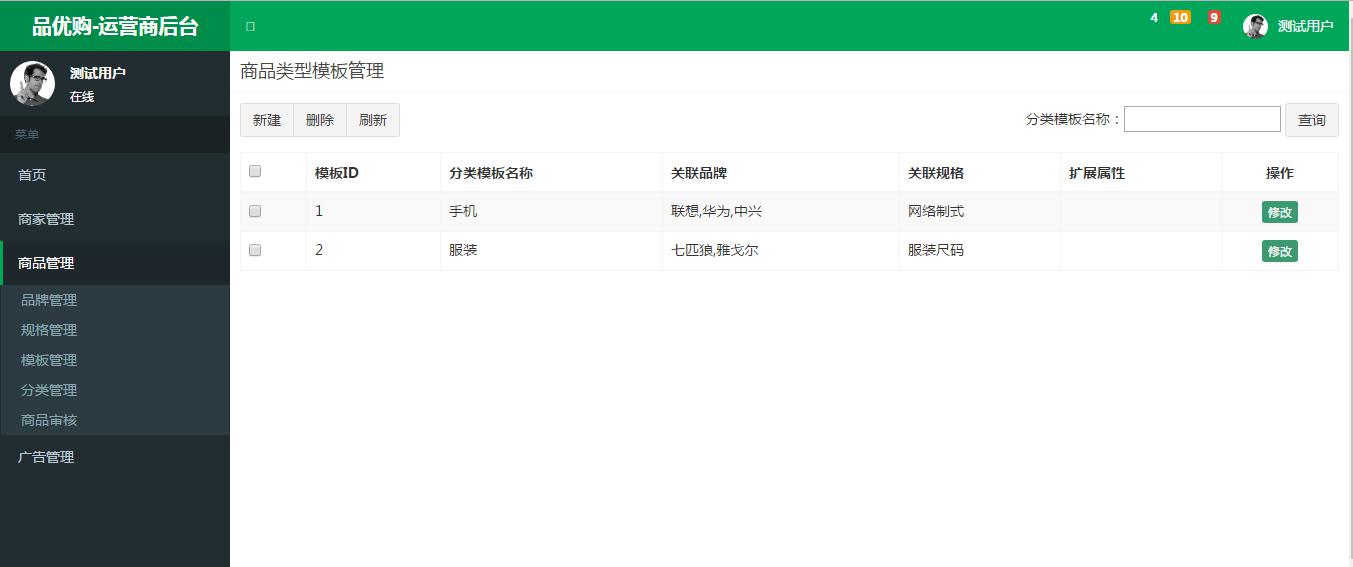
redis 是一款开源的 Key-Value 数据库，运行在内存中，由 ANSI C 编写。企业开发通常 采用 Redis 来实现缓存。

Redis在本系统中主要体现在：缓存和解决session共享

作为内存数据库，用作缓存时redis基本使用，我们在商品分页查询已经首页轮播图的展示时均使用了redis作为缓存数据库，当查询请求过来时，首先回去redis查询是否有相关数据，如果有，直接返回，反之，会发送sql语句给数据库，将相关数据查询出来，并将数据写入redis中，最终返回给前端页面。

另外使用redis的地方就是在解决我们单点登录系统时候出现的session共享问题，何为session共享。我们都知道用户登陆成功后，用户信息是存储到session中的，而我们的系统在分布式系统，每个模块都是一个独立的web工程，也就是说，每个模块时部署在不同的tomcat服务器中的。但是每个服务器的session是不共享的，这样就会出现一个问题，当我从登陆登入首页时，会重新跳转到登陆页面，因为单点登录系统中的用户信息无法传递到前台系统中。即：session共享问题。由于redis是k-v数据库，session同样是存储k-v的，可以用redis来模拟session，将redis部署到服务器中，这样就可以使得每个系统在运行的时候，回去服务器中请求加载存储到redis的用户信息，session共享问题引刃而解。

### 运营商后台



功能描述：

是运营商的运营人员的管理后台。 主要包括商家审核、品牌管理、规格管理、模板管理、商品分类管理、商品审核、广告类型管理、广告管理、订单查询、商家结算等。

#### 主要模块

1，运营商后台商品列表页的分页实现是如何做的？



主要使用了angular.js的分页插件。其中重点需要关注的几个参数：

分页显示

paginationConf 变量各属性的意义：

currentPage：当前页码

totalItems:总条数

itemsPerPage:每页显示的记录数

perPageOptions：页码选项

onChange：更改页面时触发事件

2，电商概念SPU和SKU

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。通俗点讲，属性值、特性相同的商品就可以称为一个SPU。

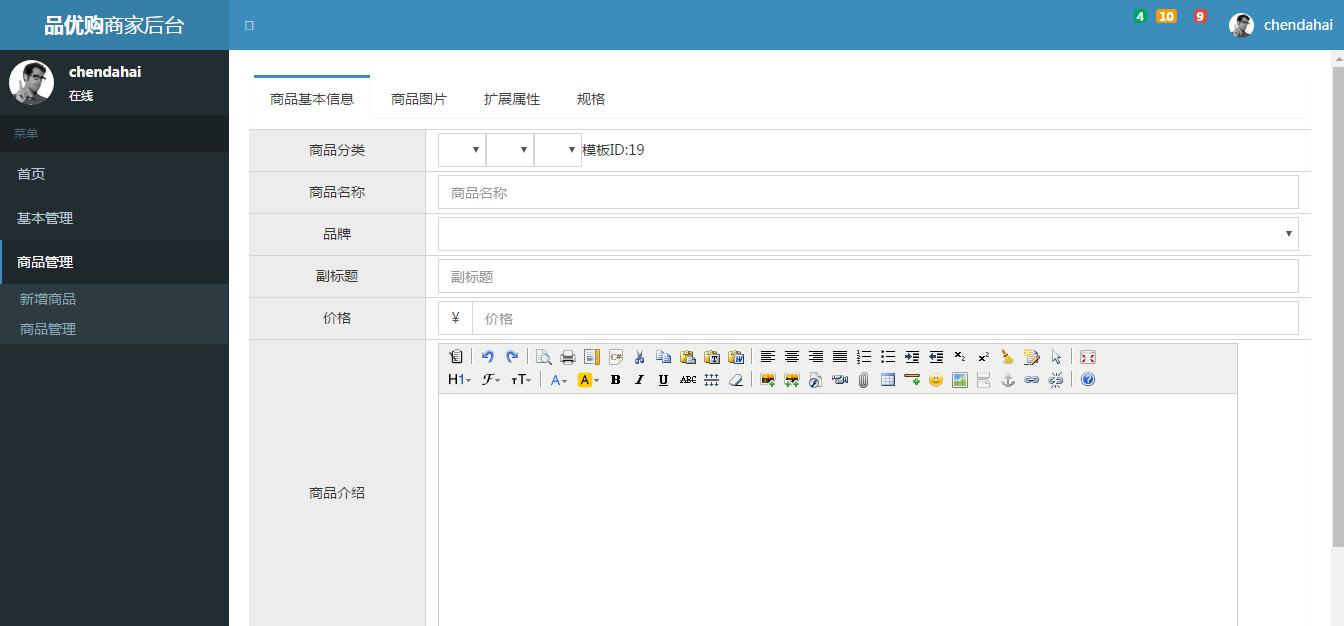
例如：iphone7就是一个SPU，与商家，与颜色、款式、套餐都无关

SKU即库存进出计量的单位， 可以是以件、盒、托盘等为单位。

SKU是物理上不可分割的最小存货单元。在使用时要根据不同业态，不同管理模式来处理。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。

### 商家管理后台



功能描述

入驻的商家进行管理的后台，主要功能是对商品的管理以及订单查询统计、资金结算等功能。

#### 主要模块

1，商品录用关于富文本编辑器

富文本编辑器介绍

富文本编辑器，Rich Text Editor, 简称 RTE, 它提供类似于 Microsoft Word 的编辑功能。常用的富文本编辑器：

KindEditor http://kindeditor.net/

UEditor http://ueditor.baidu.com/website/

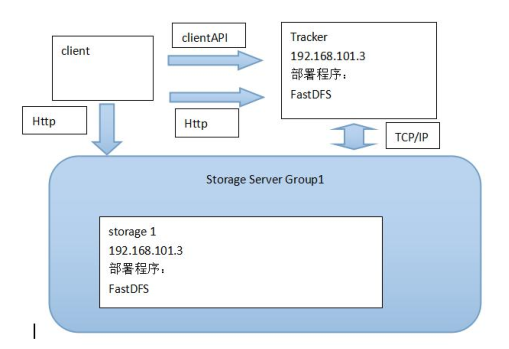
CKEditor <http://ckeditor.com/>

本系统中使用KindEditor完成商品介绍的录入

2，关于图片的处理方式你怎么是处理的？

FastDFS 是用 c 语言编写的一款开源的分布式文件系统。FastDFS 为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用 FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务

FastDFS 架构包括 Tracker server 和 Storage server。客户端请求 Tracker server 进行文件上传、下载，通过 Tracker server 调度最终由 Storage server 完成文件上传和下载。



最简单的fastDFS实现

3，商品类目的级联查询是怎么实现的？

在商品录入界面实现商品分类的选择（三级分类）

当用户选择一级分类后，二级分类列表要相应更新，当用户选择二级分类后，三级列表要相应更新。

这里使用到angularjs的$.watch监听标签，分别监听一、二、三级分类，当一级分类选择后，发送请求查询二级分类，当三级选择后，查询模版id、品牌、扩展属性、规格，涉及到很多表，逻辑思维很复杂

4，关于商品的上下架，你们是怎么处理的？

其实上下架也是商品的一个状态，但是不同于审核状态。审核状态的控制权在运营商手中，而上下架的控制权在商户手中。商户可以随时将一个已审核状态的商品上架或下架。上架表示正常销售，而下架则表示暂停销售。其实商品的上下架就是对上下架状态的修改。字段为tb\_goods表的is\_marketable字段。1表示上架、0表示下架。

5，我们目前的系统已经实现了广告后台管理和广告前台展示，但是对于首页每天有大量的人访问，对数据库造成很大的访问压力，甚至是瘫痪。那如何解决呢？我们通常的做法有两种：一种是数据缓存、一种是网页静态化。我们今天讨论第一种解决方案

回答该问题主要将redis是如何实现存储数据的以及在业务逻辑中的是如何实现的。但是需要主要的是读取缓存，增删改后需清除缓存