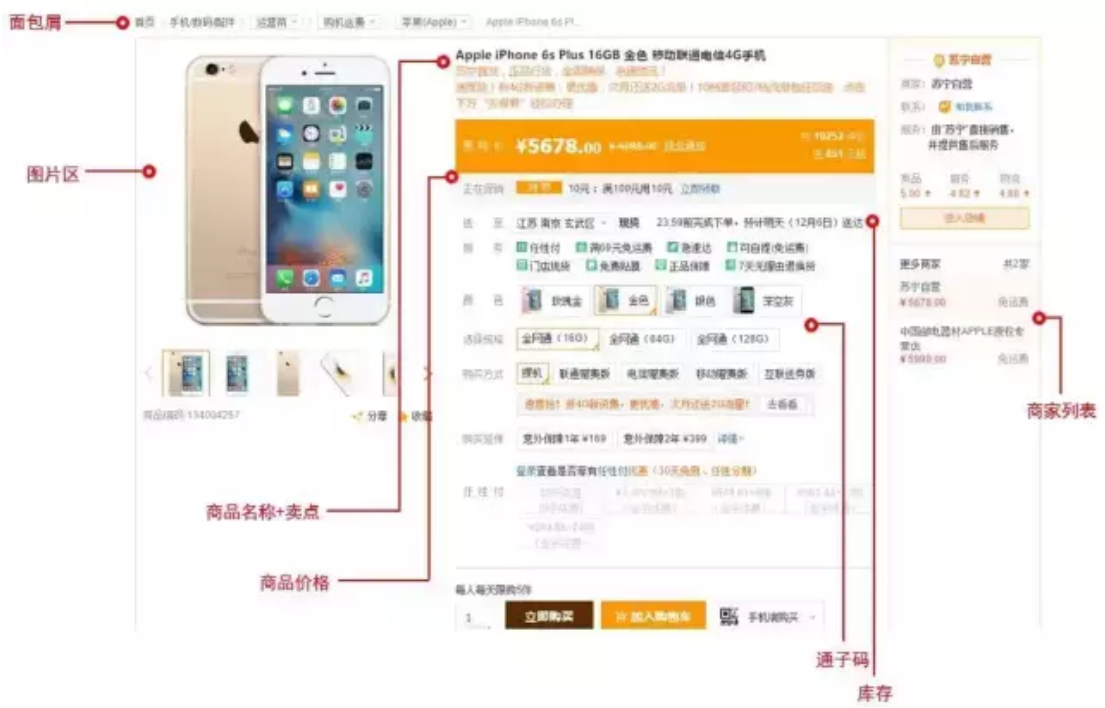
架构案例丨苏宁易购：商品详情系统架构设计

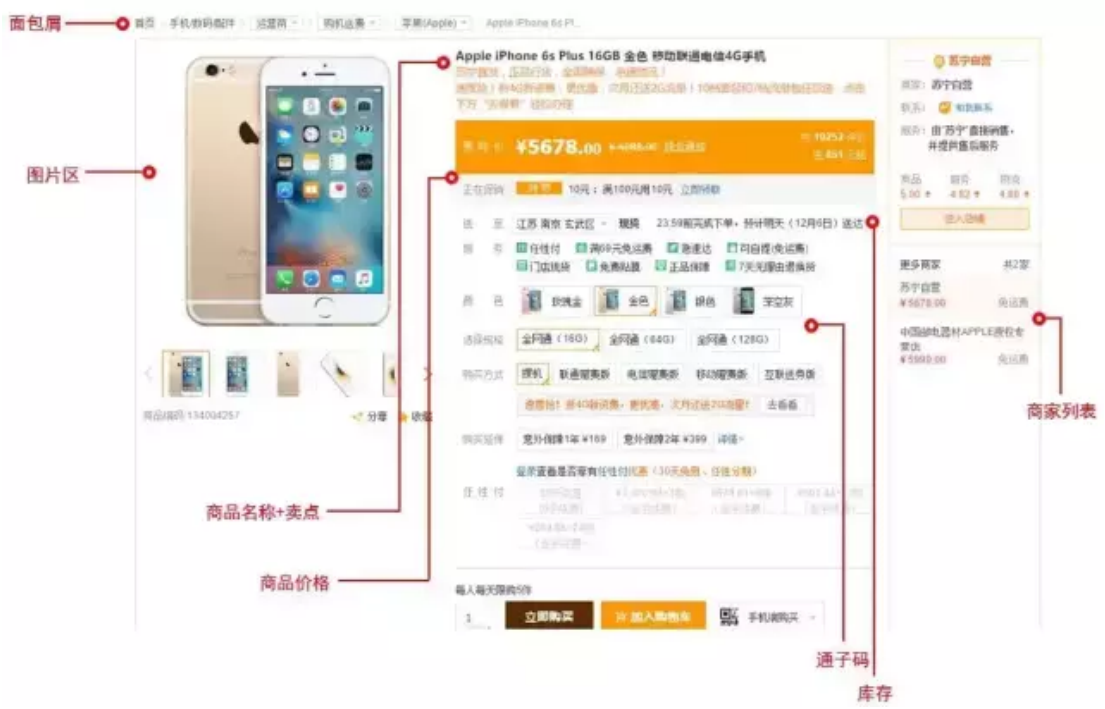
原创：尹坚 InfoQ 2015-12-30

商品详情系统介绍

基本介绍

商品详情系统是一个展示商品基本信息、参数等详情的系统，是商品购买的入口。它是电商平台中访问量最大的系统之一，苏宁易购大促期间PV量和UV量很大，这么大的访问量对系统的并发能力要求高。在业务上它与周边系统的关系是高耦合。依赖商品详情系统的的系统特别多，比如：促销系统、推荐系统、大聚惠、等众多营销系统、还有主数据系统、购物车、收藏夹等，业务复杂度高对系统设计提出更多的要求。





业务特点

1. 重点在于数据展示
2. 页面信息丰富，如：商品详情、商家列表、推荐、排行榜等
3. 部分数据时效要求高，如：价格、库存等
4. 业务上依赖的系统多

商品详情系统三要素

1. 展示

产品上需要设计好页面区分展示的内容

技术上主要是页面缓存设计、前端页面模版和JAVA程序的解耦

2. 数据处理

数据全部来源于其它系统，在数据上分为：

基本数据，外部系统传过来直接就可以使用的数据

聚合数据，需要加工才能使用的数据

3. 服务依赖

通过MQ解耦，异构数据

解决好以上三个问题就解决了此系统核心问题。

系统逻辑架构

商品详情系统在设计上分成前、中、后三层结构

1. 前台负责展示，做为VIEW层不处理业务逻辑，负责渲染。
2. 中台负责业务逻辑处理，提供数据给前台，同时还会对外部系统提供服务
3. 后台负责主数据管理，做为数据管理层处理商品主数据、参数、品牌、供应商等，同时部分内容开放给运营进行维护、管理和异常处理等。

前台设计

页面设计：

1. 动静分离

JavaScript、CSS统一放到公共的静态服务器上，完全独立的子域名，防止脏Cookie问题和动态域名中无用Cookie问题，通过文件版本号解决系统新版和旧版本之间冲突问题。

所有图片由独立的分布式图片系统管理，对原图进行不同规格的无损裁减和压缩。

2. 异步加载和懒加载

商品价格、营销活动信息、库存等动态数据通过异步加载

非首屏数据做懒加载处理，提高首屏加载时间，比如评价、商品详情等内容

3. 多级缓存策略

a. 浏览器本地缓存

协商缓存，对于某些时效要求较高的资源通过Last-modified控制数据。做到StatusCode=304

强缓存，JS、CSS等静态资源或者一些页面碎片伪静态数据通过Expires、Cache-Control(http1.1支持)设置做到强缓存，在不强制刷新的情况下可以做到200(from cache)

b. CDN缓存

CDN分两条线有自营CDN和合作商的CDN，图片、静态资源与伪静态数据分别放在不到的CDN上

c. Varnish缓存

Varnish在设计上负载使用轮询方式，不使用URL HASH策略，用空间换时间的策略， 从而避免热数据问题，也支持横向扩展。

Varnish 缓存和CDN缓存在失效时间错开，从而避免同时失效回源压力过大。

d. 精准缓存

精准缓存失效用于促销活动准时展示的场景，基于Varnish缓存，通过精准控制缓存有效期实现缓存精准失效保证促销活动准时切换。

组件逻辑设计：

商品详情系统中的购买按钮和加入购物车会因商品不同走不同的流程。如：大聚惠商品、定金团商品、预售商品等因促销方式不同，走不同的业务处理流程。促销模式变化多端，可能每个月都会有变化，通常的面向接口编程和加上工厂方法或者依赖管理框架Spring也很难做到真正的解耦，虽然这样做已经符合开闭原则。我们通过观察者模式很好的解决了这个问题。让前端的页面模版和JAVA应用程序之间真正的解耦。

后台设计

商品数据统一处理设计

商品详情系统商品主数据通过MQ消息来源于外部系统，比如：商品基本信息、参数、参数模版、品牌、品类等。我们设计时把共性抽出来分成三部分：

1. 接受MQ消息并持久化通过Listener
2. 解析报文
3. 业务处理上简化为add、update、delete三个动作

4. 异常组件以观察者模式实现，记录处理失败的MQ消息并对消息进行截取，并供下次再反向执行(一条MQ消息中会有一到多条参数、品牌，所以这里用截取)

SOA服务治理

解耦分两块，系统交互间的解耦和商品详情系统组件间的解耦以及业务流程的解耦

系统间的解耦通过SOA服务治理来解决，但是由于业务的特殊性在服务治理和性能以及一些其它因素的权衡中，我们还选择了一种共享Redis的方式来解决解耦

商品详情系统组件间解耦以及业务流程的解耦。

架构演变

1.0时代中规中矩，移动端完全移值PC的做法

我们使用中规中矩的部署方式Varnish+Apache+JBoss。

这种架构在针对中小系统没有什么问题，但像商品详情系统这种访问量巨大的系统会显的有点吃力。移动端对性能的要求更高。

2.0时代PC和移动端服务分离，移动端服务合并，性能优化

a. 服务分离与服务合并

PC和移动端的服务分离，以前是同一个接口支持多端，现在是每端都有独立的应用层服务，原子层服务共享。

移动端处理器和内存性能上的限制，采用服务的合并，且移动端用Nginx+Lua。

b. 公共服务

提出了一个公共服务，公共服务用来接受PC、WAP、APP公共的异步请求的服务。

c. 分布式文件系统

商品详情页在回源过程中压力很大，基于其不可降级，我们提出了把商品详情页做为一个静态页放到分布式文件系统，当DB和Redis压力过大，直接调取分布式文件系统中数据

3.0时代重点优化移动端性能，接口合并颗粒度更细，增加聚合服务层
多端都使用Nginx+Lua，Nginx 的异步非阻塞型事件处理机制资源消耗少，并发能力高。

1. 用Nginx+Lua做为整体的接入层
2. 在Nginx接入层 加入三层缓存
3. 只有聚合信息才会调用服务层，减少依赖关系
4. 服务层数据通过Worker推送和刷新缓存，这亲服务层完全和DB隔离
5. 移动端连接复用、链路复用、防劫持SDK开发等

商品详情系统数据流结构

上面介绍了商品详情系统前、中、后三层逻辑架构以及各层的设计方法，还介绍了部署架构演变，下面是商品详情系统数据流程结构的

1.0版本：

这个结构有两个问题：
数据异构结果没有和前端展示关联起来，数据变更不能在前端及时展示
还是没有解决前端接口依赖问题。

2.0版本：

把前端分成了三部分：

基础信息组件 不需要加工的消息、聚合信息组件(需要组合消息或者计算才能提供服务的)、实时数据组件处理对外部的依赖
数据异构后会以MQ形式通知基础服务，并会刷新缓存，这种结构后前端与数据层无直接依赖。

回源方案

回源是缓存中最头痛的问题，随着系统业务复杂度的上升，很难从整体上把控各种业务数据在回源时给一个系统带来的压力，如果回源处理不端在极端情况下会导致DB

压力瞬间上升，DB不可用或者连接数满了等问题，会发生以前类似JVM GC回收时的“stop-the-world”问题。我们回源从被动更新缓存数据更改为主动推送缓存数据从根本上解决这问题。

数据变更通过listener推送缓存至varnish

多端融合

组织架构融合

原来PC端、移动端、TV端产品、开发、测试是分中心分部门，为真正做到多端融合，进行组织架构融合，产品、开发、测试合并到一个中心，统一协调。合并后工作效率变高，产品质量提升，进行小团队做战。

展示分离逻辑融合分离

展示分离是指在结合公司业务特性、产品自身特性以及降耦合指导思想进行PC、WAP、APP端(IOS、ANDROID)、TV端的展示端进行分离处理。

逻辑融合分离是指在原子服务层进行融合共享，从服务单一职责原则出发在不同端分别提供独立的服务并加上各自特性，做到接口可扩展性和服务隔离。真正做到一包部署多端使用互不影响，在业务可扩展性和可维护性上做到成本最低。

展示层分端独立部署

在物理层为了避免多端进行资源竞争、相互干扰进行独立部署

分布式存储

数据库

1. 商品详情系统数据库用Mysql，采用主从加读写分离结构，注意：主从不在同一个物理机上，也不在同一组路由器中。应用层中业务上对时效性要求高的数据在写库中操作，业务上对于时效性要求不高数据在读库中操作。主从结构保证在主库出现故障比如岩机自动切换到从库。读库通过LVS做负载均衡做到高可用。
2. DalClient组件支持对数据库的分库分表，同时支持横向扩展。

分布式Redis缓存

1. 应用层逻辑优先从Reids中获取业务数据，如果Redis中没有，再从DB中获取。Redis采用sharding方案，每个sharding由一个master和一个salve组成，再通过sentinel保证高可用。当master出现不故障，比如网络跳动，sentinel会自动把salve切换为master，这个切换是毫秒级的。master和salve通过主动和被动两种方式来同步，做到最终一致性，符合CAP理论演变过的BASE理论。
2. 借鉴JAVA GC中对内存分代思路解决Redis缓存过期产生的惊群现象。

- 更多干货内容，敬请关注InfoQ［id：infoqchina］微信公众号

荐文

- 案例丨详解当当网的分布式作业框架elastic-job
- Q新闻丨Oracle确认Java 9跳票！微软发布Azure备份服务器……
- Q调查丨技术人都是怎样走上编程之路的?
- 软件开发如何规避时间碎片化的坑?