

京东网站单品页 6·18实战



京东
JD W

[首页](#)

[首页](#) > [安睡宝](#)

全球购 > 京东全球

自营 京东全球购

NUTRICIA
Nutrilo

品编号: 19510972

品牌名称:

手机 > 手机通讯 > 手机 > 苹果 (APPLE) > 苹果iPhone 6 Plus



商品编号: 1217524



分享 关注商品

苹果 (Apple) iPhone 6 Plus (A1524) 16GB 金色 移动联通电信4G手机

对比

尺寸更大，却愈加纤薄；性能更强，却效能非凡。堪称 iPhone 新一代至为出众的大作。选择“移动老用户4G飞享合约”，无需换号，最高可返还3528元话费。选择下方“北京移动购机赠费”购买方式，推荐188元套餐，每月省30%的套餐资费，流量比套餐内多一倍哦！

京 东 价: **¥5688.00** (降价通知)

促销信息: **赠品** **× 1**

加价购 满99.0元另加69.0元即可购买热销商品 [详情>>](#)

累计评价
40805

共3项促销、

配 送 至: 北京朝阳区三环以内 有货, 支持 79免运费 | 货到付款

服务：由京东发货并提供售后服务，23:00前完成下单，预计明天(07月25日)送达

选择颜色： 金色  银色  深空灰

选择版本: [移动4G版](#) [公开版](#) [电信0元购机版](#)

选择容量: 16GB 64GB 128GB

购买方式:	北京移动购机赠费	非合约机	购机入网送话费	0元购机
-------	----------	------	---------	------

 53%的用户正在使用138元档移动4G套餐或1G流量月包 [去看看](#)

京东服务：**全** 2年全保服务免费上门取修 ¥799.00

iPhone1年意外损坏换新服务 ¥399.00

屏 iPhone屏碎保修服务1年 ¥189.00

 1年延长保免费上门取修 ¥399.00 

保 1年意外保免费上门取修 ¥499.00

[详情](#)

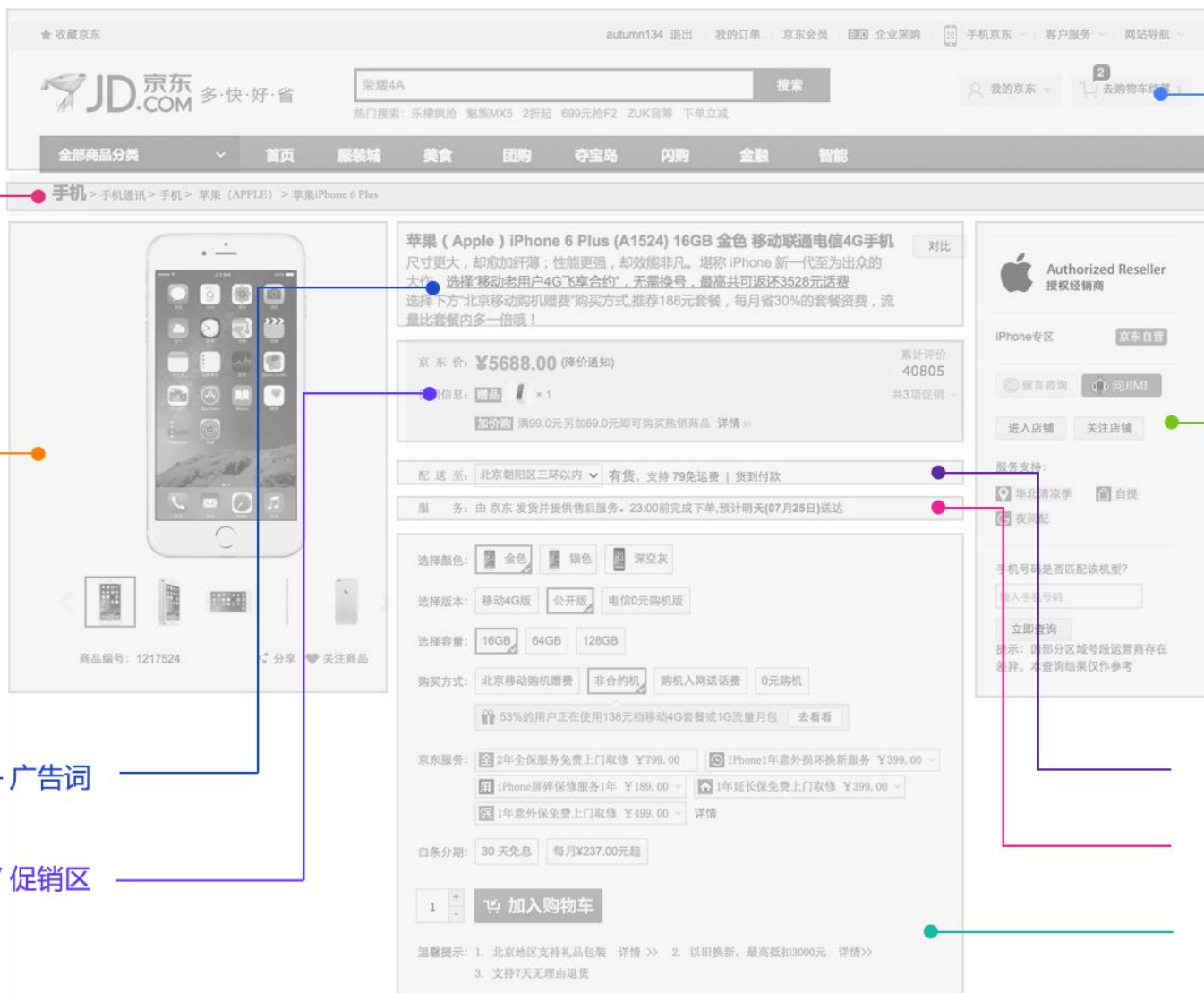
白条分期:	30 天免息	每月¥237.00元起
-------	--------	-------------

1

 加入购物车

温馨提示: 1. 北京地区支持礼品包装 [详情 >>](#) 2. 以旧换新, 最高抵扣3000元 [详情>>](#)
3. 支持7天无理由退货

单品页前端结构 / 复杂的首屏



通用头、店铺头

面包屑

图片区

商品标题 + 广告词

价格 / 促销区

自营 / 商家信息区

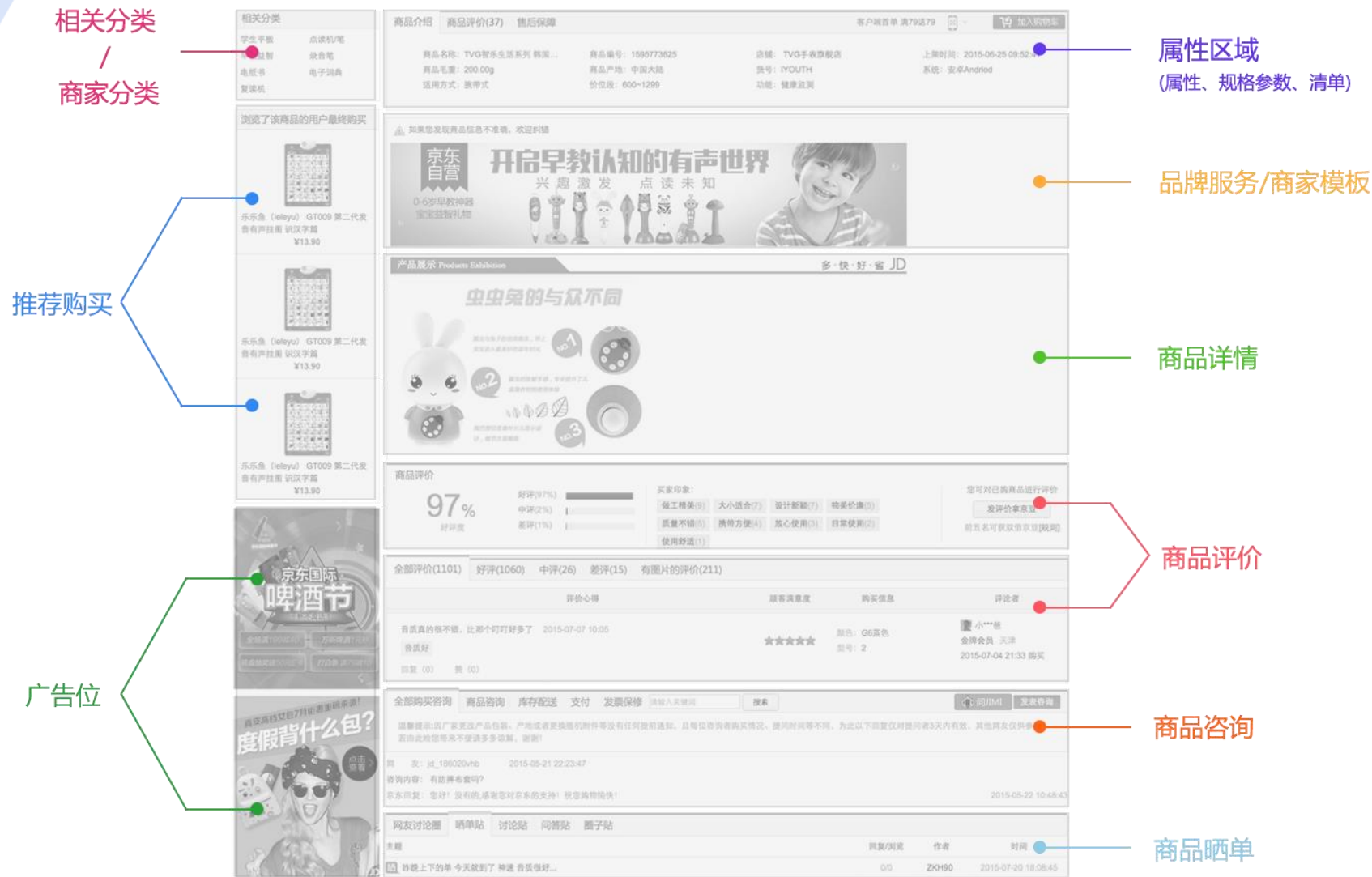
库存配送区

预售 / 服务区等

SKU区



单品页前端结构 / 其他屏





性能数据

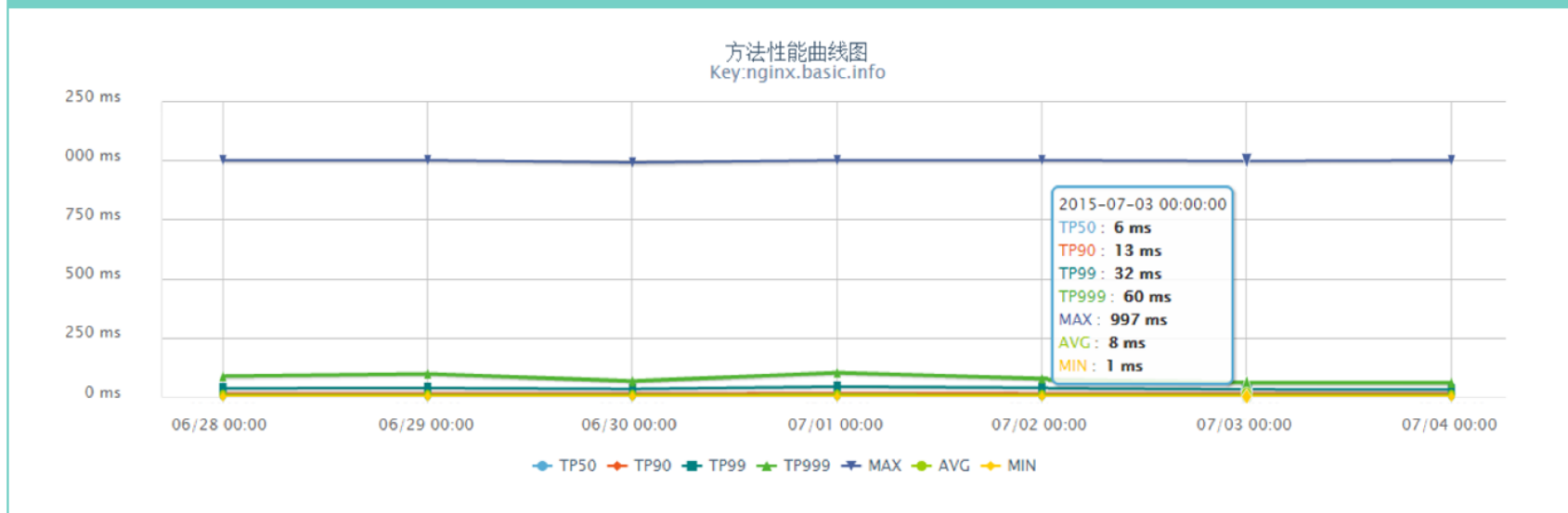
618当天PV

数十亿+

618当天
服务端响应时间

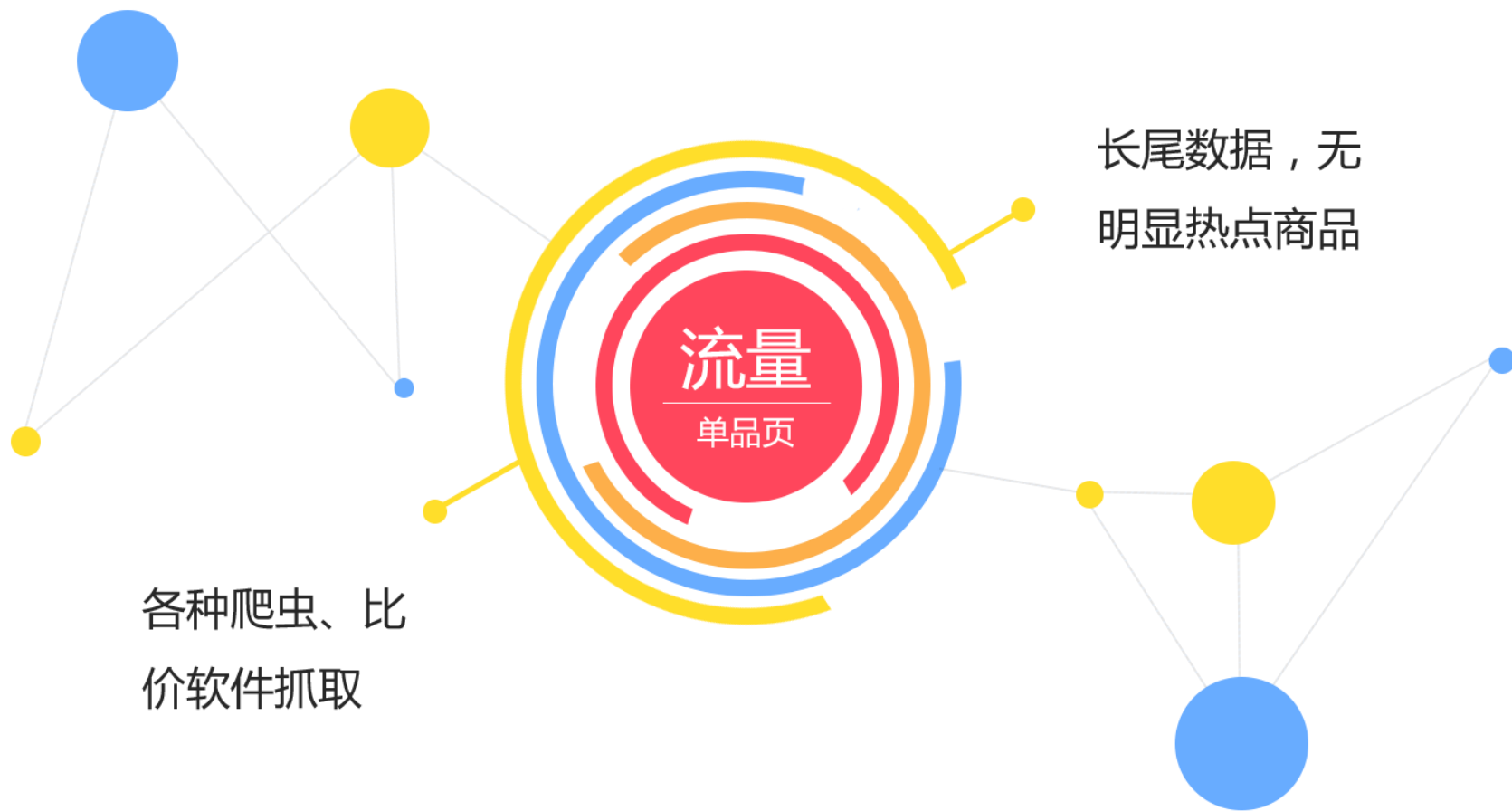
<38ms

平常性能曲线图



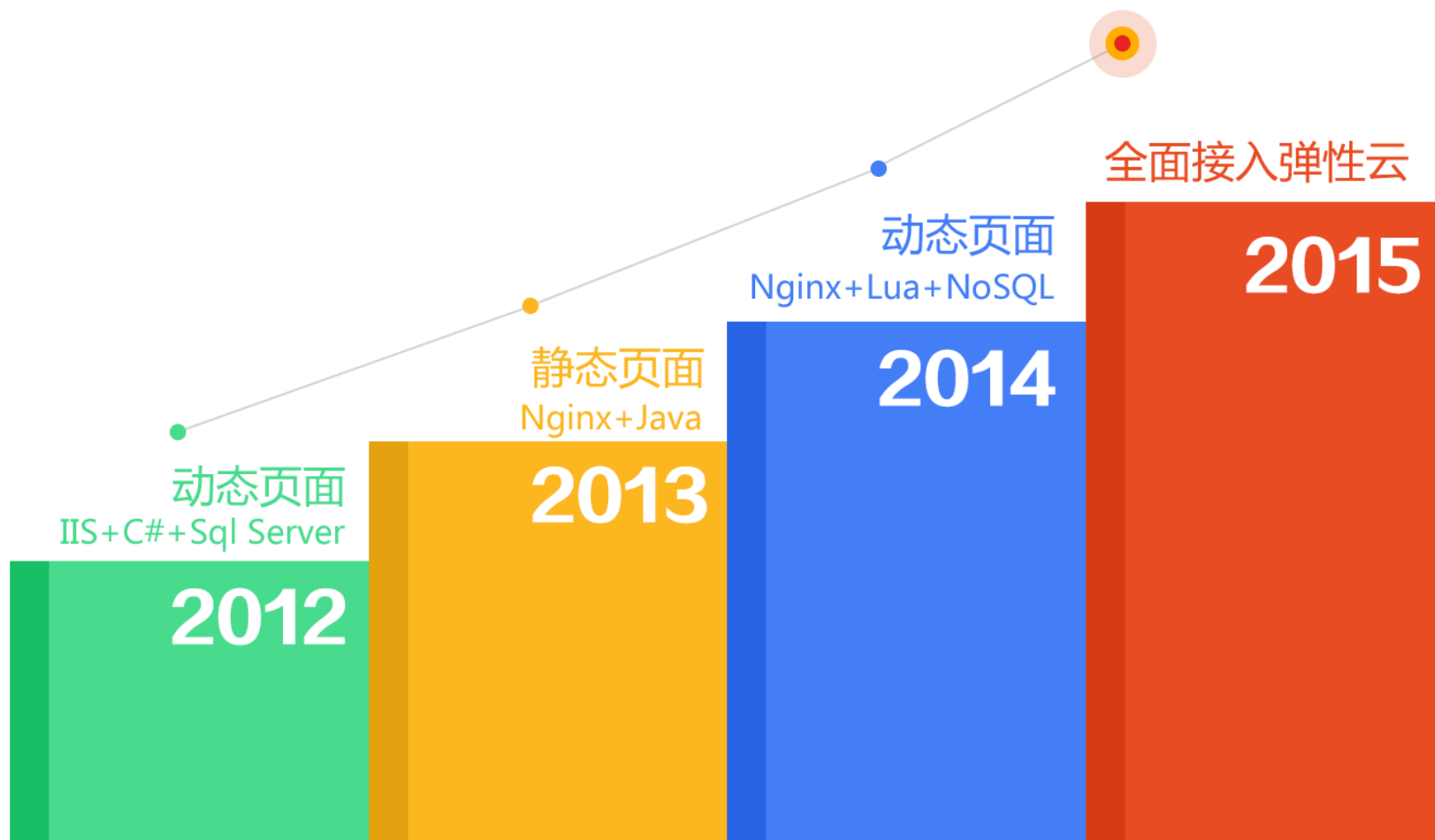


单品页流量特点



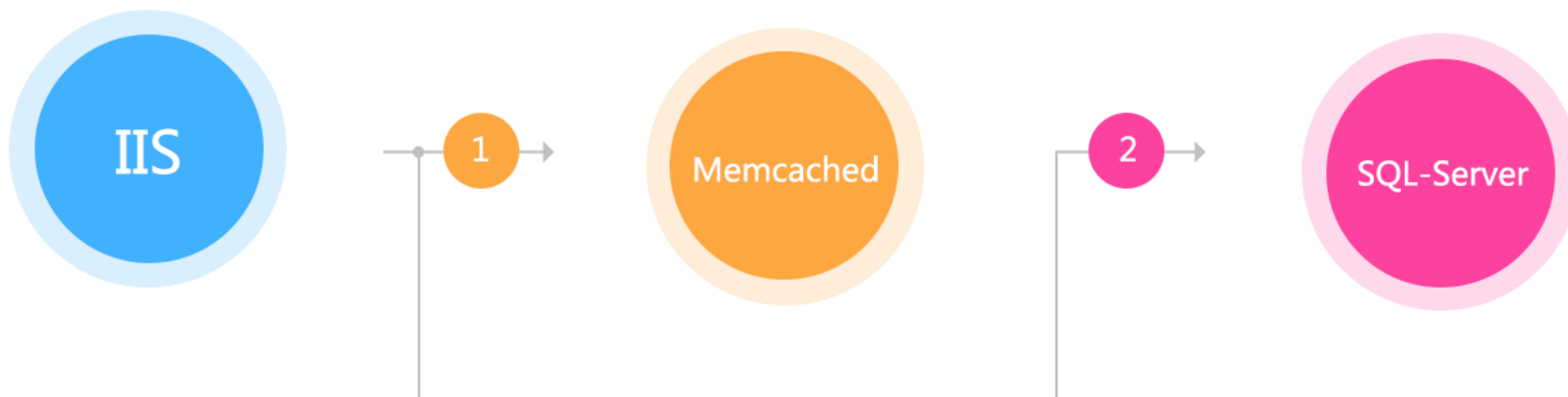


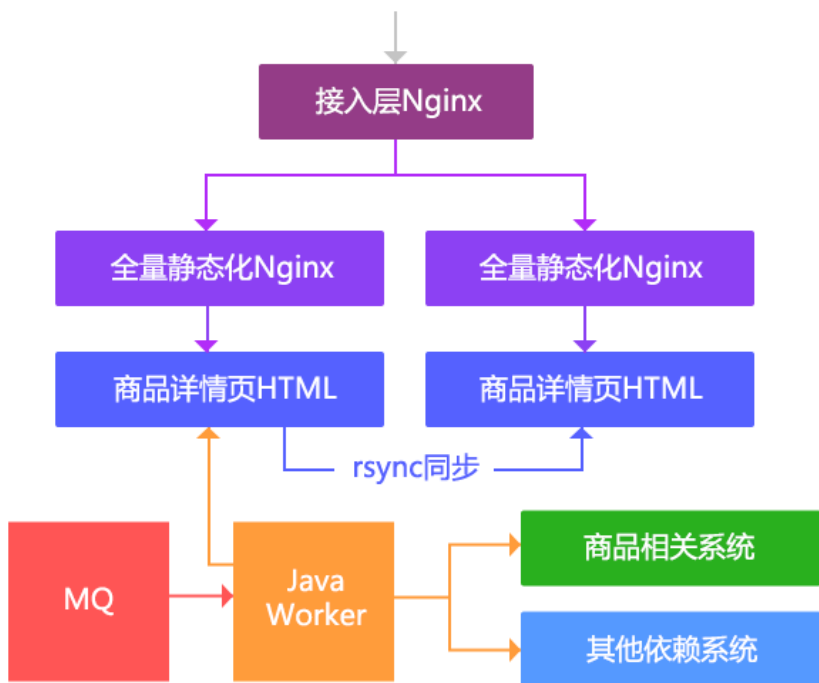
单品页技术架构发展





架构1.0





该方案主要思想：

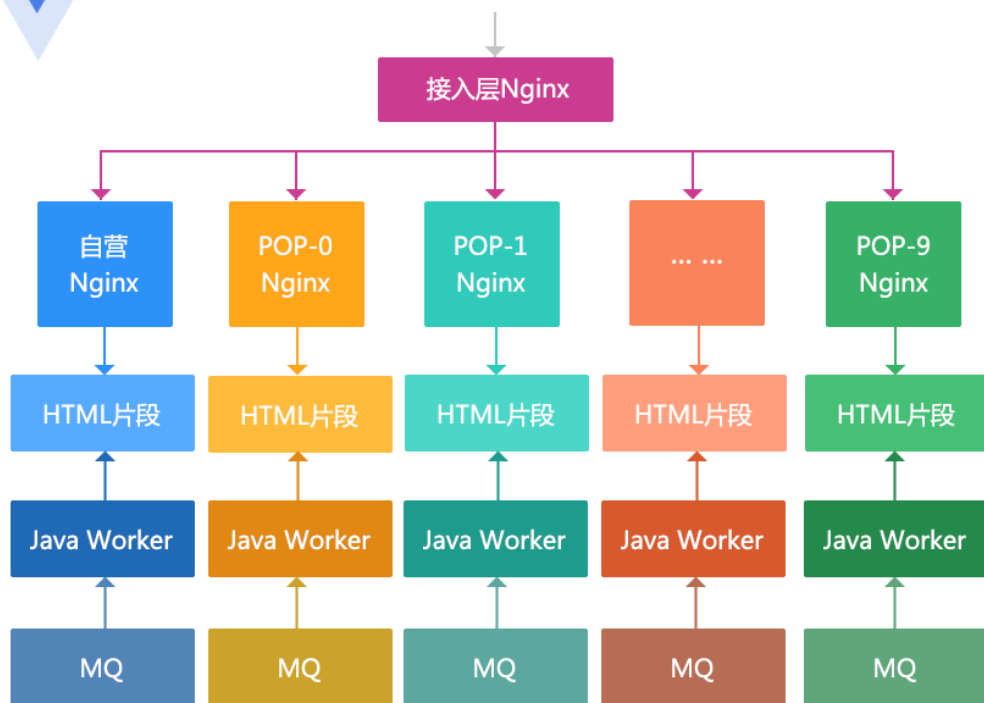
- 通过MQ得到变更通知；
- 通过Java Worker调用多个依赖系统生成详情页HTML；
- 通过rsync同步到其他机器；
- 前端Nginx直接输出静态页；
- 接入层负责负载均衡。

该方案的缺点：

- 假设只有如分类、面包屑、商家信息变更，那么所有相关的静态页都要重新生成；
- 随着商品数量的增加，rsync会成为瓶颈；



架构2.0



该方案主要思想：

- 容量问题通过商品尾号分布到多台机器 (1台自营，10台POP)；
- 按需生成HTML片段（如框架、商品介绍、规格参数、面包屑、相关分类、店铺信息等）；
- 通过SSI合并片段输出；
- 接入层按照尾号做负载均衡；
- 多机房部署多套相同架构来实现高可用。

该方案的缺点：

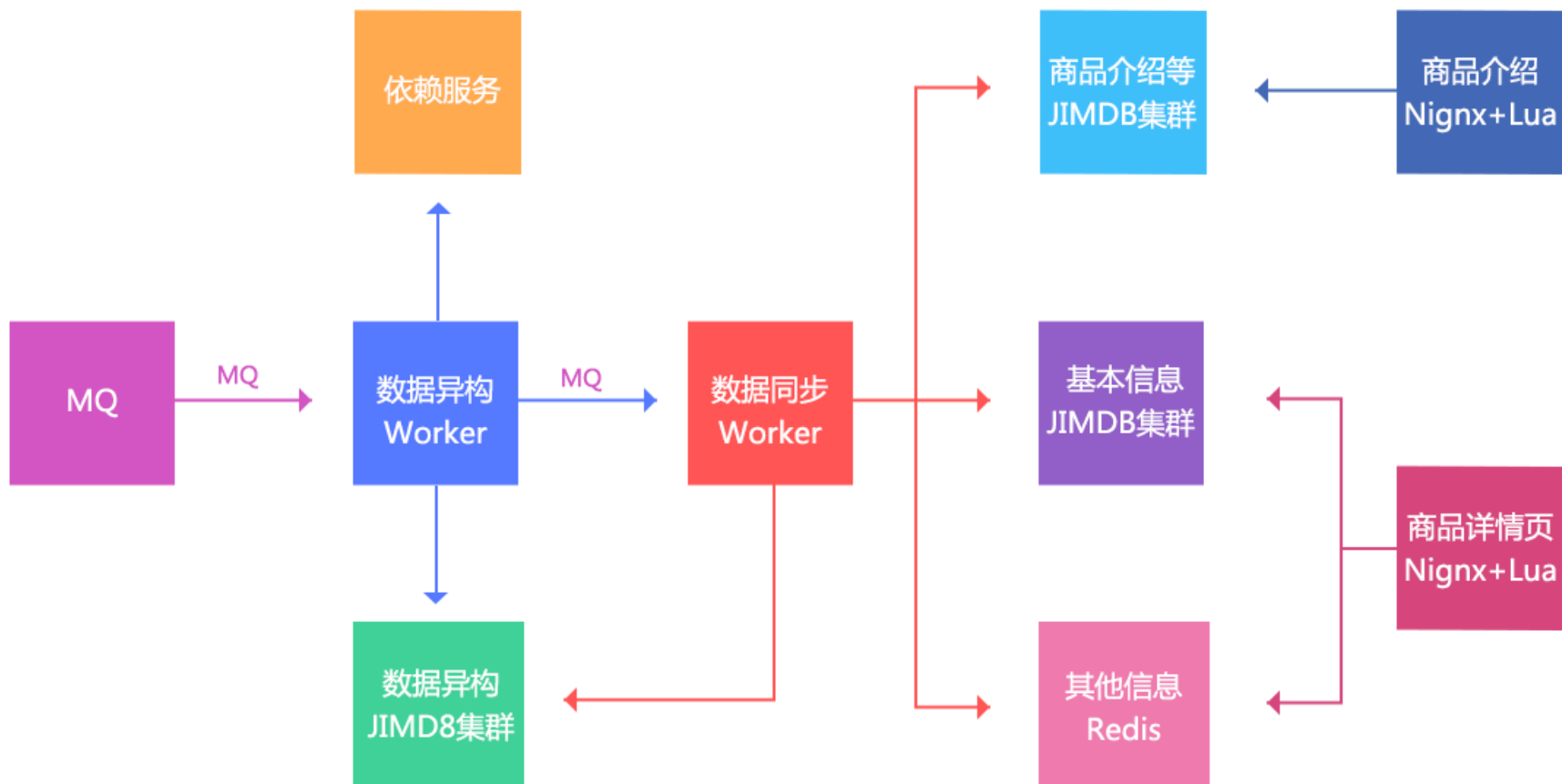
- 碎片文件太多，导致许多问题；
- 机械盘做SSI合并时，高并发时性能差；
- 模板变更需全量生成，几亿商品需数天生成；
- 无法迅速响应瞬变的需求。

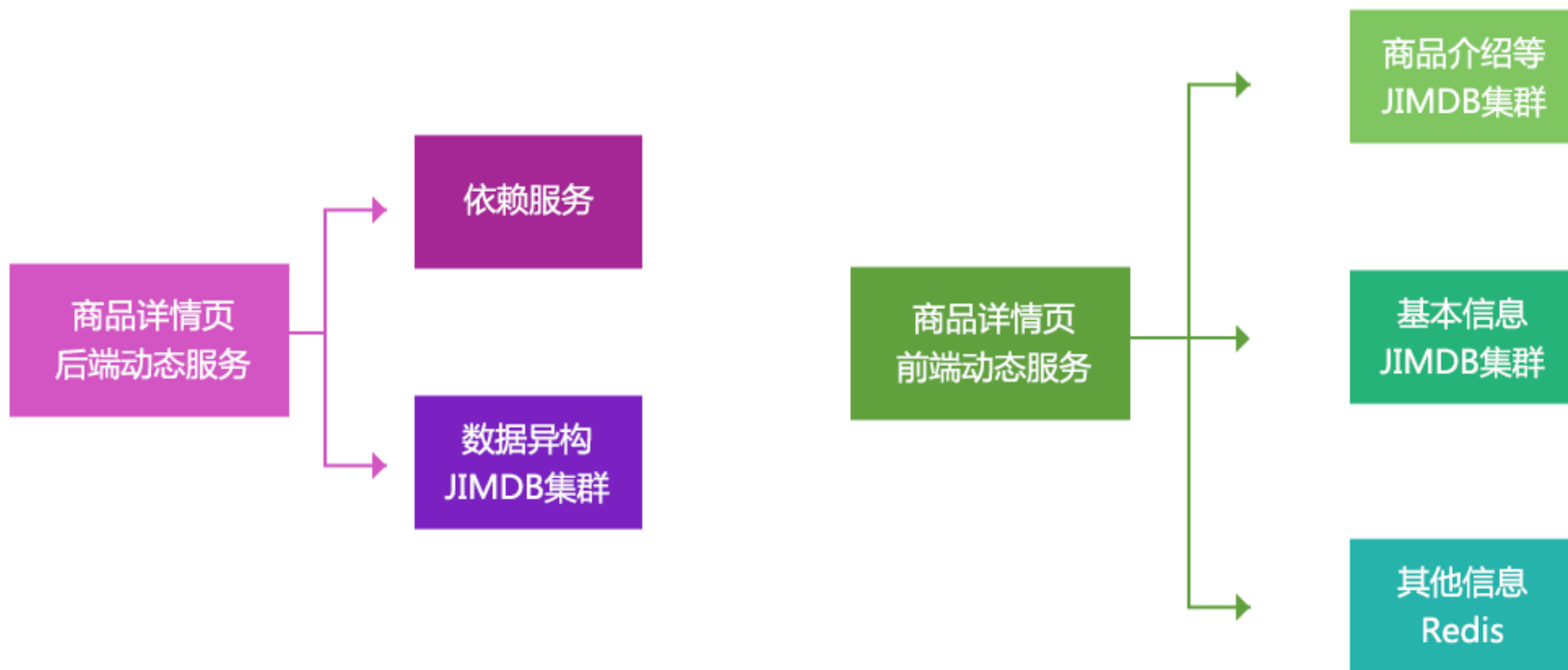


- 能迅速响瞬变的需求
- 支持各种垂直化页面改版
- 页面模块化
- AB测试
- 高性能、水平扩容
- 多机房多活、异地多活



架构3.0





*目前该动态服务为列表页、商品对比、微信单品页、总代等提供相应的数据来满足和支持其业务。



详情页架构设计原则

- 数据闭环
- 数据维度化
- 拆分系统
- Worker无状态化+任务化
- 异步化+并发化
- 多级缓存化
- 动态化
- 弹性化
- 降级开关
- 多机房多活
- 多种压测方案



详情页架构设计原则 / 数据闭环

数据闭环

- 数据自我管理，去依赖

数据异构

- 数据闭环第一步，异构数据

数据原子化

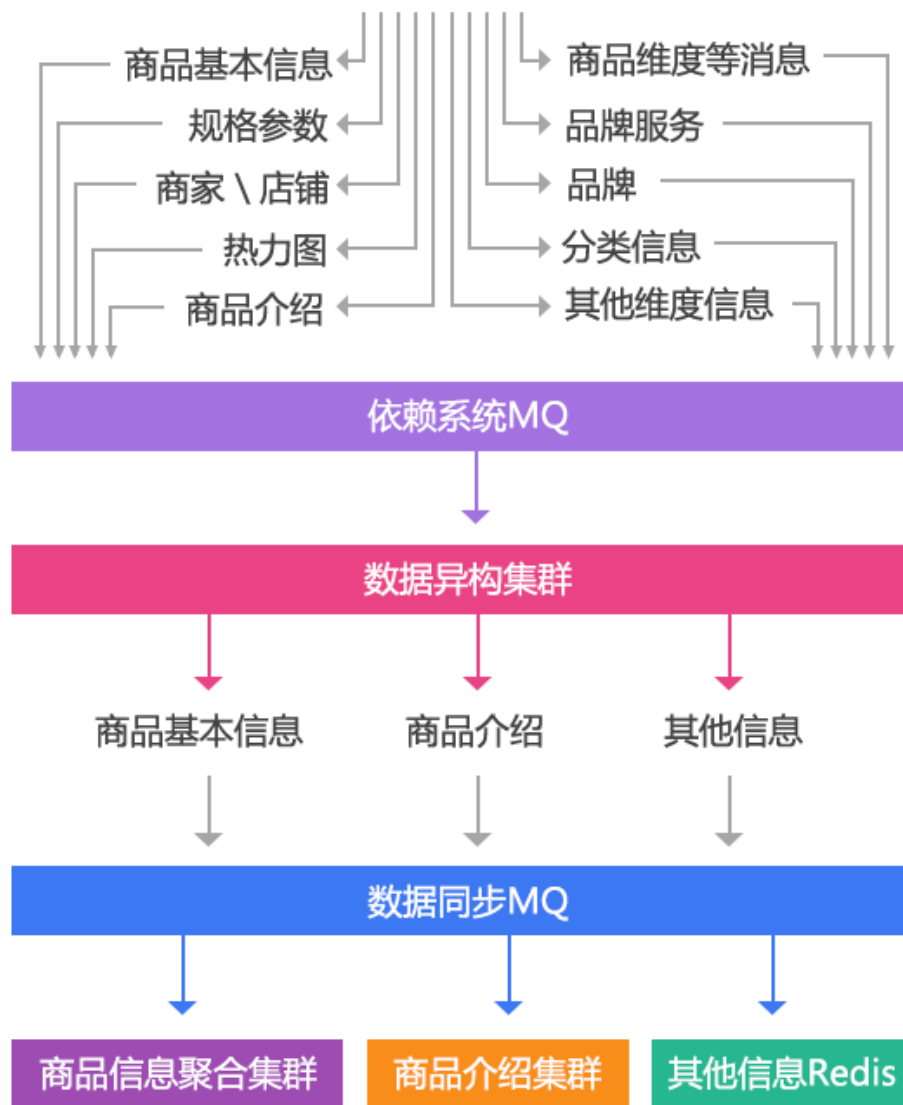
- 原子化保证数据能再加工

数据聚合

- 一次性获取所有需要的数据

数据存储

- KV存储、Hash Tag、SSD





商品基本信息

- 标题、扩展属性、特殊属性、图片、颜色尺码、规格参数等

商品介绍信息

- 商品维度商家模板、商品介绍等

非商品维度其他信息

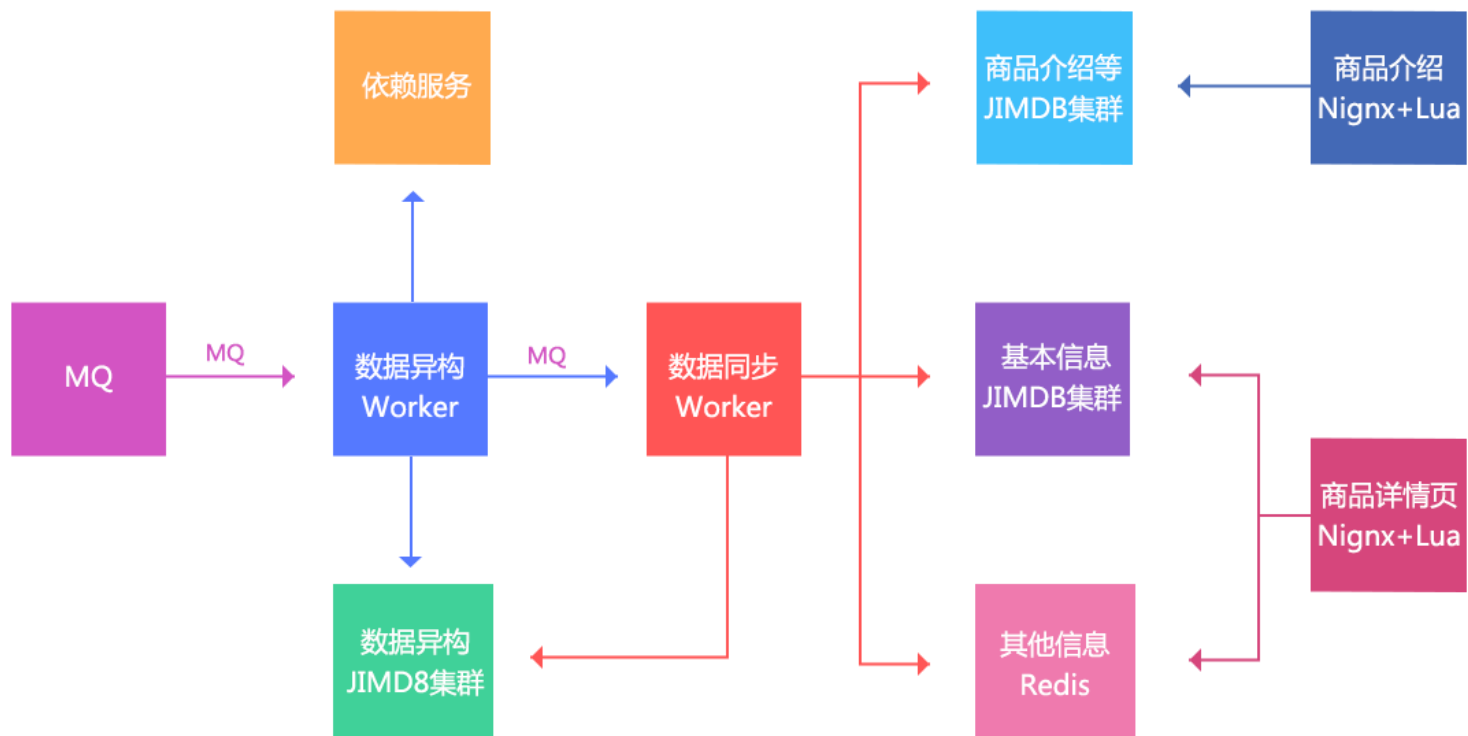
- 分类信息、商家信息、店铺信息、店铺头、品牌信息等

商品维度其他信息（异步加载）

- 价格、库存、广告词、推荐配件、最佳组合灯



详情页架构设计原则 / 拆分系统



数据异构

原子化数据存储

数据同步

原子数据聚合
增量更新

前端展示

商品详情页和商品介绍两个服务，
从数据聚合集群获取数据展示



Worker无状态化设计

- 数据异构和数据同步，可水平扩展
- 应用无状态，配置有状态

任务多队列化

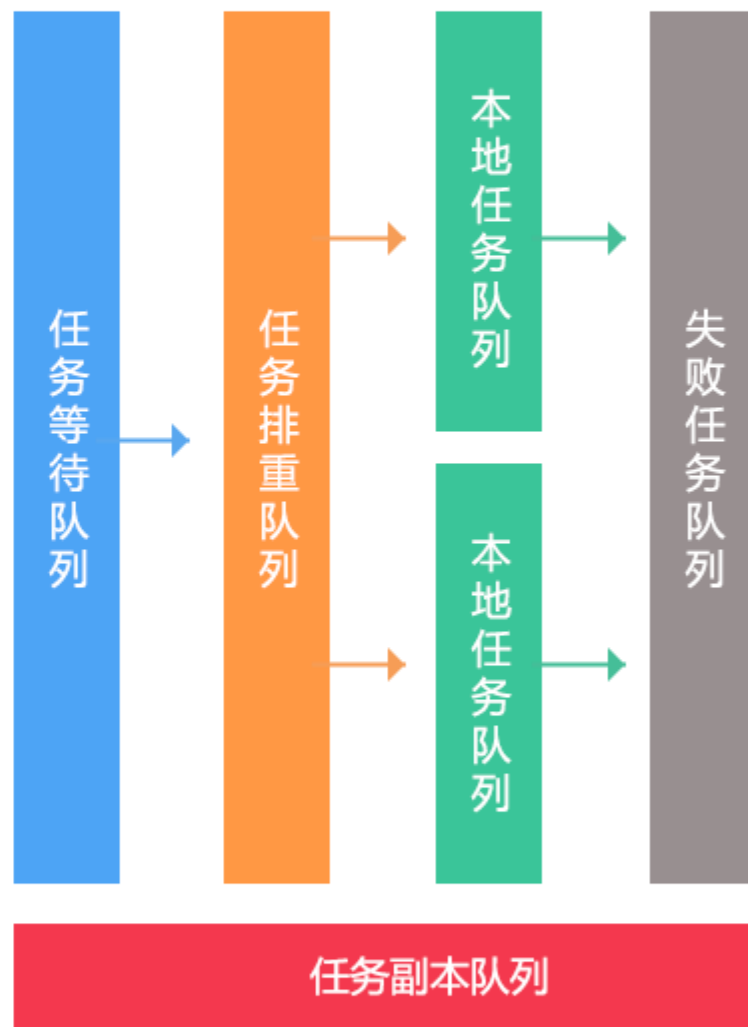
- 等待队列、排重队列、本地执行队列、失败队列

队列优先级化

- 分为：普通队列、刷数据队列、高优先级队列

副本队列

- 出问题，修正逻辑回放





消息异步化

- 系统解耦：同步推，变为异步拉

数据更新异步化

- 更新缓存：同步调用服务，异步更新缓存

可并行任务并发化

- 商品数据系统来源有多处，但是可以并发调用聚合

异步请求合并

- 异步请求做合并，然后一次请求调用

前端服务异步化/聚合

- 实时价格、实时库存异步化
- 使用如线程或协程机制将多个可并发的服务聚合



浏览器缓存

CDN缓存

服务端应用本地缓存

- Nginx+Lua架构，使用HttpLuaModule模块的shared dict做本地缓存（ reload不丢失 ）或内存级Proxy Cache，减少带宽
- 使用一致性哈希（如商品编号/分类）做负载均衡内部对URL重写提升命中率；
- mget优化（先读local cache，不命中的再remote cache）

服务端分布式缓存

- 内存+SSD JIMDB存储



数据获取动态化

- 商品详情页：按维度获取数据，商品基本数据、其他数据（分类、商家信息等）
- 可以根据数据属性，按需做逻辑

模板渲染实时化

- 支持随时变更模板需求

重启应用秒级化

- 使用Nginx+Lua架构，重启速度快，重启不丢共享字典缓存数据

需求上线速度化

- 能迅速响应紧急复杂的需求



接入弹性云（Docker容器）

镜像复制新应用

- 无需重复部署生产环境，直接镜像克隆

按需扩容容器

- 带宽、CPU等



推送服务器推送降级开关

- 开关集中化维护

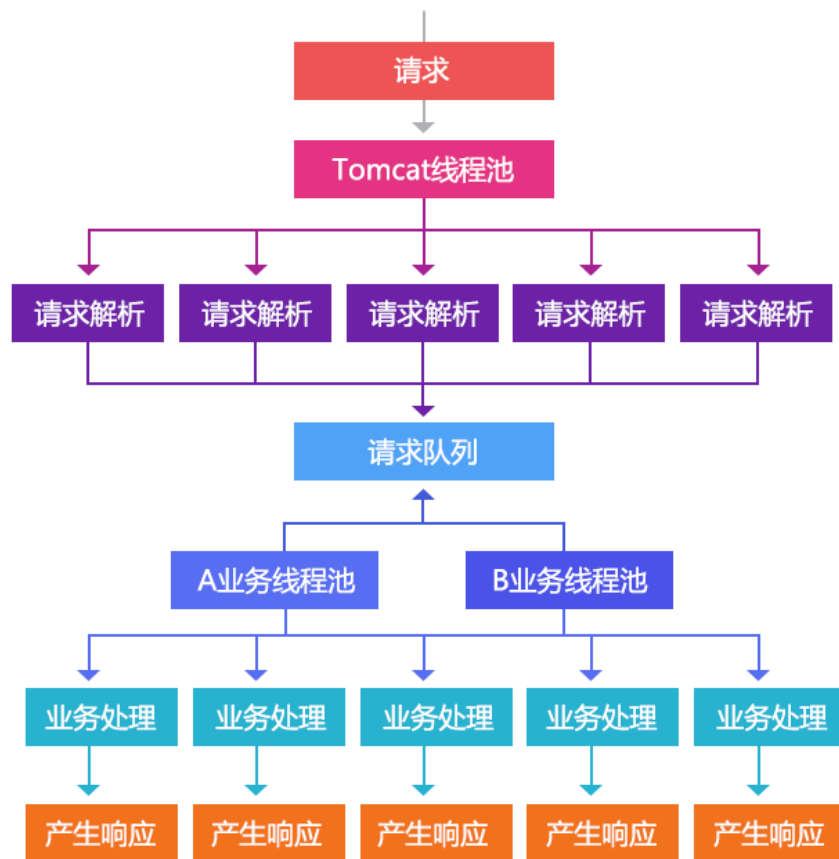
可降级的多级读服务

- 前端数据集群--->数据异构集群--
->动态服务(调用依赖系统)

开关前置化

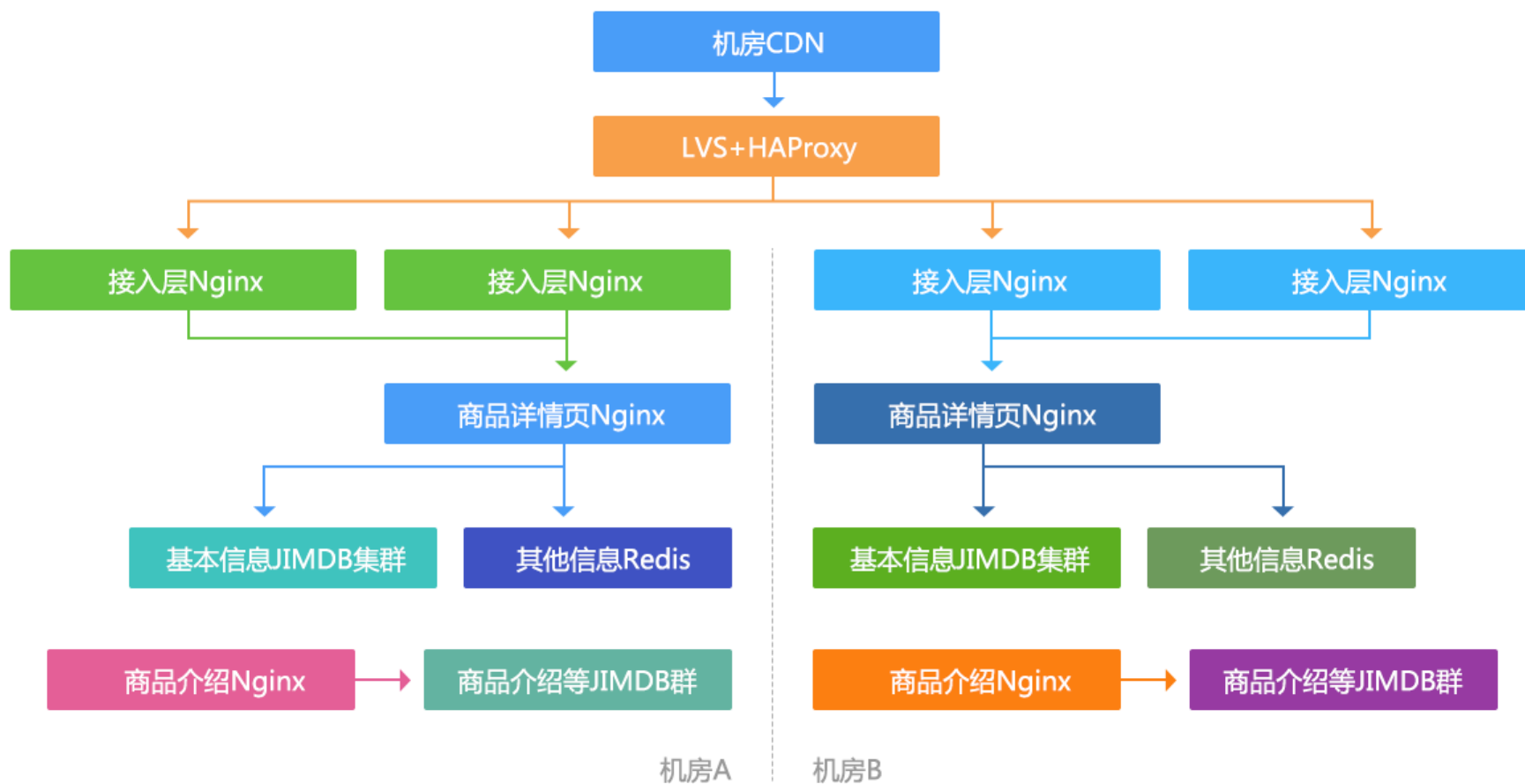
- 如Nginx-->Tomcat, 在Nginx上做开关, 不请求到后端

可降级的业务线程池隔离





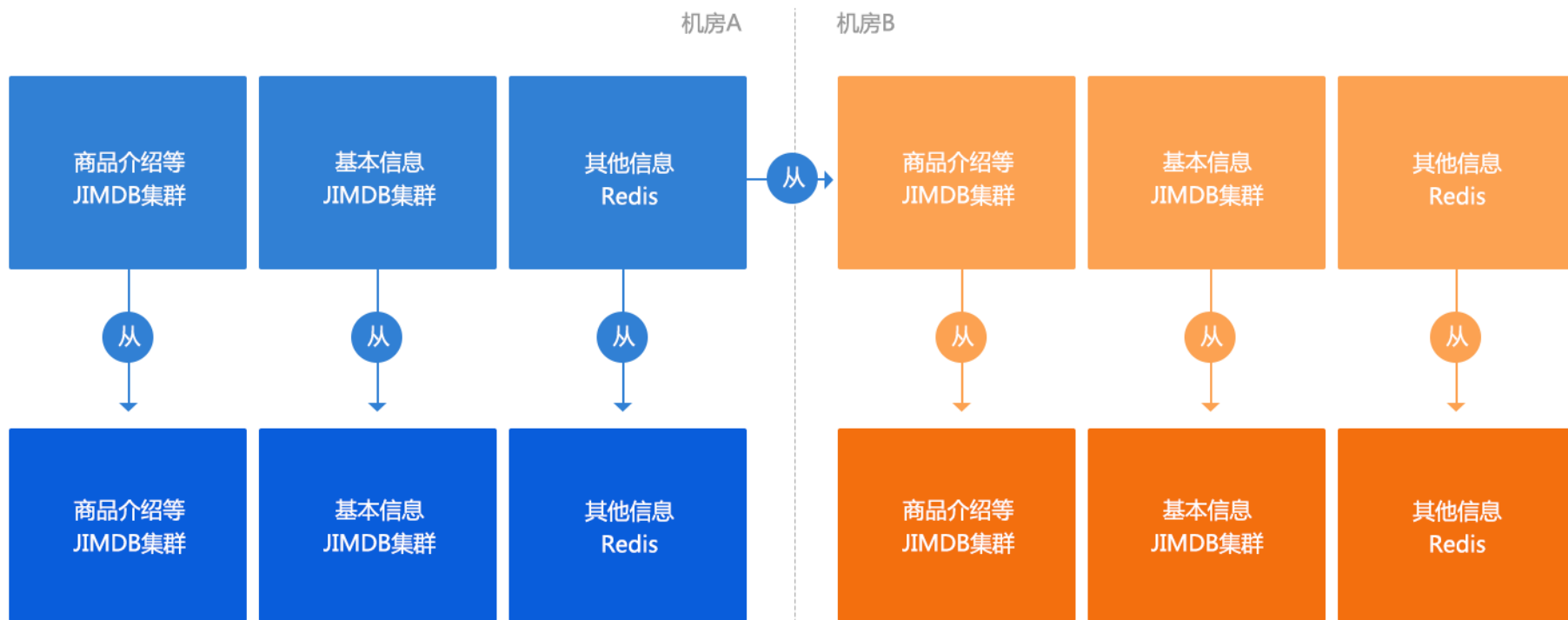
多机房多活 / 前段展示集群部署





多机房多活 / 前段展示集群部署

同城机房一主三从；从提供服务





多种压测方案



线下压测

- Apache ab
- Apache Jmeter



线上压测

- Tpcopy
- Nginx+Lua协程
- 用户端压测（页面直接埋异步请求）



如何压测

- 根据场景进行：
读写分离压测、
读写同时压测



使用SSD做KV存储时发现磁盘IO非常低。

- 配置成RAID10的性能只有3~6MB/s；配置成RAID0的性能有~130MB/s
- 系统中没有发现CPU，MEM，中断等瓶颈。

一台服务器从RAID1改成RAID0后，性能只有~60MB/s.

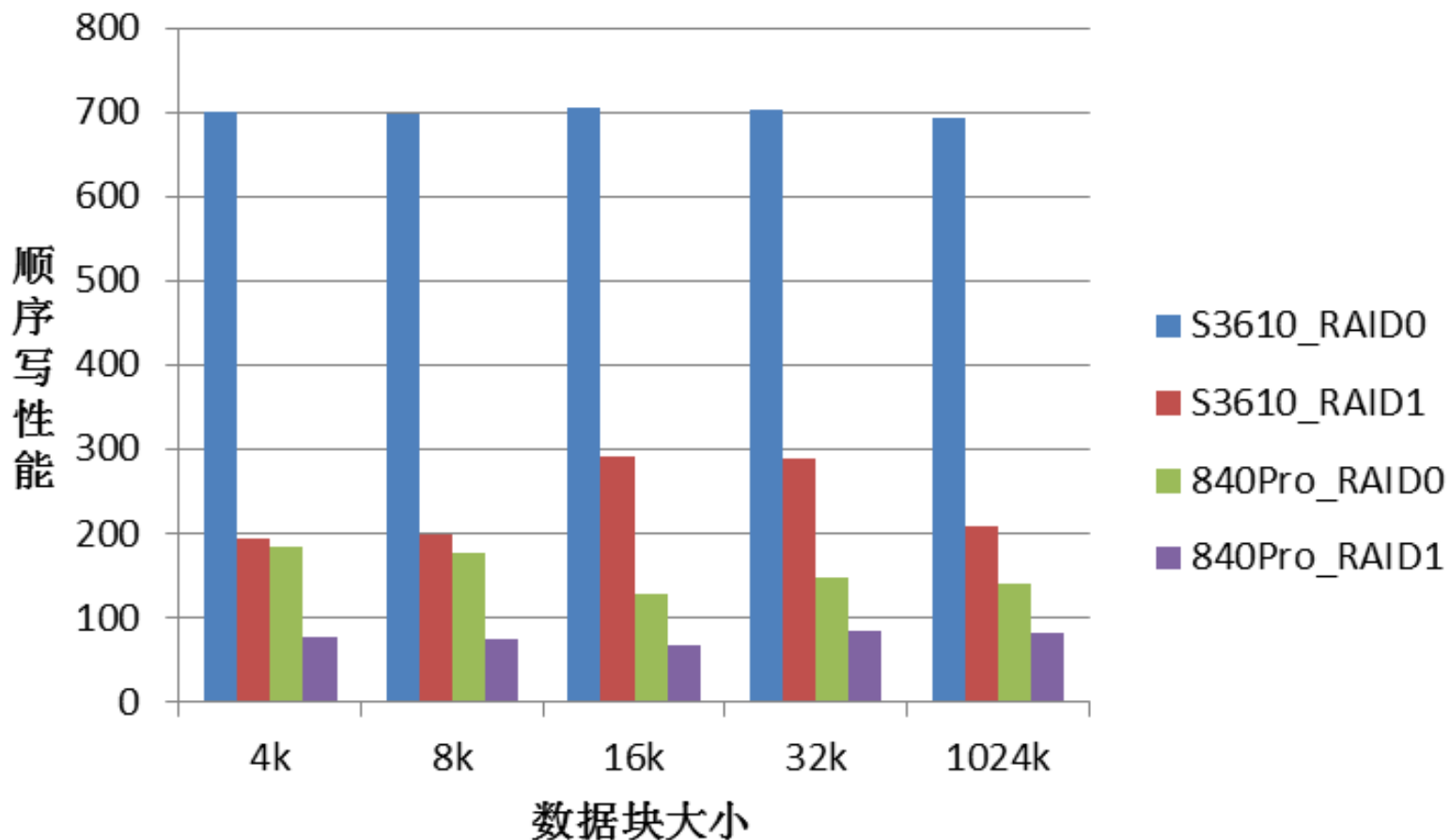
- 说明SSD盘性能不稳定

根据以上现象，初步怀疑以下几点：

- SSD盘，线上系统用的三星840Pro是消费级硬盘。
- RAID卡设置，Write back和Write through策略。(后来测试验证，有影响，但不是关键)
- RAID卡类型，线上系统用的是LSI 2008，比较陈旧。



遇到的一些坑 / KV存储选型压测



* 本实验使用dd顺序写操作简单测试，严格测试需要用FIO等工具

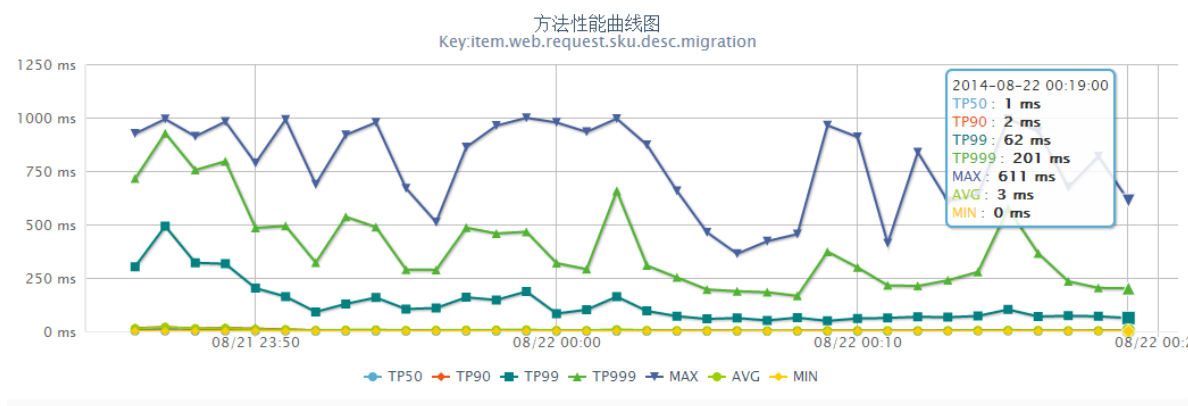
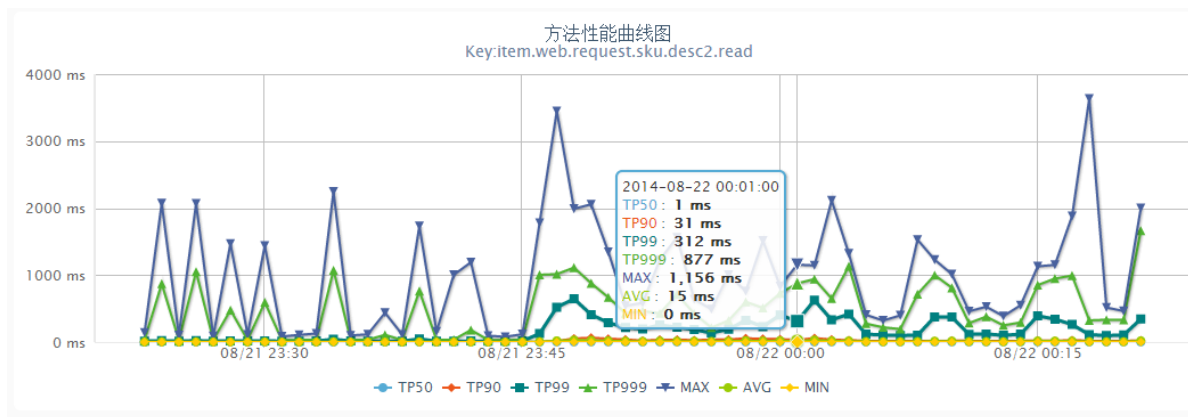


- 机器：2台
- 配置：32核CPU、32GB内存、
SSD（(512GB)三星840Pro-->
(600GB)Intel 3500 / Intel S3610）
- 数据：1.7亿数据（800多G数据）、大小5~30KB左右
- KV存储引擎：LevelDB、RocksDB、LMDB，每台启动2个实例
- 压测工具：tcpcopy直接线上导流
- 压测用例：随机写+随机读



遇到的一些坑 / KV存储选型压测

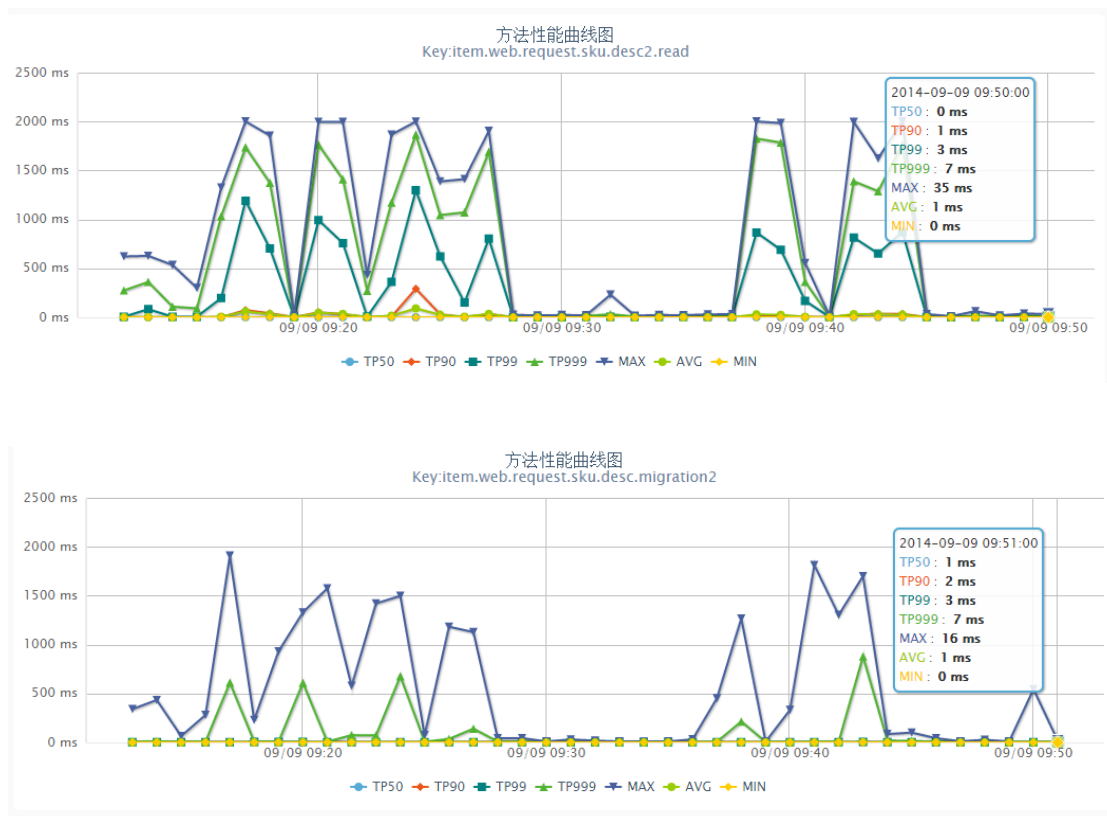
LevelDB 读：50w/分钟，写：5w/分钟





遇到的一些坑 / KV存储选型压测

RocksDB 读：80w/分钟，写：2.3w/分钟

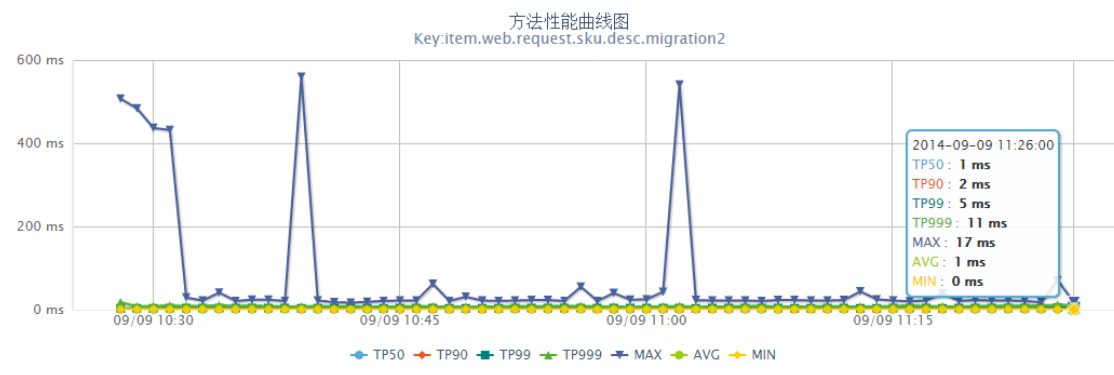
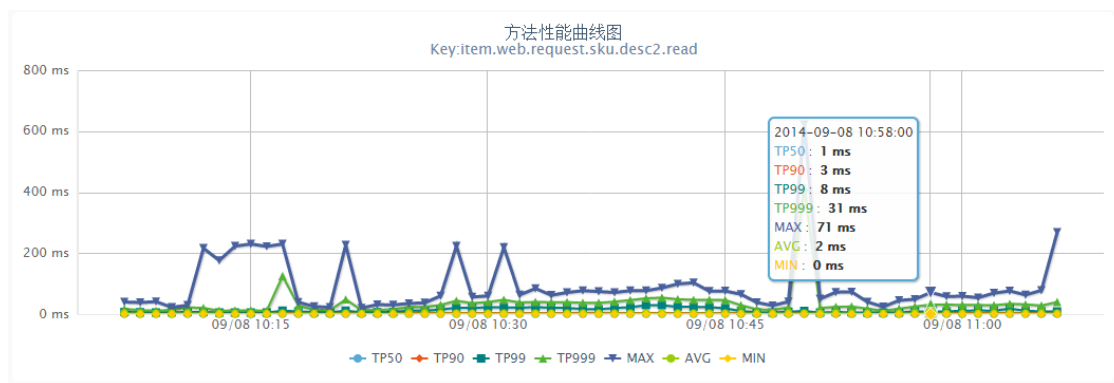


--dsk/sda--	
read	writ
0	0
0	0
0	40k
0	0
0	64k
0	0
0	0
0	12k
0	0
0	0
0	124M
0	117M
4096B	147M
0	129M
0	128M
0	128M
0	129M
0	129M
0	129M
0	129M
0	129M
0	129M
0	135M
0	120M
4096B	142M
0	0
0	0
0	0
0	0
0	20k
0	168k
0	0
0	0
0	0
0	0
0	125M



遇到的一些坑 / KV存储选型压测

LMDB 读：80w/分钟，写：9w/分钟





遇到的一些坑 / 数据量大时Jimdb同步不动



问题

Jimdb数据同步时要dump数据，SSD盘容量用了50%以上，dump到同一块磁盘容量不足。



解决

- 1、一台物理机挂2块SSD(512GB)，单挂raid0；启动8个jimdb实例；这样每实例差不多125GB左右；
- 2、目前是千兆网卡同步，同步峰值在100MB/s左右；
- 3、dump和sync数据时是顺序读写，因此挂一块SAS盘专门来同步数据；
- 4、使用文件锁保证一台物理机多个实例同时只有一个dump；



遇到的一些坑 / 切换主从



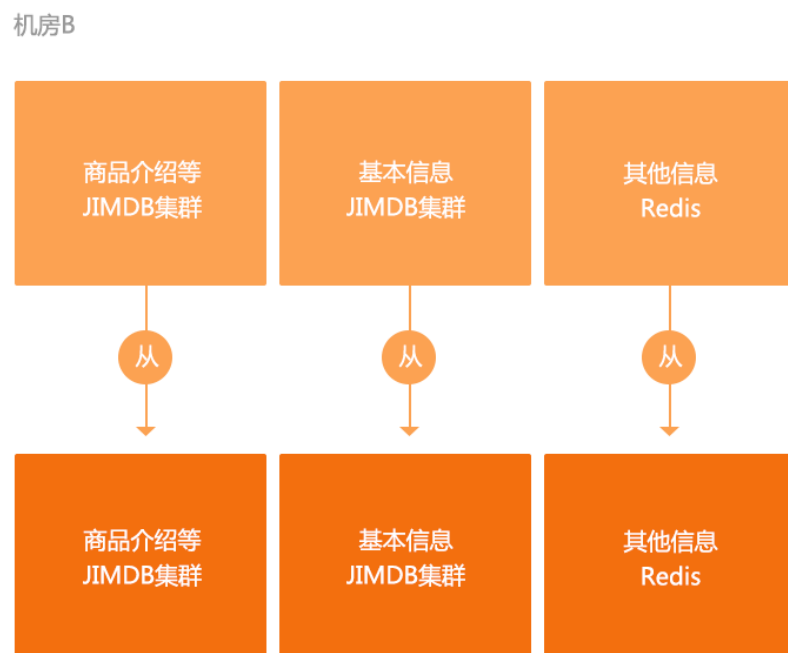
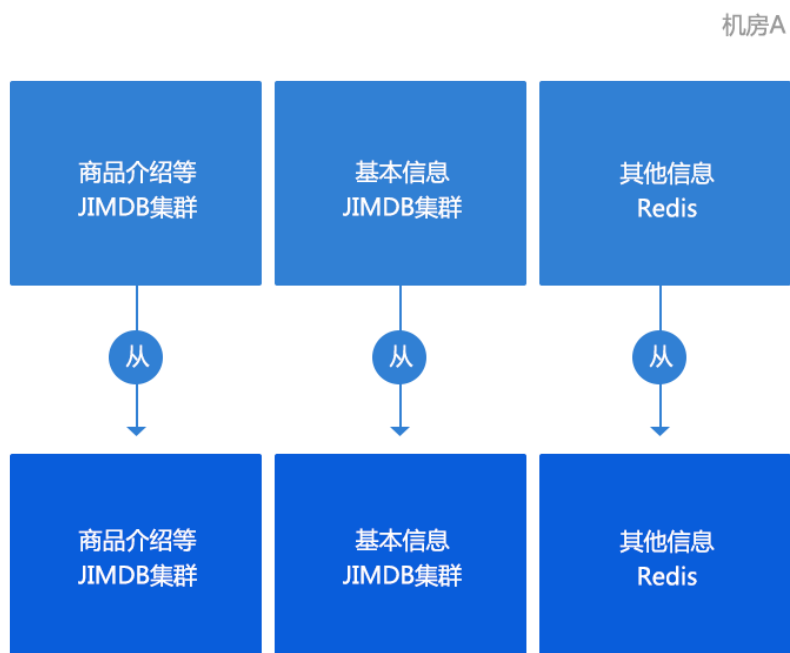
问题

一主三从（主机房一主一从，备机房一从）切换主从时直接读主性能很差。



解决

备机房从上再挂一个从。





遇到的一些坑 / 分片配置



问题

分片逻辑分散到多个子系统，切换时需要操作很多系统。



解决

- 1、使用本地部署的Twemproxy来维护分片逻辑；
- 2、使用自动部署系统推送配置和重启应用，重启之前暂停mq消费保证数据一致性；
- 3、用unix domain socket减少连接数和端口占用不释放启动不了服务的问题；



遇到的一些坑 / 模板元数据存储HTML



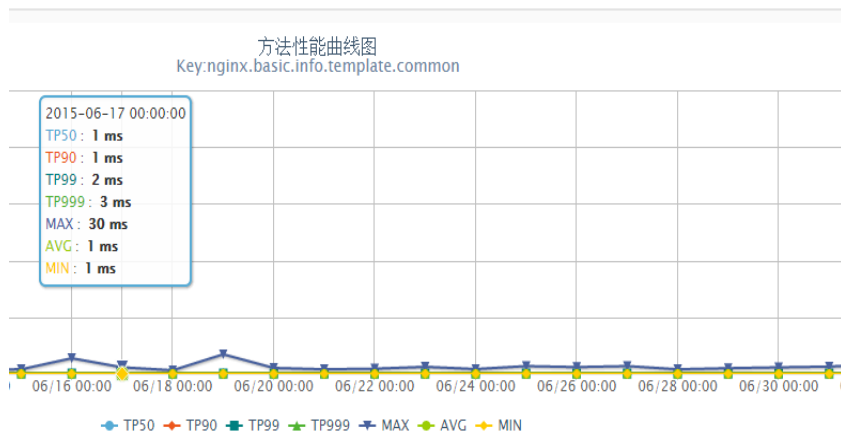
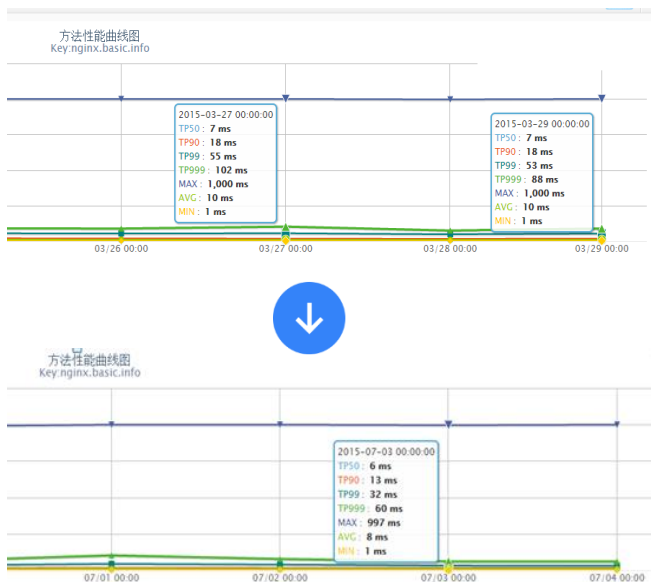
问题

起初不确定Lua做逻辑和渲染模板性能如何，就尽量减少for、if/else之类的逻辑；通过java worker组装html片段存储到jimdb；



解决

通过线上不断压测，最终jimdb只存储元数据，lua做逻辑和渲染；逻辑代码在3000行以上；模板代码1500行以上，其中大量for、if/else，目前渲染性能可以接受。





遇到的一些坑 / 库存接口访问量600w/分钟



问题

曾经商品详情页库存接口
2014年被恶意刷，每分钟
超过600w访问量，tomcat
机器只能定时重启；



解决

因为是详情页展示的数据，缓存
几秒钟是可以接受的，因此开启
nginx proxy cache来解决该问
题，开启后降到正常水平；



后续优化

url重写+一致性哈希负载均衡：不怕随机URL、提升10%+的命中率。



遇到的一些坑 / 微信接口调用量暴增



问题

通过访问日志发现某IP频繁抓取；而且按照商品编号遍历，但是会有一些不存在的编号；



解决

- 1、读取KV存储的部分不限流；
- 2、回源到服务接口的进行请求限流，保证服务质量；



遇到的一些坑 / 开启Nginx Proxy Cache性能不升反降



问题

开启Nginx Proxy Cache后，性能下降，而且过一段内存使用率到达98%；



解决

通过/proc/slabinfo可以看到inode_cache和dentry_cache占了大多数内存；

1、通过修改内核参数

```
sysctl -w vm.extra_free_kbytes=6436787
```

```
sysctl -w vm.vfs_cache_pressure=10000
```

2、通过tmpfs缓存或nginx共享字典缓存元数据



遇到的一些坑 / 配送至读服务因依赖服务太多，响应时间偏慢



问题

- 配送至服务每天有数十亿调用量，响应时间偏慢。



维度1
数据



维度2
数据



维度1
数据



维度3
数据



维度2
数据



维度4
依赖数据



维度4
数据



维度3
数据

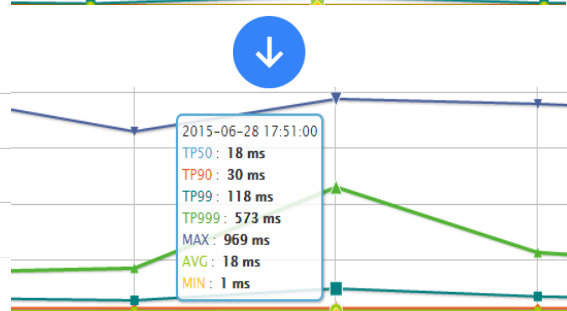
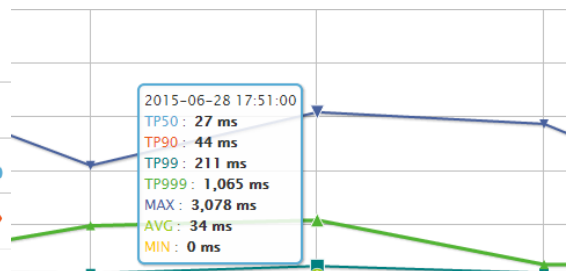
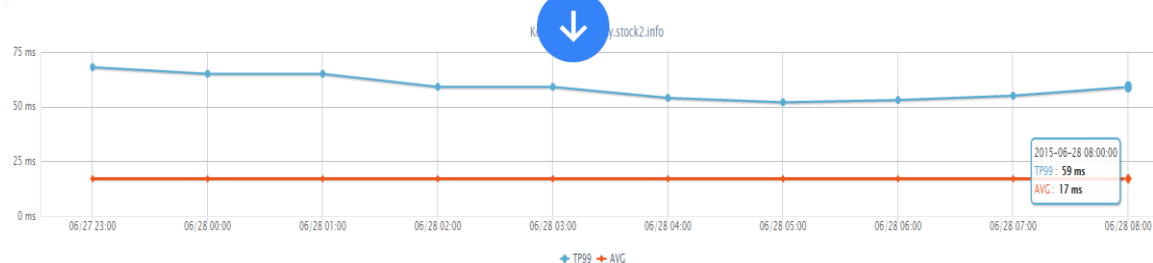
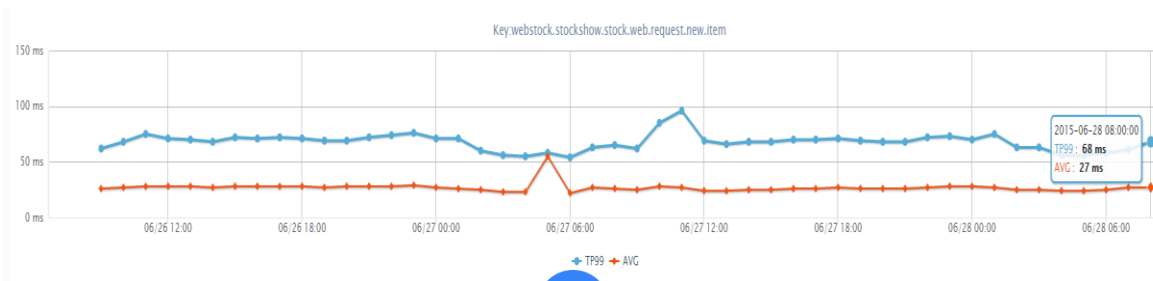


维度4
数据



解决

- 串行获取变并发获取；
- 预取依赖数据回传；





遇到的一些坑 / 网络抖动时，返回502错误



问题

Twemproxy配置的timeout时间太长，而且没有分别针对连接、读、写设置超时。



解决

减少超时时间，内网设置在150ms以内，当超时时访问动态服务。



遇到的一些坑 / 机器流量太大



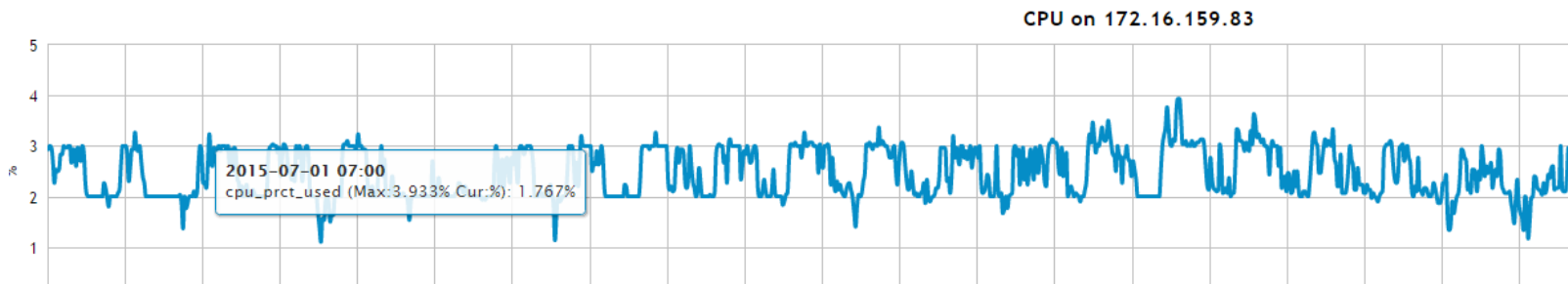
问题

2014年双11期间，服务器网卡流量到了400Mbps，CPU 30%左右。原因是接入层没有传入相关请求头，导致数据没有进行GZIP压缩。



解决

添加相应的请求头，Nginx GZIP压缩级别在2~4吞吐量最高；应用服务器流量降了差不多5倍；目前正常情况CPU在4%以下。





其他

- Nginx接入层线上灰度引流
- 接入层转发时只保留有用请求头
- 使用不需要cookie的无状态域名（如c.3.cn），减少入口带宽
- Nginx Proxy Cache只缓存有效数据，如托底数据不缓存
- 使用非阻塞锁应对local cache失效时突发请求到后端应用(lua-resty-lock/proxy_cache_lock)
- 使用Twemproxy减少Redis连接数
- 使用unix domain socket套接字减少本机TCP连接数
- 设置合理的超时时间（连接、读、写）
- 使用长连接减少内部服务的连接数
- 去数据库依赖，服务化
- 客户端同域连接限制，进行域名分区：c0.3.cn c1.3.cn



E-mail : zhangkaitao@jd.com

PPT细化: <http://jinnianshilongnian.iteye.com/blog/2235572>

博客: <http://jinnianshilongnian.iteye.com/>