# 看京东系统架构师如何让笨重的架构变得灵巧

京东技术 6月8日

来这里找志同道合的小伙伴！

作者：徐贤军

京东系统架构师，从事架构设计与开发工作，熟悉各种开源软件架构。在Web开发、架构优化上有较丰富实战经历。

随着业务的复杂性增大、系统吞吐量增长，所有功能统一部署难度加大，各个功能模块相互影响，使系统变的笨重且脆弱；因此需要对业务进行拆分、对系统进行解耦、对系统内部架构升级，来提升系统容量及健壮性。

接下来主要分两部分介绍：系统拆分与结构演变；

## 系统拆分

系统拆分从资源角度分为：应用拆分和数据库拆分；

从采用的先后顺序可分为：水平扩展、垂直拆分、业务拆分、水平拆分；


图1 系统分解原则

### 1、水平扩展

水平扩展是最初始的解决的手段，也是系统遇到瓶颈的首选方案，主要从以下两个方面扩展：

- 应用加实例，搞集群，把系统吞吐量扩上去。

- 数据库利用主从进行读写分离，数据库其实是系统最应该保护的资源。

## 2、垂直拆分

垂直拆分才是真正开始拆分系统，主要是从业务功能角度拆分。如拆出用户系统、商品系统、交易系统等。为了解决拆分后各个子系统之间相互依赖调用的问题，这时会引入服务调用治理。系统复杂度有所加大，但系统基本解耦，稳定性相对提高，做好降级就能避免因其它系统功能异常导致系统崩溃。

业务对应的库也会按照对应的业务进行拆分出用户库、商品库、交易库等。

## 3、业务拆分

业务拆分主要是针对应用层面按功能特点拆分，如交易拆分出：购物车、结算页、订单、秒杀等系统。然后根据业务的特点，针对性做处理，如秒杀系统，由于同时参加秒杀的商品有限，可以提前把商品信息加载到JVM缓存中，自身减少外部调用提高性能，同时商品系统也减轻压力。

数据库拆分也可以分为几步：垂直分表、垂直分库、水平分表、水平分库分表；

**垂直分表**是指大表拆多张小表，可以根据字段更新或查询频次拆分；

表 1 商品表拆分前字段

| 商品编号 | 商品名称 | 品牌 | 类目 | 图片 | 广告语 | 重量 | 商家信息 | 产地 | 包装 |
|---|---|---|---|---|---|---|---|---|---|

表 2 商品拆分后字段说明

| 商品主表 | | 商品扩展表 | |
|---|---|---|---|
| 表字段 | 字段说明 | 表字段 | 字段说明 |
| 商品编号 | 唯一标识 | 商品编号 | 唯一标识 |
| 商品名称 | 常更新 | 重量 | 查询较少 |
| 品牌 | 查询多 | 产地 | 查询较少 |
| 类目 | 查询多 | 包装 | 查询较少 |
| 图片 | 常更新 | | |
| 广告语 | 常更新 | | |
| 商家信息 | 查询多 | | |

图2 商品表拆分

**垂直分库**是指按业务拆库，如拆出订单库、商品库、用户库等

**水平分表**是解决数据量大，把一张表拆成多张表；

**水平分库分表**是更进一步拆分表；


图3 分库分表

### 4、水平拆分

服务分层，系统服务积木化，拆分功能与非功能系统，以及业务组合的系统，如最近比较火的大中台或前台拆分；中台为积木组件，承担服务功能输出。前台更多的是组合积木服务，及时响应业务发展，如在电商网站单品页能看见主图、价格、库存、优惠券或推荐等信息，都是组合各积木组件呈现。

数据库也可以进行冷热数据分离；过期或过季商品可以归档，比如诺基亚3210手机，早已经停产且没有销售；用户查看订单时，更多的只是查看最近1、2年信息，2年前数据查看量少，在存储设计时可以区别处理。

## 结构演变

结构演变主要是随着系统复杂度增加及对性能要求提高而不得不做的系统内部架构升级；

早期系统基本是应用直联数据库，但在系统进行拆分后，功能本系统不能单独完成，需要依赖其它系统，就出现远程调用；

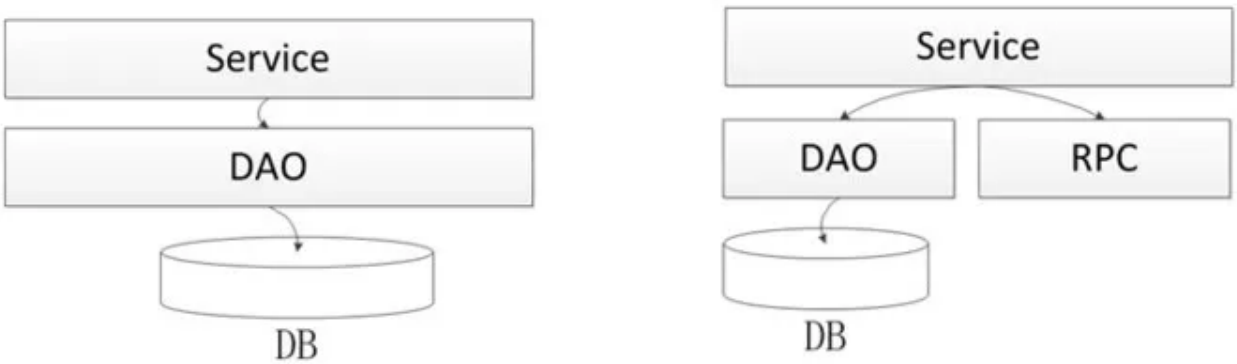


图4 早期应用结构

随着自身系统的业务发展，对性能要求高，而数据库一定程度上成为瓶颈，就会引入缓存及索引，分别解决key-value及复杂检索；索引加缓存现在已经成为解决高并发的基本方案，但在实施过程会有所区别；

14年对3亿热数据的系统升级时，技术选型为solr+redis，考虑到数据量过大，数据在solr中只存index，而结果只存并返回主键id，再通过id从redis中读取数据，redis也不存放全部数据，数据设置过期时间，若未命中redis，回源数据库查询并反写

redis；主要考虑资源与性能的平衡，solr的存储减少及IO性能提高，结果数据只在redis存放一份，redis的数据经过运行大部分是热数据；当然现在也流行ES+Hbase组合。

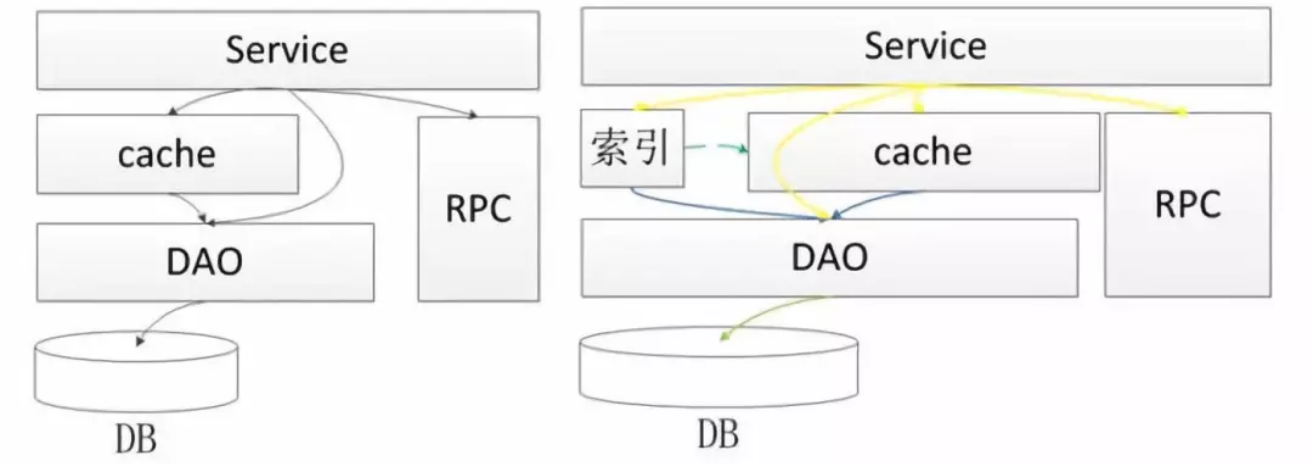


图5 增加缓存及索引

对于频繁使用的数据，从集中缓存读取，不一定达到性能要求，可以考虑把数据入JVM缓存，如类目信息，类目是电商系统基本数据，数据量不多，调用量大；

个别情况下，使用ThreadLocal做线程内缓存也是种有效手段，但需要考虑数据清除及有效性；

在修改商品信息时，业务对商品信息的校验有名称长度、状态、库存及各业务模式等，而为了参数的统一校验方法参数为商品编号，导致各校验方法都需要读取一次商品，使用线程缓存可以解决该问题，性能提高了尽20ms，读取商品每分钟减少近万次；

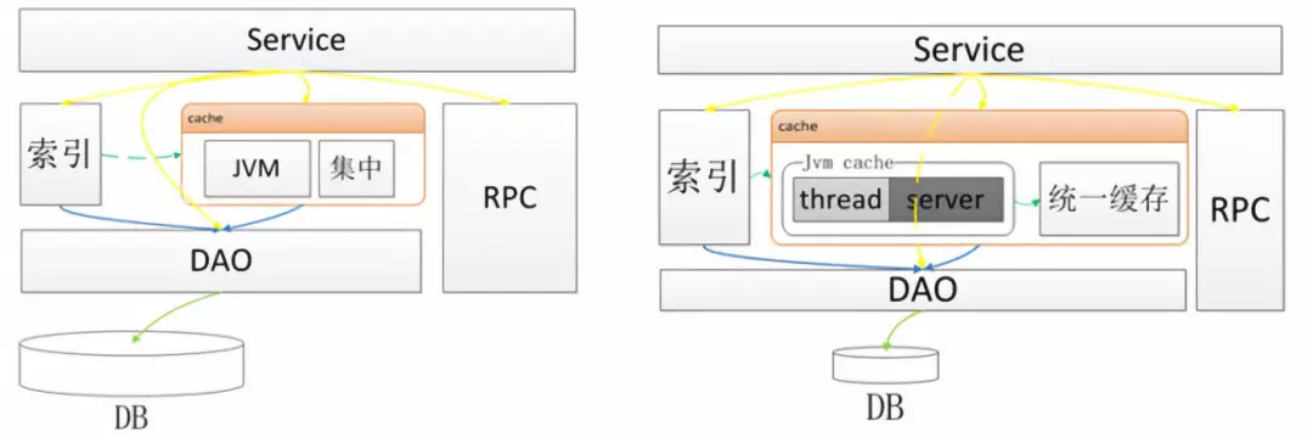


图6 增加本地缓存

有时所依赖系统性能不太稳定，避免出现因第三方系统影响系统，把依赖的服务进行数据闭环，与Dao一样当成系统的数据源；如商品系统强依赖商家系统的商家信息服务，若商家服务不稳定，商品系统一半服务都不稳定，采取对商家信息缓存一份，降低外部风险，把风险控制在自己手上；

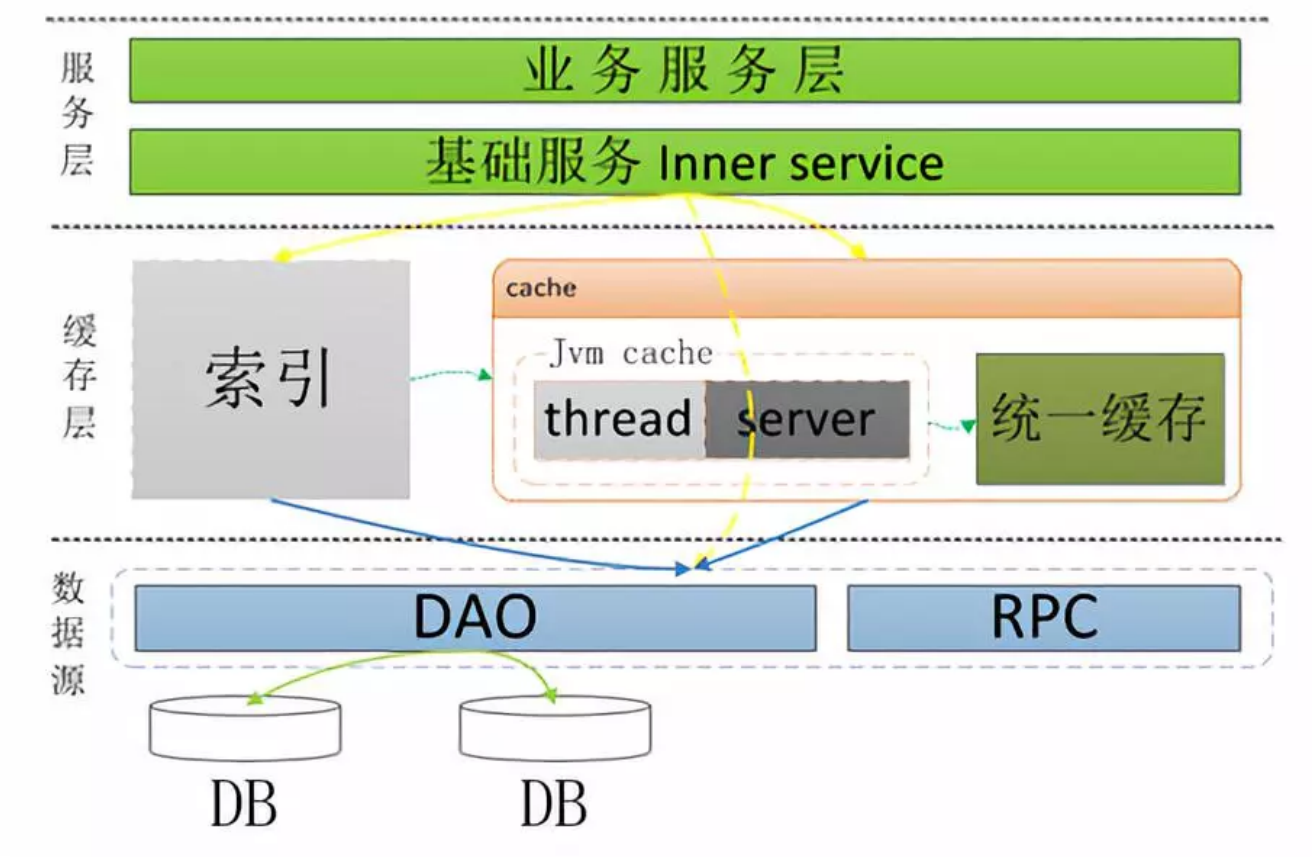


图7 远程服务进化成数据源

用户体验最近越来越重视，系统响应时间性能要求也越来越高，异步化是很好的一种选择：消息中间件；电商下单就是个很好的案例，在用户点击下单时，服务端不直接保存数据，给订单系统发送消息，就直接返回支付页面，在用户支付过程中，订单系统异步进行数据保存；

业务层、数据层的范围越来越宽泛，业务层可以分为基础服务与组合服务；数据层分为数据源与索引缓存；依赖的技术或中间件需要有效的结合，用于解决系统所遇到各种问题。

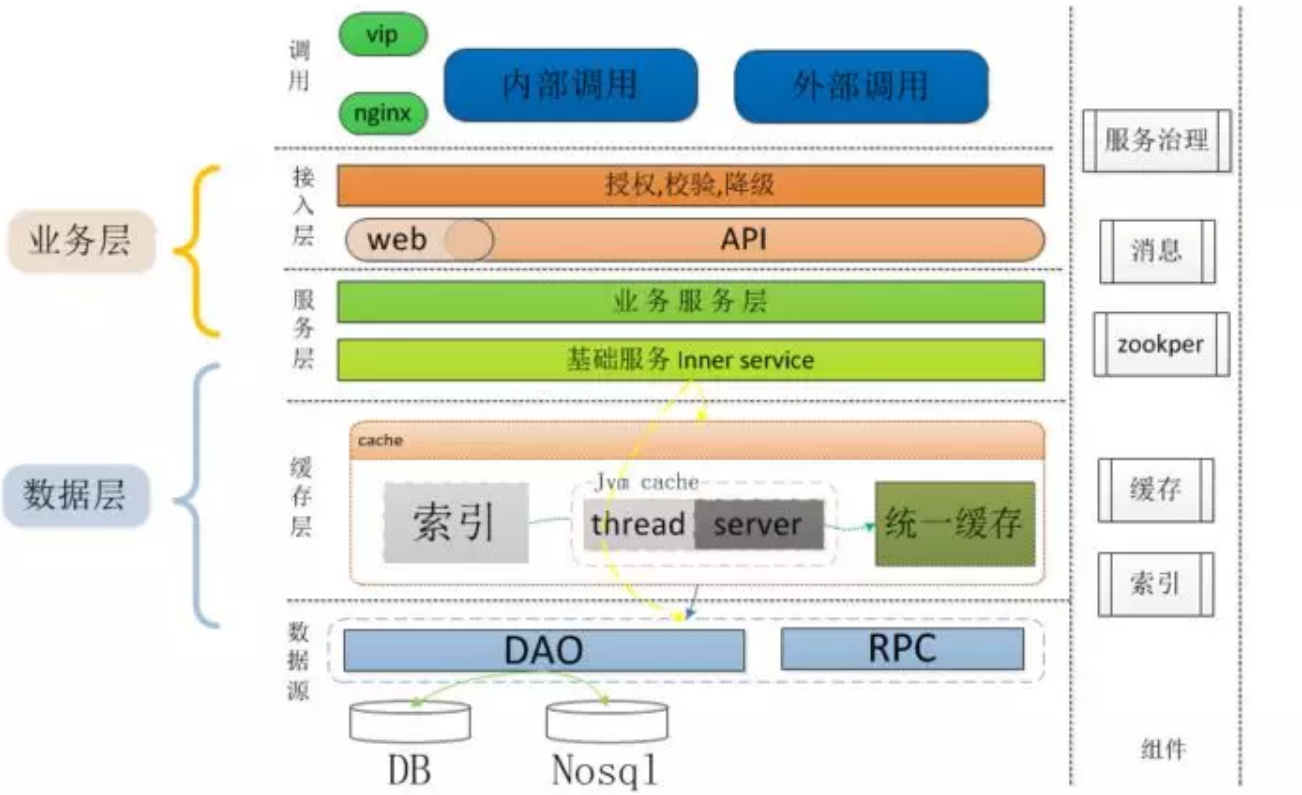


图8 复杂的结构

## 最后

系统结构慢慢变复杂，稳定性、健壮性逐渐提高；技术选择都需要结合业务痛点、技术储备以及资源情况，否则就有些不切实际，泛泛而谈；

以上是近几年自己经历的技术变革及升级的总结，后续可以针对个别点进行详细分享。

系统拆分的最后是微服务，结构的演变是技术的升级。



**京东技术** | 关注技术的公众号


**长按，识别二维码，加关注**