大促系统全流量压测及稳定性保证——京东交易架构分享（含PPT）

原创：杨超 高可用架构 2016-06-28

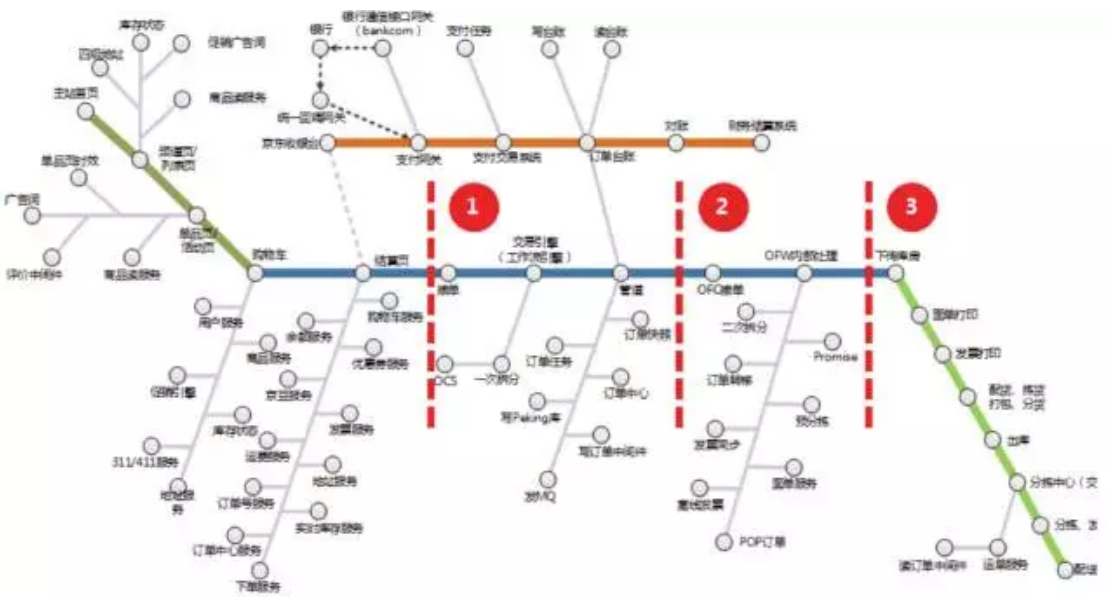
导读：对于应对突发的峰值访问，每个技术团队都有自己的经验及方法，但是这些方法远没有得到体系化的讨论。高可用架构在 6 月 25 日举办了『高压下的架构演进』专题活动，进行了闭门私董会研讨及对外开放的四个专题的演讲，期望能促进业界对应对峰值的方法及工具的讨论，本文是杨超介绍京东交易系统如何应对高压的实践。

杨超，京东商城架构师，2011 年 10 月加入京东。先后负责和参与京东的 IM 项目、交易系统 .NET 转 Java、购物车、库存、多中心交易等核心系统的研发和架构升级工作。

大家好！我是来自京东商城交易平台的杨超，今天特别高兴能够来给大家分享每年 618 及双十一所做过的工作，我是 2011 年加入京东，在这 5 年中我经历了不少技术演进，也看到了不少变化，在这里给大家做一个分享。

先介绍一下交易系统基本情况。

交易系统的三阶段



（点击图片全屏缩放）

这张图是整个京东商城数据流向结构。这个图主要分为三个部分。

1. **订单生成前**，包括单品页，购物车，架构，促销等功能，我们每个用户进来需要访问。它的特点是大促期间访问量非常大，后面会详细介绍如何应对。
2. **订单预处理**，订单生成之后，这是一个原始生成单，之后需要对订单进行预处理，进行拆包，包裹的拆分、大家电小家电拆开包裹运送等。因为是统一下单，这一块是订单预处理。挑战是访问量大，各个模块如拆单、订单转移，支付台帐等可能会承受非常大的压力，我们会采取扩容存储，限流，数据结构优化等方法去应对。
3. **订单履约阶段**，真正到后面是整个处理过程，配送黏合起来做的系统。这是京东商城的服务结构。

交易系统的三层结构

下面是交易结构图，从左到右，单品页是网站平台做的，今天来到现场的也有一些参与过的同事。移动端、微信、手Q，等入口都会调到购物车服务。

从购物车开始，我们从上到下是一个典型的分层结构，上面是调用来源，中间是我们的服务，下面是依赖的底层服务。其中强依赖服务是关键路径所需要调用的服务，是主流程中不可缺少的一部分。

（点击图片全屏缩放）

强依赖服务在大促期间不能被降级，我们需要提前扩容，以及进行代码重构、拆分、按来源单独部署等方法提前进行优化。

交易系统的访问特征

为了更好应对，我们需要对用户访问的特点进行分析。参看下面图，2011 年到 2015 年整个的单量，2011 年，618 是几十万的单量，去年单量一天是几千万。看着单量往上增

长的这张图，就能感受到系统压力有多大。

为了应对大促的压力，我们必须清楚知道用户访问系统功能的流量及分布。经过数据统计，接单前面这波系统，正常每一单，有一个前几年的大概的统计。购物车：结算页：产生订单页面访问的比例是 16 ： 4 ： 1，也就是说购物车访问 16 次，结算页访问 4 次，提交订单页面访问 1 次。到 618 及双十一，每天 PV 就是几十亿，几百亿、上千亿，因此我看到最大的量 1 分钟是几千万。我需要清楚知道这几千访问落到那几个页面。

根据京东传统，每年定一个目标，618 当天或者三天需要达到多少亿的指标，比如说一百亿或者几百亿，后面我们会把钱换算成我们的单量，我们客单价是多少？如果客单价是 300，目标要 100 亿，则我们单量需要达到百万或者三千万，这样通过预估出来当天的单量会有多少，这是提前的准备的整体规划过程。

为了每年的 618、双十一系统的稳定，京东研发如何应对？

应对大促的第一步
全链路全流量线上压测

系统底层的调用量是知道的，往年的 618 或者往年的双十一也可以找到。经过半年的业务跟进，我们系统会有很多的变更，数据变更或者是代码变更结构变更都会产生，我们知道这个系统能够承受多大量，上来对它进行压测。

压测分为线上压测、线下压测，**主力做线上压测**。

为什么我们会采用线上压测？早年我们只做线下压测，环境跟线上不一样，路由器和机器CPU，物理机，每一个不相同或者架设的路由超过 3 层，丢包，各种数据不一样，压测出来的数据经常会差异。

线上压测分开是怎么样做的？**需要将读业务跟写业务区分开**。读业务，我们正常可以看到读价格读库存、读购物车场景的分开，读跟写，看到购物车上的分布，就能知道是读还是写。

演练缩减服务器

从压测上，在集群中将服务器缩减，因为我们支撑的量，最高量达到 1 分钟达到 1 亿左右，平常最少有几十万、几百万的量。集群肯定是比较大的，最少也是几十台的机器，我们会把集群机器逐台往下缩减，真正看到线上量能扛到什么情况。

做到这儿，大家会有疑问？风险挺大。对，风险的确挺大，比如一个集群的 30 台机器一个一个往下缩，比如缩到 5 台，如果扛不住，所有的机器就崩溃，就会面临很大风险。所以梳理完每个架构之后，每年我们冒着风险，找到这个点，往上一点的量进行缩减，缩到一定程度再强行缩。

复制流量

主要通过 TCPCopy 复制端口流量，多层翻倍放大流量。如下图就直接将每层流量翻倍整体就是 1,000 倍，工具实现简单，可以实现多条线组合进行流量复制。通过这种方式发起超负荷的请求，检验服务能够承载的容量。

模拟流量

我们做了一个成立了一个压力小组，线上压力测试小组，我们做线上压测。用非常简单的底层工具去做压测。底层发起的量特别快而且特别多，集群，我们只做了压测平台，把这些工具集成起来做模拟流量压测。

在数据模拟上，我们是自己事先会准备一批数据。比如说几万个用户，几万个 SKU。商家各种库存模式，促销模式，在模拟平台我们会准备好。

流量泄洪

我们把订单在这个结构，接住堵在这个地方不往下放，往后拽都是密集的一些服务。从这一块把量堵住，堵几十万，突然有一天打开，看到一个峰值，看每一分钟处理量，往后能承受多大量，是不是能够承受发起的量，

实施方法

大家可能在朋友圈看到照片，各个服务的核心人员，集中在一个会议室，进行压测。一步一步往上加量，严密监控线上响应情况、订单量情况、各个服务器，以及各个缓存、数据库等机器的实际负载情况。出现任何风吹草动就停止发起压力，并进行记录和排查问题。

然后压测订单提交，往主集群写数据。跟购物车不同，这种压测会直接在生成集群上进行压测，并会写入数据。**因此需要将写入数据进行隔离操作，并将垃圾数据进行数据删除**，不能进入生产环境。

根据业务和技术维度筛选一批商品、一批用户，主要覆盖存储分布、用户每个等级以及业务分支。促销组帮忙建立能覆盖所有环节的促销数据。将这些用户的提交订单后清空购物车的功能禁用，保证能不停的重复下单。另外这些用户的订单提交流程中的邮件、短信提醒等相关功能禁用，产生的订单进行隔离，不往生产系统下发，并在测试完成后进行删除。

线上压测时，组织各个相关组核心人员严密监控各项数据。出现问题立即停止压测。先进行恢复，同时进行数据记录和问题排查，如分钟级无法恢复则直接切亦庄备用集群。

每个服务分别进行一轮压测，记录每个服务和购物车、订单提交压测得出的数据。根据线上实际用户调用比例进行换算，得出一个相对精准的整体集群承载数据。

订单生产后系统，主要用憋单，快速释放流量进行压测。形成对整个后续系统的，持续性高流量冲击，得出整体系统的处理订单能力。

下面是压测的 DPMP 系统结构图。

通过压测，就知道目前京东系统，压测完能承受多大量，面临我们目标差距有多少？压测完之后，我们就要运维优化。

在打压时候，我们按照交易系统的流量分布来模拟流量，比如正常访问购物车与结算页是 16 ： 4 的流量，下图的在打压时候我们也严格按照这个流量来执行，确保压力接近大促时候的真实访问场景。

应对大促的第二步
根据压力表现进行调优

调优一：多级缓存

缓存从前面比较多，CDN、Nginx、Java 都会有缓存。

缓存是逐级往下做，是一个漏斗状，最开始做缓存，到缓存的持续性在很短的时间内，一分钟或者一秒钟或者毫秒，这样给用户的感知是看不到缓存的，如果你要承载这么大量，必须逐级做缓存，前面做一些静态缓存掉，后面会做一些基础数据缓存，最后大数据，一层一层往上能挡住整个这一块，

调优二：善用异步

这是购物车大概的结构。这里有一个异步双写，我们会写丢这个数据，写丢没关系，我们购物车是整体的，加一个商品，写不过来，下次过来又会全覆盖。这样购物车就有一个多机房多活的可用性。

调优三：超热数据的缓存

购物车里面做热数据缓存，这种数据的缓存，比如促销服务直接影响到价格，缓存效率必须是在秒级毫秒级，在一秒钟怎么筛选十亿商品里面最热的商品？

我们利用 Queue 的原理，不断往里塞 SKU，队列的长只有 50。传进来之后，这里有的位置往前移，我们很快知道在一秒钟知道，排在前面肯定是访问次数最多的，每一个阶段应用存储访问最多的数据，如果是秒杀商品，500 万的请求有十万到二十万，它肯定大部分的请求在这块就出去了，不会穿透进来，这是我们自己做的热数据缓存。

调优四：数据压缩

对 Redis 存储的数据进行压缩，这样空间又缩小四分之一或是三分之一，我们数据到后面就会很小。当量小之后，访问效率就会升高，你数据量弹出很小，丢单率很小，可以提高我们的可用性。

异步和异构

什么是异步？购物车会调三个服务，如果是串行，一个 2 毫秒，累计起来是 6 毫秒。同时去处理，同时往后并行，这三个服务同时去调，根据业务规则要先调商品，再调促销，调两次，肯定在三毫秒之内。

异步数据落地，中间存储来解决访问流量过大，冲击原始存储的问题。包括库存状态库存数据的剥离，库存状态调用量太大，穿透直接到库存数据。

接单系统的异构

异步异构用得最多在这个系统里面。这一步接单系统的异构。接单系统在这一块的异构。我们整个接单，一次性提交订单、购物车是提交成一份数据，这样提高效率。

如果说按照原来的做法，直接写到表里面，有很多订单明细、促销明细、优惠券很多要写，这样访问效率会存在瓶颈。因此后面写到接单服务，再异步，调动某一个状态机，通过管道服务再衍生出来拆分成订单中心数据，支付台账，异构之后，单个系统的应对峰值的能力都得到了提升。

订单中心的异构

提交订单、接单做了异步处理。

再往下订单中心又会有很大异构，分成了 4 个子系统去分别调用。订单中心会产生列表服务数据，列表服务数据根据PIN的维度用户维度看到数据存储，第一步直接写，写不成功就是状态机，异步写到这一块存储。

订单中心有一个列表服务的存储，再有订单详情的存储，订单详情的存储是根据订单维度去存，这一块是根据 PIN 存的。

第三、四部分是单拎出来的状态服务及跟踪服务，后续生产跟它直接挂钩。通过异构之后，提高了订单中心应对峰值的能力。

商品页的异构

后面看一下看不到商品服务的一切异构。ERP 过来，是采销或者 POP 用户，进入我们的 MySQL，通过商品发布系统，接到商品消息在这一块之前是不能售卖，这一块直接发出去，这个消息会写自身的存储，我们会装 A 包、B 包、C包。

- A 包就是基础数据，比如商品名称；
- B 包扩充数据；
- C 包比如特殊标识，生鲜、易碎。

这样可以把数据分开，因为数据量太大了，十个亿的数据就有几百个 G，分开几个包。这样性能和可扩展性都得到了提升。

商品服务调用方，我只要调，订单系统查你，我知道是生鲜、易碎。但是前面需要基础数据，商品的名字，特殊属性，只有订单结算页需要，所以分这么多的包，我可以分开部署商品服务。为什么异构这些包。

发一个通知给别的系统，大部分系统依赖基础服务，发一个消息给他，商品变更了，它会缓存自己需要的数据，实时计算通过这个地方过来，异构出来存储，内部异构出来这么多存储，这是后台用户存储。

一个商品服务，真正做大了可以参考对一个胖的大系统进行拆分的异构方法，如果达不到这种规模之前也不建议过细去分。

应对大促的第三步
分流与限流

我们假设系统当超过一定流量后，超过的流量做直接拒绝处理，以便保护后端的服务，这就是限流。

Web 的限流根据 PIN 来限流，这是根据 IP 加 PIN 风控数据限流，这一块根据业务逻辑，一个单一天能下多少单，根据这个逻辑去限流。渠道可以按 App、PC、微信等分开，分流和限流这么做。

下面讲秒杀系统是怎么来的。秒杀系统是限流和分流的典型。

秒杀，假设预约是 1500 万，在那一分钟之内，这么多用户过来抢手机，也就是单个商品，就把流量直接导到秒杀系统。

秒杀系统从 Ngnix 进来就有各种的限制，到我们会识别用户供应商或者商贩去刷的数据，这块调用是从正常访问的单品页分出来，不影响主流程。

通过 IP、PIN、每一步怎么来、用户以提交记录，一秒钟提交多少次，一分钟提交多少次等一堆的规则做判断来限流。到最后再验证有没有预约、常用地址服务等，都通过后再调到接单系统。

整个秒杀系统就是一个典型的沙漏的系统，当流量跑到后面，实际上只剩很小的一部分，只有真实的写流量到接单。

接单提交服务单独出来两台机器给它用，后面的存储得到保护，两台机器最多也就几十万，也能承载住，这就是分流跟限流。

促销与价格

促销里面也有一个限购，比如前 30 个用户享受促销，发一个码出去，需要对这个码进行处理，这是一种限流。

促销分流中需要把价格服务单拎出来，分出去，单品页搜索，手机微信，购物车的架构从这里出来，最实时的价格。这样产生分流，这一块有一个存储分流，还有更多其它的就没有一一列举，这只是一个示意图。

这就是我们整个的分流跟限流。根据前面的渠道，调用量、做多少程度，相对于影响力，做分流和限流。

应对大促的第四步
容灾降级

如果分流、限流还没抗住，系统进一步出现压力问题，再要做准备做容灾降级。

容灾降级有机房容灾，我们做多中心机房，网络容灾、内网外网的容灾，应用的容灾，分组、托底容器，最后保证基础的服务是正常的。

网络及 IDC 降级

这是容灾降级，这是网络大概示意图。我们的 ISP 进入机房，核心交换机、柜顶交换机、这是交换级的容灾，网络共享容灾。

业务降级

购物车结算页的降级，当订单出现过大，延保服务、预约服务如果不行，直接保主流层，就属于业务层面的降级。

（点击图片全屏缩放）

整个 618、双十一准备下来了，后面准备降级方案、容灾方案列出很多，每个业务需要根据自己的情况去考虑，上面只是简单列举了几个供大家参考。

应对大促的最后一步
完善监控

最后到临近双十一、618，需要网络监控、机器监控、以及了解订单量、登录量、注册量等应用级的监控。

这是 IDC 机房的监控，运维部做的。这是单个机器，物理机、Docker 的监控。以及对交换机、IP 进行监控，这是网络监控。

下面是方法监控，每个方法监控到 TP99，99 是最近几次的峰值是达到多少毫秒？成功率、失误率、调用次数等。

订单量的监控，一旦出问题都会报警，这是大概的监控系统，这是依赖的一些监控系统，我们还会衍生出来自己应用的监控。比如说库存，预算是最重要，提交订单，保证提交订单成功是最重要，从那一堆，库存自己的一套写一个页面，写一套监控系统出来，优惠券、购物车写一个小监控系统去监控。**监控是到大促的眼睛。**

一次大促，总结下来就这么多，谢谢大家！

Q&A

Q：您提到做线上的压测，会产生很多脏数据，这个数据是最终怎么处理。还有一个，您做了好多异构的数据，这个数据怎么保持它的一致性。

杨超：先回答你脏数据，所谓的脏数据，写数据是为了隔离出来，有的是打标，有的是另外起表，在数据库里面把这些数据隔阂开来打标，一边写一边删，压测，30分钟到一个小时，我们是在凌晨做这些事情，压测完写的数据是非常危险，写的数据会爆，引起瘫痪。我们实时监控它，打标清理，首先切到小集群承载。真实的集群，真正的量不会达到618那个量，我们的峰值可能会很高，正常到晚上凌晨那个量很小，我们就把它切到另外一个小集群承载，后面再把它转回来。

异构出来的数据怎么保证它完整性一致性？异构都是小维度，不成功会补全，补全不成功会穿透，写 Ngnix，会一层一层穿透到 MySQL 数据库。

Q：一个库存性能与一致性的问题，客户下单肯定会判断一下有没有库存，并且他下单的时候还要实时扣减库存，这方面怎么解决性能问题？如果把库存的量放在 MySQL，一拿会慢也会影响你的下单，并且渠道会比较多。

杨超：库存大部分用前端 Redis 防重。用业务维度做防重，第一次查出来的业务属性、商品属性、库存数量数据，我们的一些业务数据、一致性查出来，进行一次校验，校验成功就通过，校验不成功就告诉它库存不足。目前看到这个单量一天几千万单，几乎没有超卖的部分。

Q：SKU 库存量是放在 Redis，通过 Redis 实时判断?

杨超：对，基本上卡的第一道都在前面，查的第一项，如果查到数据就会写到 Redis，最后 MySQL 是落地存储，内部存储。MySQL 写入量最大达到 60、70 万。

Q：还有一个问题关于线上压测，还有线下压测。线下压测指的是你们把流量镜像到相同环境里面实时做压测还是走的同一个环境?

杨超：线下一个测试机房，拿一堆机器按照你线上结构部署，导一点数据下来进行压测。线上是我们真实布局的线上的环境，配置差不多一样。

Q：数据库对应的客户能否看到你的压测数据?

杨超：压测的订单，从接单到后面就会把它屏蔽，直接截住，相关数据要打一个标，正常数据打一个标识，不能往下传，不能让客户看到。

相关阅读

- 亿级商品详情页架构演进技术解密
- 双十一大型电商统一服务架构实战

本文相关 PPT 链接如下，也可点击阅读原文直接下载

http://pan.baidu.com/s/1eRMW4zK

*想更多了解本期高压下的技术演进沙龙内容，请关注「ArchNotes」微信公众号以阅读后续文章。**申请北京城市圈可以更及时了解后续活动信息。**转载请注明来自高可用架构及包含以下二维码。*

高可用架构
改变互联网的构建方式

长按二维码 关注「高可用架构」公众号

*** 回复『城市圈』进一步了解**

阅读原文