

《京东技术解密》——海量订单处理

发表于 2015-01-15 17:34 | 10299次阅读 | 来源 程序员 | 13 条评论 | 作者 京东研发体系彭青

图书

京东

技术


技术架构

电子商务

订单处理

OFC

业务流程

 **摘要：**《京东技术解密》从618大促销、产品演进、技术演进、创新激发、牛人专家五个侧面详细描述了京东研发团队的发展。本文摘自《京东技术解密》第4章节：完美风暴——海量订单处理。

OFC的重要性

2014年的618显得和以往任何店庆促销日都不同，不仅仅是因为电子商务本身在中国不断飞速发展对京东系统带来的挑战，更为重要的是2014年5月22日刚走入美国纳斯达克殿堂的京东聚集了最耀眼的光芒，能不能保持这样的光芒，618则会是一份很有说服力的答卷，当然我们最终给出了满意的结果。作为一个普通的购物者，当我们在浏览器中输入www.jd.com并回车，便可以看到京东商城的首页，根据自己的需要选择喜欢的商品，然后加入购物车，提交订单后，即可享受京东的极速物流体验，最终完成一次简单快乐的购物历程。其实，订单提交后，需要经历多个环节和各个系统的处理才能完成使命。其中有一个环节是订单必须经过的，而且这个环节连接了用户下单和订单在库房的的生产，就是订单履约中心（OFC，Order Fulfillment Center），本章我们就为各位解密这个部门。

2014年的618后，京东技术团队分享了如何应对店庆日及以往促销活动在技术方面的经验和探索。其中有一讲“电商海量订单处理OFC系统的关键技术环节”（见[京东技术开放日第一期](#)），说的就是这个部门做的事情。

这个部门的职责，用彭青的话讲就是，转换用户订单为各终端系统的生产单，并且按要求送达到相应终端系统。举个例子，就好比我们将采集到的原始食材按照客户的不同口味（不同系统）进行烹制，并且在指定的时间内做好后送到客人（终端系统）那里，整个过程包括订单的拆分转移和订单的下传。其实我们从网站下的订单（也叫原始单）在库房是直接生产不了的，需要经过OFC这个环节的处理后，才能到达各个生产系统。由此可见，这个环节必然会有大量数据需要处理，而且还要保证时间。

想必大家知道关于京东的211、311等配送方式，如果用户选择不同的配送时间，京东的快递就必须在用户指定的时间送达，而如果下游的生产系统收到订单数据比较晚，就会影响后续的配送时间，最终会影响客户体验。现在订单下传，对接的全国库房近150个，需要调用的外部处理订单服务也有近20个，而每个系统的处理能力和响应能力又各不同，这就需要我们进行相应的调节流量的配置，这其中只要有一个系统存在问题，就可能影响订单的下传。

OFC的形成

2003年京东网站创立之后，迎来全国电商的快速发展，京东的业务随之不断增加，导致相应的业务系统也在持续增加。直到2011年，随着系统增多及业务的需要，逐渐成长出来一个小团队，负责在各个系统之间进行数据的传输，将客户订单数据传到库房，同时需要将非客户订单，如采购单、供应商、内配单等二十多个业务传输到相应的业务系统，至此OFC成形。

初期由于各个系统的不完善，导致数据传输总是存在各种各样的问题，订单总会卡在这一环节，而作为传输环节又必须保证数据能正确传输完毕，这就需要我们必须了解上下游系统的业务及数据处理的过程，只有这样才能知道问题产生的原因，才能知道该如何去处理问题。但是由于上下游的系统是需要经常上线新需求的，我们每天需要及时了解上下游系统业务的处理，才能保证我们的环节没有问题。

那段时间兄弟们真的很辛苦，每天需要处理上千个工单，加班更是常事，以至于一个同事说睡梦中都在处理问题单。由于业务领域划分存在问题，系统边界不明确，导致出现许多莫名其妙的问题，兄弟们干得很苦很累，直到彭青来了之后。彭青带着一个厚厚的黑色全框眼镜，个子不算太高却有个小肚腩，学生族的发型让人感觉很亲切。彭青根据他多年的工作经验和对系统的理解，对这块业务进行重新划分，逐渐将非客户工单的数据传输工作交接给对应系统进行处理，将兄弟们解放出来，开始客户订单处理的新征程。

到2011年底的时候，在彭青的带领下我们已经成为了二十多人的团队。为了更进一步拓展在客户订单方面的业务领域，我们主动接手了订单拆分系统和订单转移系统。这两套元老级系统是用.Net编写的，由于前辈们大都不在职了，文档也不完善，对于系统内部业务了解的人很少，修改非常难。而此时每天由于系统问题导致的事件单多达上百，也就是说每天运维带来的工作量都是可观的，在这样的情况下，接手这两个系统自然全无反力。

技术的改造

.Net到Java

系统接过来了，第一步要做的事情当然是重写。对技术的选取，根据当时公司技术发展战略以及Java的普及，我们选用了Java。重写过程中需要梳理已有的业务，自然少不了不断和原来系统的人员进行交流，去确认业务流程和技术处理细节。经过一个多月，系统的重写总算完成，接下来的工作就是上线了。

开始小流量地切，我们通过开关进行控制，通过省市县区域分流，到2012年2月系统算是上线了，而之前的.Net系统也逐渐退休了。到这时候，OFC逐渐根据业务划分为3块，第一块是订单拆分，第二块是订单的转移，第三块就是我们前面提到的订单下传和回传。

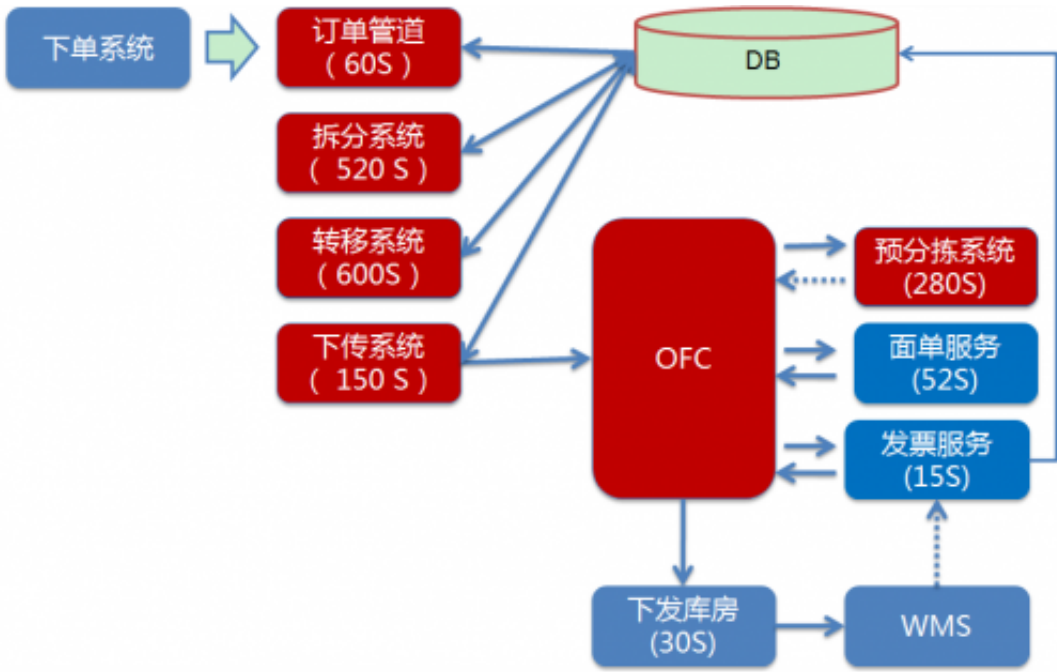
这里要给大家解释一下：

1. 订单拆分，可能大家想到了，就是将用户下的订单，根据我们库房的不同分布，比如大家电仓、百货仓等进行拆分。当然这是拆分最初的也是最基本的理解，到后面会给大家讲讲现在的拆分是什么样子。
2. 订单的转移则是根据拆分完的订单以及库存等属性，将订单转移到下游的系统。
3. 订单下传和回传主要是指，转移之后的订单调用库房相关的预分解、打包等服务环节，进行订单下传和生产，当订单顺达客户手里，还需要将订单的相关数据进行回传。当然这里只是粗略介绍，后面我们会更详细地给大家呈现。

211订单履约率提升项目

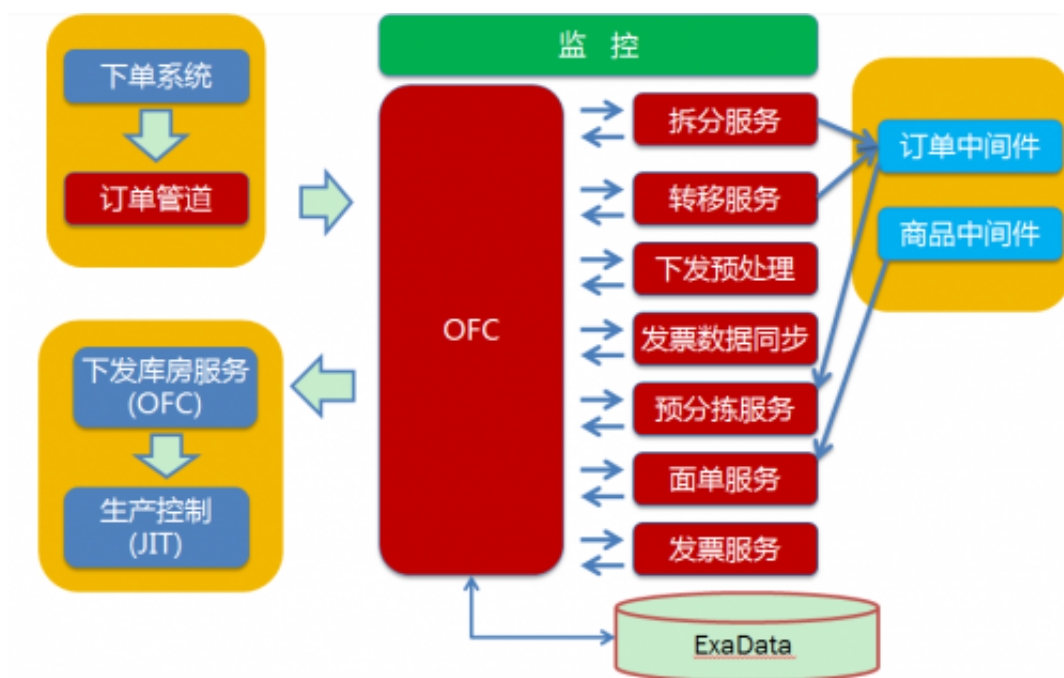
重写完之后，系统总算可以正常运转了，而接下来的事情就是对系统的进一步梳理和优化，以更好地支持将来业务需求的发展以及技术方面的扩展。当然有的时候系统的改进往往是由于外部业务的无法适应导致的，这也符合变革的本质。用户体验至上一直是每一位京东人追求的目标，就连我们的老刘也会隔三差五在网上下单来

体验一下，而这次老刘发现了一些问题。当他下单后，等了半天才收到订单，这让老刘无法忍受。经过调查后发现，从下单到库房竟然花了两个多小时。



改造前的系统整体设计图

于是老刘立即成立了一个叫作“211订单履约率提升”的项目，该项目涉及11个系统的升级改造，包括订单交易系统、订单管道系统、拆分系统、转移系统、订单任务系统、OFC相关系统、预分拣系统、面单系统、增值税资质服务、发票系统、WMS系统。其中4个系统需要全面改造，3个系统需要大量改造，剩下的4个需要少量改造，而且由于与订单相关的业务点多且逻辑复杂，无法在测试环境下全面测试。这不仅影响着整个订单的正常生产，甚至会影响财务相关业务。项目任务非常重要，要求两个月内保证订单从下单到库房的时间缩短到5分钟内。大家马上开始了工作——需求讨论6天，设计方案5天，开发15天，功能测试20天，性能测试44天，上线部署调试26天，总计工时达5066小时，最终实现了项目目标。与此同时订单下传各环节的服务性能指标也得到了规范，使得订单下传趋于稳定，理顺了订单下传流程。技术方面也得到了锻炼，使用了Zookeeper分布式配置、CXF Timeout设置、Log4j多Tomcat示例配置、Oracle数据库分区转历史方案，数据库使用了OracleExadata、MySQL。在这过程中和Oracle技术团队直接沟通多达10次，在数据库设计方面、性能调优、转历史数据方面都得到了提升。更为重要的是锻炼了团队，对于战胜艰巨任务有了更大的信心。下面是系统的整体设计图。



改造后的系统整体设计图

对于拆分，在几位大牛对系统的业务进行梳理后，发现部分业务有些混乱，业务领域划分得不是很清楚，拆分系统中除了需要根据商品的不同属性进行拆分外，还需要对订单中使用的金额、优惠、运费等信息进行分摊处理。这几位大牛敏锐地发现系统这样设计是有问题的，于是就把金额信息处理逻辑拿出来，专门做成一个服务——OCS订单金额计算服务（OCS），拆分只需要对其调用就可以。同时，我们对OCS分摊结果的数据进行了持久化数据存储。系统这样设计，不仅解耦了拆分服务之前混乱的业务处理逻辑，OCS的数据也一举填补了公司这方面数据的空白，成为其他系统使用和处理业务逻辑的数据基础来源。到现在为止，直接使用OCS数据的系统就有二十多个，其重要性不言而喻。

SOP合页单项目

2013年，公司级项目SOP合页单要启动，即用户购物车里既有京东自营的商品同时有POP商家的商品（SOP）。在结算的时候只需要提交一次（之前只能分开提交，类似淘宝多商家的商品只能单独提交）。为了改善用户体验，同时需要将提交之后拆分完的子单结果显示出来，需要我们团队提供一个拆分服务供交易组使用，这是一个重大的考验。下单环节的速度非常快，TP99一般都是几十毫秒，而我们目前的服务则是几十秒，完全不在一个数量级。为了保证项目能够顺利完成，我们既需要满足日常的业务需求，同时要新切出一个分支进行修改，用于此次项目，同时需要将针对新需求的代码及时同步到这两个分支上，任务非常艰巨。解决了开发问题后，就要想着如何在性能上有所提升，比如，可以放在内存里处理的就放在内存中操作；尽量减少对外部服务的依赖；对于非同步化的操作进行异步化；对于部分服务我们甚至采用降级的方式，在必要时通过开关进行降级，保证整个服务的整体性能。如此这般后，我们主动要求性能测试组对我们的服务进行性能测试，在代码级别进行了优化，最后在指定的时间内成功地完成项目。而此时我们在维护着同样级别的3份拆分服务代码，老的下单对应的我们前面说的老拆分，新的下单对应的我们新的拆分，还有我们为交易系统提供的预拆分。

而在此时最困扰我们的不是维护这些系统，而是经常会由于网络不好，使一个订单的服务超时，进而导致服务进行重试，而事实上订单已经提交成功。这就可能使我们错误地提交两次甚至是多次订单，比如客户下一个原

始单，需要拆分成两单，但是由于上述原因可能会得到多单；如果用户选择货到付款，会给用户造成困扰，会带来配送的成本，如果是在线支付的话则会导致公司的损失。刚开始的时候没有解决方案，只能通过监控去发现，发现后人为锁定这些订单，而这样不但增加了运维压力，而且人工处理难免会有失误。由于我们在提交子单之前会获取订单号，每一次获取的订单号都是新的，这会导致调用这个服务时对订单号是无法防重的。后来海波想到一个防重方案，就是我们在调用这个服务之前将订单号信息输入自己的防重库，新订单来的时候先在防重库中进行查储，如果有订单信息则说明之前提交过，本次提交失败，然后直接把库里相同订单号的数据拿出来提交即可，这样还可以节省订单号。如果库里没有查到，我们将该订单号插入库中，同时调用服务。问题得到有效解决。本次提交经过这一系列的优化，系统总算还是比较稳定了。

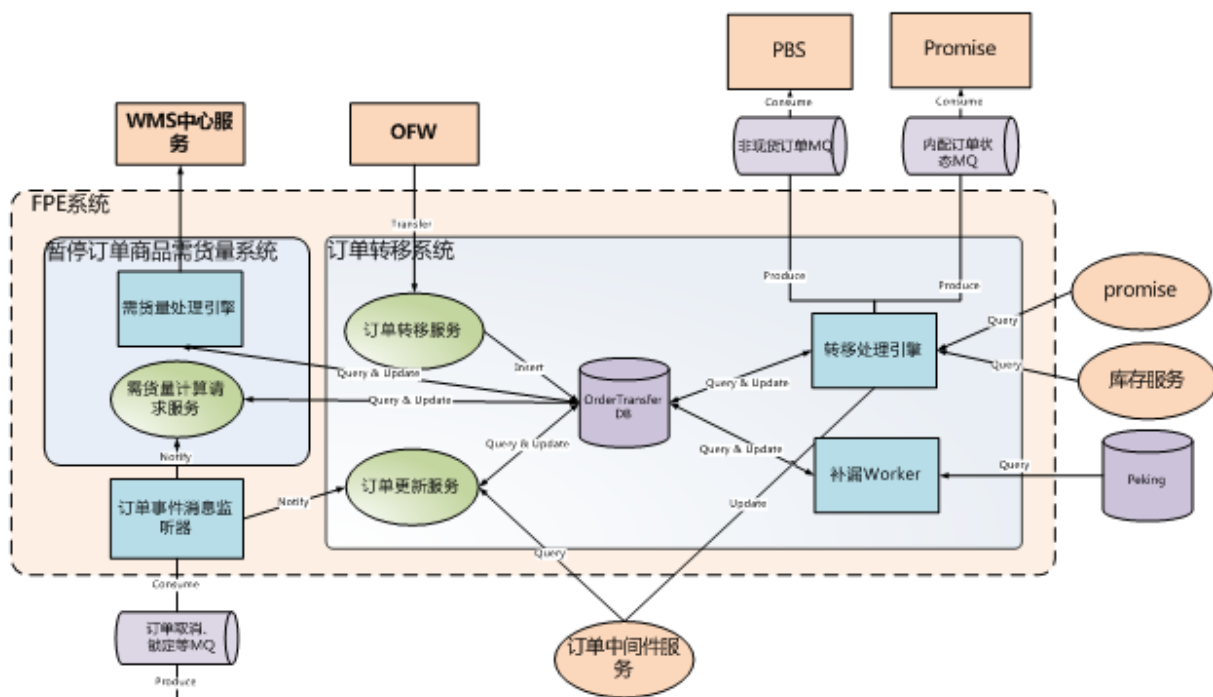
转移架构升级

转移系统也进行了大规模的调整，为了进一步保证订单及时准确地转移到下游的库房系统，转移团队在业务和技术架构上进行了一系列的改进：业务和数据处理异步化，即将可以异步化处理的业务和数据放入分布式队列，由对应的模块处理；使主流程业务简单快速流转；数据处理并行化，将数据切割成多个业务单元，并行处理业务单元；针对变化少、实时性要求不严格的热点数据，使用缓存并配以更新机制，以提高性能；对于业务洪峰，通过平滑控制保护后续系统不被洪峰压垮。

在业务流程方面也进行了优化。由于涉及订单生产流程，需求变化速度非常快，需要不断梳理现有流程，去除不必要的流程，减少有新需求时对不必要业务流程和分支的考虑。同时，需要对现有分散业务不断地抽象和改造，以方便业务扩展。

面对这么多的优化和改进，每次上线的风险无疑是巨大的，如何规避风险呢？那就是要分流、可配置化，以及运营工具先行。由于新上线的项目风险较高，特别是容易忽视一些对外交互的小功能，而发生线上问题时又无法及时切换。因此，需要对业务上线进行分流，并且通过灵活便捷的配置中心随时进行控制。对于异常情况一定要优先考虑，并且开发相应运营工具，以备紧急情况使用。尤其不能抱有侥幸心理，认为小概率事件不会发生在自己身上。

转移团队的负责人铁总（大家总是这样称呼他）已经从事电商十余年了。这个来自湘西的汉子对待工作总是严肃认真，但面对生活却又充满热情；结婚前总会泡在游戏中，或者痴迷羽毛球，女儿出生后便成为了他的一切。在谈到转移未来的规划和发展时，他充满自信地说：“将来会在保证客户体验的同时，更多地通过在本上和流程上优化来降低成本。库存分配将在保证订单履约的前提下，打破现在先下单先占库存的规则，提高商品库存周转率和现货率，同时给客户提供更早的收货时间选择”。



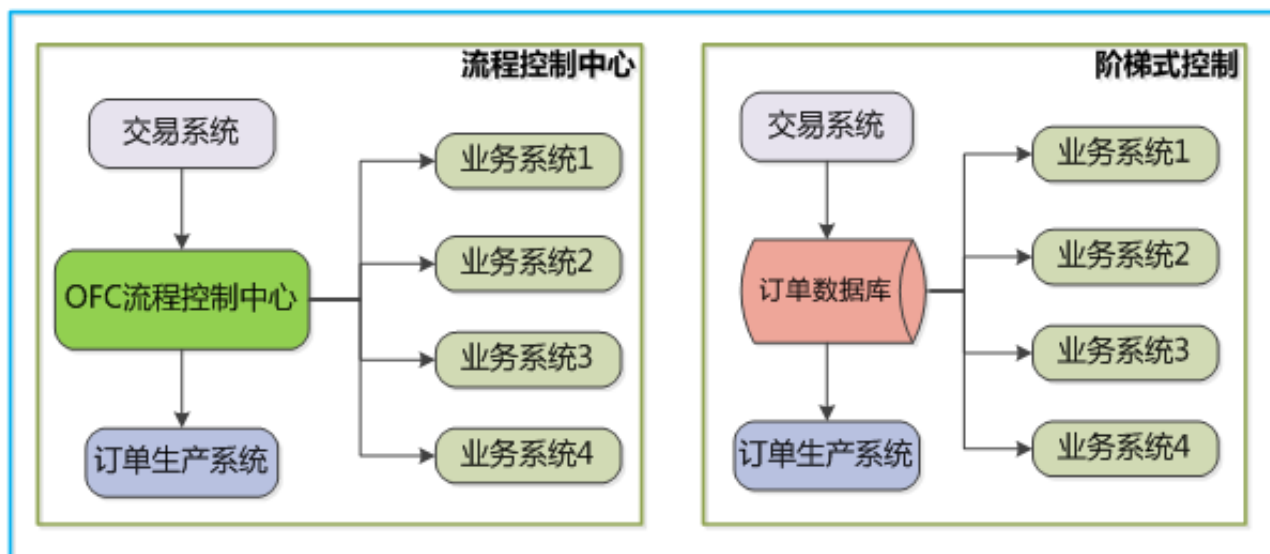
转移系统整体流程图

不得不爱的运维

刚开始负责客户订单系统时，每天要处理上千条Ticket（订单事件），而现在只需处理几十条。这种锐减，不仅说明了系统日渐健康、业务逐渐规范，更证明了我们的运维流程和运维制度正日趋成熟。这些成果都离不开善于分析总结的文杰，他是一位来自山东的80后，在团队中主要负责运营流程优化和与协调相关的工作，团队在运维方面的问题目前还没有他解决不了的。OFC是连接用户和全国终端库房的重要的通道枢纽，这其中的任何系统出了问题，都会导致订单无法正确实时地到达终端库房，后果都是不堪设想的。因此，每新增一个库房都需要团队进行库房的终端系统部署和调试，直至生产系统测试完成为止，我们称此为开仓过程。随着公司不断发展壮大，订单业务不断完善，全国现存仓库已超过150个，这都是文杰和团队无数日夜的付出换来的。支持审计也是不可忽略的一部分工作，每季度我们都会给同事讲解新业务，给他们解释差异订单的原因。同时，我们还负责新业务的学习推广，让团队的新成员能够快速了解业务知识、熟悉业务系统。伴随着业务和系统的日渐完善，我们也在不断地尝试和推广系统的智能化运维与支持，相信在不久的将来我们一定会实现无人化系统运维。

从618到双11

从2012年开始，店庆促销活动力度在逐次增加，订单量则成倍增长。伴随着订单量的增加，系统面临的挑战与日俱增——订单业务越来越繁杂，业务处理流程也越来越多，很容易出现数据不一致问题。因此，在处理海量订单时保障数据一致性非常关键。系统整体控制上要采用流程控制中心，而不是阶梯式控制。之前由于直接依赖数据库，数据库最终会成为影响订单处理的瓶颈，数据的一致性很难得到保证，而采用流程控制中心模式则可以大大减少数据不一致发生的几率，同时可以借助工作流和状态机实现中心控制，这样既便于运营，又方便及时发现和解决问题单据。



流程控制中心和阶梯式控制

支持海量订单处理

无论系统如何优化，单个系统总有瓶颈，要支持不断增长的订单处理量，关键在于提高系统的扩展能力。首先，核心系统的每一层都要有扩展能力，可以以实例为单位进行扩展，也可以以集群为单位进行扩展。其次，系统整体要有扩展能力，可以根据实际业务特点，从业务上进行垂直拆分以实现扩展，也可以通过分布式部署来方便地增加一个具备整体功能的集群，从而快速增加处理能力。这相比仅做备份系统而言，节约了成本。

所有核心的OFC订单处理系统已实现了水平扩展能力，部分系统实现了分布式部署改造。在2014年618大促前，正是由于系统具备这种扩展能力，才能够在非常短的时间内扩展了处理能力，保障了大促的顺利开展。我们的最终目标是，所有核心系统都要完成分布式部署。

解决数据一致性问题

早期的订单处理流程分散到多个应用系统中，数据来源不统一，也缺乏统一的状态机控制，经常出现数据不一致问题。但同时，也不可能由一个系统来管理所有的流程，因为维护和管理工作会非常庞杂。解决办法是，梳理出订单处理的主流程和状态机，然后由主流程系统负责整体流程的调度和数据的推送。这个主流程可能跨大的业务域，如物流领域和资金领域，每个领域内可以有工作流，但不能与主流程冲突。识别出主流程系统还有其他的优点：一是可只重点建设主流程相关系统，使其成为稳定的系统集群，而非主流系统则可以投入较少的成本，从而既有利于保障业务顺利开展，又能降低整体建设成本；二是主流程系统可以有效地保障生产计划的执行；三是主流程系统可调节系统流量，有效地平滑业务高峰，保护主流程相关各主要系统的稳定运行。

支撑运营工作

对运营工作的支持，包括抢险、预防，以及“治理+预防的升级”。在早期阶段，系统架构主要是支撑业务功能的实现，没有为运营而设计，线上系统会因为各种意外而影响业务，让系统团队疲于应付。后来，确立了为可运营而设计的理念和原则，设计时必须考虑可监控、可运营，同时把可用性、稳定性、健壮性等列为设计的重点，在实践过程中确立了自己的方法论。

第一，对系统进行梳理，识别出核心系统，把核心系统建设成为可用性高、可靠性高的系统，保障这些系统少出问题，出问题时系统要能自动恢复。

第二，保证系统出现问题后能快速发现问题，甚至在问题发生前发出报警。为此需要有对数据积压量趋势的监控，以及在有积压情况下吞吐能力的监控。这些监控需要及时，我们针对分布式系统，开发了分布式监控系统，能够迅速地反应每个部署的每一个实例的情况，又能收集整体的运行情况。

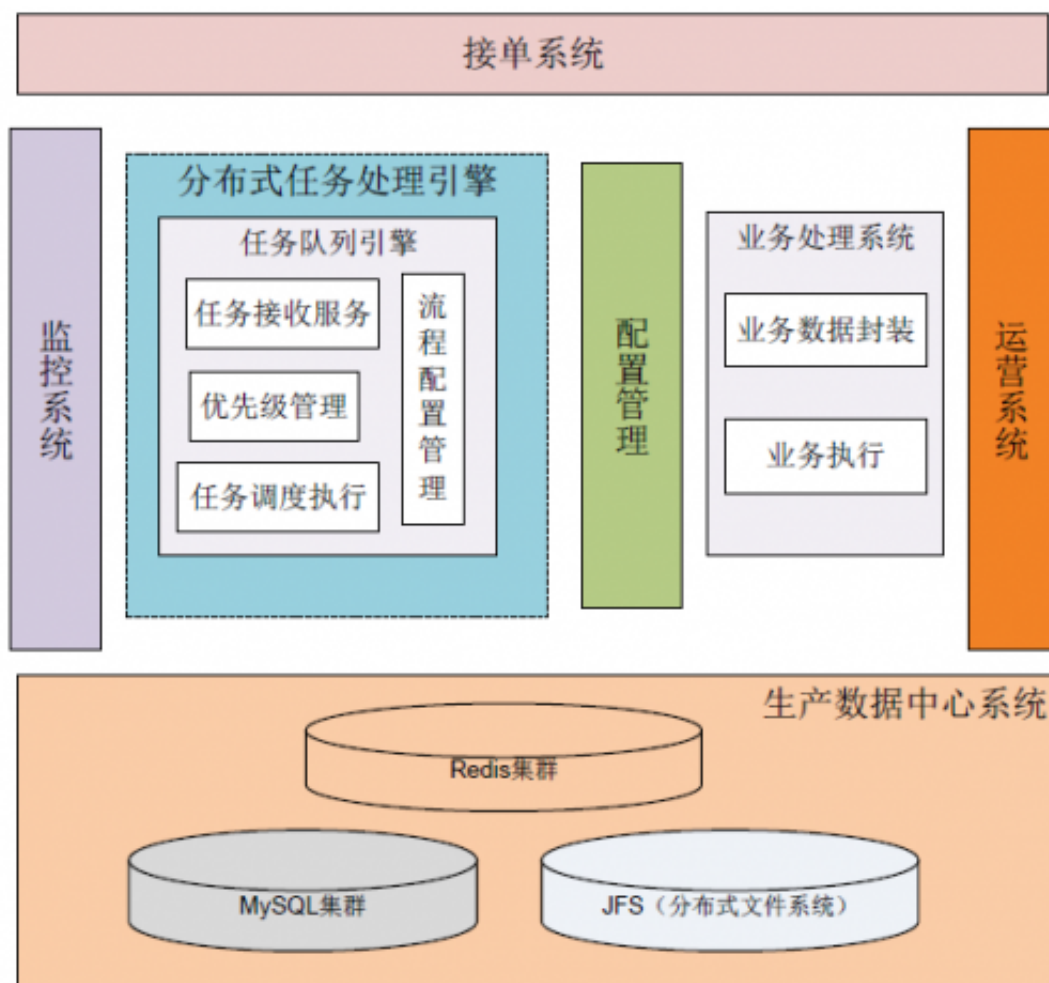
第三，保证发现问题后可以快速定位和处理。为此我们设计了集系统处理能力、数据积压情况、数据处理情况、日志、系统负载于一体的统合分析工具。

第四，一个系统出现问题，往往会将影响传递到其他系统，系统治理势在必行，目前我们已有SOA治理平台（正在优化过程中），目标是能够理清各系统的血缘关系，完善SLA体系；在出现问题后可以及时评估出受影响的系统，快速做出应急响应。

海量数据的开始

总原则

订单处理系统与交易系统本身是存在区别的。交易系统直接面对客户，所以系统可用性要求和性能要求非常高，特别是在高并发情况下的系统表现，所以交易系统在架构上的重点在于解决这两个方面的问题。而订单处理则不同，系统短时间不可用，响应出现延迟不会对客户造成直接影响，也就是说我们关心的是平均值而不是某时刻的峰值。订单处理系统架构设计的关键在于如何处理海量数据，以及数据一致性的保障。近年来，京东的业务领域不断拓展，订单量飞速增加，所以必须保障系统吞吐能力得到提升。与此同时由于涉及的系统众多，各系统业务处理方式和流程不同，导致各系统性能指标差异较大，所以要定义好各个系统的SLA指标。由于订单的业务越来越复杂，那么系统的业务流程也会越来越多，这就需要我们划分好主次业务流程以及优先级，同时需要设计灵活多样的降级方案，保证主业务正常运营。



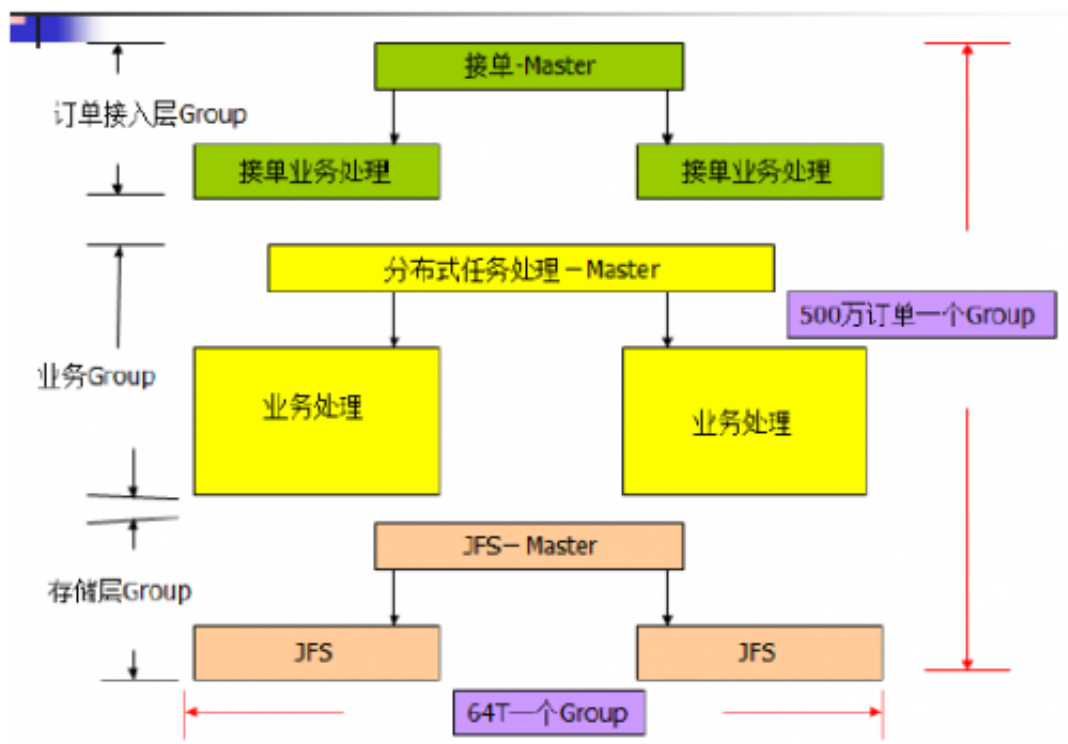
OFC整体架构图

系统保护

涉及OFC调用的订单系统很多，而各个系统处理的能力均不相同，不是所有的系统都要承担高峰值处理的压力。这就需要有针对性地控制、调用这些系统，具备削峰和流量控制的功能，以间接保护上下游系统，防止调用方的系统雪崩式挂掉。还有就是监控，要有统一的产能监控；要防止过载，在过载之前要能进行控制，要保证自身系统的安全稳定，还可以采用快速拒绝的机制。

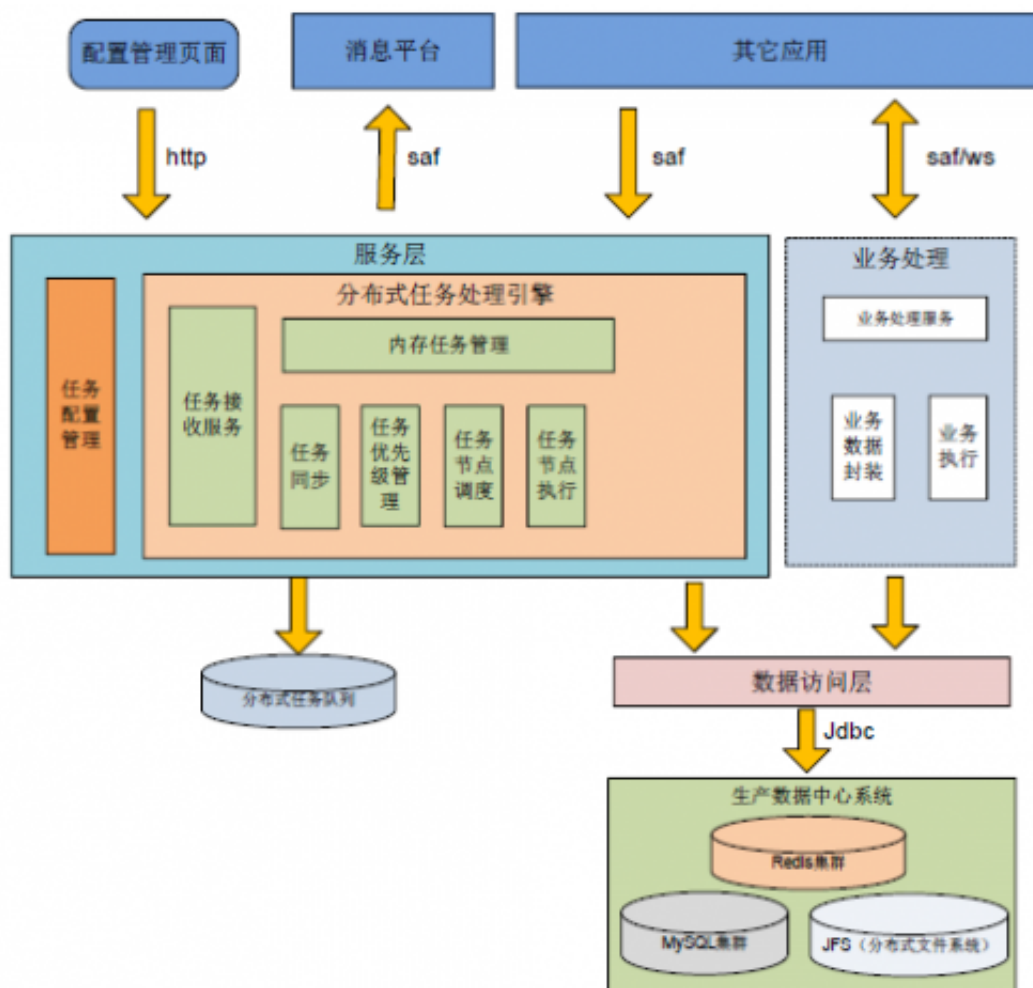
分布式系统

主要是在扩展方面进行设计，保证系统每个切片可以水平扩展，也可以以集群为单位进行扩展，实现分布式任务队列。我们每一个Group能处理的订单量在可控制的范围内，一旦某一处出现瓶颈，可以随时部署一个或一套Group。下图中不同Group可以彼此独立部署，也可以整体部署，当某一处出现问题时可以单独进行部署，或者整体流量大时也可以复制部署。



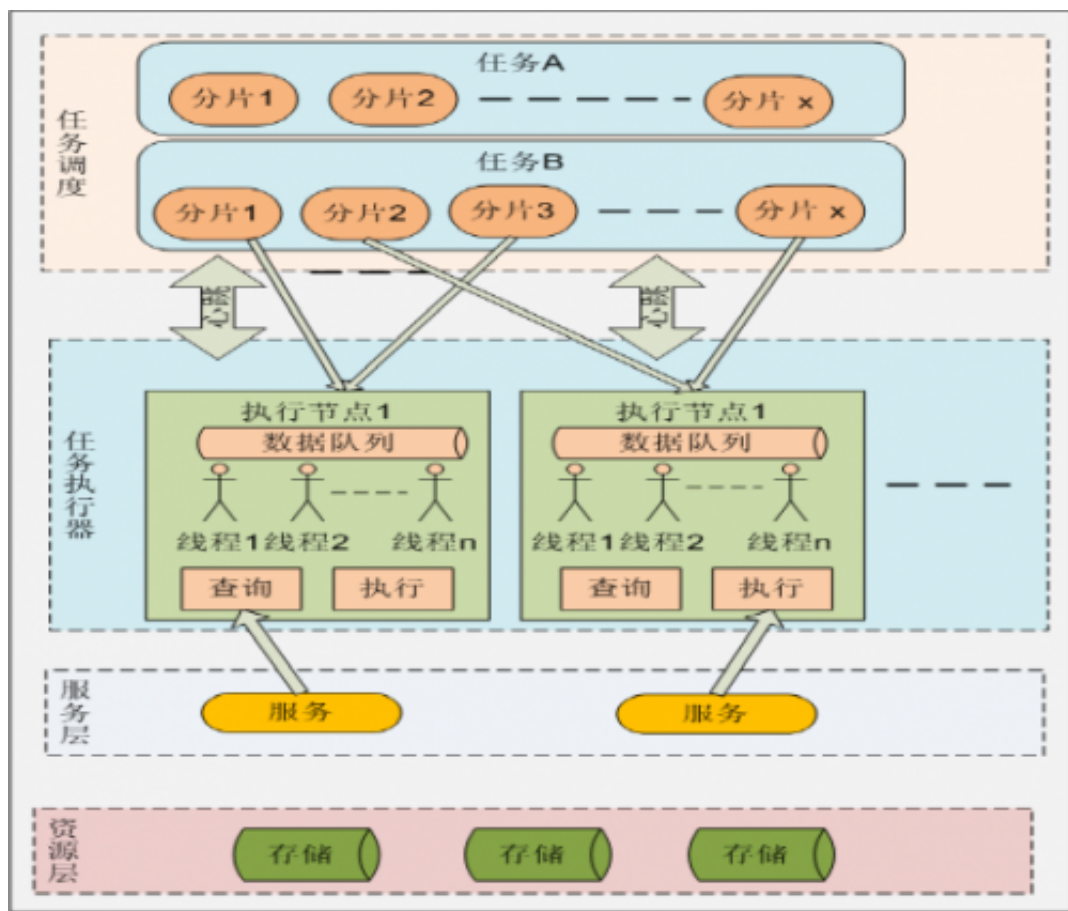
分布式系统架构

而分布式任务处理架构图如下，在分布式任务处理引擎的基础上，我们可以通过在图下方左侧的分布式任务队列（Redis缓存）中取任务，然后由我们的引擎一步一步去调度相应的服务，将结果返回给对应的服务进行业务处理，同时也返回我们需要的结果，真正将交易快照数据转换为生产单据，最后将数据推送到客户端系统，比如库房系统、POP商家系统。



分布式任务处理架构

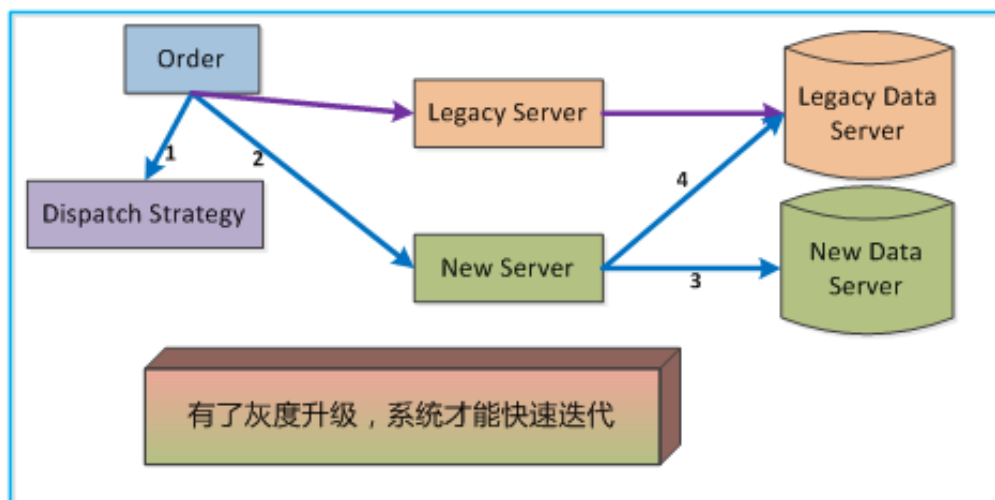
分布式任务队列设计可以通过下图说明。首先我们会对任务进行分片，定义每一个任务为一片，然后将任务在任务执行器中加以处理，而每个任务执行器又有多个线程去处理和调用不同的服务，最终再将结果返回。这里还会有一个异常任务队列，对于那些处理失败的任务，会将其放入异常任务执行队列，进行异常处理流程的执行。可能大家会想，如果系统挂了，那个任务的数据也将丢失，没有执行完任务怎么办。事实上只要我们的原始订单数据存在，完全可以恢复为再次执行，最多执行会缓慢，但不会导致数据一致性出现问题。



分布式任务队列设计

我们的分布式任务队列采用了工作流的机制，支持灵活的流程配置，这主要通过Zookeeper来进行分布式配置，可以动态添加业务处理环节。同时，可以自动调节系统的吞吐量，当任何一个环节出现问题时，都会进行自动降速，当问题得以解决后，我们会进行自动增速，保证系统的吞吐量；我们还可以通过配置对一些高级别的订单进行优先生成处理。

系统部署如下图所示：



系统部署图

一直在路上的OFC人

从2011年6月初只有五六个人的小组，到今天三十多人的团队，从开始只负责非客户订单相关业务系统，到今天负责公司核心的0级订单系统，OFC业务范围横跨了整个订单生产过程的大半个生命周期。是的，就是这些普通人，在纷乱中不迷茫，在欢笑中不忘使命，表面屌丝内心渴望前进的OFC人，义无反顾地坚持在订单系统的一线上。回望着走过的路，我们竟如此平静而淡定，遥望远方，我们依旧充满激情，依旧会绽放阳光般的笑容。因为这是个伟大的时代，伟大的国度，因为这里有一群伟大的人在做着一件伟大的事情——为了让购物变得简单快乐，而我们便是那群人。



OFC开发团队及测试团队合影



本文摘自《京东技术解密》第4章节：完美风暴——海量订单处理，电子工业出版社出版。

京东高速增长、闪电响应的供应链、庞大的团队规模等背后内幕，对于业界一直像谜一样神秘。随着成为中国B2C领导厂商以及在纳斯达克上市，京东越来越需要开放自己，与业界形成更好的交流与融合。《京东技术解密》的面世，就是京东技术团队首次向业界集体亮相。本书用翔实的内容为读者逐一解答——如何用技术支撑网站的综合竞争实力，如何把握技术革新的时间点，如何应对各种棘手问题及压力，如何在网站高速运转的情况下进行系统升级等备受关注的关键词。