2

0

分享

40页PPT分享万亿级交易量下的支付平台设计

数据和云 发表于 数据和云 269

本文主要是根据作者在2018QCon演讲内容整理而成：

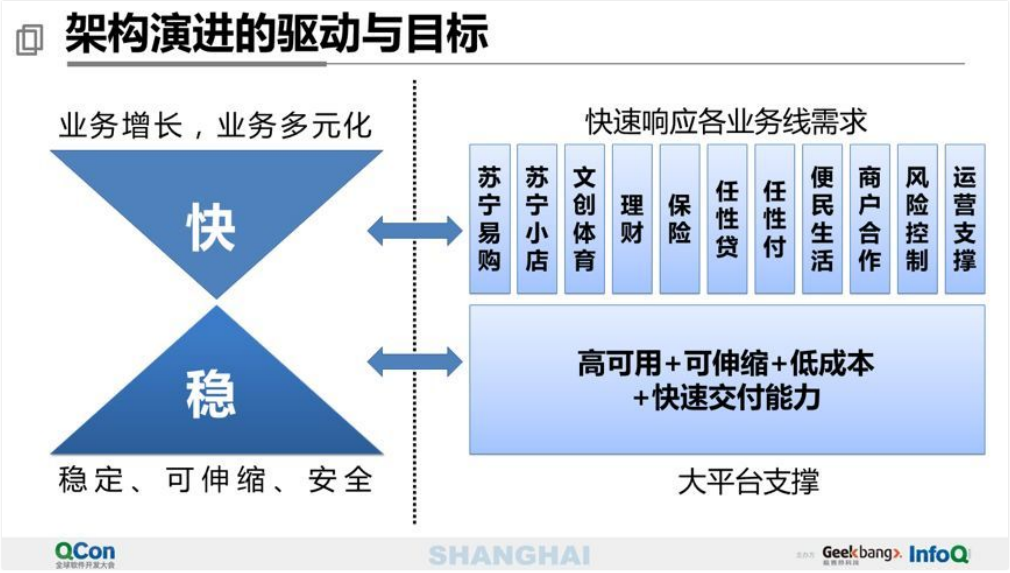
苏宁金融交易量3年内从1000亿增长到万亿+，服务用户3亿+，服务场景从服务于苏宁易购内部生态，扩展到服务全渠道，全场景，多业态的线上线下智慧零售的开放生态圈，一方面要满足公司业务发展要求，快速研发新产品，另一方面要满足818大促，双11等大促设计要求；

本次主要介绍苏宁支付系统如何实现**500天性能提升2000倍，从100笔/秒提升到20万笔/秒**，给飞行中的飞机换引擎，将包括三大章节六个部分： **苏宁支付平台发展历程**，以及现在**运行的总体架构设计**，以及**配套的可视化作战指挥系统**，以及在业务急速变化，万亿级交易量的状态下，如何对全**局架构进行优雅地重构**，以及**重构过程中的实战案例**，最后介绍一下我们**目前规划的、对未来的展望；**

具体技术包括**高可用设计技巧，高伸缩性设计思路，弹性的流量和资源控制，异地多活，全链路压测，消除数据瓶颈与单点，热点追踪与防护，故障自愈，账务系统之大账户瓶颈解决方案，以及未来怎么实现机器人自动巡检和自动修复等实战经验分享。**

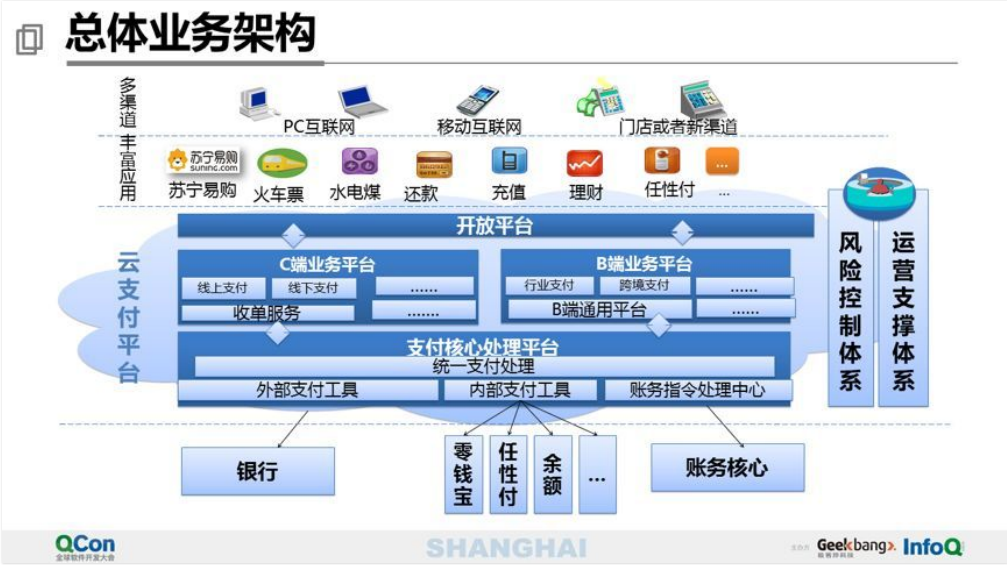
第一部分，我们介绍苏宁支付平台的演进路线， 架构演进的驱动与目标；

苏宁支付平台演进经历了四个阶段：从传统的架构，到SOA架构，到云计算架构以及目前的智能支付引擎；服务场景也从单一的服务苏宁易购，到服务苏宁内外部生态圈，再到提供行业解决方案。TPS从100到20w+的支付处理能力；交付周期也从最初的按月交付到现在的准实时交付。

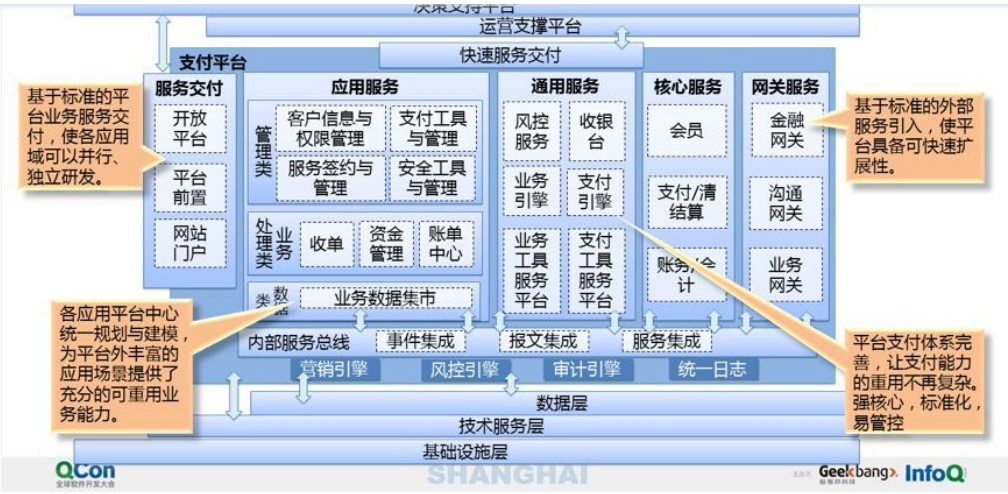


那是什么驱动我们进行一次次的架构演进呢？驱动力和目标是什么呢？支付平台是整个金融的基础设施，也是公共设施，服务于几十个事业部的几百条产品线，如果每一条产品线提一个需求，那就要同时响应几百个需求，同时还要面对业务的大促，因为苏宁是O2O的模式，业务场景会更加复杂，线上线下都有：**线下的**五一、国庆；**线上的**418、618、818、双11、双12，**基本上每两个月就有一个S级大促**；一方面要保证业务需求的快速响应，另一方面也需要保证大促的安全稳定，**对来说**

第二部分介绍现在正在**运行的总体架构设计，包括总体业务架构设计，总体系统架构设计，总体技术架构设计，关键子域的架构设计；**



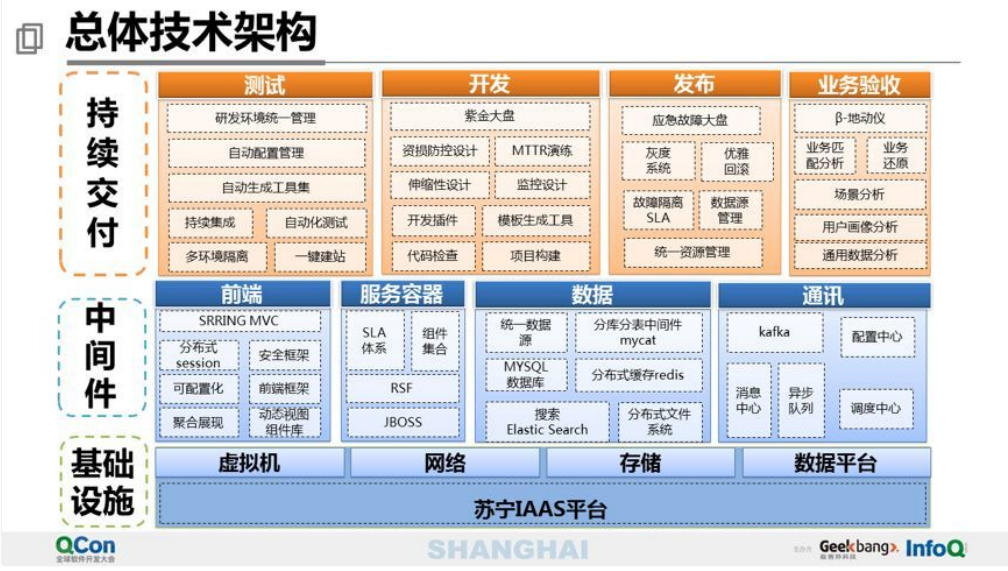
首先是总体业务架构设计，主要包括4个部分：第1部分是我们服务的渠道和场景，包括SDK,WAP,PC各端，线上线下门店；以及电商体系的应用，金融APP的应用等；第2是我们的合作方银行：包括中农工建交等以前的直连银行，以后后来的网联，银联等；第3是贯穿整个全流程的风险控制体系与运营支撑体系，包括欺诈风险，信用风险，以及配置产品，运营的各种支撑系统；第4是**云支付平台本身。在云支付平台中包含三大子架构域**：一是开放平台，把我们内部的服务统一开放出去给各个渠道、各个服务去使用；二是对这个平台进行层层抽象，将c端业务平台当中线上线下的公用逻辑抽取到c端业务平台，b端业务平台当中各个行业支付的公用逻辑，我们会抽取到b端公用平台。第3作为支付核心，会统一整合内外部支付工具以及账户核心操作指令统一提供给上层使用。



业务架构决定了系统架构，**从纵向来看**，我们分为应用层、数据层、技术服务层、基础设施层，以及贯穿整个全流程的决策支持与运营支撑层。

从横向来看，分成面向用户和商户的服务交付层（通过开放平台交付给我们的合作伙伴，通过这个平台前置服务于苏宁易购生态圈的各个应用场景）；应用服务层（包括业务处理类、管理类、数据类）；通用服务层（即平时常见的支付收银台、风控、合同计费等）；核心服务层（包括会员、账务核心、清结算等）；网关服务层（因为我们需要集成外部的一些服务，包括金融服务，通过金融交换服务去做；沟通网关，面向运营商的；业务网关，面向和我们合作的商户的）；

整体架构的设计，我们采用了插件式的架构设计思想，比如服务交付层，我们基于标准的平台业务进行服务交付，这样可以使各应用域独立并行的研发；对网关服务层，我们基于标准的外部服务引入，使平台具有快速可扩展性。

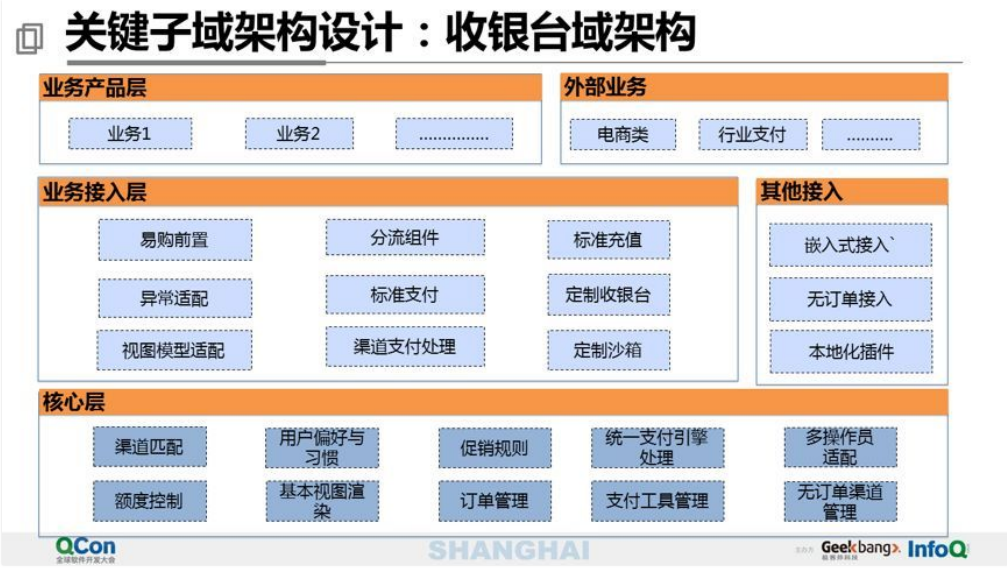


业务架构和系统架构决定了我们的技术架构，技术架构包括三大部分：

持续交付层，以及支撑我们持续交付的**中间件层**以及**基础设施部分**：持续交付重点有两个，**第一是快：**开发快，所以我们有开发的插件、模板生成工具；测试快，从自动化测试到持续集成，到一键建站的统一拉起；发布快，有现成的发布流程支持；**业务验收快，这个是我们支付平台独有的一项，上线之后要做业务匹配分析和还原**，这个有两点好处：（1）对业务来说，可以快速地知道需求有没有按照业务预期的开发；（2）对研发人员来说，可以快速地知道研发的需求是否获得了业务收益。这样，业务和研发人员有了统一的视图。

第二是稳：开发的过程会做这一些非功能性的设计，如：伸缩型设计，监控设计，资损防控设计；资损防控设计有三层：第1层是开发与测试，第2次是监控与核对，第3层是止损与追款。平稳，发布

现回滚，对用户没有影响；



接下来讲关键子域架构设计，从收银台到交易，从支付服务到支付引擎，我们如何实现标准化和插件化。首先收银台分为三层：第1层是业务产品层；第2层是业务接入层，会做一些异常的适配，如不同的errorCode可以展示不同的异常界面，对用户体验比较好。第3是核心层，用户偏好与习惯、额度控制等。收银台是从一个简单的收银门面，千人一面，逐渐发展到千人千面，再到一人千面；就是不同的用户在不同场景下看到的收银台是不一样的，到同一个用户在不同场景下也可以进行个性化的适配；

这个是收单系统和交易系统的介绍：分成两部分，一个是公用的部分，就是我通用的上下层的依赖，比如支付引擎、会员、收银台、收费计费等，中间的收单服务层可以通过设计模式去封装，根据不同的业务请求，然后统一做收单路由到不同的插件去处理。这样即使有几十条产品线同时请求，我们也能同时响应，因为只需要改动一个插件即可；

基于这种插件式架构的设计思想，接下来的支付服务也是这样设计的。

支付引擎会整合银行相关的**外部支付工具**，以及零钱宝、任性付、信用支付、现金贷等**内部支付工具**，同时进行**账户操作指令的封装**（主账务核心和各个业务的微账务核心）。对我们来说，账务核心、支付工具群和支付外转接中心都是一个个插件，开发速度快；**在具体的一个支付工具内部，也是插件式的**；这样就完全可以实现大规模的并行研发；

最后是网关层，面向接入银行渠道和接入合作大商户：第1层是报文组装解析层，第2层是适配器层，第3层是路由引擎；由图可见，每家银行的公用逻辑相同，可以通过设计模式封装。不同的是**输入参数的获取策略**以及**输出参数的不同阐释策略**。具体实践时代码结构上可以用设计模式来封装；实现代码上每个银行的输入报文的不同，可以用velocity模板来做。在返回报文时，每个银行的错误码和异常处理机制也不一样，可以通过groovy脚本来解析，这样对于接入新银行和商户不用做系统发布，直接配置即可，实现插件化可配置；2015年前公司一个月接一家银行，2016年后可以实现一个月接几十家。

基于前面的整体架构，我们思考，如果每个系统中的模块、调用链都能可视化，不管新的优化、重构、还是做业务需求都能快速实现。

基于这个理念，我们做了一个可视化作战指挥系统。包括三大部分： **研发的可视化**： 聚焦统一目标下的交付全链路、全资源可视化；统一目标是指公司的战略目标，从上图可见，战略目标KPI一定极简指标，要定北极星指标，一般我们会定三项，战略目标分解到事业部，事业部分解到研发中心对应具体需求，而需求的整个研发周期已经可视化了，可以清楚知道每一个需求、每天做的事情是不是帮助整个集团在完成战略目标。

管控的可视化： 组件自治，资源弹性调度；每逢大促尤其是洪峰时候，需要执行应急预案，我们就需要知道，执行应急预案之后影响的用户场景，以及各个硬件执行过程当中的操作的步骤。

可视化作战系统架构设计，这里面除了平时的一些常用的技术设计之外，还有三点核心的设计思想：第1点，对研发来说，体现在可视化整个研发生命周期，但是起点与平常不同，**平常的起点可能就是一个需求，但这个的起点是战略。**第2点，**运行时关注不稳定性的因素，去主动分析、依赖分析和变更感知。**分析变更感知的前提，是需要对每个系统做SLA；

第3点，为了进行平滑的管控，需要做几个工作：制定应急预案后进行线下，线上的演练，除了线下测试环境的演练外，生产环境也需要实际的演练，比如划拨一定量的生产用户，调度一定量的业务场景，也会**自动注入一些故障，尽量让演练流量的结构构成接近真实流量，以保证演练的真实性。**如果故障没有经过演练，真实发生是不可控的；**故障能否快速恢复，能否自愈，对用户来说是不是感到平滑。**

苏宁金融集团年交易量已过万亿，日均资金流水几十亿，需要保证每一笔交易资金的安全；**对这样业务需求极速变化的高并发金融资金系统进行重构，就犹如对发射出去的导弹进行二次加速，任何一个小失误，都可能导致上亿的资损，影响上亿的用户体验；**那么在重构过程中，如何保证优雅就非常重要？首先需要确定我们的目标是什么？基于这个目标我们的困难是什么？解决方案是什么？怎么去实施？怎么去演进？怎么去验证？

基于这个愿景，就要满足两点：**快和稳**； 第一个是快：我们需要对业务敏捷、快速响应业务的变化；这个也是研发中心的核心使命，能对集团业务能够很好的支撑，甚至是驱动业务的发展；第2个是稳，性能要高（20万TPS、高可用），平时会考核MTTR，出现故障后多久能恢复，10秒、20秒还是一分钟？通过这个指标去牵动其它所有的工作的优化，避免指标太多，工作没有重点，不能聚焦；**比如定10个指标，每个指标权重10%，看似面面俱到，其实没有重点；**但是系统指标有很多，有成功率，耗时，异常率，各个硬件的使用率等等，**作为负责人要找到北极星指标**；我们的北极星指标就是MTTR；以及这么多系统能否实现**弹性治理。**

基于目标，然后识别关键问题，那我们的关键问题有四类：

1、交付速度：基于标准的复用，并行、分布研发；2、高可用：需要分析故障点，建设DB单点/热点防护、自动化运维、服务自愈、应用级灾备能力3、可伸缩：从应用到IDC、服务器、网络，做到全网伸缩；4、低成本：不仅要节流，还要开源，重点是公司盈利和控制资损，通过技术对业务驱动力产生的正向价值，产品运营效果评估，这个也是对团队负责人的一个要求：要善于做技术产品化和技术品牌的运营；

高可用的重点是故障识别与应对，即对故障源的实时感知和可视化的治理。**故障源来源：**按照我们的服务模型分三方面：提供的服务、服务本身、依赖服务。**提供的服务：**故障来源在于请求，比较经常出故障的是：重复请求、并发请求、超量请求。针对每一个请求我们用不同的策略进行处理。**外部服务：**首先检测通信是否正常，通信正常后服务是否可用，服务可用后响应是否超时，这些都没问题后功能契约SLA是否满足，这样就形成了一个体系化的处理方法，就不会遗漏，也便于团队的知识传承（不论是代码结构设计，还是团队设计思想的统一，都是比较好的）

从PAAS平台进入到IAAS实现全网可伸缩。

对外提供的服务是吞吐量、单资源存储量的上限、响应时间；**内部服务**：关注DB、数据库总连接数、单数据库每秒事务数、慢SQL；**依赖服务：**银行实时清算能力、关键服务访问量等；这个是个可伸缩的框架，接下来我们详讲一些关键系统的可伸缩设计，具体是怎么做到的，交易系统、支付引擎系统和账务核心，如何实现可伸缩的。

首先是交易系统的设计，大维度上，将B、C端拆封开，然后进行读写分离，再对写进行分库分表。这里要特别讲一下分库分表中间件有两个特别的功能：灰度支持和影子库表支持。灰度支持即用来

支付引擎即支付服务的分库分表策略，按照用户维度进行拆分。这里面有个核心设计，我们设计了一个**逻辑数据源**，而不是物理数据源，便于迁库，减少DBA的运营成本。

账务系统可伸缩，难点在于热点账户；热点账户即资金处理频繁、时间点密集、基于等待的数据库排它锁的大账户；正常情况下，我们开发人员首先是切入系统进行优化，但实际**以业务为切入点会更好。首先对业务场景进行优化**，实现进出资金隔日，业务错峰，拆分收支账户，收款方到账准时实化，收款方到账准时实化；**接下来才是系统上的优化**，系统上做到分布式锁、资金资源池机制、缓冲记账、并发控制、异步化；**最后是账户层面的优化**，不同账户制定不同的策略。比如中间账户：只登记账户明细流水，不更新余额，日终进行汇总轧差，一次性更新；待清算账户：采用单边记账方式，待清算账户不做余额更新并且不登记账户明细流水，待日终进行单边汇总；特定业务收费账户：异步分段补账等，这样的优化才是最有效的，而且扩展性好，后期的维护成本也低；

在解决了核心瓶颈点之后，设计架构的演进路线也很重要；因为我们做系统重构是不能停业务的，就好比飞机在飞行的过程中，进行换引擎的操作；基于以上目标，我们进行架构重构，就需要注意这三点：

1、**要与业务发展路线合拍，顺势而为**：切忌沉浸在自己的技术世界里面，因为公司首先是盈利组织，研发首先要服务好业务，才能驱动业务，才能长远发展，阻碍业务发展的研发团队和研发技术方案是不长久的，特别是对于一个竞争对手激烈的高度发展的公司；

2、**专注主线、边界优先，步步为营**：就像装修房子一样，可以先把墙装好，但内部的每一个房间不一定马上装修，因为我们内部是可控的，内部系统重构可以放后，但是要先做提供给边界系统的接口，这样既可以很好的控制风险，也便于多团队之间的协同作战；

3、定期可视化投入产出比：因为金融系统不可能把业务停下来，他一定是在业务发展的过程当中，需要实时评估，重构和业务发展是否合拍，这样便于获得集团的大力支持，减少各方的不理解，减少很多不必要的阻碍，消除部门壁垒，消除决策层的顾虑；

实施的过程当中，如何管控项目？

项目前：需要消除风险，获得支持，确定项目价值与范围，明确业务影响 ，获得相关干系人支持，用架构概念验证原型；

项目中：做到短、平、快，严格控制项目范围扩张，Rebase不可避免的业务需求；

项目发布时：做到稳定，用户体验连续，基于场景的立体化监控与报警，比如有时监控时我们常发现单个系统的耗时、成功率等指标都正常，但实际影响了整个调用链，影响了某个场景，所以一定需要基于场景做立体化监控。其次，**每个核心链路上的系统一定要经过应急预案的演练。**然后，**要保持悲观主义的心态**和**终结者的思维**，潜意识里面要假设重构时一定会有问题， 所以一定要守好上线的最后底线，要做到快速回滚和降级。

最后是阶段性复盘，聚焦目标，防止做的需求偏离业务目标，这一点对于研发管理者特别重要，通过这些回顾逐渐拿到和业务方的平等话语权；慢慢就会建立起和业务的一个很好的沟通渠道；

架构重构后需要验证合理性，怎么知道我们设计的架构就是满足我们的预期呢？主要包括3个方面：

1、新老场景的推演：新老场景是否都能支持；

2、核心服务的推演：上下游系统需要演练是否稳定 ，

3、非功能性的推演：系统是否可隔离、可配置、可监控、可回滚、无单点、无状态等；

主要推演架构的高可用、可伸缩、研发成本，运维成本和迁移成本5个指标，通过这5个指标在上述3个方面的推演，基本上就可以验证架构合理性了，其实做架构决策也可以参考这个依据，比如多个

接下来介绍重构过程中的一些经典案例，便于大家可以在自己的公司里面快速的复制和集成；

第一个：两周建成立体化监控体系：为什么需要监控？因为重构过程中，随时有可能出问题，所以需要一个监控系统能随时看到重构的链路情况。**为什么是两周?**

因为重构对时间其实是有要求的，需要快速完成。如何在两周之内建成一个立体化监控体系呢？有几点：1、指标要极简2、可视化3、管控全网化（规则报警、一键定位、洪峰控制、业务降级），4、统一日志模型，针对重构系统的变化，要保证监控是准确的，所以需要对服务模型抽象出三层（服务的使用者、服务的提供者、服务的集成者）打日志，每个服务的接口都会打digest摘要日志。**实现过程中**，会请架构师或资深的技术经理搭好插件化框架，完成日志模型搭建、异常体系的建设，领域模型，对输出结果的阐释策略。后面程序开发时，开发人员只需要开发对应的插件即可，这样就将**程序的设计和重构变成了工厂的批量化生产**，所以这个维度上其实就能减少很多风险。做到以上几点，才能保证重构过程比较平稳和风险可视化。

第2个案例，全链路压测：我们的压测系统经历这么几个阶段：

首先是线下压测阶段，这会面临3个问题：**测试环境和生产环境配置不一致**（比如生产环境配了10个数据库，而测试环境只有5个）；**测试环境和生产环境数据结构不一致**，（生产环境有些用户可能有50个订单行，而测试环境基本上为了测试方便，可能只构建了1个订单行）；**用户访问路径不一致**（比如生产环境用户从4级页到收银台有4步，而测试环境直接从金融App进来1步完成。漏斗不同决定了大促的流量比例不同，比如前面有1万TPS，经过3层漏斗只剩1000TPS如果只有2层漏斗，可能剩下5000TPS，对资源的消耗是不一样的）

第二个阶段，我们开始做**生产憋单压测**（比如代扣的订单憋在一起，从收单到支付服务到银行，都可以进行压测，但是对用户进入收银台前面的路径获取不到，基于这个缺点），这样可以实现部分链路的生产环境的真实流量压测；

第三节阶段就是，我们目前正在使用的**全链路生产压测**，就是把全链路串起来；当时我们做的时候，需要解决3个问题：（1）服务的用户商户怎么办？（2）银行不配合压测（3）如何保证支付链路系统的配置准确。解决方案是：在易购建立测试商户和账户，配置虚拟银行，配置影子库影子表，改造中间件，增加影子库单号判断，做到与生产用户数据隔离。

在多活这个方面，我们也经历了几个阶段的发展，初期因为业务量小，我们做了一个**稻草系统，它是一个最小可用支付系统**，只关注80%主要的银行和支付工具，即便出问题时，能保证核心链路仍可用。这个系统在交易量大肯定是有问题的，下一阶段就开始做**支付核心链路failover**，但是仍不能解决机房出问题（如停电问题，网络设备问题等）。

所以**后来做了多活**，要做多活就要解决几个核心问题：跨机房耗时问题、依赖服务部署与治理、研发体系配套改造、故障切换的工具支持。我们的解决方案主要是：

1、支付链路单元化;

2、消息同步服务化;

3、依赖服务做多活部署;

4、研发体系的配套支持，支持发布到金丝雀环境等;

5、机房选址，跨机房调用其实存在很大延迟;

6、容灾能力，支持机房级容灾，按系统、按链路容灾;

从单机房到多机房，在架构演进过程中，也要支持容灾，因为演进过程中要做系统发布，逐步切流量。比如整体流量先切换白名单用户，再切换1/256,1/8,1/4，到1/2等，也可以针对单个系统的切

热点防护包括3部分：

1、**发现热点，**感知热点源，通过埋点，关注请求，关注整个链路依赖的资源；

2、**热点诊断**，主要通过实时分析，离线分析，上下游分析;

3、**热点治理**，最粗暴的直接限流，这个是有损服务，是一刀切。

可以稍微再优雅一点，进行故障隔离，比如由于场景1导致的问题，可以直接把场景1切调，对其它场景没有影响，这样可以做到稍微精细化一点；**业务场景优化，比如账务核心收支账户分离；热点结构优化，**比如说我们在收银台上有一个活动，只取前1000名满100减50，其他人没有资格，其实是通过**优化缓存结构实现的**；**热点拆分，对数据库进行分库，对数据表进行分表， 对记录进行缓存机制处理。**基础服务包括统一日志，服务的SLA，决策支持；应用工具包括系统级的紫金大盘、产品级的地动仪，这两个其实是可视化工具，用来观测治理之后的效果。

接下来讲从100tps到20万tps的实践过程。

横向看，从总体架构优化到应用程序优化到数据程序优化到技术组件的优化；纵向看，深入到每一个架构，从收银台到收单到支付服务到渠道到账务核心到清结算进行优化。**应用层**看链路能否缩短，再看内部服务能否治理，再到线程池调优，去事务。

数据层优化，需要考虑收益，比如SQL优化排第一位，因为比分库分表的收益来的快。原则是能用单库尽量用单库，不能用单库时，才考虑分库分表。DB配置参数优化，可以优化引擎参数。因为优化过程会产生资源消耗，所以仍然**要考虑业务目标。基础技术平台优化**，要遵循体系，服务的本身，依赖方，服务方。

中间是RSF分布式服务框架（内部通过这种服务来进行路由和调度。数据方面，分库分表组件和缓存；通信方面，调度组件）的优化。**接下来是存储**，如SSD，以前是普通硬盘，那么IOPS会比较低，如果本次优化目标是提升2倍，那么只要更换SSD，速度快，成本低，对用户也没有损伤。

闭环验证中心，这个是我们做性能优化的一个亮点：特别是重构、性能优化的时候，需要快速知道结果是不是我们想要的，下面的服务日志、调用日志都是比较通用的。

监控决策中心：提供灰度方案，移动端应急管控。虽然日常有规则管控，做SLA、流控，但是关键的核心系统出问题（如支付服务），仍需人工介入确认是否执行降级和流控，因为一旦流控，会对用户产生影响。我们最近也在建设大促机器人，实现自动巡检，和智能的治理，这个后面会讲到;

然后需要验证大屏，可以直接看到对门店或交易是否有影响。 其实在做系统性能优化的时候，也会伴随研发组织与研发过程的“性能优化”;

这个我有一篇文章《业务需求极速变化的高并发金融系统性能优化实践》，里面有详细的分享，主要介绍苏宁金融对业务需求极速变化的高并发金融系统进行性能优化的实践经验，主要包括：智能监控系统，瓶颈点驱动的性能优化，全局规划驱动的性能优化以及研发组织与研发过程的“性能优化”；核心技术点包括瓶颈点的可视化诊断，瓶颈点治理，热插拔架构设计，链路failover设计，应用N+X设计，异步化，数据库单点与热点账户防护；也包括从网络，中间件，应用层，数据层，DB的横向优化方案；以及从架构，代码，会话，缓存，线程与队列，事务，堆内存与GC的纵向优化方案，横纵向结合的体系化解决方案实践;

以上讲述了指导思想和方法论，具体需要怎么操作呢?

可视化瓶颈点很重要，这里我举一个示例，比如银行网关系统，调用22次，透明化每一次调用，调用依赖系统多少次，每个系统的SQL有多少，这样可以清晰的可视化链路，保证快速知道哪里出了问题。然后就可以进行庖丁解牛式的优化了;

案，所以需要有个地方可以看到问题影响面、执行什么操作可以恢复。实现这样的系统需要三部分：故障源感知、智能诊断引擎、故障治理。

故障源感知：指标分为3类，业务指标、系统指标、基础指标。观测指标发现，几类容易出问题的地方：

（1）系统变更，出故障时首先问，昨天有没有发布？系统变更占权重很大；

（2）突发业务量，可能某个商品突然很火爆，大促前估不准业务量；

（3）操作失误，拓扑获取和链路追踪，知道调用链出了什么问题；

（4）单点追踪；

（5）安全攻击；

通过诊断业务系统暴露的问题，可以将其指标化，才能便于工作的落地与执行：比如高可用时，不能定9999多少个9的目标，可以定MTTR=1分钟的目标。以前会定A完成日志模型，B完成SQL优化，C完成异常治理等，但一段时间后发现并没有解决问题，**后来我们定了“北极星”指标MTTR=1分钟**，这样技术经理自然的知道需要完成日志模型优化等工作。通过这一个指标去牵动其他指标的达成。

故障治理有3方面：场景修复、链路修复、服务修复；服务修复需要提前定义执行引擎，WAF防火墙到负载均衡到RSF到SCM到TCC到DB，形成一个体系。出不同问题，会执行不同的应急预案。**最后针对不同的业务流和资金流，做异常数据比对。**

最后介绍我们正在做的事情和对未来的展望。

机器人巡检，因为大促对我们来说是很重要的节点，每次也耗费很多人力物力，可以实现宏观上系统的整体化构造，微观上看到每个系统的状态。

全网可视化作战沙盘，上面的几个指标被称为“北极星指标”，不是开发部门自定的，是根据集团战略目标分解的。战略目标分解到研发中心，研发中心分解到每一个项目。右上角的+号有3个功能：1、全景的产品视图，2、系统的治理，3、考核，每做一个优化需要考核，这样每个团队能形成同一个目标去做事。

这样就将人，事全部链接成了一个整体，所有的工作都将形成闭环，同时所有决策层都能可视化的看到；便于所有工作的快速推进和落地；

我写的一篇**《从百亿到万亿：如何打造一支承担企业战略使命的研发团队》文章**中，有更加详细的分享，欢迎一起交流。

> 作者：肖军，现担任苏宁易购集团总经理助理；曾先后就职于蚂蚁金服和苏宁金服，专注于金融相关的产品研发工作；擅长互联网产品设计，高可用架构设计，体系化的研发团队管理。
>
> 设计过多款行业领先的互联网金融产品，并荣获数项金融相关专利；主导过多次双11,618,818等大促稳定性设计和保障工作；丰富的从0到1大规模研发团队搭建与管理经验。

出自：技术琐话公众号（ID；TheoryPractice）

原文发布于微信公众号 - 数据和云（OraNews）
原文发表时间：2019-05-20
本文参与腾讯云自媒体分享计划，欢迎正在阅读的你也加入，一起分享。

发表于 2019-05-20

数据和云

881 篇文章 82 人订阅

订阅专栏

Oracle/云MySQL/MsSQL“大迁移”真相及最优方案

错过血亏！一文搞懂Oracle锁相关视图及相关操作

让MySQL速度提升3倍的19种优化方式

再好的素质，再完美的规章，也无法取代人自身的素质和责任心

MySQL如何实现高可用?

我来说两句

0 条评论

上一篇：优雅的解决Retrofit RxAndroid关联生命周期问题
下一篇：探寻HTTP网络超时的背后真凶：拨开云雾的生产环境排查之旅