京东话费充值系统架构演进实践

应用平台研发组 京东虚拟平台 2017-05-13

关于系统架构升级资料，网上有很多，各有优缺点，没有不适合的方案，只有不适合的业务场景。本文从实践的角度谈下京东的话费充值系统架构升级的过程，希望能给读者带来一些启发和帮助。

背景介绍

京东的财报显示，在截至2017年3月31日过去12个月内，年活跃用户数增长了40%至2.365亿 。随着活跃用户数的增加，话费充值业务线的订单量也‘水涨船高 ’，同时对系统的各项运行指标要求更高，老的系统架构不足以支撑新的业务量，需要对系统进行升级。升级方案从以下几个层面进行。

1、应用层面

引入缓存

在应用层和数据库层增加缓存层，热点数据放入缓存。如系统中常用的开关、白名单等数据，读取频率高写入频率低，针对这部分数据就可以在JimDB(Redis)中存储一份，JimDB (Redis)会把高频数据存储在内存中，读写性能很高。数据写入缓存时设置一个有效期，更新数据库成功后，异步更新缓存数据。如果实时性要求不高，也可以等缓存失效后，主动更新缓存。引入缓存层，降低数据库压力，提升系统响应速度。

编写并发处理程序

多任务并发处理，充分利用CPU资源。无依赖关系的多条任务可以并行处理，提高系统处理能力。如结算任务，每笔订单之间的结算操作没有依赖关系，可以同时执行多条结算任务

系统结构优化

核心生成流程异步处理，接收用户订单和给用户充值两个流程异步化处理，提高系统处理能力。对用户来说，用户付款成功，等待充值即可。系统可通过worker触发充值动作，设置合理的重试次数，间隔一定的时间进行重试。在到达最终状态前，给用户显示中间状态。

拆解大应用，使其微服务化。如话费充值应用，核心功能有PC端接单、移动端接单、MQ消息处理端、后台管理端、worker端等五大块。拆解之前，话费应用只有三个系统，系统之间共用service\manager\dao模块代码，通过Maven构建管理。实现一个后台的小需求，都需要考虑PC端、移动端、后台管理端代码是否会受到影响，是否需要全部回归测试。更恐怖的是上线，充值应用有将近100个实例，修改一行代码都需要全部上线，即使没有受到影响的系统也要发布新版本因为要保证线上和代码库中的代码一致。

从功能上进行微服务化以后，应用拆解为PC端、Server端、MQ消息处理端、后

台管理端、worker端等五个应用，应用之间功能独立，依赖公司的RPC框架（JSF）和消息框架（JMQ）进行通信。单个应用的内聚性更高，应用之间的耦合度更低。再实现一个后台的需求时，开发、测试、部署都只需要关注后台管理端系统即可，无需再关注其他四个系统。

系统微服务化设计，关键点是如何寻找限界。

除了可以从功能上进行切分还可以根据关注点上进行拆解为生产端和支撑端，业务场景不同，寻找限界的方法也不相同，关键是微服务后单个应用的内聚性更高，应用之间的耦合度更低。

使大系统微服务化的方案有很多种，重点是制定好目标，逐步向目标靠拢。在服务粒度方面，如话费充值应用的MQ消息处理端，在功能上保持职责单一，只负责接收、解析MQ消息内容，具体业务逻辑处理交由相应的Server端处理。在技术选择上，公司有成熟的技术框架，如RPC框架JSF，消息框架JMQ等，这些框架都有对应的服务治理和监控等相关服务和团队。

不要为了微服务化而实施对大系统微服务，要确保微服务化之后，系统运行更稳定，应对变化更快速，开发更敏捷。

读写分离

实时性要求不高的数据读取从库，降低主库压力。如对账功能，读取的是前一天的订单数据，这些数据就没必要从主库中读取。关于技术实现上，Spring框架本身有提供，实现其抽象类AbstractRoutingDataSource即可。

变化频率低的页面静态化

充值应用中有很多卡片页，如QQ页卡等，页面上的数据变化的只有广告位。这种类型的页面就可以静态化，定时更新页面，推送到存储介质上，nginx配置location，直接读取页面，降低后端服务的压力。

数据库层面

当业务量发展到一定程度后，数据库就会成为系统的瓶颈。话费充值应用包含企业订单业务和普通用户订单业务，正是由于其业务的特殊性，采用了垂直+水平分库方案。根据业务类型进行垂直切分，不同业务类型订单数据独立存储，同一种业务类型在水平上由多个库保存。垂直+水平的分库方案能够最大限度的降低不同业务类型订单数据之间的相互影响，提高数据读写并发量。

普通用户订单业务，根据账户PIN进行hash打散可以均匀的分布到每个库中，sharding规则就是hash(pin)值，同时这个hash(pin）值还做为本地订单号的前缀，这样就可以通过账户PIN和本地订单号两个维度中任一维度都可以路由到数据库。创建ERP订单成功后，把本地订单号和ERP订单的映射关系保存到JmiDB中，对于只有ERP订单号的业务流程，可以通过映射关系找到本地订单号，有了本地订单号也就可以路由到数据库了。

而企业订单业务，每个企业账户的订单量不均，差别能达到三个数量级，如果再根据账户PIN进行hash打散分布到每个书库中的订单就会不均匀，不能使用这种sharding规则。根据本地订单号进行hash，然后再作为本地订单号的前缀。创建ERP订单成功后，同样需要保存本地订单号和ERP订单号的映射关系到JmiDB中，以保证在后续的业务流程中，能够根据ERP订单号路由到数据库。

拆分完成后，有的业务场景需要聚合查询数据，如订单管理。如果没有聚合数据，就需要在应用中，开发人员自行考虑聚合。通用的聚合方案是从每个库中查询一页数据，在内存中根据条件排序，返回一页数据，如果需要翻页的话，逻辑更为复杂。话费充值应用采用了第三方存储，把每个分库中的订单数据聚合到ElasticSearch中，查询聚合数据的场景读取ElasticSearch。

模拟MySQL slave的交互协议，解析数据库的增量BinLog，同步分库的数据到ElasticSearch中。由于数据库主从同步存在延迟的风险，需要准备一个降级方案。在话费充值应用中，数据库写订单成功后，插入一条任务记录，通过任务模型立即同步数据到ElasticSearch中。保证数据同步的实时性。

应用部署

计算机的CPU、线程、IO等资源都是宝贵且有上限的，当某一个资源耗尽时，那这台计算机上所有的服务都将停止服务。例如某一个服务依赖的第三方服务性能低，响应缓慢，这时如果客户端的继续请求，会导致该服务持续创建线程等资源，最终导致服务宕机。此时，计算机资源的隔离显得尤为重要。

在JVM内部隔离分为信号量隔离和线程池隔离，Netflix Hystrix插件提供了完美的支持。JD-Peer（多机房公网出口路由系统）中使用了Hystrix对每一个商家进行了隔离。话费充值应用对接了几十个商家，通过JD-Peer系统跟商家进行交互。由于某些网络原因导致其中一个商家A响应慢，持续的调用，所有资源都会被这一个商家占用，导致其他商家服务也不可用，最终宕掉。如何使用Hystrix进行资源隔离，可以参考《微服务利器-Hystrix设计》这篇博文，地址：
http://blog.csdn.net/a_fengzi_code_110/article/details/53643527

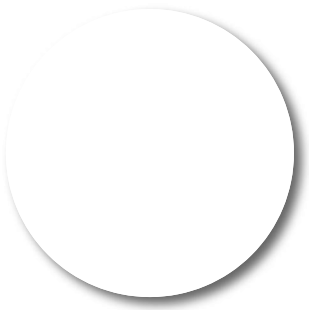
独立部署，物理隔离。每个应用分配独立容器，从硬件层面进行资源隔离。

多机房部署，从入口处分摊流量，提高系统整体的吞吐量。

版本发布

灰度发布，平滑过渡，异常情况下的版本回滚，要确保回滚前的数据在老版本中可用。如京东话费充值系统在发布数据库架构升级版本时，设置了数据流向开关，并对订单打标，同时缓存标识位。 开关打开时，数据进入新的数据库，开关关闭，数据进入老的数据库。线上验证阶段，一旦发现问题，可立即关闭开关。确保系统版本发布，对用户无感知。

在保证系统服务正常可用的情况下，进行上述一系列的升级，犹如给空中的战斗机更换引擎，稍不留神就会坠机，所以除了充分的理论储备，还需要综合业务场景，从自身业务场景出发，合理设置引擎更换方案。



经过上面一系列的改造升级，话费充值应用的吞吐量、运行稳定性都达到了最优的状态，历经数次的618和双11冲击，各项运行指标保持稳定，面对流量洪峰，岿然不动。

最后：
技术的更迭日新月异，但我们的初心不变，那就是：技术让生活简单快乐！

阅读原文