RocketMQ在联想大数据中的应用简析

# 一、前言

本文不涉及公司专利，仅为本人工作之余对RocketMQ在业务上的技术梳理和总结 。

众所周知，RocketMQ作为一款分布式、队列模型的消息中间件，具有以下特点：

1. 严格保证的消息顺序
2. 提供丰富的消息拉取模式
3. 高效的订阅者水平扩展能力
4. 实时的消息订阅机制
5. 亿级消息堆积能力。

所以，在客户的复杂应用场景中，以简化服务部署并充分发挥RocketMQ的强大优势，建议使用RocketMQ作为技术解耦的消息中间件。

# 二、场景分析

在联想大数据中，应用RocketMQ的使用场景中经常出现：异步请求，应用解耦和日志处理等场景情况。

（一）异步请求的业务处理，经常遇到如下两种情况，一种为串行方式的业务流程（如图2.1），另一种为并行方式的业务流程（如图2.2）。

串行方式：在联想大数据解决方案中，针对顺序化、流程化的业务场景经常使用串行方式，实现RocketMQ的技术解耦。如在某卷烟厂的解决方案中，针对制烟流程，将各个车间的处理流程做为独立的业务体创建一个Topic，对业务体中的各个处理阶段创建对应的Tag，并按照制烟流程顺序进行数据推送、清洗、业务处理及监控等，如下图2.1。



图2.1

并行方式：因业务的独立性，处理流程可分开进行处理，相互之间没有业务交叉。这种情况下，我们可以考虑使用并行方式对业务流程进行技术解耦。在联想大数据的相关解决方案中，针对某汽车的数据监控及相关画像的流程中，一部分是基础业务处理流程，另一部分是通过Bin Log数据进行数据监控与消息提醒等附加处理业务流程。针对这种场景，我们在给客户进行程序部署时，便采用了并行方式，如图2.2。

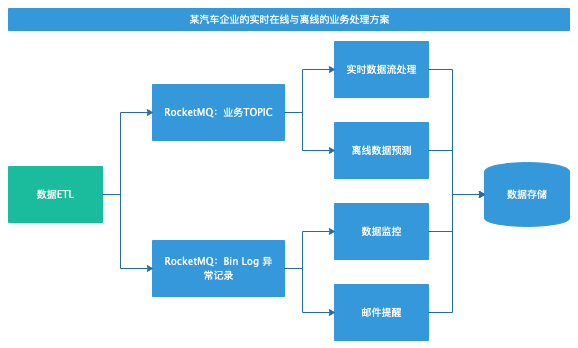


图2.2

（二）错峰削谷

虽然联想商城的并发请求数不像淘宝、天猫、京东商城等互联网公司的并发请求数高，但在技术要求上经过多年的业务总结和摸索过程中，总结出适合联想的技术架构体系，以解决高并发数据带来的业务处理压力，并实现技术架构的高可用。

在应对高并发，处理高可用，保证不丢数据的前提下，我们使用RocketMQ做为应对特殊的业务处理流程的技术手段，需要在数据生产端承接瞬时高数据流量，在数据消费端平稳地将数据推送到下游业务线。

基本处理方法采用业界普遍的“漏斗”模式，如图2.3：

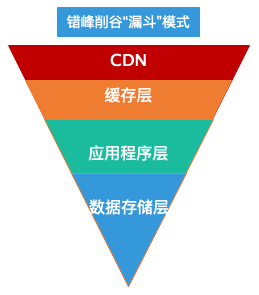


图2.3

（三）应用解耦

对于解耦架构来说，在联想物联网（IOT）的应用非常普遍。我们结合StreamSets进行二次开发，使用StreamSets通过界面上拖拽的方式制定数据流程，并在客户的解决方案中，说明RocketMQ区别于其他MQ组件的技术特点，针对客户的使用场景进行优化，使用RocketMQ进行解耦数据。

（四）日志处理

RocketMQ的设计模式借鉴于Kafka，且后者经常用于日志管理系统充当数据缓冲的角色。但Kafka过度依赖Zookeeper，而RockerMQ则可实现无Zookeeper部署，简化安装部署工作，所以，在联想大数据业务线和解决方案中，将RocketMQ应用于日志处理业务，逐渐增加RocketMQ的使用率。

# 三、应用实例

在联想大数据解决方案中，目前使用RocketMQ的实例也有不少，下面就举个实例说明一下，我们是如何使用的。

目前在联想大数据部门，我主要负责数据流组件研发，并基于StreamSets开源组件进行定制化开发。在给客户的解决方案或现场实施中，我们可以通过在界面上的操作，设定好相关参数，便可设计出符合客户需求的数据流。如图3.1所示：



在每一个功能模块上，我们只需要设置若干必要的参数配置，也可以通过在文本框中编写更多的参数配置，在启动数据流时，即可加载这些配置，并实现数据流中各个功能模块的初始化、启动等等功能。

在这个实例中，我通过开发一个RocketMQ的功能模块，并设定一些基本参数，如NameServers组、消费组名和Topic，便可从RocketMQ服务端获取数据。

通过使用这个工具进行设计，大大降低了使用人员的学习门槛，而且也简化了开发流程。在对外的解决方案或项目交付过程中，极大地赢得了产品和交付工程师的认可，在各个POI项目中也PK掉不少大厂公司。

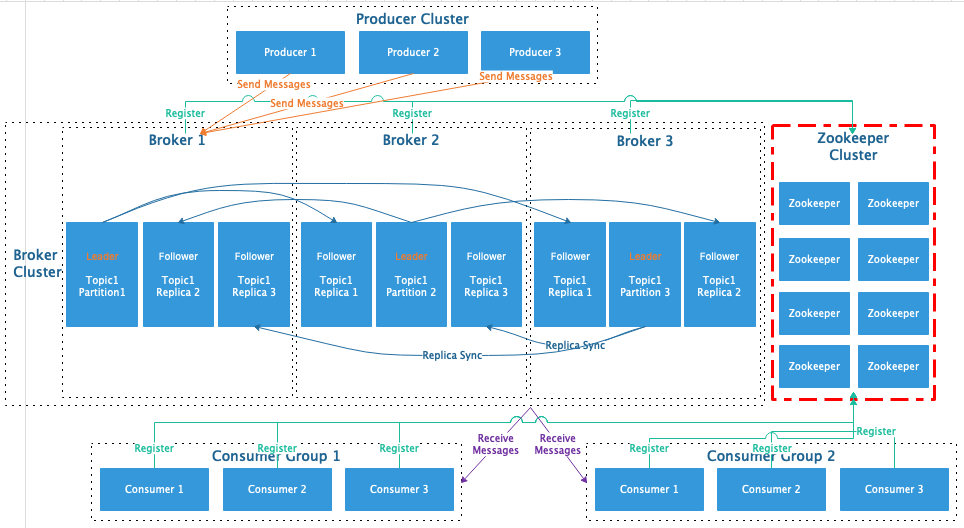
# 四、优势

在此仅对比Kafka，在实际应用中所具备的优势：

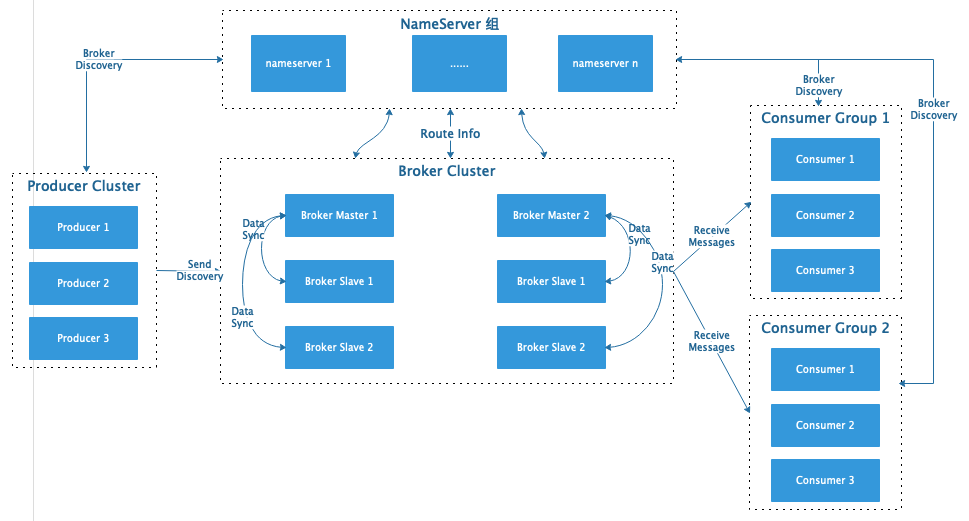
## 架构差异，简化部署

在集群部署上，Kafka的部署，必须依赖Zookeeper，并实现WaterMark的监控和Leader的选举。而RocketMQ可实现不依赖Zookeeper的部署，大大降低使用门槛和学习成本。

Kafka架构设计：



RocketMQ架构设计，参考官网架构设计：



## 性能优势

1、RocketMQ可支持上万Topic的创建和使用，并且不会影响实例的整体性能及处理能力。反观Kafka，我们在使用6000左右的Topic时，整个集群性能突然下降，并有部分节点出现卡顿现象。

2、写入效率虽略低，但磁盘IO不是瓶颈。

在实际测试过程中，同时写入Kafka和RocketMQ进行测试：

Kafka的吞吐量高达15~17w/s，体现出较高的吞吐量。这主要取决于它的队列模式保证了写磁盘的过程是线性IO，但此时broker磁盘IO已达瓶颈，写入响应已经出现略大的延时，有小部分数据出现重试写入。

RocketMQ的吞吐量基本保证在11~12w/s，磁盘IO率虽已接近100%，但消息写入内存后即返回ack，由单独的线程专门做刷盘的操作，所有的消息均是顺序写文件。

# 五、在“坑”中摸爬滚打

在近些年的使用中，联想大数据逐渐在提高RocketMQ的使用率，不仅仅因为其具有区别于其他MQ消息中间件的优势，并且学习成本略低于Kafka，更重要的是RocketMQ是阿里开源的消息组件，且在公司内部经历多年的版本更新，达到一个相当稳定的版本，而且各种学习资料也是相当丰富。

因此，在联想内部开始尝试RocketMQ作为各个业务线和解决方案中推荐使用的消息中间件。尽管在应用一种新的技术过程中会遇到各种各样的“坑”，但是在摸爬滚打的过程中我们痛苦并快乐着。

下面总结一下，我们遇到的一些比较有特点的“坑”及相应的解决方法：

* 坑一：在一个JVM实例中，初始化两个消费者Comsumer时，当关闭第一个Consumer，第二个Consumer会抛出“org.apache.rocketmq.remoting.exception.RemotingConnectException”异常信息。

此错误场景经常出现在并行处理的业务流程中。

当我们创建MQClienInstance时，因使用unregisterConsumer创建实例过程采用单例模式，并不是创建两个实例对象。所以，如果我们在一个JVM实例中，不同线程中使用实例对象时，当关闭其中一个Consumer实例时，便会出现此问题。

因此，当我们在创建Consumer实例时，

* 坑二：因consume offset 不能与commit log同时刷盘，导致offset不能对应数据。

因为consume offset最终要索引到commit log，假如commit log 没有对应的数据，那么consume queue保存的这个offset也没有意义，所以最终就算有了这个offset，还是要根据commit log来修复一遍，因为他们不是一起刷盘的。

所以，我们对源码进行了一定优化，将consume offset 和 commit log 通过一个事务进行刷盘，保证每一个consume offset 与commit log一一对应。

* 坑三：RocketMQ的common包MixAll类中默认指定WS使用8080端口，不能动态设置。

因为8080端口一般为Web应用使用端口，但当启动RocketMQ后，发现此端口被占用，导致某个Web应用程序不能正常启动。针对这个问题，我们在社区Jira中看到类似的Issue，并为提供相应的Bugfix方法。所以，我们在使用过程中，基于源码进行了二次开发，并在RocketMQ的配置文件中增加了对此端口的动态配置项。

在使用和学习过程中，还有很多的问题，我们还是处于学习和提高的阶段。

# 六、总结

通过这段时间的学习和使用，使联想大数据在对外项目和解决方案中，可以不仅仅使用Kafka作为消息中间件进行数据解耦，更能够使我们在项目交付和各种POC中，使用RocketMQ更加丰富地应对不同的使用场景。

对于RocketMQ，联想大数据还处于认识学习的过程中，目前我们更多地是在使用它，并没有完全研究透彻。希望以后积极参与到社区的讨论和研究中，贡献自己的一份力量。