**1-6 【图文】RocketMQ消息中间件集群架构与原理解析**

**初识 RocketMQ**

RocketMQ是一款分布式、队列模型的消息中间件，由阿里巴巴自主研发的一款适用于高并发、高可靠性、海量数据场景的消息中间件。早期开源2.x版本名为MetaQ；15年迭代3.x版本，更名为RocketMQ，16年开始贡献到Apache，经过1年多的孵化，最终成为Apache顶级的开源项目，更新非常频繁，社区活跃度也非常高；目前最新版本为4.5.1-release版本（2019-7-20日前）。RocketMQ参考借鉴了优秀的开源消息中间件Apache Kafka（这也是我们后面课程中重点要讲解的内容哦），其消息的路由、存储、集群划分都借鉴了Kafka优秀的设计思路，并结合自身的 “双十一” 场景进行了合理的扩展和API丰富。

**优秀的能力与支持**

接下来我们一起来看一下RocketMQ优秀的能力吧 ~

* 支持集群模型、负载均衡、水平扩展能力
* 亿级别的消息堆积能力
* 采用零拷贝的原理、顺序写盘、随机读（索引文件）
* 丰富的API使用
* 代码优秀，底层通信框架采用Netty NIO框架
* NameServer 代替 Zookeeper
* 强调集群无单点，可扩展，任意一点高可用，水平可扩展
* 消息失败重试机制、消息可查询
* 开源社区活跃度、是否足够成熟（经过双十一考验）

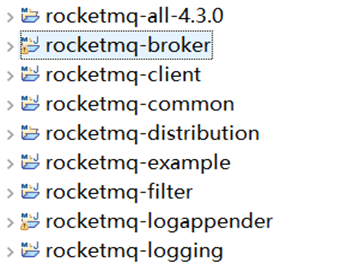
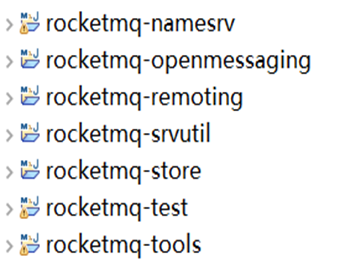
**专业术语**

任何一种技术框架，都有 “她” 的专有名词，在你刚开始接触 “她” 的时候，一定要了解 “她” 的专业术语，这样能够更快速、更高效的和 “她” 愉快的玩耍…

* Producer：消息生产者，负责产生消息，一般由业务系统负责产生消息。
* Consumer：消息消费者，负责消费消息，一般是后台系统负责异步消费。
* Push Consumer：Consumer的一种，需要向Consumer对象注册监听。
* Pull Consumer：Consumer的一种，需要主动请求Broker拉取消息。
* Producer Group：生产者集合，一般用于发送一类消息。
* Consumer Group：消费者集合，一般用于接受一类消息进行消费。
* Broker ： MQ消息服务（中转角色，用于消息存储与生产消费转发）。

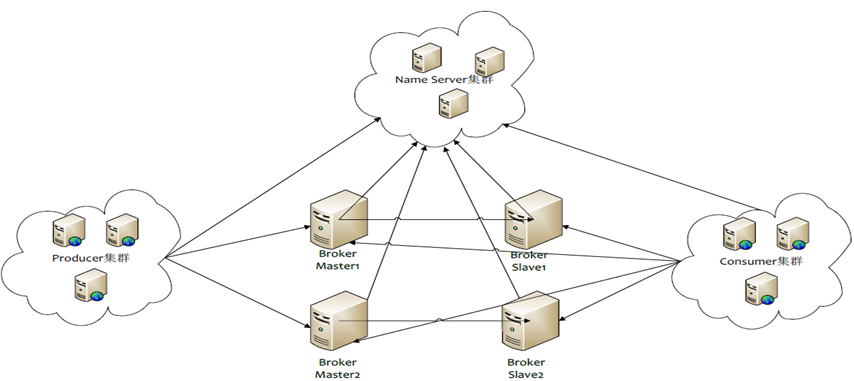
**RocketMQ核心源码包及功能说明**

如下图所示，我们要带小伙伴们一起来看看RocketMQ源码包的组成，这样更方便我们日后对他有一个深入的学习。

* rocketmq-broker 主要的业务逻辑，消息收发，主从同步, pagecache
* rocketmq-client 客户端接口，比如生产者和消费者
* rocketmq-common 公用数据结构等等
* rocketmq-distribution 编译模块，编译输出等
* rocketmq-example 示例，比如生产者和消费者
* rocketmq-fliter 进行Broker过滤的不感兴趣的消息传输，减小带宽压力
* rocketmq-logappender、rocketmq-logging日志相关
* rocketmq-namesrv Namesrv服务，用于服务协调
* rocketmq-openmessaging 对外提供服务
* rocketmq-remoting 远程调用接口，封装Netty底层通信
* rocketmq-srvutil 提供一些公用的工具方法，比如解析命令行参数
* rocketmq-store 消息存储核心包
* rocketmq-test 提供一些测试代码包
* rocketmq-tools 管理工具，比如有名的mqadmin工具

**集群架构模型**

RocketMQ为我们提供了丰富的集群架构模型，包括单点模式、主从模式、双主模式、以及生产上使用最多的双主双从模式（或者说多主多从模式），在这里我们仅介绍一下经典的双主双从集群模型，如下图所示：  


* Producer集群就是生产者集群（他们在同一个生产者组 Producer Group）
* Consumer集群就是消费者集群（他们在同一个消费者组 Consumer Group）
* NameServer集群作为超轻量级的配置中心，只做集群元数据存储和心跳工作，不必保障节点间数据强一致性，也就是说NameServer集群是一个多机热备的概念。
* 对于Broker而言，通常Master与Slave为一组服务，他们互为主从节点，通过NameServer与外部的Client端暴露统一的集群入口。Broker就是消息存储的核心MQ服务了。

**集群架构思考**

RocketMQ作为国内顶级的消息中间件，其性能主要依赖于天然的分布式Topic/Queue，并且其内存与磁盘都会存储消息数据，借鉴了Kafka的 “空中接力” 概念（这个我们后面学习Kafka的时候会详细的说明），所谓 “空中接力” 就是指数据不一定要落地，RocketMQ提供了同步/异步双写、同步/异步复制的特性。在真正的生产环境中应该选择符合自己业务的配置。下面针对于RocketMQ的高性能及其瓶颈在这里加以说明：

* 架构思考：
  + RocketMQ目前本人在公司内部实际生产环境采用8M-8S的集群架构（8主8从）硬件单点Master为32C，96G内存，500G的SSD
  + 其主要瓶颈最终会落在IOPS上面，当高峰期来临的时候，磁盘读写能力是主要的性能瓶颈，每秒收发消息IOPS达到10W+ 消息，这也是公司内部主要的可靠性消息中间件
  + 在很多时候，我们的业务会有一些非核心的消息投递，后续会进行消息中间件的业务拆分，把不重要的消息（可以允许消息丢失、非可靠性投递的消息）采用Kafka的异步发送机制，借助Kafka强大的吞吐量和消息堆积能力来做业务的分流（当然RocketMQ的性能也足够好）。
  + 为什么瓶颈在IOPS? 根本原因还是因为云环境导致的问题，云环境的SSD物理存储显然和自建机房SSD会有不小的差距，这一点我们无论是从数据库的磁盘性能、还是搜索服务（ElasticSearch）的磁盘性能，都能给出准确的瓶颈点，单机IOPS达到1万左右就是云存储SSD的性能瓶颈，这个也解释了 “木桶短板原理” 的效应，在真正的生产中，CPU的工作主要在等待IO操作，高并发下 CPU资源接近极限，但是IOPS还是达不到我们想要的效果。

**本节知识点回顾**

Hi，小伙伴们，本节课我们通过简要的图文学习，带大家快速的过了一下RocketMQ，那么小伙伴们记住一定要在脑海里建立知识的结构体系，并串联起来！无论是现在，还是说未来，本神都希望小伙伴要按照下面的步骤进行回忆和复习：

1. RocketMQ的前世今生 ？
2. RocketMQ的专业术语？
3. RocketMQ源码包的组成？
4. RocketMQ的集群架构模型
5. RocketMQ在真正生产环境中面临的瓶颈点以及解决方案