面试阿里？如果对别人开源的 RocketMQ 了如指掌，岂不是很加分！

原创：原子弹大侠 狸猫技术窝 4月23日

还没关注？伸出中指点这里！

聊技术、论职场！

为IT人打造一个“有温度”的**狸猫技术窝**

作者：***原子弹大侠***

狸猫技术窝 特约作者，阿里 P8 高级技术专家

（1）RocketMQ整体架构

如今阿里的开源项目越来越多，比如消息中间件领域的RocketMQ，分布式事务领域的Fescar，熔断限流领域的Sentinal，微服务领域的Dubbo、Nacos等等。

而现在越来越多的中小型公司也开始使用阿里开源的各种技术到自己的系统，因此有必要对阿里开源的一些技术的核心工作原理进行了解。

本文就对消息中间件领域的 RocketMQ 进行原理的分析。

首先，RocketMQ整体架构中包含了4种角色：

- NameServer
- Broker
- Producer

- Consumer

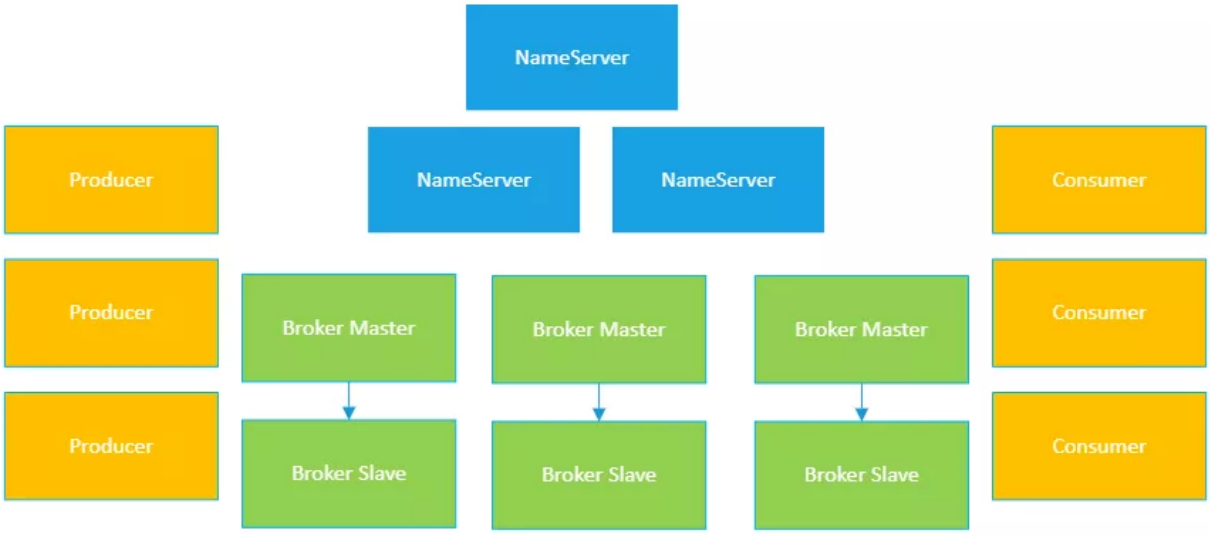
其中NameServer相当于集群管理服务，主要用于管理整个集群的元数据以及对集群进行管理的。

Broker相当于单个节点负责数据的读写的，而Producer就是负责生产消息后写入Broker的，Consumer则是从Broker来消费消息。

而且Broker在部署的时候，一般是分为主备角色的。也就是说，每个Broker都部署一个Master角色在一台机器上，然后另外会部署一个Slave角色在另外一台机器上。

这样如果某个Broker Master挂了，他的Broker Slave就可以接替继续工作，保证了高可用。

下图阐述了RocketMQ的整体架构原理：

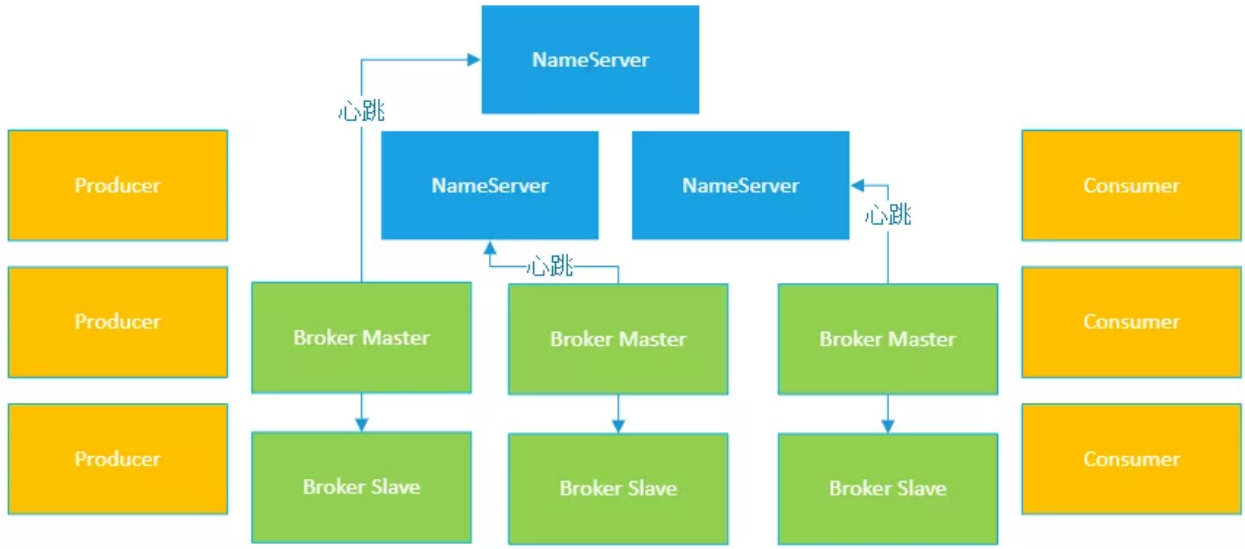


（2）RocketMQ 工作流程

首先需要先启动NameServer，接着启动Broker

Broker启动之后就会跟NameServer建立长连接，每隔一段时间发送心跳包过去

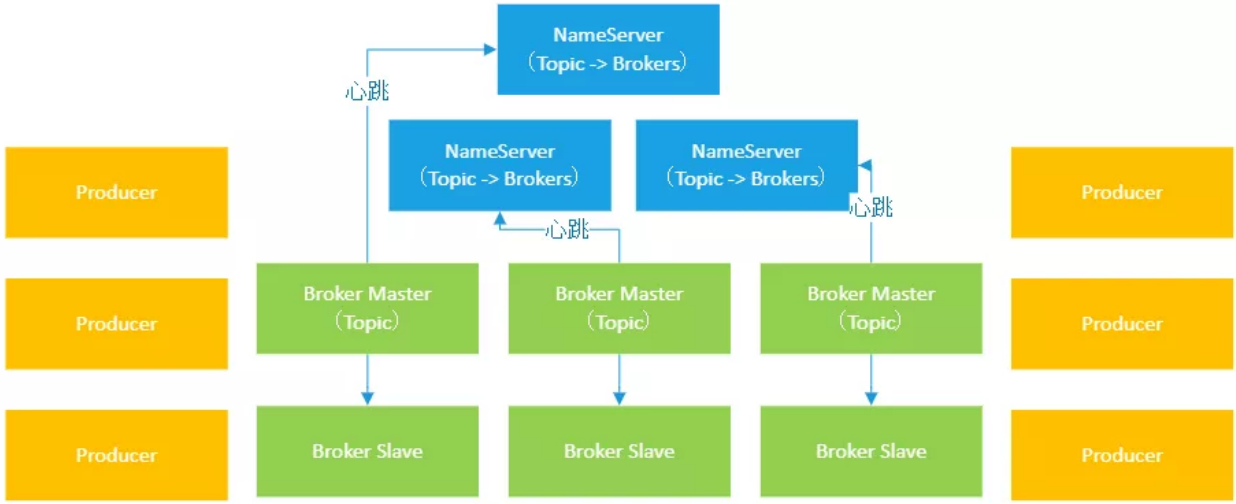
心跳包里需要包含自己当前存储的数据信息，让NameServer感知到各个Broker的情况，如下图：



然后呢？你可以创建 Topic，创建 Topic 的时候就会决定这个 Topic 的数据会放在哪些Broker上。

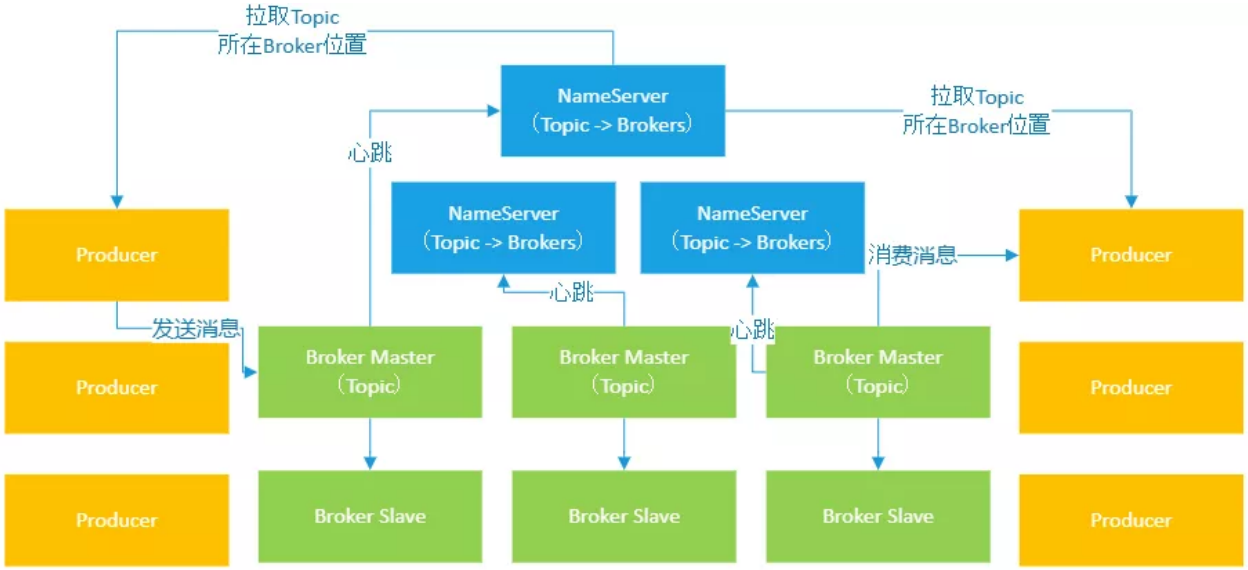
每个Broker在进行心跳的时候，就可以上报自己当前存储的Topic相关的数据信息给NameServer知道了。

而NameServer就保存了整个集群的元数据，知道每个Topic当前数据保存在哪些Broker上，如下图：



Producer在生产消息到Topic的时候，先得找NameServer问一下Topic存放在哪些Broker上，然后就可以跟那些Broker建立连接发送消息过去了。

Consumer要从Topic消费消息，也会找NameServer问一下可以从哪些Broker上消费消息，接着同样会跟Broker建立连接并且开始消费消息，如下图。



（3）NameServer原理分析

NameServer一般是集群化部署，各个机器上部署的NameServer会互相同步收到的元数据，保证各个NameServer上包含的集群元数据都是相同的。这样的架构可以保证即使任何一个NameServer宕机了也不要紧。

NameServer主要的责任就是管理Broker的心跳，如果有人一段时间内没心跳，那么就认为那个Broker已经宕机了，此时要触发对应的Slave切换为Master保证高可用。

此外，NameServer还要负责管理和维护集群元数据，比如有哪些Topic，每个Topic在哪些Broker上。

NameServer承受的请求压力是很小的，因为心跳和拉取元数据，其实是很低频的行为，所以一般**部署两三台机器**就足够了。

但是需要注意的是 ，如果集群里维护了几十万个，甚至几百万个Topic，会导致Broker每次心跳上报Topic数据时，带上几十万个Topic的数据信息，可能多达几十MB，那会导致心跳时网络负载过重。

（4）分布式架构分析

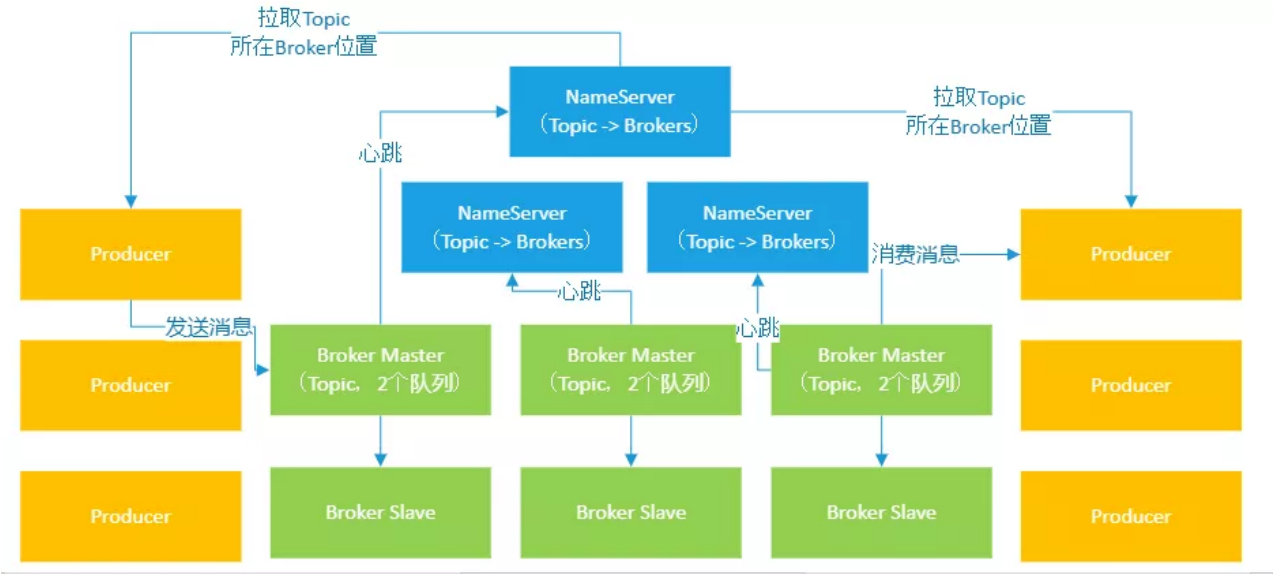
再说说Topic，每个Topic都由很多个队列组成，一个队列代表了这个Topic的部分数据，然后这个Topic的多个队列会均匀分散在多个Broker上。

比如说Topic有6个队列，对应有三台Broker，那么每个Broker上就可以放2个队列的数据。

这样就可以做到每个Topic的数据都可以分布式存储在集群中的多台Broker上，形成了分布式的存储架构，这样就可以利用多台机器的资源来存储数据。

同时，可以利用多台机器的资源来应对高并发的请求，因为对一个Topic的读写请求，就会均匀的落到多台Broker机器上去了。

如下图所示，可以看到数据分布式存储的架构。



（5）Broker原理分析

Broker主要就是两点：负责接收Producer写入的消息，同时提供消息给Consumer来读取。

写入消息时，Broker可以实现高并发高性能的写入。

这个主要采用的是对磁盘文件进行顺序写的方式，磁盘顺序写的性能是很高的，几乎跟内存随机读写的性能差不多。

另外，就是优先把数据写入os page cahce，也就是操作系统管理的缓存中。

写操作系统的内存缓存，那性能就更高了，不需要每次都直接写入物理磁盘中。

读取消息时，Broker也是主要基于os page cahce读取消息，也就是优先从操作系统管理的内存缓存中读取数据，提供给Consumer来消费。因为是优先读内存，所以同样性能也是很高的。

通过这种架构，Broker就可以保证高并发以及高性能。

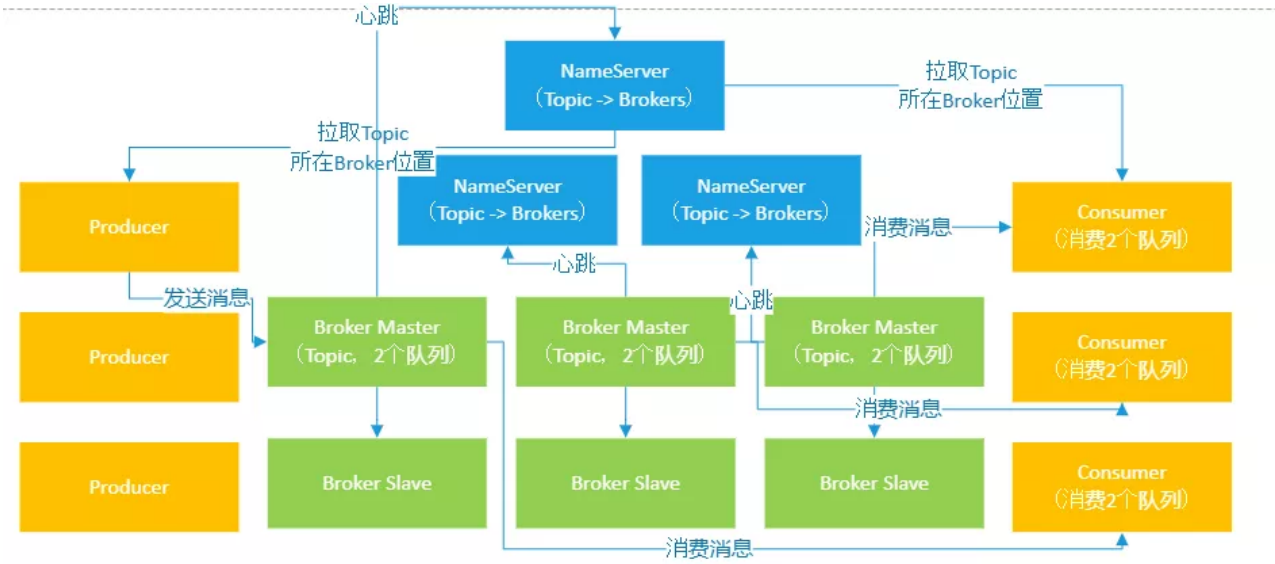
另外，Broker还实现了高可用的架构。也就是说每个Broker Master都会把自己接收到的数据同步给Broker Slave，实现每个Broker上的数据备份。

如果Broker Master宕机了，会切换Broker Slave对外提供服务，保证了高可用的架构。

（6）Consumer原理分析

如果我们部署了Consumer在多台机器上，默认也会把Topic下的多个队列的数据均匀的分配给各个Consumer实例来进行消费。

比如Topic下一共6个队列，假设部署了3个Consumer实例，此时每个Consumer实例就会消费2个队列的数据，如下图：



（7）总结

对于任何一个消息系统，对他的原理进行研究的时候，都应该关注几点：

- 集群架构（元数据如何管理、集群如何管理）
- 数据模型（Topic、队列等概念）
- 分布式架构（消息如何分布式存储）
- 消息写入以及读取的原理
- 高可用架构（如何进行数据冗余）
- 客户端原理

把这些弄明白之后，才算对一个消息系统有了一定的了解，才能开始尝试把这个消息系统应用于自己公司的生产项目中。

END

作者简介：

原子弹大侠，阿里 P8 高级技术专家

经历过每日百亿流量的互联网系统架构，尤其对上亿用户场景下的高并发系统架构设计以及性能优化相关领域有深入的研究。

阅读原文