

[首页](#) [资讯](#) [精华](#) [论坛](#) [问答](#) [博客](#) [专栏](#) [群组](#) [更多](#) ▼
[您还未登录！](#) [登录](#) [注册](#)

大步流星

- [博客](#)
- [微博](#)
- [相册](#)
- [收藏](#)
- [留言](#)
- [关于我](#)



[RocketMQ初探一：NameServer的作用](#)

博客分类：

- [RocketMQ](#)

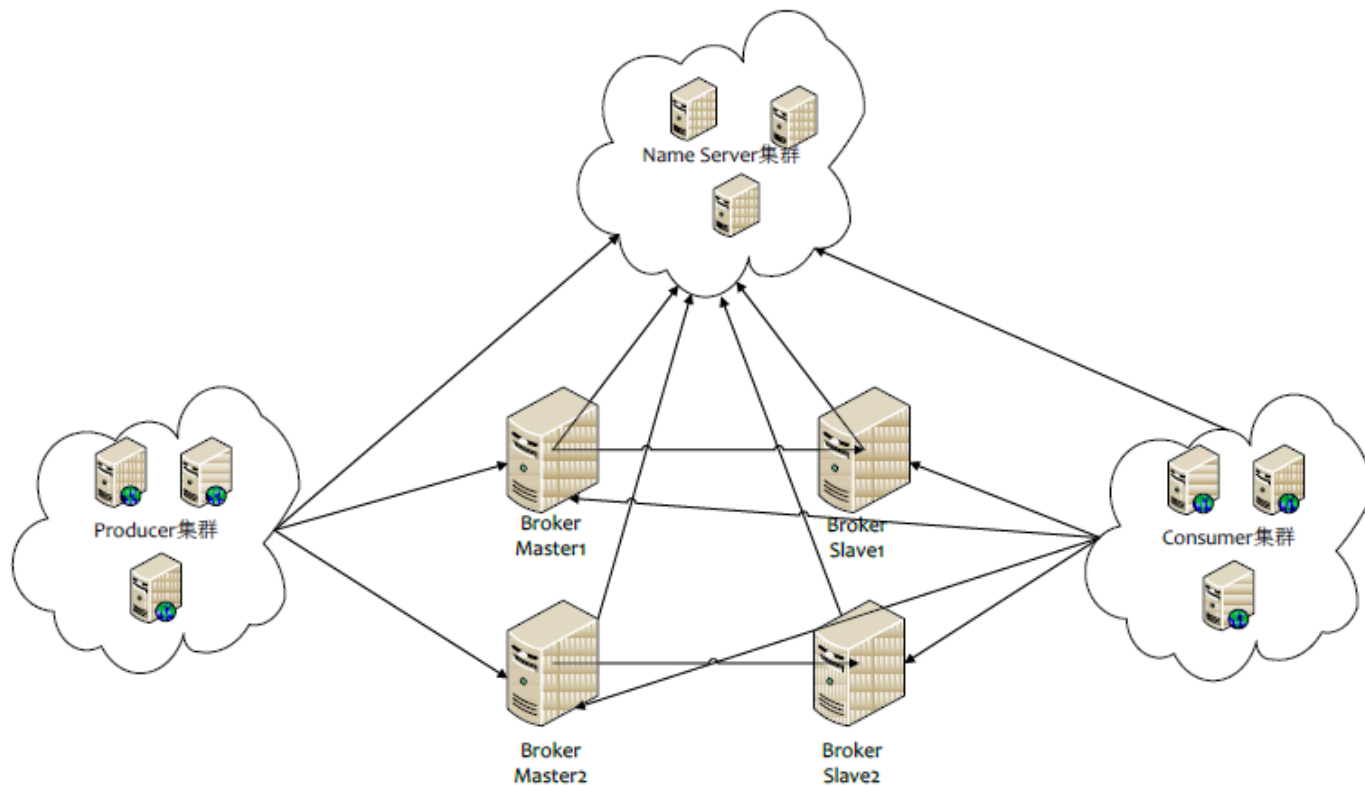
[RocketMQ初探rocket_nameservernameserver](#)

第一次真正接触Java消息服务是在2013年底，当时是给中国移动做统一支付平台，当时用的就是著名的Apache ActiveMQ，当时觉得很有趣，一个服务队列竟然可以玩出这么多花样来。当时为了尽快的入门，还把《Java Message Service》给看了一遍，这对于初学者的我收获颇多。我说知道的完全实现JMS规范的MOM有ActiveMQ/Apollo和HornetQ，都是采用Java实现。JMS中说明了Java消息服务的两种消息传送模型，即P2P（点对点）和Pub/Sub（发布订阅），在约定了一些消息服务特性的同时，并提供了一套接口API，是否实现了该API，标志着MOM是否支持JMS规范，JMS规范中定义了消息服务诸多特性，这些特性和他所面对的企业级服务场景相关，当然，这也严重限制了消息服务的吞吐量，完全实现JMS规范的MOM，性能总不会太高，而且JMS规范中没有涉及消息服务的分布式特性，导致大多数实现JMS规范的MOM分布式部署功能比较弱，只适合集群部署。

说到高性能消息中间件，第一个想到的肯定是LinkedIn开源的Kafka，虽然最初Kafka是为日志传输而生，但也非常适合互联网公司消息服务的应用场景，他们不要求数据实时的强一致性（事务），更多是希望达到数据的最终一致性。RocketMQ是MetaQ的3.0版本，而MetaQ最初的设计又参考了Kafka。最初的MetaQ 1.x版本由阿里的原作者庄晓丹开发，后面的MetaQ 2.x版本才进行了开源，这里需要注意一点的事，MetaQ 1.x和MetaQ 2.x是依赖ZooKeeper的，但RocketMQ（即MetaQ 3.x）却去掉了ZooKeeper依赖，转而采用自己的NameServer。

ZooKeeper是著名的分布式协作框架，提供了Master选举、分布式锁、数据的发布和订阅等诸多功能，为什么RocketMQ没有选择ZooKeeper，而是自己开发了NameServer，我们来具体看看NameServer在RocketMQ集群中的作用就明了了。

RocketMQ的Broker有三种集群部署方式：1.单台Master部署；2.多台Master部署；3.多Master多Slave部署；采用第3种部署方式时，Master和Slave可以采用同步复制和异步复制两种方式。下图是第3种部署方式的简单图：



图虽然是网上找的，但也足以说明问题，当采用多Master方式时，Master与Master之间是不需要知道彼此的，这样的设计直接降低了Broker实现的复查性，你可以试想，如果Master与Master之间需要知道彼此的存在，这会需要在Master之中维护一个网络的Master列表，而且必然设计到Master发现和活跃Master数量变更等诸多状态更新问题，所以最简单也最可靠的做法就是Master只做好自己的事情（比如和Slave进行数据同步）即可，这样，在分布式环境中，某台Master宕机或上线，不会对其他Master造成任何影响。

那么怎么才能知道网络中有多少台Master和Slave呢？你会很自然想到用ZooKeeper，每个活跃的Master或Slave都去约定的ZooKeeper节点下注册一个状态节点，但RocketMQ没有使用ZooKeeper，所以这件事就交给了NameServer来做了（看上图）。

结论一：NameServer用来保存活跃的broker列表，包括Master和Slave。

当然，这个结论百度一查就知道，我们移步到rocketmq-namesrv模块中最重要的一个类：RouteInfoManager，它的主要属性如下：

```
private final ReadWriteLock lock = new ReentrantReadWriteLock();
private final HashMap<String/* topic */, List<QueueData>> topicQueueTable;
private final HashMap<String/* brokerName */, BrokerData> brokerAddrTable;
private final HashMap<String/* clusterName */, Set<String/* brokerName */>>
clusterAddrTable;
private final HashMap<String/* brokerAddr */, BrokerLiveInfo> brokerLiveTable;
private final HashMap<String/* brokerAddr */, List<String/* Filter Server */>
filterServerTable;
```

每个属性通过名字就能清楚的知道是什么意思，之所以能用非线程安全的HashMap，是因为有读写锁lock来对HashMap的修改做保护。我们注意到保存broker的Map有两个，即brokerAddrTable用来保存所有的broker列表和brokerLiveTable用来保存当前活跃的broker列表，而BrokerData用来保存broker的主要新增，而BrokerLiveInfo只用来保存上次更新（心跳）时间，我们可以直接看看RouteInfoManager中扫描非活跃broker的方法：

```
// Broker Channel两分钟过期
private final static long BrokerChannelExpiredTime = 1000 * 60 * 2;
public void scanNotActiveBroker() {
    Iterator<Entry<String, BrokerLiveInfo>> it =
```

```

this.brokerLiveTable.entrySet().iterator();
while (it.hasNext()) {
    Entry<String, BrokerLiveInfo> next = it.next();
    long last = next.getValue().getLastUpdateTimestamp();
    if ((last + BrokerChannelExpiredTime) < System.currentTimeMillis()) {
        RemotingUtil.closeChannel(next.getValue().getChannel());
        it.remove();
        Log.warn("The broker channel expired, {} {}ms", next.getKey(),
BrokerChannelExpiredTime);
        this.onChannelDestroy(next.getKey(), next.getValue().getChannel());
    }
}
}
}

```

可以看出，如果两分钟内都没收到一个broker的心跳数据，则直接将其从**brokerLiveTable**中移除，注意，这还会导致该broker从**brokerAddrTable**被删除，当然，如果该broker是Master，则它的所有Slave的broker都将被删除。具体细节可以参看RouteInfoManager的onChannelDestroy方法。

结论二：NameServer用来保存所有topic和该topic所有队列的列表。

我们注意到，**topicQueueTable**的value是QueueData的List，我们看看QueueData中的属性：

```

private String brokerName; // broker的名称
private int readQueueNums; // 读队列数量
private int writeQueueNums; // 写队列数量
private int perm; // 读写权限
private int topicSynFlag; // 同步复制还是异步复制标记

```

所以，你几乎可以在NameServer这里知道topic相关的所有信息，包括topic有哪些队列，这些队列在那些broker上等。

结论三：NameServer用来保存所有broker的Filter列表。

关于这一点，讨论broker的时候再细说。

DefaultRequestProcessor是NameServer的默认请求处理器，他处理了定义在rocketmq-common模块中RequestCode定义的部分请求，比如注册broker、注销broker、获取topic路由、删除topic、获取broker的topic权限、获取NameServer的所有topic等。

在源代码中，NettyServerConfig类记录NameServer中的一些默认参数，比如端口、服务端线程数等，列出如下：

```

private int listenPort = 8888;
private int serverWorkerThreads = 8;
private int serverCallbackExecutorThreads = 0;
private int serverSelectorThreads = 3;
private int serverOnewaySemaphoreValue = 256;
private int serverAsyncSemaphoreValue = 64;
private int serverChannelMaxIdleTimeSeconds = 120;



```

这些都可以启动时指定配置文件来进行覆盖修改，具体可以参考NameServer的启动类NamesrvStartup的实现（没想到Apache还有对命令行提供支持的commons-cls的包）。

现在再回过头来看看RocketMQ为什么不使用ZooKeeper？ZooKeeper可以提供Master选举功能，比如Kafka用来给每个分区选一个broker作为leader，但对于RocketMQ来说，topic的数据在每个Master上是对等的，没有哪个Master上有topic上的全部数据，所以这里选举leader没有意义；RocketMQ集群中，需要有构件来处理一些通用数据，比如broker列表，broker刷新时间，虽然ZooKeeper也能存放数据，并有一致性保证，但处理数据之间的一些逻辑关系却比较麻烦，而且数据的逻辑解析操作得交给ZooKeeper客户端来做，如果有多种角色的客户端存在，自己解析多级数据确实是个麻烦事情；既然RocketMQ集群中没有用到ZooKeeper的一些重量级的功能，只是使用ZooKeeper的数据一致性和发布订阅的话，与其依赖重量级的ZooKeeper，还不如写个轻量级的NameServer，NameServer也可以集群部署，NameServer与NameServer之间无任何信息同步，只有一千多行代

码的NameServer稳定性肯定高于ZooKeeper，占用的系统资源也可以忽略不计，何乐而不为？当然，这些只是本人的一点理解，具体原因当然得RocketMQ设计和开发者来说。

- [查看图片附件](#)

分享到:  

[《将博客搬至CSDN》](#) | [Dubbo源代码实现二：服务调用的动态代理和 ...](#)

- 2016-08-14 11:25
- 浏览 113
- [评论\(1\)](#)
- 分类: [开源软件](#)
- [相关推荐](#)

参考知识库



[Hive知识库](#) 543 关注 | 200 收录



[Objective-C知识库](#) 493 关注 | 704 收录



[区块链知识库](#) 438 关注 | 89 收录



[深度学习知识库](#) 2302 关注 | 267 收录

评论

1 楼 [街头诗人](#) 2016-09-14

持续跟踪关注博主关于MQ的动态 😊


发表评论




[您还没有登录, 请您登录后再发表评论](#)



manzhizhen

- 浏览: 21974 次
- 性别: 

- 来自：杭州
-  我现在离线

最近访客

[更多访客>>](#)



[WZ Freedom](#)



[意不意外开不开心](#)



[javaFlood](#)



[maen 1111](#)

文章分类

- [全部博客 \(34\)](#)
- [GEF \(3\)](#)
- [Java \(6\)](#)
- [RCP \(2\)](#)
- [转载 \(3\)](#)
- [Spring \(2\)](#)
- [Web \(1\)](#)
- [ActiveMQ \(5\)](#)
- [代码 \(0\)](#)
- [Oracle \(1\)](#)
- [WebLogic \(1\)](#)
- [Git \(0\)](#)
- [Quartz \(0\)](#)
- [HttpClient \(1\)](#)
- [Dubbo \(2\)](#)
- [软件架构 \(0\)](#)
- [Flume \(5\)](#)
- [水电费 \(0\)](#)
- [RocketMQ \(1\)](#)

社区版块

- [我的资讯 \(1\)](#)
- [我的论坛 \(0\)](#)
- [我的问答 \(1\)](#)

存档分类

- [2016-09 \(1\)](#)
- [2016-08 \(1\)](#)
- [2016-07 \(3\)](#)
- [更多存档...](#)

评论排行榜

- [RocketMQ初探一：NameServer的作用](#)

最新评论

- [街头诗人](#): 持续跟踪关注博主关于MQ的动态
[RocketMQ初探一：NameServer的作用](#)
- [xiajunhust](#): LZ写的比较清楚，看来是读过源码的。赞一个！
[ActiveMQ中消费者是如何接收消息的（一）](#)
- [yuenkin](#): 说好的三呢？
[ActiveMQ中消费者是如何接收消息的（二）](#)
- [manzhizhen](#): 加你微信了
[ActiveMQ中消费者是如何接收消息的（二）](#)
- [aliahhqcheng](#): blog写的很棒。请教一个问题：生产者-->queue- ...
[ActiveMQ中消费者是如何接收消息的（二）](#)

声明：ITeye文章版权属于作者，受法律保护。没有作者书面许可不得转载。若作者同意转载，必须以超链接形式标明文章原始出处和作者。

© 2003-2016 ITeye.com. All rights reserved. [京ICP证110151号 京公网安备110105010620]