# 1. RocketMq是什么

一个纯Java、分布式队列模型的消息中间件，具有高可用、高可靠、高实时、低延迟的特点。（记住这句就行了）

# 2. RocketMq有什么功能

先答出消息中间件的通用功能

- 1、业务解耦：这也是发布订阅的消息模型。生产者发送指令到MQ中，然后下游订阅这类指令的消费者会收到这个指令执行相应的逻辑，整个过程与具体业务无关，抽象成了一个发送指令，存储指令，消费指令的过程。
- 2、前端削峰：前端发起的请求在短时间内太多后端无法处理，可以堆积在MQ中，后端按照一定的顺序处理，秒杀系统就是这么实现的。

再说说RocketMq的特点：

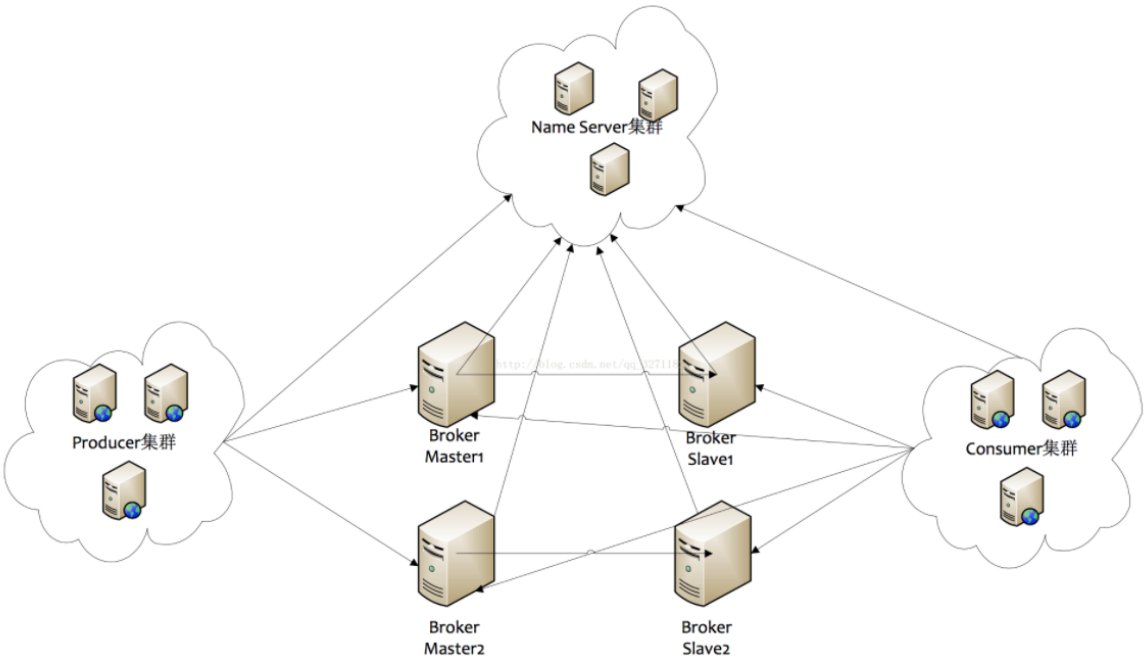
- 3、亿级消息的堆积能力，单个队列中的百万级消息的累积容量。
- 4、高可用性：Broker服务器支持多Master多Slave的同步双写以及Master多Slave的异步复制模式，其中同步双写可保证消息不丢失。
- 5、高可靠性：生产者将消息发送到Broker端有三种方式，同步、异步和单向，其中同步和异步都可以保证消息成功的成功发送。Broker在对于消息刷盘有两种策略：同步刷盘和异步刷盘，其中同步刷盘可以保证消息成功的存储到磁盘中。消费者的消费模式也有集群消费和广播消费两种，默认集群消费，如果集群模式中消费者挂了，一个组里的其他消费者会接替其消费。综上所述，是高可靠的。
- 6、支持分布式事务消息：这里是采用半消息确认和消息回查机制来保证分布式事务消息的，下面会详细描述。
- 7、支持消息过滤：建议采用消费者业务端的tag过滤
- 8、支持顺序消息：消息在Broker中是采用队列的FIFO模式存储的，也就是发送是顺序的，只要保证消费的顺序性即可。

- 9、支持定时消息和延迟消息：Broker中由定时消息的机制，消息发送到Broker中，不会立即被Consumer消费，会等到一定的时间才被消费。延迟消息也是一样，延迟一定时间之后才会被Consumer消费。

能说出这么多已经不错了😁

# 3. RoctetMq的架构

回到最开始的问题，RocketMq的原理是什么，也就是怎么实现的，先看图




RocketMq一共有四个部分组成：NameServer，Broker，Producer生产者，Consumer消费者，每一部分都是集群部署的。

## NameServer

NameServer是一个无状态的服务器，角色类似于Dubbo的Zookeeper，但比Zookeeper更轻量。
特点：

- 每个NameServer结点之间是相互独立，彼此没有任何信息交互。
- Nameserver被设计成几乎是无状态的，通过部署多个结点来标识自己是一个伪集群，Producer在发送消息前从NameServer中获取Topic的路由信息也就是发往哪个Broker,Consumer也会定时从NameServer获取topic的路由信息，Broker在启动时会向NameServer注册，并定时进行心跳连接，且定时同步维护的Topic到NameServer。
  功能主要有两个：
- 1、跟Broker结点保持长连接。
- 2、维护Topic的路由信息。

## Broker

消息存储和中转角色，负责存储和转发消息。

- Broker内部维护着一个个Message Queue，用来存储消息的索引，真正存储消息的地方是CommitLog（日志文件）。
- 单个Broker与所有的Nameserver保持着长连接和心跳，并会定时将Topic信息同步到NameServer，和NameServer的通信底层是通过Netty实现的。

## Producer

消息生产者，业务端负责发送消息，由用户自行实现和分布式部署。

### Producer的负载均衡

Producer的负载均衡是由MQFaultStratege.selectOneMessageQueue()来实现的。这个方法就是随机选择一个要发送消息的broker来达到负载均衡的效果，选择的标准：尽量不选刚刚选过的broker，尽量不选发送上条消息延迟过高或没有响应的broker，也就是找到一个可用的broker。（源码不贴了）

### 发送的三种策略

Producer发送消息有三种方式：同步、异步和单向

- 同步：同步发送是指发送方发出数据后等待接收方发回响应后在发送下一个数据包。一般用于重要的消息通知，如重要的通知邮件或者营销短信等。
- 异步：异步发送是指发送方发出数据后不等接收方发回响应就发出下一个数据包。一般用于可能链路耗时较长而对响应时间比较敏感的场景。如视频上传后通知启动转码服务。
- 单向：单向发送是指只负责发送消息而不等待接收方发送响应且没有回调函数，适合那些耗时比较短且对可靠性要求不高的场景，例如日志收集。

## Consumer

消息消费者，负责消费消息，由用户自行实现并进行集群部署。

### 推拉消费模式

- PULL：拉取型消费者主动从broker中拉取消息消费，只要拉取到消息，就会启动消费过程，称为主动型消费。
- PUSH：推送型消费者就是要注册消息的监听器，监听器是要用户自行实现的。当消息达到broker服务器后，会触发监听器拉取消息，然后启动消费过程。但是从实际上看还是从broker中拉取消息，称为被动消费型。

### 集群还是广播

看业务需求，默认是集群消费。

- 集群消费：broker中的一条消息会发送给订阅这个topic的一个消费组里的唯一一个消费者进行消费。如果这个消费者挂掉了，组里的其他消费者会接替它进行消费。
- 广播消费：broker中的一条消息会发送给订阅这个topic的一个消费组里的每一个消费者进行消费。

### Consumer的负载均衡

- Consumer的负载均衡是指将MessageQueue中的消息队列分配到消费者组里的具体消费者。
- Consumer在启动的时候会实例化rebalanceImpl，这个类负责消费端的负载均衡。通过rebalanceImpl调用allocateMesasgeQueueStratage.allocate()完成负载均衡。
- 每次有新的消费者加入到组中就会重新做一下分配。每10秒自动做一次负载均衡。

# 4. RocketMq消息模型（专业术语）

初学者可以了解下。

## Message

就是要传输的消息，一个消息必须有一个主题，一条消息也可以有一个可选的Tag（标签）和额外的键值对，可以用来设置一个业务的key，便于开发中在broker服务端查找消息。

### Topic

主题，是消息的第一级类型，每条消息都有一个主题，就像信件邮寄的地址一样。主题就是我们具体的业务，比如一个电商系统可以有订单消息，商品消息，采购消息，交易消息等。Topic和生产者和消费者的关系非常松散，生产者和Topic可以是1对多，多对1或者多对多，消费者也是这样。

### Tag

标签，是消息的第二级类型，可以作为某一类业务下面的二级业务区分，它的主要用途是在消费端的消息过滤。比如采购消息分为采购创建消息，采购审核消息，采购推送消息，采购入库消息，采购作废消息等，这些消息是同一Topic和不同的Tag，当消费端只需要采购入库消息时就可以用Tag来实现过滤，不是采购入库消息的tag就不处理。

### Group

组，可分为ProducerGroup生产者组合ConsumerGroup消费者组，一个组可以订阅多个Topic。一般来说，某一类相同业务的生产者和消费者放在一个组里。

### Message Queue

消息队列，一个Topic可以划分成多个消息队列。Topic只是个逻辑上的概念，消息队列是消息的物理管理单位，当发送消息的时候，Broker会轮询包含该Topic的所有消息队列，然后将消息发出去。有了消息队列，可以使得消息的存储可以分布式集群化，具有了水平的扩展能力。

### offset

是指消息队列中的offset，可以认为就是下标，消息队列可看做数组。offset是java long型，64位，理论上100年不会溢出，所以可以认为消息队列是一个长度无限的数据结构。

## 5. 核心问题

### 顺序消息

- 如何保证顺序消息?
  顺序由producer发送到broker的消息队列是满足FIFO的，所以发送是顺序的，单个queue里的消息是顺序的。多个Queue同时消费是无法绝对保证消息的有序性的。所以，同一个topic，同一个queue，发消息的时候一个线程发送消息，消费的时候一个线程去消费一个queue里的消息。
- 追问：怎么保证消息发到同一个queue里?
  RocketMQ给我们提供了MessageQueueSelector接口，可以重写里面的接口，实现自己的算法，比如判断i%2==0，那就发送消息到queue1否则发送到queue2。

### 消息过滤

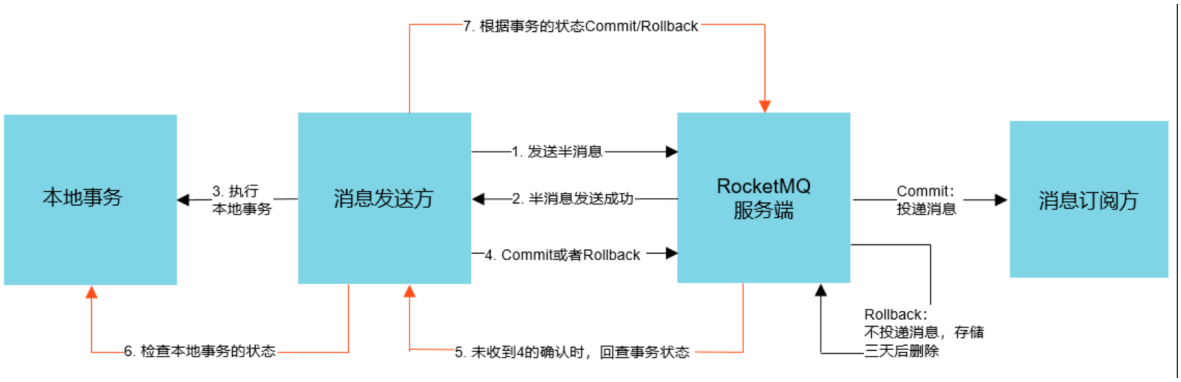
- 如何实现消息过滤?
  有两种方案，一种是在broker端按照Consumer的去重逻辑进行过滤，这样做的好处是避免了无用的消息传输到Consumer端，缺点是加重了Broker的负担，实现起来相对复杂。另一种是在Consumer端过滤，比如按照消息设置的tag去重，这样的好处是实现起来简单，缺点是有大量无用的消息到达了Consumer端只能丢弃不处理。

### 消息去重

- 如果由于网络等原因，多条重复消息投递到了Consumer端，你怎么进行消息去重？
  这个得先说下消息的幂等性原则：就是用户对于同一种操作发起的多次请求的结果是一样的，不会因为操作了多次就产生不一样的结果。只要保持幂等性，不管来多少条消息，最后处理结果都一样，需要Consumer端自行实现。
  去重的方案：因为每个消息都有一个MessageId, 保证每个消息都有一个唯一键，可以是数据库的主键或者唯一约束，也可以是Redis缓存中的键，当消费一条消息前，先检查数据库或缓存中是否存在这个唯一键，如果存在就不再处理这条消息，如果消费成功，要保证这个唯一键插入到去重表中。

## 分布式事务消息

- 你知道半消息吗？RocketMQ是怎么实现分布式事务消息的？
  半消息：是指暂时还不能被Consumer消费的消息，Producer成功发送到broker端的消息，但是此消息被标记为"暂不可投递"状态，只有等Producer端执行完本地事务后经过二次确认了之后，Consumer才能消费此条消息。




上图就是分布式事务消息的实现过程，依赖半消息，二次确认以及消息回查机制。

- 1、Producer向broker发送半消息
- 2、Producer端收到响应，消息发送成功，此时消息是半消息，标记为"不可投递"状态，Consumer消费不了。
- 3、Producer端执行本地事务。
- 4、正常情况本地事务执行完成，Producer向Broker发送Commit/Rollback，如果是Commit，Broker端将半消息标记为正常消息，Consumer可以消费，如果是Rollback，Broker丢弃此消息。
- 5、异常情况，Broker端迟迟等不到二次确认。在一定时间后，会查询所有的半消息，然后到Producer端查询半消息的执行情况。
- 6、Producer端查询本地事务的状态
- 7、根据事务的状态提交commit/rollback到broker端。（5，6，7是消息回查）

## 消息的可用性

- RocketMQ如何能保证消息的可用性/可靠性？（这个问题的另一种问法：如何保证消息不丢失）
  答案如下，要从Producer，Consumer和Broker三个方面来回答。

从Producer角度分析，如何确保消息成功发送到了Broker?

- 1、可以采用同步发送，即发送一条数据等到接受者返回响应之后再发送下一个数据包。如果返回响应OK，表示消息成功发送到了broker，状态超时或者失败都会触发二次重试。
- 2、可以采用分布式事务消息的投递方式。
- 3、如果一条消息发送之后超时，也可以通过查询日志的API，来检查是否在Broker存储成功。
  总的来说，Producer还是采用同步发送来保证的。

从Broker角度分析，如何确保消息持久化?

- 1、消息只要持久化到CommitLog（日志文件）中，即使Broker宕机，未消费的消息也能重新恢复再消费。

- 2、Broker的刷盘机制：同步刷盘和异步刷盘，不管哪种刷盘都可以保证消息一定存储在pagecache中（内存中），但是同步刷盘更可靠，它是Producer发送消息后等数据持久化到磁盘之后再返回响应给Producer。
- 3、Broker支持多Master多Slave同步双写和多Master多Slave异步复制模式，消息都是发送给Master主机，但是消费既可以从Master消费，也可以从Slave消费。同步双写模式可以保证即使Master宕机，消息肯定在Slave中有备份，保证了消息不会丢失。

从Consumer角度分析，如何保证消息被成功消费？

- Consumer自身维护了个持久化的offset（对应Message Queue里的min offset），用来标记已经成功消费且已经成功发回Broker的消息下标。如果Consumer消费失败，它会向Broker发回消费失败的状态，发回成功才会更新自己的offset。如果发回给broker时broker挂掉了，Consumer会定时重试，如果Consumer和Broker一起挂掉了，消息还在Broker端存储着，Consumer端的offset也是持久化的，重启之后继续拉取offset之前的消息进行消费。

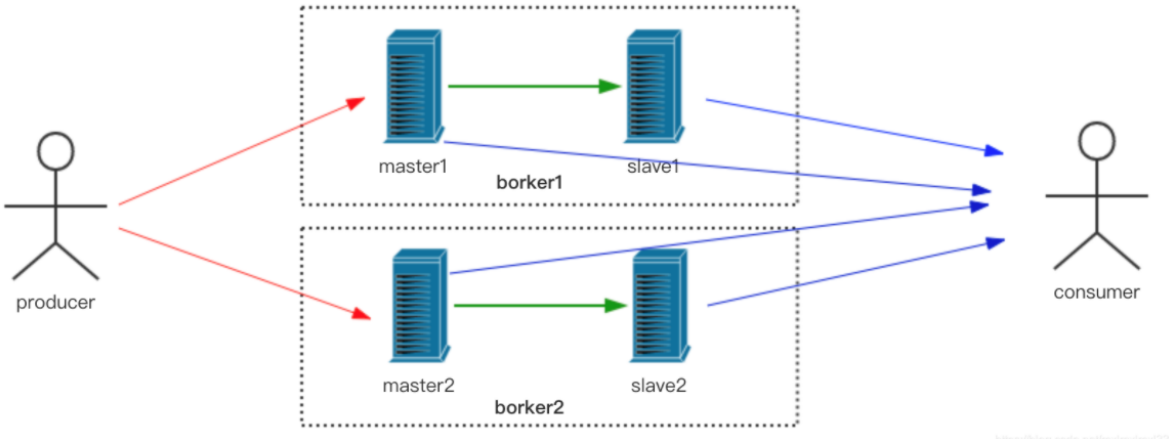
## 刷盘实现

RocketMQ提供了两种刷盘策略：同步刷盘和异步刷盘

- 同步刷盘：在消息达到Broker的内存之后，必须刷到commitLog日志文件中才算成功，然后返回Producer数据已经发送成功。
- 异步刷盘：异步刷盘是指消息达到Broker内存后就返回Producer数据已经发送成功，会唤醒一个线程去将数据持久化到CommitLog日志文件中。
  优缺点分析：同步刷盘保证了消息不丢失，但是响应时间相对异步刷盘要多出10%左右，适用于对消息可靠性要求比较高的场景。异步刷盘的吞吐量比较高，RT小，但是如果broker断电了内存中的部分数据会丢失，适用于对吞吐量要求比较高的场景。

## 负载均衡

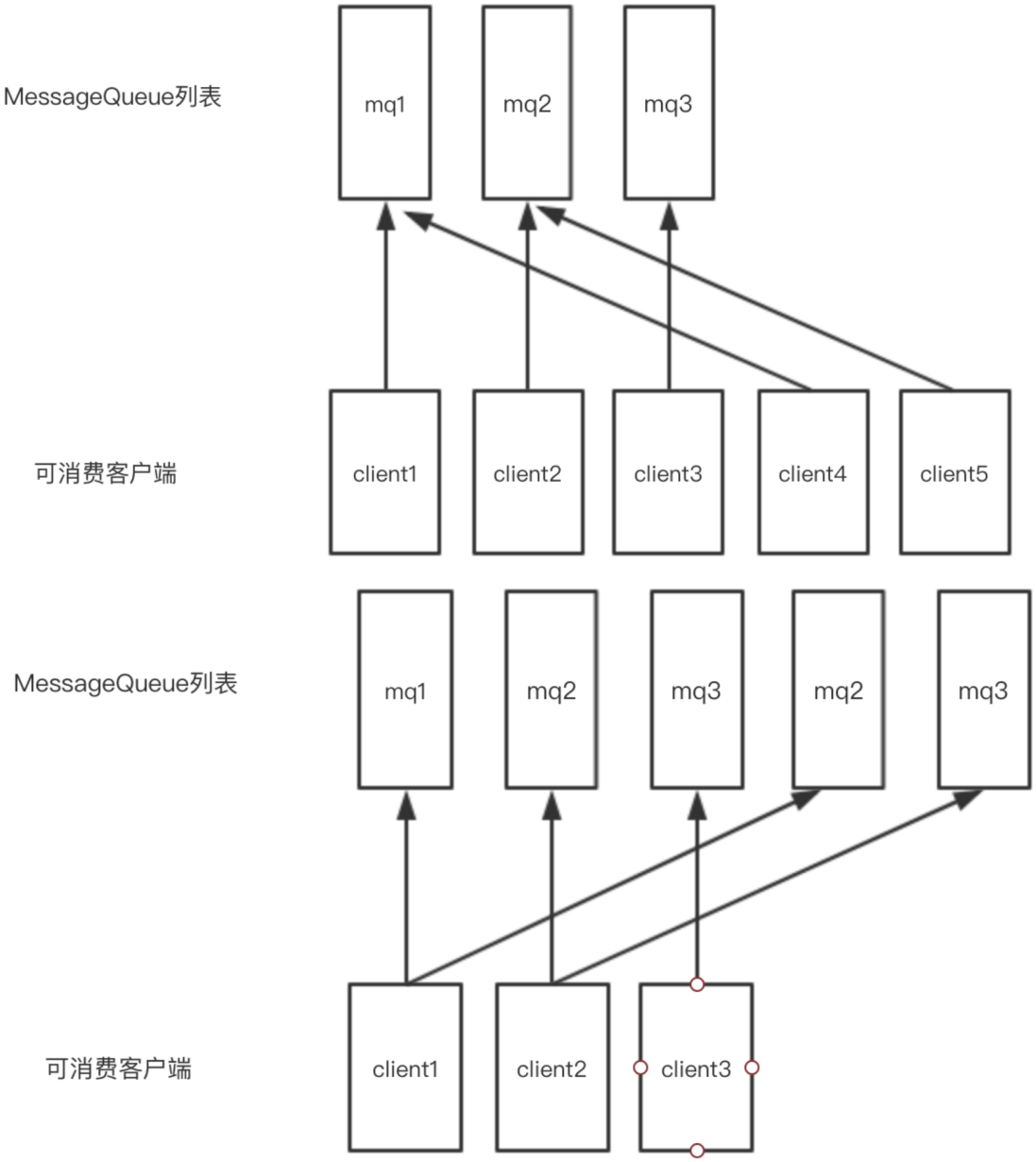
- 你说下RocketMQ的负载均衡是如何实现的?
  RocketMQ是分布式消息服务，负载均衡时再生产和消费的客户端完成的。



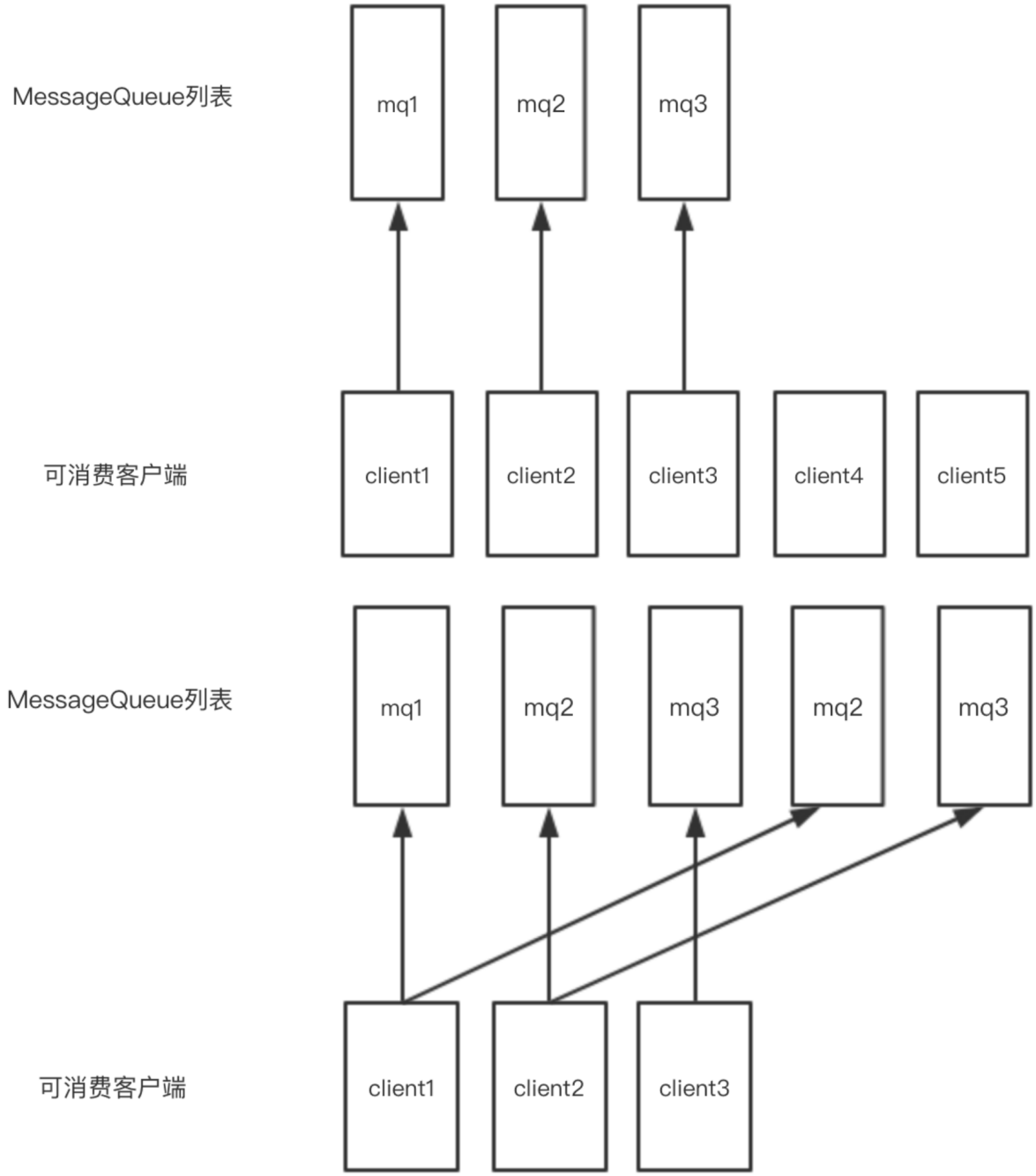

上面讲过，nameServer保存着Topic的路由信息，路由记录了broker集群节点的通讯地址，broker的名称以及读写队列数量等信息。写队列writeQueue表示生产者可以写入的队列数，如果不做配置默认为4，也就是queueId是0，1，2，3.broker收到消息后根据queueId生成消息队列，生产者负载均衡的过程的实质就是选择broker集群和queueId的过程。读队列readQueue表示broker中可以供消费者读取信息的队列个数，默认也是4个，也就是queueId也是0,1,2,3。消费者拿到路由信息后会选择queueId，从对应的broker中读取数据消费。

下面我从生产者负载均衡和消费者负载均衡两个角度来说明：

- 生产者的负载均衡：实质是在选择MessageQueue对象（内部包含了brokerName和queueId），第一种是默认策略，从MessageQueue列表中随机选择一个，算法时通过自增随机数对列表打下取余得到位置信息，但获得的MessageQueue所在集群不能是上次失败集群。第二种是超时容忍策略，先随机选择一个MessageQueue，如果因为超时等异常发送失败，会优先选择该broker集群下其他MessageQueue发送，如果没找到就从之前发送失败的Broker集群中选一个进行发送，若还没有找到才使用默认策略。
- 消费者的负载均衡：这里可选的有六种算法。 1、平均分配算法




2、环形算法

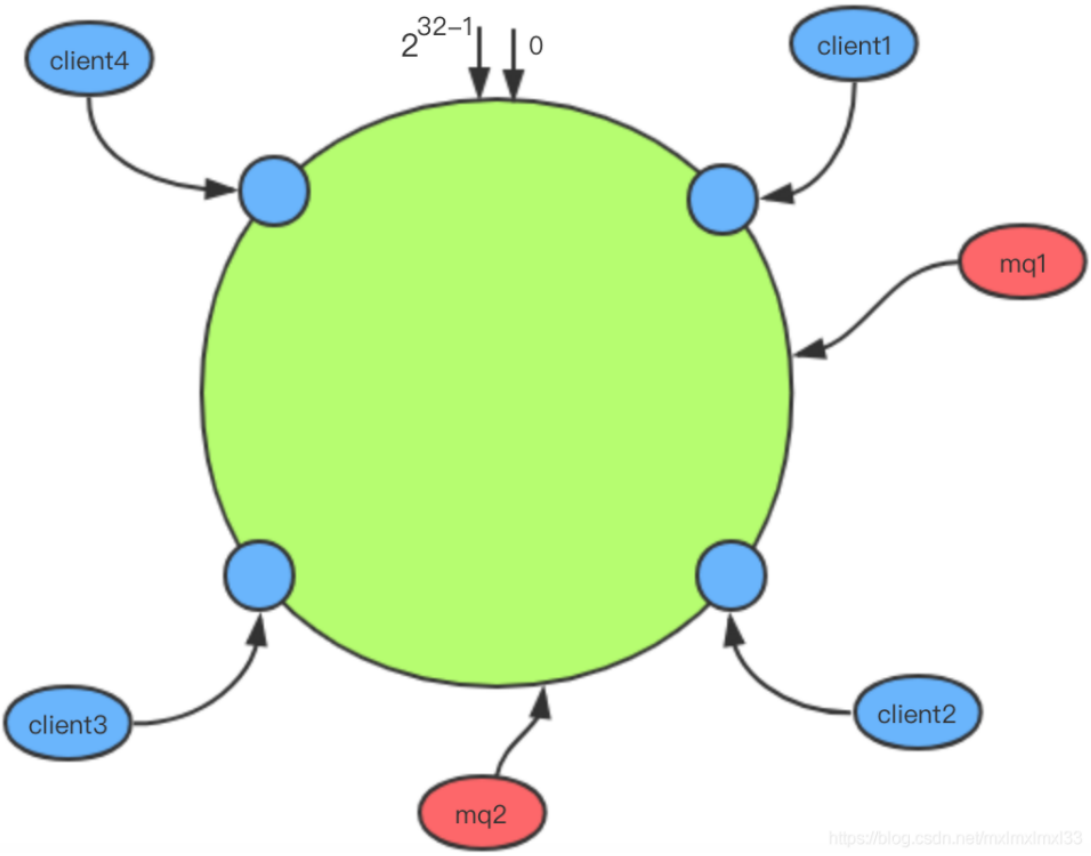



3、指定机房算法

4、就近机房算法

5、统一哈希算法

使用一致性哈希算法进行负载，每次负载都会重建一致性hash路由表，获取本地客户端负责的所有队列信息。默认的hash算法为MD5，假设有4个消费者客户端和2个消息队列mq1和mq2，通过hash后分布在hash环的不同位置，按照一致性hash的顺时针查找原则，mq1被client2消费，mq2被client3消费。




6、手动配置算法

# 6. 总结

今天主要讲了RocketMq的基本架构以及消息是怎么发送，存储和消费的，要掌握NameServer、Broker、Producer、Consumer各自的职责是什么，在消息这方面都做了什么努力，比如保证高可用、保证顺序、保证分布式、实现过滤、实现去重以及怎么做负载均衡等等。