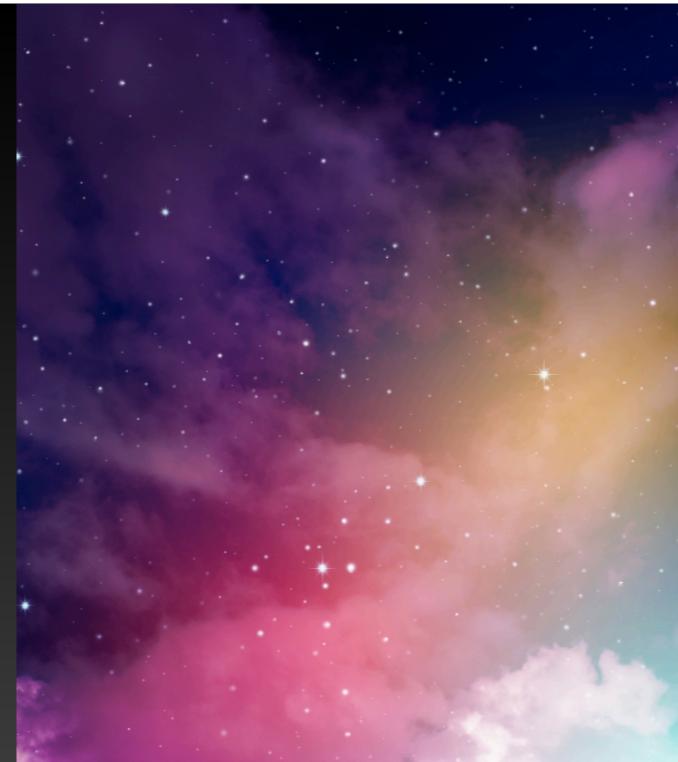


RocketMQ

分享

Arthur.lee

2021.1



Rocket MQ作用与特性

Rocket MQ架构

Rocket MQ存储模型

Rocket MQ高可靠

Rocket MQ事务

Rocket MQ定时/延时消息

Rocket MQ消息重试和幂等

Rocket MQ和Rabbit MQ消息堆积能力

Rabbit作用与特性

作用

流量削峰（主要解决瞬时写压力大于应用服务能力导致消息丢失、系统奔溃等问题）

系统解耦（解决不同重要程度、不同能力级别系统之间依赖导致一死全死）

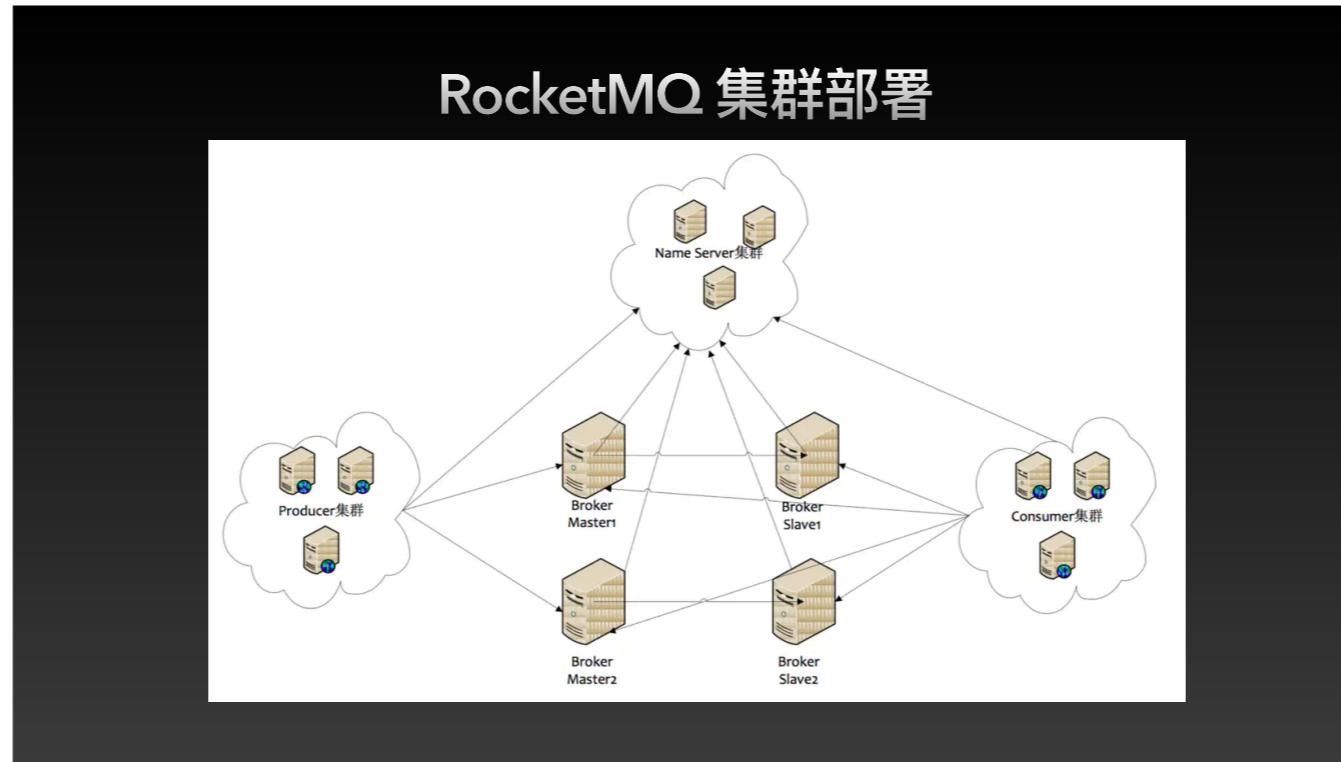
提升性能（当存在一对多调用时，可以发一条消息给消息系统，让消息系统通知相关系统）

蓄流压测（线上有些链路不好压测，可以通过堆积一定量消息再放开来压测）

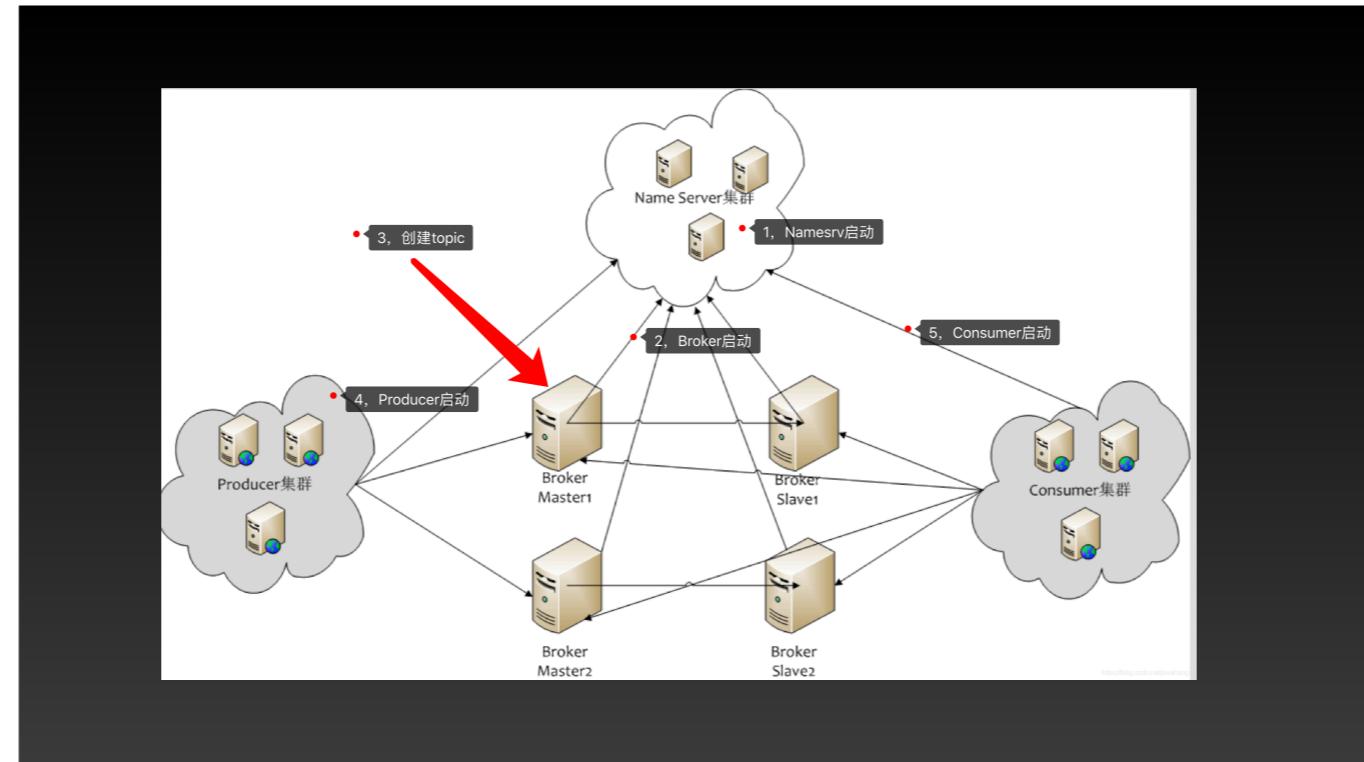
特性

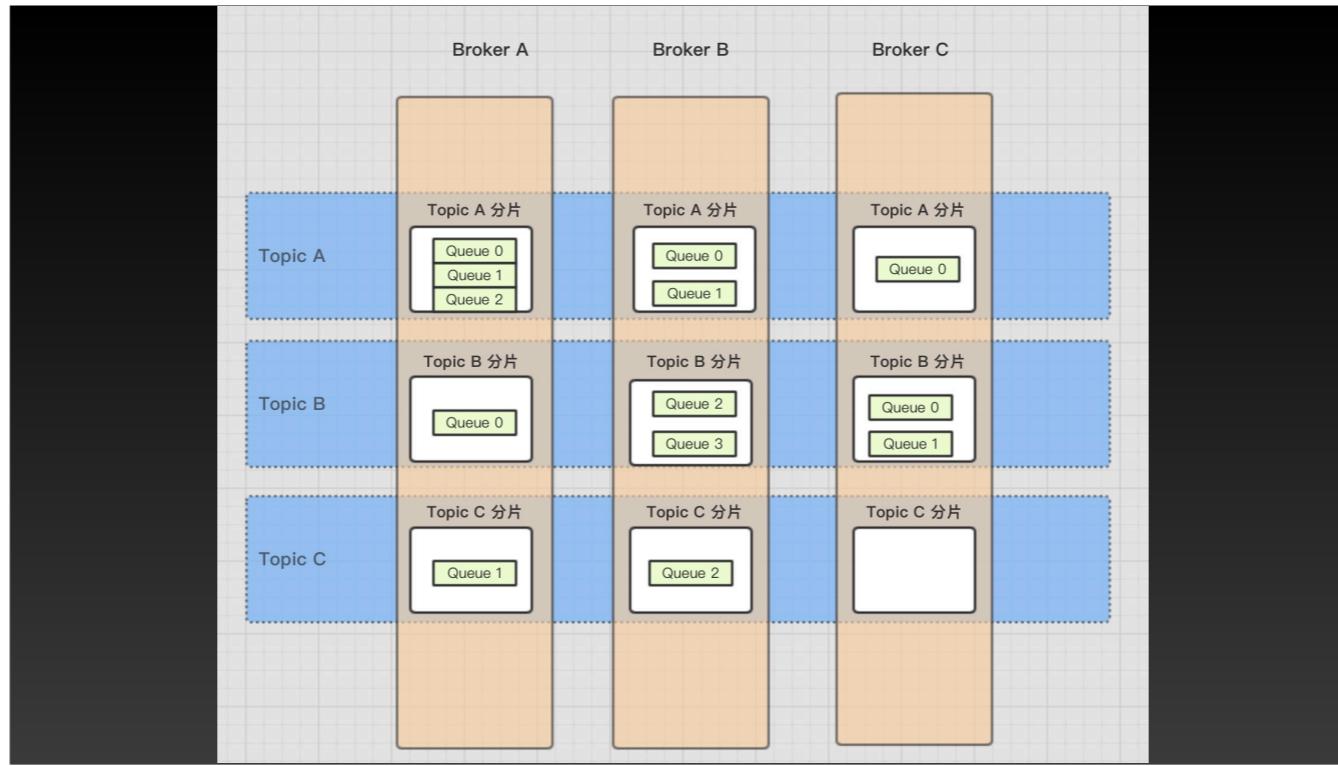
- | | | |
|---------|---------|---------|
| 1、订阅与发布 | 2 消息顺序 | 3 消息过滤 |
| 4 消息可靠性 | 5 至少一次 | 6 回溯消费 |
| 7 事务消息 | 8 定时消息 | 9 消息重试 |
| 10 消息重投 | 11 流量控制 | 12 死信队列 |

Rocketmq构架设计



- ①Name Server是一个几乎无状态节点，可集群部署，节点之间无任何信息同步。
- ②Broker部署相对复杂，Broker分为Master与Slave，一个Master可以对应多个Slave，但是一个Slave只能对应一个Master，Master与Slave的对应关系通过指定相同的BrokerName，不同的BrokerId来定义，BrokerId为0表示Master，非0表示Slave。Master也可以部署多个。每个Broker与Name Server集群中的所有节点建立长连接，定时注册Topic信息到所有Name Server。
- ③Producer与Name Server集群中的其中一个节点（随机选择）建立长连接，定期从Name Server取Topic路由信息，并向提供Topic服务的Master建立长连接，且定时向Master发送心跳。Producer完全无状态，可集群部署。
- ④Consumer与Name Server集群中的其中一个节点（随机选择）建立长连接，定期从Name Server取Topic路由信息，并向提供Topic服务的Master、Slave建立长连接，且定时向Master、Slave发送心跳。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，订阅规则由Broker配置决定。

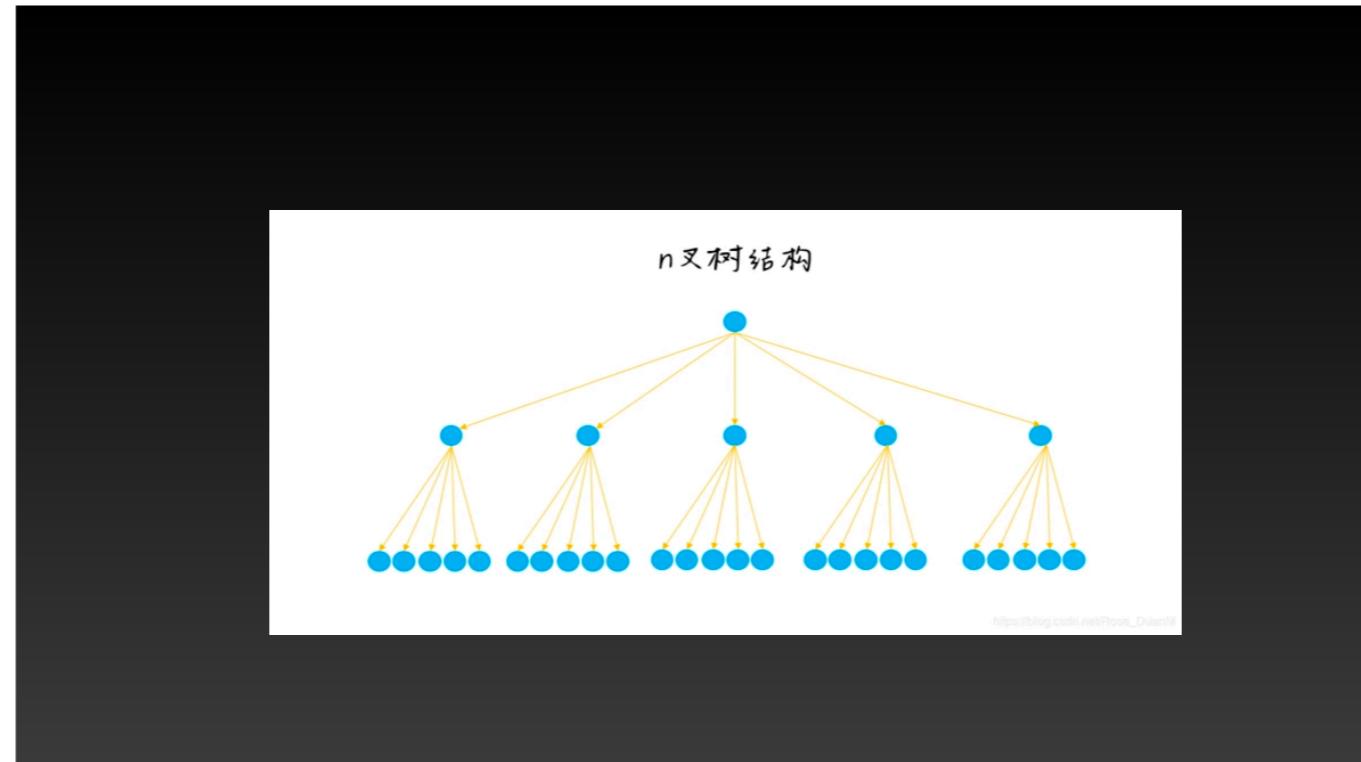




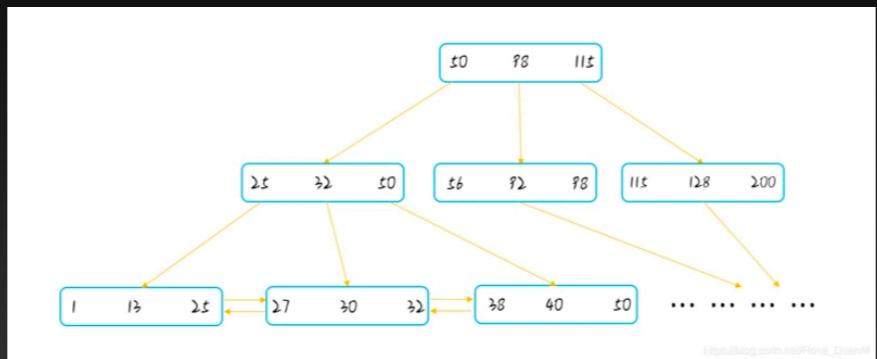
- 1、消费者发送的Message会在Broker中的Queue队列中记录。
- 2、一个Topic的数据可能会存在多个Broker中。
- 3、一个Broker存在多个Queue。
- 4、单个的Queue也可能存储多个Topic的消息。

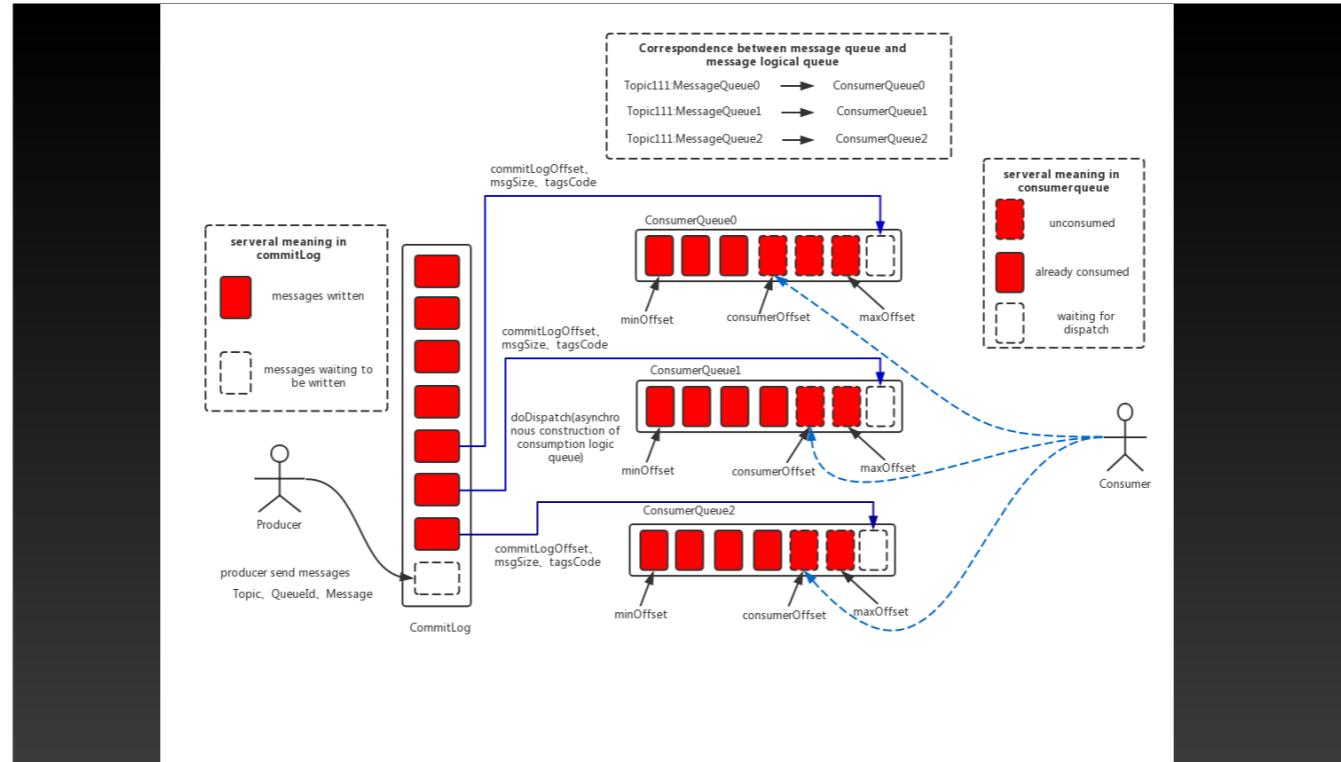
也就是说每个Topic在Broker上会划分成几个逻辑队列，每个逻辑队列保存一部分消息数据，但是保存的消息数据实际上不是真正的消息数据，而是指向commit log的消息索引。

Rocketmq存储模型



MySQL存储

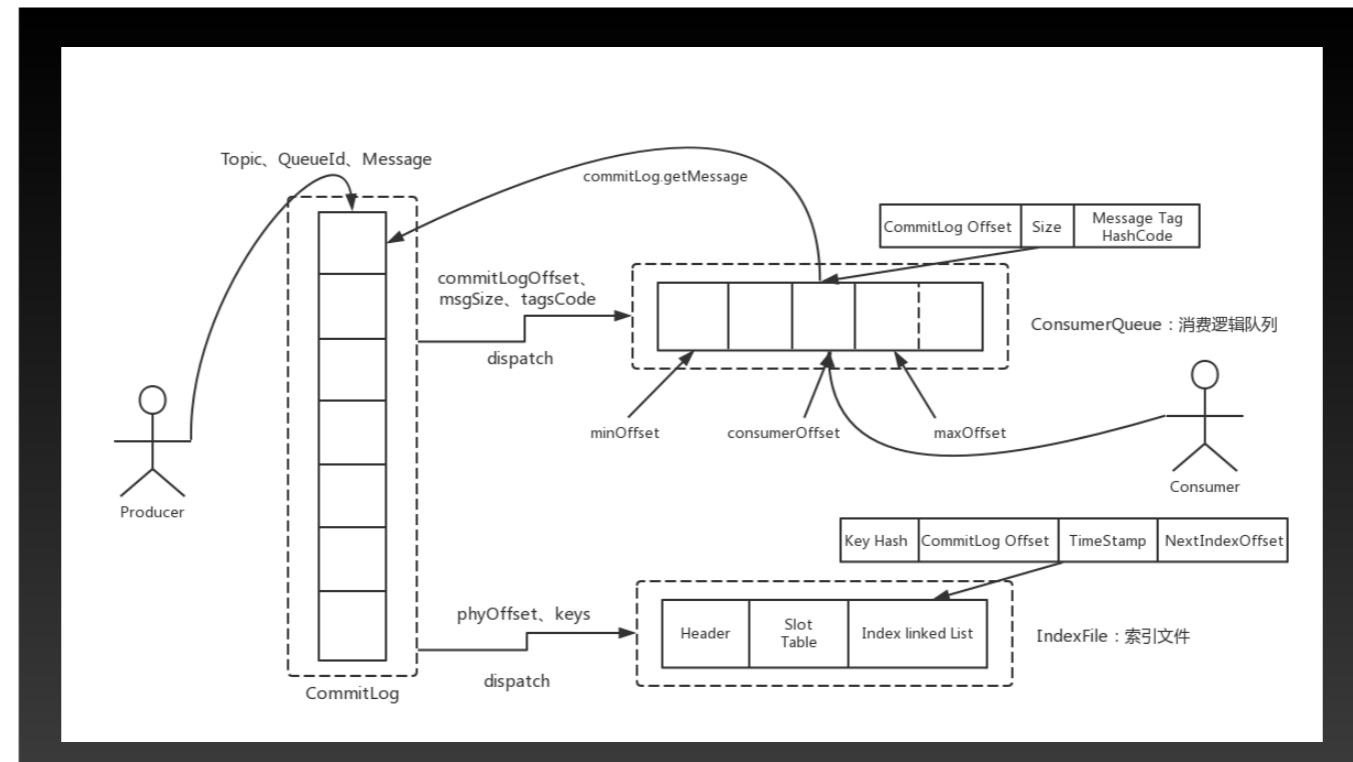




RocketMQ的混合型存储结构(多个Topic的消息实体内容都存储于一个CommitLog中)针对Producer和Consumer分别采用了数据和索引部分相分离的存储结构。

ConsumeQueue: 消息消费队列，引入的目的主要是提高消息消费的性能的，如果要遍历commitlog文件中根据topic检索消息是非常低效的。Consumer即可根据ConsumeQueue来查找待消费的消息。consumequeue文件可以看成是基于topic的 commitlog索引文件。同样consumequeue文件采取定长设计，每一个条目共20个字节，分别为8字节的commitlog物理偏移量、4字节的消息长度、8字节tag hashcode，单个文件由30W个条目组成，可以像数组一样随机访问每一个条目，每个 ConsumeQueue 文件大小约5.72M；

IndexFile: IndexFile (索引文件) 提供了一种可以通过key或时间区间来查询消息的方法。Index文件的存储位置是：`$HOME\store\index${fileName}`，文件名fileName是以创建时的时间戳命名的，固定的单个IndexFile文件大小约为400M，一个IndexFile可以保存 2000W个索引，IndexFile的底层存储设计为在文件系统中实现HashMap结构，故rocketmq的索引文件其底层实现为hash索引。



页缓存与内存映射

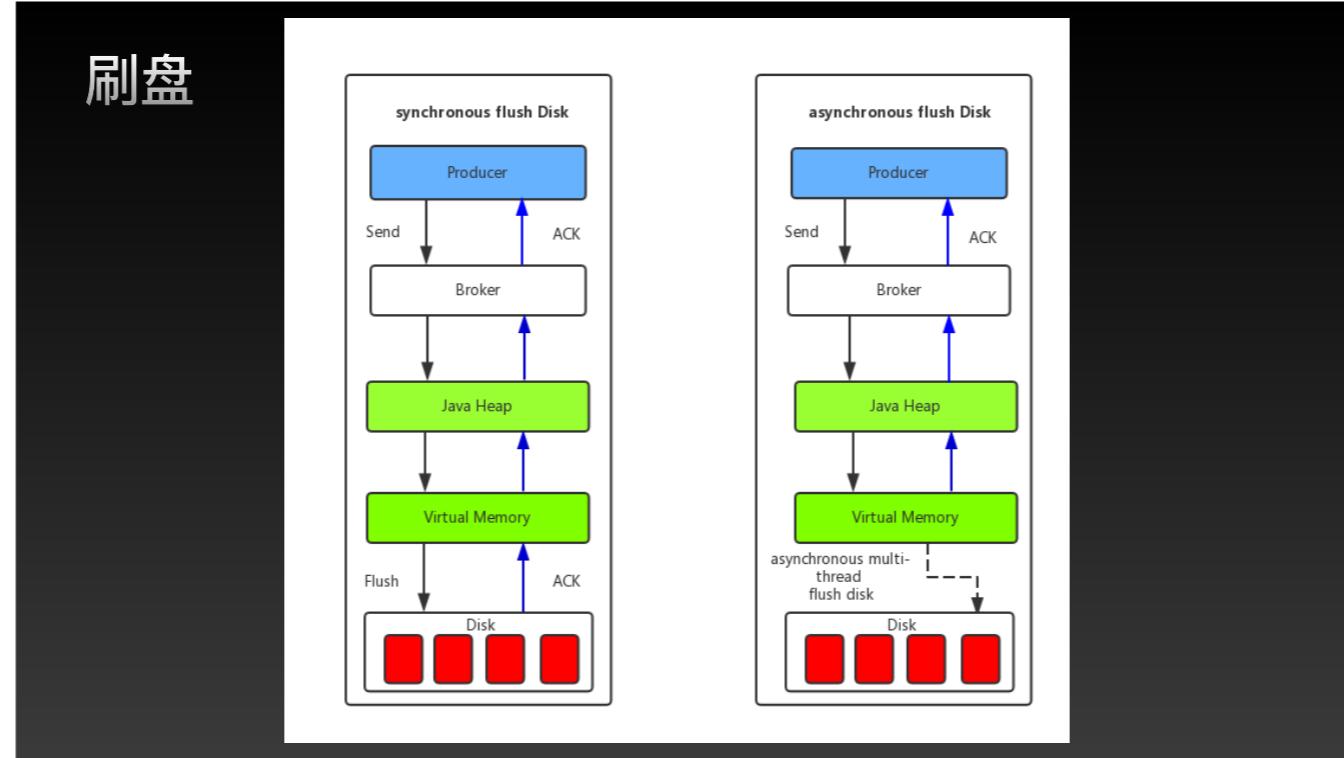
页缓存 (PageCache)是OS对文件的缓存，用于加速对文件的读写。

在RocketMQ中，ConsumeQueue逻辑消费队列存储的数据较少，并且是顺序读取，在page cache机制的预读取作用下，Consume Queue文件的读性能几乎接近读内存，即使在有消息堆积情况下也不会影响性能。而对于CommitLog消息存储的日志数据文件来说，读取消息内容时候会产生较多的随机访问读取，严重影响性能。如果选择合适的系统IO调度算法，比如设置调度算法为“Deadline”（此时块存储采用SSD的话），随机读的性能也会有所提升。、

RocketMQ主要通过MappedByteBuffer对文件进行读写操作。其中，利用了NIO中的FileChannel模型将磁盘上的物理文件直接映射到用户态的内存地址中（这种Mmap的方式减少了传统IO将磁盘文件数据在操作系统内核地址空间的缓冲区和用户应用程序地址空间的缓冲区之间来回进行拷贝的性能开销），将对文件的操作转化为直接对内存地址进行操作，从而极大地提高了文件的读写效率（正因为需要使用内存映射机制，故RocketMQ的文件存储都使用定长结构来存储，方便一次将整个文件映射至内存）。

Memory Mapped Files(后面简称mmap)也被翻译成 内存映射文件，在64位操作系统中一般可以表示20G的数据文件，它的工作原理是直接利用操作系统的Page来实现文件到物理内存的直接映射。完成映射之后你对物理内存的操作会被同步到硬盘上（操作系统在适当的时候）。

刷盘



(1) 同步刷盘：如上图所示，只有在消息真正持久化至磁盘后RocketMQ的Broker端才会真正返回给Producer端一个成功的ACK响应。同步刷盘对MQ消息可靠性来说是一种不错的保障，但是性能上会有较大影响，一般适用于金融业务应用该模式较多。

(2) 异步刷盘：能够充分利用OS的PageCache的优势，只要消息写入PageCache即可将成功的ACK返回给Producer端。消息刷盘采用后台异步线程提交的方式进行，降低了读写延迟，提高了MQ的性能和吞吐量。

Rocketmq高可靠

多 Master 多 Slave 模式，同步双写多 Master 模式

一个集群无 Slave，全是 Master，例如 2 个 Master 或者 3 个 Master

优点：配置简单，单个Master宕机或重启维护对应用无影响，在磁盘配置为 RAID10 时，即使机器宕机不可恢复情况下，由与 RAID10 磁盘非常可靠，消息也不会丢（异步刷盘丢失少量消息，同步刷盘一条不丢）。性能最高。

缺点：单台机器宕机期间，这台机器上未被消费的消息在机器恢复之前不可订阅，消息实时性会受到受到影响。

多 Master 多 Slave 模式，同步双写

每个 Master 配置一个 Slave，有多对Master-Slave，HA 采用同步双写方式，主备都写成功，向应用返回成功。

优点：数据与服务都无单点，Master宕机情况下，消息无延迟，服务可用性与数据可用性都非常高

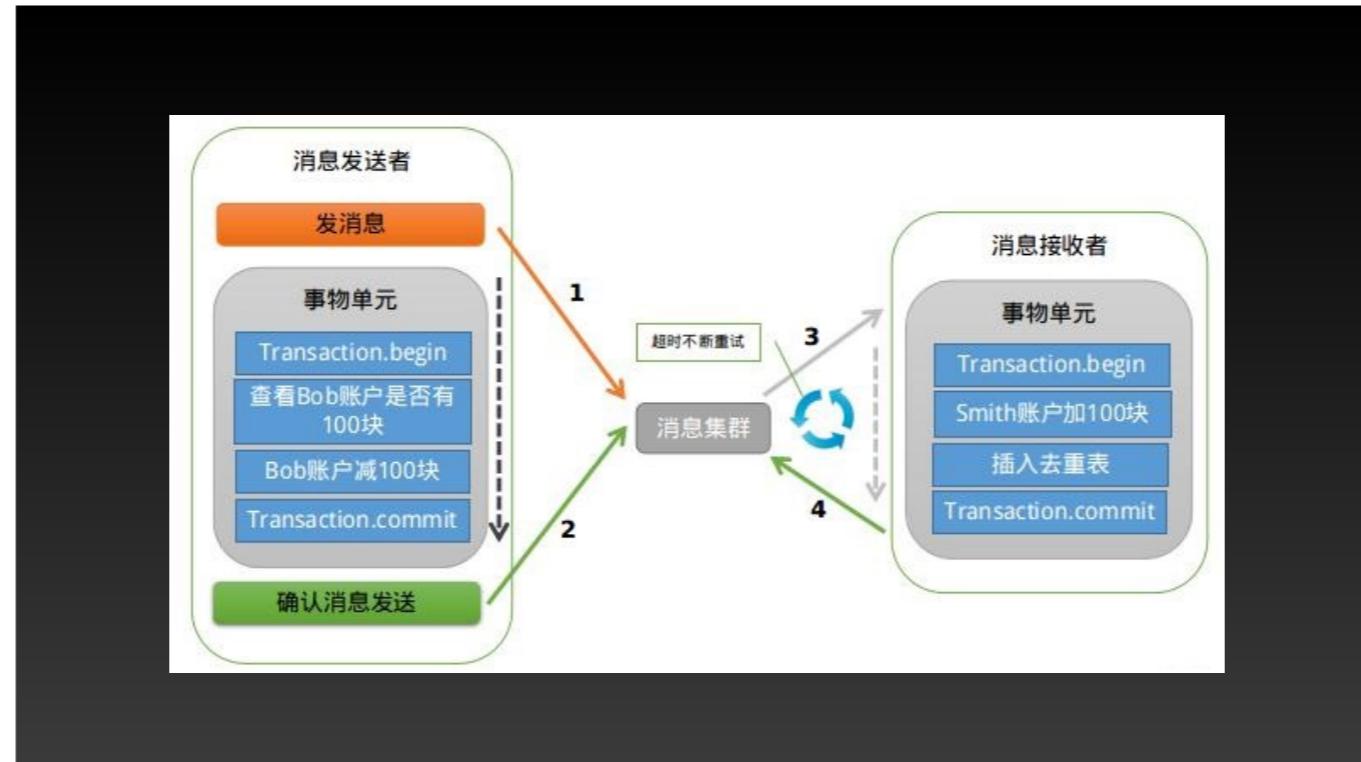
缺点：性能比异步复制模式略低，大约低 10%左右，发送单个消息的 RT 会略高。目前主宕机后，备机不能自动切换为主机，后续会支持自动切换功能。

Rocketmq事务

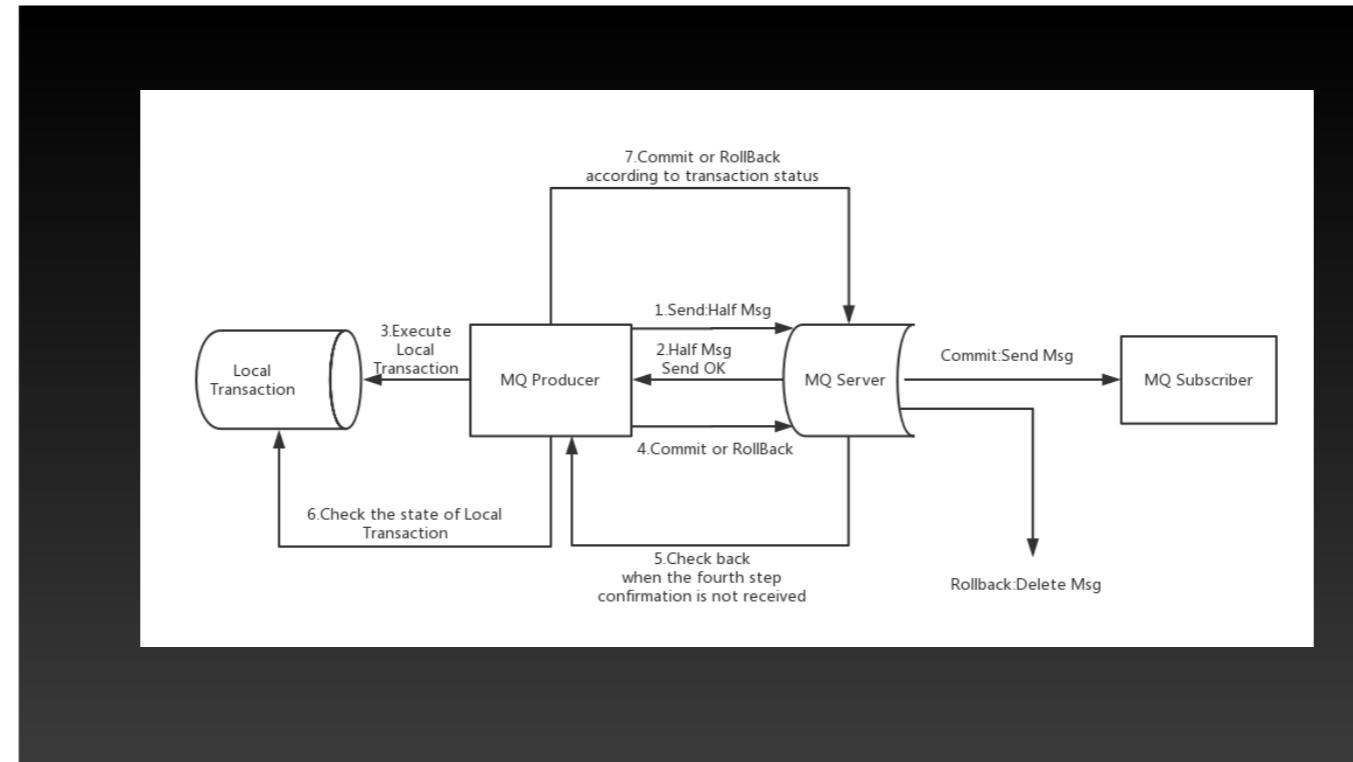
分布式事务，我们一般都是强调的最终一致性，而不是强一致性！！！

主要概括为3类：

- 基于单个JVM，数据库分库分表了（跨多个数据库）。
- 基于多JVM，服务拆分了（不跨数据库）。
- 基于多JVM，服务拆分了 并且数据库分库分表了。



rocketmq并不会无休止的的信息事务状态回查， 默认回查15次， 如果15次回查还是无法得知事务状态， rocketmq默认回滚该消息。



rocketmq并不会无休止的的信息事务状态回查， 默认回查15次， 如果15次回查还是无法得知事务状态， rocketmq默认回滚该消息。

- 1.发送方向 MQ 服务端发送消息。
- 2.MQ Server 将消息持久化成功之后， 向发送方 ACK 确认消息已经发送成功， 此时消息为半消息。
- 3.发送方开始执行本地事务逻辑。
- 4.发送方根据本地事务执行结果向 MQ Server 提交二次确认（Commit 或是 Rollback）， MQ Server 收到 Commit 状态则将半消息标记为可投递， 订阅方最终将收到该消息； MQ Server 收到 Rollback 状态则删除半消息， 订阅方将不会接受该消息。
- 5.在断网或者是应用重启的特殊情况下， 上述步骤4提交的二次确认最终未到达 MQ Server， 经过固定时间后 MQ Server 将对该消息发起消息回查。
- 6.发送方收到消息回查后， 需要检查对应消息的本地事务执行的最终结果。
- 7.发送方根据检查得到的本地事务的最终状态再次提交二次确认， MQ Server 仍按照步骤4对半消息进行操作。

Rocketmq延迟消息

18个level的延迟消息

1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m
9m 10m 20m 30m 1h 2h

Rocketmq消息的重试和幂等

生产端重试

如果由于网络抖动等原因， Producer程序向Broker发送消息时没有成功， 即发送端没有收到Broker的ACK， 导致最终 Consumer无法消费消息， 此时RocketMQ会自动进行重试。

//同步发送消息， 如果5秒内没有发送成功，则重试5次

```
DefaultMQProducer producer = new  
DefaultMQProducer("DefaultProducer");  
producer.setRetryTimesWhenSendFailed(5);  
producer.send(msg,5000L);
```

消费端重试

由于MQ的重试机制，难免会引起消息的重复消费问题。比如一个ConsumerGroup中有两个，Consumer1和Consumer2，以集群方式消费。假设一条消息发往ConsumerGroup，由Consumer1消费，但是由于Consumer1消费过慢导致超时，次数Broker会将消息发送给Consumer2去消费，这样就产生了重复消费问题。因此，使用MQ时应该对一些关键消息进行幂等去重的处理。

At least Once

是指每个消息必须投递一次。RocketMQ Consumer先pull消息到本地，消费完成后，才向服务器返回ack，如果没有消费一定不会ack消息，所以RocketMQ可以很好的支持此特性。

消息重复的场景

1/发送时消息重复

当一条消息已被成功发送到服务端并完成持久化，此时出现了网络闪断或者客户端宕机，导致服务端对客户端应答失败。如果此时生产者意识到消息发送失败并尝试再次发送消息，消费者后续会收到两条内容相同并且 Message ID 也相同的消息。

2/投递时消息重复

消息消费的场景下，消息已投递到消费者并完成业务处理，当客户端给服务端反馈应答的时候网络闪断。为了保证消息至少被消费一次，消息队列 RocketMQ 版的服务端将在网络恢复后再次尝试投递之前已被处理过的消息，消费者后续会收到两条内容相同并且 Message ID 也相同的消息。

3/负载均衡时消息重复（包括但不限于网络抖动、Broker 重启以及消费者应用重启）

当消息队列 RocketMQ 版的 Broker 或客户端重启、扩容或缩容时，会触发 Rebalance，此时消费者可能会收到重复消息。

以支付场景为例，可以将消息的 **Key** 设置为订单号，作为幂等处理的依据

```
Message message = new Message();
message.setKey("ORDERID_100");
SendResult sendResult = producer.send(message);
```

消费者收到消息时可以根据消息的 **Key**，即订单号来实现消息幂等：

```
consumer.subscribe("ons_test", "*", new MessageListener(){
    public Action consume(Message message, ConsumeContext context){
        String key = message.getKey()
        // 根据业务唯一标识的 Key 做幂等处理
    }
});
```

Message ID 有可能出现冲突（重复）的情况，所以真正安全的幂等处理，不建议以 Message ID 作为处理依据。最好的方式是以业务唯一标识作为幂等处理的关键依据，而业务的唯一标识可以通过消息 Key 设置。

Rabbit消息堆积能力

RabbitMQ消息有两种类型:

1. 持久化消息和非持久化消息。
2. 这两种消息都会被写入磁盘。

持久化消息在到达队列时写入磁盘，同时会内存中保存一份备份，当内存吃紧时，消息从内存中清除。这会提高一定的性能。非持久化消息一般只存于内存中，当内存压力大时数据刷盘处理，以节省内存空间。RabbitMQ存储层包含两个部分:队列索引和消息存储。

https://blog.csdn.net/rzpy_qifengxiaooyue/article/details/109344346

RabbitMQ队列的4种状态

alpha:消息索引和消息内容都存内存，最耗内存，很少消耗CPU

beta:消息索引存内存，消息内容存磁盘

gama:消息索引内存和磁盘都有，消息内容存磁盘

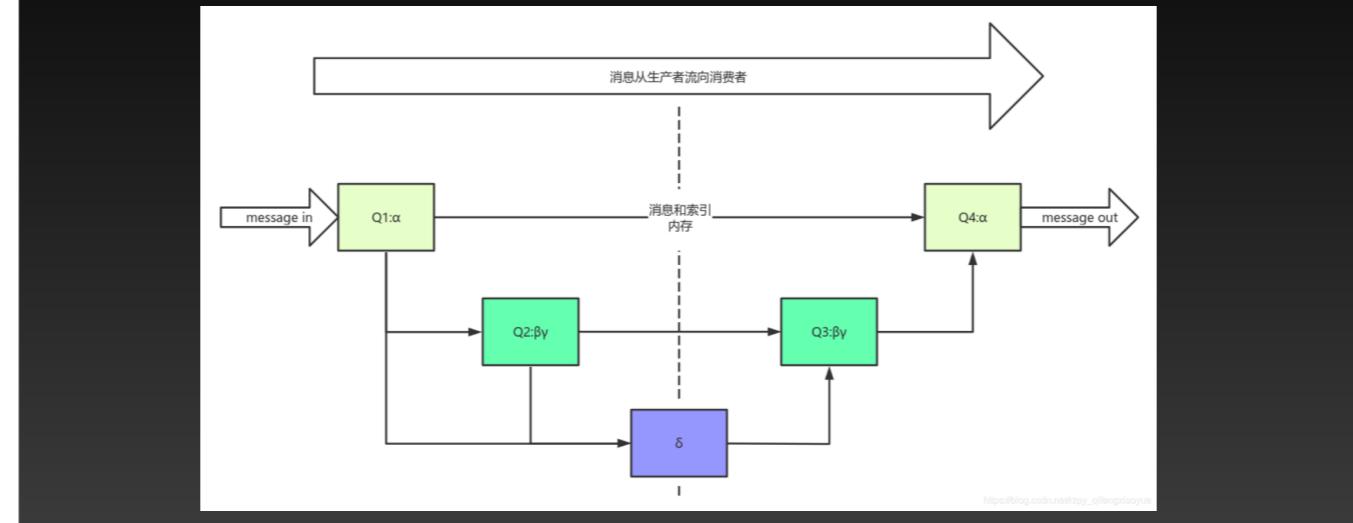
Delta:消息索引和内容都存磁盘，基本不消耗内存，消耗更多CPU和I/O操作

持久化的消息，索引和内容都必须先保存在磁盘上，才会处于上述状态中的一种

gama状态只有持久化消息才会有状态。

https://blog.csdn.net/rzpy_qifengxiaooyue/article/details/109344346

在运行时，**RabbitMQ**会根据消息传递的速度定期计算一个当前内存中能够保存的最大消息数量(**target_ram_count**)，如果**alpha**状态的消息数量大于此值，则会引起消息的状态转换，多余的消息可能会转换到**beta**、**gamma**或者**delta**状态。区分这4种状态的主要作用是满足不同的内存和**CPU**需求。



特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ 和 Kafka 低一个数量级	同 ActiveMQ	10万级，支持高吞吐	10万级，高吞吐，一般配合大数据类的系统进行实时计算、日志采集等场景
topic 数量对吞吐量的影响	-	-	topic 达到几百、几千的级别，吞吐量会有较小幅度的下降，这是 RocketMQ 的一大优势，在同等机器数量下，可以支持大量的 topic	topic 从几十到几百个的时候，吞吐量会大幅度下降，在同等的机器数量下，Kafka 尽量保证 topic 数量不要过多，如果需要支撑大规模的 topic，需要增加机器资源
时效性	毫秒级	微秒级，这是 RabbitMQ 的一大特性，低延迟	毫秒级	延迟在毫秒级以内
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
消息可靠性	有较低的概率丢失数据	-	经参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域中的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，支持分布式扩招	功能较为简单，支持简单的 MQ 功能，主要用于大数据领域，如实时计算以及日志采集等场景

Rocketmq其它参考

在Linux上的优化

<http://rocketmq.apache.org/docs/system-config/>

Apache RocketMQ开发者指南

<https://github.com/apache/rocketmq/tree/master/docs/cn>

Rocketmq顺序消费

RocketMQ顺序消费

在网购的时候，我们需要下单，那么下单需要假如有三个顺序，第一：创建订单，第二：订单付款，第三：订单完成。也就是这个三个环节要有顺序，这个订单才有意义。RocketMQ可以保证顺序消费。

方案1：rocketMq实现顺序消费的原理

produce在发送消息的时候，把消息发到同一个队列（queue）中，消费者注册消息监听器为MessageListenerOrderly，这样就可以保证消费端只有一个线程去消费消息

注意：是把消息发到同一个队列（queue），不是同一个topic，默认情况下一个topic包括4个queue

方案2：两个锁

集群模式下锁队列保证消息被同一个consumer消费，往broker定时发送锁命令
本地消费时不论集群模式和广播模式都会有本地队列锁进行锁定
保证同一个队列只会同时被一个消费者线程锁定





谢谢！

