## 部署方式

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。

## 刷盘策略

在commitLog.putMessage中决定刷盘方式，在MessageStoreConfig中配置刷盘的方式

1.RocketMQ的消息都是持久化的：所有消息保存在commitlog文件夹下的文件中

2.先写入系统PageCache：所有commitlog下的文件都是使用的直接内存，采用mmap文件映射的方法，每次接收到消息的时候先把消息写入直接内存PageCache——即mapedFile

3.然后刷盘：启动commitLog的时候会启动刷盘的线程（FlushCommitLogService）定时刷盘

## 持久化方式

大多数的MQ都是支持持久化存储，而且业务上也大多需要MQ有持久存储的能力，能大大增加系统的高可用性，下面几种存储方式：

- 分布式KV存储（levelDB,RocksDB,redis）
- 传统的文件系统
- 传统的关系型数据库

这几种存储方式从效率来看， **文件系统 > kv存储 > 关系型数据库** ，因为直接操作文件系统肯定是最快的，而关系型数据库一般的TPS都不会很高，我印象中Mysql的写不会超过5Wtps（现在不确定最新情况）,所以如果追求效率就直接操作文件系统。

但是如果从可靠性和易实现的角度来说，则是 关系型数据库 > kv存储 > 文件系统 ，消息存在db里面非常可靠，但是性能会下降很多，所以具体的技术选型都是需要根据自己的业务需求去考虑

## 一、MQ消息队列的一般存储方式

当前业界几款主流的MQ消息队列采用的存储方式主要有以下三种方式：
**分布式KV存储**：这类MQ一般会采用诸如levelDB、RocksDB和Redis来作为消息持久化的方式，由于分布式缓存的读写能力要优于DB，所以在对消息的读写能力要求都不是比较高的情况下，采用这种方式倒也不失为一种可以替代的设计方案。消息存储于分布式KV需要解决的问题在于如何保证MQ整体的可靠性？

**文件系统**：目前业界较为常用的几款产品（RocketMQ/Kafka/RabbitMQ）均采用的是消息刷盘至所部署虚拟机/物理机的文件系统来做持久化（刷盘一般可以分为异步刷盘和同步刷盘两种模式）。小编认为，消息刷盘为消息存储提供了一种高效率、高可靠性和高性能的数据持久化方式。除非部署MQ机器本身或是本地磁盘挂了，否则一般是不会出现无法持久化的故障问题。

**关系型数据库DB**：Apache下开源的另外一款MQ—ActiveMQ（默认采用的KahaDB做消息存储）可选用JDBC的方式来做消息持久化，通过简单的xml配置信息即可实现JDBC消息存储。由于，普通关系型数据库（如Mysql）在单表数据量达到千万级别的情况下，其IO读写性能往往会出现瓶颈。因此，如果要选型或者自研一款性能强劲、吞吐量大、消息堆积能力突出的MQ消息队列，那么小编并不推荐采用关系型数据库作为消息持久化的方案。在可

靠性方面，该种方案非常依赖DB，如果一旦DB出现故障，则MQ的消息就无法落盘存储会导致线上故障；

因此，综合上所述从存储效率来说，**文件系统>分布式KV存储>关系型数据库DB**，直接操作文件系统肯定是最快和最高效的，而关系型数据库TPS一般相比于分布式KV系统会更低一些（简略地说，关系型数据库本身也是一个需要读写文件server，这时MQ作为client与其建立连接并发送待持久化的消息数据，同时又需要依赖DB的事务等，这一系列操作都比较消耗性能），所以如果追求高效的IO读写，那么选择操作文件系统会更加合适一些。但是如果从易于实现和快速集成来看，**关系型数据库DB>分布式KV存储>文件系统**，但是性能会下降很多。

另外，从消息中间件的本身定义来考虑，应该尽量减少对于外部第三方中间件的依赖。一般来说依赖的外部系统越多，也会使得本身的设计越复杂，所以小编个人的理解是采用文件系统作为消息存储的方式，更贴近消息中间件本身的定义

## 二、RocketMQ消息存储整体架构