参考：

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

官网：<http://rocketmq.apache.org/docs/quick-start/>

项目样例：

<https://github.com/aecoplus/mq>

**安装rocketmq**

wget https://github.com/apache/rocketmq/archive/rocketmq-all-4.3.1.tar.gz

unzip rocketmq-all-4.3.1-source-release.zip

cd rocketmq-all-4.3.1

mvn -Prelease-all -DskipTests clean install -U

安装完rocketmq后要注意更改默认分配内存问题

https://blog.csdn.net/qq\_38416576/article/details/86153049

**启动name server：**

先进入 RocketMQ 安装目录下的  distribution/target/apache-rocketmq 目录

$ nohup sh bin/mqnamesrv & # 启动命令,并且常驻内存

$ tail -f ~/logs/rocketmqlogs/namesrv.log

在当前目录下会有一个 nohup.out 的日志文件，可以打开查看 namesrv 的启动情况

**启动 Broker：**

同样进入 RocketMQ 安装目录下的  distribution/target/apache-rocketmq 目录

> nohup sh bin/mqbroker -n localhost:9876 &

> tail -f ~/logs/rocketmqlogs/broker.log

**关闭服务：**

sh bin/mqshutdown broker

sh bin/mqshutdown namesrv

**可视化rocketmq管理工具，启动(rocketmq-console目录下)：**

#如果配置文件有name server地址：

java -jar target/rocketmq-console-ng-1.0.0.jar #如果配置文件没有填写Name Server $ java -jar target/rocketmq-console-ng-1.0.0.jar --rocketmq.config.namesrvAddr='10.0.74.198:9876;10.0.74.199:9876'

**架构：**

RocketMQ架构上主要分 NameServer、Producer、Broker、Consumer 四部分组成

**NameServer**是一个Topic路由注册中心。主要包括两个功能：Broker管理，NameServer接受Broker集群的注册信息并且保存下来作为路由信息的基本数据；路由信息管理，每个NameServer将保存关于Broker集群的整个路由信息和用于客户端查询的队列信息。

**Producer：**Producer通过MQ的负载均衡模块选择相应的Broker集群队列进行消息投递，投递的过程支持快速失败并且低延迟。

**Consumer** ：RocketMQ提供了两种消费模式，PUSH和PULL，大多数场景使用的是PUSH模式。PUSH模式实际上在内部还是使用的PULL方式实现的，通过PULL不断地轮询Broker获取消息，当不存在新消息时，Broker会挂起PULL请求，直到有新消息产生才取消挂起，返回新消息。

PushConsumer：应用通常向Consumer 对象注册一个 Listener 接口，一旦收到消息，Consumer 对象立 刻回调 Listener 接口方法

Pull Consumer：应用通常主劢调用 Consumer 的拉消息方法从 Broker 拉消息，主动权由应用控制

同时consumer也支持集群方式（consumer实例均分消息）和广播方式（所有实例均消费消息）的消费。

**Broker**：Broker主要负责消息的存储、投递和查询以及服务高可用保证，为了实现这些功能，Broker包含了以下几个重要子模块。

Remoting Module：整个Broker的实体，负责处理来自clients端的请求。

Client Manager：负责管理客户端(Producer/Consumer)和维护Consumer的Topic订阅信息

Store Service：提供方便简单的API接口处理消息存储到物理硬盘和查询功能。

HA Service：高可用服务，提供Master Broker 和 Slave Broker之间的数据同步功能。

Index Service：根据特定的Message key对投递到Broker的消息进行索引服务，以提供消息的快速查询。

**部署：**

**NameServer：**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。

**Producer**与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取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。

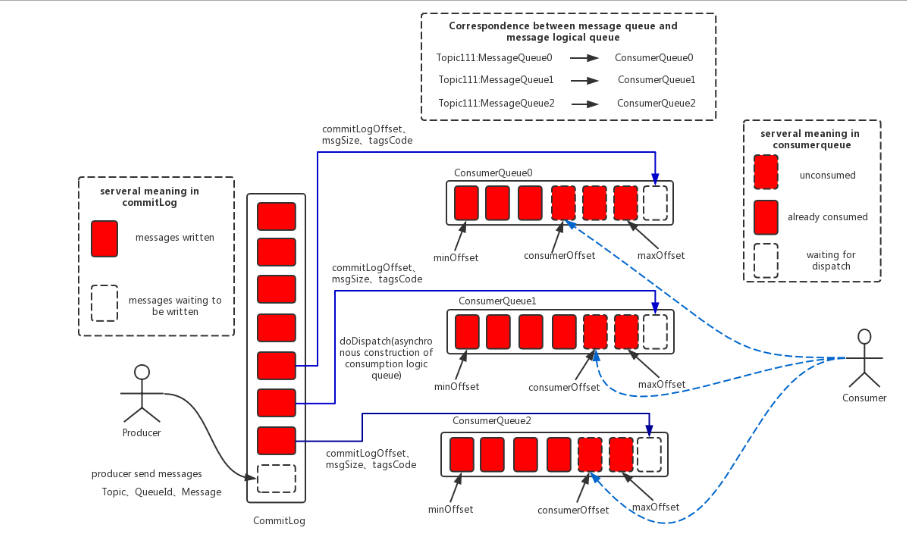
**Consumer：**同producer。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，消费者在向Master拉取消息时，Master服务器会根据拉取偏移量与最大偏移量的距离（判断是否读老消息，产生读I/O），以及从服务器是否可读等因素建议下一次是从Master还是Slave拉取。

**Broker：**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NameServer集群中的所有节点建立长连接，定时注册Topic信息到所有NameServer。

**特性（**<https://github.com/apache/rocketmq/blob/master/docs/cn/design.md>**）：**

**消息存储：**

（1）消息存储



1. CommitLog：消息主体以及元数据的存储主体，存储Producer端写入的消息主体内容,消息内容不是定长的。单个文件大小默认1G ，文件名长度为20位，左边补零，剩余为起始偏移量，比如00000000000000000000代表了第一个文件，起始偏移量为0，文件大小为1G=1073741824；当第一个文件写满了，第二个文件为00000000001073741824，起始偏移量为1073741824，以此类推。消息主要是顺序写入日志文件，当文件满了，写入下一个文件；

2. ConsumeQueue（in logic）：消息消费队列，引入的目的主要是提高消息消费的性能，ConsumeQueue（逻辑消费队列）作为消费消息的索引，保存了指定Topic下的队列消息在CommitLog中的起始物理偏移量offset，消息大小size和消息Tag的HashCode值。consumequeue文件可以看成是基于topic的commitlog索引文件

3. IndexFile：提供了一种可以通过key或时间区间来查询消息的方法，文件名fileName是以创建时的时间戳命名的。

（2）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。一般来说，程序对文件进行顺序读写的速度几乎接近于内存的读写速度，主要原因就是由于OS使用PageCache机制对读写访问操作进行了性能优化，将一部分的内存用作PageCache。对于数据的写入，OS会先写入至Cache内，随后通过异步的方式由pdflush内核线程将Cache内的数据刷盘至物理磁盘上。对于数据的读取，如果一次读取文件时出现未命中PageCache的情况，OS从物理磁盘上访问读取文件的同时，会顺序对其他相邻块的数据文件进行预读取。

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。

另外，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）。

（3）刷盘：

保证磁盘和内存都有一份数据，访问时直接从内存读取。

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。低性能，高可靠。

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。

**通信：**

1. Broker启动后需要完成一次将自己注册至NameServer的操作；随后每隔30s时间定时向NameServer上报Topic路由信息。

2. 消息生产者Producer作为客户端发送消息时候，需要根据消息的Topic从本地缓存的TopicPublishInfoTable获取路由信息。如果没有则更新路由信息会从NameServer上重新拉取，同时Producer会默认每隔30s向NameServer拉取一次路由信息。

3. 消息生产者Producer根据2）中获取的路由信息选择一个队列（MessageQueue）进行消息发送；Broker作为消息的接收者接收消息并落盘存储。

4. 消息消费者Consumer根据2）中获取的路由信息，并再完成客户端的负载均衡后，选择其中的某一个或者某几个消息队列来拉取消息并进行消费。

**消息过滤：**

**负载均衡：**

**（1）Producer端：**

**（2）COnsumer端：**

**事务消息：**

**消息查询：**

**Example：**

同步消息 异步消息 单向发送消息 顺序消息 延时消息 批量消息 过滤消息 消息事务

**集群搭建方式**

单ster模式

多Master模式

多Master多Slave模式-异步复制

多Master多Slave模式-同步双写

**FAQ:**

Generally 256KB for message Body

消息最多存几天： 最多三天，过期删除