# RocketMQ整体理解与快速实战

# 一、MQ介绍

## 1、什么是MQ？为什么要用MQ？

MQ：MessageQueue，消息队列。 队列，是一种FIFO 先进先出的数据结构。消息由生产者发送到MQ进行排队，然后按原来的顺序交由消息的消费者进行处理。QQ和微信就是典型的MQ。

MQ的作用主要有以下三个方面：

* 异步

例子：快递员发快递，直接到客户家效率会很低。引入菜鸟驿站后，快递员只需要把快递放到菜鸟驿站，就可以继续发其他快递去了。客户再按自己的时间安排去菜鸟驿站取快递。

作用：异步能提高系统的响应速度、吞吐量。

* 解耦

例子：《Thinking in JAVA》很经典，但是都是英文，我们看不懂，所以需要编辑社，将文章翻译成其他语言，这样就可以完成英语与其他语言的交流。

作用：

1、服务之间进行解耦，才可以减少服务之间的影响。提高系统整体的稳定性以及可扩展性。

2、另外，解耦后可以实现数据分发。生产者发送一个消息后，可以由一个或者多个消费者进行消费，并且消费者的增加或者减少对生产者没有影响。

* 削峰

例子：长江每年都会涨水，但是下游出水口的速度是基本稳定的，所以会涨水。引入三峡大坝后，可以把水储存起来，下游慢慢排水。

作用：以稳定的系统资源应对突发的流量冲击。

## 2、MQ的优缺点

​ 上面MQ的所用也就是使用MQ的优点。 但是引入MQ也是有他的缺点的：

* 系统可用性降低

系统引入的外部依赖增多，系统的稳定性就会变差。一旦MQ宕机，对业务会产生影响。这就需要考虑如何保证MQ的高可用。

* 系统复杂度提高

引入MQ后系统的复杂度会大大提高。以前服务之间可以进行同步的服务调用，引入MQ后，会变为异步调用，数据的链路就会变得更复杂。并且还会带来其他一些问题。比如：如何保证消费不会丢失？不会被重复调用？怎么保证消息的顺序性等问题。

* 消息一致性问题

A系统处理完业务，通过MQ发送消息给B、C系统进行后续的业务处理。如果B系统处理成功，C系统处理失败怎么办？这就需要考虑如何保证消息数据处理的一致性。

## 3、几大MQ产品特点比较

​ 常用的MQ产品包括Kafka、RabbitMQ和RocketMQ。我们对这三个产品做下简单的比较，重点需要理解他们的适用场景。



RocketMQ

淘宝内部的交易系统使用了淘宝自主研发的 Notify 消息中间件，使用 Mysql 作为消息存储媒介，可 完全水平扩容，为了进一步降低成本，我们认为存储部分可以进一步优化，2011 年初，Linkin 开源 了 Kafka 这个优秀的消息中间件，淘宝中间件团队在对 Kafka 做过充分 Review 之后，Kafka 无限消 息堆积，高效的持久化速度吸引了我们，但是同时发现这个消息系统主要定位于日志传输，对于使用 在淘宝交易、订单、充值等场景下还有诸多特性不满足，为此我们重新用 Java 语言编写了 RocketMQ， 定位于非日志的可靠消息传输(日志场景也 OK)，目前 RocketMQ 在阿里集团被广泛应用在订单， 交易，充值，流计算，消息推送，日志流式处理，binglog 分发等场景。

Kafka

Kafka 是 LinkedIn 开源的分布式发布-订阅消息系统，目前归属于 Apache 定级项目。Kafka 主要特 点是基于 Pull 的模式来处理消息消费，追求高吞吐量，一开始的目的就是用于日志收集和传输。0.8 版本开始支持复制，不支持事务，对消息的重复、丢失、错误没有严格要求，适合产生大量数据的互 联网服务的数据收集业务。

RabbitMQ

RabbitMQ 是使用 Erlang 语言开发的开源消息队列系统，基于 AMQP 协议来实现。AMQP 的主要特 征是面向消息、队列、路由(包括点对点和发布/订阅)、可靠性、安全。AMQP 协议更多用在企业 系统内，对数据一致性、稳定性和可靠性要求很高的场景，对性能和吞吐量的要求还在其次。

有关测试结论

Kafka 的吞吐量高达 17.3w/s，不愧是高吞吐量消息中间件的行业老大。这主要取决于它的队列模式 保证了写磁盘的过程是线性 IO。此时 broker 磁盘 IO 已达瓶颈。

RocketMQ 也表现不俗，吞吐量在 11.6w/s，磁盘 IO %util 已接近 100%。RocketMQ 的消息写入内 存后即返回 ack，由单独的线程专门做刷盘的操作，所有的消息均是顺序写文件。

RabbitMQ 的吞吐量 5.95w/s，CPU 资源消耗较高。它支持 AMQP 协议，实现非常重量级，为了保 证消息的可靠性在吞吐量上做了取舍。我们还做了 RabbitMQ 在消息持久化场景下的性能测试，吞吐 量在 2.6w/s 左右。

在服务端处理同步发送的性能上，Kafka>RocketMQ>RabbitMQ。

对比了最简单的小消息发送场景,Kafka 暂时胜出。但是,作为经受过历次双十一洗礼的 RocketMQ,在 互联网应用场景中更有它优越的一面。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 功能 | 消息队列 RocketMQ | Apache RocketMQ(开源) | Apache Kafka | RabbitMQ |
| 安全防护 | 支持 | 不支持 | 不支持 | 支持 |
| 主子账号支持 | 支持 | 不支持 | 不支持 | 不支持 |
| 可靠性 | - 同步刷盘 - 同步双写 - 超3份数据副本 - 99.99999999% | - 同步刷盘 - 异步刷盘 | 异步刷盘，丢数据概率高 | 同步刷盘 |
| 可用性 | - 非常好，99.95% - Always Writable | 好 | 好 | 好 |
| 横向扩展能力 | - 支持平滑扩展 - 支持百万级 QPS | 支持 | 支持 | - 集群扩容依赖前端 - LVS 负载均衡调度 |
| Low Latency | 支持 | 不支持 | 不支持 | 不支持 |
| 消费模型 | Push / Pull | Push / Pull | Pull | Push / Pull |
| 定时消息 | 支持(可精确到秒级) | 支持(只支持 18 个固定 Level) | 不支持 | 支持 |
| 事务消息 | 支持 | 不支持 | 不支持 | 不支持 |
| 顺序消息 | 支持 | 支持 | 支持 | 不支持 |
| 全链路消息轨迹 | 支持 | 不支持 | 不支持 | 不支持 |
| 消息堆积能力 | 百亿级别 不影响性能 | 百亿级别 影响性能 | 影响性能 | 影响性能 |
| 消息堆积查询 | 支持 | 支持 | 不支持 | 不支持 |
| 消息回溯 | 支持 | 支持 | 不支持 | 不支持 |
| 消息重试 | 支持 | 支持 | 不支持 | 支持 |
| 死信队列 | 支持 | 支持 | 不支持 | 支持 |
| 性能(常规) | 非常好 百万级 QPS | 非常好 十万级 QPS | 非常好 百万级 QPS | 一般 万级 QPS |
| 性能(万级 Topic 场 景) | 非常好 百万级 QPS | 非常好 十万级 QPS | 低 | 低 |
| 性能(海量消息堆积场景) | 非常好 百万级 QPS | 非常好 十万级 QPS | 低 | 低 |

**对比**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Kafka | RocketMq | RabbitMQ |
| 关注度 | 高 | 中 | 高 |
| 成熟度 | 成熟 | 比较成熟 | 成熟 |
| 所属社区/公司 | Apache | Alibaba  Apache | Mozilla  Public  License |
| 社区活跃度 | 高 | 中 | 高 |
| 文档 | 多 | 中 | 多 |
| 特点 | 吞吐量与消息积累都很强大 Topic 太多会影响性能。 | 各个环节分布式扩展设 计，主从 HA;支持上万 个队列;多种消费模式; 性能很好 | 由于Erlang 语言的并发 能力，性能很好 |
| 授权方式 | 开源 | 开源 | 开源 |
| 开发语言 | scala | Java | Erlang |
| 支持的协议 | 一套自行设计的基于 TCP 的二 进制协议 | 自己定义的一 套(社区提供 JMS--不成熟) | AMQP |
| 客户端支持语言 | C/C++,Python,Go,Erlang,Java 等 | Java  C++(不成熟) | Java、C、 C++、 Python、 PHP、 Perl、.net 等 |
| 持久化 | 磁盘文件 | 磁盘文件 | 内存、文件 |
| 事务 | 不支持 | 支持 | 不支持 |
| 集群 | Zookeeper | Nameserver |  |
| 单机支持的队列 | 单机超过 64 个队列，性能会明显 下降 | 单机最高支持 5W 个队列， 性能没有明显变化 | 依赖于内存 |
| 定时消息 | 不支持 | 开源版仅支持定时 Level | 不支持 |
| 顺序消费 | 支持顺序消费，但是一台 Broker 宕机后，顺序会乱 | 支持顺序消费，在顺序消  费场景下，消费失败时消  费队列将会暂停 | 支持顺序消费，但是一台 Broker 宕机后，顺序会乱 |
| 负载均衡 | 支持 | 支持 | 支持 |
| 管理界面 | 无 | 无社区有 web console 实现 | 好 |
| 部署依赖 | zookeeper | Nameserver | Erlang 环境 |
| 消费方式 |  | 保证严格的消费顺序 |  |
| 总结:优点 | 1、高吞吐、低延迟、高性能 2、提供多种客户端语言 3、生态完善，大数据处理方面的 必备工具 | 模型简单，接口易用。在 阿里大规模应用。目前支 付宝中的余额宝等新兴产 品均使用 rocketmq。集群 规模大概在 50 台左右，单 日处理消息上百亿;性能 非常好，可以大量堆积消 息在 broker 中;支持多种 消费，包括集群消费、广 播消费等。开发度较活跃， 版本更新很快。 | 由于 erlang 语言的特性， mq 性能较好;管理界面较 丰富，在互联网公司也有 较大规模的应用;  支持 amqp 协议，有多种语 言且支持 amqp 的客户端可 用 |
| 总结:缺点 | 消费者集群数受到分区数的限制 单机 Topic 过多，性能会明显下降 不支持事务。 容易丢数据。 | 使用者较少，生态不够 完善，消息堆积与吞吐量 上与 kafka 还是有差距。  客户端支持 java | Erlang 语言难度较大，集群 不支持动态扩展。 不支持事务，消息吞吐能 力有限 消息堆积时，性能会明显 降低。 |

# 二、RocketMQ快速实战

​ RocketMQ是阿里巴巴开源的一个消息中间件，在阿里内部历经了双十一等很多高并发场景的考验，能够处理亿万级别的消息。2016年开源后捐赠给Apache，现在是Apache的一个顶级项目。

​ 目前RocketMQ在阿里云上有一个购买即可用的商业版本，商业版本集成了阿里内部一些更深层次的功能及运维定制。我们这里学习的是Apache的开源版本。开源版本相对于阿里云上的商业版本，功能上略有缺失，但是大体上功能是一样的。

​ RocketMQ的官网地址： [http://rocketmq.apache.org](http://rocketmq.apache.org/) ，github地址是 <https://github.com/apache/rocketmq> ，当前最新的版本是4.7.1。我们就用这个4.7.1版本来进行学习。

1、下载RocketMQ 4.7.1版本

​ RocketMQ运行版本下载地址： <https://www.apache.org/dyn/closer.cgi?path=rocketmq/4.7.1/rocketmq-all-4.7.1-bin-release.zip>

​ RocketMQ源码版本下载地址： <https://www.apache.org/dyn/closer.cgi?path=rocketmq/4.7.1/rocketmq-all-4.7.1-source-release.zip>

​ 这两个版本我们都下载下来。

## 2、快速安装RocketMQ

​ RocketMQ的安装非常简单，就是上传解压就可以了。

运行RocketMQ需要先安装JDK。我们采用目前最稳定的JDK1.8版本。

把下载的rocketmq-all-4.7.1-bin-release.zip在本地完成解压，并上传到/app/rocketmq目录。完成后，把rocketmq的bin目录也配置到环境变量当中。 vi ~/.bash\_profile，加入以下内容，并执行source ~/.bash\_profile让环境变量生效：

export JAVA\_HOME=/app/jdk1.8/

export ROCKETMQ\_HOME=/app/rocketmq/rocketmq-all-4.7.1-bin-release

PATH=$ROCKETMQ\_HOME/bin:$JAVA\_HOME/bin:$PATH:$HOME/.local/bin:$HOME/bin

export PATH

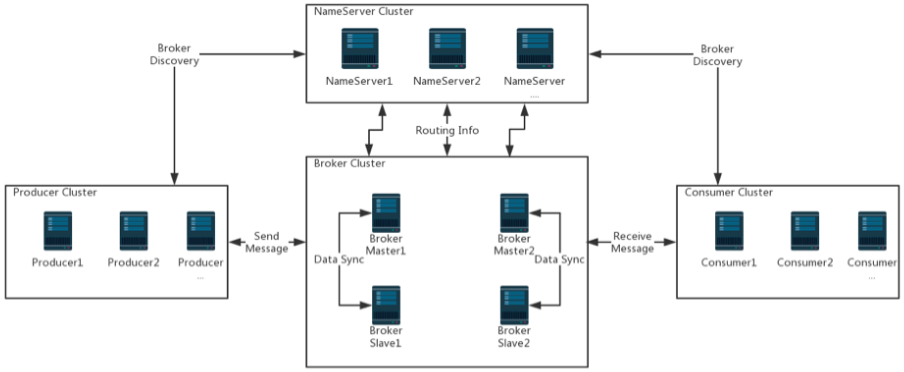
这样RocketMQ就安装完成了。我们把他运行起来。

这个ROCKETMQ\_HOME的环境变量是必须要单独配置的，如果不配置的话，启动NameSever和Broker都会报错。

这个环境变量的作用是用来加载$ROCKETMQ\_HOME/conf下的除broker.conf以外的几个配置文件。所以实际情况中，可以不按这个配置，但是一定要能找到配置文件。

## 3、 快速运行RocketMQ

​ 运行之前，我们需要对RocketMQ的组件结构有个大致的了解。



​ RocketMQ由以下这几个组件组成

* NameServer : 提供轻量级的Broker路由服务。
* Broker：实际处理消息存储、转发等服务的核心组件。
* Producer：消息生产者集群。通常是业务系统中的一个功能模块。
* Consumer：消息消费者集群。通常也是业务系统中的一个功能模块。

所以我们要启动RocketMQ服务，需要先启动NameServer。

### 3.1 启动NameServer

​ 启动NameServer非常简单， 在$ROCKETMQ\_HOME/bin目录下有个mqnamesrv。直接执行这个脚本就可以启动RocketMQ的NameServer服务。

​ 但是要注意，RocketMQ默认预设的JVM内存是4G，这是RocketMQ给我们的最佳配置。但是通常我们用虚拟机的话都是不够4G内存的，所以需要调整下JVM内存大小。[修改的方式是直接修改runserver.sh](http://xn--runserver-z89na9857bcqmtlfda85rmzcf95l5zb.sh/" \t "_blank)。 用vi runserver.sh编辑这个脚本，在脚本中找到这一行调整内存大小为512M

JAVA\_OPT="${JAVA\_OPT} -server -Xms512m -Xmx512m -Xmn256m -

XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"

​ 然后我们用静默启动的方式启动NameServer服务：

nohup bin/mqnamesrv &

​启动完成后，在nohup.out里看到这一条关键日志就是启动成功了。并且使用jps指令可以看到有一个NamesrvStartup进程。

Java HotSpot(TM) 64-Bit Server VM warning: Using the DefNew young collector with the CMS

collector is deprecated **and** will likely be removed **in** a future release

Java HotSpot(TM) 64-Bit Server VM warning: UseCMSCompactAtFullCollection is deprecated **and**

will likely be removed **in** a future release.

The Name Server boot success. serializeType=JSON

### 3.2 启动Broker

​ [启动Broker的脚本是runbroker.sh](http://xn--brokerrunbroker-wy8y53qb44gl6dtn1h9p4b.sh/)。Broker的默认预设内存是8G，启动前，如果内存不够，同样需要调整下JVM内存。vi [runbroker.sh](http://runbroker.sh/)，找到这一行，进行内存调整

JAVA\_OPT="${JAVA\_OPT} -server -Xms512m -Xmx512m -Xmn256m"

​ 然后我们需要找到$ROCKETMQ\_HOME/conf/broker.conf， vi指令进行编辑，在最下面加入一个配置：

autoCreateTopicEnable=true

​ [然后也以静默启动的方式启动runbroker.sh](http://xn--runbroker-zz6n89bg5xa221choab1448bmczanv9bfgw6s1niq8a.sh/)

nohup ./mqbroker &

​ 启动完成后，同样是检查nohup.out日志，有这一条关键日志就标识启动成功了。 并且jps指令可以看到一个BrokerStartup进程。

The broker[worker1, 192.168.232.128:10911] boot success. serializeType=JSON

在观察runserver.sh和runbroker.sh时，我们还可以查看到其他的JVM执行参数，这些参数都可以进行定制。例如我们观察到一个比较有意思的地方，nameServer使用的是CMS垃圾回收器，而Broker使用的是G1垃圾回收器。

### 3.3 命令行快速验证

​ 在RocketMQ的安装包中，提供了一个tools.sh工具可以用来在命令行快速验证RocketMQ服务。我们在worker2上进入RocketMQ的安装目录：

首先需要配置一个环境变量NAMESRV\_ADDR指向我们启动的NameServer服务。

export NAMESRV\_ADDR='localhost:9876'

然后启动消息生产者发送消息：默认会发1000条消息

bin/tools.sh org.apache.rocketmq.example.quickstart.Producer

我们可以看到发送消息的日志：

.....

SendResult [sendStatus=SEND\_OK, msgId=C0A8E88007AC3764951D891CE9A003E7, offsetMsgId=C0A8E88000002A9F00000000000317BF, messageQueue=MessageQueue [topic=TopicTest, brokerName=worker1, queueId=1], queueOffset=249]

14:59:33.418 [NettyClientSelector\_1] INFO RocketmqRemoting - closeChannel: close the connection **to** remote address[127.0.0.1:9876] result: true

14:59:33.423 [NettyClientSelector\_1] INFO RocketmqRemoting - closeChannel: close the connection **to** remote address[192.168.232.128:10911] result: true

这日志中，上面部分就是我们发送的消息的内容。后面两句标识消息生产者正常关闭。

然后启动消息消费者接收消息：

bin/tools.sh org.apache.rocketmq.example.quickstart.Consumer

启动后，可以看到消费到的消息。

......

ConsumeMessageThread\_19 Receive New Messages: [MessageExt [brokerName=worker1, queueId=2, storeSize=203, queueOffset=53, sysFlag=0, bornTimestamp=1606460371999, bornHost=/192.168.232.128:43436, storeTimestamp=1606460372000, storeHost=/192.168.232.128:10911, msgId=C0A8E88000002A9F000000000000A7AE, commitLogOffset=42926, bodyCRC=1968636794, reconsumeTimes=0, preparedTransactionOffset=0, toString()=Message{topic='TopicTest', flag=0, properties={MIN\_OFFSET=0, MAX\_OFFSET=250, CONSUME\_START\_TIME=1606460450150, UNIQ\_KEY=C0A8E88007AC3764951D891CE41F00D4, CLUSTER=DefaultCluster, WAIT=true, TAGS=TagA}, body=[72, 101, 108, 108, 111, 32, 82, 111, 99, 107, 101, 116, 77, 81, 32, 50, 49, 50], transactionId='null'}]]

日志中MessageExt后的整个内容就是一条完整的RocketMQ消息。我们要对这个消息的结构有个大概的了解，后面会对这个消息进行深入的理解。

其中比较关键的属性有：brokerName，queueId，msgId，topic，cluster，tags，body，transactionId。先找下这些属性在哪里。

而这个Consume指令并不会结束，他会继续挂起，等待消费其他的消息。我们可以使用CTRL+C停止该进程。

### 3.4 关闭RocketMQ服务

要关闭RocketMQ服务可以通过mqshutdown脚本直接关闭

**#** 1.关闭NameServer

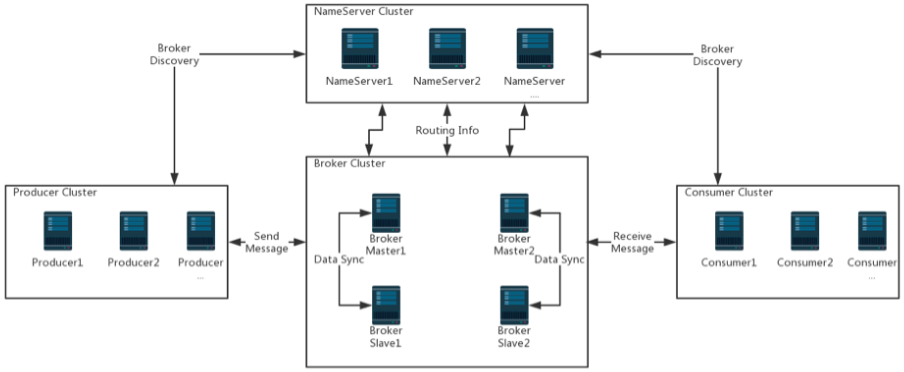
sh bin/mqshutdown namesrv

**#** 2.关闭Broker

sh bin/mqshutdown broker

# 三、RocketMQ集群架构

​ 刚才的演示中，我们已经体验到了RocketMQ是如何工作的。这样，我们回头看RocketMQ的集群架构，就能够有更全面的理解了。



## 1、RocketMQ集群中的各个角色

一个完整的RocketMQ集群中，有如下几个角色

* Producer：消息的发送者；举例：发信者
* Consumer：消息接收者；举例：收信者
* Broker：暂存和传输消息；举例：邮局
* NameServer：管理Broker；举例：各个邮局的管理机构
* Topic：区分消息的种类；一个发送者可以发送消息给一个或者多个Topic；一个消息的接收者可以订阅一个或者多个Topic消息

我们之前的测试案例中，Topic是什么？topic='TopicTest'

现在你能看懂我们之前在broker.conf中添加的autoCreateTopicEnable=true这个属性的用处了吗

* Message Queue：相当于是Topic的分区；用于并行发送和接收消息

在我们之前的测试案例中，一个queueId就代表了一个MessageQueue。有哪些queueId？ 0，1，2，3四个MessageQueue

## 2、RocketMQ集群搭建

为了方便阅读，RocketMQ集群以及RocketMQ配套的管理页面rocketmq-console的搭建写到了另外一个文档中。参见《RocketMQ集群搭建》

## 3、RocketMQ的其他参考资料

在源码中有个docs目录，里面有非常有用的资料。例如，在他的docs/cn/architecture.md文档中，有对RocketMQ架构的更详细的介绍。这里面的内容就不再搬运了

# 总结

​ 到这里，我们可以完整的搭建RocketMQ，并进行简单的使用了。

​ 首先，我们要对MQ的优缺点以及适用场景开始要有逐渐清晰的概念。成熟的MQ产品上手使用都很简单，所以，使用和面试的重点从来都不会是怎么编程，而是能结合项目场景完整落地，这才是考验程序员功力的地方。而这个功力的要点就在于对异步消息驱动场景的理解深度。这一部分的学习最好能够结合kafka、RabbitMQ和RocketMQ这几个产品一起进行横向对比。

​ 然后，我们要对RocketMQ整体的产品架构以及应用生态有个大致的了解。商业版本的RocketMQ提供了购买即用的高可用特性，并且功能也比开源版本略有改进。而在RocketMQ的开源版本之外，围绕RocketMQ的扩展生态包括管理控制台，大都整合在了rocketmq-externals社区项目中。关于RocketMQ的周边生态，其实跟kafka和RabbitMQ还是有差距的，但是RocketMQ相比这两个产品，不管是开发语言还是架构思维，对我们都更为友好，而且周边生态发展也有后发优势