目录

[1 概述 2](#_Toc34400905)

[1.1 集群体系 2](#_Toc34400906)

[1.1.1 提高学员管理效率 2](#_Toc34400907)

[1.1.2 名称服集群务 NameServer cluster 2](#_Toc34400908)

[1.1.3 代理服务集群 Broker Cluster 2](#_Toc34400909)

[1.1.4 生产者集群 Producer Cluster 3](#_Toc34400910)

[1.1.5 消费者集群 Consumer Cluster 3](#_Toc34400911)

[1.2 名称服务NameServer 3](#_Toc34400912)

[1.3 代理服务 broker server 4](#_Toc34400913)

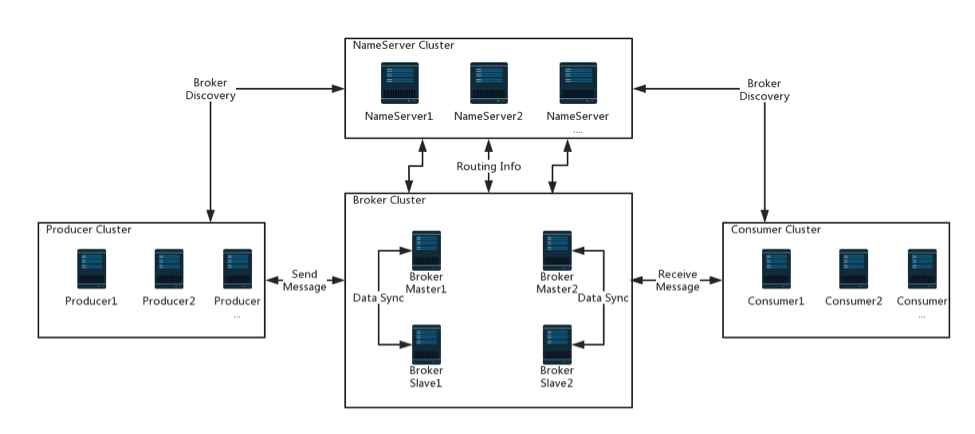
[1.4 部署Deployment 4](#_Toc34400914)

[必要条件Prerequisite 4](#_Toc34400915)

# 概述

Apache RocketMQ是一个具有低延迟、高性能和可靠性、万亿级容量同时具备灵活的可伸缩性的分布式消和流处理平台，它由四个部分组成:name servers, brokers, producers 和 consumers。它们所有都可以水平扩展避免单点故障。

## 系统架构



## 集群体系

### 提高学员管理效率

### 名称服集群务 NameServer cluster

NameServer服务提供了轻量级的服务发现和路由。每个NameServer服务记录完整的路由信息，提供一致的读写服务，支持快速存储扩展

### 代理服务集群 Broker Cluster

Broker通过提供轻量级主题和队列机制来处理消息存储。它们支持Push和Pull模型，包含容错机制(2个副本或3个副本)，提供了极强的峰值处理里能力和按照时间顺序存储数以百万记的消息存储能力，此外，代理提供了灾难恢复、丰富的度量统计和警报机制，这些都是在传统的消息传递系统中缺乏的。

### 生产者集群 Producer Cluster

produce支持分布式部署，分布式的produce通过broker集群提供的各种负载均衡策略将消息发送到broker集群中。发送过程支持快速失败是低延迟的。

### 消费者集群 Consumer Cluster

消费者也支持在推送和者拉取模式下分布式部署，它还支持集群消费和消息广播。提供实时的消息订阅机制，能够满足大多数消费者的需求。RocketMQ的网站为感兴趣的用户提供了一个简单的快速入门指南。

## 名称服务NameServer

NameServer是一个功能齐全的服务器，主要包括两个功能：  
⊙broker 管理，nameserver 接受来自broker集群的注册信息并提供心跳来检测他们是否可用。

⊙路由管理 每一个nameserver都持有关于broker集群和队列的全部路由信息，用来向客户端提供查询。

我们知道 ，rocketMQ客户端（生产者/消费者）会从nameserver查询队列的路由信息，客户端是如何知道nameserver的地址的呢？  
有四种方式能够让客户端湖区到nameserver的地址：  
⊙通过程序，像这样producer.setNamesrvAddr("ip:port")  
⊙java 配置项，这么用rocketmq.namesrv.addr  
⊙环境变量 NAMESRV\_ADDR  
⊙HTTP 端点  
更多关于nameserver地址发现的详细信息请参考[这里](http://rocketmq.apache.org/rocketmq/four-methods-to-feed-name-server-address-list/)

## 代理服务 broker server

broker server负责消息的存储传递，消息查询，保证高可用等等。

像下图所示，broker server有一些非常重要的子模块：  
⊙remoting（远程） 模块，broker的入口，处理从客户端发起的请求。  
⊙client manager（客户端管理） 管理各个客户端（生产者/消费者）还有维护消费者主题订阅。  
⊙store（存储服务），提供简单的api来在磁盘保持或者查询消息。  
⊙HA 高可用服务 提供主从broker的数据同步。  
⊙index(索引服务)为消息建立索引提供消息快速查询。

# 部署Deployment

这一部分介绍生产环境部署方案。概括来说，我们要部署一个没有单点故障的集群。

## 必要条件Prerequisite

在开始之前，确定你已经阅读过快速上手部分，并熟知RocketMq的核心概念和组件。

准备部署Production-ready Deployment

Name Server

为了确保在一个实例崩溃时集群仍然可以运行，建议使用两个或多个名称服务器实例，只要集群中有一个实例是可用的，整个集群就可以提供服务。  
Name Server遵从share-nothing设计方式。Brokers发送心跳数据到所有name server.当要发送或者消费时生产者和消费者可以从任何一台可用的Name server服务查询到信息

broker

Brokers可以按照类别分成两类：master 和slave.master同时提供读写服务，slave只提供读服务。  
要部署一个没有单点故障的高可用集群，需要部署多个broker。一个broker集群需要有一个brokerId为0的master和多个brokerId不为0的slave。这个broker集群的主从需要配置相同的brokerName，极端情况下，我们需要保证一个broker集群中至少部署两台brokers服务，每个topic都存在于两个或多个broker中。

## 配置Configuration

部署一个rocketMQ集群时，我们推荐一下配置：

Broker configuration

| **Property Name 属性名称** | **Default value 默认值** | **Details详细描述** |
| --- | --- | --- |
| listenPort | 10911 | listen port for client 客户端连接的端口 |
| namesrvAddr | null | name server address 名称服务器地址 |
| brokerIP1 | InetAddress for network interface 网址 | Should be configured if having multiple addresses 如果有多个地址需要配置多个 |
| brokerName | null | broker name 代理名称 |
| brokerClusterName | DefaultCluster | this broker belongs to which cluster 代理属于哪个集群 |
| brokerId | 0 | broker id, 0 means master, positive integers mean slave 代理id，0代表主，正数代表从 |
| storePathCommitLog | $HOME/store/commitlog/ | file path for commit log 提交日志的存放路径 |
| storePathConsumerQueue | $HOME/store/consumequeue/ | file path for consume queue 消费队列的存放路径 |
| mapedFileSizeCommitLog | 1024 \* 1024 \* 1024(1G) | mapped file size for commit log 提交日志映射文件大小 |
| deleteWhen | 04 | When to delete the commitlog which is out of the reserve time 何时删除已超出预定时间的commitlog文件 |
| fileReserverdTime | 72 | The number of hours to keep a commitlog before deleting it 删除之前保存多少小时 |
| brokerRole | ASYNC\_MASTER | SYNC\_MASTER/ASYNC\_MASTER/SLVAE 同步主/异步主/从 |
| flushDiskType | ASYNC\_FLUSH | {SYNC\_FLUSH/ASYNC\_FLUSH}. Broker of SYNC\_FLUSH mode flushes each message onto disk before acknowledging producer. Broker of ASYNC\_FLUSH mode, on the other hand, takes advantage of group-committing, achieving better performance.SYNC\_FLUSHASYNC\_FLUSH  同步模式会在响应每次生产者前写入磁盘，异步模式会提高处理生产者组的提交处理能力 |

## 命令行管理工具CLI Admin Tool

RocketMQ提供了一个管理工具，用于查询管理诊断各种问题。这个工具和RocketMQ放到了一起，无论你是下载的是编译好的版本还是自己编译的，你的环境中都已经有了。

**如何使用**

使用管理工具非常简单。为了演示，我们假设你已经在\*nix环境中切换到${PACKAGE}/bin目录，输入 bash mqadmin，你会看到如下帮助菜单

有关特定命令的信息请使用mqadmin help，如果你想知道更多的关于某个命令的信息，比如说clusterList，你只需要输入bash mqadmin help clusterList

## 复制模式

为了确保不会丢失发布成功的消息，RocketMQ提供同步和异步两种复制方式来增强消息的可靠性与高可用性。

**复制：同步/异步 Broker**

像许多复制系统一样，同步复制会等待slave响应提交日志已经被复制，相应的异步复制会在mater节点处理成功后快速返回。

**如何配置**

在conf文件加下默认为rocketMQ提供了三种配置作为参考

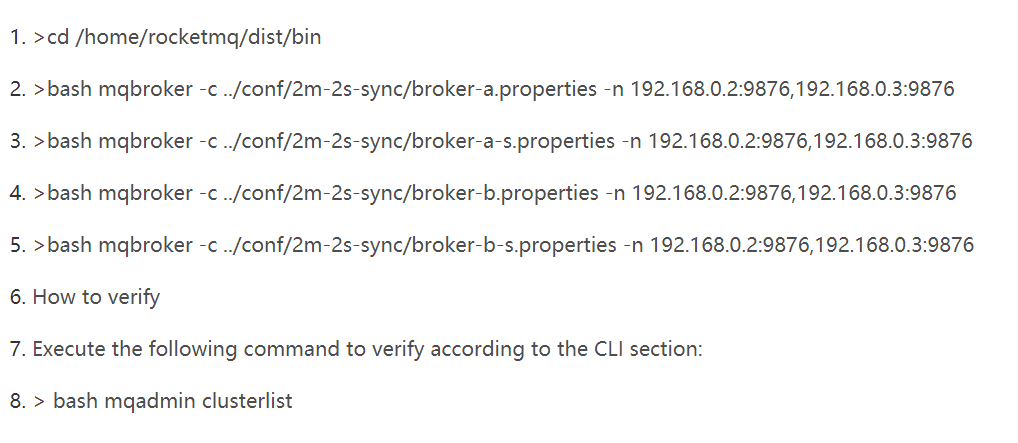
2m-2s-sync

2m-2s-async

2m-noslave  
注意:所有配置都使用ASYNC\_FLUSH

拿2m-2s-sync作为例子部署。首先启动两个name server按照在快速开始部分介绍的。假设他们的ip分别为192.168.0.2 和 192.168.0.3。

然后启动brokers假设编译好的RocketMQ在目录 /home/rocketmq/dist下



# 核心概念Core Concept

根据上面的模型，我们可以深入研究一些关于消息传递系统设计的话题：

1. 消费并发
2. 消费热点
3. 消费均衡
4. 消息路由
5. 多种链接方式
6. 金丝雀部署

## 生产者Producer

生产者发送业务中产生的消息到broker，rocketMQ提供了多种发送方式：同步，异步，单程。

**生产者组Producer Group**

把相同角色的生产者组织到一起。在事务提交后，生产组中的不同实例都可以连接broker执行提交或回滚事务，以防原生产者在提交后就挂掉。

**警告**：考虑到生产者有很强的消息发送能力，每个生产者组只允许有一个实例用来避免不必要的初始化

## 消费者Consumer

消费者从broker拉取消息给应用使用。在用户应用方面，提供了两种类型的消费者。

1. **拉取型PullConsumer**：拉取型主动从broker拉取信息，一次拉取一批，用于消费应用进行消费
2. **推送型PushConsumer** ：推送型是另一种思路，它封装了消息的拉取和消费进程和其他的内部工作细节，只提供一个在消息到达时的回调接口。

**消费者组Consumer Group**

与先前提到的生产者组类似，相同的消费者角色组织到一起起名为消费者组。消费者组是一个棒的概念，它实现了负载平衡和容错的目标，在信息消费方面，非常简单

**注意**：消费者是消费者组中的一个实例，必须订阅相同的主题topic

## 主题 topic

主题是生产者传递消息和消费者拉动消息的类别。topic与生产者与消费者之间是松耦合的。特别强调，一个主题可以有零个一个或者多个生产者向它发送消息。相反，一个生产者可以向多个不同的主题发送消息。从消费者角度讲，一个主题可以被零个或者一个或者多个消费者组订阅，同样的一个消费者组可以订阅一个或多个主题，只要这个消费者组保持一致的订阅即可。

## 消息message

消息是传递的信息。消息必须有一个主题，可以理解为你写信事邮寄的地址。消息也可能有一个标签选项是额外的键值对。例如，你可能发送一条消息时设置一个key标记，并通过这个key在broker中筛选这条消息，用来处理特定的业务。

消息队列 message queue

主题被划分为一个或多个子主题这就是消息队列。

标签tag

标签可以理解为更细一级的主题，为使用者提供更灵活的查找。有了标签同一业务产生的消息可以有相同的主题不同的标签，不同标签标记的可以有不同的用途。标签可以让你的代码变得清晰连贯，还可以给RocketMQ提供更好的查询支持。

代理broker

broker是RockerMQ系统的重要组成部分。它接受并存储来自生产者发送的消息，出来来自消费者的拉取请求。它也存储和消息有关的信息，包括消费者组，消费位置还有主题队列的信息。

**名称服务 nameserver**

名称服务提供路由信息。为生产者和消费者客户端提供一致的主题查找列表。

**消息模式 message model：**

集群clustering

广播broadcasting

## 消息顺序message order

当DefaultMQPushConsumer 被设置好，你可能需要决定消费是顺序的还是并发的

**顺序：**有序的消息意味着消息的使用顺序与生产者为每个消息队列发送的顺序相同。如果你的使用场景要求必是须顺序的，你要确保只用一个队列存放消息。

**警告**：如果消费顺序被指定，最大的消费并发数就是这个消费者组的消息队列的订阅数

**并发：**当消费消息是并发的，最大的消息消费数只受限于每个消费客户端线程池规定的数。

**警告：**这个模式下顺序不在被保证。

## 代理 Broker

### broker 角色

broker角色有ASYNC\_MASTER（异步主）, SYNC\_MASTER（同步主） or SLAVE（从）。如果不能容忍消息丢失，我建议你使用SYNC\_MASTER（同步主）并且再加上一个SLAVE(从)服务的方式部署。如果你可以忍受异常情况下的消息丢失，但是你想broker是高可用的，你可以使用ASYNC\_MASTER（异步主）加SLAVE（从方式部署。如果你想更简单的部署，你可以只部署一个ASYNC\_MASTER（异步主），不部署任何SLAVE（从）。

### 写入磁盘类型FlushDiskType

推荐使用ASYNC\_FLUSH（异步写入），SYNC\_FLUSH（同步写入）代价昂贵，会造成很大的性能浪费。如果你要求可靠性，我们推荐你使用SYNC\_MASTER 同步主并配有一个SLAVE从。

## 生产者Producer

### 发送状态SendStatus

当发送给消息时，你需要获取一个包含发送状态的结果。首先，假设消息的isWaitStoreMsgOK=true（默认配置），我们会在发送没用异常的情况下始终得到发送成功，下边是几个状态的描述

FLUSH\_DISK\_TIMEOUT

如果broker消息存储设置的参数是FlushDiskType=SYNC\_FLUSH(默认是 ASYNC\_FLUSH)，如果broker没用在配置的5秒时间内完成消息的存储，会返回这个状态值。

FLUSH\_SLAVE\_TIMEOUT

如果broker的角色是SYNC\_MASTER(默认 ASYNC\_MASTER)，如果5秒内从broker没用完成同步，会返回这个状态。

SLAVE\_NOT\_AVAILABLE

如果broker的角色是SYNC\_MASTER(默认 ASYNC\_MASTER)，但是没有从被配置，会返回这个状态。

SEND\_OK

SEND\_OK并不意味着它是可靠的，要确保消息不丢失，你需要开启SYNC\_MASTER同步主或者SYNC\_FLUSH同步写。

重发或丢失Duplication or Missing

如果你发送后返回的结果是FLUSH\_DISK\_TIMEOUT, FLUSH\_SLAVE\_TIMEOUT，并且这时broker挂掉了，你会发现刚刚发送的消息丢失了。这种情况下你有两个选择，一、丢就丢吧，可能会导致这个消息的丢失。第二，再发一次，可能会引发消息重复。通常我们建议是重发一次，并找到一个方法去处理发送重复引发的重复消费问题。除非你觉得消息丢失也没什么关系。但是记住当你得到的状态是SLAVE\_NOT\_AVAILABLE，重复发送的意义也不大。如果发生这个响应，应该记录下来，并向集群管理者报警。

### 超时Timeout

客户端发送请求到broker，等待响应，如果经过最大等待时间仍然没等到响应，客户端会抛出一个远程超时异常。默认等待时间是3秒。你可以通过超时时间设置参数来使用send（msg, timeout）方法代替send(msg)方法。记住我们不建议你设置太小的超时时间，broker需要一些时间去写磁盘还有做主从同步。这个值设置的超过syncFlushTimeout 也没什么大的影响，因为broker可能在超时之前已经返回FLUSH\_SLAVE\_TIMEOUT或FLUSH\_SLAVE\_TIMEOUT

### 消息大小 message size

建议大小不要超过512k

异步发送Async sending

默认的发送会阻塞直到响应信息返回。所以如果你关心性能，我们建议你使用，seng(msg,callback)方法，它会异步返回应答信息。

生产者组 produce group

通常来讲，发送者组没什么影响。但是如果你是复杂的事务，你需要关注它。默认在一个jvm中，相同的发送者组中你只能创建一个生产者，一般这种情况就满足需求了。

线程安全 Thread Safety

生产者是线程安全的，你放心的使用就是了。

性能Performance

如果你要使用超过一个生产者在jvm中用于处理大数据，我们建议你：  
⊙用少的生产者异步发送（3-5个足够）。  
⊙为每个生产者设置实例名称。

## 消费者Consumer

### 消费者组和订阅 Consumer Group and Subscriptions

首先你要知道不同的消费者组可以独立的消费相同的topic，每个消费者组都有它们自己的消费记录。请确认相同的消费者组中的消费者要订阅相同的主题。

### 消息监听MessageListener

顺序Orderly

消费者会锁定每个消息队列，确保顺序消费每个消息。这回导致性能的损耗，但是如果你是关心消息消费顺序的这事非常有用的。不推荐抛出异常，推荐使用状态ConsumeOrderlyStatus.SUSPEND\_CURRENT\_QUEUE\_A\_MOMENT代替。

并发Concurrently

同样要说，这种模式是并发消费消息。推荐用于高性能的处理方面。同样不推荐抛出异常，应该用返回ConsumeConcurrentlyStatus.RECONSUME\_LATER代替。

消费状态Consume Status

在并发消息监听中，你可以返回RECONSUME\_LATER 来通知消费者现在还不能消费，需要过会重试。你可以继续消费其他消息。在顺序消息监听中，因为更关心的是顺序，你不能跳过这条消息，但是你可以返回SUSPEND\_CURRENT\_QUEUE\_A\_MOMENT 让消费者等一会。

阻塞Blocking

不推荐阻塞监听者，因为会导致线程池阻塞，甚至可以导致消费过程停止。

### 线程数Thread Number

消费者内部使用线程池来处理消费，你可以通过setConsumeThreadMin 或 setConsumeThreadMax来设置线程数

### 从哪里开始消费ConsumeFromWhere

当新的消费者组建立完成，它需要决定是否消费broker中的历史消息。CONSUME\_FROM\_LAST\_OFFSET 会忽略历史消息，消费这之后的消息。CONSUME\_FROM\_FIRST\_OFFSET 会消费所有在broker中的消息。你也可以使用CONSUME\_FROM\_TIMESTAMP 来指定消费特定时间点之后的消息。

### 重复Duplication

有很多情况导致重复，例如：  
生产者重复发送（比如返回FLUSH\_SLAVE\_TIMEOUT情况）   
消费者还没来得及更新消费位置就挂掉  
如果你的系统不能容忍重复，那你需要做一些处理。例如，你要检查数据库的key。（这里要解决的是幂等性的问题）

## 名称服务NameServer

RockerMQ中，名称服务设计的目的是协调分布式系统中各个部分组件，而协调主要通过管理主题路由信息来实现。  
管理主要有一下两部分构成：  
⊙代理定期更新保存在每个名称服务器中的元数据。  
⊙名称服务器为客户端服务，包括生产者、消费者和命令行客户端提供最新的路由信息。  
因此，在启动代理和客户端之前，我们需要告诉他们如何通过给它们提供一个名称服务器地址列表来访问名称服务器。在Apache RocketMQ中，可以通过四种方式实现：

### 编码方式

在broker中，可以在配置文件中指定namesrvAddr=name-server-ip1:port;name-server-ip2:port  
在生产者和消费者中，我们可以通过以下方式获得名称服务的地址

1. DefaultMQProducer producer = new DefaultMQProducer("please\_rename\_unique\_group\_name");
2. producer.setNamesrvAddr("name-server1-ip:port;name-server2-ip:port");
4. DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("please\_rename\_unique\_group\_name");
5. consumer.setNamesrvAddr("name-server1-ip:port;name-server2-ip:port");

如果你使用的是shell管理工具，你可以通过下面这种方式指定：

sh mqadmin command-name -n name-server-ip1:port;name-server-ip2:port -X OTHER-OPTION

简单的例子是：sh mqadmin -n localhost:9876 clusterList 假设查询集群信息在相同的服务节点。  
如果你将管理工具集成到自己的仪表板中（不太理解），你可以按下方使用：

1. DefaultMQAdminExt defaultMQAdminExt = new DefaultMQAdminExt("please\_rename\_unique\_group\_name");
2. defaultMQAdminExt.setNamesrvAddr("name-server1-ip:port;name-server2-ip:port");

### java 配置项 Java Options

名称服务地址也可以在启动前通过java选项来制定rocketmq.namesrv.addr

### 环境变量Environment Variable

你可以配置NAMESRV\_ADDR环境变量。broker和客户端将检查并使用它.

### http终端

如果您没有使用前面提到的方法指定名称服务器地址列表，Apache RocketMQ将访问以下HTTP端点，每两分钟获取和更新名称服务器地址列表，初始延迟为10秒  
默认的，终端地址是：http://jmenv.tbsite.net:8080/rocketmq/nsaddr  
你可以通过Java 配置项: rocketmq.namesrv.domain覆盖jmenv.tbsite.net，你还可以通过rocketmq.namesrv.domain.subgroup覆盖nsaddr  
如果你在生产环境中使用RockerMQ我们推荐你使用这种方式，因为它给你提供了极大的灵活性，根据名称服务的负载，你可以动态添加和删除名称服务节点而不用重启broker和客户端。

### 优先权

先介绍的方法将优先于后边介绍的方法  
编码方式>java 配置项 >环境变量>http终端

## JVM/Kernel Config

### RocketMQ JVM/Linux 配置项

这部分是配置RocketMQ broker JVM/OS参数的介绍。它指出在部署RocketMQ集群之前应该考虑的某些参数的配置。

### jvm参数

推荐使用jdk1.8最终版，使用服务编译，并配置8g对内存，设置相同的Xms和Xmx值，以防止JVM调整堆大小以获得更好的性能。一个简单的JVM配置是这样的。

-server -Xms8g -Xmx8g -Xmn4g  
如果你不关心broker的启动时间，在jvm初始化时提前处理java堆确保每一页都被分配完毕是更好的方案。不关心启动时间的人可以启用它：

-XX:+AlwaysPreTouch

禁用偏向锁可以减少jvm的暂停时间

-XX:-UseBiasedLocking

jdk1.8推荐使用G1收集器  
-XX:+UseG1GC -XX:G1HeapRegionSize=16m -XX:G1ReservePercent=25 -XX:InitiatingHeapOccupancyPercent=30  
这些gc配置看起来有点激进，但是实践证明它能过带来更好的性能。  
不小把-XX:MaxGCPauseMillis设置的太小，否则JVM将使用小的年轻一代来实现这个目标，这将导致非常频繁的minor GC  
推荐使用滚动GC日志文件。

-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=30m  
如果写GC文件会增加broker的延迟，考虑将GC日志文件重定向到内存文件系统。  
--Xloggc:/dev/shm/mq\_gc\_%p.log

### LINUX内核参数

这有一个os.sh脚本，它列出了很多内核参数，文件位于bin文件夹下，可以稍微修改下用在生产环境。下边的这些参数需要注意，更多详细信息请参考/proc/sys/vm/\*[1].下的文档。  
**vm.extra\_free\_kbytes**

告诉虚拟机在后台回收(kswapd)启动的阈值和直接回收(通过分配进程)启动的阈值之间保持额外的空闲内存。RocketMQ使用这个参数来避免内存分配的高延迟。  
**vm.min\_free\_kbytes**  
如果你将其设置为低于1024KB，您的系统将容易挂掉，并且在高负载下容易产生死锁  
**vm.max\_map\_count**

 限制进程可能拥有的内存映射区域的最大数量。RocketMQ将使用mmap来加载CommitLog和ConsumeQueue，因此建议为该参数设置更大的值。  
**vm.swappiness**  
定义内核如何积极地交换内存页。更高的值会增加一致性，较低的值会减少交换的数量。为了避免交换延迟，建议使用10。  
**File descriptor limits**  
RocketMQ需要为文件(CommitLog和ConsumeQueue)和网络连接打开文件描述符。我们建议为文件描述符设置655350。  
**Disk scheduler**

推荐给RocketMQ的最后期限I/O调度器，试图为请求提供一个有保障的延迟。

# 优化配置

## 内存优化

RocketMQ 的默认内存占用非常高，是4×4g的，普通人是消耗不起的，所以第一件事情就是调整RocketMQ的占用内存。调整RocketMQ的内存目前我所知道的有两个方面

1. MQ的启动脚本可以调整内存
2. mqbroker和mqnamesrv的配置可以调整内存

### 启动脚本的内存调整

启动脚本可以配置MQ整体内存大小，有两个启动脚本，[runbroker.sh](http://runbroker.sh/) [和runserver.sh](http://xn--runserver-pw9o.sh/)。脚本里面的默认配置信息是

JAVA\_OPT="${JAVA\_OPT} -server -Xms4g -Xmx4g -Xmn2g -XX:PermSize=128m -XX:MaxPermSize=320m"

### mqbroker和mqnamesrv的内存调整

这两项配置也很重要，默认是500m到1g。具体文件配置是mqbroker.xml和mqnamesrv.xml。文件里面的配置信息是

<options>

<-Xms200m></-Xms200m>

<-Xmx200m></-Xmx200m>

<-XX:NewSize>50M</-XX:NewSize>

<-XX:MaxNewSize>50M</-XX:MaxNewSize>

<-XX:PermSize>30M</-XX:PermSize>

<-XX:MaxPermSize>30M</-XX:MaxPermSize>

</options>

### broker.conf配置文件内容

brokerClusterName = DefaultCluster

brokerName = broker-a

brokerId = 0

deleteWhen = 04

fileReservedTime = 48

brokerRole = ASYNC\_MASTER

flushDiskType = ASYNC\_FLUSH

autoCreateTopicEnable = true

autoCreateSubscriptionGroup = true

# 应用场景

## 电商

为了更加直观的展示MQ的应用场景，这里我们就用一个常见的电商系统中的几个业务，来具体说明下MQ在实际开发中应用场景。

我们的实际场景大概是一个基于微服务架构的电商系统，分为用户微服务、商品微服务、订单微服务、促销微服务等。基于微服务模式开发的系统，MQ的使用场景更多，下面我们逐一说明：

1、注册后我们可能需要做很多初始化的操作，如：调用邮件服务器发送邮件、调用促销服务赠送优惠劵、下发用户数据到客户关系系统等。那么这时候我们将这些操作去监听MQ，当用户注册成功过后，通过MQ通知其他业务进行操作。确保注册用户的性能。

2、后台发布商品的时候，商品数据需要从数据库中转换成搜索引擎数据（基于elasticsearch），那么我们应该将商品写入数据库后，再写入到MQ，然后通过监听MQ来生成elasticsearch对应的数据。  
3、用户下单后，24小时未支付，需要取消订单。以前我们可能是定时任务循环查询，然后取消订单。实际上，我更推荐类似延迟MQ的方式，避免了很多无效的数据库查询，将一个MQ设置为24小时后才让消费者消费掉，这样很大程度上能减轻服务器压力。  
4、支付完成后，需要及时的通知子系统（进销存系统发货，用户服务积分，发送短信）进行下一步操作，但是，支付回调我们都是需要保证高性能的，所以，我应该直接修改数据库状态，存入MQ，让MQ通知子系统做其他非实时的业务操作。这样能保证核心业务的高效及时。