图灵：楼兰

RocketMQ第五期增强版

一、RocketMQ介绍

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。

1.1、RocketMQ的发展历程

早期阿里使用ActiveMQ，但是，当消息开始逐渐增多后，ActiveMQ的IO性能很快达到了瓶颈。于是，阿里开始关注Kafka。但是Kafka是针对日志收集场景设计的，他的并发性能并不是很理想。尤其当他的Topic过多时，由于Partition文件也会过多，会严重影响IO性能。于是阿里才决定自研中间件，最早叫做MetaQ，后来改名成为RocketMQ。最早他所希望解决的最大问题就是多Topic下的IO性能压力。但是产品在阿里内部的不断改进，RocketMQ开始体现出一些不一样的优势。

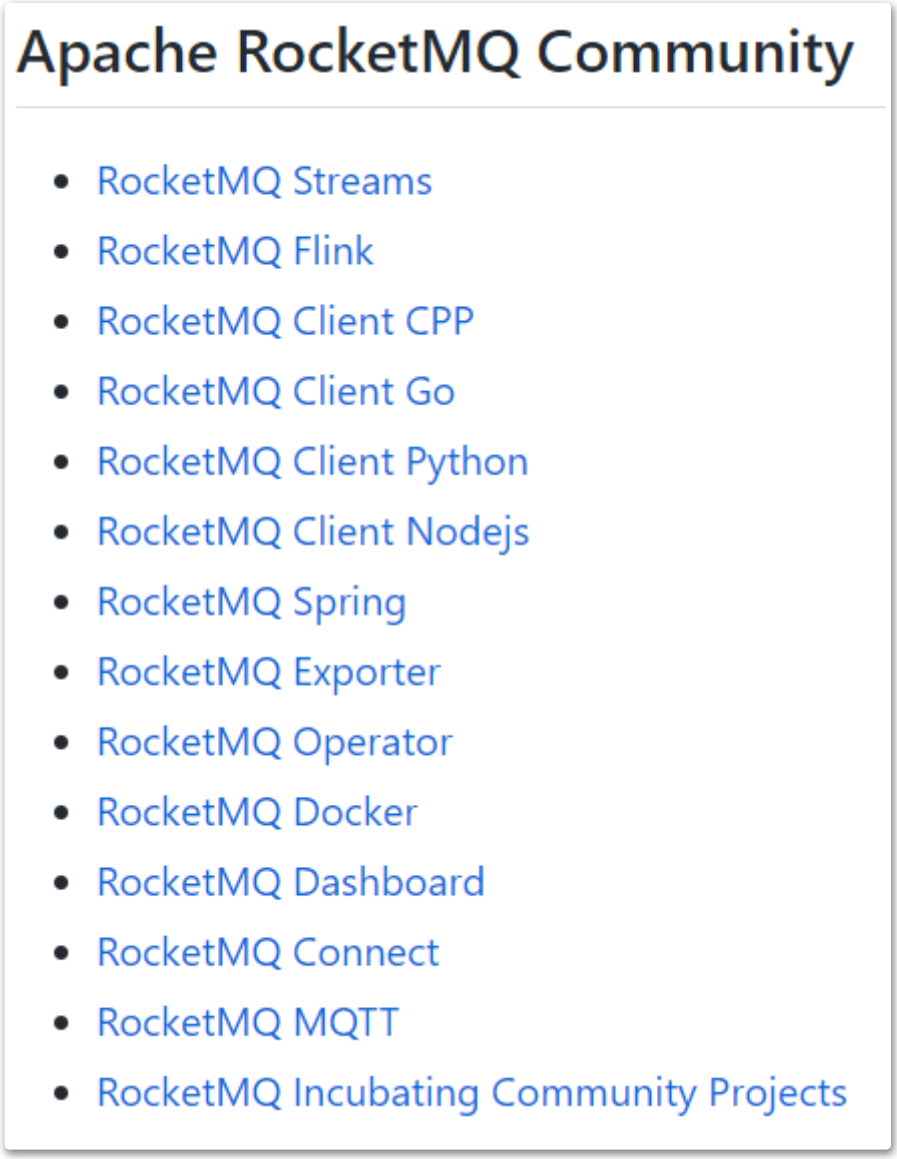
1.2、RocketMQ产品特点比较

RocketMQ的消息吞吐量虽然依然不如Kafka，但是却比RabbitMQ高很多。在阿里内部，RocketMQ集群每天处理的请求数超过5万亿次，支持的核心应用超过3000个。

RocketMQ天生就为金融互联网而生，因此他的消息可靠性相比Kafka也有了很大的提升，而消息吞吐量相比RabbitMQ也有很大的提升。另外，RocketMQ的高级功能也越来越全面，广播消费、延迟队列、死信队列等等高级功能一应俱全，甚至某些业务功能比如事务消息，已经呈现出领先潮流的趋势。

RocketMQ的源码是用Java开发的，这也使得很多互联网公司可以根据自己的业务需求做深度定制。而RocketMQ经过阿里双十一多次考验，源码的稳定性是值得信赖的，这使得功能定制有一个非常高的起点。

传统意义上，RocketMQ有一个比较大的局限，就是他的客户端只支持Java语言。但RocketMQ作为一个开源软件，自身产品不断成熟的同时，周边的技术生态也需要不断演进。RocketMQ成为Apache顶级项目后，又继续通过社区开发出了很多与主流技术生态融合的周边产品。例如在RocketMQ的社区，也正在开发GO，Python，Nodejs等语言的客户端。下图列出了RocketMQ社区目前的一些项目



二、RocketMQ快速实战

RocketMQ的官网地址： http://rocketmq.apache.org ，github地址是 https://github.com/apache/rocketmq ，当前最新的版本是4.9.3。我们这次采用4.9.1版本来学习

2.1、下载RocketMQ

最新版本的RocketMQ可以到官网上进行下载。历史版本需要到Github仓库中下载。下载地址：https://github.com/apache/rocketmq/releases。

目前RocketMQ的历史运行版本已经无法在官网下载，只能通过Github仓库下载历史版本的源码，自行编译。配套资料中提供了4.9.1版本的运行版本以及源码。

2.2、快速安装RocketMQ

RocketMQ的安装非常简单，就是上传解压就可以了。

然后我们准备一台CentOS7的Linux机器，快速把RocketMQ给运行起来。我使用的Linux版本如下：

```
[oper@worker1 jdk1.8]$ uname -a
Linux worker1 3.10.0-1127.el7.x86_64 #1 SMP Tue Mar 31 23:36:51 UTC 2020
x86_64 x86_64 x86_64 GNU/Linux
```

我们需要创建一个操作用户用来运行自己的程序，与root用户区分开。使用root用户创建一个oper用户，并给他创建一个工作目录。

```
[root@worker1 ~]# useradd oper
[root@worker1 ~]# passwd oper
设置用户密码
[root@worker1 ~]# mkdir /app
[root@worker1 ~]# chown oper:oper /app
```

运行RocketMQ需要先安装JDK。我们采用目前最稳定的JDK1.8版本。CentOS可以采用课件资料中的jdk-8u171-linux-x64.tar.gz，也可以自行去Oracle官网上下载。然后用FTP上传到oper用户的工作目录下。由oper用户解压到/app/jdk1.8目录下。

```
[oper@worker1 tools]$ tar -zxvf jdk-8u171-linux-x64.tar.gz
[oper@worker1 tools]$ mv jdk1.8.0_171/ /app/jdk1.8
```

配置环境变量。使用 vi ~/.bash_profile编辑文件，在下面加入以下内容：

```
export JAVA_HOME=/app/jdk1.8/
PATH=$JAVA_HOME/bin:$PATH:$HOME/.local/bin:$HOME/bin
export PATH
```

编辑完成后，执行 source ~/.bash_profile让环境变量生效。输入java -version能查看到以下内容表明JDK安装成功了。

```
[oper@worker1 ~]$ java -version
java version "1.8.0_171"
Java(TM) SE Runtime Environment (build 1.8.0_171-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.171-b11, mixed mode)
```

然后我们把下载的rocketmq-all-4.9.1-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-4.9.1-bin-release
PATH=$ROCKETMQ_HOME/bin:$JAVA_HOME/bin:$PATH:$HOME/.local/bin:$HOME/bin
export PATH
```

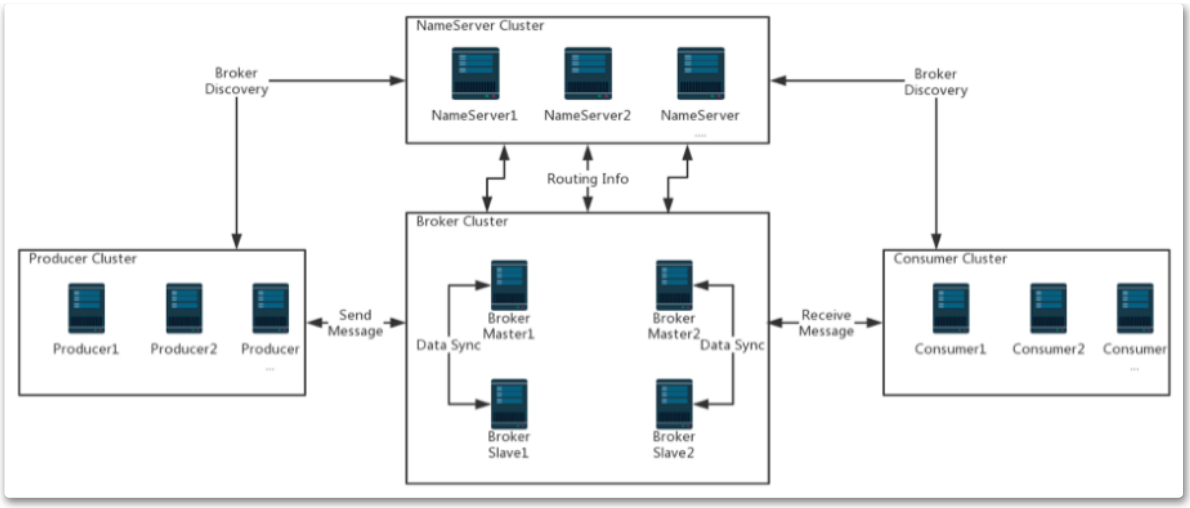
这样RocketMQ就安装完成了。我们把他运行起来。

> 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。

2.3、 快速运行RocketMQ

2.31 RocketMQ工作原理

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。

2.3.2 NameServer服务搭建

启动NameServer非常简单， 在$ROCKETMQ_HOME/bin目录下有个mqadminsrv。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。 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
```

2.3.3 Broker服务搭建

启动Broker的脚本是runbroker.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，找到这一行，进行内存调整

```
JAVA_OPT="${JAVA_OPT} -server -Xms512m -Xmx512m"
```

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

```
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。 关于垃圾回收器的知识你还记得吗?

2.3.3 命令行启动客户端

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。

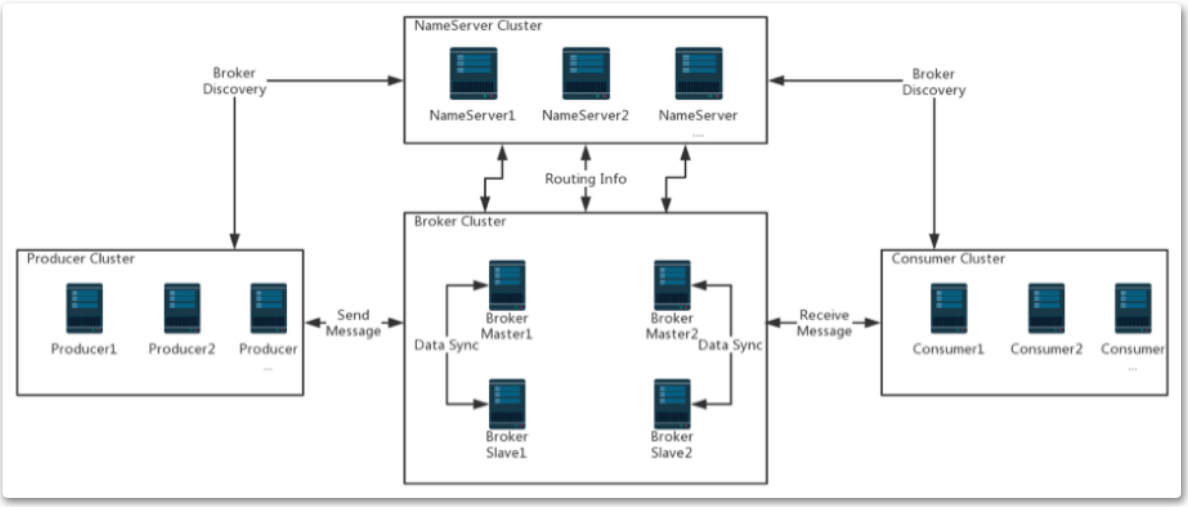
2.3.4 关闭RocketMQ服务

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。



3.1、RocketMQ集群架构解析

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，你都找到了吗？

3.2、RocketMQ集群搭建与优化

3.2.1 实验环境

准备三台虚拟机，硬盘空间建议大于4G。配置机器名。

```
#vi /etc/hosts
192.168.232.128 worker1
192.168.232.129 worker2
192.168.232.130 worker3
```

> 以后我们更多的会关注机器名，而不会太多关注IP地址。

3.2.2 创建用户--可选

useradd oper

passwd oper（密码输入 123qweasd）

3.2.3 系统配置

免密登录

切换oper用户，在worker1上 生成key

```
ssh-kengen
```

然后分发给其他机器

```
ssh-copy-id worker1
ssh-copy-id worker2
ssh-copy-id worker3
```

这样就可以在worker1上直接ssh 或者scp到另外的机器，不需要输密码了。

关闭防火墙

```
systemctl stop firewalld.service
firewall-cmd --state
```

3.2.4 安装JDK1.8 - 略

3.2.5 安装RocketMQ

上传配套的运行包rocketmq-all-4.9.1-bin-release.zip，直接解压到/app/rocketmq目录。然后配置环境变量

```
export ROCKETMQ_HOME=/app/rocketmq/rocketmq-all-4.9.1-bin-release
```

3.2.6 配置RocketMQ主从集群

我们为了便于观察，这次搭建一个2主2从异步刷盘的集群，所以我们会使用conf/2m-2s-async下的配置文件。预备设计的集群情况如下：

机器名	nemaeServer节点部署	broker节点部署
worker1	nameserver	
worker2	nameserver	broker-a, broker-b-s
worker3	nameserver	broker-b,broker-a-s

所以修改的配置文件是进入rocketmq的config目录下修改2m-2s-async的配置文件。主要是配置broker.conf文件。

> 在rocketmq的config目录下可以看到rocketmq建议的各种配置方式：
>
> - 2m-2s-async: 2主2从异步刷盘(吞吐量较大，但是消息可能丢失),

> - 2m-2s-sync:2主2从同步刷盘(吞吐量会下降，但是消息更安全)，
> - 2m-noslave:2主无从(单点故障)，然后还可以直接配置broker.conf，进行单点环境配置。
> - 而dleger就是用来实现主从切换的。集群中的节点会基于Raft协议随机选举出一个leader，其他的就都是follower。通常正式环境都会采用这种方式来搭建集群。

我们这次采用2m-2s-async的方式搭建集群。

1、配置第一组broker-a

在**worker2**上先配置borker-a的master节点。先配置2m-2s-async/broker-a.properties

```
#所属集群名字，名字一样的节点就在同一个集群内
brokerClusterName=rocketmq-cluster
#broker名字，名字一样的节点就是一组主从节点。
brokerName=broker-a
#brokerid,0就表示是Master，>0的都是表示 Slave
brokerId=0
#nameServer地址，分号分割
namesrvAddr=worker1:9876;worker2:9876;worker3:9876
#在发送消息时，自动创建服务器不存在的topic，默认创建的队列数
defaultTopicQueueNums=4
#是否允许 Broker 自动创建Topic，建议线下开启，线上关闭
autoCreateTopicEnable=true
#是否允许 Broker 自动创建订阅组，建议线下开启，线上关闭
autoCreateSubscriptionGroup=true
#Broker 对外服务的监听端口
listenPort=10911
#删除文件时间点，默认凌晨 4点
deleteWhen=04
#文件保留时间，默认 48 小时
fileReservedTime=120
#commitLog每个文件的大小默认1G
mapedFileSizeCommitLog=1073741824
#ConsumeQueue每个文件默认存30W条，根据业务情况调整
mapedFileSizeConsumeQueue=300000
#destroyMapedFileIntervalForcibly=120000
#redeleteHangedFileInterval=120000
#检测物理文件磁盘空间
diskMaxUsedSpaceRatio=88
#存储路径
storePathRootDir=/app/rocketmq/store
#commitLog 存储路径
storePathCommitLog=/app/rocketmq/store/commitlog
```

```
#消费队列存储路径存储路径
storePathConsumeQueue=/app/rocketmq/store/consumequeue
#消息索引存储路径
storePathIndex=/app/rocketmq/store/index
#checkpoint 文件存储路径
storeCheckpoint=/app/rocketmq/store/checkpoint
#abort 文件存储路径
abortFile=/app/rocketmq/store/abort
#限制的消息大小
maxMessageSize=65536
#flushCommitLogLeastPages=4
#flushConsumeQueueLeastPages=2
#flushCommitLogThoroughInterval=10000
#flushConsumeQueueThoroughInterval=60000
#Broker 的角色
#- ASYNC_MASTER 异步复制Master
#- SYNC_MASTER 同步双写Master
#- SLAVE
brokerRole=ASYNC_MASTER
#刷盘方式
#- ASYNC_FLUSH 异步刷盘
#- SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH
#checkTransactionMessageEnable=false
#发消息线程池数量
#sendMessageThreadPoolNums=128
#拉消息线程池数量
#pullMessageThreadPoolNums=128
```

该节点对应的从节点在**worker3**上。修改2m-2s-async/broker-a-s.properties

```
#所属集群名字，名字一样的节点就在同一个集群内
brokerClusterName=rocketmq-cluster
#broker名字，名字一样的节点就是一组主从节点。
brokerName=broker-a
#brokerid,0就表示是Master，>0的都是表示 Slave
brokerId=1
#nameServer地址，分号分割
namesrvAddr=worker1:9876;worker2:9876;worker3:9876
#在发送消息时，自动创建服务器不存在的topic，默认创建的队列数
defaultTopicQueueNums=4
#是否允许 Broker 自动创建Topic，建议线下开启，线上关闭
autoCreateTopicEnable=true
#是否允许 Broker 自动创建订阅组，建议线下开启，线上关闭
autoCreateSubscriptionGroup=true
#Broker 对外服务的监听端口
listenPort=11011
#删除文件时间点，默认凌晨 4点
```

```
deleteWhen=04
#文件保留时间，默认 48 小时
fileReservedTime=120
#commitLog每个文件的大小默认1G
mapedFileSizeCommitLog=1073741824
#ConsumeQueue每个文件默认存30W条，根据业务情况调整
mapedFileSizeConsumeQueue=300000
#destroyMapedFileIntervalForcibly=120000
#redeleteHangedFileInterval=120000
#检测物理文件磁盘空间
diskMaxUsedSpaceRatio=88
#存储路径
storePathRootDir=/app/rocketmq/storeSlave
#commitLog 存储路径
storePathCommitLog=/app/rocketmq/storeSlave/commitlog
#消费队列存储路径存储路径
storePathConsumeQueue=/app/rocketmq/storeSlave/consumequeue
#消息索引存储路径
storePathIndex=/app/rocketmq/storeSlave/index
#checkpoint 文件存储路径
storeCheckpoint=/app/rocketmq/storeSlave/checkpoint
#abort 文件存储路径
abortFile=/app/rocketmq/storeSlave/abort
#限制的消息大小
maxMessageSize=65536
#flushCommitLogLeastPages=4
#flushConsumeQueueLeastPages=2
#flushCommitLogThoroughInterval=10000
#flushConsumeQueueThoroughInterval=60000
#Broker 的角色
#- ASYNC_MASTER 异步复制Master
#- SYNC_MASTER 同步双写Master
#- SLAVE
brokerRole=SLAVE
#刷盘方式
#- ASYNC_FLUSH 异步刷盘
#- SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH
#checkTransactionMessageEnable=false
#发消息线程池数量
#sendMessageThreadPoolNums=128
#拉消息线程池数量
#pullMessageThreadPoolNums=128
```

2、配置第二组Broker-b

这一组broker的主节点在**worker3**上，所以需要配置worker3上的config/2m-2s-async/broker-b.properties

```
#所属集群名字，名字一样的节点就在同一个集群内
brokerClusterName=rocketmq-cluster
#broker名字，名字一样的节点就是一组主从节点。
brokerName=broker-b
#brokerid,0就表示是Master，>0的都是表示 Slave
brokerId=0
#nameServer地址，分号分割
namesrvAddr=worker1:9876;worker2:9876;worker3:9876
#在发送消息时，自动创建服务器不存在的topic，默认创建的队列数
defaultTopicQueueNums=4
#是否允许 Broker 自动创建Topic，建议线下开启，线上关闭
autoCreateTopicEnable=true
#是否允许 Broker 自动创建订阅组，建议线下开启，线上关闭
autoCreateSubscriptionGroup=true
#Broker 对外服务的监听端口
listenPort=10911
#删除文件时间点，默认凌晨 4点
deleteWhen=04
#文件保留时间，默认 48 小时
fileReservedTime=120
#commitLog每个文件的大小默认1G
mapedFileSizeCommitLog=1073741824
#ConsumeQueue每个文件默认存30W条，根据业务情况调整
mapedFileSizeConsumeQueue=300000
#destroyMapedFileIntervalForcibly=120000
#redeleteHangedFileInterval=120000
#检测物理文件磁盘空间
diskMaxUsedSpaceRatio=88
#存储路径
storePathRootDir=/app/rocketmq/store
#commitLog 存储路径
storePathCommitLog=/app/rocketmq/store/commitlog
#消费队列存储路径存储路径
storePathConsumeQueue=/app/rocketmq/store/consumequeue
#消息索引存储路径
storePathIndex=/app/rocketmq/store/index
#checkpoint 文件存储路径
storeCheckpoint=/app/rocketmq/store/checkpoint
#abort 文件存储路径
abortFile=/app/rocketmq/store/abort
#限制的消息大小
maxMessageSize=65536
#flushCommitLogLeastPages=4
#flushConsumeQueueLeastPages=2
#flushCommitLogThoroughInterval=10000
#flushConsumeQueueThoroughInterval=60000
#Broker 的角色
#- ASYNC_MASTER 异步复制Master
#- SYNC_MASTER 同步双写Master
```

```
#- SLAVE
brokerRole=ASYNC_MASTER
#刷盘方式
#- ASYNC_FLUSH 异步刷盘
#- SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH
#checkTransactionMessageEnable=false
#发消息线程池数量
#sendMessageThreadPoolNums=128
#拉消息线程池数量
#pullMessageThreadPoolNums=128
```

然后他对应的slave在worker2上，修改work2上的 conf/2m-2s-async/broker-b-s.properties

```
#所属集群名字，名字一样的节点就在同一个集群内
brokerClusterName=rocketmq-cluster
#broker名字，名字一样的节点就是一组主从节点。
brokerName=broker-b
#brokerid,0就表示是Master，>0的都是表示 Slave
brokerId=1
#nameServer地址，分号分割
namesrvAddr=worker1:9876;worker2:9876;worker3:9876
#在发送消息时，自动创建服务器不存在的topic，默认创建的队列数
defaultTopicQueueNums=4
#是否允许 Broker 自动创建Topic，建议线下开启，线上关闭
autoCreateTopicEnable=true
#是否允许 Broker 自动创建订阅组，建议线下开启，线上关闭
autoCreateSubscriptionGroup=true
#Broker 对外服务的监听端口
listenPort=11011
#删除文件时间点，默认凌晨 4点
deleteWhen=04
#文件保留时间，默认 48 小时
fileReservedTime=120
#commitLog每个文件的大小默认1G
mapedFileSizeCommitLog=1073741824
#ConsumeQueue每个文件默认存30W条，根据业务情况调整
mapedFileSizeConsumeQueue=300000
#destroyMapedFileIntervalForcibly=120000
#redeleteHangedFileInterval=120000
#检测物理文件磁盘空间
diskMaxUsedSpaceRatio=88
#存储路径
storePathRootDir=/app/rocketmq/storeSlave
#commitLog 存储路径
storePathCommitLog=/app/rocketmq/storeSlave/commitlog
#消费队列存储路径存储路径
```

```
storePathConsumeQueue=/app/rocketmq/storeSlave/consumequeue
#消息索引存储路径
storePathIndex=/app/rocketmq/storeSlave/index
#checkpoint 文件存储路径
storeCheckpoint=/app/rocketmq/storeSlave/checkpoint
#abort 文件存储路径
abortFile=/app/rocketmq/storeSlave/abort
#限制的消息大小
maxMessageSize=65536
#flushCommitLogLeastPages=4
#flushConsumeQueueLeastPages=2
#flushCommitLogThoroughInterval=10000
#flushConsumeQueueThoroughInterval=60000
#Broker 的角色
#- ASYNC_MASTER 异步复制Master
#- SYNC_MASTER 同步双写Master
#- SLAVE
brokerRole=SLAVE
#刷盘方式
#- ASYNC_FLUSH 异步刷盘
#- SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH
#checkTransactionMessageEnable=false
#发消息线程池数量
#sendMessageThreadPoolNums=128
#拉消息线程池数量
#pullMessageThreadPoolNums=128
```

这样2主2从的集群配置基本就完成了。搭建过程中需要注意的配置项：

1、同一机器上两个实例的store目录不能相同，否则会报错 Lock failed,MQ already started
2、同一机器上两个实例的listenPort也不能相同。否则会报端口占用的错
nameserver不需要进行配置，直接启动就行。这也看出nameserver是无状态的。
3、**如果是多网卡的机器，比如云服务器，那么需要在broker.conf中增加brokerIP1属性，指定所在机器的外网网卡地址。**

3.2.7 启动RocketMQ

启动就比较简单了，直接调用bin目录下的脚本就行。只是启动之前要注意看下他们的JVM内存配置，默认的配置都比较高。

1、先启动nameServer

修改三个节点上的bin/runserver.sh，调整里面的jvm内存配置。找到下面这一行调整下内存

```
JAVA_OPT="${JAVA_OPT} -server -Xms512m -Xmx512m -Xmn256m -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
```

直接在三个节点上启动nameServer。

```
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。

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

使用jps指令可以看到一个NamesrvStartup进程。

> 这里也看到，RocketMQ在runserver.sh中是使用的CMS垃圾回收期，而在runbroker.sh中使用的是G1垃圾回收期。

2、再启动broker

启动broker是使用的mqbroker指令，只是注意启动broker时需要通过-c 指定对应的配置文件。

在**worker2**上启动broker-a的master节点和broker-b的slave节点

```
nohup ./mqbroker -c ../conf/2m-2s-async/broker-a.properties &
nohup ./mqbroker -c ../conf/2m-2s-async/broker-b-s.properties &
```

在work3上启动broker-b的master节点和broker-a的slave节点

```
nohup ./mqbroker -c ../conf/2m-2s-async/broker-b.properties &
nohup ./mqbroker -c ../conf/2m-2s-async/broker-a-s.properties &
```

> 启动slave时，如果遇到报错 Lock failed,MQ already started ，那是因为有多个实例共用了同一个storePath造成的，这时就需要调整store的路径。

3、启动状态检查

使用jps指令，能看到一个NameSrvStartup进程和两个BrokerStartup进程。

nohup.out中也有启动成功的日志。

对应的日志文件：

```
# 查看nameServer日志
tail -500f ~/logs/rocketmqlogs/namesrv.log
# 查看broker日志
tail -500f ~/logs/rocketmqlogs/broker.log
```

4、测试mqadmin管理工具

RocketMQ源码中并没有提供管理控制台，只提供了一个mqadmin指令来管理RocketMQ。指令的位置在bin目录下。直接使用该指令就会列出所有支持的命令。

```
loper@worker1 conf]$ mqadmin
The most commonly used mqadmin commands are:
   updateTopic          Update or create topic
   deleteTopic          Delete topic from broker and NameServer.
   updateSubGroup       Update or create subscription group
   deleteSubGroup       Delete subscription group from broker.
   updateBrokerConfig   Update broker's config
   updateTopicPerm      Update topic perm
   topicRoute           Examine topic route info
   topicStatus          Examine topic Status info
   topicClusterList     get cluster info for topic
   brokerStatus         Fetch broker runtime status data
   queryMsgById         Query Message by Id
   queryMsgByKey        Query Message by Key
   queryMsgByUniqueKey  Query Message by Unique key
   queryMsgByOffset     Query Message by offset
   printMsg             Print Message Detail
   printMsgByQueue      Print Message Detail
   sendMsgStatus        send msg to broker.
   brokerConsumeStats   Fetch broker consume stats data
   producerConnection   Query producer's socket connection and client version
   consumerConnection   Query consumer's socket connection, client version and subscription
   consumerProgress     Query consumers's progress, speed
   consumerStatus       Query consumer's internal data structure
   cloneGroupOffset     clone offset from other group.
   clusterList          List all of clusters
   topicList            Fetch all topic list from name server
   updateKvConfig       Create or update KV config.
   deleteKvConfig       Delete KV config.
   wipeWritePerm        Wipe write perm of broker in all name server
   resetOffsetByTime    Reset consumer offset by timestamp(without client restart).
   updateOrderConf      Create or update or delete order conf
   cleanExpiredCQ       Clean expired ConsumeQueue on broker.
   cleanUnusedTopic     Clean unused topic on broker.
   startMonitoring      Start Monitoring
   statsAll             Topic and Consumer tps stats
   allocateMQ           Allocate MQ
   checkMsgSendRT       check message send response time
   clusterRT            List All clusters Message Send RT
   getNamesrvConfig     Get configs of name server.
   updateNamesrvConfig  Update configs of name server.
   getBrokerConfig      Get broker config by cluster or special broker!
   queryCq              Query cq command.
   sendMessage          Send a message
   consumeMessage       Consume message
   updateAclConfig      Update acl config yaml file in broker
   deleteAccessConfig   Delete Acl Config Account in broker
```

使用方式都是 **mqadmin {command} {args}。** 如果有某个指令不会使用，可以使用 **mqadmin help {command}** 指令查看帮助。

5、命令行快速验证

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

发送消息：默认会发1000条消息

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

接收消息：

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

> 注意，这是官方提供的Demo，但是官方的源码中，这两个类都是没有指定nameServer的，所以运行会有点问题。要指定NameServer地址，可以配置一个环境变量NAMESRV_ADDR，这样默认会读取这个NameServer地址。可以配到.bash_profile里或者直接临时指定。
>
> ```
> export NAMESRV_ADDR='worker1:9876;worker2:9876;worker3:9876'
> ```
>
> 然后就可以正常执行了。

这个tooles.sh实际上是封装了一个简单的运行RocketMQ的环境，上面指令中指定的Java类，都在lib/rocketmq-example-4.7.1.jar包中。未来如果自己有一些客户端示例，也可以打成jar包放到这个lib目录下，通过tools.sh运行。

3.2.8 搭建管理控制台

RocketMQ源代码中并没有提供控制台，但是有一个Rocket的社区扩展项目中提供了一个控制台，地址： https://github.com/apache/rocketmq-dashboard

下载下来后，解压并进入对应的目录，使用maven进行编译

```
mvn clean package -Dmaven.test.skip=true
```

编译完成后，获取target下的jar包，就可以直接执行。但是这个时候要注意，在这个项目的application.yml中需要指定nameserver的地址。默认这个属性是指向本地。如果配置为空，会读取环境变量NAMESRV_ADDR。

那我们可以在jar包的当前目录下增加一个application.yml文件，覆盖jar包中默认的一个属性：

```
rocketmq:
  config:
    namesrvAddrs:
      - worker1:9876
      - worker2:9876
      - worker3:9876
```

> 其他更多配置信息可以参考dashboard源码中的配置文件。

然后执行：

```
java -jar rocketmq-dashboard-1.0.1-SNAPSHOT.jar
```

启动完成后，可以访问 http://192.168.232.128:8080看到管理页面

3.2.9 搭建Dledger高可用集群--了解

通过这种方式，我们搭建了一个主从结构的RocketMQ集群，但是我们要注意，这种主从结构是只做数据备份，没有容灾功能的。也就是说当一个master节点挂了后，slave节点是无法切换成master节点继续提供服务的。注意这个集群至少要是3台，允许少于一半的节点发生故障。

> 如果slave挂了，对集群的影响不会很大，因为slave只是做数据备份的。但是影响也是会有的，例如，当消费者要拉取的数据量比较大时，RocketMQ有一定的机制会优先保证Master节点的性能，只让Master节点返回一小部分数据，而让其他部分的数据从slave节点去拉取。
>
> 另外，需要注意，Dleger会有他自己的CommitLog机制，也就是说，使用主从集群累计下来的消息，是无法转移到Dleger集群中的。

而如果要进行高可用的容灾备份，需要采用Dledger的方式来搭建高可用集群。注意，这个Dledger需要在RocketMQ4.5以后的版本才支持，我们使用的4.7.1版本已经默认集成了dledger。

搭建方法

要搭建高可用的Broker集群，我们只需要配置conf/dleger下的配置文件就行。

> 这种模式是基于Raft协议的，是一个类似于Zookeeper的paxos协议的选举协议，也是会在集群中随机选举出一个leader，其他的就是follower。只是他选举的过程跟paxos有点不同。Raft协议基于随机休眠机制的，选举过程会比paxos相对慢一点。

首先：我们同样是需要修改runserver.sh和runbroker.sh，对JVM内存进行定制。

然后：我们需要修改conf/dleger下的配置文件。 跟dleger相关的几个配置项如下：

name	含义	举例
enableDLegerCommitLog	是否启动 DLedger	true
	DLedger Raft Group的名	

name	含义	举例
dLegerGroup	DLedger Raft Group的名字，建议和 brokerName 保持一致	RaftNode00
dLegerPeers	DLedger Group 内各节点的端口信息，同一个 Group 内的各个节点配置必须要保证一致	n0-127.0.0.1:40911;n1-127.0.0.1:40912;n2-127.0.0.1:40913
dLegerSelfId	节点 id, 必须属于 dLegerPeers 中的一个；同 Group 内各个节点要唯一	n0
sendMessageThreadPoolNums	发送线程个数，建议配置成 Cpu 核数	16

配置完后，同样是使用 nohup bin/mqbroker -c $conf_name & 的方式指定实例文件。

> 在bin/dleger下有个fast-try.sh，这个脚本是在本地启动三个RocketMQ实例，搭建一个高可用的集群，读取的就是conf/dleger下的broker-no.conf，broker-n1.conf和broker-n2.conf。使用这个脚本同样要注意定制下JVM内存，他给每个实例默认定制的是1G内存，虚拟机肯定是不够的。
>
> 这种单机三实例的集群搭建完成后，可以使用 bin/mqadmin clusterList -n worker1.conf的方式查看集群状态。

3.2.10 系统参数调优 -- 重要

到这里，我们的整个RocketMQ的服务就搭建完成了。但是在实际使用时，我们说RocketMQ的吞吐量、性能都很高，那要发挥RocketMQ的高性能，还需要对RocketMQ以及服务器的性能进行定制

1、配置RocketMQ的JVM内存大小：

之前提到过，在runserver.sh中需要定制nameserver的内存大小，在runbroker.sh中需要定制broker的内存大小。这些默认的配置可以认为都是经过检验的最优化配置，但是在实际情况中都还需要根据服务器的实际情况进行调整。这里以runbroker.sh中对G1GC的配置举例，在runbroker.sh中的关键配置：

```
JAVA_OPT="${JAVA_OPT} -XX:+UseG1GC -XX:G1HeapRegionSize=16m -XX:G1ReservePercent=25 -XX:InitiatingHeapOccupancyPercent=30 -XX:SoftRefLRUPolicyMSPerMB=0"
JAVA_OPT="${JAVA_OPT} -verbose:gc -Xloggc:${GC_LOG_DIR}/rmq_broker_gc_%p_%t.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCApplicationStoppedTime -XX:+PrintAdaptiveSizePolicy"
JAVA_OPT="${JAVA_OPT} -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=30m"
```

-XX:+UseG1GC: 使用G1垃圾回收器， -XX:G1HeapRegionSize=16m 将G1的region块大小设为16M，-XX:G1ReservePercent：在G1的老年代中预留25%空闲内存，这个默认值是10%，RocketMQ把这个参数调大了。-XX:InitiatingHeapOccupancyPercent=30：当堆内存的使用率达到30%之后就会启动G1垃圾回收器尝试回收垃圾，默认值是45%，RocketMQ把这个参数调小了，也就是提高了GC的频率，但是避免了垃圾对象过多，一次垃圾回收时间太长的问题。

然后，后面定制了GC的日志文件，确定GC日志文件的地址、打印的内容以及控制每个日志文件的大小为30M并且只保留5个文件。这些在进行性能检验时，是相当重要的参考内容。

2、RocketMQ的其他一些核心参数

例如在conf/dleger/broker-n0.conf中有一个参数：sendMessageThreadPoolNums=16。这一个参数是表明RocketMQ内部用来发送消息的线程池的线程数量是16个，其实这个参数可以根据机器的CPU核心数进行适当调整，例如如果你的机器核心数超过16个，就可以把这个参数适当调大。

3、Linux内核参数定制

我们在部署RocketMQ的时候，还需要对Linux内核参数进行一定的定制。例如

- **ulimit**，需要进行大量的网络通信和磁盘IO。
- **vm.extra_free_kbytes**，告诉VM在后台回收（kswapd）启动的阈值与直接回收（通过分配进程）的阈值之间保留额外的可用内存。RocketMQ使用此参数来避免内存分配中的长延迟。（与具体内核版本相关）
- **vm.min_free_kbytes**，如果将其设置为低于1024KB，将会巧妙的将系统破坏，并且系统在高负载下容易出现死锁。
- **vm.max_map_count**，限制一个进程可能具有的最大内存映射区域数。RocketMQ将使用mmap加载CommitLog和ConsumeQueue，因此建议将为此

参数设置较大的值。

- **vm.swappiness**，定义内核交换内存页面的积极程度。较高的值会增加攻击性，较低的值会减少交换量。建议将值设置为10来避免交换延迟。
- **File descriptor limits**，RocketMQ需要为文件（CommitLog和ConsumeQueue）和网络连接打开文件描述符。我们建议设置文件描述符的值为655350。

> 这些参数在CentOS7中的配置文件都在 /proc/sys/vm目录下。
>
> 另外，RocketMQ的bin目录下有个os.sh里面设置了RocketMQ建议的系统内核参数，可以根据情况进行调整。

四、RocketMQ消息转发模型

这一部分我们可以结合一下管理控制台，先来理解下RocketMQ的一些重要的基础概念：

1 消息模型（Message Model)

RocketMQ主要由 Producer、Broker、Consumer 三部分组成，其中Producer负责生产消息，Consumer 负责消费消息，Broker 负责存储消息。Broker 在实际部署过程中对应一台服务器，每个 Broker 可以存储多个Topic的消息，每个Topic的消息也可以分片存储于不同的 Broker。Message Queue 用于存储消息的物理地址，每个Topic中的消息地址存储于多个 Message Queue 中。ConsumerGroup由多个Consumer 实例构成。

2 消息生产者（Producer）

负责生产消息，一般由业务系统负责生产消息。一个消息生产者会把业务应用系统里产生的消息发送到broker服务器。RocketMQ提供多种发送方式，同步发送、异步发送、顺序发送、单向发送。同步和异步方式均需要Broker返回确认信息，单向发送不需要。

生产者中，会把同一类Producer组成一个集合，叫做生产者组。同一组的Producer被认为是发送同一类消息且发送逻辑一致。

3 消息消费者（Consumer）

负责消费消息，一般是后台系统负责异步消费。一个消息消费者会从Broker服务器拉取消息、并将其提供给应用程序。从用户应用的角度而言提供了两种消费形式：拉取式消费、推动式消费。

- 拉取式消费的应用通常主动调用Consumer的拉消息方法从Broker服务器拉消息、主动权由应用控制。一旦获取了批量消息，应用就会启动消费过程。
- 推动式消费模式下Broker收到数据后会主动推送给消费端，该消费模式一般实时性较高。

消费者同样会把同一类Consumer组成一个集合，叫做消费者组，这类Consumer通常消费同一类消息且消费逻辑一致。消费者组使得在消息消费方面，实现负载均衡和容错的目标变得非常容易。要注意的是，消费者组的消费者实例必须订阅完全相同的Topic。RocketMQ 支持两种消息模式：集群消费（Clustering）和广播消费（Broadcasting）。

- 集群消费模式下，相同Consumer Group的每个Consumer实例平均分摊消息。
- 广播消费模式下，相同Consumer Group的每个Consumer实例都接收全量的消息。

4 主题（Topic）

表示一类消息的集合，每个主题包含若干条消息，每条消息只能属于一个主题，是RocketMQ进行消息订阅的基本单位。

Topic只是一个逻辑概念，并不实际保存消息。同一个Topic下的消息，会分片保存到不同的Broker上，而每一个分片单位，就叫做MessageQueue。MessageQueue是一个具有FIFO特性的队列结构，生产者发送消息与消费者消费消息的最小单位。

5 代理服务器（Broker Server）

消息中转角色，负责存储消息、转发消息。代理服务器在RocketMQ系统中负责接收从生产者发送来的消息并存储、同时为消费者的拉取请求作准备。代理服务器也存储消息相关的元数据，包括消费者组、消费进度偏移和主题和队列消息等。

Broker Server是RocketMQ真正的业务核心，包含了多个重要的子模块：

- 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。

而Broker Server要保证高可用需要搭建主从集群架构。RocketMQ中有两种Broker架构模式：

- 普通集群：

这种集群模式下会给每个节点分配一个固定的角色，master负责响应客户端的请求，并存储消息。slave则只负责对master的消息进行同步保存，并响应部分客户端的读请求。消息同步方式分为同步同步和异步同步。

这种集群模式下各个节点的角色无法进行切换，也就是说，master节点挂了，这一组Broker就不可用了。

- Dledger高可用集群：

Dledger是RocketMQ自4.5版本引入的实现高可用集群的一项技术。这个模式下的集群会随机选出一个节点作为master，而当master节点挂了后，会从slave中自动选出一个节点升级成为master。

Dledger技术做的事情：1、从集群中选举出master节点 2、完成master节点往slave节点的消息同步。

6 名字服务（Name Server）

名称服务充当路由消息的提供者。Broker Server会在启动时向所有的Name Server注册自己的服务信息，并且后续通过心跳请求的方式保证这个服务信息的实时性。生产者或消费者能够通过名字服务查找各主题相应的Broker IP列表。多个Namesrv实例组成集群，但相互独立，没有信息交换。

这种特性也就意味着NameServer中任意的节点挂了，只要有一台服务节点正常，整个路由服务就不会有影响。当然，这里不考虑节点的负载情况。

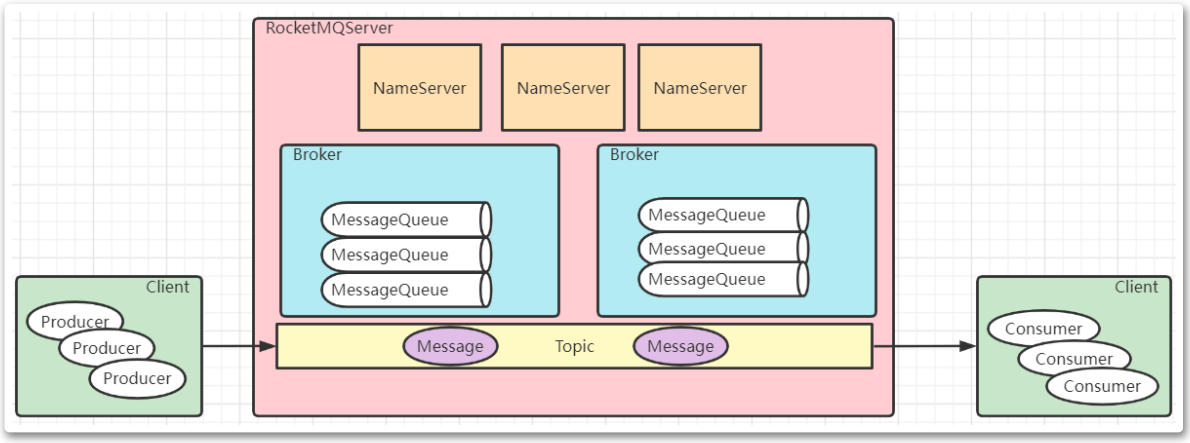
7 消息（Message）

消息系统所传输信息的物理载体，生产和消费数据的最小单位，每条消息必须属于一个主题Topic。RocketMQ中每个消息拥有唯一的Message ID，且可以携带具有业务标识的Key。系统提供了通过Message ID和Key查询消息的功能。

并且Message上有一个为消息设置的标志，Tag标签。用于同一主题下区分不同类型的消息。来自同一业务单元的消息，可以根据不同业务目的在同一主题下设置不同标签。标签能够有效地保持代码的清晰度和连贯性，并优化RocketMQ提供的查询系统。消费者可以根据Tag实现对不同子主题的不同消费逻辑，实现更好的扩展性。

> 关于Message的更详细字段，在源码的docs/cn/best_practice.md中有详细介绍。

整体的基础概念如下图总结：



其中有一些比较常见的面试题需要注意一下：

1、为什么RocketMQ不用Zookeeper而要自己实现一个NameServer来进行注册？

2、Consumer分组有什么用？Producer分组呢？

3、RocketMQ如何保证集群高可用？

> 有道云笔记链接：
>
> 文档：五期VIP01-RocketMQ快速实战以及集群架...
> 链接：http://note.youdao.com/noteshare?id=fa3c32cea984593b3c8cf5faeb16e634&sub=E3E504F2D31A4558A6E2AAFB83BE9C76