图灵：楼兰
你的神秘技术宝藏

快速搭建RocketMQ集群

1、机器环境

准备三台虚拟机，root密码 root ;IP地址:

```
192.168.232.128 worker1
192.168.232.129 worker2
192.168.232.130 worker3
```

> 这里特意不把每个机器的机器名定义得太过规范，比如master slave这样的，有助于更理解各项配置。

2、创建用户

useradd oper

passwd oper（密码输入 123qweasd)

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
```

4、安装java

给oper创建/app目录

上传jdk的tar包

修改~/.bash_profile，配置环境变量。source生效。

```
export JAVA_HOME=/app/jdk1.8/
```

5、安装RocketMQ

上传tar包，直接解压。然后配置环境变量

```
export ROCKETMQ_HOME=/app/rocketmq/rocketmq-all-4.7.1-bin-release
```

> RocketMQ在4.5版本之前都不支持master宕机后slave自动切换。在4.5版本后，增加了基于Dleger实现的主从切换。这里用的目前最新的4.7.1版本

6、配置RocketMQ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，实际项目中，为了达到高可用，一般会使用dleger。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。--只需要配置broker.conf。

> 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 `只需要修改brokerId和brokerRole`

```
#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broker就配置完成了。
>
> 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、其他的配置项参见《RcoketMQ全部配置表.pdf》

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/mqadminsrv &
```

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的源代码中并没有为我们提供类似于Nacos或者RabbitMQ那样的控制台，只提供了一个mqadmin指令来管理RocketMQ，命令在bin目录下。使用方式是 ./mqadmin {command} {args}

所有指令如下：

Topic相关：

名称	含义	命令选项	说明
updateTopic	创建更新Topic配置	-b	Broker 地址，表示 topic 所在 Broker，只支持单台Broker，地址为ip:port
		-c	cluster 名称，表示 topic 所在集群（集群可通过 clusterList 查询）
		-h-	打印帮助
		-n	NameServer服务地址，格式 ip:port
		-p	指定新topic的读写权限(W=2|R=4|WR=6)
		-r	可读队列数（默认为 8）
		-w	可写队列数（默认为 8）
		-t	topic 名称（名称只能使用字符 ^[a-zA-Z0-9_-]+$ ）
deleteTopic	删除Topic	-c	cluster 名称，表示删除某集群下的某个 topic （集群 可通过 clusterList 查询）
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-t	topic 名称（名称只能使用字符 ^[a-zA-Z0-9_-]+$ ）
topicList	查看 Topic 列表信息	-h	打印帮助

		-c	不配置-c只返回topic列表，增加-c返回clusterName, topic, consumerGroup信息，即topic的所属集群和订阅关系，没有参数
		-n	NameServer 服务地址，格式 ip:port
topicRoute	查看 Topic 路由信息	-t	topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
topicStatus	查看 Topic 消息队列 offset	-t	topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
topicClusterList	查看 Topic 所在集群列表	-t	topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
updateTopicPerm	更新 Topic 读写权限	-t	topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-b	Broker 地址，表示 topic 所在 Broker，只支持单台Broker，地址为ip:port
		-p	指定新 topic 的读写权限(W=2|R=4|WR=6)

		-c	cluster 名称，表示 topic 所在集群（集群可通过 clusterList 查询），-b优先，如果没有-b，则对集群中所有Broker执行命令
updateOrderConf	从NameServer上创建、删除、获取特定命名空间的kv配置，目前还未启用	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-t	topic，键
		-v	orderConf，值
		-m	method，可选get、put、delete
allocateMQ	以平均负载算法计算消费者列表负载消息队列的负载结果	-t	topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-i	ipList，用逗号分隔，计算这些ip去负载Topic的消息队列
statsAll	打印Topic订阅关系、TPS、积累量、24h读写总量等信息	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-a	是否只打印活跃topic
		-t	指定topic

集群相关

名称	含义	命令选项	说明
clusterList	查看集群信息，集群、BrokerName、BrokerId、TPS等信息	-m	打印更多信息 (增加打印出如下信息 #InTotalYest, #OutTotalYest, #InTotalToday ,#OutTotalToday)
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-i	打印间隔，单位秒
clusterRT	发送消息检测集群各 Broker RT。消息发往 ${BrokerName} Topic。	-a	amount，每次探测的总数，RT = 总时间 / amount
		-s	消息大小，单位B
		-c	探测哪个集群
		-p	是否打印格式化日志，以|分割，默认不打印
		-h	打印帮助
		-m	所属机房，打印使用
		-i	发送间隔，单位秒
		-n	NameServer 服务地址，格式 ip:port

Broker相关

名称	含义	命令选项	说明
updateBrokerConfig	更新 Broker 配置文件，会修改 Broker.conf	-b	Broker 地址，格式为 ip:port
		-c	cluster 名称
		-k	key 值
		-v	value 值
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
brokerStatus	查看 Broker 统计信息、运行状态（你想要的信息几乎都在里面）	-b	Broker 地址，地址为 ip:port
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
brokerConsumeStats	Broker中各个消费者的消费情况，按Message Queue维度返回Consume Offset，Broker Offset，Diff，TImestamp等信息	-b	Broker 地址，地址为 ip:port
		-t	请求超时时间
		-l	diff阈值，超过阈值才打印
		-o	是否为顺序topic，一般为false
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
getBrokerConfig	获取Broker配置	-b	Broker 地址，地址为 ip:port
		-n	NameServer 服务地址，格式 ip:port
wipeWritePerm	从NameServer上清除 Broker写权限	-b	Broker 地址，地址为 ip:port
		-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
cleanExpiredCQ	清理Broker上过期的Consume Queue，如果手动	-n	NameServer 服务地址，格式 ip:port

	减少对列数可能产生过期队列	-h	打印帮助
		-b	Broker 地址，地址为 ip:port
		-c	集群名称
cleanUnusedTopic	清理Broker上不使用的Topic，从内存中释放Topic的 Consume Queue，如果手动删除Topic会产生不使用的 Topic	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
		-b	Broker 地址，地址为 ip:port
		-c	集群名称
sendMsgStatus	向Broker发消息，返回发送状态和RT	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
		-b	BrokerName，注意不同于Broker地址
		-s	消息大小，单位B
		-c	发送次数

消息相关

名称	含义	命令选项	说明
queryMsgById	根据offsetMsgId查询msg，如果使用开源控制台，应使用offsetMsgId，此命令还有其他参数，具体作用请阅读QueryMsgByIdSubCommand。	-i	msgId
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
queryMsgByKey	根据消息 Key 查询消息	-k	msgKey
		-t	Topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
queryMsgByOffset	根据 Offset 查询消息	-b	Broker 名称，（这里需要注意 填写的是 Broker 的名称，不是 Broker 的地址，Broker 名称可以在 clusterList 查到）
		-i	query 队列 id
		-o	offset 值
		-t	topic 名称
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
queryMsgByUniqueKey	根据msgId查询，msgId不同于offsetMsgId，区别详见常见运维问题。-g，-d配合使用，查到消息后尝试让特定的消费者消费消息并返回消费结果	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-i	uniqe msg id
		-g	consumerGroup

		-d	clientId
		-t	topic名称
checkMsgSendRT	检测向topic发消息的RT，功能类似clusterRT	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-t	topic名称
		-a	探测次数
		-s	消息大小
sendMessage	发送一条消息，可以根据配置发往特定Message Queue，或普通发送。	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-t	topic名称
		-p	body，消息体
		-k	keys
		-c	tags
		-b	BrokerName
		-i	queueId
consumeMessage	消费消息。可以根据offset、开始&结束时间戳、消息队列消费消息，配置不同执行不同消费逻辑，详见ConsumeMessageCommand。	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-t	topic名称
		-b	BrokerName
		-o	从offset开始消费
		-i	queueId
		-g	消费者分组
		-s	开始时间戳，格式详见-h
		-d	结束时间戳
		-c	消费多少条消息
printMsg	从Broker消费消息并打印，可选时间段	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port

		-t	topic名称
		-c	字符集，例如UTF-8
		-s	subExpress，过滤表达式
		-b	开始时间戳，格式参见-h
		-e	结束时间戳
		-d	是否打印消息体
printMsgByQueue	类似printMsg，但指定Message Queue	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-t	topic名称
		-i	queueId
		-a	BrokerName
		-c	字符集，例如UTF-8
		-s	subExpress，过滤表达式
		-b	开始时间戳，格式参见-h
		-e	结束时间戳
		-p	是否打印消息
		-d	是否打印消息体
		-f	是否统计tag数量并打印
resetOffsetByTime	按时间戳重置offset，Broker和consumer都会重置	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-g	消费者分组
		-t	topic名称
		-s	重置为此时间戳对应的offset

		-f	是否强制重置，如果false，只支持回溯offset，如果true，不管时间戳对应offset与consumeOffset关系
		-c	是否重置c++客户端offset

消费者和消费者组相关

名称	含义	命令选项	说明
consumerProgress	查看订阅组消费状态，可以查看具体的client IP的消息积累量	-g	消费者所属组名
		-s	是否打印client IP
		-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
consumerStatus	查看消费者状态，包括同一个分组中是否都是相同的订阅，分析Process Queue是否堆积，返回消费者jstack结果，内容较多，使用者参见ConsumerStatusSubCommand	-h	打印帮助
		-n	NameServer 服务地址，格式 ip:port
		-g	consumer group
		-i	clientId
		-s	是否执行jstack
getConsumerStatus	获取 Consumer 消费进度	-g	消费者所属组名
		-t	查询主题
		-i	Consumer 客户端 ip
		-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
updateSubGroup	更新或创建订阅关系	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
		-b	Broker地址
		-c	集群名称
		-g	消费者分组名称
		-s	分组是否允许消费
		-m	是否从最小offset开始消费
		-d	是否是广播模式

		-q	重试队列数量
		-r	最大重试次数
		-i	当slaveReadEnable开启时有效，且还未达到从slave消费时建议从哪个BrokerId消费，可以配置备机id，主动从备机消费
		-w	如果Broker建议从slave消费，配置决定从哪个slave消费，配置BrokerId，例如1
		-a	当消费者数量变化时是否通知其他消费者负载均衡
deleteSubGroup	从Broker删除订阅关系	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
		-b	Broker地址
		-c	集群名称
		-g	消费者分组名称
cloneGroupOffset	在目标群组中使用源群组的offset	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
		-s	源消费者组
		-d	目标消费者组
		-t	topic名称
		-o	暂未使用

连接相关

名称	含义	命令选项	说明
consumerConnec tion	查询 Consumer 的网络连接	-g	消费者所属组名
		-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
producerConnec tion	查询 Producer 的网络连接	-g	生产者所属组名
		-t	主题名称
		-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助

NameServer相关

名称	含义	命令选项	说明
updateKvConfig	更新NameServer的kv配置，目前还未使用	-s	命名空间
		-k	key
		-v	value
		-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
deleteKvConfig	删除NameServer的kv配置	-s	命名空间
		-k	key
		-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
getNamesrvConfig	获取NameServer配置	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
updateNamesrvConfig	修改NameServer配置	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助
		-k	key
		-v	value

其他

名称	含义	命令选项	说明
startMonitoring	开启监控进程，监控消息误删、重试队列消息数等	-n	NameServer 服务地址，格式 ip:port
		-h	打印帮助

> 注意：
>
> 1、几乎所有指令都需要通过-n参数配置nameServer地址，格式为ip:port
>
> 2、几乎所有执行都可以通过-h参数获得帮助
>
> 3、当既有Broker地址(-b)又有集群名称clustername(-c)配合项，则优先以Broker地址执行指令。如果不配置Broker地址，则对集群中所有主机执行指令。

5、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：

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NameServer地址的读取方式见源码中
org.apache.rocketmq.common.utils.NameServerAddressUtils

```
    public static String getNameServerAddresses() {
        return System.getProperty("rocketmq.namesrv.addr",
System.getenv("NAMESRV_ADDR"));
    }
```

这个方法就是在DefaultMQProducer中默认的设置NameServer地址的方式，这个rokcetmq.namesrv.addr属性可以在java中使用System.setproperties指定，也可以在SpringBoot中配到配置文件里。

> 这个tools.sh就封装了一个简单的运行RocketMQ的环境，可以运行源码中的其他示例，然后自己的例子也可以放到RocketMQ的lib目录下去执行。

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-externals

下载下来后，进入其中的rocket-console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.properties中需要指定nameserver的地址。默认这个属性是空的。

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

```
rocketmq.config.namesrvAddr=worker1:9876;worker2:9876;worker3:9876
```

然后执行：

```
java -jar rocketmq-console-ng-1.0.1.jar
```

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

> 在管理页面的右上角可以选择语言。

Dleger高可用集群搭建

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
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。
>
> 单机状态下一般一次主从切换需要大概10S的时间。

9、调整系统参数

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：

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

> 有道云分享链接：
>
> 文档：RocketMQ集群搭建详解.md
> 链接：http://note.youdao.com/noteshare?id=aabfdd11fa8a2713b84d479c9e7f7d98&sub=FEA013B8A199478FBB956CD8B0899881