图灵：楼兰
你的神秘技术宝藏

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。

	优点	缺点	使用场景
kafka	吞吐量非常大， 性能非常好， 集群高可用。	会丢数据， 功能比较单一。	日志分析， 大数据采集
Rabbit MQ	消息可靠性高， 功能全面。	吞吐量比较低， 消息积累会影响性能， erlang语言不好定制。	小规模场景
Rocket MQ	高吞吐，高性能，高可用， 功能全面。	开源版功能不如云上版， 官方文档比较简单， 客户端只支持java。	几乎全场景

另外，关于这三大产品更详细的比较，可以参见《kafka vs rabbitmq vs rocketmq.pdf》

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 ，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。

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-4.7.1-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：

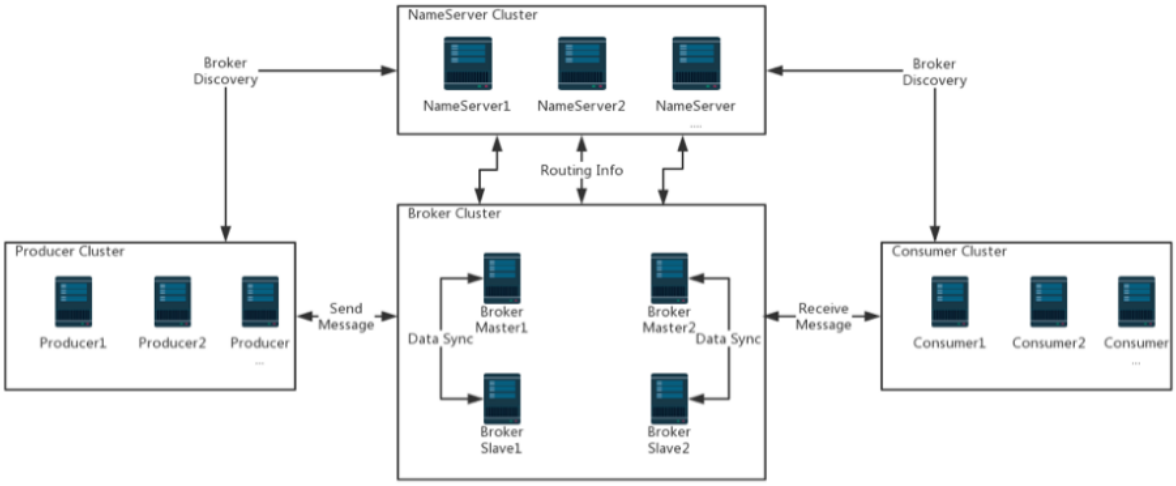
```
export JAVA_HOME=/app/jdk1.8/
export ROCKETMQ_HOME=/app/rocketmq/rocketmq-all-4.7.1-bin-release
PATH=$ROCKETMQ_HOME/bin:$JAVA_HOME/bin:$PATH:$HOME/.local/bin:$HOME/bin
export PATH
```

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
```

3.2 启动Broker

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 -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

```
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？

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=C0A8E88000002A9F0000000000317BF, 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。

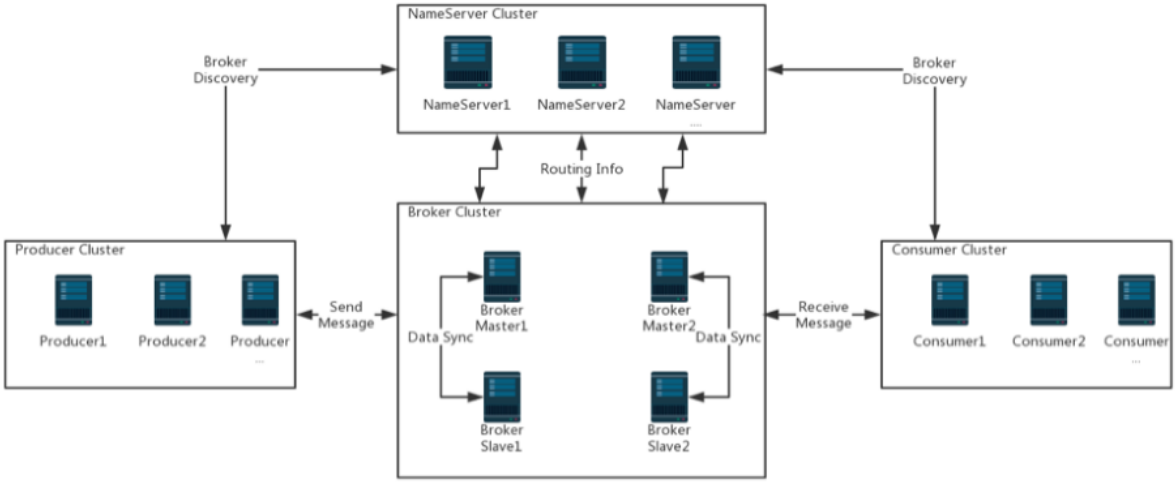
3.4 关闭RocketMQ服务

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，你都找到了吗？

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集群搭建详解.MD》

3、RocketMQ的其他参考资料

还记得我们之前把RocketMQ的源代码也下载下来了吗？我们现在不需要去看源代码，但是在源码中有个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，我们直接看看把。

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。当然，没有基础的同学也不用着急，但是在以后的学习中要有这个意识。

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，所以对RocketMQ要抱着学习，改进的态度，从点到面横向拓宽技术视野。

最后，我们要对RocketMQ的整体架构有一个全面的了解。并且在后续的细节学习时，要保持对第一个问题的好奇心。

> 有道云分享链接：
>
> 文档：VIP01-RocketMQ整体理解与快速实战.md
> 链接：http://note.youdao.com/noteshare?id=9a677e8cbccb2808802d608abe2b8a8d&sub=B0EBBB4932F34B6290D76C4AC502BA78