

《RocketMQ（ONS）的编译、部署与基本测试》

版本	内容	修改者	时间
Ver 0.1	创建，单机部署	马青宇（QQ：446192924）	2015.12.20
Ver 0.2	增加集群部署方式	马青宇	2016.01.14

作为淘宝几大著名神器之一的 RocketMQ 是一个分布式消息队列，现在阿里也把这个叫做 ONS，RocketMQ 的作用简单的说就是“削峰填谷”。

官方对 RocketMQ 的定义：一个队列模型的消息中间件，具有高性能、高可靠、高实时、分布式特点。CPU 晶体材料研发由于受到物理定律的限制，短期内不可能有突飞猛进的发展，在传说中的量子计算机还远没有达到实用之前，集群是不得已的处理复杂问题的方法，RocketMQ 就在于集群部署后可以显现出强大的威力。

淘宝每天处理上亿级别的订单，RocketMQ 功不可没，阿里把这样的一个小神器都拿出来共享，令人感慨良多，但愿能有更多的企业向阿里学习。

考虑到 RocketMQ 的应用文档比较少，而且有些内容写的比较晦涩难懂，导致 RocketMQ 的使用非常困难，遇到问题无从着手，严重打击了初学者学习 RocketMQ 的积极性，进而也影响了 RocketMQ 的广泛应用，特意编写一份简单的文档，供各位对 RocketMQ 有兴趣的爱好者参考。我感觉到我在本文档中已经把有关操作说明得足够细致了，初学者按照本文档一步一步操作，完全可以搭建起一个自己的集群，运行 RocketMQ 提供的各种示例代码，体会分布式消息处理的各种应用场景。

说句实在话，要想把 RocketMQ 运行起来是一件非常困难的事，在这个过程中你会遇到各种各样的问题，碰到各种各样的坑，而且很难找到问题解决的明确方法，郁郁数日而不得解；但是现在不同了，你可以在我编写的这个“红宝书”的英明指引下，快步前进，迈向幸福快乐的康庄大道。建议初学者（高手飘过）首先仔细阅读这份资料，在基本了解 RocketMQ 的使用方法等之后，再阅读其它技术文档。

阿里把 RocketMQ 这样的神器都能贡献出来，那我就贡献一份文档吧，希望她能发挥作用。

几个基本概念的简单解释：

- NameServer** : 名称服务器，其作用大致相当于 Hadoop 的 zookeeper。
- Broker** : 消息中转角色，负责存储消息，转发消息；相当于媒婆，把消息从 **Producer** 传递到 **Consumer**。
- Producer** : 消息生产者，负责产生消息，一般由业务系统负责产生消息
- Consumer** : 消息消费者，负责消费消息，一般是后台系统负责异步消费消息

本文的结构比较简单，不像上次的《手把手教你搭建云计算平台》写了那么多的内容，但是这篇短文费的功夫一点也不比安装 Hadoop 少，因为现在研究 Hadoop 的人远远多过研究 RocketMQ 的，Hadoop 遇到问题可以比较容易的找到解决的办法，RocketMQ 就不一样了；短文包括如下六个部分：

- 1、前提条件
- 2、硬件需求

- 3、简要安装过程说明
- 4、详细安装操作过程
- 5、RocketMQ 消息示例测试
- 6、本次集群部署架构说明

一、前提条件:

心理准备: 要有搭建不是一次就能完成的思想, 我都不记得重复了多少次了, 总是遇到各种各样的问题。

软件准备:

VMware Workstation 11.1.0	虚拟机软件, 这个就不多说了。
CentOS-6.5-x86_64-bin-DVD1.iso	CentOS 操作系统, 64 位; 我习惯使用这个版本。
RocketMQ-3.2.6.tar.gz	目前能够得到的 RocketMQ 的最新源代码。

二、硬件需求:

这是一个演示的“双 master 无 slave” RocketMQ 集群配置, 需要同时启动 4 台虚拟机; 我的 ThinkPad T410 笔记本只有 6G 的内存, 原来给每个虚拟机分配 1G 内存的时候, 虚拟机运行的比较顺畅, 只是笔记本主机运行的不顺畅; 后来把虚拟机分配 512M 内存, 有些命令在运行的时候会提示 3 秒超时, 再次运行命令就好了, 看来 512M 内存是一个下限了, 如果硬件条件好, 建议分配 1G 以上的内存给每个虚拟机。四台虚拟机总计约需要 20 个 G 的硬盘空间。

三、简要安装过程说明:

- 1、在第一台虚拟机中安装好 Linux 系统, 完成基本的环境配置。
- 2、安装 JDK 和 Maven, 及配置环境变量。
- 3、关闭防火墙和关闭 SELinux。
- 4、编译 RocketMQ 源代码。
- 5、部署 RocketMQ 执行代码到指定路径和配置 broker 属性文件。
- 6、依据第一台虚拟机克隆出另外三台虚拟机。
- 7、分别运行 NameServer 和 Broker。
- 8、运行简单的 RocketMQ 测试程序。

虚拟机名称及 IP 地址规划:

在创建第一台虚拟机时, 命名为“VM_RM_Q_M1”, 虚拟机文件也存放在 VM_RM_Q_M1 目录中; 作为 RocketMQ 集群中的 master1 主机, 分配的 IP 地址是 192.168.2.10。四台虚拟机配置如下:

虚拟机 1:	VM_RM_Q_M1	192.168.2.10	运行 namesrv 和 broker
虚拟机 2:	VM_RM_Q_M2	192.168.2.11	运行 namesrv 和 broker
虚拟机 3:	VM_RM_Q_PR	192.168.2.12	运行 producer
虚拟机 4:	VM_RM_Q_CS	192.168.2.13	运行 consumer

四、详细搭建操作过程:

1、在虚拟机中安装 Linux 系统

在 VMWare 中创建 Linux 虚拟机的部分, 这里不详细说明; 需要注意的几点, 首先, 创建 Linux 虚拟机的时候, 网络选项我们这里选择的是 “NAT”, 为的是通过 yum 的方式安装 JDK 和 Maven; 其次, 配置第一台虚拟机网络地址为 192.168.2.10, 255.255.255.0, 192.168.2.1, 网卡设置为开机自动连接; 其次, 安装选项选择 “Minimal Desktop”, 最小桌面, 并选择上定制安装选项; 删除 Base System 中的 Java Platform 选项, 我们在后面安装 Maven 的时候, 它会安装上比较新的 JDK 版本。后面的操作都是在 root 用户下完成的; 为方便记忆, 这里设置 root 用户登录 Linux 的密码为: 123456

安装 JDK 和 Maven 之后, 还要把网络选项由 “NAT” 修改为 “Host_only” 模式, 使这四台机器处于一个孤立的网络环境, 并且互相之间是可以连通的, 因为我们这里就是做一个简单的实验, 暂时不与外部连通也是没有什么影响的, 还能减少因为外面环境的干扰导致的各种问题。本来直接安装 1.6 以上版本的 JDK 就是可以的, 但是不知道什么原因, 试了几个版本的都不行, namesrv 都启动不起来, 总是说什么 “JAVA_HOME” 环境变量没有配置, 只有这么安装 JDK 才管用, 等待高手出手解决这个问题吧!!!

Linux 操作系统本身就是一个多用户操作系统, 与 Windows 下的操作习惯有差别, 有时出现的一些问题也往往是出在这里。

2、安装 JDK 和 Maven;

编译和运行 RocketMQ 需要 Java 环境和 Maven 软件; 需要 root 用户, 两个命令分别如下:

```
wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
```

```
yum -y install apache-maven
```

运行完成上面两个命令后, 设置环境变量, 命令行如下:

```
gedit /etc/profile
```

在 profile 文件末尾追加下面的内容:

```
# set java environment
```

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.91.x86_64
```

```
export JRE_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.91.x86_64/jre
```

```
export
```

```
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib:$CLASSPATH
```

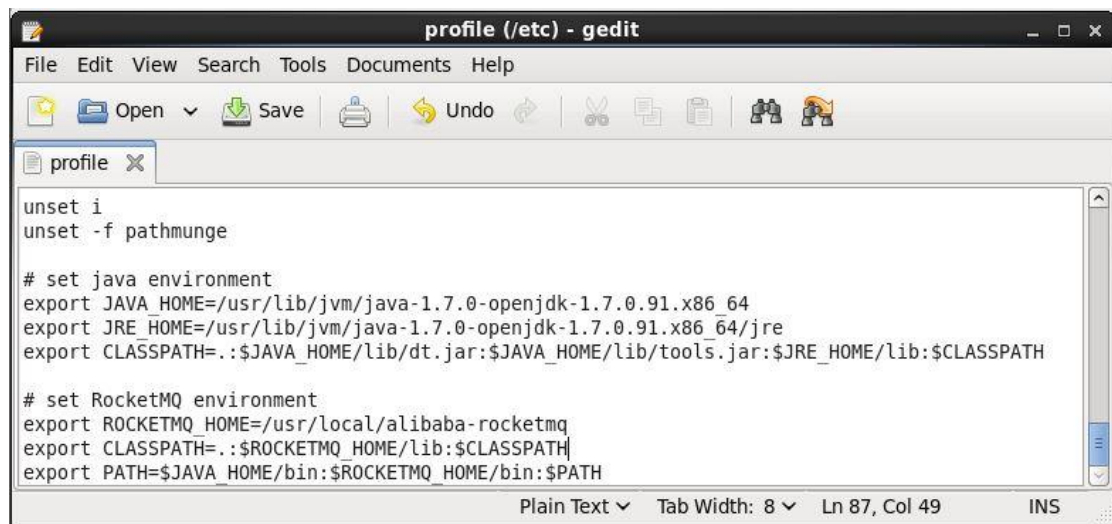
```
# set RocketMQ environment
```

```
export ROCKETMQ_HOME=/usr/local/alibaba-rocketmq
```

```
export CLASSPATH=.:$ROCKETMQ_HOME/lib:$CLASSPATH
```

```
export PATH=$JAVA_HOME/bin:$ROCKETMQ_HOME/bin:$PATH
```

截图如下:



这里把 RocketMQ 的环境变量设置也放在一起完成。

分别验证一下 JDK 和 Maven 是否都已经安装成功，在命令行分别运行如下两个命令：

`java -version`

`mvn -v`

截图如下：

```
[root@localhost ~]# java -version
java version "1.7.0_91"
OpenJDK Runtime Environment (rhel-2.6.2.2.el6_7-x86_64 u91-b00)
OpenJDK 64-Bit Server VM (build 24.91-b01, mixed mode)
[root@localhost ~]# mvn -v
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T19:57:37+08:00)
Maven home: /usr/share/apache-maven
Java version: 1.7.0_91, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.91.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-431.el6.x86_64", arch: "amd64", family: "unix"
[root@localhost ~]#
```

至此，完成 JDK 和 Maven 的安装与配置。

3、关闭防火墙和关闭 SELinux

如下图所示，需要在 root 用户下运行如下 4 个命令：

<code>service iptables stop</code>	<code>#停止 iptables 服务</code>
<code>chkconfig iptables off</code>	<code>#配置系统启动时关闭 iptables</code>
<code>service ip6tables stop</code>	<code>#停止 ip6tables 服务</code>
<code>chkconfig ip6tables off</code>	<code>#配置系统启动时关闭 ip6tables</code>

```
[root@Master ~]# service iptables stop
iptables: Setting chains to policy ACCEPT: filter      [ OK ]
iptables: Flushing firewall rules:                    [ OK ]
iptables: Unloading modules:                           [ OK ]
[root@Master ~]# chkconfig iptables off
[root@Master ~]# service ip6tables stop
ip6tables: Setting chains to policy ACCEPT: filter    [ OK ]
ip6tables: Flushing firewall rules:                   [ OK ]
ip6tables: Unloading modules:                           [ OK ]
[root@Master ~]# chkconfig ip6tables off
```

用 root 用户运行下面的命令

`gedit /etc/sysconfig/selinux` #编辑 selinux 文件

打开 selinux 文件后，修改

`SELINUX=enforcing`

为

`SELINUX=disabled`

保存后退出 `gedit`。接着再执行如下命令，注意 `setenforce` 后面有空格：

`setenforce 0` #设置 SELinux 状态

`getenforce` #获取 SELinux 状态

截图如下：

```
[root@Master ~]# gedit /etc/sysconfig/selinux
[root@Master ~]# setenforce 0
[root@Master ~]# getenforce
Permissive
```

至此就完成了关闭防火墙和关闭 SELinux 的操作。

注意：关闭防火墙和 SELinux 的配置这个操作非常重要，如果缺少这个步骤，在后面运行 `producer` 发送消息的时候，在出现的一大堆各种运行提示中，会有一个不起眼的内容，说 “No route info of this topic”，然后消息也没有发送出去，短短的一行提示，也能让你晕好几天，找不出所以然来。因为咱们这里就是一个测试环境，就想验证一下 `RocketMQ` 怎么用，也是在自己的机器上，干脆就把这些安全机制都关掉了，省得各种各样的麻烦。

4、编译 RocketMQ 源代码。

`RocketMQ` 源代码我是在 Windows 环境下编译的；在配置好 JDK 和 Maven 的 Windows 环境下，解压缩此源代码到指定路径，然后在此路径下可以看到 `install.bat` 文件，在命令行下运行此批处理文件，经过一段时间的编译之后，在当前路径的 `target` 子目录下，会生成 “`alibaba-rocketmq-3.2.6-alibaba-rocketmq.tar.gz`” 文件，此压缩文件中包含所有 `RocketMQ` 的 jar 包、集群配置、可执行脚本文件等；我看 “`RocketMQ` 用户交流” 的群共享中已经有这个文件了，不想自己编译的话，直接下载这个使用也是可以的。

5、部署 RocketMQ 执行代码到指定路径和配置 broker 属性文件。

复制前面生成的 “`alibaba-rocketmq-3.2.6-alibaba-rocketmq.tar.gz`” 文件 Linux 的 home 路径下，执行下面的解压缩命令：

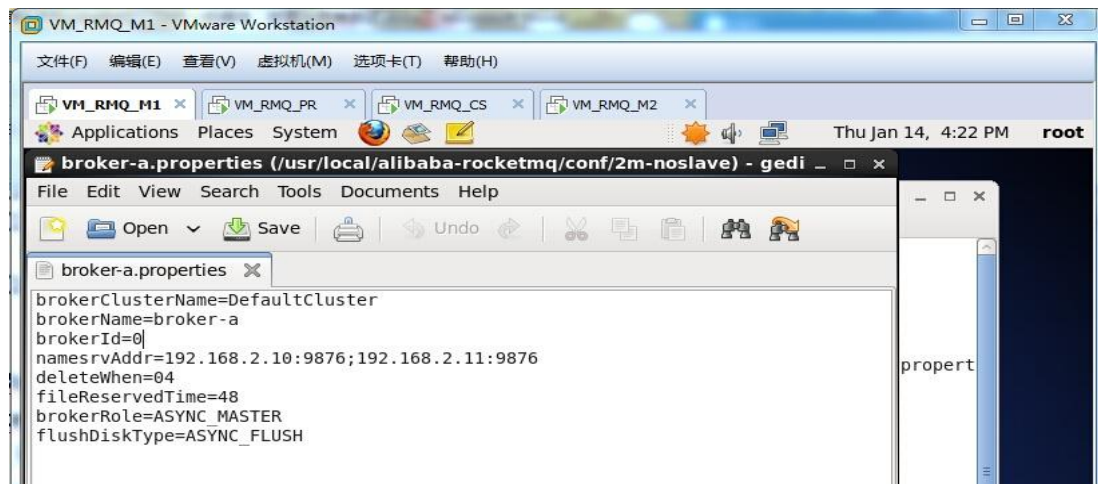
`tar -zxvf -3.2.6-alibaba-rocketmq.tar.gz`

会产生 alibaba-rocketmq 目录，剪切此目录到/usr/local 路径下；
赋予 RocketMQ 的 bin 路径下的文件以可执行权限，运行如下两个命令：
cd /usr/local/alibaba-rocketmq/bin
chmod +x *

截图如下：

```
[root@localhost bin]# ll
total 80
-rwxr-xr-x. 1 root root 677 Mar 28 2015 mqadmin
-rwxr-xr-x. 1 root root 554 Mar 28 2015 mqadmin.xml
-rwxr-xr-x. 1 root root 673 Mar 28 2015 mqbroker
-rwxr-xr-x. 1 root root 651 Mar 28 2015 mqbroker.numanode0
-rwxr-xr-x. 1 root root 651 Mar 28 2015 mqbroker.numanode1
-rwxr-xr-x. 1 root root 651 Mar 28 2015 mqbroker.numanode2
-rwxr-xr-x. 1 root root 651 Mar 28 2015 mqbroker.numanode3
-rwxr-xr-x. 1 root root 546 Mar 28 2015 mqbroker.xml
-rwxr-xr-x. 1 root root 678 Mar 28 2015 mqfiltersrv
-rwxr-xr-x. 1 root root 552 Mar 28 2015 mqfiltersrv.xml
-rwxr-xr-x. 1 root root 674 Mar 28 2015 mqnamesrv
-rwxr-xr-x. 1 root root 548 Mar 28 2015 mqnamesrv.xml
-rwxr-xr-x. 1 root root 791 Mar 28 2015 mqshutdown
-rwxr-xr-x. 1 root root 1228 Mar 28 2015 os.sh
-rwxr-xr-x. 1 root root 226 Mar 28 2015 play.sh
-rwxr-xr-x. 1 root root 643 Mar 28 2015 README.md
-rwxr-xr-x. 1 root root 1831 Mar 28 2015 runbroker.sh
-rwxr-xr-x. 1 root root 1555 Mar 28 2015 runserver.sh
-rwxr-xr-x. 1 root root 686 Mar 28 2015 startfsv.sh
-rwxr-xr-x. 1 root root 1095 Mar 28 2015 tools.sh
[root@localhost bin]#
```

至此，完成 RocketMQ 的部署；RocketMQ 的环境变量已经在前面配置过了。
然后修改 /usr/local/alibaba-rocketmq/conf/2m-noslave/ 路径下的 broker-a.properties 和
broker-a.properties 文件的内容，只在第 3 行（就是 brokerId=0 这一行）的后面插入一行：
namesrvAdd=192.168.2.10:9876; 192.168.2.11:9876
只插入这样一行即可，不要像其它文档中说的要增加好多行别的内容，咱们这里就是简单的
测试，先能正常运行起来再说，以后别的功能可以慢慢添加，开始的时候，不要弄得那么复
杂，出现问题，反而理不出头绪。截图如下：



两个文件都修改完成后，保存退出，并关闭虚拟机，为后面的克隆虚拟机做准备。

6、依据第一台虚拟机克隆出另外三台虚拟机。

在 vmware 中克隆操作，各位同学先自己补习一下吧，克隆出的机器会增加一个新的网卡，MAC 是与被克隆的虚拟机不同的，复制这个新的 MAC 值，替换 eth0 的 MAC 值，并修改 IP 为前面指定的地址。保存后，重新启动机器，运行 ifconfig 确认本从机的网络配置是正确的。还要使用 ping 命令测试机器之间网络是否连通。

在前面的这些操作没有处理好并且确认无误之前，没有必要进行后面的步骤，这些是基础环境配置，处理不好或者处理得不完整，后面的其它操作肯定是会遇到各种各样的问题，白白浪费时间和精力。

5、启动 master1 和 master2 虚拟机中的 NameServer 和 Broker。

因为前面已经设置此虚拟机的 IP 地址为：192.168.2.10，所以我们直接在 /usr/local/alibaba-rocketmq/bin 路径下启动 nameserver，命令行如下：

```
nohup sh mqnamesrv > ns.log &
```

截图如下：

```
[root@localhost bin]# nohup sh mqnamesrv > ns.log &
[1] 2724
[root@localhost bin]# nohup: ignoring input and redirecting stderr to stdout
[root@localhost bin]#
```

然后查看 ns.log 文件的内容应该如下所示：

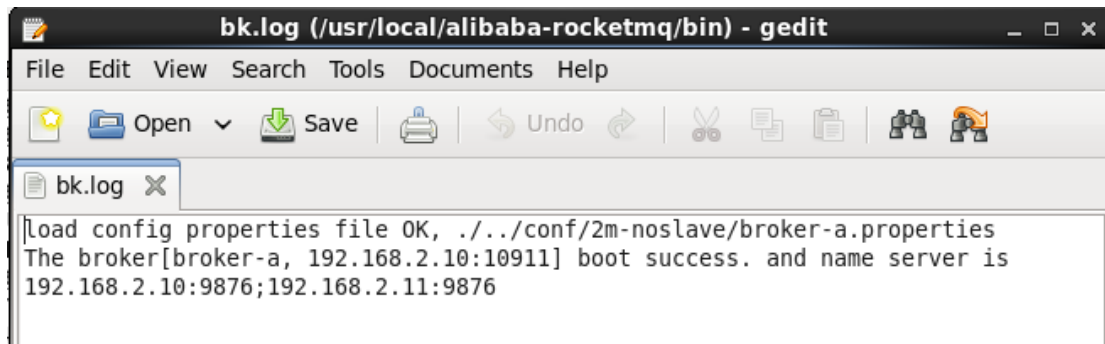
The Name Server boot success.

如果不是这个内容，请检查 JDK 的安装、环境变量的配置、rocketmq 的部署等是否都正常，在这个问题没有解决之前，没有必要进行下面的其它操作。（可能要先配置好网络环境，就是在更新 Maven 后，配置好本地网的机器 IP，然后再运行 nameserver）

然后在 /usr/local/alibaba-rocketmq/bin 路径下启动 broker，命令行如下：

```
nohup sh mqbroker -c ../../conf/2m-noslave/broker-a.properties > bk.log &
```

然后查看 bk.log 文件的内容应该如下所示：



如果不是这个内容，也请检查 JDK 的安装、环境变量的配置、rocketmq 的部署、nameserver 的启动等是否都正常，在这个问题没有解决之前，没有必要进行下面的其它操作。

再次确认 nameserver 和 broker 的运行情况，运行 jps 命令，截图如下：



可以看到 NamesrvStartup 和 BrokerStartup 进程都正常启动了。
至此，在 master1 机器上的 NameServer 和 Broker 启动已经成功完成。

在 master2 执行相同的操作，确保两个 master 都正常运行了 broker。
在 master2 执行的命令行如下，第一个命令与 master1 相同，第二个命令使用 broker-b.properties 这个属性文件。如下：

```
nohup sh mqnamesrv > ns.log &  
nohup sh mqbroker -c ../../conf/2m-noslave/broker-b.properties > bk.log &
```

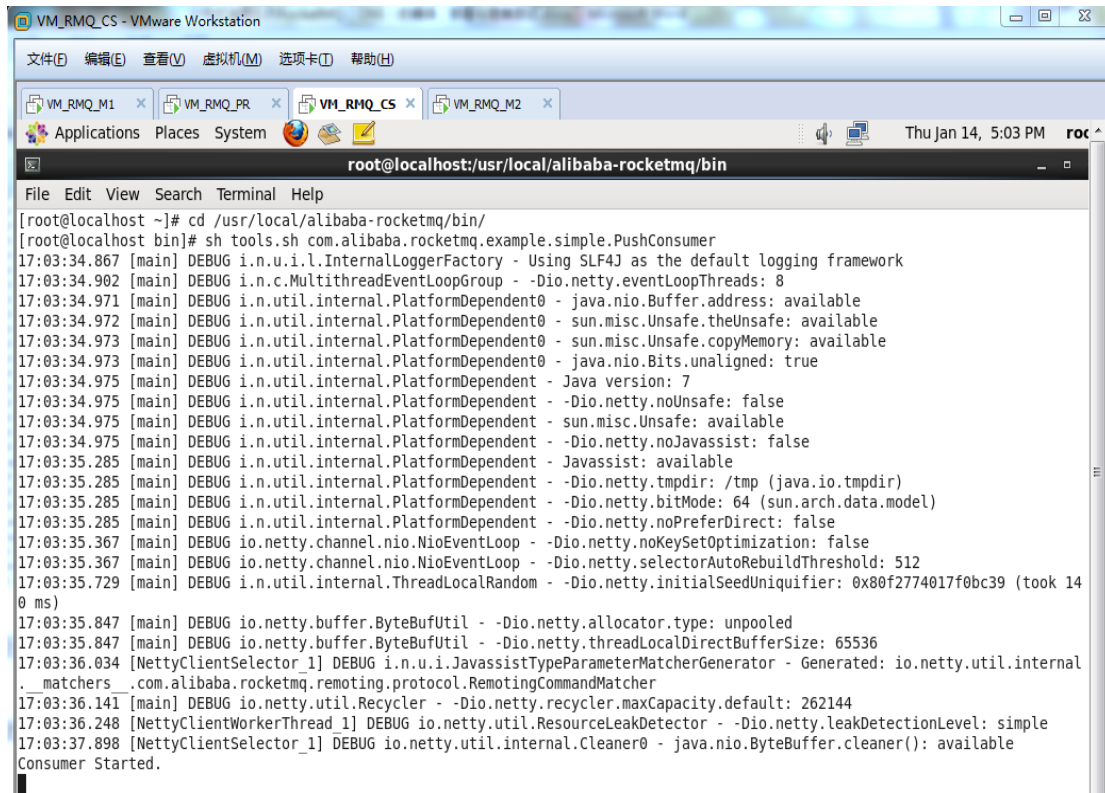
至此，在 master1 和 master2 机器上的 NameServer 和 Broker 启动都已经成功完成。

7、运行简单的 RocketMQ 测试程序。

既然四台虚拟机都已经配置完成了，两台 master 上的 nameserver 和 broker 也都启动起来了，那就真正的让 producer 发布一些消息，看看 consumer 是否可以接收到消息吧；

首先在“**VM_RM_Q_CS**”这台虚拟机上运行 consumer，仍然是在 /usr/local/alibaba-rocketmq/bin 路径下，命令行如下：

```
sh tools.sh com.alibaba.rocketmq.example.simple.PushConsumer
```

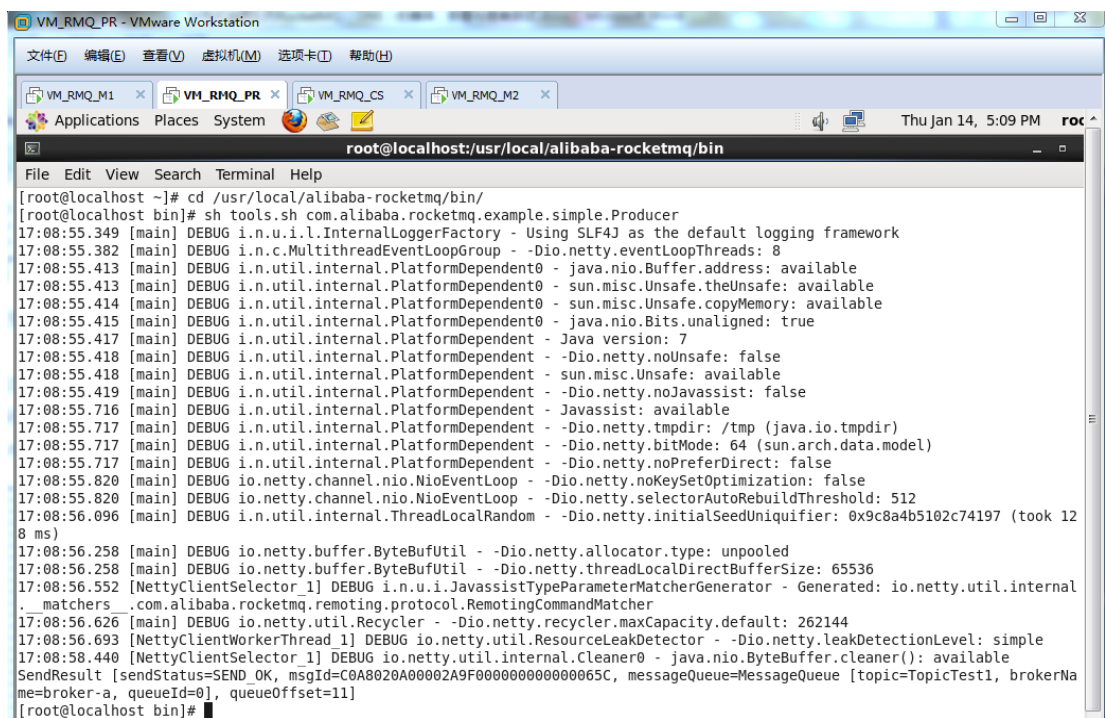
```
VM_RM_Q_CS - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
VM_RM_Q_M1 x VM_RM_Q_PR x VM_RM_Q_CS x VM_RM_Q_M2 x
Applications Places System
root@localhost:/usr/local/alibaba-rocketmq/bin
File Edit View Search Terminal Help
[root@localhost ~]# cd /usr/local/alibaba-rocketmq/bin/
[root@localhost bin]# sh tools.sh com.alibaba.rocketmq.example.simple.PushConsumer
17:03:34.867 [main] DEBUG i.n.u.i.l.InternalLoggerFactory - Using SLF4J as the default logging framework
17:03:34.902 [main] DEBUG i.n.c.MultithreadEventLoopGroup - -Dio.netty.eventLoopThreads: 8
17:03:34.971 [main] DEBUG i.n.util.internal.PlatformDependent0 - java.nio.Buffer.address: available
17:03:34.972 [main] DEBUG i.n.util.internal.PlatformDependent0 - sun.misc.Unsafe.theUnsafe: available
17:03:34.973 [main] DEBUG i.n.util.internal.PlatformDependent0 - sun.misc.Unsafe.copyMemory: available
17:03:34.973 [main] DEBUG i.n.util.internal.PlatformDependent0 - java.nio.Bits.unaligned: true
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - Java version: 7
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noUnsafe: false
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - sun.misc.Unsafe: available
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noJavassist: false
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - Javassist: available
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.tmpdir: /tmp (java.io.tmpdir)
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.bitMode: 64 (sun.arch.data.model)
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noPreferDirect: false
17:03:35.367 [main] DEBUG io.netty.channel.nio.NioEventLoop - -Dio.netty.noKeySetOptimization: false
17:03:35.367 [main] DEBUG io.netty.channel.nio.NioEventLoop - -Dio.netty.selectorAutoRebuildThreshold: 512
17:03:35.729 [main] DEBUG i.n.util.internal.ThreadLocalRandom - -Dio.netty.initialSeedUniquifier: 0x80f2774017f0bc39 (took 14
0 ms)
17:03:35.847 [main] DEBUG io.netty.buffer.ByteBufUtil - -Dio.netty allocator.type: unpooled
17:03:35.847 [main] DEBUG io.netty.buffer.ByteBufUtil - -Dio.netty.threadLocalDirectBufferSize: 65536
17:03:36.034 [NettyClientSelector_1] DEBUG i.n.u.i.JavassistTypeParameterMatcherGenerator - Generated: io.netty.util.internal
. matchers . com.alibaba.rocketmq.remoting.protocol.RemotingCommandMatcher
17:03:36.141 [main] DEBUG io.netty.util.Recycler - -Dio.netty.recycler.maxCapacity.default: 262144
17:03:36.248 [NettyClientWorkerThread_1] DEBUG io.netty.util.ResourceLeakDetector - -Dio.netty.leakDetectionLevel: simple
17:03:37.898 [NettyClientSelector_1] DEBUG io.netty.util.internal.Cleaner0 - java.nio.ByteBuffer.cleaner(): available
Consumer Started.
```

看到最后面一行显示的“Consumer Started.”，表示 Consumer 已经运行起来了。

然后在“VM_RM_Q_PR”这台虚拟机上运行 producer，还是在/usr/local/alibaba-rocketmq/bin路径下，命令行如下：

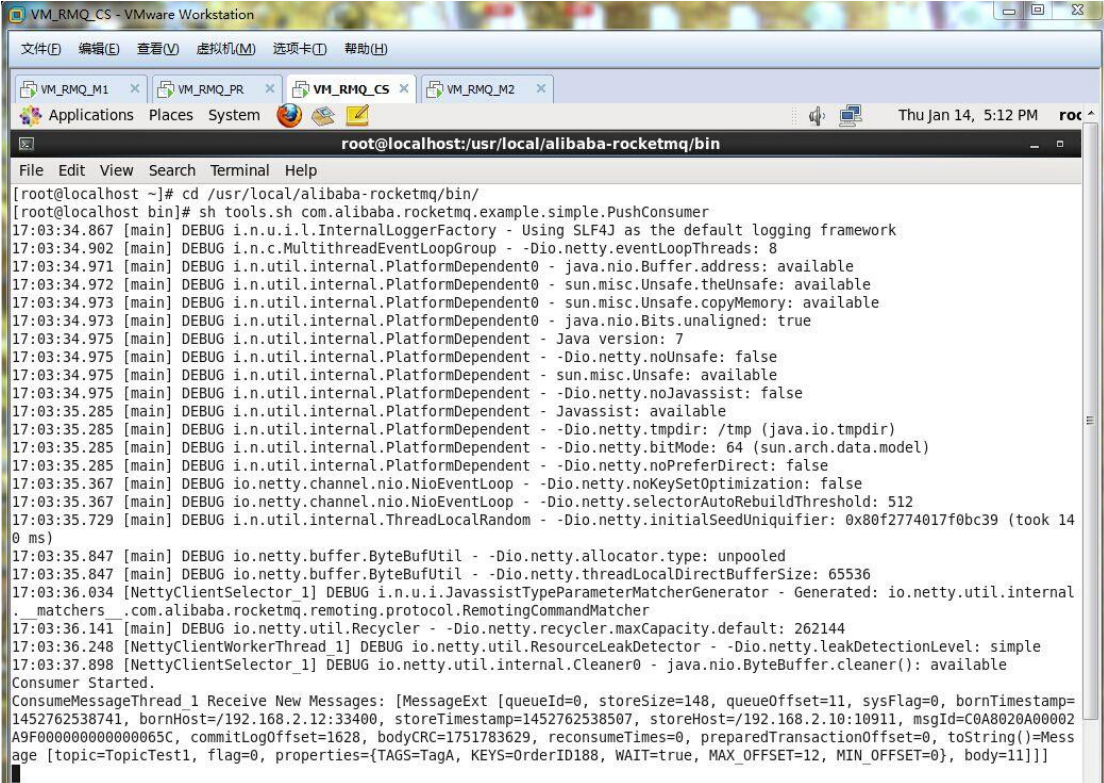
```
sh tools.sh com.alibaba.rocketmq.example.simple.Producer
```

在最后面两行，可以看到 producer 已经发送了一条消息。



```
VM_RM_Q_PR - VMware Workstation
文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H)
VM_RM_Q_M1 x VM_RM_Q_PR x VM_RM_Q_CS x VM_RM_Q_M2 x
Applications Places System
root@localhost:/usr/local/alibaba-rocketmq/bin
File Edit View Search Terminal Help
[root@localhost ~]# cd /usr/local/alibaba-rocketmq/bin/
[root@localhost bin]# sh tools.sh com.alibaba.rocketmq.example.simple.Producer
17:08:55.349 [main] DEBUG i.n.u.i.l.InternalLoggerFactory - Using SLF4J as the default logging framework
17:08:55.382 [main] DEBUG i.n.c.MultithreadEventLoopGroup - -Dio.netty.eventLoopThreads: 8
17:08:55.413 [main] DEBUG i.n.util.internal.PlatformDependent0 - java.nio.Buffer.address: available
17:08:55.413 [main] DEBUG i.n.util.internal.PlatformDependent0 - sun.misc.Unsafe.theUnsafe: available
17:08:55.414 [main] DEBUG i.n.util.internal.PlatformDependent0 - sun.misc.Unsafe.copyMemory: available
17:08:55.415 [main] DEBUG i.n.util.internal.PlatformDependent0 - java.nio.Bits.unaligned: true
17:08:55.417 [main] DEBUG i.n.util.internal.PlatformDependent - Java version: 7
17:08:55.418 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noUnsafe: false
17:08:55.418 [main] DEBUG i.n.util.internal.PlatformDependent - sun.misc.Unsafe: available
17:08:55.419 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noJavassist: false
17:08:55.716 [main] DEBUG i.n.util.internal.PlatformDependent - Javassist: available
17:08:55.717 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.tmpdir: /tmp (java.io.tmpdir)
17:08:55.717 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.bitMode: 64 (sun.arch.data.model)
17:08:55.717 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noPreferDirect: false
17:08:55.820 [main] DEBUG io.netty.channel.nio.NioEventLoop - -Dio.netty.noKeySetOptimization: false
17:08:55.820 [main] DEBUG io.netty.channel.nio.NioEventLoop - -Dio.netty.selectorAutoRebuildThreshold: 512
17:08:56.096 [main] DEBUG i.n.util.internal.ThreadLocalRandom - -Dio.netty.initialSeedUniquifier: 0x9c8a4b5102c74197 (took 12
8 ms)
17:08:56.258 [main] DEBUG io.netty.buffer.ByteBufUtil - -Dio.netty allocator.type: unpooled
17:08:56.258 [main] DEBUG io.netty.buffer.ByteBufUtil - -Dio.netty.threadLocalDirectBufferSize: 65536
17:08:56.552 [NettyClientSelector_1] DEBUG i.n.u.i.JavassistTypeParameterMatcherGenerator - Generated: io.netty.util.internal
. matchers . com.alibaba.rocketmq.remoting.protocol.RemotingCommandMatcher
17:08:56.626 [main] DEBUG io.netty.util.Recycler - -Dio.netty.recycler.maxCapacity.default: 262144
17:08:56.693 [NettyClientWorkerThread_1] DEBUG io.netty.util.ResourceLeakDetector - -Dio.netty.leakDetectionLevel: simple
17:08:58.440 [NettyClientSelector_1] DEBUG io.netty.util.internal.Cleaner0 - java.nio.ByteBuffer.cleaner(): available
SendResult [sendStatus=SEND_OK, msgId=C0A8020A00002A9F000000000000065C, messageQueue=MessageQueue [topic=TopicTest1, brokerNa
me=broker-a, queueId=0], queueOffset=11]
[root@localhost bin]#
```

再回头看看“**VM_RM_Q_CS**”虚拟机中的 consumer 是否收到了消息呢？



```
[root@localhost ~]# cd /usr/local/alibaba-rocketmq/bin/
[root@localhost bin]# sh tools.sh com.alibaba.rocketmq.example.simple.PushConsumer
17:03:34.867 [main] DEBUG i.n.u.i.l.InternalLoggerFactory - Using SLF4J as the default logging framework
17:03:34.902 [main] DEBUG i.n.c.MultithreadEventLoopGroup - -Dio.netty.eventLoopThreads: 8
17:03:34.971 [main] DEBUG i.n.util.internal.PlatformDependent0 - java.nio.Buffer.address: available
17:03:34.972 [main] DEBUG i.n.util.internal.PlatformDependent0 - sun.misc.Unsafe.theUnsafe: available
17:03:34.973 [main] DEBUG i.n.util.internal.PlatformDependent0 - sun.misc.Unsafe.copyMemory: available
17:03:34.973 [main] DEBUG i.n.util.internal.PlatformDependent0 - java.nio.Bits.unaligned: true
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - Java version: 7
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noUnsafe: false
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - sun.misc.Unsafe: available
17:03:34.975 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noJavassist: false
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - Javassist: available
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.tmpdir: /tmp (java.io.tmpdir)
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.bitMode: 64 (sun.arch.data.model)
17:03:35.285 [main] DEBUG i.n.util.internal.PlatformDependent - -Dio.netty.noPreferDirect: false
17:03:35.367 [main] DEBUG io.netty.channel.nio.NioEventLoop - -Dio.netty.noKeySetOptimization: false
17:03:35.367 [main] DEBUG io.netty.channel.nio.NioEventLoop - -Dio.netty.selectorAutoRebuildThreshold: 512
17:03:35.729 [main] DEBUG i.n.util.internal.ThreadLocalRandom - -Dio.netty.initialSeedUniquifier: 0x80f2774017f0bc39 (took 14
0 ms)
17:03:35.847 [main] DEBUG io.netty.buffer.ByteBufUtil - -Dio.netty allocator.type: unpooled
17:03:35.847 [main] DEBUG io.netty.buffer.ByteBufUtil - -Dio.netty.threadLocalDirectBufferSize: 65536
17:03:36.034 [NettyClientSelector_1] DEBUG i.n.u.i.JavassistTypeParameterMatcherGenerator - Generated: io.netty.util.internal
. matchers
.com.alibaba.rocketmq.remoting.protocol.RemotingCommandMatcher
17:03:36.141 [main] DEBUG io.netty.util.Recycler - -Dio.netty.recycler.maxCapacity.default: 262144
17:03:36.248 [NettyClientWorkerThread_1] DEBUG io.netty.util.ResourceLeakDetector - -Dio.netty.leakDetectionLevel: simple
17:03:37.898 [NettyClientSelector_1] DEBUG io.netty.util.internal.Cleaner0 - java.nio.ByteBuffer.cleaner(): available
Consumer Started.
ConsumeMessageThread_1 Receive New Messages: [MessageExt [queueId=0, storeSize=148, queueOffset=11, sysFlag=0, bornTimestamp=
1452762538741, bornHost=/192.168.2.12:33400, storeTimestamp=1452762538507, storeHost=/192.168.2.10:10911, msgId=C0A8020A000002
A9F00000000000065C, commitLogOffset=1628, bodyCRC=1751783629, reconsumeTimes=0, preparedTransactionOffset=0, toString()=Mess
age [topic=TopicTest1, flag=0, properties={TAGS=TagA, KEYS=OrderID188, WAIT=true, MAX_OFFSET=12, MIN_OFFSET=0}, body=11]]]
```

从后面三行的显示中可以看出，consumer 已经接收到了订阅的消息。发送消息的 IP 地址是 192.168.2.12，就是“**VM_RM_Q_PR**”虚拟机的地址。

到此，RocketMQ 的“双 master”部署测试已经完成，但这只是一个简单、基本的部署和测试，只是比上次的在单台虚拟机上“单 master”测试稍微复杂一下；这次是通过四台虚拟机共同完成这个测试。

Broker 分为 Master 与 Slave，一个 Master 可以对应多个 Slave，但是一个 Slave 只能对应一个 Master。

Broker 集群部署方式主要有以下几种：

（1）单个 Master

上次（Ver 0.1）演示的就是这种方式，producer、consumer、broker、namesrv 都运行在一台机器上，本机发送消息，也在本机接收消息；仅作为演示，生产环境不要这样使用。

（2）多 Master 模式

一个集群无 Slave，全是 Master，例如 2 个 Master 或者 3 个 Master。

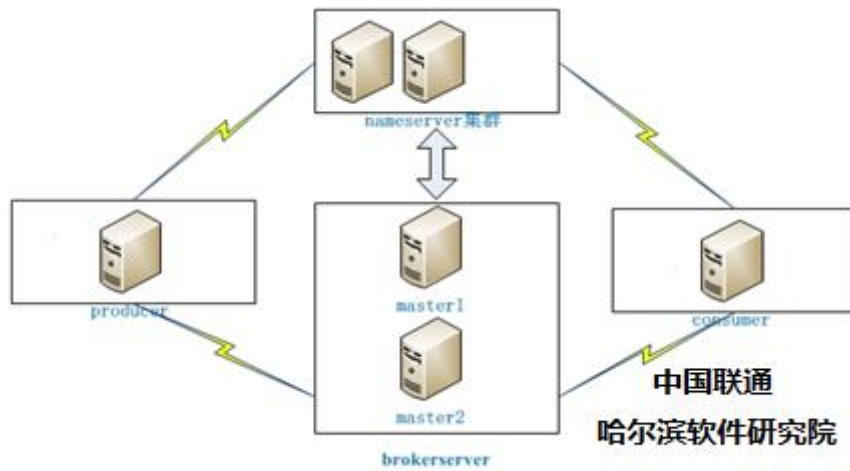
（3）多 Master 多 Slave 模式，异步复制

每个 Master 配置一个 Slave，有多对 Master-Slave，HA 采用异步复制方式，主备有短暂消息延迟，毫秒级。

（4）多 Master 多 Slave 模式，同步双写

每个 Master 配置一个 Slave，有多对 Master-Slave，HA 采用同步双写方式，主备都写成功，向应用返回成功。

我们上面部署的这个集群属于第二种类型：多 Master 模式；
两个 master，无 slave，架构如下图：



其中 nameserver 没有使用专门的机器，而是分别运行在两个 master 上，producer 是一台机器，这里只运行了 “com.alibaba.rocketmq.example.simple.Producer” 程序，consumer 是另一台机器这里只运行了 “com.alibaba.rocketmq.example.simple.PushConsumer” 程序。