RocketMQ 学习笔记(1)_Linux 下编译部署 RocketMQ

作者：coffeelifelau　　编辑日期：20161026

博客：http://blog.csdn.net/coffeelifelau

微服务架构中，消息队列和远程服务调用已是两大必不可少的组件，而 RocketMQ 和 Dubbo 正是阿里系贡献的对应的两大精品开源，作为两个已经得到广泛应用的框架，好好学习研究是必需的。

1. 软件准备

官方文档：https://github.com/alibaba/RocketMQ/wiki/quick-start

根据文档说明，需要以下软件来完成这个快速开始示例：

1. 64bit OS, best to have Linux/Unix/Mac;
2. 64bit JDK 1.6+;
3. Maven 3.x
4. Git
5. Screen

1.1 关于 Linux 和 Windows

作为纯 Java 程序，RocketMQ 在 Windows 下也是可以运行的，官方还准备了 exe 执行文件方便 Windows 环境下进行开发部署。

Windows 下的编译部署大同小异，有兴趣可以参考下面这个网址：

http://blog.csdn.net/ruishenh/article/details/22390809

1.2 关于 JDK

如果 Linux 已经自带 JDK，可以使用命令查看 JDK 版本，如果版本不符合 64 位 1.6+，需要先卸载旧版本然后安装新版本。

我安装的是 jdk-8u111-linux-x64.rpm，过程略过，有问题可以参考下面的网址。

下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.htm

安装教程：http://www.cnblogs.com/benio/archive/2010/09/14/1825909.html

按理 32 位 JDK 也是可以运行的，只是需要调整内存配置，但可能不适合生产环境。未经测试，不妄言。

1.3 安装 Maven

1.3.1 下载

```
cd /usr/javawork
wget
http://mirrors.hust.edu.cn/apache/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
```

1.3.2 解压

```
tar -zxvf apache-maven-3.3.9-bin.tar.gz
```

1.3.3 设置环境变量

```
vi /etc/profile
```

文件末尾添加两行配置：

```
export M2_HOME=/usr/javawork/apache-maven-3.3.9
export PATH=$PATH:$M2_HOME/bin
```

退出 vi 执行命令使其生效：

```
source /etc/profile
```

1.3.4 添加 alibaba 的 Maven 仓库镜像（下载速度飞快）

```
vi /usr/javawork/apache-maven-3.3.9/conf/settings.xml
```

在<mirrors>项下添加镜像信息：

```
<mirror>
    <id>alimaven</id>
    <name>aliyun maven</name>
    <mirrorOf>central</mirrorOf>
    <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
</mirror>
```

1.4 安装 Git

```
yum install git
```

如果已下载 RocketMQ 源码包，Git 可以无需安装。shell 安装脚本中有 git pull 命令，如果未安装 git，会提示 command not found，但不影响后面的编译。

如果嫌烦，vi install.sh 打开文件删掉 git pull 这条命令即可。

1.5 安装 screen

```
yum install screen
```

screen 非必需，但安装后多任务切换会话非常方便。官方文档中使用了这条命令，所以还是装上较好。

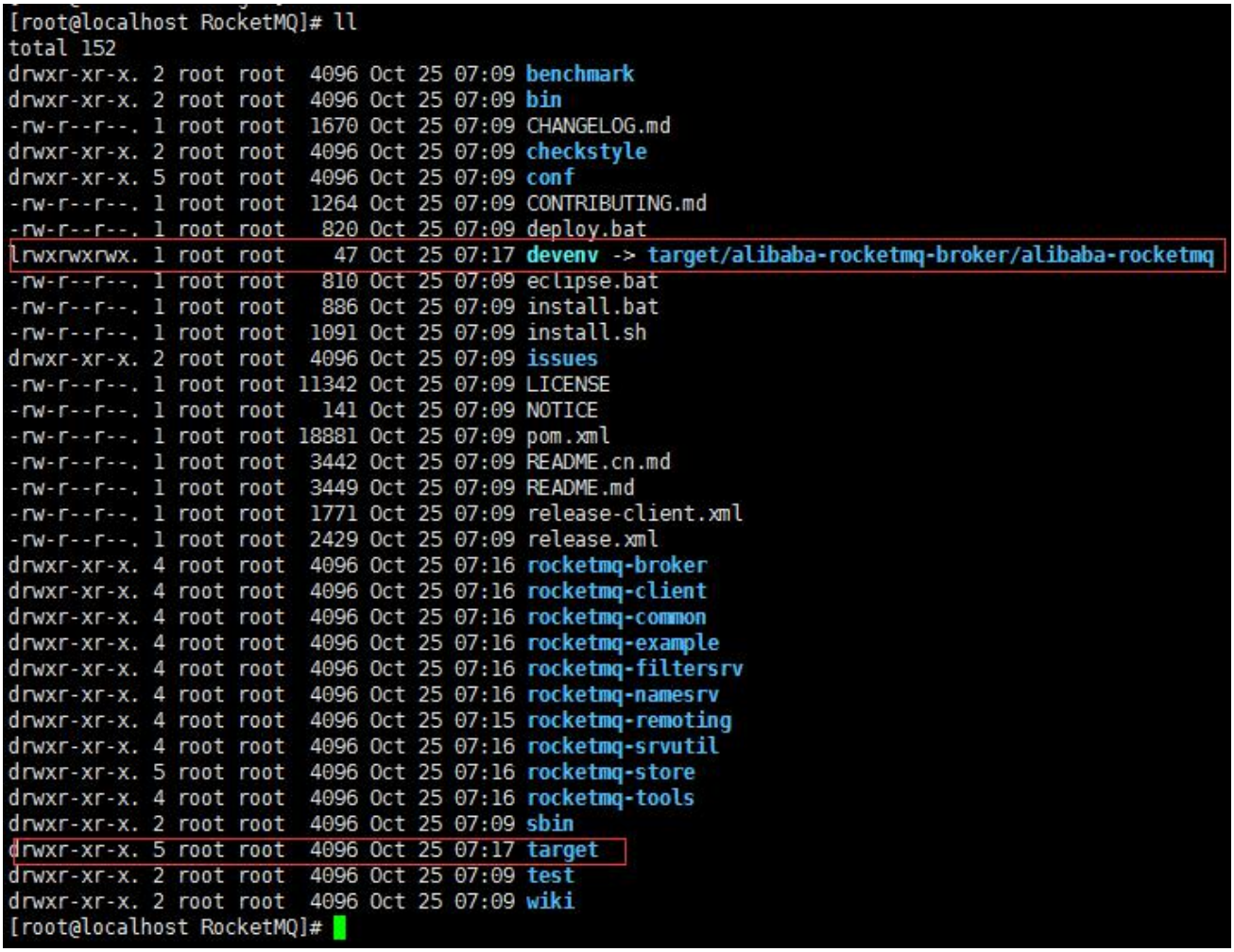
命令介绍：http://www.cnblogs.com/mchina/archive/2013/01/30/2880680.html

2. 安装 RocketMQ

2.1 下载编译

```
git clone https://github.com/alibaba/RocketMQ.git
cd RocketMQ
bash install.sh
```

如果编译成功，最终得到的目录如下图。devenv 是软链接，源文件在 target 目录。



2.2 环境变量

设置环境变量

```
cd devenv
echo "ROCKETMQ_HOME=`pwd`" >> ~/.bash_profile
```

使环境变量生效：

```
source ~/.bash_profile
```

3. 启动 RocketMQ

```
cd bin
```

3.1 启动 Name Server

```
screen bash mqnamesrv
```

如果未安装 screen，可以使用下面这条命令：

```
nohup sh mqnamesrv &（加 & 可以后台运行，否则 Ctrl+c 命令退出当前会话，服务会停止）
```

启动成功的信息：

Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=320m; support was removed in 8.0
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

备注：出现三条警告信息是因为 JDK1.8 已经不支持设置方法区大小和指定 CMS 垃圾收集算法进行 FullGC，后面会讲如何去掉这些信息。

3.2 启动 Broker

先按 ctrl+a，然后再按 d 挂起当前会话，然后再执行以下命令：

```
screen bash mqbroker -n localhost:9876
```

启动成功后除了警告信息外，会出现以下信息：

```
The broker[localhost.localdomain, 192.168.0.11:10911] boot success. serializeType=JSON and name server is localhost:9876
```

3.3 去除警告信息和调整内存占用

3.3.1 修改 xml 文件

```
cd bin
vi mqadmin.xml
vi mqbroker.xml
vi mqnamesrv.xml
vi mqfiltersrv.xml
```

依次打开这些 xml 文件，并删除下图红色框中的标记的配置信息。

```
<options>
    <-Xms512m></-Xms512m>
    <-Xmx1g></-Xmx1g>
    <-XX:NewSize>256M</-XX:NewSize>
    <-XX:MaxNewSize>512M</-XX:MaxNewSize>
    <-XX:PermSize>128M</-XX:PermSize>
    <-XX:MaxPermSize>128M</-XX:MaxPermSize>
</options>
```

3.3.2 修改启动文件

```
vi runserver.sh
vi runbroker.sh
```

依次打开这两个文件，去除下图红色标记的配置信息。

```
#=====================================================================================
# JVM Configuration
#=====================================================================================
JAVA_OPT="${JAVA_OPT} -server -Xms4g -Xmx4g -Xmn2g -XX:PermSize=128m -XX:MaxPermSize=320m"
JAVA_OPT="${JAVA_OPT} -XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -XX:CMSInitiatingOccu
pancyFraction=70 -XX:+CMSParallelRemarkEnabled -XX:SoftRefLRUPolicyMSPerMB=0 -XX:+CMSClassUnloadingEna
bled -XX:SurvivorRatio=8 -XX:+DisableExplicitGC"
JAVA_OPT="${JAVA_OPT} -verbose:gc -Xloggc:${HOME}/rmq_srv_gc.log -XX:+PrintGCDetails"
JAVA_OPT="${JAVA_OPT} -XX:-OmitStackTraceInFastThrow"
JAVA_OPT="${JAVA_OPT} -Djava.ext.dirs=${BASE_DIR}/lib"
#JAVA_OPT="${JAVA_OPT} -Xdebug -Xrunjdwp:transport=dt_socket,address=9555,server=y,suspend=n"
JAVA_OPT="${JAVA_OPT} -cp ${CLASSPATH}"
```

如图，还看到了黄色部分的内存设置，因为当前是虚拟机搭建的开发环境，没有必要使用 4g 内存，所以我将内存调整成如下图：

```
JAVA_OPT="${JAVA_OPT} -server -Xms1g -Xmx1g -Xmn500m"
```

3.4 停止服务

关闭 Name Server

```
sh mqshutdown namesrv
```

关闭 Broker

```
sh mqshutdown broker
```

3.5 日志目录

```
cd ~/logs/rocketmqlogs/
```

3.6 收发消息测试

3.6.1 设置地址

```
export NAMESRV_ADDR=localhost:9876
```

3.6.2 测试命令

生产者：bash tools.sh com.alibaba.rocketmq.example.quickstart.Producer

消费者：bash tools.sh com.alibaba.rocketmq.example.quickstart.Consumer

4. 代码示例

4.1 代码

```
/** 生产者 */
public class Producer {

    public static void main(String[] args) throws MQClientException,
InterruptedException {
        /**
         * 一个应用创建一个 Producer，由应用来维护此对象，可以设置为全局对象或者单例<br>
         * 注意：ProducerGroupName 需要由应用来保证唯一<br>
         * ProducerGroup 这个概念发送普通的消息时，作用不大，但是发送分布式事务消息时，比较关键，
```

```
         * 因为服务器会回查这个 Group 下的任意一个 Producer
         */
        final DefaultMQProducer producer = new DefaultMQProducer("ProducerGroupName");
        producer.setNamesrvAddr("192.168.0.11:9876");
        producer.setInstanceName("Producer");

        /**
         * Producer 对象在使用之前必须要调用 start 初始化，初始化一次即可<br>
         * 注意：切记不可以在每次发送消息时，都调用 start 方法
         */
        producer.start();

        /**
         * 下面这段代码表明一个 Producer 对象可以发送多个 topic，多个 tag 的消息。
         * 注意：send 方法是同步调用，只要不抛异常就标识成功。但是发送成功也可会有多种状态，<br>
         * 例如消息写入 Master 成功，但是 Slave 不成功，这种情况消息属于成功，但是对于个别应用如果对消息可靠性要求极高，<br>
         * 需要对这种情况做处理。另外，消息可能会存在发送失败的情况，失败重试由应用来处理。
         */
        for (int i = 0; i < 10; i++) {
            try {
                {
                    Message msg = new Message("TopicTest1", // topic
                            "TagA", // tag
                            "OrderID001", // key
                            ("Hello MetaQA").getBytes());// body
                    SendResult sendResult = producer.send(msg);
                    System.out.println(sendResult);
                }

                {
                    Message msg = new Message("TopicTest2", // topic
                            "TagB", // tag
                            "OrderID0034", // key
                            ("Hello MetaQB").getBytes());// body
                    SendResult sendResult = producer.send(msg);
                    System.out.println(sendResult);
                }

                {
                    Message msg = new Message("TopicTest3", // topic
```

```
                    "TagC", // tag
                    "OrderID061", // key
                    ("Hello MetaQC").getBytes());// body
                SendResult sendResult = producer.send(msg);
                System.out.println(sendResult);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        TimeUnit.MILLISECONDS.sleep(1000);
    }

    /**
     * 应用退出时，要调用 shutdown 来清理资源，关闭网络连接，从 MetaQ 服务器上注销自己
     * 注意：我们建议应用在 JBOSS、Tomcat 等容器的退出钩子里调用 shutdown 方法
     */
    // producer.shutdown();
    Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
        public void run() {
            producer.shutdown();
        }
    }));
    System.exit(0);
  }
}

/** 消费者 */
public class PushConsumer {
    /**
     * 当前例子是 PushConsumer 用法，使用方式给用户感觉是消息从 RocketMQ 服务器推到了应用客户端。<br>
     * 但是实际 PushConsumer 内部是使用长轮询 Pull 方式从 MetaQ 服务器拉消息，然后再回调用户 Listener 方法<br>
     */
   public static void main(String[] args) throws InterruptedException, MQClientException {
        /**
         * 一个应用创建一个 Consumer，由应用来维护此对象，可以设置为全局对象或者单例<br>
         * 注意：ConsumerGroupName 需要由应用来保证唯一
         */
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("ConsumerGroupName");
```

```
        consumer.setNamesrvAddr("192.168.0.11:9876");
        consumer.setInstanceName("Consumber");

        /**
         * 订阅指定 topic 下 tags 分别等于 TagA 或 TagC 或 TagD
         */
        consumer.subscribe("TopicTest1", "TagA || TagC || TagD");
        /**
         * 订阅指定 topic 下所有消息<br>
         * 注意：一个 consumer 对象可以订阅多个 topic
         */
        consumer.subscribe("TopicTest2", "*");

        consumer.registerMessageListener(

                new MessageListenerConcurrently() {

                    public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt>
msgs,
                            ConsumeConcurrentlyContext context) {

                        System.out.println(Thread.currentThread().getName()    +    "
Receive New Messages: " + msgs.size());

                        MessageExt msg = msgs.get(0);
                        if (msg.getTopic().equals("TopicTest1")) {
                            // 执行 TopicTest1 的消费逻辑
                            if (msg.getTags() != null && msg.getTags().equals("TagA"))
{
                                // 执行 TagA 的消费
                                System.out.println(new String(msg.getBody()));
                            }     else     if     (msg.getTags()     !=     null     &&
msg.getTags().equals("TagC")) {
                                // 执行 TagC 的消费
                                System.out.println(new String(msg.getBody()));
                            }     else     if     (msg.getTags()     !=     null     &&
msg.getTags().equals("TagD")) {
                                // 执行 TagD 的消费
                                System.out.println(new String(msg.getBody()));
                            }
                        } else if (msg.getTopic().equals("TopicTest2")) {
                            System.out.println(new String(msg.getBody()));
                        }
```

```
                    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;

                }
            });

        /**
         * Consumer 对象在使用之前必须要调用 start 初始化，初始化一次即可<br>
         */
        consumer.start();

        System.out.println("ConsumerStarted.");
    }
}
```

4.2 运行结果

分别运行以上两个程序，正常会输出如下信息：

生产者端：

```
SendResult [sendStatus=SEND_OK, msgId=C0A80008326873D16E9381E943250000,offsetMsgId=C0A8000B00002A9F00000000000318 62, messageQueue=MessageQueue [topic=TopicTest1, brokerName=localhost.localdomain, queueId=2], queueOffset=9]
SendResult [sendStatus=SEND_OK, msgId=C0A80008326873D16E9381E943540001,offsetMsgId=C0A8000B00002A9F00000000000319 21, messageQueue=MessageQueue [topic=TopicTest2, brokerName=localhost.localdomain, queueId=0], queueOffset=10]
SendResult [sendStatus=SEND_OK, msgId=C0A80008326873D16E9381E943630002,offsetMsgId=C0A8000B00002A9F00000000000319 E1, messageQueue=MessageQueue [topic=TopicTest3, brokerName=localhost.localdomain, queueId=2], queueOffset=11]
```

消费者端：

```
ConsumerStarted.
ConsumeMessageThread_1 Receive New Messages: 1
Hello MetaQA
ConsumeMessageThread_4 Receive New Messages: 1
Hello MetaQA
```

4.3 错误备忘

```
com.alibaba.rocketmq.client.exception.MQClientException: No route info of this topic, TopicTest1
See http://docs.aliyun.com/cn#/pub/ons/faq/exceptions&topic_not_exist for further details.
```

如果出现以上错误，是因为启动 Broker 时后面的主机和端口未指定

screen bash mqbroker -n localhost:9876

5. 其它信息：
示例下载：

参考资料：

https://github.com/alibaba/RocketMQ/wiki/quick-start

http://blog.csdn.net/ruishenh/article/details/22390809