

0 关于Java1234_锋哥

作者: java1234_小锋

官网站点: <http://www.java1234.vip>

关于锋哥: <http://www.java1234.vip/article/14>

java学习路线图: <http://www.java1234.vip/article/1>

QQ: 554605804



加锋哥微信 java1239
关注锋哥朋友圈, 天天干货



关注 java1234 公众号
送java新人福利

学习RocketMQ之前, 建议大家先学习下经典的RabbitMQ 课程地址: <http://www.java1234.vip/course/149>

所以关于消息队列的基本思想, 基本概念以及应用场景就不再赘述。

1RocketMQ简介

RocketMQ是由阿里捐赠给Apache的一款低延迟、高并发、高可用、高可靠的分布式消息中间件。经历了淘宝双十一的洗礼。RocketMQ既可为分布式应用系统提供异步解耦和削峰填谷的能力, 同时也具备互联网应用所需的海量消息堆积、高吞吐、可靠重试等特性。

官方文档: <https://rocketmq.apache.org/docs/quick-start/>

github中文主页: <https://github.com/apache/rocketmq/tree/master/docs/cn>

核心概念

- **Topic**: 消息主题, 一级消息类型, 生产者向其发送消息。

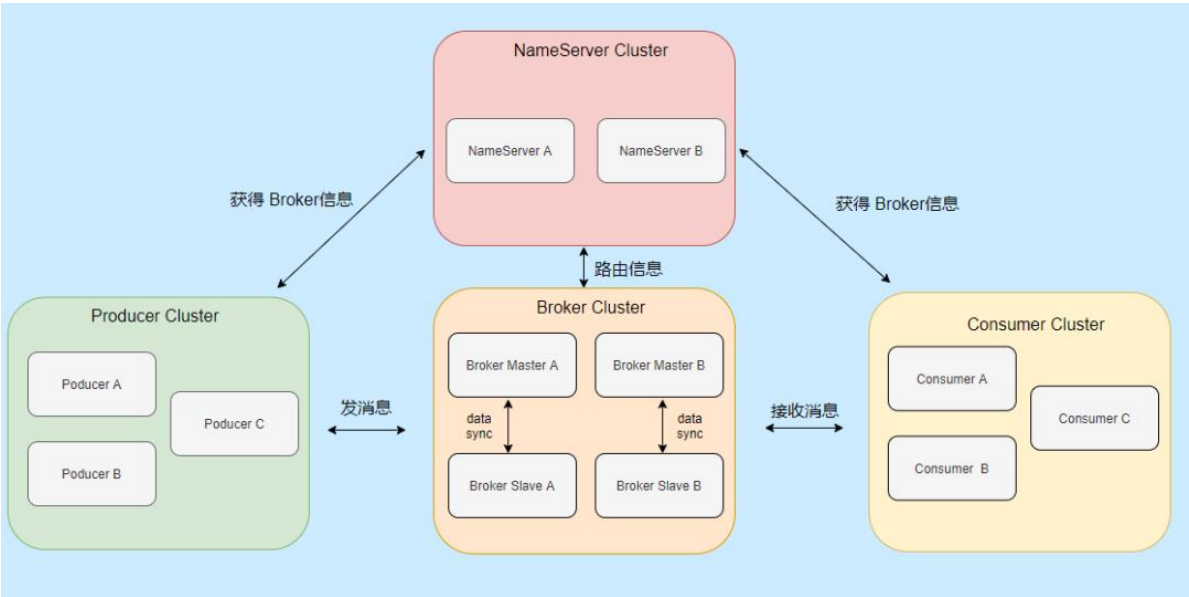
- **Message**: 生产者向Topic发送并最终传送给消费者的数据消息的载体。
- **消息属性**: 生产者可以为消息定义的属性, 包含Message Key和Tag。
- **Message Key**: 消息的业务标识, 由消息生产者 (Producer) 设置, 唯一标识某个业务逻辑。
- **Message ID**: 消息的全局唯一标识, 由消息队列RocketMQ系统自动生成, 唯一标识某条消息。
- **Tag**: 消息标签, 二级消息类型, 用来进一步区分某个Topic下的消息分类
- **Producer**: 也称为消息发布者, 负责生产并发送消息至Topic。
- **Consumer**: 也称为消息订阅者, 负责从Topic接收并消费消息。
- **分区**: 即Topic Partition, 物理上的概念。每个Topic包含一个或多个分区。
- **消费位点**: 每个Topic会有多个分区, 每个分区会统计当前消息的总条数, 这个称为最大位点MaxOffset; 分区的起始位置对应的位置叫做起始位点MinOffset。
- **Group**: 一类生产者或消费者, 这类生产者或消费者通常生产或消费同一类消息, 且消息发布或订阅的逻辑一致。
- **Group ID**: Group的标识。
- **队列**: 个Topic下会由一到多个队列来存储消息。
- **Exactly-Once投递语义**: Exactly-Once投递语义是指发送到消息系统的消息只能被Consumer处理且仅处理一次, 即使Producer重试消息发送导致某消息重复投递, 该消息在Consumer也只被消费一次。
- **集群消费**: 一个Group ID所标识的所有Consumer平均分摊消费消息。例如某个Topic有9条消息, 一个Group ID有3个Consumer实例, 那么在集群消费模式下每个实例平均分摊, 只消费其中的3条消息。
- **广播消费**: 一个Group ID所标识的所有Consumer都会各自消费某条消息一次。例如某个Topic有9条消息, 一个Group ID有3个Consumer实例, 那么在广播消费模式下每个实例都会各自消费9条消息。
- **定时消息**: Producer将消息发送到消息队列RocketMQ服务端, 但并不期望这条消息立马投递, 而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费, 该消息即定时消息。
- **延时消息**: Producer将消息发送到消息队列RocketMQ服务端, 但并不期望这条消息立马投递, 而是延迟一定时间后才投递到Consumer进行消费, 该消息即延时消息。
- **事务消息**: RocketMQ提供类似X/Open XA的分布事务功能, 通过消息队列RocketMQ的事务消息能达到分布式事务的最终一致。
- **顺序消息**: RocketMQ提供的一种按照顺序进行发布和消费的消息类型, 分为全局顺序消息和分区顺序消息。
- **全局顺序消息**: 对于指定的一个Topic, 所有消息按照严格的先入先出 (FIFO) 的顺序进行发布和消费。
- **分区顺序消息**: 对于指定的一个Topic, 所有消息根据Sharding Key进行区块分区。同一个分区内的消息按照严格的FIFO顺序进行发布和消费。Sharding Key是顺序消息中用来区分不同分区的关键字段, 和普通消息的Message Key是完全不同的概念。
- **消息堆积**: Producer已经将消息发送到消息队列RocketMQ的服务端, 但由于Consumer消费能力有限, 未能在短时间内将所有消息正确消费掉, 此时在消息队列RocketMQ的服务端保存着未被消费的消息, 该状态即消息堆积。
- **消息过滤**: Consumer可以根据消息标签 (Tag) 对消息进行过滤, 确保Consumer最终只接收被过滤后的消息类型。消息过滤在消息队列RocketMQ的服务端完成。
- **消息轨迹**: 在一条消息从Producer发出到Consumer消费处理过程中, 由各个相关节点的时间、地点等数据汇聚而成的完整链路信息。通过消息轨迹, 您能清晰定位消息从Producer发出, 经由消息队列RocketMQ服务端, 投递给Consumer的完整链路, 方便定位排查问题。
- **重置消费位点**: 以时间轴为坐标, 在消息持久化存储的时间范围内 (默认3天), 重新设置Consumer对已订阅的Topic的消费进度, 设置完成后Consumer将接收设定时间点之后由Producer发送到消息队列RocketMQ服务端的消息。
- **死信队列**: 死信队列用于处理无法被正常消费的消息。当一条消息初次消费失败, 消息队列RocketMQ会自动进行消息重试; 达到最大重试次数后, 若消费依然失败, 则表明Consumer在正常情况下无法正确地消费该消息。此时, 消息队列RocketMQ不会立刻将消息丢弃, 而是将这条消息发送到该Consumer对应的特殊队列中。

消息队列RocketMQ将这种正常情况下无法被消费的消息称为死信消息（Dead-Letter Message），将存储死信消息的特殊队列称为死信队列（Dead-Letter Queue）。

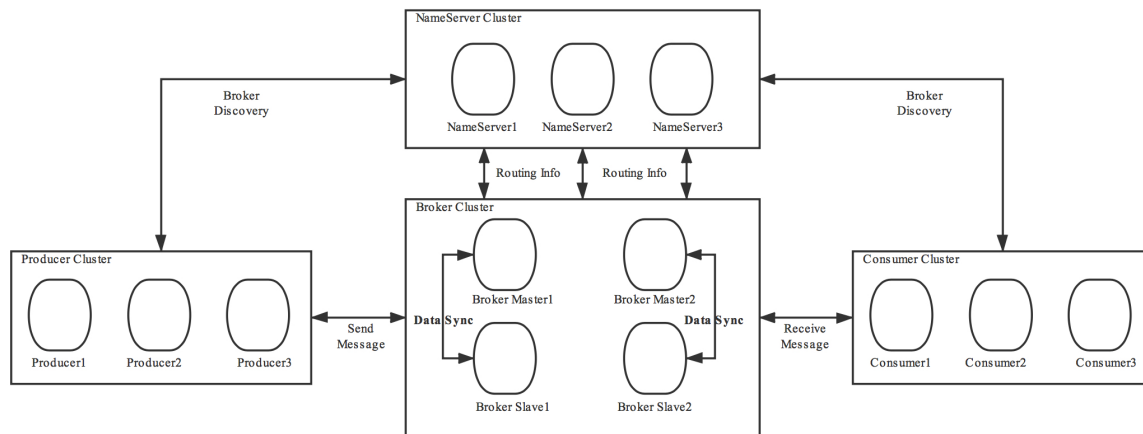
应用场景

- **削峰填谷**：诸如秒杀、抢红包、企业开门红等大型活动时皆会带来较高的流量脉冲，或因没做相应的保护而导致系统超负荷甚至崩溃，或因限制太过导致请求大量失败而影响用户体验，消息队列RocketMQ可提供削峰填谷的服务来解决该问题。
- **异步解耦**：交易系统作为淘宝和天猫主站最核心的系统，每笔交易订单数据的产生会引起几百个下游业务系统的关注，包括物流、购物车、积分、流计算分析等等，整体业务系统庞大而且复杂，消息队列RocketMQ可实现异步通信和应用解耦，确保主站业务的连续性。
- **顺序收发**：细数日常中需要保证顺序的应用场景非常多，例如证券交易过程时间优先原则，交易系统下的订单创建、支付、退款等流程，航班中的旅客登机消息处理等等。与先进先出FIFO（First In First Out）原理类似，消息队列RocketMQ提供的顺序消息即保证消息FIFO。
- **分布式事务一致性**：交易系统、支付红包等场景需要确保数据的最终一致性，大量引入消息队列RocketMQ的分布式事务，既可以实现系统之间的解耦，又可以保证最终的数据一致性。
- **大数据分析**：数据在“流动”中产生价值，传统数据分析大多是基于批量计算模型，而无法做到实时的数据分析，利用阿里云消息队列RocketMQ与流式计算引擎相结合，可以很方便的实现业务数据的实时分析。
- **分布式缓存同步**：天猫双11大促，各个分会场琳琅满目的商品需要实时感知价格变化，大量并发访问数据库导致会场页面响应时间长，集中式缓存因带宽瓶颈，限制了商品变更的访问流量，通过消息队列RocketMQ构建分布式缓存，实时通知商品数据的变化。

架构设计

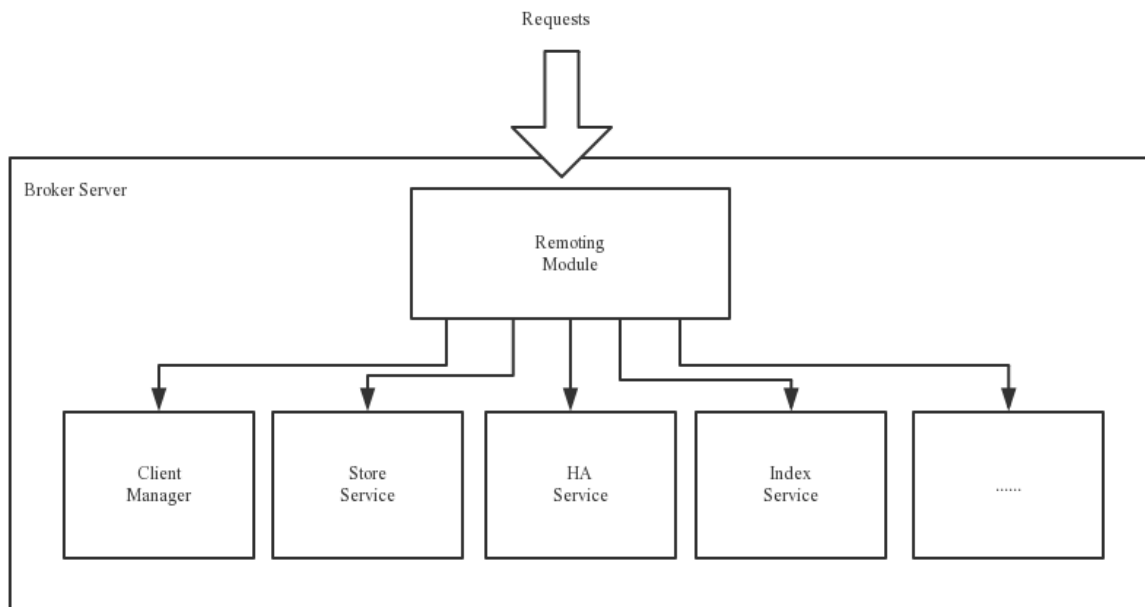


1 技术架构



RocketMQ架构上主要分为四部分，如上图所示：

- **Producer:** 消息发布的角色，支持分布式集群方式部署。Producer通过MQ的负载均衡模块选择相应的Broker集群队列进行消息投递，投递的过程支持快速失败并且低延迟。
- **Consumer:** 消息消费的角色，支持分布式集群方式部署。支持以push推，pull拉两种模式对消息进行消费。同时也支持集群方式和广播方式的消费，它提供实时消息订阅机制，可以满足大多数用户的需求。
- **NameServer:** NameServer是一个非常简单的Topic路由注册中心，其角色类似Dubbo中的zookeeper，支持Broker的动态注册与发现。主要包括两个功能：Broker管理，NameServer接受Broker集群的注册信息并且保存下来作为路由信息的基本数据。然后提供心跳检测机制，检查Broker是否还存活；路由信息管理，每个NameServer将保存关于Broker集群的整个路由信息和用于客户端查询的队列信息。然后Producer和Consumer通过NameServer就可以知道整个Broker集群的路由信息，从而进行消息的投递和消费。NameServer通常也是集群的方式部署，各实例间相互不进行信息通讯。Broker是向每一台NameServer注册自己的路由信息，所以每一个NameServer实例上面都保存一份完整的路由信息。当某个NameServer因某种原因下线了，Broker仍然可以向其它NameServer同步其路由信息，Producer,Consumer仍然可以动态感知Broker的路由的信息。
- **BrokerServer:** Broker主要负责消息的存储、投递和查询以及服务高可用保证，为了实现这些功能，Broker包含了以下几个重要子模块。
 1. Remoting Module: 整个Broker的实体，负责处理来自clients端的请求。
 2. Client Manager: 负责管理客户端(Producer/Consumer)和维护Consumer的Topic订阅信息
 3. 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。
 4. HA Service: 高可用服务，提供Master Broker 和 Slave Broker之间的数据同步功能。
 5. 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。



2 RocketMQ Server安装

RocketMQ依赖Java环境，要求有JDK 1.8以上版本；

支持Windows和Linux平台；支持源码方式安装和使用已经编译好的安装包安装；

我们用windows平台安装RocketMQ Server编译好的安装包，来讲解RocketMQ；

下载地址：<https://rocketmq.apache.org/downloading/releases/>

我们用最新版本：4.9.0 release：<https://www.apache.org/dyn/closer.cgi?path=rocketmq/4.9.0/rocketmq-all-4.9.0-bin-release.zip>

解压后的目录：

rocketmq-all-4.9.0-bin-release		
名称		修改日期
benchmark	可执行脚本	2021/8/19/星期四 6
bin	配置文件	2021/8/19/星期四 6
conf		2021/8/19/星期四 6
lib	依赖的jar包	2021/8/19/星期四 6
LICENSE		2021/6/9/星期三 13
NOTICE		2021/6/9/星期三 13
README.md		2021/6/9/星期三 13

benchmark：里面是测试Demo；

bin：可执行脚本；

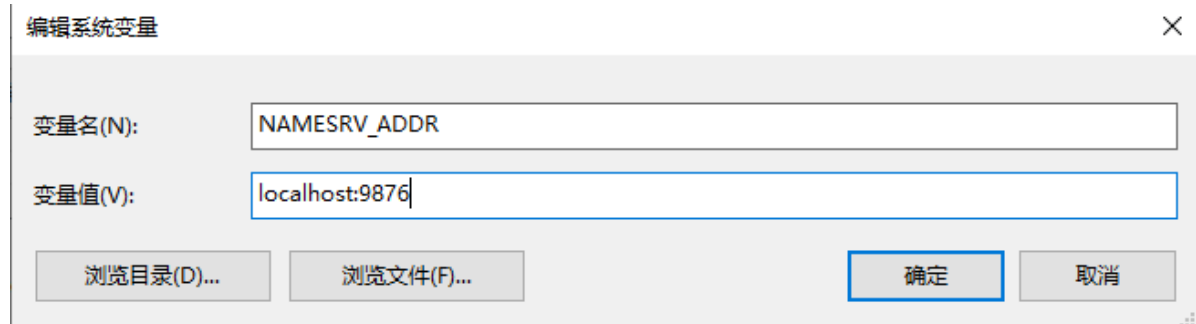
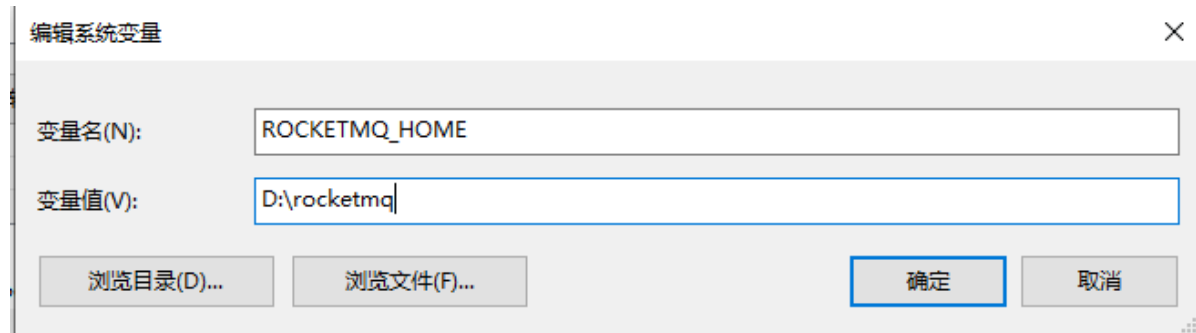
conf: 配置文件;

lib: 依赖的jar包;

我们把rocketmq解压包放到D盘根目录, 重命名 `rocketmq`;

第一步: 系统环境变量加两个配置

```
ROCKETMQ_HOME="D:\rocketmq"  
NAMESRV_ADDR="localhost:9876"
```



第二步: 启动Name Server

进入命令行执行:

```
.\bin\mqnamesrv.cmd
```

```
C:\Windows\System32\cmd.exe - mqnamesrv.cmd
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

D:\rocketmq\bin>mqnamesrv.cmd
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
```

第三步：启动Broker

进入命令行执行：

```
.\bin\mqbroker.cmd -n localhost:9876 autoCreateTopicEnable=true
```

```
C:\Windows\System32\cmd.exe - mqbroker.cmd -n localhost:9876 autoCreateTopicEnable=true
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

D:\rocketmq\bin>mqbroker.cmd -n localhost:9876 autoCreateTopicEnable=true
The broker[LAPTOP-84DB3AUV, 192.168.0.103:10911] boot success. serializeType=JSON and name server is localhost:9876
```

第四步：发送和接收消息测试

进入命令行消息发送执行：

```
.\bin\tools.cmd org.apache.rocketmq.example.quickstart.Producer
```



```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

D:\rocketmq\bin>tools.cmd org.apache.rocketmq.example.quickstart.Producer
07:40:41.797 [main] DEBUG i.n.u.i.l.InternalLoggerFactory - Using SLF4J as the default logging framework
RocketMQLog:WARN No appenders could be found for logger (io.netty.util.internal.PlatformDependent0).
RocketMQLog:WARN Please initialize the logger system properly.
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842840000, offsetMsgId=COA8006700002A9F0000000000000000, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=0]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E58429F0001, offsetMsgId=COA8006700002A9F000000000000000C9, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=0]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842A10002, offsetMsgId=COA8006700002A9F00000000000000192, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=0]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842A50003, offsetMsgId=COA8006700002A9F0000000000000025B, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=0]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842A70004, offsetMsgId=COA8006700002A9F00000000000000324, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=1]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842A90005, offsetMsgId=COA8006700002A9F000000000000003ED, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=1]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842AA0006, offsetMsgId=COA8006700002A9F000000000000004B6, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=1]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842AC0007, offsetMsgId=COA8006700002A9F0000000000000057F, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=1]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842AD0008, offsetMsgId=COA8006700002A9F00000000000000648, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=2]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842AF0009, offsetMsgId=COA8006700002A9F00000000000000711, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=2]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842B0000A, offsetMsgId=COA8006700002A9F000000000000007DA, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=2]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842B2000B, offsetMsgId=COA8006700002A9F000000000000008A4, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=2]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842B5000C, offsetMsgId=COA8006700002A9F0000000000000096E, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=3]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842B6000D, offsetMsgId=COA8006700002A9F00000000000000A38, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=3]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842B8000E, offsetMsgId=COA8006700002A9F00000000000000B02, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=3]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842BA000F, offsetMsgId=COA8006700002A9F00000000000000BCC, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=3]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842BC0010, offsetMsgId=COA8006700002A9F00000000000000C96, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=4]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842BE0011, offsetMsgId=COA8006700002A9F00000000000000D60, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=4]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842BF0012, offsetMsgId=COA8006700002A9F00000000000000E2A, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=4]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842C10013, offsetMsgId=COA8006700002A9F00000000000000EF4, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=4]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842C30014, offsetMsgId=COA8006700002A9F00000000000000FBE, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=5]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842C40015, offsetMsgId=COA8006700002A9F00000000000001088, messageQ
ueue=MessageQueue [topic=TopicTest, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=5]
SendResult [sendStatus=SEND_OK, msgId=7F00000140C82FF4ACD05E5842C60016, offsetMsgId=COA8006700002A9F00000000000001152, messageQ
```

消息发送成功；

进入命令行消息接收执行：

```
.\bin\tools.cmd org.apache.rocketmq.example.quickstart.Consumer
```

```
C:\Windows\System32\cmd.exe - tools.cmd org.apache.rocketmq.example.quickstart.Consumer
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

D:\rocketmq\bin>tools.cmd org.apache.rocketmq.example.quickstart.Consumer
07:43:17.924 [main] DEBUG i.n.u.i.l.InternalLoggerFactory - Using SLF4J as the default logging framework
Consumer Started.
ConsumeMessageThread_4 Receive New Messages: [MessageExt [brokerName=LAPTOP-84DB3AUV, queueId=2, storeSize=201, queueOffset=0, sys
Flag=0, bornTimestamp=1629330042533, bornHost=/192.168.0.103:52626, storeTimestamp=1629330042534, storeHost=/192.168.0.103:10911,
msgId=COA8006700002A9F0000000000000025B, commitLogOffset=603, bodyCRC=1032136437, reconsumeTimes=0, preparedTransactionOffset=0, to
String()=Message[topic=TopicTest, flag=0, properties={MIN_OFFSET=0, MAX_OFFSET=250, CONSUME_START_TIME=1629330198421, UNIQ_KEY=7
F00000140C82FF4ACD05E5842A50003, CLUSTER=DefaultCluster, WAIT=true, TAGS=TagA}, body=[72, 101, 108, 108, 111, 32, 82, 111, 99, 107,
101, 116, 77, 81, 32, 51], transactionId=null]]]
ConsumeMessageThread_13 Receive New Messages: [MessageExt [brokerName=LAPTOP-84DB3AUV, queueId=3, storeSize=202, queueOffset=3, sy
sFlag=0, bornTimestamp=1629330042549, bornHost=/192.168.0.103:52626, storeTimestamp=1629330042549, storeHost=/192.168.0.103:10911,
msgId=COA8006700002A9F0000000000000096E, commitLogOffset=2414, bodyCRC=1913501626, reconsumeTimes=0, preparedTransactionOffset=0,
toString()=Message[topic=TopicTest, flag=0, properties={MIN_OFFSET=0, MAX_OFFSET=250, CONSUME_START_TIME=1629330198421, UNIQ_KEY
=7F00000140C82FF4ACD05E5842B5000C, CLUSTER=DefaultCluster, WAIT=true, TAGS=TagA}, body=[72, 101, 108, 108, 111, 32, 82, 111, 99, 1
07, 101, 116, 77, 81, 32, 49, 50], transactionId=null]]]
ConsumeMessageThread_17 Receive New Messages: [MessageExt [brokerName=LAPTOP-84DB3AUV, queueId=2, storeSize=202, queueOffset=4, sy
sFlag=0, bornTimestamp=1629330042561, bornHost=/192.168.0.103:52626, storeTimestamp=1629330042562, storeHost=/192.168.0.103:10911,
msgId=COA8006700002A9F000000000000000EF4, commitLogOffset=3828, bodyCRC=1918600882, reconsumeTimes=0, preparedTransactionOffset=0,
toString()=Message[topic=TopicTest, flag=0, properties={MIN_OFFSET=0, MAX_OFFSET=250, CONSUME_START_TIME=1629330198421, UNIQ_KEY
=7F00000140C82FF4ACD05E5842C10013, CLUSTER=DefaultCluster, WAIT=true, TAGS=TagA}, body=[72, 101, 108, 108, 111, 32, 82, 111, 99, 1
07, 101, 116, 77, 81, 32, 49, 57], transactionId=null]]]
ConsumeMessageThread_16 Receive New Messages: [MessageExt [brokerName=LAPTOP-84DB3AUV, queueId=0, storeSize=202, queueOffset=4, sy
sFlag=0, bornTimestamp=1629330042558, bornHost=/192.168.0.103:52626, storeTimestamp=1629330042558, storeHost=/192.168.0.103:10911,
msgId=COA8006700002A9F00000000000000D60, commitLogOffset=3424, bodyCRC=367242165, reconsumeTimes=0, preparedTransactionOffset=0, t
oString()=Message[topic=TopicTest, flag=0, properties={MIN_OFFSET=0, MAX_OFFSET=250, CONSUME_START_TIME=1629330198421, UNIQ_KEY
=7F00000140C82FF4ACD05E5842B80011, CLUSTER=DefaultCluster, WAIT=true, TAGS=TagA}, body=[72, 101, 108, 108, 111, 32, 82, 111, 99, 10
7, 101, 116, 77, 81, 32, 49, 55], transactionId=null]]]
ConsumeMessageThread_19 Receive New Messages: [MessageExt [brokerName=LAPTOP-84DB3AUV, queueId=1, storeSize=202, queueOffset=4, sy
sFlag=0, bornTimestamp=1629330042559, bornHost=/192.168.0.103:52626, storeTimestamp=1629330042560, storeHost=/192.168.0.103:10911,
msgId=COA8006700002A9F00000000000000E2A, commitLogOffset=3626, bodyCRC=89962020, reconsumeTimes=0, preparedTransactionOffset=0, to
String()=Message[topic=TopicTest, flag=0, properties={MIN_OFFSET=0, MAX_OFFSET=250, CONSUME_START_TIME=1629330198421, UNIQ_KEY=7
F00000140C82FF4ACD05E5842BF0012, CLUSTER=DefaultCluster, WAIT=true, TAGS=TagA}, body=[72, 101, 108, 108, 111, 32, 82, 111, 99, 107,
101, 116, 77, 81, 32, 49, 56], transactionId=null]]]
ConsumeMessageThread_15 Receive New Messages: [MessageExt [brokerName=LAPTOP-84DB3AUV, queueId=1, storeSize=202, queueOffset=3, sy
```

消息接收成功；



















第五步：关闭服务

windows下直接关闭命令行窗口即可；

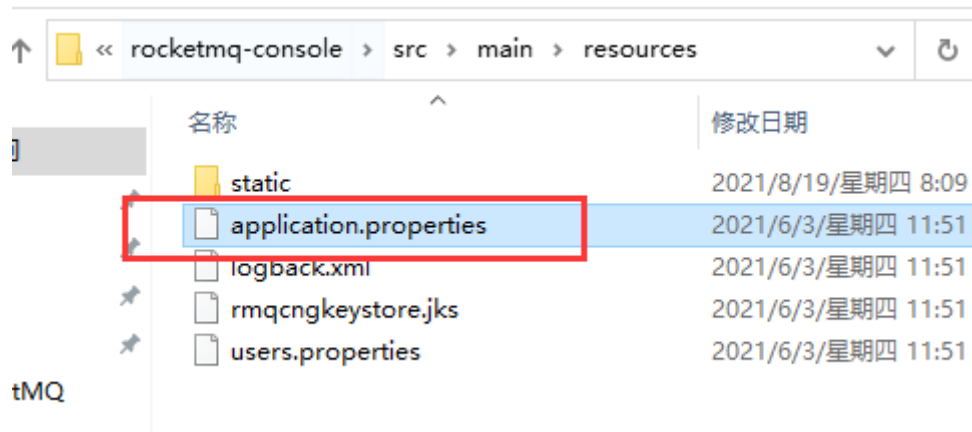
3 RocketMQ可视化控制台安装与使用

RocketMQ提供了一些扩展项目支持，地址：<https://github.com/apache/rocketmq-externals>

其中一个 `rocketmq-connect-console` 项目，就是我们需要的可视化控制台；

	<code>.github</code>	Update PULL_REQUEST_TEMPLATE.md	
	<code>dev</code>	[ROCKETMQ-236] Script to merge github pull request	
	<code>docs/connect/cn</code>	docs(connect): upload gitbook markdown code (#548)	1
	<code>rocketmq-cloudevents-binding</code>	Add rocketmq and cloudevents binding	
	<code>rocketmq-connect-activemq</code>	Change README.md (#301)	
	<code>rocketmq-connect-cassandra</code>	[ISSUE #570] ASoC runtime optimization: Cassandra connectors (#587)	1
	<code>rocketmq-connect-console</code>	Init rocketmq connect repo	
	<code>rocketmq-connect-es</code>	feat(connect) init rocketmq elastic search connector	
	<code>rocketmq-connect-jdbc</code>	Update README.md (#553)	
	<code>rocketmq-connect-jms</code>	fix(connect-jms) should put .iml file in gitignore	1
	<code>rocketmq-connect-kafka</code>	[ISSUE #420]remove openmessage-runtime dependency	
	<code>rocketmq-connect-mongo</code>	bump up the connect version to 0.1.1	1
	<code>rocketmq-connect-rabbitmq</code>	[ISSUE #312] Implement rocketmq connect RabbitMQ (#313)	
	<code>rocketmq-connect-redis</code>	Update rocketmq-connect-redis pom.xml (#592)	
	<code>rocketmq-connect</code>	[rocketmq-connect] Solve the problem of message duplication caused b...	
	<code>rocketmq-flink</code>	[#715] Support the RocketMQ TableSource based on the legacy Source i...	
	<code>rocketmq-flume</code>	[ISSUE #573] (rocketmq-flume)Replace the DefaultMQPullConsumer with...	1
	<code>rocketmq-hbase</code>	Added rocketmq-hbase (provides integration with the HBase datastore)....	

我们把整个项目下载下来，打开 `rocketmq-console` 项目；项目是SpringBoot开发；



打开 application.properties 配置文件，我们至少需要修改两个配置项；

server.port=8080，这个是可视化项目启动端口，我们改成8888；

rocketmq.config.namesrvAddr=，这个是指定nameServer地址和端口，我们暂时先搞成localhost:9876，等后面搞集群的话，要再修改；

```
server.address=0.0.0.0
server.port=8888

### SSL setting
#server.ssl.key-store=classpath:rmqcnngkeystore.jks
#server.ssl.key-store-password=rocketmq
#server.ssl.keyStoreType=PKCS12
#server.ssl.keyAlias=rmqcnngkey

#spring.application.index=true
spring.application.name=rocketmq-console
spring.http.encoding.charset=UTF-8
spring.http.encoding.enabled=true
spring.http.encoding.force=true
logging.level.root=INFO
logging.config=classpath:logback.xml
#if this value is empty, use env value rocketmq.config.namesrvAddr NAMESRV_ADDR | now, you can set it in
rocketmq.config.namesrvAddr=localhost:9876
#if you use rocketmq version < 3.5.8, rocketmq.config.isVIPChannel should be false.default true
rocketmq.config.isVIPChannel=
#rocketmq-console's data path:dashboard/monitor
rocketmq.config.dataPath=/tmp/rocketmq-console/data
#set it false if you don't want use dashboard.default true
rocketmq.config.enableDashBoardCollect=true
#set the message track trace topic if you don't want use the default one
rocketmq.config.msgTrackTopicName=
rocketmq.config.ticketKey=ticket

#Must create userInfo file: ${rocketmq.config.dataPath}/users.properties if the login is required
rocketmq.config.loginRequired=false

#set the accessKey and secretKey if you used acl
#rocketmq.config.accessKey=
#rocketmq.config.secretKey=
```

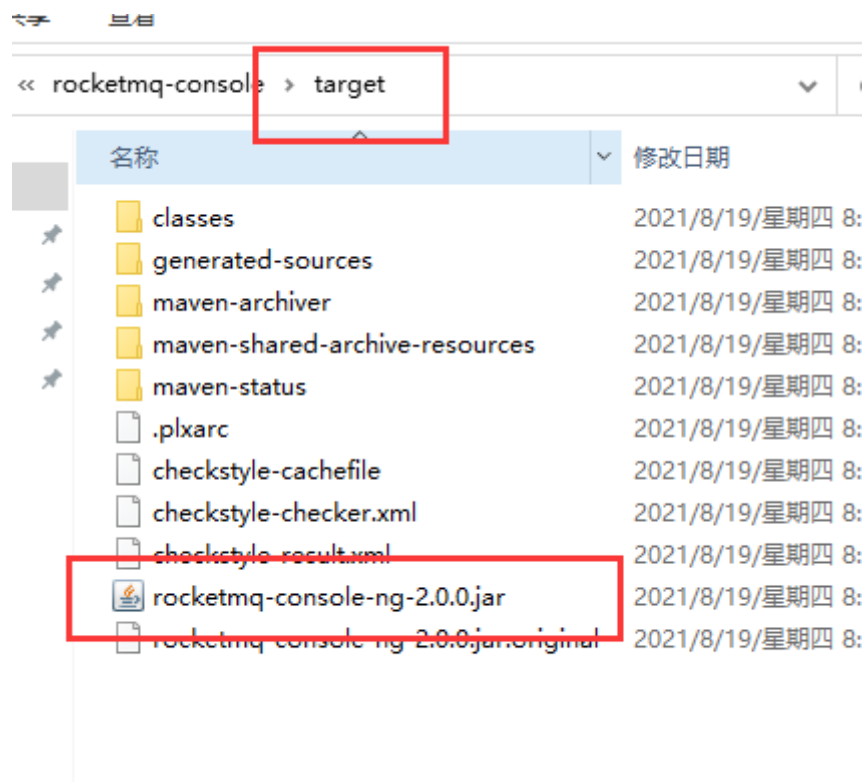
修改后保存，进入命令行，执行：

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

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\java1234\Desktop\rocketmq-externals-master\rocketmq-console>mvn clean package -Dmaven.test.skip=true
[INFO] Scanning for projects...
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/apache/18/apache-18.pom
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/apache/18/apache-18.pom (16 KB at 15.6 KB/sec)
[INFO]
[INFO] -----
[INFO] Building rocketmq-console-ng 2.0.0
[INFO] -----
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-remote-resources-plugin/1.5/maven-remote-resources-plugin-1.5.pom
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-remote-resources-plugin/1.5/maven-remote-resources-plugin-1.5.pom (14 KB at 16.3 KB/sec)
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-plugins/24/maven-plugins-24.pom
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-plugins/24/maven-plugins-24.pom (11 KB at 23.0 KB/sec)
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/maven-parent/23/maven-parent-23.pom
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/maven-parent/23/maven-parent-23.pom (32 KB at 59.1 KB/sec)
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/apache/13/apache-13.pom
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/apache/13/apache-13.pom (14 KB at 36.1 KB/sec)
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-remote-resources-plugin/1.5/maven-remote-resources-plugin-1.5.jar
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-remote-resources-plugin/1.5/maven-remote-resources-plugin-1.5.jar (68 KB at 122.3 KB/sec)
```

打包执行完后，在target目录，会生成一个可运行jar `rocketmq-console-ng-2.0.0.jar`



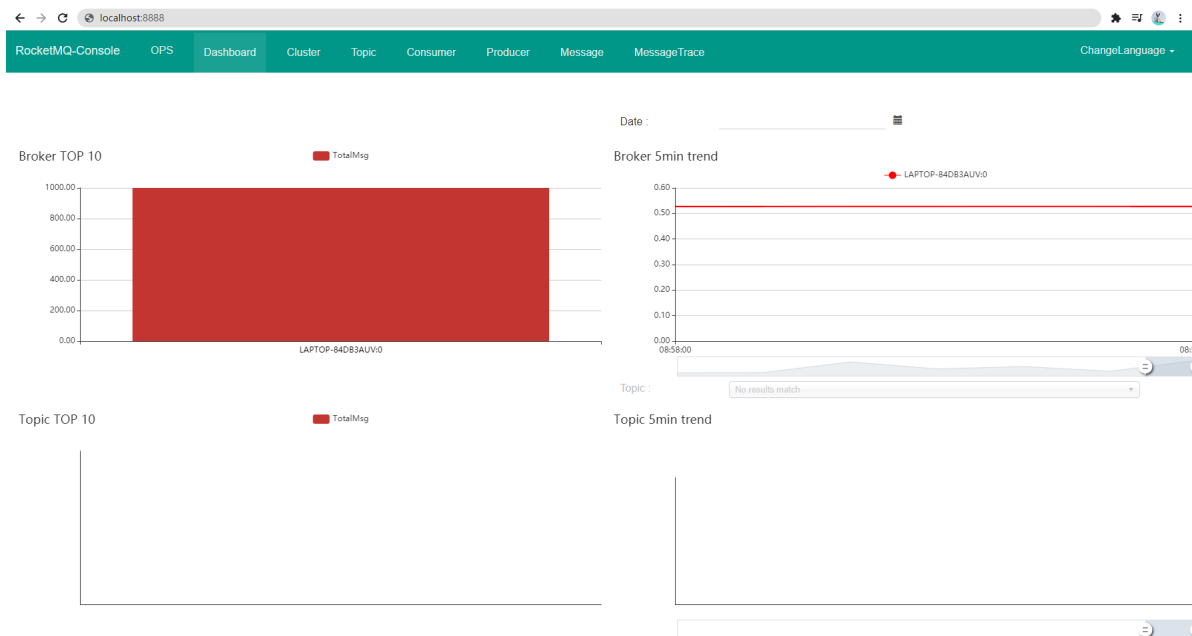
我们运行这个jar，进入命令行执行：

```
java -jar rocketmq-console-ng-2.0.0.jar
```

```
C:\Windows\System32\cmd.exe - java -jar rocketmq-console-ng-2.0.0.jar
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\java1234\Desktop\rocketmq-externals-master\rocketmq-console\target>java -jar rocketmq-console-ng-2.0.0.jar
08:52:28,979 -INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback-test.xml]
08:52:28,979 -INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.groovy]
08:52:28,979 -INFO in ch.qos.logback.classic.LoggerContext[default] - Found resource [logback.xml] at [jar:file:/C:/Users/java1234/Desktop/rocketmq-externals-master/rocketmq-console/target/rocketmq-console-ng-2.0.0.jar!/BOOT-INF/classes!/logback.xml]
08:52:28,991 -INFO in ch.qos.logback.core.joran.spi.ConfigurationWatchList@77a567e1 - URL [jar:file:/C:/Users/java1234/Desktop/rocketmq-externals-master/rocketmq-console/target/rocketmq-console-ng-2.0.0.jar!/BOOT-INF/classes!/logback.xml] is not of type file
08:52:29,021 -INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - debug attribute not set
08:52:29,021 -INFO in ch.qos.logback.core.joran.action.AppenderAction - About to instantiate appender of type [ch.qos.logback.core.ConsoleAppender]
08:52:29,026 -INFO in ch.qos.logback.core.joran.action.AppenderAction - Naming appender as [STDOUT]
08:52:29,029 -INFO in ch.qos.logback.core.joran.action.NestedComplexPropertyIA - Assuming default type [ch.qos.logback.classic.encoder.PatternLayoutEncoder] for [encoder] property
08:52:29,057 -INFO in ch.qos.logback.core.joran.action.AppenderAction - About to instantiate appender of type [ch.qos.logback.core.rolling.RollingFileAppender]
08:52:29,058 -INFO in ch.qos.logback.core.joran.action.AppenderAction - Naming appender as [FILE]
08:52:29,070 -INFO in c.q.l.core.rolling.TimeBasedRollingPolicy@1936628443 - No compression will be used
08:52:29,071 -INFO in c.q.l.core.rolling.TimeBasedRollingPolicy@1936628443 - Will use the pattern C:/Users/java1234/logs/consolelogs/rocketmq-console-%d{yyyy-MM-dd}.%i.log for the active file
08:52:29,072 -INFO in ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP@6d21714c - The date pattern is 'yyyy-MM-dd' from file name pattern 'C:/Users/java1234/logs/consolelogs/rocketmq-console-%d{yyyy-MM-dd}.%i.log'.
08:52:29,072 -INFO in ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP@6d21714c - Roll-over at midnight.
08:52:29,074 -INFO in ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP@6d21714c - Setting initial period to Thu Aug 19 08:52:29 CST 2021
08:52:29,075 -WARN in ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP@6d21714c - SizeAndTimeBasedFNATP is deprecated.
```

启动成功后，浏览器输入：<http://localhost:8888/>

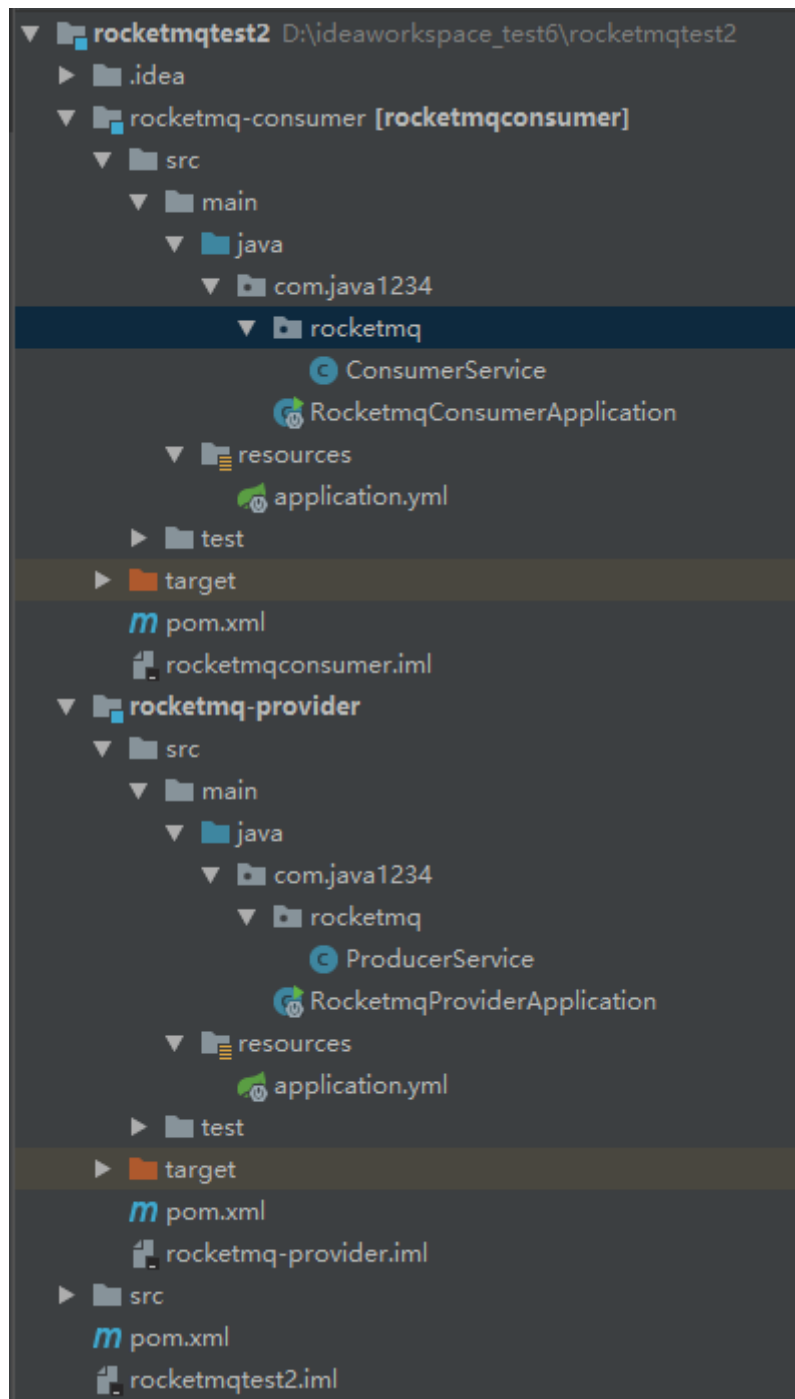


说明一切OK；

4 SpringBoot整合RocketMQ实现消息发送和接收

我们使用主流的SpringBoot框架整合RocketMQ来讲解，使用方便快捷；

最终项目结构如下：



具体步骤如下:

第一步: 我们新建一个父项目 `rocketmq-test`, `pom` 类型, 主要是依赖管理, 包括版本的管理, 以及管理 `module` 子项目

`pom.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <packaging>pom</packaging>
  <modules>
    <module>rocketmq-provider</module>
    <module>rocketmq-consumer</module>
  </modules>
```

```

<groupId>com.java1234</groupId>
<artifactId>rocketmq-test2</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <springboot.version>2.3.2.RELEASE</springboot.version>
    <rocketmq.version>2.2.0</rocketmq.version>
</properties>

<dependencyManagement>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>${springboot.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>

        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-spring-boot-starter</artifactId>
            <version>${rocketmq.version}</version>
        </dependency>
    </dependencies>

</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

</project>

```

第二步：新建消息生产者 rocketmq-provider 子项目

pom.xml 加下依赖：

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>

```

```

        <artifactId>rocketmq-test2</artifactId>
        <groupId>com.java1234</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>rocketmq-provider</artifactId>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-spring-boot-starter</artifactId>
        </dependency>

    </dependencies>

</project>

```

新建项目配置文件 `application.yml`，指定name-server，以及producer-group组

```

rocketmq:
  name-server: 127.0.0.1:9876
  producer:
    group: producer-demo1

```

新建消息生产者Service类 `ProducersService`

```

package com.java1234.rocketmq;

import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/**
 * 消息生产者
 * @author java1234_小锋
 * @site www.java1234.com
 * @company 南通小锋网络科技有限公司
 * @create 2021-08-22 22:16
 */
@Component("producersService")
public class ProducersService {

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

```



```

/**
 * 发送简单消息
 */
public void sendMessage(){
    for(int i=0;i<10;i++){
        rocketMQTemplate.convertAndSend("java1234-rocketmq","rocketmq大爷, 你好! "+i);
    }
}
}

```

SpringBoot给我们提供了 `RocketMQTemplate` 模板类，我们利用这个类可以以多种形式发送消息；另外这个类我们要加下 `@Component` 注解，让Spring来管理实例，方便其他地方获取bean来使用；发送方法指定Topic主题 `java1234-rocketmq`；

启动类获取 `ProducerService` 实例，调用发送消息方法

```

package com.java1234;

import com.java1234.rocketmq.ProducerService;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication
public class RocketmqTestApplication {

    public static void main(String[] args) {
        ConfigurableApplicationContext run =
        SpringApplication.run(RocketmqTestApplication.class, args);
        ProducerService producerService = (ProducerService)
        run.getBean("producerService");
        producerService.sendMessage();
    }

}

```

我们获取 `ProducerService` 实例，调用 `sendMessage` 方法发送消息；

第三步：新建消息消费者 `rocketmq-consumer` 子项目

`pom.xml` 加下依赖：

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>rocketmq-test2</artifactId>
        <groupId>com.java1234</groupId>
        <version>1.0-SNAPSHOT</version>

```

```

</parent>
<modelVersion>4.0.0</modelVersion>

<artifactId>rocketmq-consumer</artifactId>

<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.apache.rocketmq</groupId>
        <artifactId>rocketmq-spring-boot-starter</artifactId>
    </dependency>

</dependencies>

</project>

```

新建项目配置文件 `application.yml`，指定 `name-server`，以及 `consumer-group` 组

```

server:
  port: 8084
  servlet:
    context-path: /

rocketmq:
  name-server: 127.0.0.1:9876
  consumer:
    group: consumer-demo1

```

新建消息消费者 `Service` 类 `ConsumerService`，监听消息，消费消息

```

package com.java1234.rocketmq;

import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

/**
 * 消息消费者
 * @author java1234_小锋
 * @site www.java1234.com
 * @company 南通小锋网络科技有限公司
 * @create 2021-08-22 22:40
 */
@RocketMQMessageListener(topic = "java1234-rocketmq", consumerGroup = "${rocketmq.consumer.group}")
@Component
public class ConsumerService implements RocketMQListener<String> {

```

```
@Override
public void onMessage(String s) {
    System.out.println("收到消息内容: "+s);
}
}
```

消费者类要实现 `RocketMQListener` 接口，以及动态指定消息类型String。

类上要加上`@RocketMQMessageListener`注解，指定topic主题 `java1234-rocketmq`，以及消费者组 `${rocketmq.consumer.group}`

同样这个类上也要加上 `@Component` 注解，让Spring来管理bean实例；

启动类:

```
package com.java1234;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * @author java1234_小锋
 * @site www.java1234.com
 * @company 南通小锋网络科技有限公司
 * @create 2021-09-05 11:59
 */
@SpringBootApplication
public class RocketmqConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(RocketmqConsumerApplication.class, args);
    }
}
```

第四步：测试

先启动 rocketmq-consumer 项目，监听消息

[illegible]

再启动 `rockeqmq-provider` 项目，发送消息

```
Dr\jdk1.8\bin\java.exe ...

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  __/
      |_|_|_|

:: Spring Boot :: (v2.3.2.RELEASE)

2021-08-26 13:02:38.926 INFO 168588 --- [main] c.java1234.RocketmqProviderApplication : Starting RocketmqProviderApplication on LAPTOP-84DB3AUV with PID 168588 (D:\ideawor
2021-08-26 13:02:38.928 INFO 168588 --- [main] c.java1234.RocketmqProviderApplication : No active profile set, falling back to default profiles: default
2021-08-26 13:02:39.418 INFO 168588 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-08-26 13:02:39.423 INFO 168588 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-08-26 13:02:39.423 INFO 168588 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2021-08-26 13:02:39.490 INFO 168588 --- [main] o.s.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-08-26 13:02:39.490 INFO 168588 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 537 ms
2021-08-26 13:02:41.709 INFO 168588 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-08-26 13:02:41.828 INFO 168588 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-08-26 13:02:41.834 INFO 168588 --- [main] c.java1234.RocketmqProviderApplication : Started RocketmqProviderApplication in 3.119 seconds (JVM running for 3.826)
```

消息消费者端收到消息：

```
消费者2: 收到消息内容: rocketmq大爷, 你好! 0
消费者2: 收到消息内容: rocketmq大爷, 你好! 1
消费者2: 收到消息内容: rocketmq大爷, 你好! 2
消费者2: 收到消息内容: rocketmq大爷, 你好! 3
消费者2: 收到消息内容: rocketmq大爷, 你好! 5
消费者2: 收到消息内容: rocketmq大爷, 你好! 6
消费者2: 收到消息内容: rocketmq大爷, 你好! 4
消费者2: 收到消息内容: rocketmq大爷, 你好! 7
消费者2: 收到消息内容: rocketmq大爷, 你好! 8
消费者2: 收到消息内容: rocketmq大爷, 你好! 9
```

测试OK，成功消费！

5 RocketMQ发送同步消息

发送同步消息是指producer向 broker 发送消息，执行 API 时同步等待，直到broker 服务器返回发送结果；

相对异步发送消息，同步会阻塞线程，性能相对差点，但是可靠性高，这种方式得到广泛使用，比如：短信通知，邮件通知，站内重要信息通知等。

RocketMQTemplate 给我们提供了syncSend方法(有多个重载)，来实现发送同步消息；

下面给一个实例：

```
/**
 * 发送同步消息
 */
public void sendSyncMessage(){
    for(int i=0;i<10;i++){
        SendResult sendResult = rocketMQTemplate.syncSend("java1234-rocketmq", "rocketmq同步消息! "+i);
        System.out.println(sendResult);
    }
}
```

这里执行完发送同步消息返回执行结果 SendResult；

```
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366D7A0000, offsetMsgId=C0A806700002A9F0000000000021E378, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=789]
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E870002, offsetMsgId=C0A806700002A9F0000000000021E4AD, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=789]
收到消息内容: rocketmq同步消息! 0
收到消息内容: rocketmq同步消息! 1
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E8E0004, offsetMsgId=C0A806700002A9F0000000000021E5E2, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=789]
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E90000A, offsetMsgId=C0A806700002A9F0000000000021E717, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=789]
收到消息内容: rocketmq同步消息! 2
收到消息内容: rocketmq同步消息! 3
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E91000C, offsetMsgId=C0A806700002A9F0000000000021E84C, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=800]
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E920012, offsetMsgId=C0A806700002A9F0000000000021E961, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=790]
收到消息内容: rocketmq同步消息! 4
收到消息内容: rocketmq同步消息! 5
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E930014, offsetMsgId=C0A806700002A9F0000000000021EA86, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=2], queueOffset=784]
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E94001A, offsetMsgId=C0A806700002A9F0000000000021EBE3, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=3], queueOffset=784]
收到消息内容: rocketmq同步消息! 6
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E95001C, offsetMsgId=C0A806700002A9F0000000000021ED20, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=0], queueOffset=801]
SendResult [sendStatus=SEND_OK, msgId=7F00000132D01B84AC278366E960020, offsetMsgId=C0A806700002A9F0000000000021EE55, messageQueue=MessageQueue [topic=java1234-rocketmq, brokerName=LAPTOP-84DB3AUV, queueId=1], queueOffset=791]
收到消息内容: rocketmq同步消息! 7
收到消息内容: rocketmq同步消息! 8
收到消息内容: rocketmq同步消息! 9
```

运行测试OK;

6 RocketMQ发送异步消息

发送异步消息是指producer向 broker 发送消息时指定消息发送成功及发送异常的回调方法，调用 API 后立即返回，producer发送消息线程不阻塞，消息发送成功或失败的回调任务在一个新的线程中执行。

相对发送同步消息，异步消息性能更高，可靠性略差。适合对响应时间要求高的业务场景。

`RocketMQTemplate` 给我们提供了`asyncSend`方法(有多个重载)，来实现发送异步消息；

下面给一个实例：

```
/**
 * 发送异步消息
 */
public void sendAsyncMessage(){
    for(int i=0;i<10;i++){
        rocketMQTemplate.asyncSend("java1234-rocketmq", "rocketmq异步消息! "+i,
        new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                System.out.println("发送成功! ");
            }

            @Override
            public void onException(Throwable throwable) {
                System.out.println("发送失败! ");
            }
        });
    }
}
```

类似发送同步消息，多了一个`SendCallback`回调接口参数，实现`onSuccess`和`onException`方法，分别表示异步发送成功和失败；

```
发送成功！
发送成功！
发送成功！
发送成功！
发送成功！
发送成功！
发送成功！
发送成功！
发送成功！
发送成功！
收到消息内容：rocketmq异步消息！ 5
收到消息内容：rocketmq异步消息！ 1
收到消息内容：rocketmq异步消息！ 7
收到消息内容：rocketmq异步消息！ 9
收到消息内容：rocketmq异步消息！ 4
收到消息内容：rocketmq异步消息！ 0
收到消息内容：rocketmq异步消息！ 2
收到消息内容：rocketmq异步消息！ 6
收到消息内容：rocketmq异步消息！ 8
收到消息内容：rocketmq异步消息！ 3
|
```

运行测试OK!

7 RocketMQ发送单向消息

发送单向消息是指producer向 broker 发送消息，执行 API 时直接返回，不等待broker 服务器的结果。

这种方式主要用在不特别关心发送结果的场景，举例：日志发送；

`RocketMQTemplate` 给我们提供了`sendOneWay`方法(有多个重载)，来实现发送单向消息；

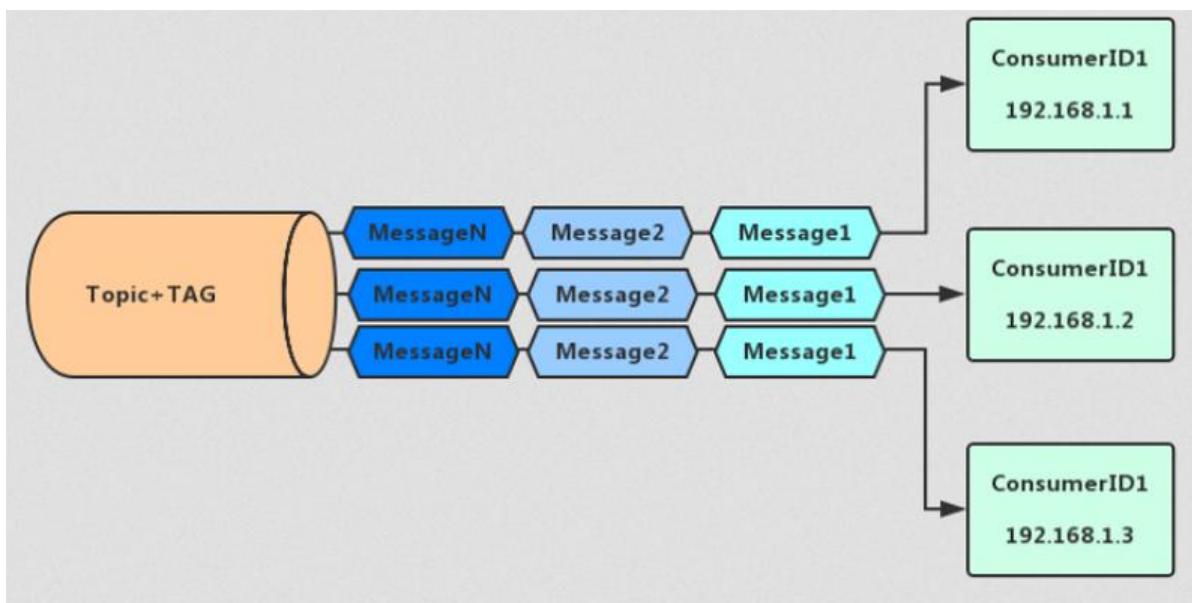
下面给一个实例：

```
/**
 * 发送单向消息
 */
public void sendOneWayMessage(){
    for(int i=0;i<10;i++){
        rocketMQTemplate.sendOneWay("java1234-rocketmq", "rocketmq单向消息! "+i);
    }
}
```

```
收到消息内容: rocketmq单向消息! 0
收到消息内容: rocketmq单向消息! 1
收到消息内容: rocketmq单向消息! 2
收到消息内容: rocketmq单向消息! 3
收到消息内容: rocketmq单向消息! 4
收到消息内容: rocketmq单向消息! 8
收到消息内容: rocketmq单向消息! 7
收到消息内容: rocketmq单向消息! 6
收到消息内容: rocketmq单向消息! 5
收到消息内容: rocketmq单向消息! 9
```

运行测试OK!

8 RocketMQ消费者广播模式和负载均衡模式



如上图，假如我们有多消费者，消息生产者发送的消息，是每一个消费者都消费一次呢？还是通过一些机制，比如轮询机制，每个消息只被某一个消费者消费一次呢？

这里涉及到消费者的消费模式，一种是广播模式，还有一种是负载均衡模式；

广播模式是每个消费者，都会消费消息；

负载均衡模式是每一个消息只会被某一个消费者消费一次；

我们业务上一般用的是负载均衡模式，当然一些特殊场景需要用到广播模式，比如发送一个信息到邮箱，手机，站内提示；

我们可以通过 `@RocketMQMessageListener` 的 `messageModel` 属性值来设置，

`MessageModel.BROADCASTING` 是广播模式，`MessageModel.CLUSTERING` 是默认集群负载均衡模式；

我们先集群负载均衡测试，加上 `messageModel=MessageModel.CLUSTERING`


```
package com.java1234.rocketmq;

import org.apache.rocketmq.spring.annotation.MessageModel;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

/**
 * 消息消费者
 * @author java1234_小峰
 * @site www.java1234.com
 * @company 南通小峰网络科技有限公司
 * @create 2021-08-22 22:40
 */
@RocketMQMessageListener(topic = "java1234-rocketmq", consumerGroup = "${rocketmq.consumer.group}", messageModel = MessageModel.CLUSTERING)
@Component
public class ConsumerService implements RocketMQListener<String> {

    @Override
    public void onMessage(String s) { System.out.println("消费者2: 收到消息内容: "+s); }

}
```

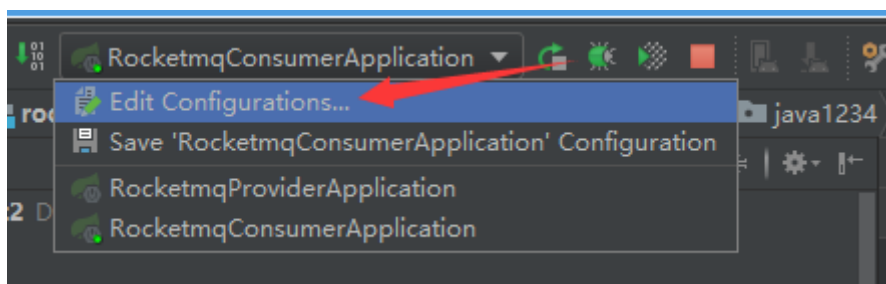
我们启动两个实例，先启动一个RocketmqConsumer消费者实例，端口8084

```
Spring Boot
Running
RocketmqConsumerApplication 8084
Configured
RocketmqProviderApplication

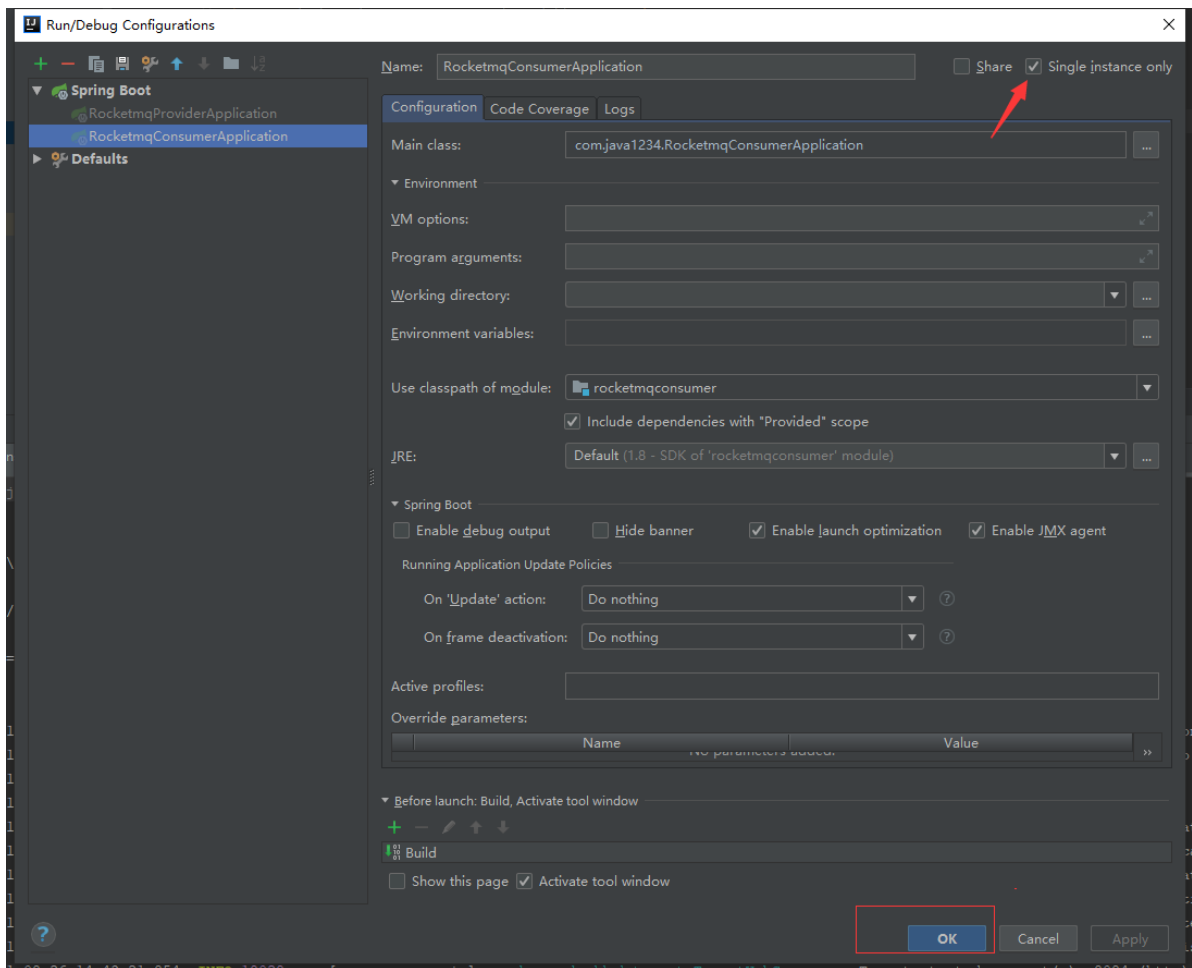
D:\jdk1.8\bin\java.exe ...

:: Spring Boot :: (v2.3.2.RELEASE)

2021-08-26 14:43:18.567 INFO 18028 --- [main] c.java1234.RocketmqConsumerApplication : Starting RocketmqConsumerApplication on LAPTOP-84DE3ADV
2021-08-26 14:43:18.569 INFO 18028 --- [main] c.java1234.RocketmqConsumerApplication : No active profile set, falling back to default profiles
2021-08-26 14:43:19.052 INFO 18028 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialised with port(s): 8084 (http)
2021-08-26 14:43:19.056 INFO 18028 --- [main] o.s.a.c.catalina.core.StandardService : Starting service [Tomcat]
2021-08-26 14:43:19.056 INFO 18028 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2021-08-26 14:43:19.117 INFO 18028 --- [main] o.s.c.o.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-08-26 14:43:19.118 INFO 18028 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in ...
2021-08-26 14:43:19.207 INFO 18028 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-08-26 14:43:21.840 INFO 18028 --- [main] o.s.s.a.DefaultRocketMQListenerContainer : running container: DefaultRocketMQListenerContainer(com
2021-08-26 14:43:21.841 INFO 18028 --- [main] o.s.s.a.a.ListenerContainerConfiguration : Register the listener [o container, listenerBeanName:com
2021-08-26 14:43:21.854 INFO 18028 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8084 (http) with context pat
2021-08-26 14:43:21.860 INFO 18028 --- [main] c.java1234.RocketmqConsumerApplication : Started RocketmqConsumerApplication in 3.497 seconds (U
```



编辑配置，



把 `Single instance only` 的勾选去掉，然后点OK按钮；

然后修改代码，启动端口改成 8085

```
server:
  port: 8085
  servlet:
    context-path: /

rocketmq:
  name-server: 127.0.0.1:9876
  consumer:
    group: consumer-demo1
```

消费者输出信息改下；

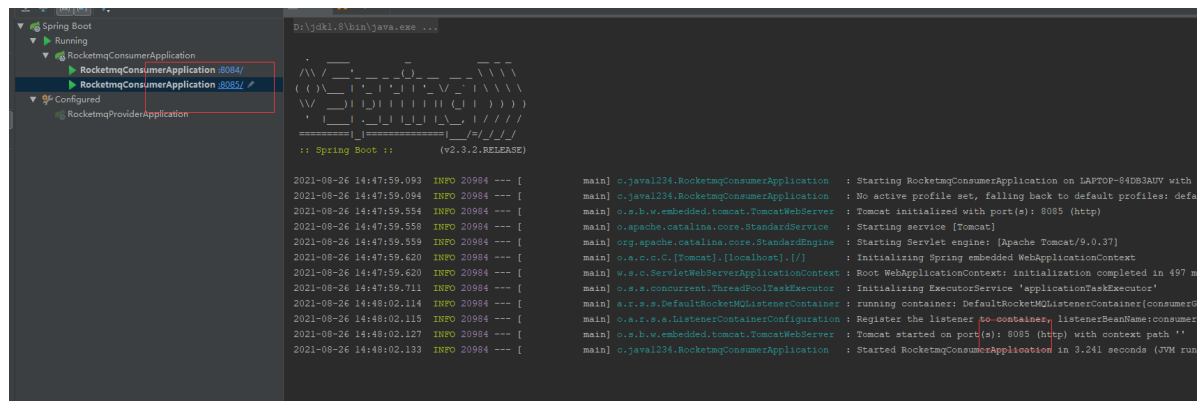
```

* @company 南通小锋网络科技有限公司
* @create 2021-08-22 22:40
*/
@RocketMQMessageListener(topic = "java1234-rocketmq", consumerGroup = "${rocketmq.consumer.group}", messageModel = MessageModel.CLUSTERING)
@Component
public class ConsumerService implements RocketMQListener<String> {

    @Override
    public void onMessage(String s) { System.out.println("消费者2: 收到消息内容: "+s); }

}
```

最后我们再启动一个消费者实例；

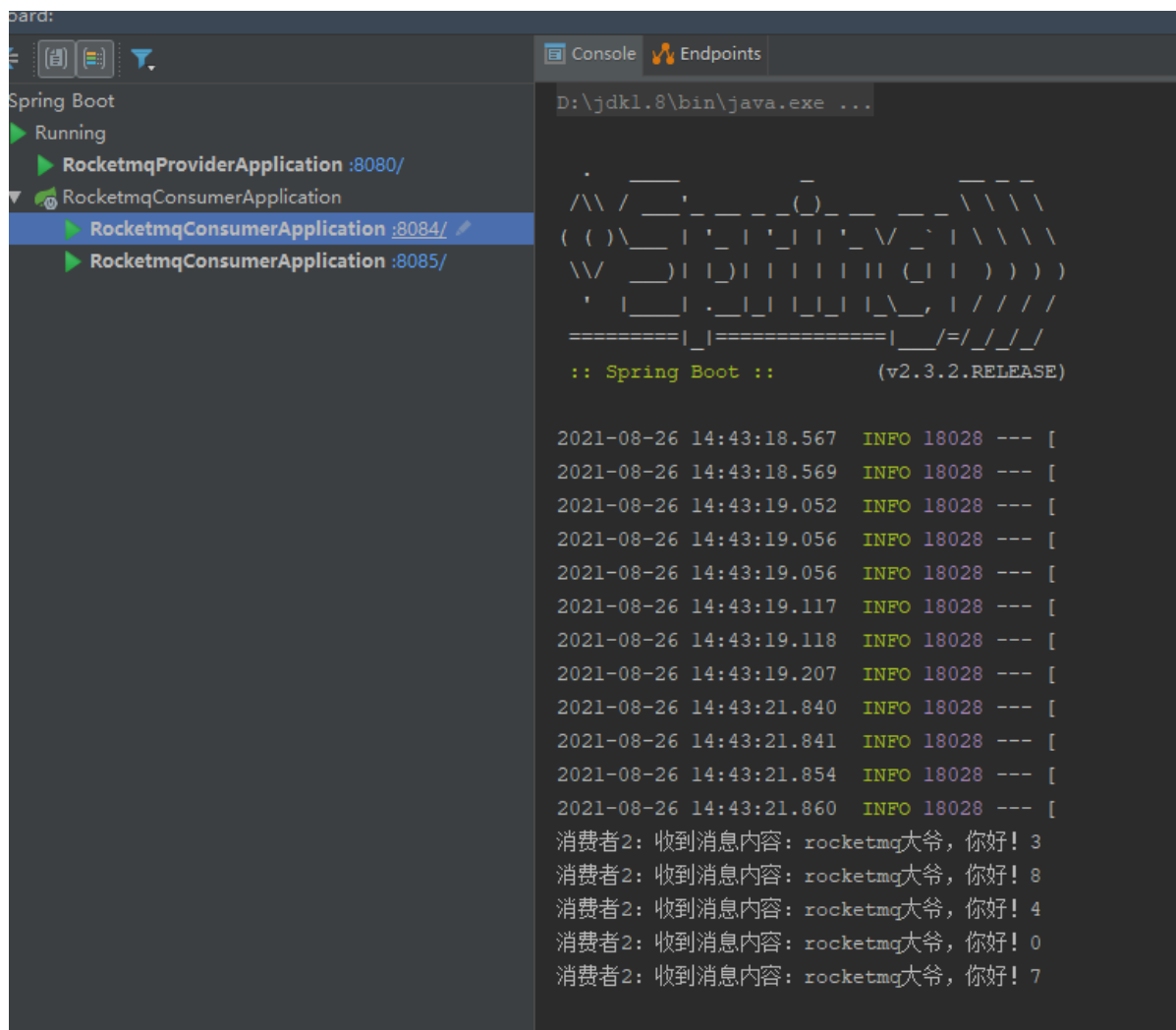


```
2021-08-26 14:47:59.093 INFO 20984 --- [
2021-08-26 14:47:59.094 INFO 20984 --- [
2021-08-26 14:47:59.554 INFO 20984 --- [
2021-08-26 14:47:59.558 INFO 20984 --- [
2021-08-26 14:47:59.559 INFO 20984 --- [
2021-08-26 14:47:59.620 INFO 20984 --- [
2021-08-26 14:47:59.711 INFO 20984 --- [
2021-08-26 14:48:02.114 INFO 20984 --- [
2021-08-26 14:48:02.115 INFO 20984 --- [
2021-08-26 14:48:02.127 INFO 20984 --- [
2021-08-26 14:48:02.133 INFO 20984 --- [

main] c:\java\1234.RocketmqConsumerApplication : Starting RocketmqConsumerApplication on LAPTOP-84DB3AUV with
main] c:\java\1234.RocketmqConsumerApplication : No active profile set, falling back to default profiles: defe
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8085 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
main] o.s.c.s.c.[Tomcat].[LocalHost].[/] : Initializing Spring embedded WebApplicationContext:
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 497 m
main] o.s.c.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
main] o.s.s.s.DefaultRocketmqListenerContainer : running container: DefaultRocketmqListenerContainer[consumer
main] o.s.s.s.s.ListenerContainerConfiguration : Register the listener so-consumer, listenerBeanName:consumer
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8085 (http) with context path ''
main] c:\java\1234.RocketmqConsumerApplication : Started RocketmqConsumerApplication in 3.241 seconds (JVM run
```

我们启动消息生产者测试：

启动后，两个消费者控制台分别输出：



```
2021-08-26 14:43:18.567 INFO 18028 --- [
2021-08-26 14:43:18.569 INFO 18028 --- [
2021-08-26 14:43:19.052 INFO 18028 --- [
2021-08-26 14:43:19.056 INFO 18028 --- [
2021-08-26 14:43:19.056 INFO 18028 --- [
2021-08-26 14:43:19.117 INFO 18028 --- [
2021-08-26 14:43:19.118 INFO 18028 --- [
2021-08-26 14:43:19.207 INFO 18028 --- [
2021-08-26 14:43:21.840 INFO 18028 --- [
2021-08-26 14:43:21.841 INFO 18028 --- [
2021-08-26 14:43:21.854 INFO 18028 --- [
2021-08-26 14:43:21.860 INFO 18028 --- [

消费者2: 收到消息内容: rocketmq大爷, 你好! 3
消费者2: 收到消息内容: rocketmq大爷, 你好! 8
消费者2: 收到消息内容: rocketmq大爷, 你好! 4
消费者2: 收到消息内容: rocketmq大爷, 你好! 0
消费者2: 收到消息内容: rocketmq大爷, 你好! 7
```


Spring Boot

Running

RocketmqProviderApplication :8080/

RocketmqConsumerApplication

RocketmqConsumerApplication :8085/

RocketmqConsumerApplication :8084/

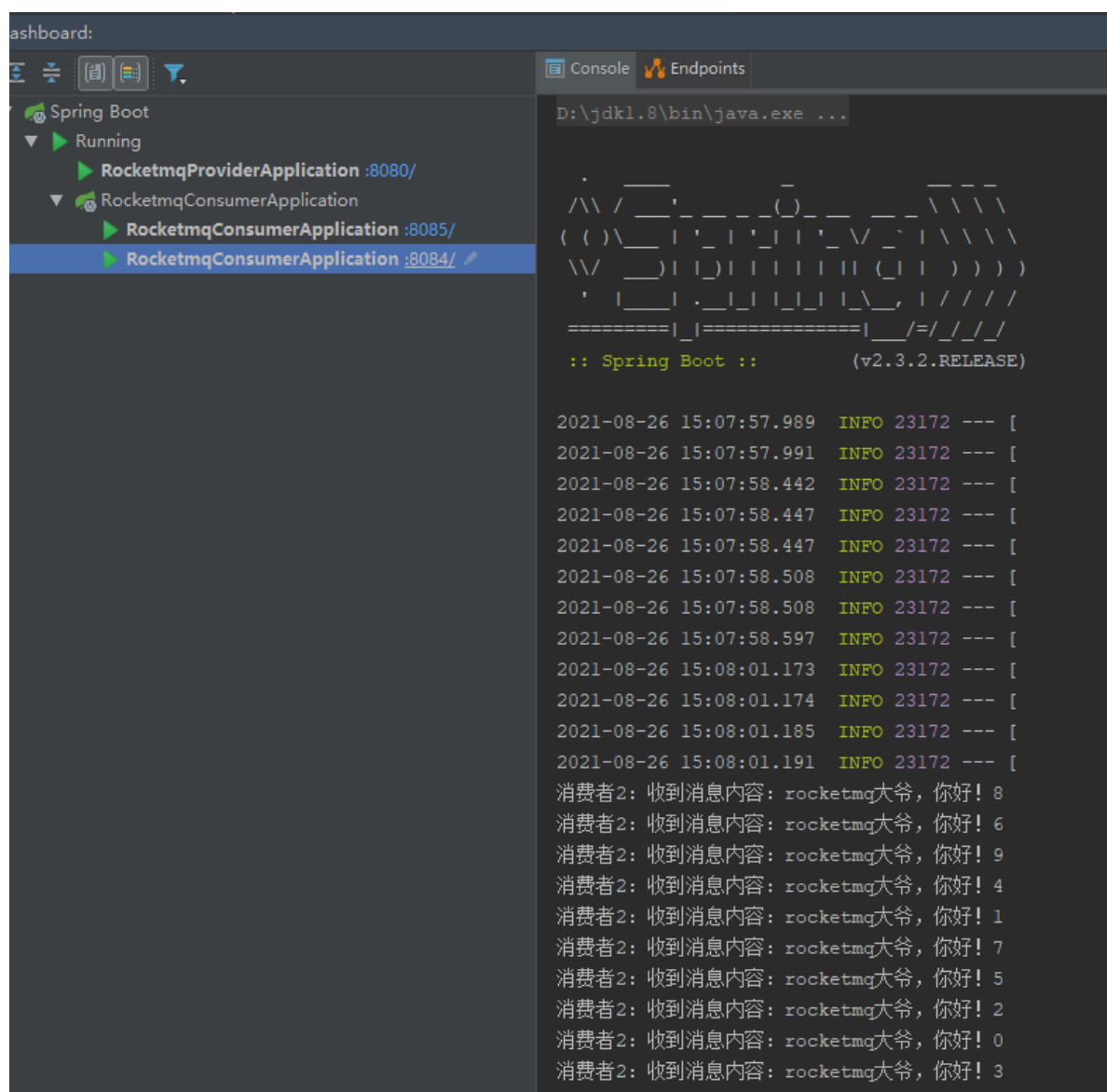
Console

Endpoints

D:\jdk1.8\bin\java.exe ...

:: Spring Boot :: (v2.3.2.RELEASE)

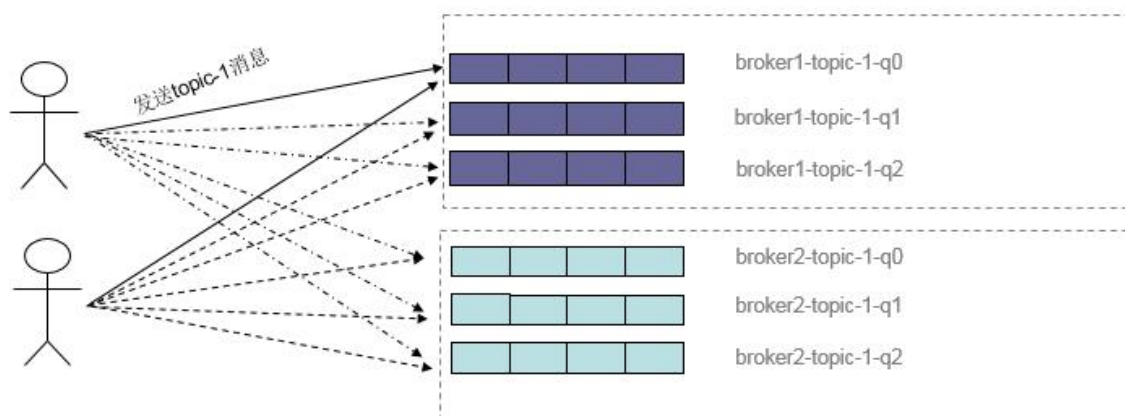
2021-08-26 15:07:24.474 INFO 2948 --- [
2021-08-26 15:07:24.476 INFO 2948 --- [
2021-08-26 15:07:24.961 INFO 2948 --- [
2021-08-26 15:07:24.966 INFO 2948 --- [
2021-08-26 15:07:24.967 INFO 2948 --- [
2021-08-26 15:07:25.030 INFO 2948 --- [
2021-08-26 15:07:25.030 INFO 2948 --- [
2021-08-26 15:07:25.119 INFO 2948 --- [
2021-08-26 15:07:27.606 INFO 2948 --- [
2021-08-26 15:07:27.606 INFO 2948 --- [
2021-08-26 15:07:27.618 INFO 2948 --- [
2021-08-26 15:07:27.624 INFO 2948 --- [
消费者1: 收到消息内容: rocketmq大爷, 你好! 4
消费者1: 收到消息内容: rocketmq大爷, 你好! 1
消费者1: 收到消息内容: rocketmq大爷, 你好! 5
消费者1: 收到消息内容: rocketmq大爷, 你好! 2
消费者1: 收到消息内容: rocketmq大爷, 你好! 3
消费者1: 收到消息内容: rocketmq大爷, 你好! 6
消费者1: 收到消息内容: rocketmq大爷, 你好! 7
消费者1: 收到消息内容: rocketmq大爷, 你好! 8
消费者1: 收到消息内容: rocketmq大爷, 你好! 0
消费者1: 收到消息内容: rocketmq大爷, 你好! 9



测试发现，两个消费者客户端把消息都各自消费了一遍。广播模式测试OK；

9 RocketMQ实现顺序消息

rocketmq默认发送的消息是进入多个消息队列，然后消费端多线程并发消费，所以默认情况，不是顺序消费消息的；



有时候，我们需要实现顺序消费一批消息，比如电商系统，订单创建，支付，完成等操作，需要顺序执行；

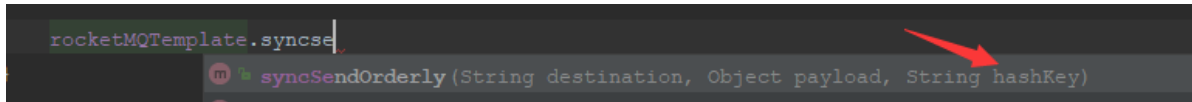
RocketMQTemplate 给我们提供了SendOrderly方法(有多个重载), 来实现发送顺序消息; 包括以下:

syncSendOrderly, 发送同步顺序消息;

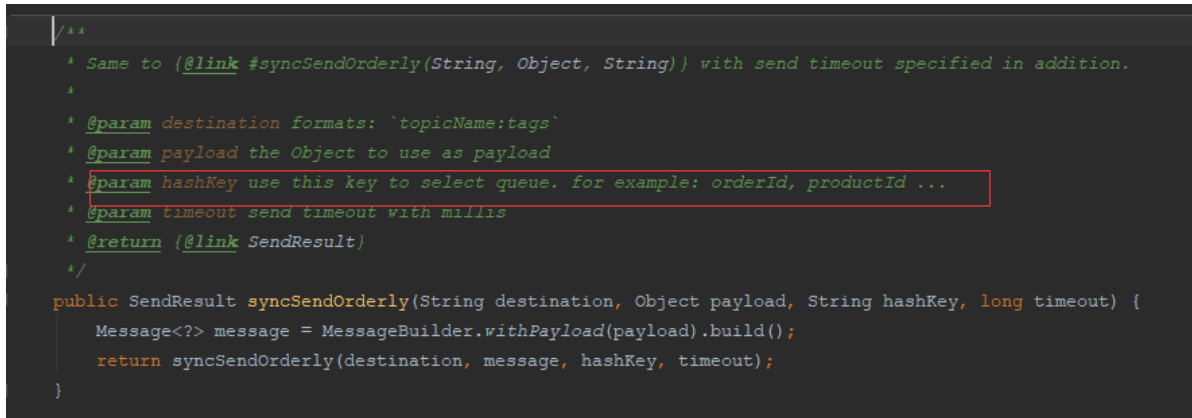
asyncSendOrderly, 发送异步顺序消息;

sendOneWayOrderly, 发送单向顺序消息;

一般我们用第一种发送同步顺序消息;



第三个参数 hashCode, 方法点进去:



因为broker会管理多个消息队列, 这个hashCode参数, 主要用来计算选择队列的, 一般可以把订单ID, 产品ID作为参数值;

发送到一个队列, 这样方便搞顺序队列;

以及消费端接收的时候, 默认是并发多线程去接收消息。RocketMQMessageListener有个属性consumeMode, 默认是ConsumeMode.CONCURRENTLY, 我们要改成ConsumeMode.ORDERLY, 单线程顺序接收消息;

下面给具体事例, 模拟两个订单发送消息;

消息生产者端:

```
/**
 * 发送同步顺序消息
 */
public void sendOrderlyMessage(){
    // hashCode用来计算决定消息发送到哪个消息队列 一般是订单ID, 产品ID等
    rocketMQTemplate.syncSendOrderly("java1234-rocketmq-orderly", "98456231,创建", "98456231");
    rocketMQTemplate.syncSendOrderly("java1234-rocketmq-orderly", "98456231,支付", "98456231");
    rocketMQTemplate.syncSendOrderly("java1234-rocketmq-orderly", "98456231,完成", "98456231");

    rocketMQTemplate.syncSendOrderly("java1234-rocketmq-orderly", "98456232,创建", "98456232");
    rocketMQTemplate.syncSendOrderly("java1234-rocketmq-orderly", "98456232,支付", "98456232");
    rocketMQTemplate.syncSendOrderly("java1234-rocketmq-orderly", "98456232,完成", "98456232");
}
```


消费者端：

```
/**
 * 消息消费者
 * @author java1234_小锋
 * @site www.java1234.com
 * @company 南通小锋网络科技有限公司
 * @create 2021-08-22 22:40
 */
@RocketMQMessageListener(topic = "java1234-rocketmq-orderly", consumerGroup
= "${rocketmq.consumer.group}", consumeMode = ConsumeMode.ORDERLY )
@Component
public class ConsumerService implements RocketMQListener<String> {

    @Override
    public void onMessage(String s) {
        System.out.println("消费者: 收到消息内容: "+s);
    }

}
```

运行测试：

```
2021-08-27 17:12:05.809 INFO 10496 ---
2021-08-27 17:12:05.815 INFO 10496 ---
消费者: 收到消息内容: 98456232,创建
消费者: 收到消息内容: 98456231,创建
消费者: 收到消息内容: 98456231,支付
消费者: 收到消息内容: 98456232,支付
消费者: 收到消息内容: 98456231,完成
消费者: 收到消息内容: 98456232,完成
```

没问题！

10 RocketMQ实现延迟消息

延迟消息

对于消息中间件来说，producer 将消息发送到mq的服务器上，但并不希望这条消息马上被消费，而是推迟到当前时间节点之后的某个时间点，再将消息投递到 queue 中让 consumer 进行消费。

延迟消息的使用场景很多，一种比较常见的场景就是在电商系统中，订单创建后，会有一个等待用户支付的时间窗口，一般为30分钟，30分钟后 customer 会收到这条订单消息，然后程序去订单表中检查当前这条订单的支付状态，如果是未支付的状态，则自动清理掉，这样就不需要使用定时任务的方式去处理了。

Rocket的延迟消息

RocketMQ 支持定时的延迟消息，但是不支持任意时间精度，仅支持特定的 level，例如定时 5s，10s，1m 等。其中，level=0 级表示不延时，level=1 表示 1 级延时，level=2 表示 2 级延时，以此类推。

```
messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
```

```
public void sendDelayMessage() {
    rocketMQTemplate.asyncSend( destination: "java1234-rocketmq", MessageBuilder.withPayload("rocketmq延迟1秒消息").build(), timeout: 3000, delayLevel: 1);
    rocketMQTemplate.asyncSend( destination: "java1234-rocketmq", MessageBuilder.withPayload("rocketmq延迟5秒消息").build(), timeout: 3000, delayLevel: 2);
    rocketMQTemplate.asyncSend( destination: "java1234-rocketmq", MessageBuilder.withPayload("rocketmq延迟10秒消息").build(), timeout: 3000, delayLevel: 3);
    rocketMQTemplate.send
}

sendOneWayOrderly(String destination, Object payload, String hashCode) void
sendOneWay(String destination, Message<?> message) void
sendOneWay(String destination, Object payload) void
send(Message<?> message) void
send(String destination, Message<?> message) void
sendOneWayOrderly(String destination, Message<?> message, String hashCode) void
Endpoints
sendAndReceive(String destination, Object payload, Type type) T
sendAndReceive(String destination, Message<?> message, Type type) T
sendAndReceive(String destination, Object payload, Type type, long timeout) T
sendAndReceive(String destination, Object payload, Type type, String hashCode) T
sendAndReceive(String destination, Message<?> message, Type type, long timeout) T
sendAndReceive(String destination, Message<?> message, Type type, String hashCode) T
sendAndReceive(String destination, Object payload, Type type, long timeout, int delayLevel) T
sendAndReceive(String destination, Object payload, Type type, String hashCode, long timeout) T
sendAndReceive(String destination, Message<?> message, Type type, long timeout, int delayLevel) T
sendAndReceive(String destination, Message<?> message, Type type, String hashCode, long timeout) T
sendAndReceive(String destination, Object payload, Type type, String hashCode, long timeout, int delayLevel) T
sendAndReceive(String destination, Object payload, RocketMQLocalRequestCallback rocketMQLocalRequestCallback) void
sendAndReceive(String destination, Message<?> message, Type type, String hashCode, long timeout, int delayLevel) T
sendAndReceive(String destination, Message<?> message, RocketMQLocalRequestCallback rocketMQLocalRequestCallback) void
sendAndReceive(String destination, Object payload, RocketMQLocalRequestCallback rocketMQLocalRequestCallback, long timeout) void
sendAndReceive(String destination, Object payload, RocketMQLocalRequestCallback rocketMQLocalRequestCallback, String hashCode) void
sendAndReceive(String destination, Message<?> message, RocketMQLocalRequestCallback rocketMQLocalRequestCallback, long timeout) void
sendAndReceive(String destination, Message<?> message, RocketMQLocalRequestCallback rocketMQLocalRequestCallback, String hashCode) void
sendAndReceive(String destination, Object payload, RocketMQLocalRequestCallback rocketMQLocalRequestCallback, long timeout, int delayLevel) void
```

我们会发现，所有消息发送方法都有一个带int类型的delayLevel参数重载方法，这个就是设置延迟消息级别的参数。

同时注意，每个带delayLevel参数的方法，也同时带有long类型的timeout参数，这个是设置消息发送超时时间，默认是3秒，我们也可以自行设置；

```
70
71      * See {@link http://rocketmq.apache.org/docs/core}
72      */
73      private String producerGroup;
74
75      /**
76       * Just for testing or demo program
77       */
78      private String createTopicKey = TopicValidator.AUTO_CREATE;
79
80      /**
81       * Number of queues to create per default topic.
82       */
83      private volatile int defaultTopicQueueNums = 4;
84
85      /**
86       * Timeout for sending messages.
87       */
88      private int sendMsgTimeout = 3000;
89
90      /**
91       * Compress message body threshold, namely, message body
92       */
93      private int compressMsgBodyOverHowmuch = 1024 * 4;
94
95      /**
96       * Maximum number of retry to perform internally before
97       */
```

同时还有 Message 参数，如果发送这种类型的消息，可以携带具体的消息数据；

我们给下实例：

```
/**
 * 发送延迟消息
 */
public void sendDelayMessage(){
    rocketMQTemplate.syncSend("java1234-rocketmq",MessageBuilder.withPayload("rocketmq延迟1秒消息").build(),3000,1);
    rocketMQTemplate.syncSend("java1234-rocketmq",MessageBuilder.withPayload("rocketmq延迟5秒消息").build(),3000,2);
    rocketMQTemplate.syncSend("java1234-rocketmq",MessageBuilder.withPayload("rocketmq延迟10秒消息").build(),3000,3);
}
```

运行测试：

```
2021-08-28 10:41:26.777 INFO 25852 --- [main] c.javal234
2021-08-28 10:41:26.779 INFO 25852 --- [main] c.javal234
2021-08-28 10:41:27.238 INFO 25852 --- [main] o.s.b.w.emb
2021-08-28 10:41:27.243 INFO 25852 --- [main] o.apache.ca
2021-08-28 10:41:27.243 INFO 25852 --- [main] org.apache
2021-08-28 10:41:27.305 INFO 25852 --- [main] o.a.c.c.C.
2021-08-28 10:41:27.305 INFO 25852 --- [main] w.s.c.Serv
2021-08-28 10:41:27.396 INFO 25852 --- [main] o.s.s.conc
2021-08-28 10:41:29.761 INFO 25852 --- [main] a.r.s.s.De
2021-08-28 10:41:29.762 INFO 25852 --- [main] o.a.r.s.a.f
2021-08-28 10:41:29.774 INFO 25852 --- [main] o.s.b.w.emb
2021-08-28 10:41:29.780 INFO 25852 --- [main] c.javal234
消费者：收到消息内容：rocketmq延迟1秒消息
消费者：收到消息内容：rocketmq延迟5秒消息
消费者：收到消息内容：rocketmq延迟10秒消息
```

没问题！

11 RocketMQ实现事务消息

事务消息是RocketMQ提供的非常重要的一个特性，在4.x版本之后开源，可以利用事务消息轻松地实现分布式事务。

RocketMQ在其消息定义的基础上，对事务消息扩展了两个相关的概念：

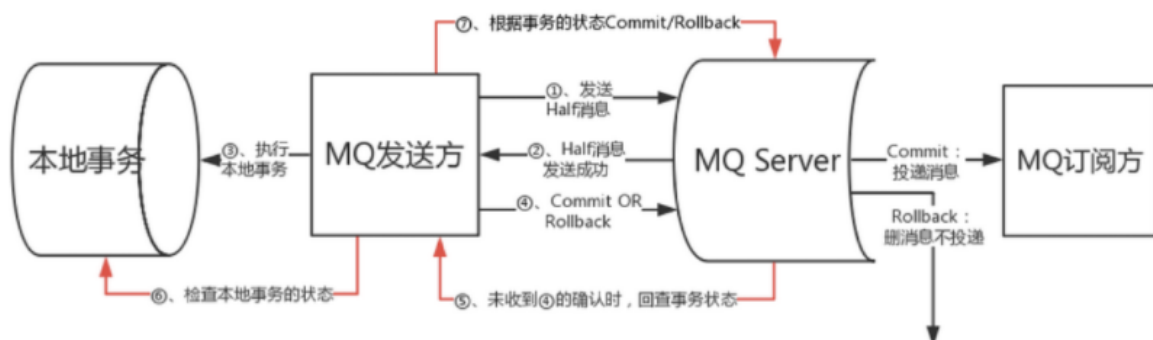
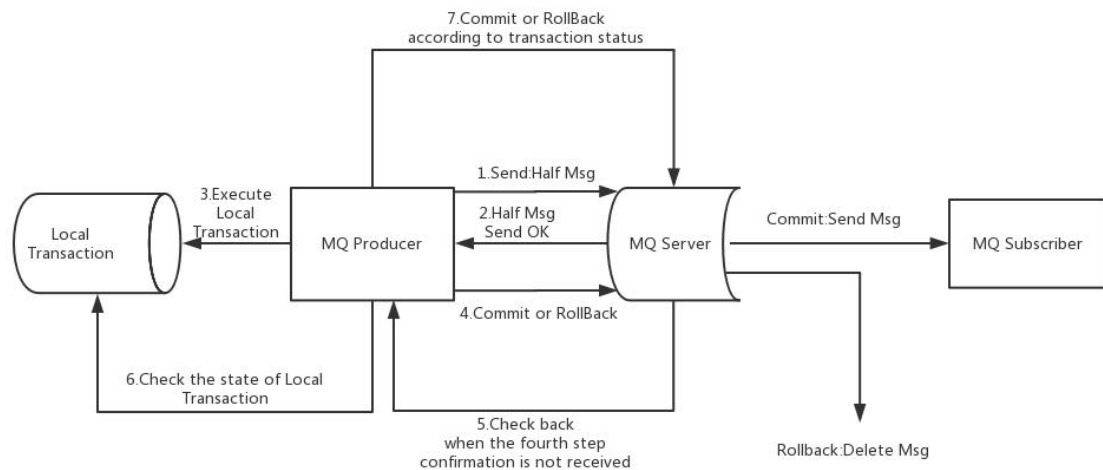
Half(Prepare) Message——半消息(预处理消息)

半消息是一种特殊的消息类型，该状态的消息暂时不能被Consumer消费。当一条事务消息被成功投递到Broker上，但是Broker并没有接收到Producer发出的二次确认时，该事务消息就处于"暂时不可被消费"状态，该状态的事务消息被称为半消息。

Message Status Check——消息状态回查

由于网络抖动、Producer重启等原因，可能导致Producer向Broker发送的二次确认消息没有成功送达。如果Broker检测到某条事务消息长时间处于半消息状态，则会主动向Producer端发起回查操作，查询该事务消息在Producer端的事务状态(Commit 或 Rollback)。可以看出，Message Status Check主要用来解决分布式事务中的超时问题。

执行流程：



1. 应用模块遇到要发送事务消息的场景时，先发送prepare消息给MQ。
2. prepare消息发送成功后，应用模块执行数据库事务（本地事务）。
3. 根据数据库事务执行的结果，再返回Commit或Rollback给MQ。
4. 如果是Commit，MQ把消息下发给Consumer端，如果是Rollback，直接删掉prepare消息。
5. 第3步的执行结果如果没响应，或是超时的，启动定时任务回查事务状态（最多重试15次，超过了默认丢弃此消息），处理结果同第4步。
6. MQ消费的成功机制由MQ自己保证。

具体实例：

通过 `rocketMQTemplate` 的 `sendMessageInTransaction` 方法发送事务消息

```
/**
 * 发送事务消息
 */
public void sendTransactionMessage(){
    // 构造消息
    Message msg = MessageBuilder.withPayload("rocketmq事务消息-01").build();
    rocketMQTemplate.sendMessageInTransaction("java1234-transaction-rocketmq",msg,null);
}
```

定义本地事务处理类，实现 `RocketMQLocalTransactionListener` 接口，以及加上 `@RocketMQTransactionListener` 注解，这个类似方法的调用是异步的；

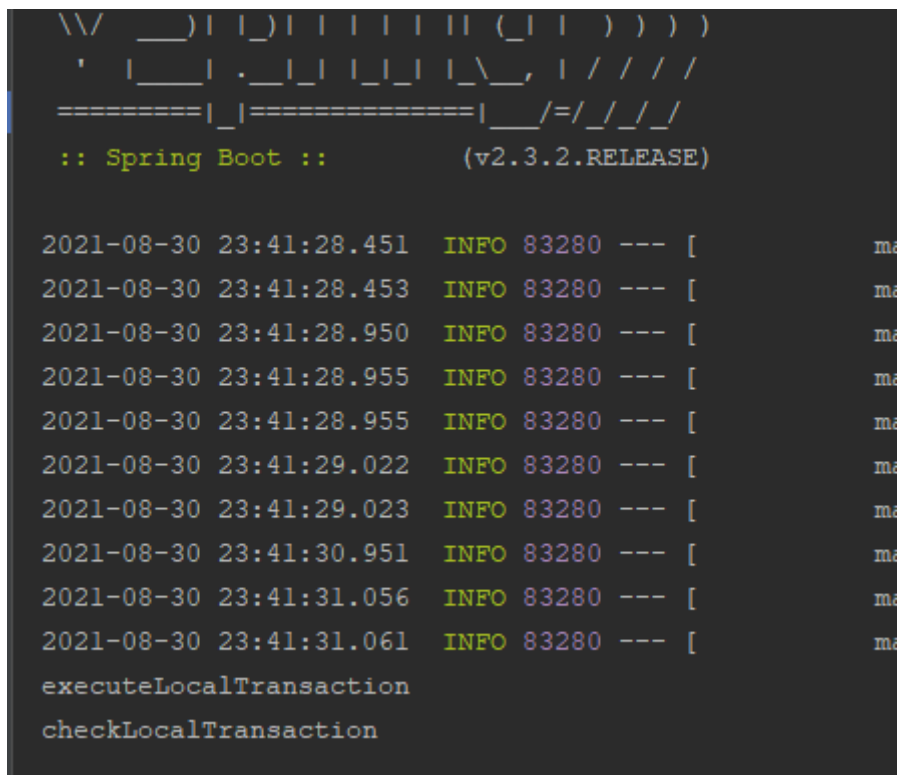
`executeLocalTransaction` 方法，当我们处理完业务后，可以根据业务处理情况，返回事务执行状态，有 `rollback`，`commit` 或 `unknown` 三种，分别是回滚事务，提交事务和未知；根据事务消息执行流程，如果返回`rollback`，则直接丢弃消息；如果是返回`commit`，则消费消息；如果是`unknown`，则继续等待，然后调用`checkLocalTransaction`方法，最多重试15次，超过了默认丢弃此消息；

`checkLocalTransaction` 方法，是当MQ Server未得到MQ发送方应答，或者超时的情况，或者应答是`unknown`的情况，调用此方法进行检查确认，返回值和上面的方法一样；

```
@RocketMQTransactionListener
class TransactionListenerImpl implements RocketMQLocalTransactionListener {
    @Override
    public RocketMQLocalTransactionState executeLocalTransaction(Message msg,
    Object arg) {
        // ... local transaction process, return rollback, commit or unknown
        System.out.println("executeLocalTransaction");
        return RocketMQLocalTransactionState.UNKNOWN;
    }

    @Override
    public RocketMQLocalTransactionState checkLocalTransaction(Message msg) {
        // ... check transaction status and return rollback, commit or unknown
        System.out.println("checkLocalTransaction");
        return RocketMQLocalTransactionState.COMMIT;
    }
}
```

运行：



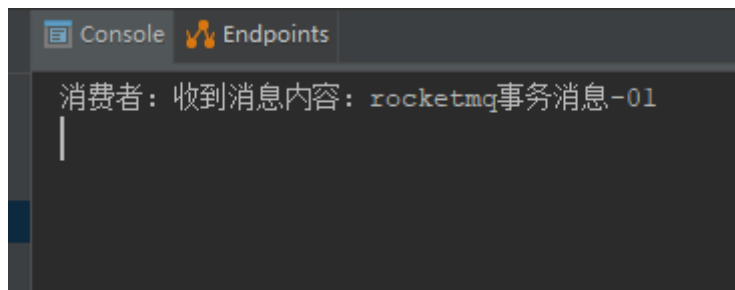
```
\W _ _ ) | | _ | | | | | | | | ( _ | ) ) ) )
' | _ _ | . _ | | | | | _ \ , | / / / /
=====|_|=====|_|_/ / / / /

:: Spring Boot ::                (v2.3.2.RELEASE)

2021-08-30 23:41:28.451 INFO 83280 --- [           ma
2021-08-30 23:41:28.453 INFO 83280 --- [           ma
2021-08-30 23:41:28.950 INFO 83280 --- [           ma
2021-08-30 23:41:28.955 INFO 83280 --- [           ma
2021-08-30 23:41:28.955 INFO 83280 --- [           ma
2021-08-30 23:41:29.022 INFO 83280 --- [           ma
2021-08-30 23:41:29.023 INFO 83280 --- [           ma
2021-08-30 23:41:30.951 INFO 83280 --- [           ma
2021-08-30 23:41:31.056 INFO 83280 --- [           ma
2021-08-30 23:41:31.061 INFO 83280 --- [           ma

executeLocalTransaction
checkLocalTransaction
```

生产者端两个方法都执行到了，



消费端也获取到了消息；

执行如下：

生产者端发送half消息到MQ-SERVER，然后异步执行executeLocalTransaction方法，返回unknown，MQ-SERVER接收到unknown后，继续等待，然后再执行checkLocalTransaction确认，返回commit，MQ-SERVER得到commit后，消费端才可以消费消息；

12 RocketMQ过滤消息

在消费端进行消息消费的时候，我们根据业务需求，可以对消息进行过滤，处理需要的消息；

尤其是广播模式下，消息过滤经常使用；

RocketMQ提供了TAG和SQL表达式两种消息过滤方式；

12.1 根据TAG方式过滤消息

消息发送端只能设置一个tag，消息接收端可以设置多个tag。

接收消息端通过 '|' 设置多个tag，如下：tag1 || tag2 || tag3 || ...

上实例,生产端发送三个消息，TAG分别是TAG1,TAG2,TAG3

```
/**
 * 发送带Tag消息，测试根据Tag过滤消息
 */
public void sendMessageWithTag(){
    // 构造消息1
    Message msg1 = MessageBuilder.withPayload("rocketmq过滤消息测试-TAG01").build();
    // 构造消息2
    Message msg2 = MessageBuilder.withPayload("rocketmq过滤消息测试-TAG02").build();
    // 构造消息3
    Message msg3 = MessageBuilder.withPayload("rocketmq过滤消息测试-TAG03").build();

    rocketMQTemplate.convertAndSend("java1234-filter-rocketmq" + ":" + "TAG1", msg1);
    rocketMQTemplate.convertAndSend("java1234-filter-rocketmq" + ":" + "TAG2", msg2);
    rocketMQTemplate.convertAndSend("java1234-filter-rocketmq" + ":" + "TAG3", msg3);
}
```


消费端，通过selectorExpression = "TAG1 || TAG2",selectorType = SelectorType.TAG，指定需要消费的TAG

```
@RocketMQMessageListener(topic = "java1234-filter-rocketmq",consumerGroup
="${rocketmq.consumer.group}" ,selectorExpression = "TAG1 || TAG2",selectorType
= SelectorType.TAG)
@Component
public class ConsumerService implements RocketMQListener<String> {

    @Override
    public void onMessage(String s) {
        System.out.println("消费者: 收到消息内容: "+s);
    }
}
```

运行测试:

```
( ( \ ) _ | ' _ | | | | _ V _ | | \ \ \ \
V \ _ | | | | | | | | | | ( | ) ) )
' _ | _ | | | | | | | _ / / / /
=====|_|=====/_|_/=/_/_/_/

:: Spring Boot ::      (v2.3.2.RELEASE)

2021-08-31 22:21:10.115 INFO 84288 --- [          main] c.javal234.RocketmqConsumerApplication : Starting RocketmqConsumerApplication on LAPTOP
2021-08-31 22:21:10.117 INFO 84288 --- [          main] c.javal234.RocketmqConsumerApplication : No active profile set, falling back to default
2021-08-31 22:21:10.581 INFO 84288 --- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8084 (http)
2021-08-31 22:21:10.586 INFO 84288 --- [          main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-08-31 22:21:10.586 INFO 84288 --- [          main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
2021-08-31 22:21:10.652 INFO 84288 --- [          main] o.a.c.c.C.[Tomcat].[localhost].[/]     : Initializing Spring embedded WebApplicationContext
2021-08-31 22:21:10.652 INFO 84288 --- [          main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2021-08-31 22:21:10.752 INFO 84288 --- [          main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExe
2021-08-31 22:21:13.077 INFO 84288 --- [          main] a.r.s.a.DefaultRocketMQListenerContainer : Running container: DefaultRocketMQListenerContai
2021-08-31 22:21:13.077 INFO 84288 --- [          main] o.a.r.s.a.ListenerContainerConfiguration : Register the listener to container, listenerBe
2021-08-31 22:21:13.093 INFO 84288 --- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8084 (http) with co
2021-08-31 22:21:13.099 INFO 84288 --- [          main] c.javal234.RocketmqConsumerApplication : Started RocketmqConsumerApplication in 3.193 s
消费者:收到消息内容:{"payload":"rocketmq过滤消息测试-TAG02","headers":{"id":"f182f40f-e315-9c53-dlaf-a5963973f4eb","timestamp":1630419719697}}
消费者:收到消息内容:{"payload":"rocketmq过滤消息测试-TAG01","headers":{"id":"2c0ceec1-ba0a-4c15-4385-el7fce316dde","timestamp":1630419719697}}
```

发现只消费了TAG1和TAG2的消息，TAG3消息没有被消费；

12.2 根据SQL表达式过滤消息

SQL表达式方式可以根据发送消息时输入的属性进行一些计算。

RocketMQ的SQL表达式语法 只定义了一些基本的语法功能。

数字比较, 如>, >=, <, <=, BETWEEN, =;

字符比较, 如: =, <>, IN;

IS NULL or IS NOT NULL;

逻辑运算符: AND, OR, NOT;

常量类型:

数值, 如: 123, 3.1415;

字符, 如: 'abc', 必须使用单引号;

NULL, 特殊常量

Boolean, TRUE or FALSE;

上实例,发送三个消息, 分别带上不同的header头信息;

```
/**
 * 发送SQL表达式头信息消息, 测试根据SQL表达式过滤消息
 */
public void sendMessageWithSQL(){
    // 构造消息1
    Message msg1 = MessageBuilder.withPayload("rocketmq过滤消息测试01").build();
    Map<String, Object> headers = new HashMap<>() ;
    headers.put("type", "pay") ;
    headers.put("a", 10) ;
    rocketMQTemplate.convertAndSend("java1234-filter-rocketmq", msg1, headers);

    // 构造消息2
    Message msg2 = MessageBuilder.withPayload("rocketmq过滤消息测试02").build();
    Map<String, Object> headers2 = new HashMap<>() ;
    headers2.put("type", "store") ;
    headers2.put("a", 4) ;
    rocketMQTemplate.convertAndSend("java1234-filter-rocketmq", msg2, headers2);

    // 构造消息3
    Message msg3 = MessageBuilder.withPayload("rocketmq过滤消息测试03").build();
    Map<String, Object> headers3 = new HashMap<>() ;
    headers3.put("type", "user") ;
    headers3.put("a", 7) ;
    rocketMQTemplate.convertAndSend("java1234-filter-rocketmq", msg3, headers3);
}
```

消费者端, selectorExpression = "type='user' or a <7", selectorType = SelectorType.SQL92, 指定 selectorType 以及设置表达式 selectorExpression

```
@RocketMQMessageListener(topic = "java1234-filter-rocketmq", consumerGroup
= "${rocketmq.consumer.group}" , selectorExpression = "type='user' or a
<7", selectorType = SelectorType.SQL92)
@Component
public class ConsumersService implements RocketMQListener<String> {

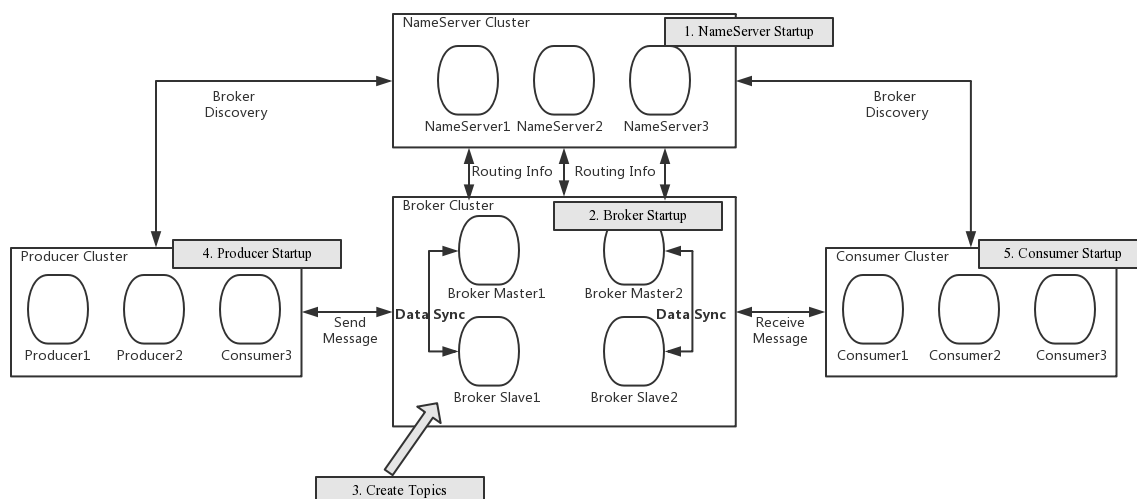
    @Override
    public void onMessage(String s) {
        System.out.println("消费者: 收到消息内容: "+s);
    }
}
```

默认不支持SQL表达式, 启动报错:

The broker does not support consumer to filter message by SQL92

找到 broker.conf 配置文件

13.1 RocketMQ集群介绍



- NameServer是一个几乎无状态节点，可集群部署，节点之间无任何信息同步。
- Broker部署相对复杂，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，Master与Slave的对应关系通过指定相同的BrokerName，不同的BrokerId来定义，BrokerId为0表示Master，非0表示Slave。Master也可以部署多个。每个Broker与NameServer集群中的所有节点建立长连接，定时注册Topic信息到所有NameServer。注意：当前RocketMQ版本在部署架构上支持一Master多Slave，但只有BrokerId=1的从服务器才会参与消息的读负载。
- Producer与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取Topic路由信息，并向提供Topic服务的Master建立长连接，且定时向Master发送心跳。Producer完全无状态，可集群部署。
- Consumer与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取Topic路由信息，并向提供Topic服务的Master、Slave建立长连接，且定时向Master、Slave发送心跳。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，消费者在向Master拉取消息时，Master服务器会根据拉取偏移量与最大偏移量的距离（判断是否读老消息，产生读I/O），以及从服务器是否可读等因素建议下一次是从Master还是Slave拉取。

13.2 RocketMQ集群模式介绍

单 master 模式

- 也就是只有一个 master 节点，称不上是集群，一旦这个 master 节点宕机，那么整个服务就不可用，适合个人学习使用。

多 master 模式

- 多个 master 节点组成集群，单个 master 节点宕机或者重启对应用没有影响。
- 优点：所有模式中性能最高
- 缺点：单个 master 节点宕机期间，未被消费的消息在节点恢复之前不可用，消息的实时性就受到影响。

- 注意：使用同步刷盘可以保证消息不丢失，同时 Topic 相对应的 queue 应该分布在集群中各个节点，而不是只在某个节点上，否则，该节点宕机会对订阅该 topic 的应用造成影响。

多 master 多 slave 异步复制模式

- 在多 master 模式的基础上，每个 master 节点都有至少一个对应的 slave。
- master 节点可读可写，但是 slave 只能读不能写，类似于 mysql 的主从模式。
- 优点：在 master 宕机时，消费者可以从 slave 读取消息，消息的实时性不会受影响，性能几乎和多 master 一样。
- 缺点：使用异步复制的方式有可能会有消息丢失的问题。

多 master 多 slave 同步双写模式

- 同多 master 多 slave 异步复制模式类似，区别在于 master 和 slave 之间的数据同步方式。
- 优点：同步双写的同步模式能保证数据不丢失。
- 缺点：发送单个消息 RT 会略长，性能相比异步复制低10%左右。
- 刷盘策略：同步刷盘和异步刷盘（指的是节点自身数据是同步还是异步存储）
- 同步方式：同步双写和异步复制（指的一组 master 和 slave 之间数据的同步）
- 注意：要保证数据可靠，需采用同步刷盘和同步双写的方式，但性能会较其他方式低。

13.3 RocketMQ双主双从集群模式工作流程介绍

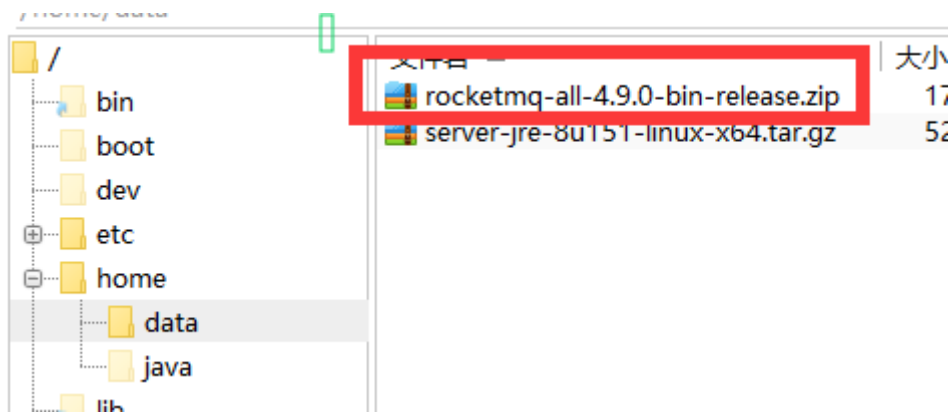
- 启动NameServer，NameServer起来后监听端口，等待Broker、Producer、Consumer连上来，相当于一个路由控制中心。
- Broker启动，跟所有的NameServer保持长连接，定时发送心跳包。心跳包中包含当前Broker信息(IP+端口等)以及存储所有Topic信息。注册成功后，NameServer集群中就有Topic跟Broker的映射关系。
- 收发消息前，先创建Topic，创建Topic时需要指定该Topic要存储在哪些Broker上，也可以在发送消息时自动创建Topic。
- Producer发送消息，启动时先跟NameServer集群中的其中一台建立长连接，并从NameServer中获取当前发送的Topic存在哪些Broker上，轮询从队列列表选择一个队列，然后与队列所在的Broker建立长连接从而向Broker发消息。
- Consumer跟Producer类似，跟其中一台NameServer建立长连接，获取当前订阅Topic存在哪些Broker上，然后直接跟Broker建立连接通道，开始消费消息。

13.4 RocketMQ集群搭建

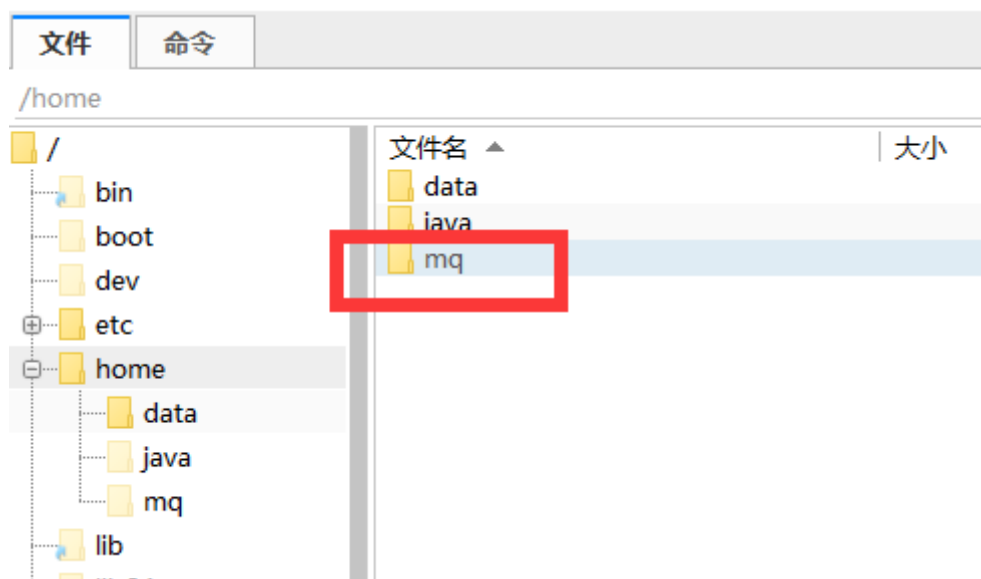
13.4.1 Centos单机搭建Rocketmq

先把 rocketmq 上传到 /hom/data/ 目录下；

为了方便，我们统一用 finalshell 工具上传；



/home/ 下面再新建一个 mq 目录用来存放 rocketmq 安装文件;



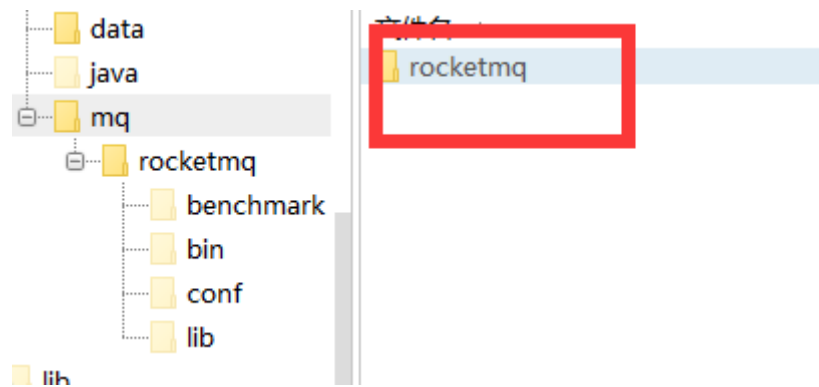
进入 data 目录，解压rocketmq压缩包到 mq 目录

```
unzip rocketmq-all-4.9.0-bin-release.zip -d ../mq
```

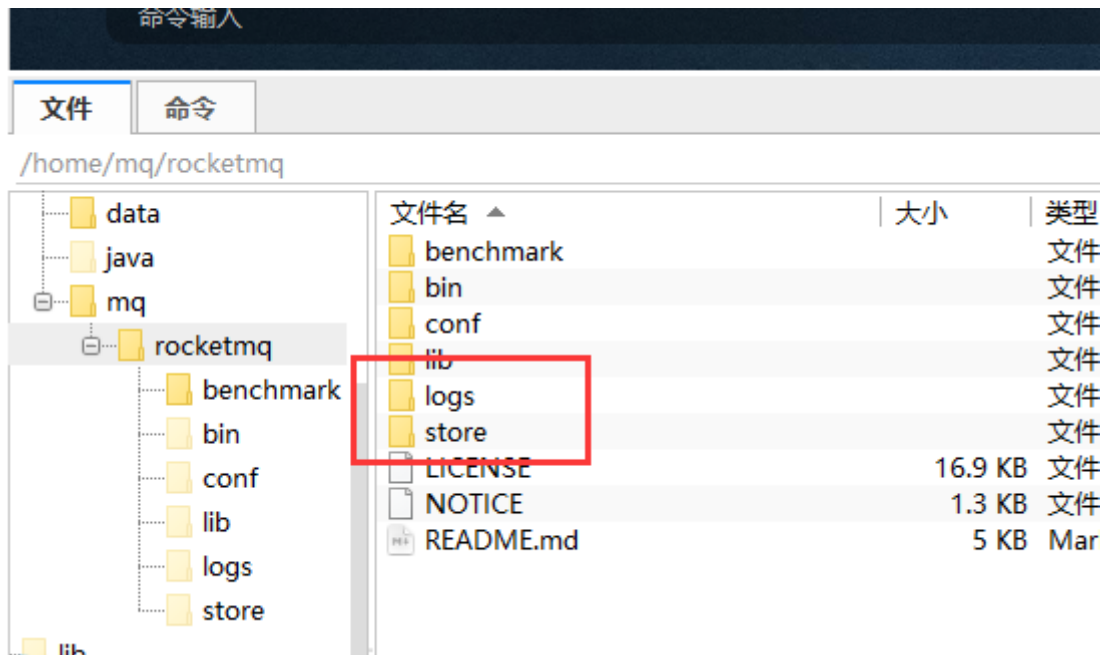
假如unzip没安装，可以安装下

```
yum install -y unzip zip
```

为了操作方便我们把解压后的文件名改成 rocketmq，工具里右击重命名即可;



rocketmq目录下，新建logs和store两个目录；

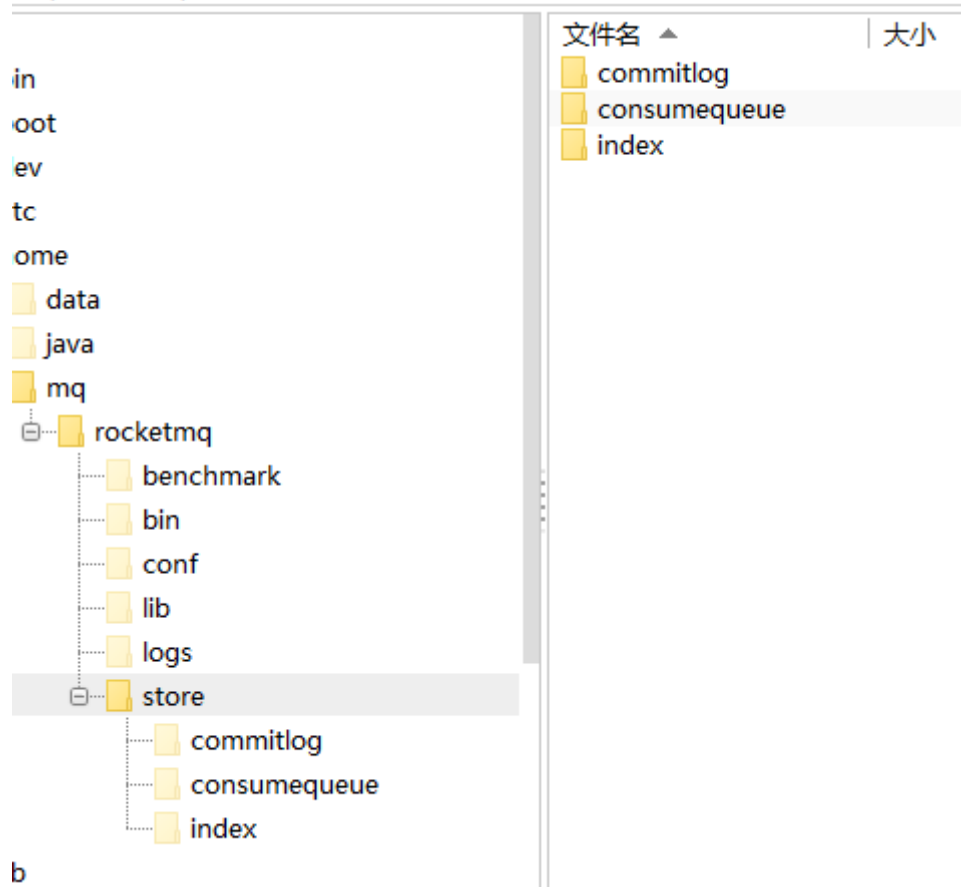


logs 是存储 rocketmq 日志的目录；

store 是存储 rocketmq 数据文件的目录；

在store目录再新建 commitlog，consumequeue，index 三个目录；

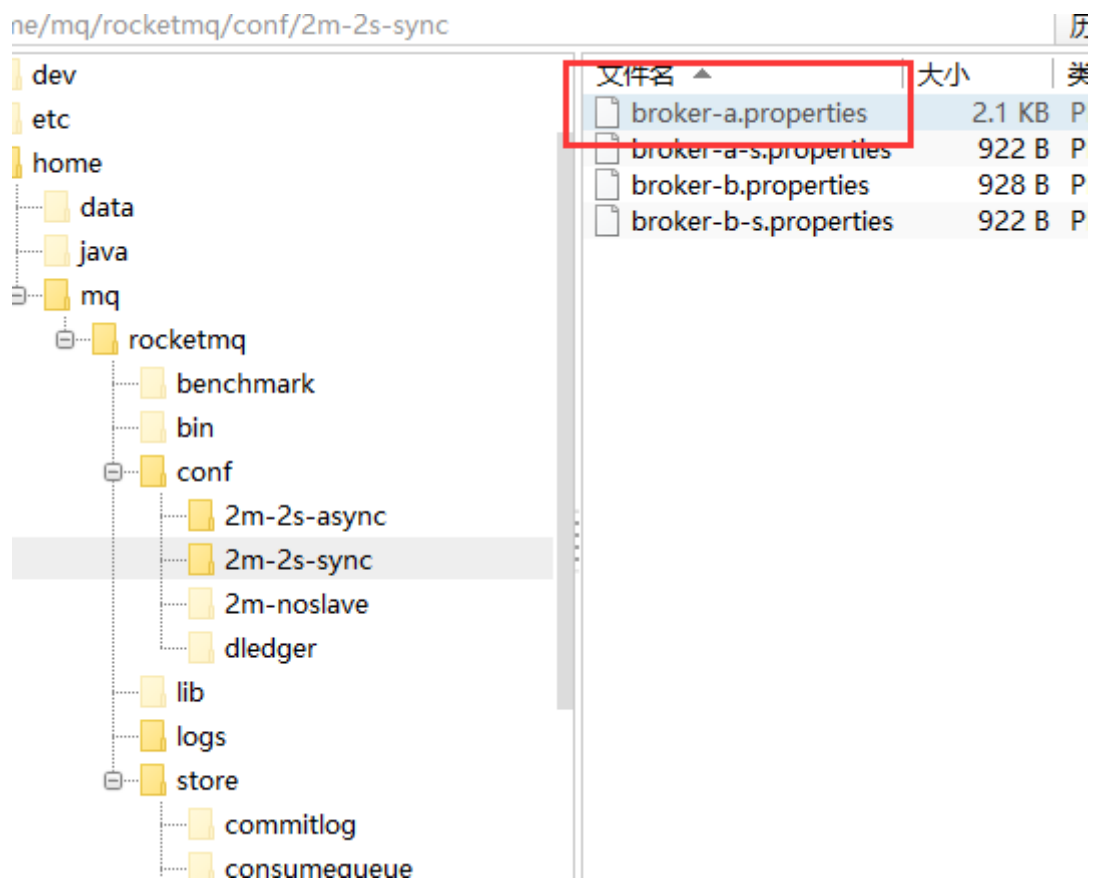
/mq/rocketmq/store



commitlog 存储RocketMQ消息信息目录;

consumequeue、index 存储索引文件数据目录;

我们先配置单节点, 修改 conf 下的 2m-2s-sync 配置文件;

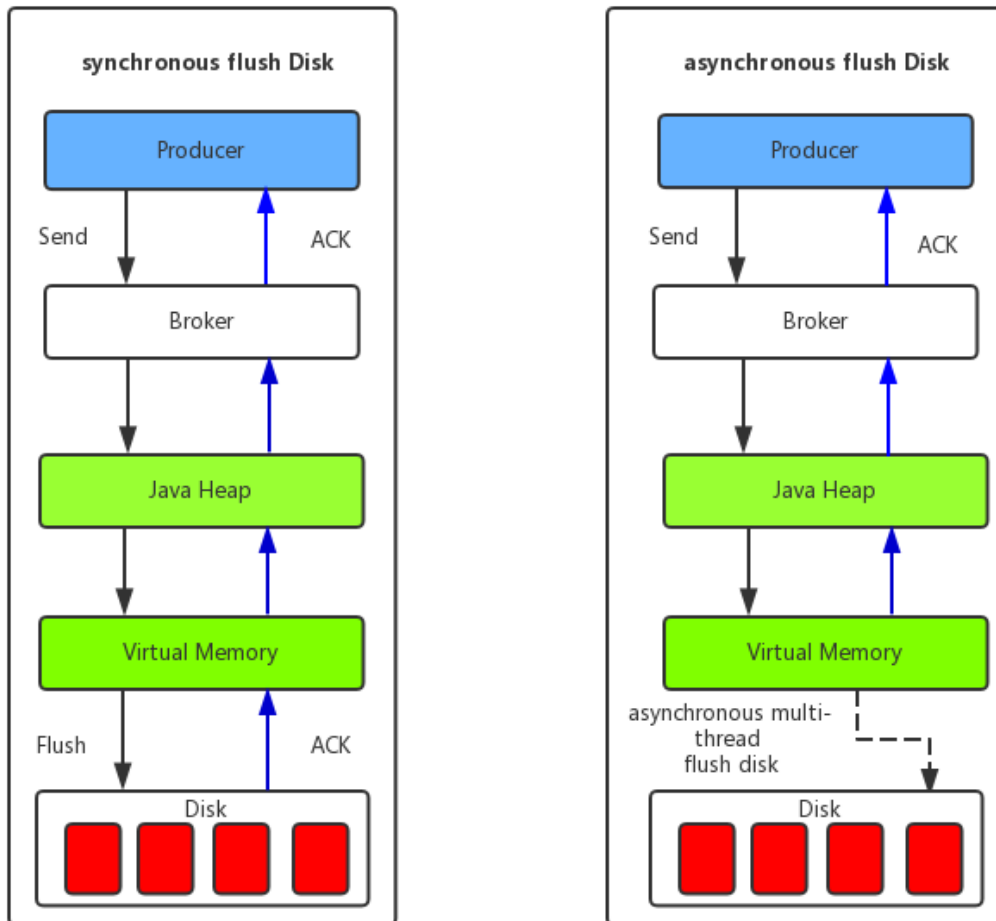


覆盖掉 broker-a.properties 配置文件内容:


```
brokerClusterName=rocketmq-cluster
#broker名字, 名字可重复,为了管理,每个master起一个名字,他的slave同他,eg:Amaster叫broker-a,
他的slave也叫broker-a
brokerName=broker-a
#0 表示 Master, >0 表示 slave
brokerId=0
#nameServer地址, 分号分割
namesrvAddr=192.168.0.110:9876
#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
mappedFileSizeCommitLog=1073741824
#ConsumeQueue每个文件默认存30w条, 根据业务情况调整
mappedFileSizeConsumeQueue=300000
#destroyMappedFileIntervalForcibly=120000
#redeleteHangedFileInterval=120000
#检测物理文件磁盘空间
diskMaxUsedSpaceRatio=88
#存储路径
storePathRootDir=/home/mq/rocketmq/store
#commitLog 存储路径
storePathCommitLog=/home/mq/rocketmq/store/commitlog
#消费队列存储路径存储路径
storePathConsumeQueue=/home/mq/rocketmq/store/consumequeue
#消息索引存储路径
storePathIndex=/home/mq/rocketmq/store/index
#checkpoint 文件存储路径
storeCheckpoint=/home/mq/rocketmq/store/checkpoint
#abort 文件存储路径
abortFile=/usr/local/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=SYNC_MASTER
#刷盘方式
#- ASYNC_FLUSH 异步刷盘
#- SYNC_FLUSH 同步刷盘
flushDiskType=SYNC_FLUSH
#checkTransactionMessageEnable=false
#发消息线程池数量
```

```
#sendMessageThreadPoolNums=128
#拉消息线程池数量
#pullMessageThreadPoolNums=128
```

这里有个 刷盘方式 具体讲下：



RocketMQ提供了两种刷盘策略同步刷盘、异步刷盘

同步刷盘：在消息到达MQ后，RocketMQ需要将数据持久化，同步刷盘是指数据到达内存之后，必须刷到commitlog日志之后才算成功，然后返回producer数据已经发送成功。

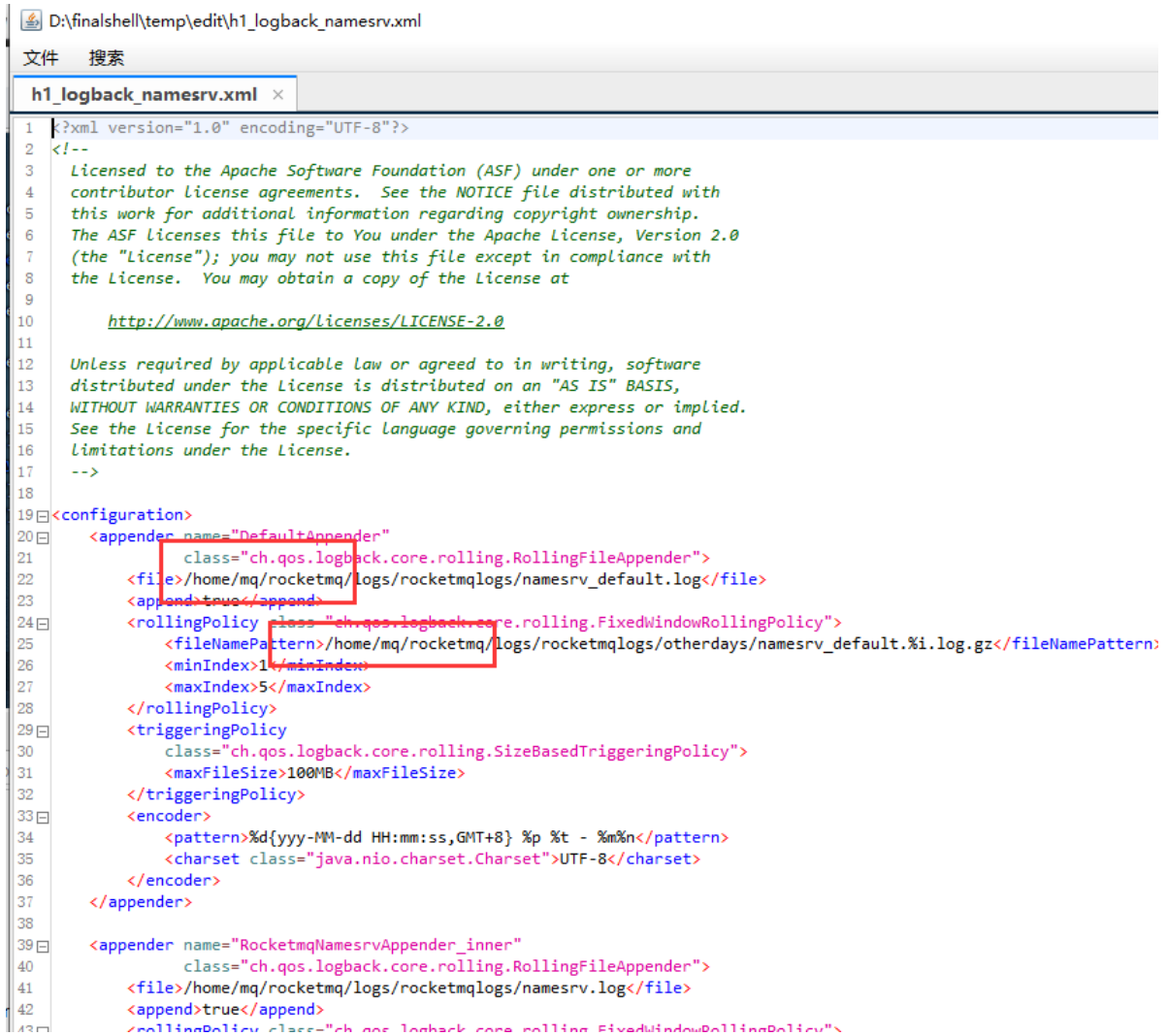
异步刷盘：，同步刷盘是指数据到达内存之后,返回producer说数据已经发送成功。，然后再写入commitlog日志。

复制方式	优点	缺点	适应场景
同步刷盘	保证了消息不丢失	吞吐率相对于异步刷盘要低	消息可靠性要求较高的场景
异步刷盘	系统的吞吐量提高	系统断电等异常时会有部分丢失	对应吞吐量要求较高的场景

进入conf目录下，替换掉所有xml中的\${user.home}，确保日志路径正确

执行:

```
sed -i 's${user.home}#/home/mq/rocketmq#g' *.xml
```



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Licensed to the Apache Software Foundation (ASF) under one or more
4 contributor license agreements. See the NOTICE file distributed with
5 this work for additional information regarding copyright ownership.
6 The ASF licenses this file to You under the Apache License, Version 2.0
7 (the "License"); you may not use this file except in compliance with
8 the License. You may obtain a copy of the License at
9
10 http://www.apache.org/licenses/LICENSE-2.0
11
12 Unless required by applicable law or agreed to in writing, software
13 distributed under the License is distributed on an "AS IS" BASIS,
14 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15 See the License for the specific language governing permissions and
16 limitations under the License.
17 -->
18
19 <configuration>
20   <appender name="DefaultAppender"
21     class="ch.qos.logback.core.rolling.RollingFileAppender">
22     <file>/home/mq/rocketmq/logs/rocketmqlogs/namesrv_default.log</file>
23     <append>true</append>
24     <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
25       <fileNamePattern>/home/mq/rocketmq/logs/rocketmqlogs/otherdays/namesrv_default.%i.log.gz</fileNamePattern>
26       <minIndex>1</minIndex>
27       <maxIndex>5</maxIndex>
28     </rollingPolicy>
29     <triggeringPolicy
30       class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
31       <maxFileSize>100MB</maxFileSize>
32     </triggeringPolicy>
33     <encoder>
34       <pattern>%d{yyy-MM-dd HH:mm:ss,GMT+8} %p %t - %m%n</pattern>
35       <charset class="java.nio.charset.Charset">UTF-8</charset>
36     </encoder>
37   </appender>
38
39   <appender name="RocketmqNamesrvAppender_inner"
40     class="ch.qos.logback.core.rolling.RollingFileAppender">
41     <file>/home/mq/rocketmq/logs/rocketmqlogs/namesrv.log</file>
42     <append>true</append>
43     <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
```

路径全部替换了;

Rocketmq启动对内存要求比较高, 一般至少1个G内存, 否则会影响RocketMQ的性能;

默认配置内存比较高, 虚拟机不方便演示, 所以我们修改下bin目录下的runbroker.sh和runserver.sh文件;

runbroker.sh

```
JAVA_OPTS="${JAVA_OPTS} -server -Xms8g -Xmx8g -Xmn4g"
```

改成

```
JAVA_OPTS="${JAVA_OPTS} -server -Xms1g -Xmx1g -Xmn1g"
```

runserver.sh

```
JAVA_OPT="${JAVA_OPT} -server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
```

改成

```
JAVA_OPT="${JAVA_OPT} -server -Xms1g -Xmx1g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
```

先启动nameserver（后台运行方式启动）：

```
nohup sh mqnamesrv &
```

再启动broker（后台运行方式启动）

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-a.properties &
```

可以用 `jps` 查看进程：

```
jps
83545 Jps
65852 NamesrvStartup
83310 BrokerStartup
```

备注：关闭命令

先关闭broker，再关闭nameserver

```
sh mqshutdown broker
sh mqshutdown namesrv
```

我们代码测试下：

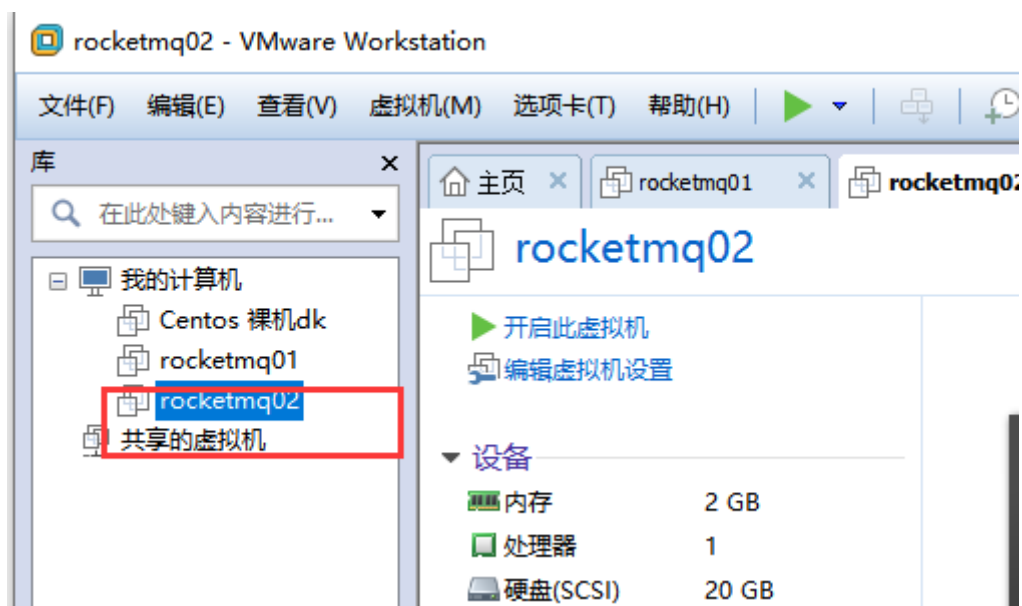
把消费端，生产端的rocketmq.name-server值改成：虚拟机IP地址:9876

```
server:
  port: 8086
servlet:
  context-path: /
rocketmq:
  name-server: 192.168.0.110:9876
  producer:
    group: producer-demol
```

启动测试OK;

13.4.2 Centos集群一主一从同步搭建Rocketmq

基于前面的单机模式，我们VM里面克隆一下系统；



192.168.0.110 机器 作为主节点

192.168.0.103 机器作为从节点

配置项要点：

- brokerClusterName集群名称一样；
- brokerName同一组主从节点名称一样；
- brokerId为0表示Master主节点，非0表示Slave从节点；

所以，从节点机器，我们修改 `conf` 下的 `2m-2s-sync` 配置文件 `broker-a-s.properties`

从原来的 `broker-a.properties` 复制一份内容到 `broker-a-s.properties`，然后修改三个地方：

```
#0 表示 Master, >0 表示 slave
brokerId=1
```

```
#Broker 的角色
#- ASYNC_MASTER 异步复制Master
#- SYNC_MASTER 同步双写Master
#- SLAVE
brokerRole=SLAVE
```

```
namesrvAddr=192.168.0.110:9876;192.168.0.103:9876;
```

我们先把两台机器的nameserver启动起来：

192.168.0.110 主机：

```
nohup sh mqnamesrv &
```

192.168.0.103 从机器

```
nohup sh mqnamesrv &
```

再把两台机器的broker启动起来：

192.168.0.110 主机：

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-a.properties &
```

192.168.0.103 从机器

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-a-s.properties &
```

可视化控制台项目namesrvAddr配置改下：

```
1 spring.application.name=rocketmq-console
2 spring.http.encoding.charset=UTF-8
3 spring.http.encoding.enabled=true
4 spring.http.encoding.force=true
5 logging.level.root=INFO
6 logging.config=classpath:logback.xml
7 #if this value is empty,use env value rocketmq.config.namesrvAddr NAMESRV_ADDR | now, y
8 rocketmq.config.namesrvAddr=192.168.0.110:9876;192.168.0.103:9876
9 #if you use rocketmq version < 3.5.8, rocketmq.config.isVIPChannel should be false.defau
10 rocketmq.config.isVIPChannel=
11 #rocketmq-console's data path:dashboard/monitor
12 rocketmq.config.dataPath=/tmp/rocketmq-console/data
13 #set it false if you don't want use dashboard.default true
14 rocketmq.config.enableDashBoardCollect=true
15 #set the message track trace topic if you don't want use the default one
```

启动控制台项目：

RocketMQ-Console

OPS

Dashboard

Cluster

Topic

Consumer

Producer

Message

MessageTrace

ChangeLanguage

Cluster :

rocketmq-cluster

Broker	NO.	Address	Version	Produce Massage TPS	Consumer Massage TPS	Yesterday Produce Count	Yesterday Consume Count	Today Produce Count	Today Consume Count	Operation
broker-a	0(master)	192.168.0.110:10911	V4_9_0	0.00	0.00	0	0	0	0	<div>STATUS</div> <div>CONFIG</div>
broker-a	1(slave)	192.168.0.103:10911	V4_9_0	0.00	0.00	0	0	0	0	<div>STATUS</div> <div>CONFIG</div>

项目代码，生产端和消费端name-server都改下；

```

rocketmq:
  name-server: 192.168.0.110:9876;192.168.0.103:9876
  consumer:

```

启动测试：

RocketMQ-Console

OPS

Dashboard

Cluster

Topic

Consumer

Producer

Message

MessageTrace

ChangeLanguage

Cluster :

rocketmq-cluster

Broker	NO.	Address	Version	Produce Massage TPS	Consumer Massage TPS	Yesterday Produce Count	Yesterday Consume Count	Today Produce Count	Today Consume Count	Operation
broker-a	0(master)	192.168.0.110:10911	V4_9_0	0.00	0.00	0	0	134	250	<div>STATUS</div> <div>CONFIG</div>
broker-a	1(slave)	192.168.0.103:10911	V4_9_0	0.00	0.00	0	0	134	0	<div>STATUS</div> <div>CONFIG</div>

我们发现，消费消息都是从主节点broker消费；

我们模拟下，让主节点broker挂掉；

```
sh mqshutdown broker
```

```

2021-09-04 23:37:20.130 INFO 164440 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-09-04 23:37:20.130 INFO 164440 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 522 ms
2021-09-04 23:37:22.350 INFO 164440 --- [main] o.s.c.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-09-04 23:37:22.465 INFO 164440 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8086 (http) with context path ''
2021-09-04 23:37:22.471 INFO 164440 --- [main] c.java1234.RocketmqProviderApplication : Started RocketmqProviderApplication in 3.154 seconds (JVM running for 3.818)
2021-09-04 23:37:22.481 ERROR 164440 --- [main] o.a.r.spring.core.RocketMQTemplate : syncSend failed, destination:java1234-rocketmq, message:GenericMessage [payload=byte[
Exception in thread "main" org.springframework.messaging.MessagingException: No route info of this topic: java1234-rocketmq
See http://rocketmq.apache.org/docs/faq/ for further details.; nested exception is org.apache.rocketmq.client.exception.MQClientException: No route info of this topic: java1234-rocketmq
See http://rocketmq.apache.org/docs/faq/ for further details.
at org.apache.rocketmq.spring.core.RocketMQTemplate.syncSend(RocketMQTemplate.java:563)
at org.apache.rocketmq.spring.core.RocketMQTemplate.syncSend(RocketMQTemplate.java:484)
at org.apache.rocketmq.spring.core.RocketMQTemplate.syncSend(RocketMQTemplate.java:472)
at org.apache.rocketmq.spring.core.RocketMQTemplate.doSend(RocketMQTemplate.java:886)
at org.apache.rocketmq.spring.core.RocketMQTemplate.doSend(RocketMQTemplate.java:57)
at org.springframework.messaging.core.AbstractMessageSendingTemplate.send(AbstractMessageSendingTemplate.java:109)
at org.springframework.messaging.core.AbstractMessageSendingTemplate.convertAndSend(AbstractMessageSendingTemplate.java:151)
at org.springframework.messaging.core.AbstractMessageSendingTemplate.convertAndSend(AbstractMessageSendingTemplate.java:129)
at org.springframework.messaging.core.AbstractMessageSendingTemplate.convertAndSend(AbstractMessageSendingTemplate.java:122)
at com.java1234.rocketmq.ProducerService.sendMessage(ProducerService.java:35)
at com.java1234.RocketmqProviderApplication.main(RocketmqProviderApplication.java:14)

```

启动项目生产端发送消息报错：

```
Exception in thread "main" org.springframework.messaging.MessagingException: No route info of this topic: java1234-rocketmq
```

消费端没有问题，可以继续订阅；

所有这种一主一从模式还是有问题；我们继续后面双主双从；

13.4.3 Centos集群双主双从同步搭建Rocketmq

我们企业级开发，一般采用的是双主双从同步，以及异步刷盘；

同步消息保证消息不丢失，异步刷盘提高吞吐量；

我们VM里再克隆两台机器；

192.168.0.110 机器 作为m1主节点

192.168.0.103 机器作为s1从节点

192.168.0.111 机器 作为m2主节点

192.168.0.112 机器作为s2从节点

首先，我们把namesrvAddr配置修改，每个 broker 都要注册到所有 nameserver ；

```
namesrvAddr=192.168.0.110:9876;192.168.0.103:9876;192.168.0.111:9876;192.168.0.112:9876;
```

刷盘机制都改成异步：

```
#- ASYNC_FLUSH 异步刷盘
#- SYNC_FLUSH 同步刷盘
flushDiskType=ASYNC_FLUSH
```

192.168.0.111 机器 作为m2主节点从m1主节点的 broker-a.properties 复制内容到 broker-b.properties 文件；

修改 broker-b.properties 配置文件；

修改内容如下：

```
brokerName=broker-b
```

192.168.0.112 机器 作为s2主节点从s1主节点的 broker-a-s.properties 复制内容到 broker-b-s.properties 文件；

修改 broker-b-s.properties 配置文件；

修改内容如下：

```
brokerName=broker-b
```

然后分别启动四个机器；

先启动nameserver：

```
nohup sh mqnamesrv &
```


再启动broker:

192.168.0.110 机器 作为m1主节点

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-a.properties &
```

192.168.0.103 机器作为s1从节点

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-a-s.properties &
```

192.168.0.111 机器 作为m2主节点

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-b.properties &
```

192.168.0.112 机器作为s2从节点

```
nohup sh mqbroker -c /home/mq/rocketmq/conf/2m-2s-sync/broker-b-s.properties &
```

可视化控制台项目namesrvAddr配置改下:

```
spring.messaging.properties class
logging.level.root=INFO
logging.config=classpath:logback.xml
#if this value is empty, use env value rocketmq.config.namesrvAddr NAMESRV_ADDR | now, you can set it in ops pa
rocketmq.config.namesrvAddr=192.168.0.110:9876;192.168.0.103:9876;192.168.0.111:9876;192.168.0.112:9876;
#if you use rocketmq version < 3.5.8, rocketmq.config.isVIPChannel should be false.default true
rocketmq.config.isVIPChannel=
#rocketmq-console's data path:dashboard/monitor
rocketmq.config.dataPath=/tmp/rocketmq-console/data
#set it false if you don't want use dashboard.default true
```

启动控制台项目:

RocketMQ-Console OPS Dashboard Cluster Topic Consumer Producer Message MessageTrace ChangeLanguage -										
Cluster: rocketmq-cluster										
Broker	NO.	Address	Version	Produce Message TPS	Consumer Message TPS	Yesterday Produce Count	Yesterday Consume Count	Today Produce Count	Today Consume Count	Operation
broker-b	0(master)	192.168.0.111:10911	V4_9_0	0.00	0.00	0	0	53	50	<button>STATUS</button> <button>CONFIG</button>
broker-b	1(slave)	192.168.0.112:10911	V4_9_0	0.00	0.00	0	0	53	0	<button>STATUS</button> <button>CONFIG</button>
broker-a	0(master)	192.168.0.110:10911	V4_9_0	0.00	0.00	0	0	61	50	<button>STATUS</button> <button>CONFIG</button>
broker-a	1(slave)	192.168.0.103:10911	V4_9_0	0.00	0.00	0	0	61	0	<button>STATUS</button> <button>CONFIG</button>

项目代码, 生产端和消费端name-server都改下;

```
rocketmq:
  name-server: 192.168.0.110:9876;192.168.0.103:9876;192.168.0.111:9876;192.168.0.112:9876;
  producer:
    group: producer-demol
```

启动测试:

RocketMQ-ConsoleOPSDashboardClusterTopicConsumerProducerMessageMessageTraceChangelanguage

Cluster : rocketmq-cluster

Broker	NO.	Address	Version	Produce Massage TPS	Consumer Massage TPS	Yesterday Produce Count	Yesterday Consume Count	Today Produce Count	Today Consume Count	Operation
broker-b	0(master)	192.168.0.111:10911	V4_9_0	0.00	0.00	0	0	53	50	<div>STATUS</div> <div>CONFIG</div>
broker-b	1(slave)	192.168.0.112:10911	V4_9_0	0.00	0.00	0	0	53	0	<div>STATUS</div> <div>CONFIG</div>
broker-a	0(master)	192.168.0.110:10911	V4_9_0	0.00	0.00	0	0	61	50	<div>STATUS</div> <div>CONFIG</div>
broker-a	1(slave)	192.168.0.103:10911	V4_9_0	0.00	0.00	0	0	61	0	<div>STATUS</div> <div>CONFIG</div>

两个主节点一起分担消息处理；

我们模拟下，让a主节点broker挂掉；

```
sh mqshutdown broker
```

RocketMQ-ConsoleOPSDashboardClusterTopicConsumerProducerMessageMessageTraceChangelanguage

Cluster : rocketmq-cluster

Broker	NO.	Address	Version	Produce Massage TPS	Consumer Massage TPS	Yesterday Produce Count	Yesterday Consume Count	Today Produce Count	Today Consume Count	Operation
broker-b	0(master)	192.168.0.111:10911	V4_9_0	10.39	10.00	0	0	157	150	<div>STATUS</div> <div>CONFIG</div>
broker-b	1(slave)	192.168.0.112:10911	V4_9_0	10.40	0.00	0	0	157	0	<div>STATUS</div> <div>CONFIG</div>
broker-a	1(slave)	192.168.0.103:10911	V4_9_0	0.00	0.00	0	0	61	0	<div>STATUS</div> <div>CONFIG</div>

再运行代码测试，发现b主节点承担了所有消息接收和处理；实现了高可用；