

RocketMQ实战（二）



张丰哲 (/u/cb569cce501b) 已关注

2017.04.15 16:40 字数 1957 阅读 3238 评论 4 喜欢 27

(/u/cb569cce501b)

在上一篇《RocketMQ实战（一）》(<http://www.jianshu.com/p/3afd610a8f7d>)中已经为大家初步介绍了下RocketMQ以及搭建了双Master环境，接下来继续为大家介绍！

Quick Start

写一个简单的生产者、消费者，带大家快速体验RocketMQ~

Maven配置：

```
<dependency>
  <groupId>com.alibaba.rocketmq</groupId>
  <artifactId>rocketmq-client</artifactId>
  <version>3.2.6</version>
</dependency>
```

pom.xml

生产者：

```
public class Producer {

    public static void main(String[] args) throws MQClientException, InterruptedException {

        DefaultMQProducer producer = new DefaultMQProducer("ProducerGroup");
        producer.setNamesrvAddr("192.168.99.121:9876;192.168.99.122:9876");
        producer.start();

        for (int i = 0; i < 100; i++) {
            try {
                Message msg = new Message("TopicTest", // topic
                    "TagA", // tag
                    ("Hello RocketMQ " + i).getBytes("UTF-8") // body
                );
                SendResult sendResult = producer.send(msg);
                System.out.println(sendResult);
            } catch (Exception e) {
                e.printStackTrace();
                Thread.sleep(1000);
            }
        }

        producer.shutdown();
    }
}
```

生产者代码

消费者：

^

+

🔖

🔗

<http://www.jianshu.com/p/790d6bc4a1c1>

1/9

```
public class Consumer {

    public static void main(String[] args) throws InterruptedException, MQClientException {
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("ConsumerGroup");
        consumer.setNamesrvAddr("192.168.99.121:9876;192.168.99.122:9876");
        consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
        consumer.subscribe("TopicTest", "*");

        consumer.registerMessageListener(new MessageListenerConcurrently() {

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

                for(MessageExt ext : msgs){
                    try {
                        System.out.println(new Date() + new String(ext.getBody(),"UTF-8"));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });

        consumer.start();
        System.out.println("Consumer Started.");
    }
}
```

消费者代码

无论生产者、消费者都必须给出GroupName，而且具有唯一性！

生产到哪个Topic的哪个Tag下，消费者也是从Topic的哪个Tag进行消费，可见这个Tag有点类似于JMS Selector机制，即实现消息的过滤。

生产者、消费者需要设置NameServer地址。

这里，采用的是Consumer Push的方式，即设置Listener机制回调，相当于开启了一个线程。以后为大家介绍Consumer Pull的方式。

我们看一下运行结果：

```
[sendStatus=SEND_OK, msgId=COA8637900002A9F00000000000000198, messageQueue=MessageQueue [topic=TopicTest, brokerName=broker-a, queueId=31, queueOffset=0]
[sendStatus=SEND_OK, msgId=COA8637A000002A9F00000000000000000, messageQueue=MessageQueue [topic=TopicTest, brokerName=broker-b, queueId=0, queueOffset=0]
[sendStatus=SEND_OK, msgId=COA8637A000002A9F00000000000000088, messageQueue=MessageQueue [topic=TopicTest, brokerName=broker-b, queueId=1, queueOffset=0]
[sendStatus=SEND_OK, msgId=COA8637A000002A9F00000000000000110, messageQueue=MessageQueue [topic=TopicTest, brokerName=broker-b, queueId=2, queueOffset=0]
[sendStatus=SEND_OK, msgId=COA8637A000002A9F00000000000000198, messageQueue=MessageQueue [topic=TopicTest, brokerName=broker-b, queueId=3, queueOffset=0]
[sendStatus=SEND_OK, msgId=COA86379000002A9F00000000000000220, messageQueue=MessageQueue [topic=TopicTest, brokerName=broker-a, queueId=0, queueOffset=1]
```

生产者运行结果

仔细看看生产者结果输出，你会发现，有的消息发往broker-a，有的在broker-b上，自动实现了消息的负载均衡！

```
Consumer Started.
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 4
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 36
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 39
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 47
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 55
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 63
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 71
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 79
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 87
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 95
Sun Apr 09 18:03:33 CST 2017Hello RocketMQ 5
```

消费者运行结果



这里消费消息是没有什么顺序的，以后我们在来谈消息的顺序性。

我们再来看一看管控台：

#Broker Name	#Broker Instance								
broker-a	#BID	#Addr	#Version	#InTPS	#OutTPS	#InTotalYest	#OutTotalYest	#InTotalToday	#OutTotalToday
	0	192.168.99.121:10911	V3_2_6	0.0	0.0	0	0	52	0
broker-b	#BID	#Addr	#Version	#InTPS	#OutTPS	#InTotalYest	#OutTotalYest	#InTotalToday	#OutTotalToday
	0	192.168.99.122:10911	V3_2_6	0.0	0.0	0	0	48	0

消息分布在2个broker上

#Cluster Name	#Broker Detail									
rocketmq-cluster	#Broker Name	#Broker Instance								
	broker-a	#BID	#Addr	#Version	#InTPS	#OutTPS	#InTotalYest	#OutTotalYest	#InTotalToday	#OutTotalToday
		0	192.168.99.121:10911	V3_2_6	0.0	0.0	0	0	52	52
	broker-b	#BID	#Addr	#Version	#InTPS	#OutTPS	#InTotalYest	#OutTotalYest	#InTotalToday	#OutTotalToday
		0	192.168.99.122:10911	V3_2_6	0.0	0.0	0	0	48	48

消费后

在多Master模式中，如果某个Master进程挂了，显然这台broker将不可用，上面的消息也将无法消费，要知道开源版本的RocketMQ是没有提供切换程序，来自自动恢复故障的，因此在实际开发中，我们一般提供一个监听程序，用于监控Master的状态。

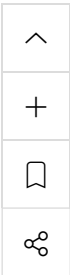
在ActiveMQ中，生产消息的时候会提供是否持久化的选择，但是对于RocketMQ而言，消息是一定会被持久化的！

上面的消费者采用的是Push Consumer的方式，那么监听的Listener中的消息List到底是多少条呢？虽然提供了API，如 consumer.setConsumeMessageBatchMaxSize(10)，实际上即使设置了批量的条数，但是注意了，是最大是10，并不意味着每次batch的都是10，只有在消息有挤压的情况下才有可能。而且Push Consumer的最佳实践方式就是一条条的消费，如果需要batch，可以使用Pull Consumer。

务必保证先启动消费者进行Topic订阅，然后在启动生产者进行生产（否则极有可能导致消息的重复消费，重复消费，重复消费！重要的事情说三遍！关于消息的重复问题后续给大家介绍~）。而且在实际开发中，有时候不会批量的处理消息，而是原子性的，单线程的去一条一条的处理消息，这样就是实时的在处理消息。（批量的处理海量的消息，可以考虑Kafka）

初步了解消息失败重试机制

消息失败，无非涉及到2端：从生产者端发往MQ的失败；消费者端从MQ消费消息的失败；



生产者端的失败重试

```
DefaultMQProducer producer = new DefaultMQProducer("ProducerGroup");
producer.setNamesrvAddr("192.168.99.121:9876;192.168.99.122:9876");
producer.setRetryTimesWhenSendFailed(3);
producer.start();

for (int i = 0; i < 100; i++) {
    try {
        Message msg = new Message("TopicTest", // topic
            "TagA", // tag
            ("Hello RocketMQ " + i).getBytes("UTF-8") // body
        );
        SendResult sendResult = producer.send(msg, 1000);
        System.out.println(sendResult);
    } catch (Exception e) {
        e.printStackTrace();
        Thread.sleep(1000);
    }
}

producer.shutdown();
```

生产者端失败重试

生产者端的消息失败：比如网络抖动导致生产者发送消息到MQ失败。

上图代码示例的处理手段是：如果该条消息在1S内没有发送成功，那么重试3次。

消费者端的失败重试

消费者端的失败，分为2种情况，一个是timeout，一个是exception

timeout，比如由于网络原因导致消息压根就没有从MQ到消费者上，在RocketMQ内部会不断的尝试发送这条消息，直至发送成功为止！（比如集群中一个broker失败，就尝试另一个broker）

exception，消息正常的到了消费者，结果消费者发生异常，处理失败了。这里涉及到一些问题，需要我们思考下，比如，消费者消费消息的状态有哪些定义？如果失败，MQ将采取什么策略进行重试？假设一次性批量PUSH了10条，其中某条数据消费异常，那么消息重试是10条呢，还是1条呢？而且在重试的过程中，需要保证不重复消费吗？

```
package com.alibaba.rocketmq.client.consumer.listener;

public enum ConsumeConcurrentlyStatus {
    CONSUME_SUCCESS,
    RECONSUME_LATER;

    private ConsumeConcurrentlyStatus() {
    }
}
```

ConsumeConcurrentlyStatus

消息消费的状态，有2种，一个是成功（CONSUME_SUCCESS），一个是失败&稍后重试（RECONSUME_LATER）

```
INFO main - brokerRole=ASYNC_MASTER
INFO main - flushDiskType=ASYNC_FLUSH
INFO main - syncFlushTimeout=5000
INFO main - messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
INFO main - flushDelayOffsetInterval=10000
INFO main - cleanFileForciblyEnable=true
INFO main - user specified name server address: rocketmq-nameserver-1:9876;rocketmq-nameserver-2:9876
```

broker启动日志

在启动broker的过程中，可以观察下日志，你会发现RECONSUME_LATER的策略。

如果消费失败，那么1S后再次消费，如果失败，那么5S后，再次消费，.....直至2H后如果消费还失败，那么该条消息就会终止发送给消费者了！

RocketMQ为我们提供了这么多次数的失败重试，但是在实际中也许我们并不需要这么多重试，比如重试3次，还没有成功，我们希望把这条消息存储起来并采用另一种方式处理，而且希望RocketMQ不要在重试呢，因为重试解决不了问题了！这该如何做呢？

我们先来看一下一条消息MessageExt对象的输出：

```
MessageExt [queueId=0, storeSize=137, queueOffset=0, sysFlag=0,
bornTimestamp=1492213846916, bornHost=/192.168.99.219:50478,
storeTimestamp=1492213846981, storeHost=/192.168.99.121:10911,
msgId=C0A8637900002A9F0000000000000000, commitLogOffset=0,
bodyCRC=613185359, reconsumeTimes=0, preparedTransactionOffset=0,
toString()=Message [topic=TopicTest2, flag=0, properties={TAGS=TagA,
WAIT=true, MAX_OFFSET=3, MIN_OFFSET=0}, body=16]]
```

注意到reconsumeTimes属性，这个属性就代表消息重试的次数！来看一段代码：

```
consumer.registerMessageListener(new MessageListenerConcurrently() {
    @Override
    public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
        ConsumeConcurrentlyContext context) {

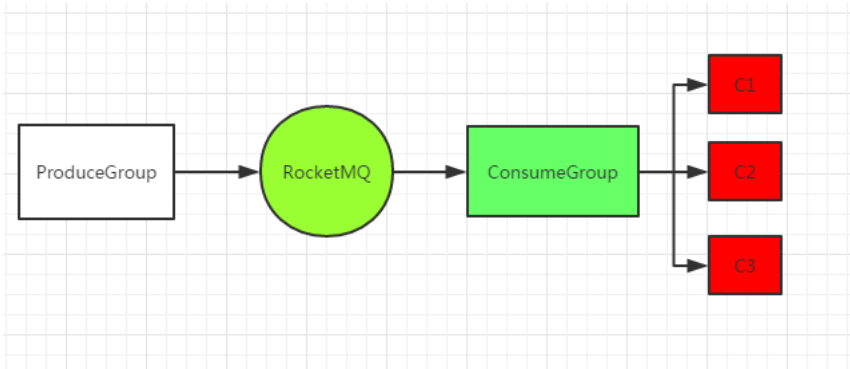
        MessageExt messageExt = msgs.get(0);
        System.out.println(messageExt);

        try{
            //业务处理 模拟消息消费异常
            //.....
            if(new Random().nextInt(3) == 2){
                throw new Exception();
            }
        }catch (Exception e){
            if(messageExt.getReconsumeTimes() == 3){
                //该条消息可以存储到DB或者LOG中 采用其他方式处理
                //.....
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
            return ConsumeConcurrentlyStatus.RECONSUME_LATER;
        }
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    }
});
```

reconsumeTime的使用

注意了，对于消费消息而言，存在2种指定的状态（成功 OR 失败重试），如果一条消息在消费端处理没有返回这2个状态，那么相当于这条消息没有达到消费者，势必会再次发送给消费者！也即是消息的处理必须有返回值，否则就进行重发。

天然的消息负载均衡及高效的水平扩展机制



消息的负载均衡

对于RocketMQ而言，通过ConsumeGroup的机制，实现了天然的消息负载均衡！通俗点来说，RocketMQ中的消息通过ConsumeGroup实现了将消息分发到C1/C2/C3/.....的机制，这意味着我们将非常方便的通过加机器来实现水平扩展！

我们考虑一下这种情况：比如C2发生了重启，一条消息发往C3进行消费，但是这条消息的处理需要0.1S，而此时C2刚好完成重启，那么C2是否可能会收到这条消息呢？答案是肯定的，也就是consume broker的重启，或者水平扩容，或者不遵守先订阅后生产消息，都可能导致消息的重复消费！关于去重的话题会在后续中予以介绍！

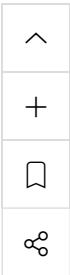
至于消息分发到C1/C2/C3，其实也是可以设置策略的。

```
AllocateMessageQueueStrategy (com.alibaba.rocketmq.client.co
AllocateMessageQueueAveragely (com.alibaba.rocketmq.clie
AllocateMessageQueueByMachineRoom (com.alibaba.rocketm
AllocateMessageQueueByConfig (com.alibaba.rocketmq.client
AllocateMessageQueueAveragelyByCircle (com.alibaba.rocket
```

消息负载策略

集群消费 AND 广播消费

RocketMQ的消费方式有2种，在默认情况下，就是集群消费，也就是上面提及的消息的负载均衡消费。另一种消费模式，是广播消费。广播消费，类似于ActiveMQ中的发布订阅模式，消息会发给Consume Group中的每一个消费者进行消费。



```
public enum MessageModel {  
    BROADCASTING,  
    CLUSTERING;  
  
    private MessageModel() {  
    }  
}
```

消费模式

```
//设置广播消费模式  
consumer.setMessageModel(MessageModel.BROADCASTING);
```

设置消费模式

OK，到这里，本期的RocketMQ就结束了，咱们下期见~

周末愉快！



张丰哲 (/u/cb569cce501b) ♂

写了 63893 字，被 2144 人关注，获得了 1674 个喜欢

(/u/cb569cce501b)

✓ 已关注

资深Java工程师 51CTO博客【2014-2016】：<http://zhangfengzhe.blog.51cto.com/>

好好学习，天天赞赏~


赞赏支持

喜欢 | 27



更多分享

(<http://cwb.assets.jianshu.io/notes/images/1114179>)




写下你的评论...

4条评论

只看作者

按喜欢排序 按时间正序 按时间倒序



小圆脸儿同学 (/u/ea39ca22ae5b)
2楼 · 2017.04.17 17:21
(/u/ea39ca22ae5b)
会不会有rmq的源码分析

赞 回复

张丰哲 (/u/cb569cce501b)：只会适当的涉及，比较重点分析的是RMQ的架构啦， 😊
2017.04.17 18:47 回复

添加新评论





小圆脸儿同学 (/u/ea39ca22ae5b)

3楼 · 2017.04.17 21:43

(/u/ea39ca22ae5b)

能否部署一套多M多S的环境。如果数据大量积压。多M多S的环境会不会影响生产速度。我现在碰见的问题就是大量数据积压。会影响生产速度。

👍 赞 💬 回复

张丰哲 (/u/cb569cce501b)：理论上来说，多MASTER模式是性能最好的，因为毕竟少了MASTER-SLAVE环节（异步复制、同步双写）。数据积压，应该考虑的是业务处理消息的时间吧，你可以先水平扩展下消费者，检查下时间花在哪里。RocketMQ即便积压了消息也不会影响生产速度的，写入是低延迟的。

2017.04.18 16:11 💬 回复

📝 添加新评论

被以下专题收入，发现更多相似内容

+ 收入我的专题



程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)



Java学习笔记 (/c/04cb7410c597?

utm_source=desktop&utm_medium=notes-included-collection)



高并发，分布式事务 (/c/6fd7fda0a677?

utm_source=desktop&utm_medium=notes-included-collection)



RocketMQ (/c/613c1ca3873c?utm_source=desktop&utm_medium=notes-included-collection)



MQ (/c/0c6b140b87f8?utm_source=desktop&utm_medium=notes-included-collection)



Java · ... (/c/0e8a2277d178?utm_source=desktop&utm_medium=notes-included-collection)



RocketMQ专辑 (/c/eafe0bc74f99?

utm_source=desktop&utm_medium=notes-included-collection)

展开更多 ▾

推荐阅读

更多精彩内容 > (/)

分布式利器Zookeeper（三） (/p/baf738d35614?utm...

(/p/baf738d35614?

前言 《分布式利器Zookeeper（一）》 《分布式利器Zookeeper（二）：分布式锁》 本篇博客是分布式利器Zookeeper系列的最后一篇，涉及的话题是：...

utm_campaign=maleskine&utm_content=note&utm...

张丰哲 (/u/cb569cce501b?

utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

纯手写实现JDK动态代理 (/p/58759fef38b8?utm_cam...

(/p/58759fef38b8?

前言 在Java领域，动态代理应用非常广泛，特别是流行的Spring/MyBatis等框架。JDK本身是有实现动态代理技术的，不过要求被代理的类必须实现接口，...

utm_campaign=maleskine&utm_content=note&utm...

张丰哲 (/u/cb569cce501b?

utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

一个人生活是种怎样的体验？ (/p/d223ae239332?utm...

(/p/d223ae239332?

有人说耐得住寂寞，才守得住繁华。有人说孤单是一个人的狂欢。一个人吃火

utm_campaign=maleskine&utm_content=note&utm...

锅，一个人看电影是一种怎样的体验？那就是点菜太多，老板怕你吃不完，你...

桃枝天妖 (/u/64c832a931b6?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

集体宿舍，将3000万大学生拉入深渊 (/p/b37597bf008... (/p/b37597bf0086?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)
1 毁掉一个人最好的两种方法，一是强迫他做毫无意义的重复性工作，另一种是让他过集体主义生活。 小明从高中开始就将这句话奉为真理，虽然他不知道...

智_先生 (/u/98a3b5fc6851?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

请你不要平平淡淡的将就好吗？ (/p/2288a2919541?ut... (/p/2288a2919541?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation)
说实话，我一点也不愿平平淡淡的就这么将就着过一辈子。 我不想才二十几岁就过着所谓平平淡淡的日子，我不想年纪轻轻就过着一眼望到底的生活。 我...

有备而来的路人甲 (/u/21a7a893f4b7?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

