# 源码研究RocketMQ

分析了NameServer实现原理、消息发送与高可用设计、消息存储、消息消费、消息拉取、消息队列负载、消息重试机制、定时消息、消息消费进度管理、消息文件、消息消费队列文件、消息Hash索引机制、消息过滤机制、过期文件删除、顺序消息实现原理、消息主从同步实现原理等。

<https://blog.csdn.net/prestigeding/column/info/20603>

RocketMQ原理解析-Index： https://www.iteye.com/blog/technoboy-2367786

RocketMQ原理解析-Setup：https://www.iteye.com/blog/technoboy-2368078

RocketMQ原理解析-Name Server：https://www.iteye.com/blog/technoboy-2368379

RocketMQ原理解析-Broker：https://www.iteye.com/blog/technoboy-2368391

RocketMQ原理解析-Producer：https://www.iteye.com/blog/technoboy-2353663

RocketMQ原理解析-Consumer：<https://www.iteye.com/blog/technoboy-2368553>

RocketMQ原理解析-HA：https://www.iteye.com/blog/technoboy-2368458

<https://www.jianshu.com/p/b9d4439599a8>

客户端配置详情：<https://blog.csdn.net/yousite1/article/details/79101469>

常见面试题

<https://mp.weixin.qq.com/s/D6mWrkr8Cy2XKMzImkxQCA>

<https://mp.weixin.qq.com/s/tLiEj_eofWe62d5WlVZe6w>

<https://github.com/apache/rocketmq/blob/master/docs/cn>

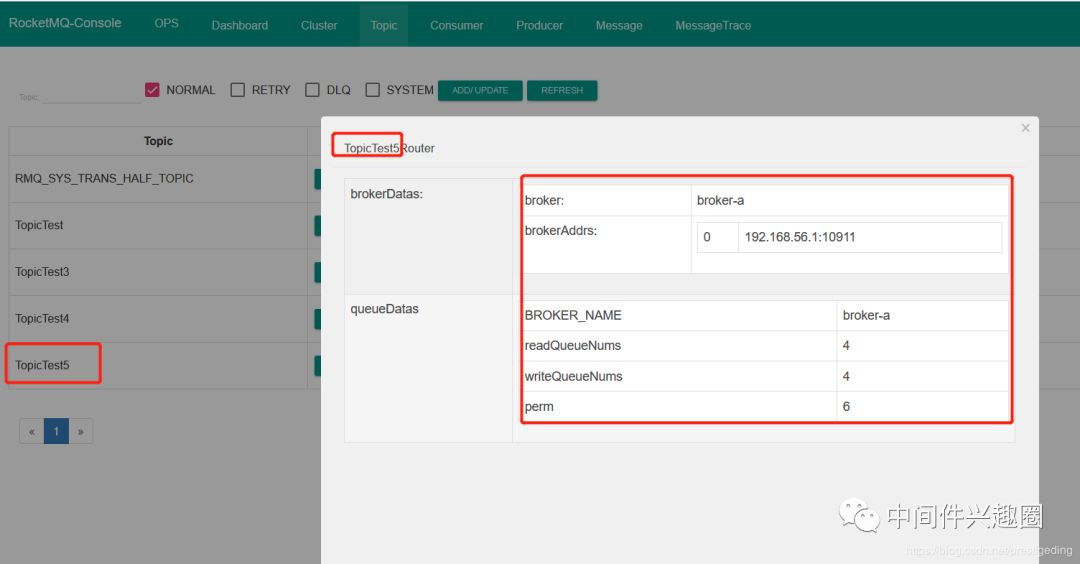
<https://www.sohu.com/a/129521820_487514>

# 不建议设置topic自动创建

## 问题

很多网友会问，为什么明明集群中有多台Broker服务器，autoCreateTopicEnable设置为true，表示开启Topic自动创建，但新创建的Topic的路由信息只包含在其中一台Broker服务器上，这是为什么呢？

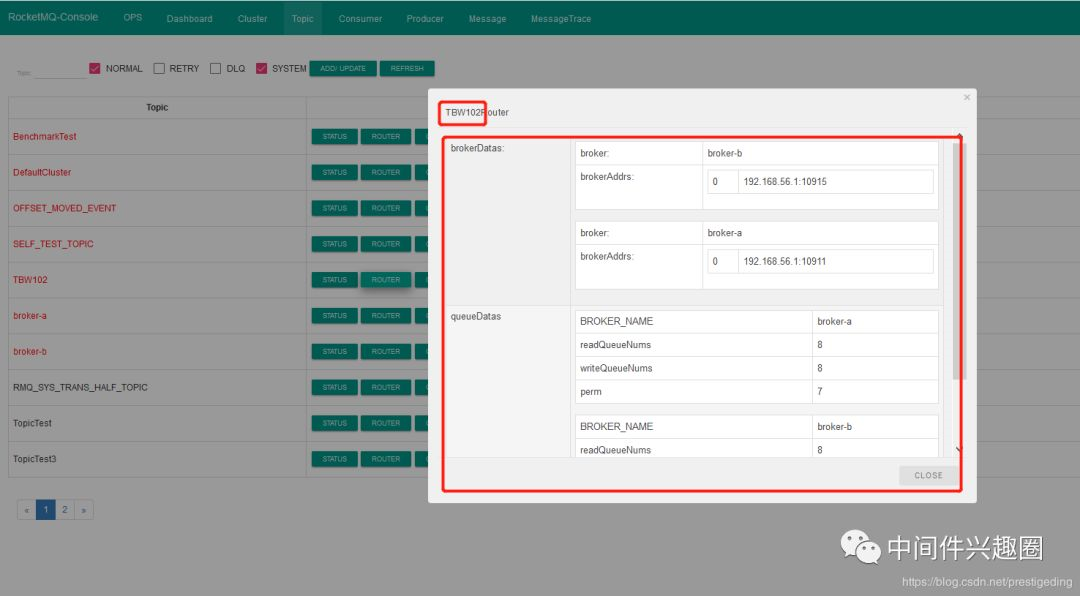
期望值：为了消息发送的高可用，希望新创建的Topic在集群中的每台Broker上创建对应的队列，避免Broker的单节点故障。



正如上图所示，自动创建的topicTest5的路由信息：

* topicTest5只在broker-a服务器上创建了队列，并没有在broker-b服务器创建队列，不符合期望。
* 默认读写队列的个数为4。

我们再来看一下RocketMQ默认topic的路由信息截图如下：



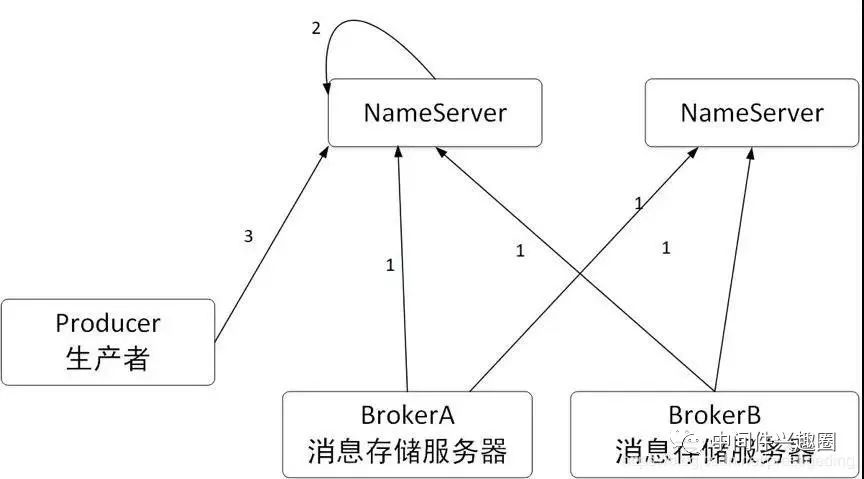
从图中可以默认Topic的路由信息为broker-a、broker-b上各8个队列。

## 思考

默认Topic的路由信息是如何创建的？

* Topic的路由信息是存储在哪里？Nameserver？broker?
* RocketMQ Topic默认队列个数是多少呢？

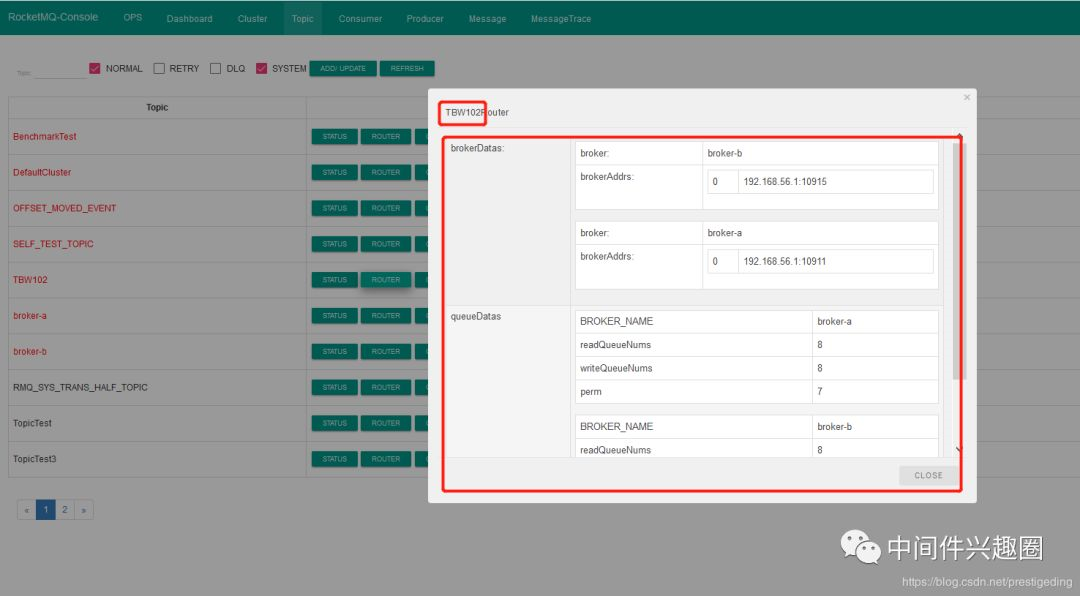
## 原理



* Broker在启动时向Nameserver注册存储在该服务器上的路由信息，并每隔30s向Nameserver发送心跳包，并更新路由信息。
* Nameserver每隔10s扫描路由表，如果检测到Broker服务宕机，则移除对应的路由信息。
* 消息生产者每隔30s会从Nameserver重新拉取Topic的路由信息并更新本地路由表；在消息发送之前，如果本地路由表中不存在对应主题的路由消息时，会主动向Nameserver拉取该主题的消息。

回到本文的主题：autoCreateTopicEnable，开启自动创建主题，试想一下，如果生产者向一个不存在的主题发送消息时，上面的任何一个步骤都无法获取到路由信息，那该如何处理这种情况呢？

在RocketMQ中，如果autoCreateTopicEnable设置为true，消息发送者向NameServer查询主题的路由消息返回空时，会尝试用一个系统默认的主题名称(MixAll.AUTO\_CREATE\_TOPIC\_KEY\_TOPIC)，此时消息发送者得到的路由信息为



默认Topic在集群的每一台Broker上创建8个队列，那问题来了，为啥新创建的Topic只在一个Broker上创建4个队列呢？

## autoCreateTopicEnable机制

### 默认Topic路由创建机制

温馨提示：本文不会详细跟踪整个创建过程，只会点出源码的关键入口点，如想详细了解NameServer路由消息、消息发送高可用的实现原理，建议查阅笔者的书籍《RocketMQ技术内幕》第二、三章。

Step1：在Broker启动流程中，会构建TopicConfigManager对象，其构造方法中首先会判断是否开启了允许自动创建主题，如果启用了自动创建主题，则向topicConfigTable中添加默认主题的路由信息。

该topicConfigTable中所有的路由信息，会随着Broker向Nameserver发送心跳包中，Nameserver收到这些信息后，更新对应Topic的路由信息表。

BrokerConfig的defaultTopicQueueNum默认为8。两台Broker服务器都会运行上面的过程，故最终Nameserver中关于默认主题的路由信息中，会包含两个Broker分别各8个队列信息。

Step2：生产者寻找路由信息

生产者首先向NameServer查询路由信息，由于是一个不存在的主题，故此时返回的路由信息为空，RocketMQ会使用默认的主题再次寻找，由于开启了自动创建路由信息，NameServer会向生产者返回默认主题的路由信息。然后从返回的路由信息中选择一个队列（默认轮询）。消息发送者从Nameserver获取到默认的Topic的队列信息后，队列的个数会改变吗？答案是会的

消息发送者在到默认路由信息时，其队列数量，会选择DefaultMQProducer#defaultTopicQueueNums与Nameserver返回的的队列数取最小值，DefaultMQProducer#defaultTopicQueueNums默认值为4，故自动创建的主题，其队列数量默认为4。

Step3：发送消息

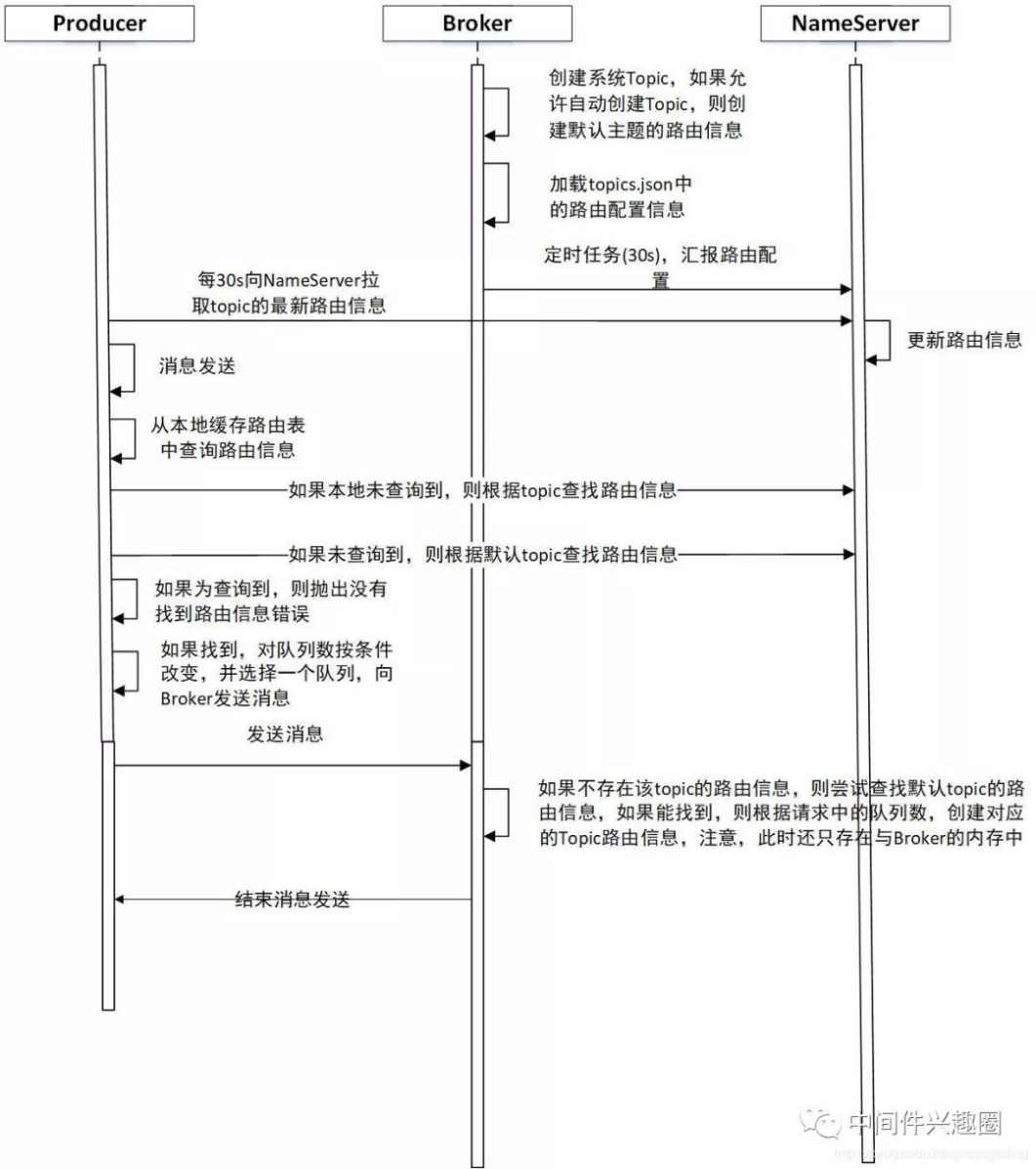
在消息发送时的请求报文中，设置默认topic名称，消息发送topic名称，使用的队列数量为DefaultMQProducer#defaultTopicQueueNums，即默认为4。

Step4：Broker端收到消息后的处理流程

服务端收到消息发送的处理器为：SendMessageProcessor，在处理消息发送时，会调用super.msgCheck方法：

在Broker端，首先会使用TopicConfigManager根据topic查询路由信息，如果Broker端不存在该主题的路由配置(路由信息),此时如果Broker中存在默认主题的路由配置信息，则根据消息发送请求中的队列数量，在Broker创建新Topic的路由信息。这样Broker服务端就会存在主题的路由信息。

在Broker端的topic配置管理器中存在的路由信息，一会向Nameserver发送心跳包，汇报到Nameserver，另一方面会有一个定时任务，定时存储在broker端，具体路径为${ROCKET\_HOME}/store/config/topics.json中，这样在Broker关闭后再重启，并不会丢失路由信息。



## 现象分析

经过上面自动创建路由机制的创建流程，我们可以比较容易的分析得出如下结论：

因为开启了自动创建路由信息，消息发送者根据Topic去NameServer无法得到路由信息，但接下来根据默认Topic从NameServer是能拿到路由信息(在每个Broker中，存在8个队列)，因为两个Broker在启动时都会向NameServer汇报路由信息。此时消息发送者缓存的路由信息是2个Broker，每个Broker默认4个队列（原因见3.2.1:Step2的分析）。

消息发送者然后按照轮询机制，发送第一条消息选择(broker-a的messageQueue:0)，向Broker发送消息，Broker服务器在处理消息时，首先会查看自己的路由配置管理器(TopicConfigManager)中的路由信息，此时不存在对应的路由信息，然后尝试查询是否存在默认Topic的路由信息，如果存在，说明启用了autoCreateTopicEnable，则在TopicConfigManager中创建新Topic的路由信息，此时存在与Broker服务端的内存中，然后本次消息发送结束。此时，在NameServer中还不存在新创建的Topic的路由信息。

这里有三个关键点：

启用autoCreateTopicEnable创建主题时，在Broker端创建主题的时机为，消息生产者往Broker端发送消息时才会创建。

然后Broker端会在一个心跳包周期内，将新创建的路由信息发送到NameServer，于此同时，Broker端还会有一个定时任务，定时将内存中的路由信息，持久化到Broker端的磁盘上。

消息发送者会每隔30s向NameServer更新路由信息，如果消息发送端一段时间内未发送消息，就不会有消息发送集群内的第二台Broker，那么NameServer中新创建的Topic的路由信息只会包含Broker-a，然后消息发送者会向NameServer拉取最新的路由信息，此时就会消息发送者原本缓存了2个broker的路由信息，将会变为一个Broker的路由信息，则该Topic的消息永远不会发送到另外一个Broker，就出现了上述现象。

原因就分析到这里了，现在我们还可以的大胆假设，开启autoCreateTopicEnable机制，什么情况会在两个Broker上都创建队列，其实，我们只需要连续快速的发送9条消息，就有可能在2个Broker上都创建队列，验证代码如下

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

DefaultMQProducer producer = new DefaultMQProducer("please\_rename\_unique\_group\_name");

producer.setNamesrvAddr("127.0.0.1:9876");

producer.start();

for (int i = 0; i < 9; i++) {

try {

Message msg = new Message("TopicTest10" ,"TagA" , ("Hello RocketMQ " + i).getBytes(RemotingHelper.DEFAULT\_CHARSET));

SendResult sendResult = producer.send(msg);

System.out.printf("%s%n", sendResult);

} catch (Exception e) {

e.printStackTrace();

Thread.sleep(1000);

}

}

producer.shutdown();

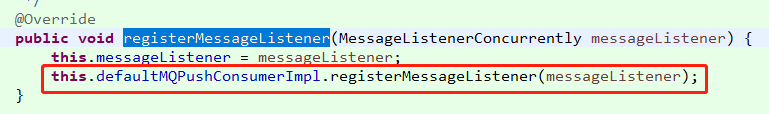
}

# Consumer启动过程

## 自定义程序启动mq消费者监听器

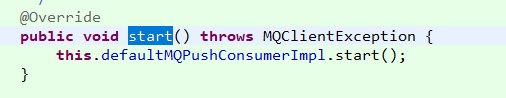


设置消费者参数后，我们将监听传给defaultMQPushConsumerImpl和DefaultMQPushConsumer两个类的实例



## 启动监听

当我们自定义的程序调用start()方法的时候，DefaultMQPushConsumer调用的是defaultMQPushConsumerImpl的start()方法



初始化消费者的信息。以及通过实现的MessageListenerConcurrently类型，创建消费处理业务类。一般我们不选择顺序执行的时候，会创建ConsumeMessageConcurrentlyService。下面是defaultMQPushConsumerImpl的start()。

再创建ConsumeMessageConcurrentlyService实例的时候，调用了ConsumeMessageConcurrentlyService的start()去初始化mq消息处理业务类。



## ConsumeMessageConcurrentlyService的实现

ConsumeMessageConcurrentlyService内部采用多线程的方式处理消息。

其中需要注意的是核心线程数与最大线程数，读取的是DefaultMQPushConsumer的数据。如果程序中，开启很多这样的mq消息的时候，需要注意这些。降低创建线程的数量，避免不必要的浪费。

Int consumeThreadMin = 20

int consumeThreadMax = 64;



除此之外，还会开启两个单例的线程。一个用于接受mq消息，一个用于处理过期的mq消息。

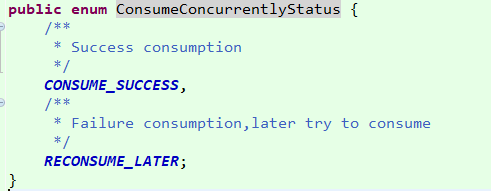
消费者在获的消息的时候（消费者通过开启一个线程向nameserver发送数据请求获取数据）

submitConsumeRequest用于提交一个消费者的拉去请求。

## 处理消息

submitConsumeRequest在获的消息的时候，调用ConsumeRequest的run()方法。

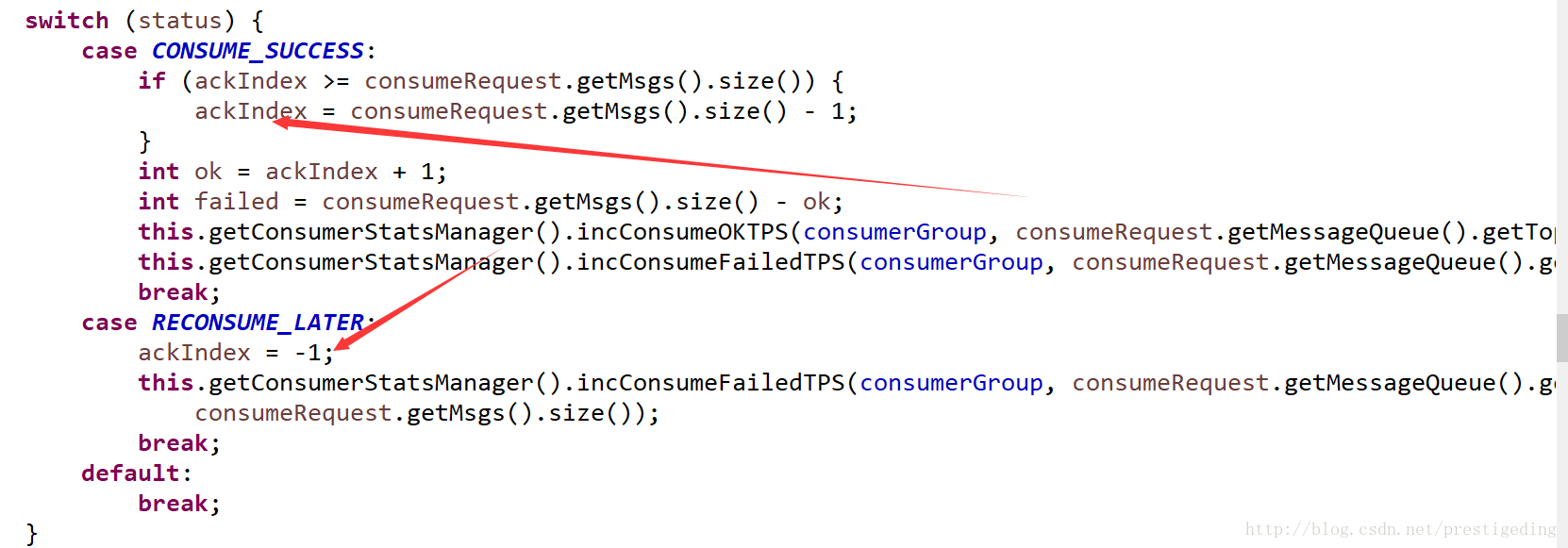
当处理消息之后，需要返回ConsumeConcurrentlyStatus类型的结果，一个是成功，另一个是稍后再试。



这两个是正常的返回值，如果消息返回null甚至抛出异常。程序会自动判断出来，并做出对用的结果返回值。



然后进入到结果处理：ConsumeMessageConcurrentlyService processConsumeResult



如果返回结果是 CONSUME\_SUCCESS，此时 ackIndex = msg.size() - 1,,再看发送sendMessageBack 循环的条件，for (int i = ackIndex + 1; i < msg.size() ;;)从这里可以看出如果消息成功，则无需发送sendMsgBack给broker

如果返回结果是RECONSUME\_LATER， 此时 ackIndex = -1 ，则这批所有的消息都会发送消息给Broker,也就是这一批消息都得重新消费。

如果发送ack消息失败，则会延迟5S后重新在消费端重新消费。

1）根据消费结果，设置ackIndex的值

2）如果是消费失败，根据消费模式（集群消费还是广播消费），广播模式，直接丢弃，集群模式发送sendMessageBack

3) 更新消息消费进度，不管消费成功与否，上述这些消息消费成功，其实就是修改消费偏移量。（失败的，会进行重试，会创建新的消息)

# 与容器整合

问题主要是Rocketmq的客户端发生

在例如k8s中，有一下两者场景：

* 每次上线的时候，容器其实都是重新拉取的镜像，所以项目启动后，环境都重新初始化了。
* K8s集群，每次上线我们的服务可能都在不同的k8s Node中启动，node的ip变化了，pod的ip也会变化

根据上面的两个场景

Mq cluster model，没有问题

Mq broadcast model，出现消息重复消费，偶尔出现broker消费者发现异常的问题等2个问题

1.消息重复消费，原因是容器中pod的ip会发生变化，而rocketmq broker 与 客户端连接的时候使用的是client=客户端ip + @ + 客户端进程id，ip是重要的校验指标。尤其是当ip发生变化的时候，这个pod会被认为是一个新的客户端实例，这样这个实例就会重新消费。

2.因为上线后，环境重新初始化了，原本保存在本地消费进度（consumeroffset.json）消失了。这个时候客户端可能会在之前的node中启动，这样broker认为这个服务还是原来的服务，因为ip没有发生变化，但是客户端本地的offset文件不见了，这个和broker校验的时候是校验不过的，到时broker心跳检测client的时候出现异常，就是两边的消费进度不匹配。

综上我们发现，无论是node的ip变化与否，都是有问题的。

解决方案：

1. 不使用Rocketmq broadcase model。使用redis的发布/订阅模式

2. 不使用Rocketmq broadcase model。采用dubbo 容错机制中的broadcastCluster模式，需要返回值的话，还可以自定义dubbo 的 merge result 采集调用结果

3. 使用rocketmq cluster model 的伪广播，每个节点都有不同的group，这个实现的时候需要注意group是启动后生成的，可以使用zk临时节点发方式去实现作为唯一性的分布式锁。（生产中有这样的问题，pod上线后，新的3个pod启动后，旧的pod还得运行一段时间才能暂停，所以还是有可能出现重复消费的问题）

4. 携程的方案是对rocketmq进行二次开发，可能是offset的存储方式获clientid的校验方式进行改造

5. pod启动脚本增加统一挂载的共享磁盘。这样offset文件不会丢失；然后，我们重复消费的问题，我们可以使用一个在消息中增加一个version，每次上线启动的时候，version+1，如果当前的version版本一直的话就消费。（这个方案也存在一定的不稳定性）

# 【好未来】介绍下Rocketmq？Rocketmq的事务消息的实现原理？知道哪些分布式事务的框架

<https://www.cnblogs.com/qdhxhz/p/11191399.html>

<https://blog.csdn.net/d729332647/article/details/79324767>

# 如何保证消息不丢失

大致的方案：<https://www.jianshu.com/p/3213d8c29fd0>

结合源码分析分案：<https://www.cnblogs.com/goodAndyxublog/p/12563813.html>

分别从Producer发送机制、Broker的持久化机制，以及消费者的offSet机制来最大程度保证消息不易丢失

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

或者再实时性和数据量不大的情况下，采用多master同步刷盘性能最好，但可用性较低。

实际中，mq作为公共资源，是没有办法轻易修改刷盘与同步方式的。所以对数据要求严格的场景，我们会在生产者端记录消息到数据库，之后去生产者端进行数据校验。

# RocketMQ Consumer如何获取并维护消费进度？

<https://blog.csdn.net/GAMEloft9/article/details/103999826>

# RocketMQ相关流程图/原理图

<https://www.javazhiyin.com/44018.html>

# 【玩吧】rocketmq延迟消息的原理？

<https://blog.csdn.net/qq924862077/article/details/84987179>

rocketmq 中有一个topic是 schedule\_message\_xxxx的消息队列，用来保存延迟的消息，这个topic是不能被订阅的。rocketmq 使用的是jdk的timer类实现的定时轮询，重队列里面根据每个延迟消息的offset（保存的位置）将消息拿出来，并还原到原来的消息队列中（默认队列尾），这样消费者就能看到消息了。

消息失败达到最大重试次数的消息会被发送到死信队列，销毁消息。

# 事务消息的原理？

<https://www.jianshu.com/p/7f517398d27e>

代码入口： TransactionalMessageCheckService。这里涉及到3个topic

RMQ\_SYS\_TRANS\_HALF\_TOPIC保存半消息，每60s取一次消息，判断是否有需要会查的消息。

RMQ\_SYS\_TRANS\_OP\_HALF\_TOPIC保存完成半消息的数据。要是判断需要校验的话，就像生产者发送消息进行会查，否则就将消息转移到真正的topic中。

# RocketMQ消费批拉超过32条不生效

<https://www.jianshu.com/p/4a2d8dfc71b6>

Pull数据的时候，最大拉取数据在broker限制为32了。这个是为了让broker上内存的合理使用设计的。我们可以通过设置配置文件配置最大限制。

# 如果一次拉取消息n条（n > 0），消费的时候，最后一条消费失败了，前n-1条是成功的。重试的时候是怎么消费的?

RocketMQ的消费过程：<https://www.jianshu.com/p/7126aaee5f55>

这n条都会重新消费。

# rocketmq 顺序消息实现原理?

rocketmq 发送消息，提供了两种消息的负载均衡的规则。随机和hash。 SelectMessageQueueByRandom 和 SelectMessageQueueByHash

我们一个可以自定义消息发送负载均衡算法，我们可以指定一个queue中。实现消息的FIFO。

但是需要注意的是，我们的消费方，应该使用单线程进行消费。否则可以出现多线程乱序消费，导致有序失败