**原创；微信公众号：千里行走；**

**受限图片大小限制，有些图片不是很清晰，可以到微信公众号查看；**

**强调：**

**a.本文基于rocketmq-4.3.2版本，不同版本是存在机制差异的，请阅读rocketmq源码。**

**b.本文基于的mq流程：client使用sync方式发送消息；broker落盘方式是async；broker-master和broker-slave之间的同步使用async。**

**目录**

**(1).总述**

1.异常概述

2.线上影响

3.原因概述

4.解决方案

**(2).rocketmq中broker的流控机制详解**

1.commitlog锁时间超过阈值的流控触发

2.queue中的task的等待时间超过阈值时的流控触发

3.注意

**(3).我们为何要对流控丢失的消息进行重试**

**(4).rocketmq-client的重试机制**

**(5).总结**

**正文**

**(1).总述**

**线上问题，原因，解决方案总述。**

**1.异常概述：**

producer发送消息报错broker busy.

**2.线上影响：**

很小，小于rpc同步调用下的失败/超时率：线上单节点每发送1600万+(共6个节点)，每个节点大约300条左右produce失败。

**3.原因概述：**

**严格讲这个不算mq的问题。**

rocketmq-broker默认情况下(brokerFastFailureEnable=true)，rocketmq集群本身对producer的message写请求有流控，这个流控机制在broker端，包含两层：

a.broker的msg request queue中的msgReq的等待时间超过阈值，触发流控。

b.broker向磁盘写入时的lock时间超过阈值，触发流控。

**4.解决方案：**

**方案一(我们选择的方案):**

**通过分析发生流控的毫秒数分布发现，加重试可以解决(blocking-queue，线程池异步重试即可)，queue-size设置200即可。**

**方案二：**

**修改rocketmq的配置，比如关闭流控(brokerFastFailureEnable=false)，或者调大相关的时间阈值(osPageCacheBusyTimeOutMills, waitTimeMillsInSendQueue, waitTimeMillsInPullQueue, waitTimeMillsInHeartbeatQueue, waitTimeMillsInTransactionQueue)。**

**没有选择这个方案的原因：mq/我们更关心的是处理速度，不希望隐藏/延迟业务潜在问题；而且默认配置足够支持很高的并发(我们线上峰值单broker是1.5~2W的TPS，其实也不高)。**

**其他优化措施：**

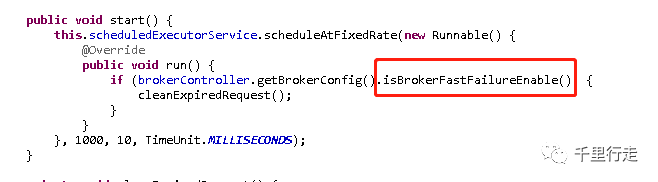
**a.后续计划在总成本边的前提下用ssd替换物理磁盘。**理由&依据如下：  
b.成本可以接受：线上使用的是1TB物理磁盘，price(1TB物理磁盘)=price(360GB SSD磁盘)。  
c.线上数据不需要这么大容量：  
进过线上运行的分析，不需要这么大的磁盘；目前单机只使用了不到10GB磁盘空间，一天至少1亿条消息，单体消息<100B,完全可以使用SSD。  
最初估计的1TB不必要。  
d.完善监控：需要自己写。

**解决后效果预计：**

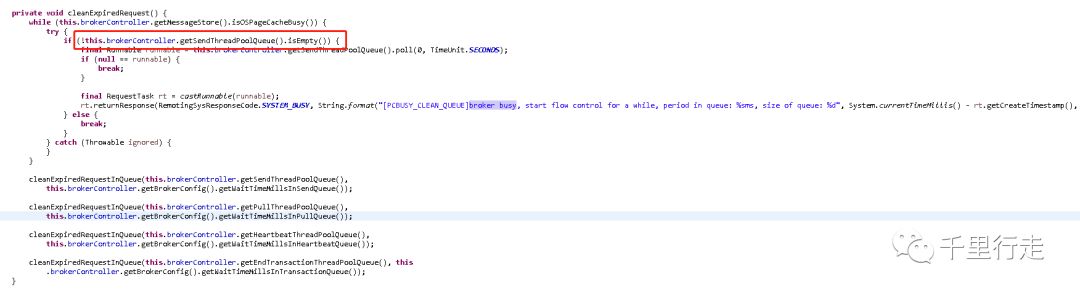
producer消息发送失败次数趋近于0(网络抖动等不可抗拒因素，概率很小)。   
注：  
完全做到0代价太大，而且没有必要；监控做好根据实际调控资源即可。

**(2).rocketmq中broker的流控机制详解**

**默认情况下，broker开启流控开关：brokerFastFailureEnable=true，broker每隔10毫秒会做一次流控处理，如下图：**

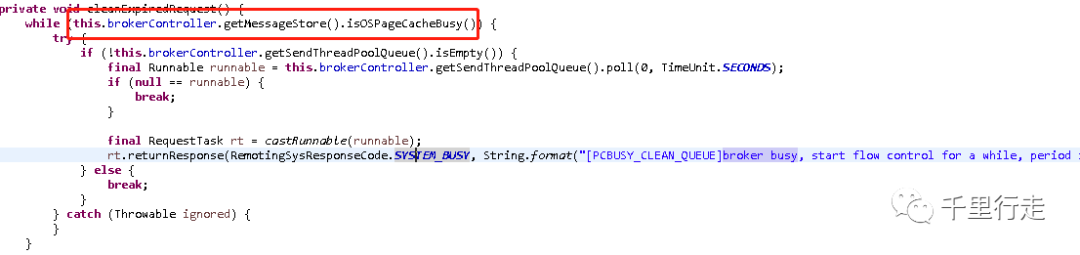


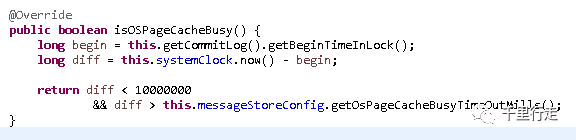
**流控主体方法，包含两步，commitlog锁时间超过阈值的流控触发，和queue中待处理任务的等待时间超过指定阈值时的流控触发。**



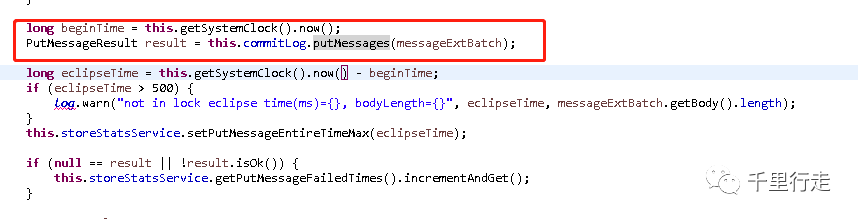
1.**commitlog锁时间超过阈值的流控触发**

**从下面两处代码可以看到判断逻辑：**



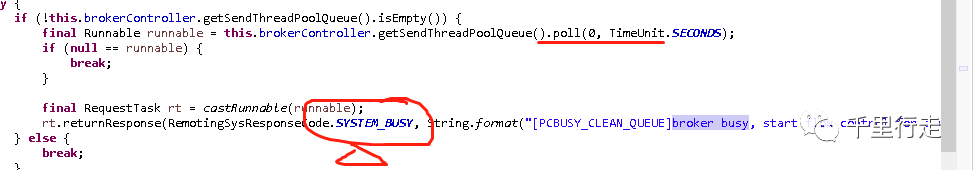


实际的putMessage(写入commitlog操作)，上图中isOSPageCacheBusy方法中的getCommitLog().getBeginTimeInLock()在下图中的标注方法中生成，通过这个时间lockTime(commitLog)是否超过既定阈值，从而决定是否触发流控。



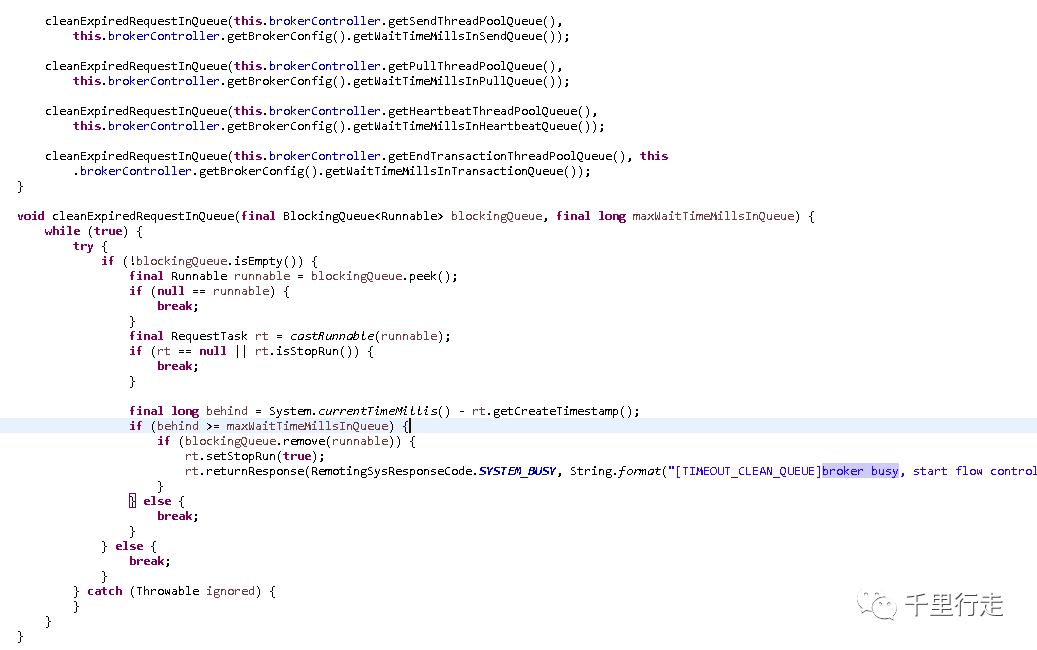


**当确认要进行流控时，处理很简单(通行操作)，从Queue中直接poll，然后设置response后直接返回，注意responseCode是SYSTEM\_BUSY，rocketmq-client发现是system\_busy，不会重试/重发消息，因为此时broker已经超出负载，rocketmq认为应该直接丢弃(此处没有问题，流控的处理模式)。**



2.**queue中的task的等待时间超过阈值时的流控触发**

**如下图，也是类似的处理，有些许不同，先peak，然后stop task，再remove，最后同样直接返回response，responseCode=SYSTEM\_BUSY。**



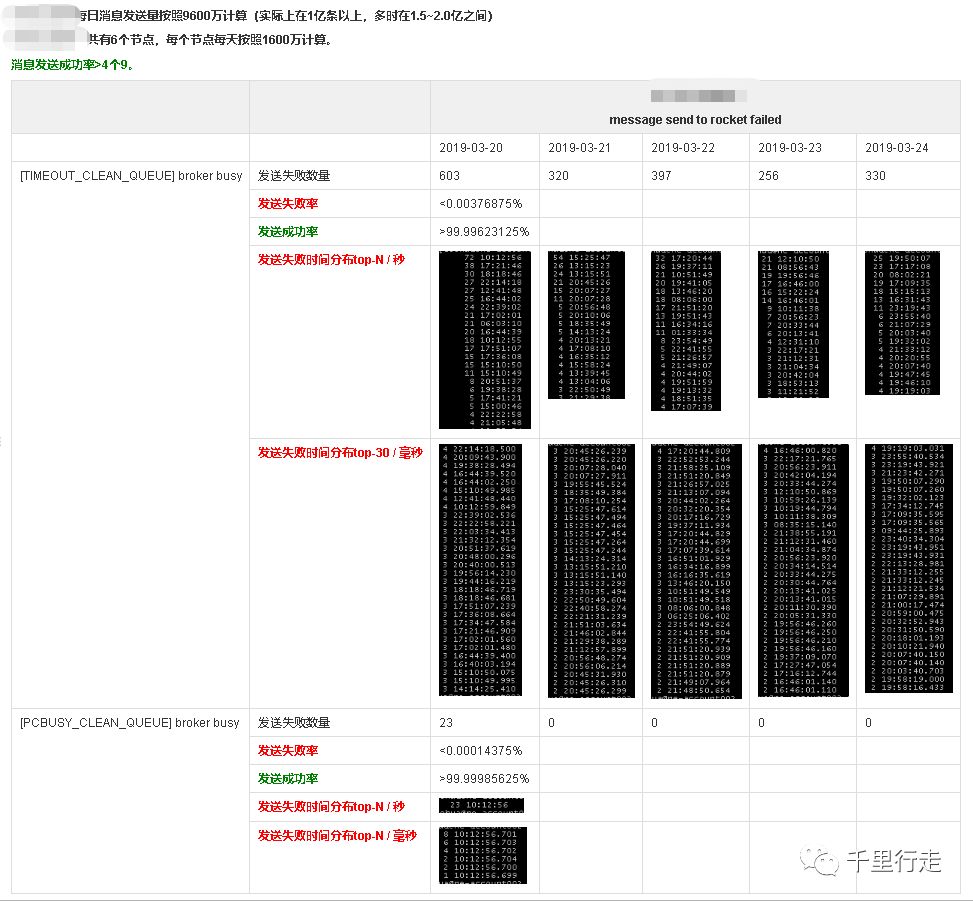
**3.注意**

**一旦触发了rocketmq-broker的流控，被remove掉的message直接丢失，这是流控的语义。**

**(3).我们为何要对流控丢失的消息进行重试**

也是通过数据分析，这样做性价比最高。

通过分析峰值时的broker busy的数量与时间分布，数据依据，如下图



**重试code，可以看到，有多个prometheus统计指标，我们对send的整个流程的各个节点做了监控：**



**(4).rocketmq-client的重试机制**

**如下图：**

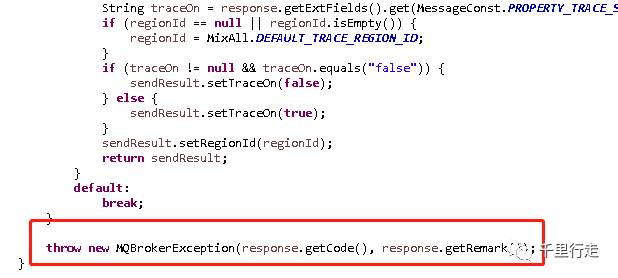
**1.可以看到，client对各种异常进行了处理，发生异常时进行重试。**

**2.但是，并不是对所有异常都会进行重试，当捕获MQBrokerException时，会判断responseCode的值，当responseCode为一些特定值时，比如前边提到的SYSTEM\_BUSY，还有topicNotExist等，是不会进行重试的，这是显然的。**



再往下看代码，可以发现，client对broker返回的response做了处理，会包装成**MQBrokerException，这也是为什么会出现上图中的逻辑，见下图（代码长，截了两个图）：**





**(5).总结**

**多看源码，并且一定要多总结。**

**写本文重新梳理时，发现很多细节记不清了，并且发生了对代码的怀疑。一定要对总结落地。**

