

GSoC结题报告

汇总

问题描述

问题一：Pop 模式的长轮询不会因 V1 版本的重试消息而被唤醒，导致显著的消费延迟

问题二：V1 重试主题的命名冲突可能导致跨主题消费和消息混淆

V2版本的代码会有很多无效唤醒

解决方案

最终解决方案

解决方案二

解决方案三

解决方案的现象

实验结果

现象一：改动前：使用remoting协议pushconsumer，一次性发送32条

现象二：改动前使用remoting协议pushconsumer，但持续发送消息

现象三：改动前使用grpc协议pushconsumer，一次性发送32条

现象四：改动后使用grpc协议pushconsumer，持续发送消息

现象五：改动后使用grpc协议simpleconsumer，一次性发送32条

其他问题探索

思考使用attemptId减少pop orderly阻塞的问题

ack不可靠可能导致pop orderly阻塞

汇总

<https://github.com/apache/rocketmq/issues/9632>

<https://github.com/apache/rocketmq/issues/9633>

<https://github.com/apache/rocketmq/pulls>

问题描述

ConsumerGroup	Topic	RetryTopic
MyConsumerGroup	Order_Topic	%RETRY%MyConsumerGroup_Order_Topic
MyConsumerGroup_Order	Topic	%RETRY%MyConsumerGroup_Order_Topic

问题一：Pop 模式的长轮询不会因 V1 版本的重试消息而被唤醒，导致显著的消费延迟

在 Pop 消费模式下，当一条消息在其“不可见时间”（invisible time）内没有被消费者确认（ACK）时，它会被转移到一个重试主题（retry topic）中，以便后续重新消费。按照设计，系统应该唤醒正在等待原始主题消息的长轮询请求，从而使这条消息能够被迅速地再次消费。

目前，这个唤醒机制对于 V2 版本的重试主题（格式为 `%RETRY%group+topic`）是正常工作的，因为系统可以从其中可靠地解析出原始的主题和消费组。

然而，对于 V1 版本的重试主题（格式为 `%RETRY%group_topic`），Broker（消息服务器）无法从这个重试主题名称中解析出原始的主题名。结果导致 `notifyMessageArrivingWithRetryTopic` 这个方法无法识别并唤醒正确的长轮询请求。这种情况迫使消费者的长轮询请求只能一直等待，直到超时（由 `BROKER_SUSPEND_MAX_TIME_MILLIS` 参数控制，通常为 15 秒），从而给消息的重试过程带来了严重的延迟。

问题二：V1 重试主题的命名冲突可能导致跨主题消费和消息混淆

在 V1 的重试主题命名规则（`%RETRY%group_topic`）下，不同的（主题，消费组）组合可能会映射到同一个重试主题名称。当这种情况发生时，从一个主题/消费组重试的消息，可能会被另一个共享了相同 V1 重试主题名称的主题/消费组所消费，从而引发跨主题消费和潜在的数据泄露问题。

根本原因：V1 的重试主题命名方式（`%RETRY%group_topic`）会丢失分隔信息，并且不是一对一的映射关系。不同的组合可能会产生命名冲突，例如：

- 组合一：（主题： `Order_Topic` ，消费组： `MyConsumerGroup` ）
- 组合二：（主题： `Topic` ，消费组： `MyConsumerGroup_Order` ）

这两个组合都会生成相同的 V1 重试主题名。

由于 Broker 仅仅根据重试主题的名称来路由重试消息，因此这种命名冲突会导致来自不同源头的消息流被合并到了一起。

V2版本的代码会有很多无效唤醒

之前retrytopic进来消息后，会唤醒原始topic下面的所有consumergroup的连接，其实只需要唤醒一个，因此会造成很多空拉，资源浪费。

解决方案

最终解决方案

```
1 public void notifyMessageArrivingWithRetryTopic(final String topic, final
   int queueId, long offset,
2                                     Long tagsCode, long msgStoreTime, byte[] filterBitMap, Map<String, String> properties) {
3     String prefix = MixAll.RETRY_GROUP_TOPIC_PREFIX;
4     if (topic.startsWith(prefix)) {
5         // 从properties获取原始topic名称
6         String originTopic = properties.get(MessageConst.PROPERTY_ORIGIN_TOPIC);
7         //根据原始topic和retryTopic, 最后获得retryTopic对应的cid (可能还可以与topicCidMap验证一下)
8         String suffix = "_" + originTopic; //这里把下划线换成加号也是一样的
9         String cid = topic.substring(prefix.length(), topic.length() - suffix.length());
10        POP_LOGGER.info("Processing retry topic: {}, originTopic: {}, properties: {}",
11                        topic, originTopic, properties); //grep "Processing retry topic" ~/logs/rocketmqlogs/pop.log可以看到日志
12        POP_LOGGER.info("Extracted cid: {} from retry topic: {}", cid, topic);
13        //然后调用包含cid的notifyMessageArriving
14        long interval = brokerController.getBrokerConfig().getPopLongPollingForceNotifyInterval();
15        boolean force = interval > 0L && offset % interval == 0L;
16        if (queueId >= 0) {
17            notifyMessageArriving(originTopic, -1, cid, force, tagsCode, msgStoreTime, filterBitMap, properties);
18        }
19        notifyMessageArriving(originTopic, queueId, cid, force, tagsCode, msgStoreTime, filterBitMap, properties);
20    } else {
21        //普通消息（非重试消息）还是走之前的逻辑不变
22        notifyMessageArriving(topic, queueId, offset, tagsCode, msgStoreTime, filterBitMap, properties);
23    }
24 }
```

1. 在common/message/MessageConst.java中增加一个新的字段“PROPERTY_ORIGIN_TOPIC”用于存储重试队列的消息的原始topic名称。
2. 在pop ck处理过程中，从ck中提取吃醋原始topic名称，加到 PROPERTY_ORIGIN_TOPIC 字段里。
3. 在notifyMessageArrivingWithRetryTopic中，根据提取出的原始topic名称、进而解析出cid，可以定向唤醒某个consumergroup。
4. 解决了之前V1版本retrytopic无法正确唤醒，V2版本有很多无效空唤醒的问题（根据topicCidMap唤醒某个topic下面所有cid）。

解决方案二

1. 在notifyMessageArrivingWithRetryTopic中检查是否启动了popKV
2. 获取popKV的记录中的cid字段
3. 使用KeyBuilder.parseNormalTopic(topic, cid)这个方法返回原是topic名称
4. 还是调用notifyMessageArriving

解决方案三

```

1  public void notifyMessageArrivingWithRetryTopic(final String topic, final
    int queueId, long offset,
2      Long tagsCode, long msgStoreTime, byte[] filterBitMap, Map<String, String> properties) {
3      String notifyTopic;
4      if (KeyBuilder.isPopRetryTopicV2(topic)) {
5          notifyTopic = KeyBuilder.parseNormalTopic(topic);
6      } else {
7          notifyTopic = findTopicForV1RetryTopic(topic);
8      }
9      notifyMessageArriving(notifyTopic, queueId, offset, tagsCode, msgStore
    Time, filterBitMap, properties);
10 }
11
12 /**
13     * Find the correct topic name for V1 retry topic by checking topicCid
    Map
14     * @param retryTopic V1 retry topic name
15     * @return the original topic name, retryTopic otherwise
16     */
17 private String findTopicForV1RetryTopic(String retryTopic) {
18     // Check if the potential group exists in topicCidMap
19     boolean hasDuplicatedTopic = false;
20     String originalTopic = null;
21     for (String topic : topicCidMap.keySet()) {
22         ConcurrentHashMap<String, Byte> cids = topicCidMap.get(topic);
23         if (cids != null) {
24             for (String cid : cids.keySet()) {
25                 // Check if this cid could be the correct consumer group
26                 String expectedRetryTopic = KeyBuilder.buildPopRetryTopicV1(to
    pic, cid);
27                 if (expectedRetryTopic.equals(retryTopic)) {
28                     if (originalTopic == null) {
29                         originalTopic = topic;
30                     }
31                 } else {
32                     hasDuplicatedTopic = true;
33                     break;
34                 }
35             }
36         }
37     }
38 }
39 if (hasDuplicatedTopic) {
40     return retryTopic;
41 } else {
42     return originalTopic;

```

```
43     }  
44     }
```

1. topicCidMap是PopLongPollingService的一个核心数据结构，存储Topic下的消费者组映射，由两层map构成，内层map存储的key就是消费者组ID (ConsumerGroup/CID)，在polling方法中被赋值。
2. 借助这个数据结构，我设计了反向查找机制，避免了字符串解析的歧义问题
 - a. 通过遍历topicCidMap中的所有topic-consumerGroup组合
 - b. 对每个组合重建V1重试Topic名称，与输入的重试Topic进行匹配
3. 当发现多个原始Topic都能生成相同的重试Topic时，标记为hasDuplicatedTopic
 - a. 在有歧义的情况下，返回原始的重试Topic名称，避免错误的消息路由，不会因为Topic命名冲突导致消息通知错误（虽然如果有冲突，错误一定会发生的，但不会因为这段代码的修改引起）

解决方案的现象

```

1  Received message #33 = MessageViewImpl{messageId=01EA2C235B8585B70D08C1575
   B000000015, topic=Topic, bornHost=U-6MCWWN14-2342.local, bornTimestamp=1756
   347739198, endpoints=ipv4:30.221.148.187:9081, deliveryAttempt=2, tag=Ta
   g, keys=[yourMessageKey-1c151062f96e], messageGroup=null, deliveryTimestam
   p=null, properties={}}
2  [RETRY] RETRY MESSAGE DETECTED!
3      MessageId: 01EA2C235B8585B70D08C1575B000000015
4      Delivery Attempt: 2
5  Received message #34 = MessageViewImpl{messageId=01EA2C235B8585B70D08C1575
   B00000000B, topic=Topic, bornHost=U-6MCWWN14-2342.local, bornTimestamp=1756
   347739178, endpoints=ipv4:30.221.148.187:9081, deliveryAttempt=2, tag=Ta
   g, keys=[yourMessageKey-1c151062f96e], messageGroup=null, deliveryTimestam
   p=null, properties={}}
6  [RETRY] RETRY MESSAGE DETECTED!
7      MessageId: 01EA2C235B8585B70D08C1575B00000000B
8      Delivery Attempt: 2
9  Received message #35 = MessageViewImpl{messageId=01EA2C235B8585B70D08C1575
   B000000008, topic=Topic, bornHost=U-6MCWWN14-2342.local, bornTimestamp=1756
   347739171, endpoints=ipv4:30.221.148.187:9081, deliveryAttempt=2, tag=Ta
   g, keys=[yourMessageKey-1c151062f96e], messageGroup=null, deliveryTimestam
   p=null, properties={}}
10 [RETRY] RETRY MESSAGE DETECTED!
11      MessageId: 01EA2C235B8585B70D08C1575B000000008
12      Delivery Attempt: 2
13      Time since first failure: 3072 ms (3.072 seconds)
14      Time since first failure: 3074 ms (3.074 seconds)
15      Time since first failure: 3073 ms (3.073 seconds)
16 Received message #36 = MessageViewImpl{messageId=01EA2C235B8585B70D08C1575
   B000000003, topic=Topic, bornHost=U-6MCWWN14-2342.local, bornTimestamp=1756
   347739152, endpoints=ipv4:30.221.148.187:9081, deliveryAttempt=2, tag=Ta
   g, keys=[yourMessageKey-1c151062f96e], messageGroup=null, deliveryTimestam
   p=null, properties={}}
17 [RETRY] RETRY MESSAGE DETECTED!
18      MessageId: 01EA2C235B8585B70D08C1575B000000003
19      Delivery Attempt: 2
20      Time since first failure: 3075 ms (3.075 seconds)
21      First failed at: Thu Aug 28 10:22:27 CST 2025
22      First failed at: Thu Aug 28 10:22:27 CST 2025
23      Retried at: Thu Aug 28 10:22:30 CST 2025
24 Received message #37 = MessageViewImpl{messageId=01EA2C235B8585B70D08C1575
   B00000001C, topic=Topic, bornHost=U-6MCWWN14-2342.local, bornTimestamp=1756
   347739209, endpoints=ipv4:30.221.148.187:9081, deliveryAttempt=2, tag=Ta
   g, keys=[yourMessageKey-1c151062f96e], messageGroup=null, deliveryTimestam
   p=null, properties={}}
25 [RETRY] RETRY MESSAGE DETECTED!
26      MessageId: 01EA2C235B8585B70D08C1575B00000001C
27      Delivery Attempt: 2

```

```

28 First failed at: Thu Aug 28 10:22:27 CST 2025
29 Time since first failure: 3074 ms (3.074 seconds)
30 First failed at: Thu Aug 28 10:22:27 CST 2025
31 First failed at: Thu Aug 28 10:22:27 CST 2025
32 Retried at: Thu Aug 28 10:22:30 CST 2025
33 -----
34 -----
35 [✓] Message #33 - SUCCESS
36 Retried at: Thu Aug 28 10:22:30 CST 2025
37 -----
38 [✓] Message #34 - SUCCESS
39 Retried at: Thu Aug 28 10:22:30 CST 2025
40 -----
41 [✓] Message #37 - SUCCESS
42 Retried at: Thu Aug 28 10:22:30 CST 2025
43 -----
44 [✓] Message #35 - SUCCESS
45 [✓] Message #36 - SUCCESS
46

```

几乎立刻唤醒，符合预期。

实验结果

现象一：改动前：使用remoting协议pushconsumer，一次性发送32条

```

1 [NORMAL] Topic: TopicA, QueueId: 4, ReconsumeTimes: 0, Content: Hello world
2 [NORMAL] Topic: TopicA, QueueId: 3, ReconsumeTimes: 0, Content: Hello world
3     ✗ 不ACK此消息，让其进入重试队列
4     ✗ 不ACK此消息，让其进入重试队列
5 [RETRY] Topic: TopicA, QueueId: 0, ReconsumeTimes: 1, Content: Hello world
6     ✗ 第1条重试消息延迟: 14955毫秒 (15.0秒)
7 [RETRY] Topic: TopicA, QueueId: 0, ReconsumeTimes: 1, Content: Hello world
8     ✗ 第2条重试消息延迟: 14955毫秒 (15.0秒)

```

1. 对于收到的前两个消息我们选择不ack
2. 生产者不发送任何消息
3. 15s之后，收到重试队列的消息，可以重新消费

4. 实验现象：

- a. 消息第一次被nack的时间: 2025-08-22 16:05:21,414
- b. 消费者从retry拉取到消息的时间: 2025-08-22 16:05:36,385

c. mqadmin topicStatus 查看重试队列更新时间为 2025-08-22 16:05:33,497

5. 推论: `BROKER_SUSPEND_MAX_TIME_MILLIS = 1000 * 15` 常量在起作用。当V1重试主题无法正确唤醒长轮询时, Pop请求会等待直到这个15秒超时, 然后返回并重新发起请求, 此时才能拉取到重试队列中的消息。

现象二: 改动前使用remoting协议pushconsumer, 但持续发送消息

```
1 [NORMAL] Topic: TopicTestForNormal, QueueId: 3, ReconsumeTimes: 0, Content: Hello world
2     ✗ 不ACK此消息, 让其进入重试队列
3 [NORMAL] Topic: TopicTestForNormal, QueueId: 4, ReconsumeTimes: 0, Content: Hello world
4     ✗ 不ACK此消息, 让其进入重试队列
5 [NORMAL] Topic: TopicTestForNormal, QueueId: 5, ReconsumeTimes: 0, Content: Hello world
6     ✗ 不ACK此消息, 让其进入重试队列
7 [RETRY] Topic: TopicTestForNormal, QueueId: 0, ReconsumeTimes: 1, Content: Hello world
8     ⚡ 第1条重试消息延迟: 8067毫秒 (8.1秒)
9     🔍 Pop属性: 20 1755848819966 5000 1 1 broker-a 0 20
10 [RETRY] Topic: TopicTestForNormal, QueueId: 0, ReconsumeTimes: 1, Content: Hello world
11     ⚡ 第2条重试消息延迟: 9075毫秒 (9.1秒)
12     🔍 Pop属性: 21 1755848820971 5000 3 1 broker-a 0 21
13 [RETRY] Topic: TopicTestForNormal, QueueId: 0, ReconsumeTimes: 1, Content: Hello world
14     ⚡ 第3条重试消息延迟: 11087毫秒 (11.1秒)
```

1. 对于收到的前两个消息我们选择不ack
2. 生产者每秒发送一条消息, 消息不可见时间设置为5s
3. 8s之后, 收到第一个重试队列的消息, 可以重新消费
4. 推论: 在持续发送消息时, 在invisibletime结束的时候, 因为新消息的到来可以触发长轮询, 可以及时地收到retrytopic里的消息。

现象三: 改动前使用grpc协议pushconsumer, 一次性发送32条

```
1 Time since first failure: 19973 ms (19.973 seconds)
2 First failed at: Wed Aug 27 20:14:24 CST 2025
3 Retried at: Wed Aug 27 20:14:44 CST 2025
```

从Nack到第二次拉取的时间间隔为20s

```
1 sh mqadmin topicstatus -n 127.0.0.1:9876 -t %RETRY%ConsumerGroupPush_Topic
  -c DefaultCluster
2 #Last Updated
3 2025-08-27 20:14:27,841
```

但是进入重试队列的时间是3s

现象四：改动后使用grpc协议pushconsumer，持续发送消息

```
1 ✓ Message #33 - SUCCESS
2 First failed at: 2025-08-29 11:20:36,645
3 Retried at: 2025-08-29 11:20:39,724
```

11:20:39,716进入重试队列，11:20:39,724的时候被拉取到，因此重试时间只有3s。

```
1 bin % sh mqadmin topicstatus -n 127.0.0.1:9876 -t %RETRY%ConsumerGroup2_Top
  ic
2 #Broker Name                #QID  #Min Offset          #Max Offset
  #Last Updated
3 broker-a                    0      0                    20
  2025-08-29 11:20:39,716
```

证明修改之后，pop消费可以立刻被唤醒。

pop日志如下，可以看到我添加的originTopic Property字段可以被正确解析，可以用来唤醒对应的topic+cid的pop长轮询。


- 1 ▾ 2025-08-29 11:20:39 INFO ReputMessageService - Processing retry topic: %RETRY%ConsumerGroup2_Topic, originTopic: Topic, properties: {ORIGIN_TOPIC=Topic, MSG_REGION=DefaultRegion, UNIQ_KEY=0106E51CE80829952C08C2B62600000008, CLUSTER=DefaultCluster, 1ST_POP_TIME=1756437636661, PGROUP=Topic, RECONSUME_TIME=0, TAGS=Tag, __BORNHOST=U-6MCWWN14-2342.local, BORN_TIMESTAMP=1756437541560, KEYS=yourMessageKey-1c151062f96e, TRACE_ON=true}
- 2 ▾ 2025-08-29 11:20:39 INFO ReputMessageService - Processing retry topic: %RETRY%ConsumerGroup2_Topic, originTopic: Topic, properties: {ORIGIN_TOPIC=Topic, MSG_REGION=DefaultRegion, UNIQ_KEY=0106E51CE80829952C08C2B62F000000011, CLUSTER=DefaultCluster, 1ST_POP_TIME=1756437636658, PGROUP=Topic, RECONSUME_TIME=0, TAGS=Tag, __BORNHOST=U-6MCWWN14-2342.local, BORN_TIMESTAMP=1756437550700, KEYS=yourMessageKey-1c151062f96e, TRACE_ON=true}
- 3 ▾ 2025-08-29 11:20:39 INFO ReputMessageService - Processing retry topic: %RETRY%ConsumerGroup2_Topic, originTopic: Topic, properties: {ORIGIN_TOPIC=Topic, MSG_REGION=DefaultRegion, UNIQ_KEY=0106E51CE80829952C08C2B62F000000011, CLUSTER=DefaultCluster, 1ST_POP_TIME=1756437636658, PGROUP=Topic, RECONSUME_TIME=0, TAGS=Tag, __BORNHOST=U-6MCWWN14-2342.local, BORN_TIMESTAMP=1756437550700, KEYS=yourMessageKey-1c151062f96e, TRACE_ON=true}

现象五：改动后使用grpc协议simpleconsumer，一次性发送32条

设置不可见时间为10s

第一次拉取消息的时间为2025-08-29 15:56:39,876

第二次拉取到重试消息的时间为2025-08-29 15:56:52,958

- 1 ▾ Received message #15 = MessageViewImpl{messageId=010EC51B9D3D4AD91E08C2F737000000004, topic=Topic, bornHost=U-6MCWWN14-2342.local, bornTimestamp=1756454199849, endpoints=ipv4:30.221.148.229:9081, deliveryAttempt=2, tag=Tag, keys=[yourMessageKey-1c151062f96e], messageGroup=null, deliveryTimestamp=null, properties={}}
- 2  RETRY MESSAGE DETECTED!
- 3 MessageId: 010EC51B9D3D4AD91E08C2F737000000004
- 4 Delivery Attempt: 2
- 5 Time since first failure: 13082 ms (13.082 seconds)
- 6 First failed at: 2025-08-29 15:56:39,876
- 7 Retried at: 2025-08-29 15:56:52,958

最后一条消息进入%RETRY%ConsumerGroup3_Topic重试队列的时间为15:56:52,944

```

1 bin % sh mqadmin topicstatus -n 127.0.0.1:9876 -t %RETRY%ConsumerGroup3_Top
  ic
2 #Broker Name                                #QID  #Min Offset          #Max Offset
   #Last Updated
3 broker-a                                0      0                      20
   2025-08-29 15:56:52,944

```

pop日志如下:

```

1 2025-08-29 15:56:52 INFO ReputMessageService - Processing retry topic: %RET
  RY%ConsumerGroup3_Topic, originTopic: Topic, properties: {ORIGIN_TOPIC=Topi
  c, MSG_REGION=DefaultRegion, UNIQ_KEY=010EC51B9D3D4AD91E08C2F73600000003, C
  LUSTER=DefaultCluster, 1ST_POP_TIME=1756454198837, PGROUP=Topic, RECONSUME_
  TIME=0, TAGS=Tag, __BORNHOST=U-6MCWWN14-2342.local, BORN_TIMESTAMP=17564541
  98821, KEYS=yourMessageKey-1c151062f96e, TRACE_ON=true}
2 2025-08-29 15:56:52 INFO ReputMessageService - Processing retry topic: %RET
  RY%ConsumerGroup3_Topic, originTopic: Topic, properties: {ORIGIN_TOPIC=Topi
  c, MSG_REGION=DefaultRegion, UNIQ_KEY=010EC51B9D3D4AD91E08C2F737000000004, C
  LUSTER=DefaultCluster, 1ST_POP_TIME=1756454199864, PGROUP=Topic, RECONSUME_
  TIME=0, TAGS=Tag, __BORNHOST=U-6MCWWN14-2342.local, BORN_TIMESTAMP=17564541
  99849, KEYS=yourMessageKey-1c151062f96e, TRACE_ON=true}
3 2025-08-29 15:56:52 INFO ReputMessageService - Processing retry topic: %RET
  RY%ConsumerGroup3_Topic, originTopic: Topic, properties: {ORIGIN_TOPIC=Topi
  c, MSG_REGION=DefaultRegion, UNIQ_KEY=010EC51B9D3D4AD91E08C2F737000000004, C
  LUSTER=DefaultCluster, 1ST_POP_TIME=1756454199864, PGROUP=Topic, RECONSUME_
  TIME=0, TAGS=Tag, __BORNHOST=U-6MCWWN14-2342.local, BORN_TIMESTAMP=17564541
  99849, KEYS=yourMessageKey-1c151062f96e, TRACE_ON=true}

```

可以证明我们现在的消费延迟时间从之前的几秒十几秒, 压缩到了毫秒级唤醒。

其他问题探索

思考使用attemptId减少pop orderly阻塞的问题

难点不是在于找到合适的解决方案, 只要复用attemptId是可以很简单地实现重入。这个点问题在于定义什么场景需要解决阻塞, 很多时候我发现阻塞已经是合理的解决方案。

经过深入地阅读相关的代码, 我发现只有在grpc的push consumer里面使用了attemptId, 而且只在网络超时之后复用。对于remoting协议, 对于attemptId每次都传的是null, 因此服务端把每次请求都当作不一样的请求。

因此, 这个问题如果想要深入做, 更多地利用attemptId, 那么可能的选择就是

1. 归纳一些阻塞场景，区分哪些是可重入哪些是不可重入的。
2. 将这个概念透出，又simple consumer的用户自己控制它的逻辑。

ack不可靠可能导致pop orderly阻塞

顺序消息场景下，pushconsumer在一定程度上保证了消息被按顺序消费，除了网络问题理应按顺序收到ack。（但是网络并不可靠，有可能存在因为有些消息的ack没有被投递成功，导致消费阻塞）

对于这个问题：可能可以采取累计确认的方式（后续的ACK可以确认之前所有的数据）/在一个队列只能顺序消费的时候没有问题，但是如果增加了并行消费的能力，可能就会出错了。而且目前这种策略问题也不大，没有收到ack其实就应该阻塞的。