消息队列 RocketMQ 版

客户端实践教程

腾讯云

客户端实践教程

最近更新时间：2024-07-11 16:19:32

发送方实践

选择 Topic 还是 Tags

Topic 是消息主题，用来区分同一类型的业务消息，Tag 是消息的一个特殊属性或者叫子类型，下游可以通过过滤表达式在 Broker 端做高效的消费过滤。
选用 Topic 还是 Tag 可以通过以下几方面判断：

1. 消息类型是否一致：目前普通消息、定时消息、事务消息、顺序消息是不能混用的，如果不一样一定要选用不同的 Topic。
2. 业务场景是否一致：例如订单消息和支付消息，建议选用不同的 Topic，如果订单消息下游有不同省份地域过滤的诉求，可以将城市标记为 Tag。
3. 消息量是否一致：因为 RocketMQ 消费进度按 Topic 维护的，如果不同类型之前量级差异非常大，量小的 Tag 要过滤掉大量无用消息才能消费到，造成无效资源消耗和消费延迟增加。

综合看，推荐 Topic 类区分没有业务关联的消息，使用 Tag 来区分同一类 topic 中的关键属性，方便不同下游高效的过滤处理部分消息。

Keys 使用

推荐给将每个消息在业务层面的唯一业务标识码设置到 Key 字段，服务器会为每个消息创建索引（哈希索引），用户可以通过 Topic、Key 来查询这条消息内容，以及消息轨迹，方便排查问题。

选择合适的发送方式

RocketMQ 支持三种发送方式：

- 同步发送：适用于可靠性要求比较高的场景，如支付消息、短信通知等。
- 异步发送：适用于对响应时间敏感的业务场景，即发送端不能容忍长时间地等待 Broker 的响应。
- 单向发送：适用于不特别关心发送结果的场景，例如日志发送。

send(msg)同步发送只有发送成功才会返回结果，但会同步阻塞，增大发送耗时，如果有性能要求，可以使用异步的方式:send(msg, callback)，在回调中检查发送结果，某些应用如果不关注消息是否发送成功，例如日志收集类应用，请直接使用 sendOneWay 方法发送消息。

发送重试策略

对于消息不可丢失应用，务必要有消息重发机制，Producer的send方法本身支持内部重试：

1. 至多重试2次（同步发送为2次，异步发送为0次）。
2. 如果发送失败，则轮转到下一个 Broker。这个方法的总耗时时间不超过 sendMsgTimeout 设置的值，默认 3s。

3. 如果本身向 broker 发送消息产生超时异常，就不会再重试。

如果是关键业务消息，消息发送失败后建议将消息存储到 db，然后由定时器类线程进行定时重试，确保 RocketMQ 服务恢复后消息重新发送到下游消费者。

打印发送消息日志

无论发送成功还是失败，都推荐将发送结果 sendresult、msgid、key、tag 等属性都打印到日志中，方便排查定位问题。

消费方实践教程

使用消费组隔离不同的下游业务

不同的消费业务可以使用消费组独立地消费同样的主题，并且每个消费者都有自己的消费偏移量(offsets)，确保同一组中的每个消费者订阅相同的主题，使用相同的过滤规则。

消费幂等处理

RocketMQ 在原理上保证至少消费一次，无法避免消息重复，例如发送时网络抖动会造成重试，投递时消费异常也会重试，以及消费者重启变化时负载均衡也会造成消息重复，所以如果业务对消费重复非常敏感，务必要在业务层面进行去重处理。可以借助分布式缓存或关系数据库进行去重。
幂等处理时，首先需要确定消息的唯一键，不推荐依赖 msgid，推荐使用设置到 keys 字段的唯一业务标识字段，例如订单Id等，在消费之前判断唯一键是否在分布式缓存或关系数据库中存在，如果不存在则插入，并消费，否则跳过，或者根据业务逻辑做进一步去重处理。

消费失败重试

并发消费：并发消费消费失败采用的是退避重试的机制，消费失败的消息将发回系统延时队列，每一次消费失败，delayLevel 将 +1，默认最大重试次数为 16 次，超过最大次数将进入死信队列。重试 16 次的时间间隔分别对应延时消息的 level3 ~ level18，时间为 10s ~ 2h，详见 消息重试 。
顺序消费：顺序消费为了保证顺序，不会发回服务端，采用的是本地不断重试的机制，默认重试消费次数为 Integer.MAX_VALUE, 可设置最大重试次数，超过最大次数将进入死信队列。
也可以通过 maxReconsumeTimes 配置最大重试次数。

提升消费并行度

大部分消息消费行为都属于 IO 密集型，即可能在处理消息过程中操作数据库或者调用 RPC，这类消费行为的消费速度在于后端数据库或者外系统的吞吐量，通过增加消费并行度，可以提高总的消费吞吐量。

1. 增加消费节点个数，同一个 ConsumerGroup 下，通过增加 Consumer 实例节点数量来提高并行度（需要注意的是超过订阅队列数的 Consumer 实例无效）。可以通过加机器，或者在已有机器启动多个进程的方式。
2. 增加单个消费节点的线程数，通过修改Consumer 的参数 consumeThreadMin、consumeThreadMax，增加并发线程个数实现更高的并发度。
3. 批量方式消费——某些业务流程如果支持批量方式消费，则可以很大程度上提高消费吞吐量，例如订单扣款类应用，一次处理一个订单耗时 1 s，一次处理 10 个订单可能也只耗时 2 s，这样即可大幅度提高消费的吞吐量，通

过设置 consumer的 consumeMessageBatchMaxSize 返个参数，默认是 1，即一次只消费一条消息，例如设置为 N，那么每次消费的消息数小于等于 N。

4. 跳过非关键的消息，发生消息堆积时，如果业务对数据要求不高时，可以通过根据时间过滤跳过过期消息，或根据业务选择丢弃不重要的消息，提升消息处理效率。

订阅关系一致性

一个 ConsumerGroup 通常代表一组消费业务逻辑一致的多个节点，订阅关系一致指的是同一个 Consumer Group 下所有 Consumer 实例的订阅关系一致，否则消息消费的逻辑可能会混乱，甚至导致消息丢失。
尽量保证订阅关系变化的兼容性，建议不要变化订阅 Topic， 过滤表达式只做增量订阅，在同一个 Group 不同的消费节点发布的过程中，就会出现不同消费节点之间过滤规则不一致，Broker 过滤消息时，不同消费节点生效不同的过滤规则，所以同一个 Group 的订阅关系一旦变化，要关注订阅关系的兼容性，避免漏收到消息。
腾讯云 RocketMQ 管控台在消费者详情页面提供了 订阅关系不一致的诊断功能，可以看到哪些节点出现了订阅关系不一致的情况。

打印消费消息日志

无论发送成功还是失败，都推荐收到消息先打印一条日志，包括 msgid、key、tag 等属性都打印到日志中，方便排查定位问题，等消息处理完成，再将msgid、key、tag，重试次数和消费结果也打印到日志中。

其他消费建议

关于消费者和订阅

第一件需要注意的事情是，不同的消费者组可以独立的消费一些 Topic，并且每个消费者组都有自己的消费偏移量，请确保同一组内的每个消费者订阅信息保持一致。

关于有序消息

消费者将锁定每个消息队列，以确保他们被逐个消费，虽然这将会导致性能下降，但是当您关心消息顺序的时候会很有用。我们不建议抛出异常，您可以返回 ConsumeOrderlyStatus.SUSPEND_CURRENT_QUEUE_A_MOMENT 作为替代。

关于并发消费

顾名思义，消费者将并发消费这些消息，建议您使用它来获得良好性能，我们不建议抛出异常，您可以返回 ConsumeConcurrentlyStatus.RECONSUME_LATER 作为替代。

关于消费状态 Consume Status

对于并发的消费监听器，您可以返回 RECONSUME_LATER 来通知消费者现在不能消费这条消息，并且希望可以稍后重新消费它。然后，您可以继续消费其他消息。对于有序的消息监听器，因为您关心它的顺序，所以不能跳过消息，但是您可以返回SUSPEND_CURRENT_QUEUE_A_MOMENT 告诉消费者等待片刻。

关于 Blocking

不建议阻塞监听器，因为它会阻塞线程池，并最终可能会终止消费进程

关于消费位点

当建立一个新的消费者组时，需要决定是否需要消费已经存在于 Broker 中的历史消息 CONSUME_FROM_LAST_OFFSET 将会忽略历史消息，并消费之后生成的任何消息。CONSUME_FROM_FIRST_OFFSET 将会消费每个存在于 Broker 中的信息。您也可以使用 CONSUME_FROM_TIMESTAMP 来消费在指定时间戳后产生的消息。

问题排查实践

SDK日志

RocketMQ 客户端与服务端的通信协议是比较复杂的，例如消费队列的分配，主题的寻址，都是客户端 SDK 的实现决定的，这些过程的重要信息都保存在SDK 日志里。因此，当出现生产或消费问题时，SDK 日志是排查问题的最重要手段之一，请务必要保存这些日志。通常情况，SDK 日志不与业务日志打印在同一个文件中，我们给出常用 SDK 的日志默认路径，如下：

SDK 语言	SDK 协议	路径
Java	remoting	~/logs/rocketmqlogs/rocketmq_client.log
Java	grpc	~/logs/rocketmq/rocketmq_client.log
Go	remoting	/tmp/rocketmq-client.log
Go	grpc	~/logs/rocketmqlogs/rocketmq_client_go.log