链滴

rabbitmq 如何提高可靠性并保证消费端幂等

作者：Ahian

原文链接：https://ld246.com/article/1585495036072

来源网站：链滴

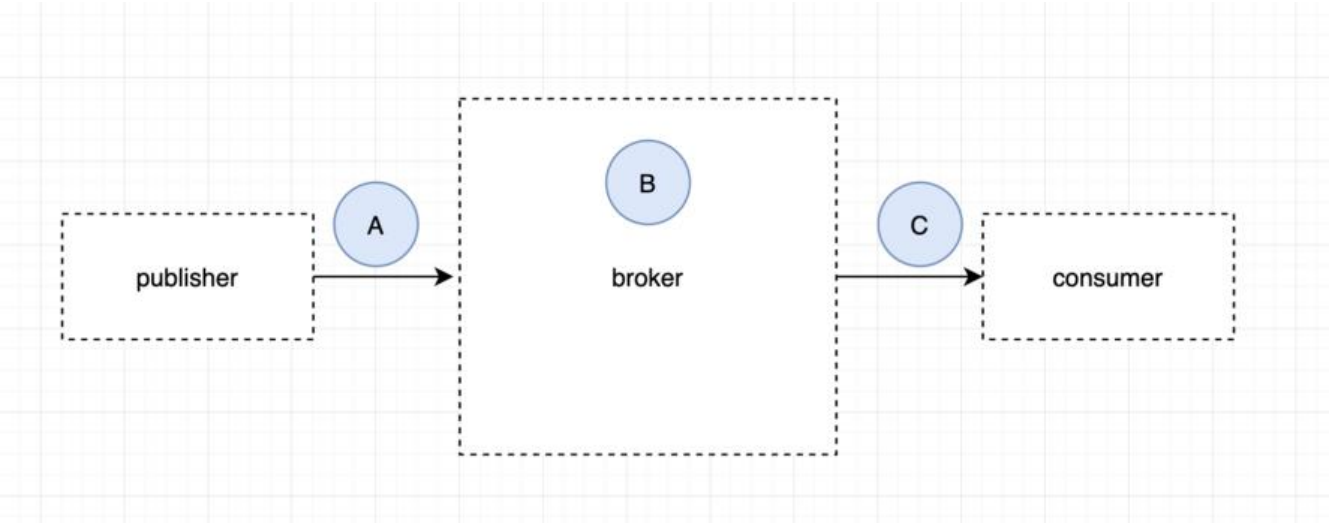
什么是消息的可靠性

简单讲就是，一条消息由生产者发送出来，到 broker 上，存储到消息队列，再被消费者成功的消费
如果消息传着传着就传没了，此时消息就是不可靠的。

为什么要提高消息可靠性

拿订单或者交易举例，但凡涉及到与钱相关的系统，不允许出现任何数据偏差，如果因为在使用消息列而丢失了数据，那这算是一个重大事故，好比你申请了几万元的退款，但是后台在发消息的时候恰把你的这条退款消息丢了，还得去打客服电话，虽然钱不会少但是心里肯定会不爽。

如何提高消息的可靠性



提高消息消息可靠性需要考虑什么情况下消息会丢失，如上图，可以分三个部分来讲（排除一些非正的使用，比如说删队列、删交换机这样的）

A：publisher 到 broker 之间可能出现的情况

- 消息发出去后网络出问题，没到 broker上
- 消息到达 broker 后没有对应的 exchange
- 消息到 exchange 后没有与之绑定的队列
- 生产者发送完消息后重启（不影响可靠性）

B: broker 可能出现的情况

- broker 宕机情况

C: broker 到 consumer 可能出现的情况

- 消费者应用刚要处理消息，结果就重启了

处理

原文链接：rabbitmq 如何提高可靠性并保证消费端幂等

针对 A 情况

官网提供两种确保消费者成功投递的方式，一种是使用**事务**，一种是使用 **confirm 机制**。对于具体果使用本文不做展开，开启事务是一种昂贵的操作，官方建议使用 confirm 机制，能比事务机制快上0倍左右，对于 confirm，一般使用 Spring AMQP 中的 ConfirmCallback 来实现

```
@FunctionalInterface
public interface ConfirmCallback {

    /**
     * Confirmation callback.
     * @param correlationData correlation data for the callback.
     * @param ack true for ack, false for nack
     * @param cause An optional cause, for nack, when available, otherwise null.
     */
    void confirm(@Nullable CorrelationData correlationData, boolean ack, @Nullable String ca
se);
}
```

其中 CorrelationData 是在发送端发送消息时一并发送的，里面有一个不可变的唯一 id，在 confirm 中尤为重要，可以利用它实现补偿机制。

Confirm 机制能够保证消息到达 broker 上并且**有对应的交换机**，但是不能保证是否路由到队列，此就需要 ReturnCallback 来实现

```
@FunctionalInterface
public interface ReturnCallback {

    /**
     * Returned message callback.
     * @param message the returned message.
     * @param replyCode the reply code.
     * @param replyText the reply text.
     * @param exchange the exchange.
     * @param routingKey the routing key.
     */
    void returnedMessage(Message message, int replyCode, String replyText,
            String exchange, String routingKey);
}
```

消息到不了队列，就没法存储，此时会发生三种情况：

1. 直接丢弃
2. 重新发送
3. 转到另一个交换机上

方式1，一般使用时不会采取直接丢弃的方式，风险较大

对于方式2，要配置 **Mandatory** 为 true，告诉 broker 说如果交换机找不到匹配的队列得把消息再回给生产者。如果为 false 那就是第1种情况，此时需要注意重试的次数和方式，不能连续发送，需留出异常的处理时间，比如重试三次，间隔5秒。

方式3，这是 rabbitmq 的一个功能，称为备份交换机（alternate-exchange），是在创建正常的交

机时追加一个 alternate-exchange="XXX"的参数，XXX 为另一个交换机，接收无法路由的消息。

可能有人会发现，当消息无法路由到队列中，此时 ConfirmCallback 返回的 ack 为 true ，在这可能有很多困惑，此处列出几种情况：

● 消息投递到 exchange，触发 confirmCallback，成功为 true , 失败为 false（例如 exchange 不在）

● Exchange 无法路由到队列，此时触发 returnCallback 打印错误信息，再触发 confirmCallback，时 **ack = true**

● 消息正常路由到队列，只会触发 confirmCallback

上面几种情况是针对于 Spring 实现的两个监听器，RabbitMQ 官方客户端实现可能不同。

对于发送者来说，消息到了 exchange 其实它的任务就已经完成了。

针对 B 情况

此时我们说的主要是 broker 中的队列，因为 broker 中只有队列有存储功能，此时要保证队列的可性，需要设置持久化，能够保证 broker 重启后自动恢复数据。

1. 设置 exchange 持久化
2. 声明队列时设置为持久化队列，这样能保证队列自身能够恢复（不包括里面的消息）
3. 对消息设置 deliveryMode.PERSISTENT（默认，不需要改）

搭配 publisher confirm 机制可以确保消息写到磁盘上以后才触发 confirmCallback，能够保证消息丢失。

针对 C 情况

大多数同学可能会对消费端 ack 比较熟，对于 Spring 环境来说，设置消费者手动 ack，broker 收到费者 ack 后删除消息或者 nack 继续消费。

消息补偿机制

此处的实现思路为：

1. 发送端创建 CorrelationData 对象跟随消息一同发送
2. 将CorrelationData中的 id 作为key，发送的消息体为 value，存到本地缓存
3. 当 confirm ack 为 true 时说明发送成功，根据返回的 CorrelationData id 将缓存中的对应数据删除
4. 当 confirm ack 为 fasle 时说明失败了，此时根据 CorrelationData id 在缓存中拿到对应的消息新发送
5. ReturnCallback 的重试机制在上面已经写出

消息幂等操作

生产端为了保证消息在投递过程中不丢失，会对一条消息多次重试，消费端没有手动 ack 导致重复消，这些算是消息重复投递，在可靠性机制上无法避免，此时需要考虑如何多次处理同一条消息但是产同样效果，即幂等。

保证幂等的方式因为业务的不同实现方式也不同，可以用 redis，可以用数据库主键，一个简单的思：

1. 消费端获取消息的 MessageId 当作 key
2. 消息内容当作 value
3. 放到合适的容器中

拿 Map 举例，id 为 123 的消息，消费之前看 map 中是否存在 key=123，如果有的话跳过，没有话处理消息并添加到map中。

参考文章

- https://www.rabbitmq.com/blog/2011/02/10/introducing-publisher-confirms/
- https://www.jianshu.com/p/6579e48d18ae
- https://blog.51cto.com/4925054/2095467
- https://www.jianshu.com/p/8b77d4583bab
- http://www.throwable.club/2018/11/25/rabbitmq-send-consume-confirm/