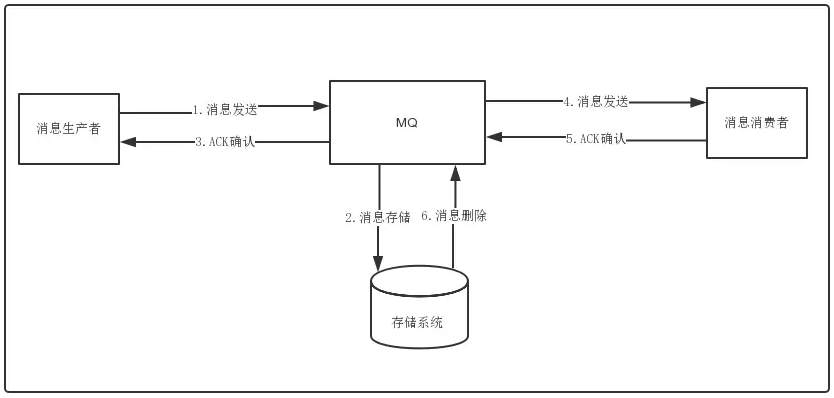
# mq

## 事务

### 普通消息

#### 执行流程

参考：<https://blog.csdn.net/qq_14828239/article/details/103967892>



1. 生成者发送消息

2. MQ收到消息，将消息进行持久化，在存储中新增一条记录

3. ACK给生产者

4. push 消息给对应的消费者，然后等待消费者返回ACK

5. 消息消费者在指定时间内成功返回ack，那么MQ认为消息消费成功，在存储中删除消息，即执行第6步；如果MQ在指定时间内没有收到ACK，则认为消息消费失败，会尝试重新push消息,重复执行4、5、6步骤

6. 删除消息

#### 一致性问题

我们以订单创建为例，订单系统先创建订单(本地事务)，再发送消息给下游处理；如果订单创建成功，然而消息没有发送出去，那么下游所有系统都无法感知到这个事件，会出现脏数据；

public void processOrder() {

// 订单处理(业务操作)

orderService.process();

// 发送订单处理成功消息(发送消息)

sendBizMsg ();

}

如果先发送订单消息，再创建订单；那么就有可能消息发送成功，但是在订单创建的时候却失败了，此时下游系统却认为这个订单已经创建，也会出现脏数据。

public void processOrder() {

// 发送订单处理成功消息(发送消息)

sendBizMsg ();

// 订单处理(业务操作)

orderService.process();

}

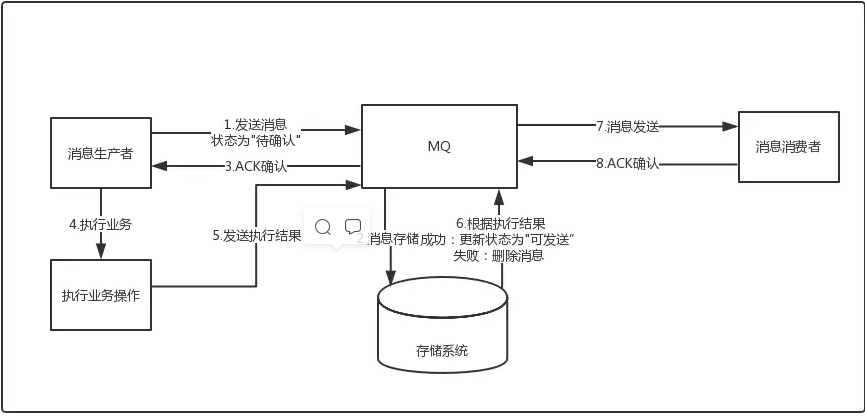
消息发送的异常情况分析



从上面的情况分析，我们可以看到，使用普通的处理方式，无论如何，都无法保证业务处理与消息发送两边的一致性，其根本的原因就在于：远程调用，结果最终可能为成功、失败、超时；而对于超时的情况，处理方最终的结果可能是成功，也可能是失败，调用方是无法知晓的。 笔者就曾经在项目中出现类似的情况，调用方先在本地写数据，然后发起RPC服务调用，但是处理方由于DB数据量比较大，导致处理超时，调用方在出现超时异常后，直接回滚本地事务，从而导致调用方这边没数据，而处理方那边数据却已经写入了，最终导致两边业务数据的不一致。为了保证两边数据的一致性，我们只能从其他地方寻找新的突破口。

### 消息事务

#### 事务消息处理的流程



1. 事务消息与普通消息的区别就在于消息生产环节，生产者首先预发送一条消息到MQ(这也被称为发送half消息)
2. MQ接受到消息后，先进行持久化，则存储中会新增一条状态为待发送的消息
3. 然后返回ACK给消息生产者，此时MQ不会触发消息推送事件
4. 生产者预发送消息成功后，执行本地事务
5. 执行本地事务，执行完成后，发送执行结果给MQ
6. MQ会根据结果删除或者更新消息状态为可发送
7. 如果消息状态更新为可发送，则MQ会push消息给消费者，后面消息的消费和普通消息是一样的

注意点：由于MQ通常都会保证消息能够投递成功，因此，如果业务没有及时返回ACK结果，那么就有可能造成MQ的重复消息投递问题。因此，对于消息最终一致性的方案，消息的消费者必须要对消息的消费支持幂等，不能造成同一条消息的重复消费的情况。

#### 事务消息异常情况分析



### 思考问题分析

1. 问：如果预发送消息失败，是不是业务就不执行了？

答：是的，对于基于消息最终一致性的方案，一般都会强依赖这步，如果这个步骤无法得到保证，那么最终也 就不可能做到最终一致性了。

1. 问：为什么要增加一个消息预发送机制，增加两次发布出去消息的重试机制，为什么不在业务成功之后，发送失败的话使用一次重试机制？

答：如果业务执行成功，再去发消息，此时如果还没来得及发消息，业务系统就已经宕机了，系统重启后，根本没有记录之前是否发送过消息，这样就会导致业务执行成功，消息最终没发出去的情况。

1. 如果consumer消费失败，是否需要producer做回滚呢？

答：这里的事务消息，producer不会因为consumer消费失败而做回滚，采用事务消息的应用，其所追求的是高可用和最终一致性，消息消费失败的话，MQ自己会负责重推消息，直到消费成功。因此，事务消息是针对生产端而言的，而消费端，消费端的一致性是通过MQ的重试机制来完成的。

1. 如果consumer端因为业务异常而导致回滚，那么岂不是两边最终无法保证一致性?

答：基于消息的最终一致性方案必须保证消费端在业务上的操作没障碍，它只允许系统异常的失败，不允许业务上的失败，比如在你业务上抛出个NPE之类的问题，导致你消费端执行事务失败，那就很难做到一致了。

参考：https://blog.csdn.net/lixiangfei4/article/details/104306911?spm=1001.2101.3001.6650.1&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc\_relevant\_default&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc\_relevant\_default