<https://blog.csdn.net/u013256816/article/details/60875666/>

消息的可靠性是RabbitMQ的一大特色，那么RabbitMQ是如何保证消息可靠性的呢——消息持久化。

为了保证RabbitMQ在退出或者crash等异常情况下数据没有丢失，需要将queue，exchange和Message都持久化。

## queue的持久化

channel.queueDeclare("queue.persistent.name", true, false, false, null);

关键的是第二个参数设置为true,即durable=true.

queue的持久化是通过durable=true来实现的。

## 设置消息的持久化：

channel.basicPublish("exchange.persistent", "persistent", MessageProperties.PERSISTENT\_TEXT\_PLAIN, "persistent\_test\_message".getBytes());

这里的关键是：MessageProperties.PERSISTENT\_TEXT\_PLAIN

这里关键的是BasicProperties props这个参数了，这里看下BasicProperties的定义：

这里的deliveryMode=1代表不持久化，deliveryMode=2代表持久化。

## exchange的持久化

一般只需要：channel.exchangeDeclare(exchangeName, “direct/topic/header/fanout”, true);即在声明的时候讲durable字段设置为true即可。

## 首先，从consumer端来说，

如果这时autoAck=true，那么当consumer接收到相关消息之后，还没来得及处理就crash掉了，那么这样也算数据丢失，这种情况也好处理，只需将autoAck设置为false(方法定义如下)，然后在正确处理完消息之后进行手动ack（channel.basicAck）.

String basicConsume(String queue, boolean autoAck, Consumer callback) throws IOException;

### 消息确认（Consumer端）

为了保证消息从队列可靠地到达消费者，RabbitMQ提供消息确认机制(message acknowledgment)。消费者在声明队列时，可以指定noAck参数，当noAck=false时，RabbitMQ会等待消费者显式发回ack信号后才从内存(和磁盘，如果是持久化消息的话)中移去消息。否则，RabbitMQ会在队列中消息被消费后立即删除它。

采用消息确认机制后，只要令noAck=false，消费者就有足够的时间处理消息(任务)，不用担心处理消息过程中消费者进程挂掉后消息丢失的问题，因为RabbitMQ会一直持有消息直到消费者显式调用basicAck为止。

当noAck=false时，对于RabbitMQ服务器端而言，队列中的消息分成了两部分：一部分是等待投递给消费者的消息；一部分是已经投递给消费者，但是还没有收到消费者ack信号的消息。如果服务器端一直没有收到消费者的ack信号，并且消费此消息的消费者已经断开连接，则服务器端会安排该消息重新进入队列，等待投递给下一个消费者（也可能还是原来的那个消费者）。

RabbitMQ不会为未ack的消息设置超时时间，它判断此消息是否需要重新投递给消费者的唯一依据是消费该消息的消费者连接是否已经断开。这么设计的原因是RabbitMQ允许消费者消费一条消息的时间可以很久很久。

## 引入RabbitMQ的mirrored-queue即镜像队列

其次，关键的问题是消息在正确存入RabbitMQ之后，还需要有一段时间（这个时间很短，但不可忽视）才能存入磁盘之中，RabbitMQ并不是为每条消息都做fsync的处理，可能仅仅保存到cache中而不是物理磁盘上，在这段时间内RabbitMQ broker发生crash, 消息保存到cache但是还没来得及落盘，那么这些消息将会丢失。那么这个怎么解决呢？首先可以引入RabbitMQ的mirrored-queue即镜像队列，这个相当于配置了副本，当master在此特殊时间内crash掉，可以自动切换到slave，这样有效的保障了HA, 除非整个集群都挂掉，这样也不能完全的100%保障RabbitMQ不丢消息，但比没有mirrored-queue的要好很多，很多现实生产环境下都是配置了mirrored-queue的。

## producer引入事务机制或者Confirm机制

还有要在producer引入事务机制或者Confirm机制来确保消息已经正确的发送至broker端，有关RabbitMQ的事务机制或者Confirm机制可以参考：RabbitMQ之消息确认机制（事务+Confirm）. 幸亏本文的主题是讨论RabbitMQ的持久化而不是可靠性，不然就一发不可收拾了。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### 事务机制

这里首先探讨下RabbitMQ事务机制。

RabbitMQ中与事务机制有关的方法有三个：txSelect(), txCommit()以及txRollback(), txSelect用于将当前channel设置成transaction模式，txCommit用于提交事务，txRollback用于回滚事务，在通过txSelect开启事务之后，我们便可以发布消息给broker代理服务器了，如果txCommit提交成功了，则消息一定到达了broker了，如果在txCommit执行之前broker异常崩溃或者由于其他原因抛出异常，这个时候我们便可以捕获异常通过txRollback回滚事务了。

已经在transaction事务模式的channel是不能再设置成confirm模式的，即这两种模式是不能共存的。

RabbitMQ的可靠性涉及producer端的确认机制、broker端的镜像队列的配置以及consumer端的确认机制，要想确保消息的可靠性越高，那么性能也会随之而降，鱼和熊掌不可兼得，关键在于选择和取舍。

### producer端confirm模式的实现原理

生产者将信道设置成confirm模式，一旦信道进入confirm模式，所有在该信道上面发布的消息都会被指派一个唯一的ID(从1开始)，一旦消息被投递到所有匹配的队列之后，broker就会发送一个确认给生产者（包含消息的唯一ID）,这就使得生产者知道消息已经正确到达目的队列了，如果消息和队列是可持久化的，那么确认消息会将消息写入磁盘之后发出，broker回传给生产者的确认消息中deliver-tag域包含了确认消息的序列号，此外broker也可以设置basic.ack的multiple域，表示到这个序列号之前的所有消息都已经得到了处理。

confirm模式最大的好处在于他是异步的，一旦发布一条消息，生产者应用程序就可以在等信道返回确认的同时继续发送下一条消息，当消息最终得到确认之后，生产者应用便可以通过回调方法来处理该确认消息，如果RabbitMQ因为自身内部错误导致消息丢失，就会发送一条nack消息，生产者应用程序同样可以在回调方法中处理该nack消息。

在channel 被设置成 confirm 模式之后，所有被 publish 的后续消息都将被 confirm（即 ack） 或者被nack一次。但是没有对消息被 confirm 的快慢做任何保证，并且同一条消息不会既被 confirm又被nack 。

**开启confirm模式的方法**

生产者通过调用channel的confirmSelect方法将channel设置为confirm模式，如果没有设置no-wait标志的话，broker会返回confirm.select-ok表示同意发送者将当前channel信道设置为confirm模式(从目前RabbitMQ最新版本3.6来看，如果调用了channel.confirmSelect方法，默认情况下是直接将no-wait设置成false的，也就是默认情况下broker是必须回传confirm.select-ok的)。

### ****编程模式****

对于固定消息体大小和线程数，如果消息持久化，生产者confirm(或者采用事务机制)，消费者ack那么对性能有很大的影响.

消息持久化的优化没有太好方法，用更好的物理存储（SAS, SSD, RAID卡）总会带来改善。生产者confirm这一环节的优化则主要在于客户端程序的优化之上。归纳起来，客户端实现生产者confirm有三种编程方式：

1. 普通confirm模式：每发送一条消息后，调用waitForConfirms()方法，等待服务器端confirm。实际上是一种串行confirm了。
2. 批量confirm模式：每发送一批消息后，调用waitForConfirms()方法，等待服务器端confirm。
3. 异步confirm模式：提供一个回调方法，服务端confirm了一条或者多条消息后Client端会回调这个方法。

## 消息什么时候刷到磁盘？

写入文件前会有一个Buffer,大小为1M,数据在写入文件时，首先会写入到这个Buffer，如果Buffer已满，则会将Buffer写入到文件（未必刷到磁盘）。

有个固定的刷盘时间：25ms,也就是不管Buffer满不满，每个25ms，Buffer里的数据及未刷新到磁盘的文件内容必定会刷到磁盘。

每次消息写入后，如果没有后续写入请求，则会直接将已写入的消息刷到磁盘：使用Erlang的receive x after 0实现，只要进程的信箱里没有消息，则产生一个timeout消息，而timeout会触发刷盘操作。