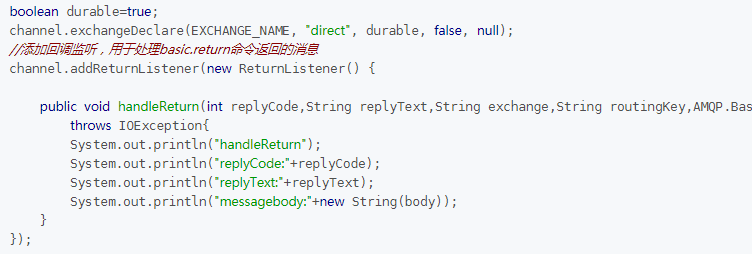
通过生产者和消费者都声明同样的交换机同样的队列，可以保证无论生产者还是消费者先运行起来，都不会由于交换机或者队列不存在而出现错误

## Mandatory

事务机制和消息确认机制都是为了保证异常状态下的消息不丢失，其实正常状态下也可能存在消息丢失问题，例如交换机按照路由规则未找到该消息对应的队列。confirm机制配合mandatory标志使用可以实现消息发送的可靠性，且性能较好。

上一篇博客中，曾经介绍过生产者发送消息时可以设置的一个标志位mandatory。

当mandatory标志位设置为true时，如果exchange根据自身类型和消息routeKey无法找到一个符合条件的queue，那么会调用basic.return方法将消息返还给生产者, channel.addReturnListener添加一个监听器，当broker执行basic.return方法时，会回调handleReturn方法，这样我们就可以处理变为死信的消息了；当mandatory设为false时，出现上述情形broker会直接将消息扔掉;通俗的讲，mandatory标志告诉broker代理服务器至少将消息route到一个队列中，否则就将消息return给发送者。



## Publisher Confirm

1. 事务机制 VS Publisher Confirm

       如果采用标准的 AMQP 协议，则唯一能够保证消息不会丢失的方式是利用事务机制 -- 令 channel 处于 transactional 模式、向其 publish 消息、执行 commit 动作。在这种方式下，事务机制会带来大量的多余开销，并会导致吞吐量下降 250% 。为了补救事务带来的问题，引入了 confirmation 机制（即 Publisher Confirm）。

     为了使能 confirm 机制，client 首先要发送 confirm.select 方法帧。取决于是否设置了 no-wait 属性，broker 会相应的判定是否以 confirm.select-ok 进行应答。一旦在 channel 上使用 confirm.select方法，channel 就将处于 confirm 模式。处于 transactional 模式的 channel 不能再被设置成 confirm 模式，反之亦然。

    一旦 channel 处于 confirm 模式，broker 和 client 都将启动消息计数（以 confirm.select 为基础从 1 开始计数）。broker 会在处理完消息后，在当前 channel 上通过发送 basic.ack 的方式对其进行 confirm 。delivery-tag 域的值标识了被 confirm 消息的序列号。broker 也可以通过设置 basic.ack 中的 multiple 域来表明到指定序列号为止的所有消息都已被 broker 正确的处理了。

       在异常情况中，broker 将无法成功处理相应的消息，此时 broker 将发送 basic.nack 来代替 basic.ack 。在这个情形下，basic.nack 中各域值的含义与 basic.ack 中相应各域含义是相同的，同时 requeue 域的值应该被忽略。通过 nack 一或多条消息，broker 表明自身无法对相应消息完成处理，并拒绝为这些消息的处理负责。在这种情况下，client 可以选择将消息 re-publish 。

      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 。

2. 消息在什么时候确认

broker 将在下面的情况中对消息进行 confirm （这里的confirm是会触发回调方法，此时可能是ack，也可能是nack）：

a、broker 发现当前消息无法被路由到指定的 queues 中（如果设置了 mandatory 属性，则 broker 会先发送 basic.return）

（如果路由失败，失败通知的原因是NO\_ROUTE，但是发布者确认模式触发的是ack方法）

b、非持久属性的消息到达了其所应该到达的所有 queue 中（和镜像 queue 中）

c、持久消息到达了其所应该到达的所有 queue 中（和镜像 queue 中），并被持久化到了磁盘（被 fsync）

d、持久消息从其所在的所有 queue 中被 consume 了（如果必要则会被 acknowledge）

broker 会丢失持久化消息，如果 broker 在将上述消息写入磁盘前异常。在一定条件下，这种情况会导致 broker 以一种奇怪的方式运行。例如，考虑下述情景：

   1.  一个 client 将持久消息 publish 到持久 queue 中

   2.  另一个 client 从 queue 中 consume 消息（注意：该消息具有持久属性，并且 queue 是持久化的），当尚未对其进行 ack

   3.  broker 异常重启

   4.  client 重连并开始 consume 消息

   在上述情景下，client 有理由认为消息需要被（broker）重新 deliver 。但这并非事实：重启（有可能）会令 broker 丢失消息。为了确保持久性，client 应该使用 confirm 机制。如果 publisher 使用的 channel 被设置为 confirm 模式，publisher 将不会收到已丢失消息的 ack（这是因为 consumer 没有对消息进行 ack ，同时该消息也未被写入磁盘）。

## 死信交换器和死信队列

主生产者发送三条消息（aaa，bbb，ccc）到topic交换器，分别绑定的路由键是（info，error，warning）

主队列绑定topic交换器，路由键绑定为“#”，即监听所有级别日志，同时也绑定了死信交换器

消费者从队列获取消息（aaa，bbb，ccc），消费掉（aaa）,拒绝了（bbb，ccc）

拒绝的消息（bbb，ccc）转发到了死信交换器

此时（bbb，ccc）消息的路由键有两种情况：

1. 在声明死信交换器的时候，没有配置死信路由键（x-dead-letter-routing-key），

那么（bbb，ccc）沿用之前的路由键（error，warning）；

1. 在声明死信交换器的时候，配置死信路由键（x-dead-letter-routing-key），

args.put("x-dead-letter-routing-key", "all")，那么（bbb，ccc）的路由键都是 “all”

此时（bbb，ccc）消息的发送规则由死信队列和死信交换器的绑定有关，跟普通队列和普通交换器绑定一样。

## 交换器和队列autoDelete=true

1、队列的autoDelete=true：

当此队列没有消费者与之关联的时候，或者此队列没有消息，或者此队列的消息过期并且被消费者来确认过，则自动删除队列。

2、交换器的autoDelete=true：

启动生产者和消费者，创建自动删除的队列和交换器，如果关闭消费者，会删除自动删除的队列，同时也会删除自动删除的交换器，此时生产者再发送消息会报错，没有交换器；

只启动生产者，创建自动删除的队列和交换器，生产者发送消息不会报错，只有交换器存在，没有队列。

## 队列exclusive=true

此队列只允许绑定一个消费者，如果A和B两个消费者同时绑定，先启动A可以，再启动B报错，或者先启动B可以，再启动A报错。

## 队列durable=true

队列持久化的前提是：队列autoDelete=false，否则，在没有消费者的情况下，还是会删除队列。

即使队列持久化，但重启服务的时候还是会丢失没有持久化的消息。