前面几篇记录了收发消息的demo,今天记录下关于 消息确认方面的 问题.

下面是几个问题:

1.为什么要进行消息确认?

2.rabbitmq消息确认 机制是什么样的?

3.发送方如何确认消息发送成功?什么样才算发送成功?

4.消费方如何告知rabbitmq消息消费成功或失败?

5.使用spring的代码示例

1.为什么要进行消息确认?

经常会听到丢消息的字眼, 对于前面的demo来说,就存在丢消息的隐患.

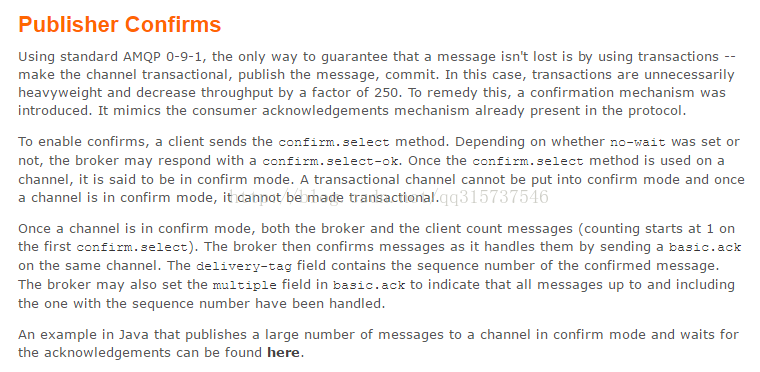
发送者没法确认是否发送成功,消费者处理失败也无法反馈.

没有消息确认机制,就会出现消息莫名其妙的没了,也不知道什么情况.

2.rabbitmq消息确认 机制是什么样的?

首先看官网对消息确认的介绍<http://www.rabbitmq.com/confirms.html>

网上会有很多总结的博客(包括现在看的),很多就是对官网的翻译.所以看资料首先要去官网看看,这很关键.



看上图官网的介绍.唯一保证消息不丢失的是使用事务,但是性能太差,作为补偿,有了消息确认机制.

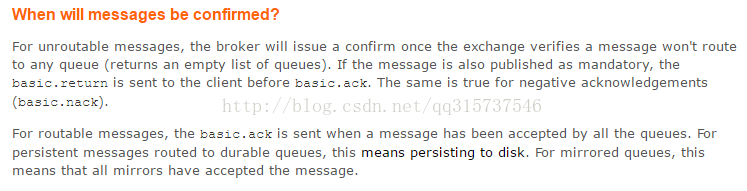
并说明了开启方法,以及和事务模式不共存.

还写了一个例子,但是点进去那个链接已经失效了,新版的源码上也没有这个例子,我找了最近一版是3.6.7上面还有.

[点这里看官方的例子](https://github.com/rabbitmq/rabbitmq-java-client/blob/rabbitmq_v3_6_7_milestone2/src/test/java/com/rabbitmq/examples/ConfirmDontLoseMessages.java)

3.发送的消息什么样才算成功或失败? 如何确认?

判断消息成功或失败,其实就是看进行消息确认的时机,因为成功或失败后就会把结果告诉发送方.还是看官方解释:



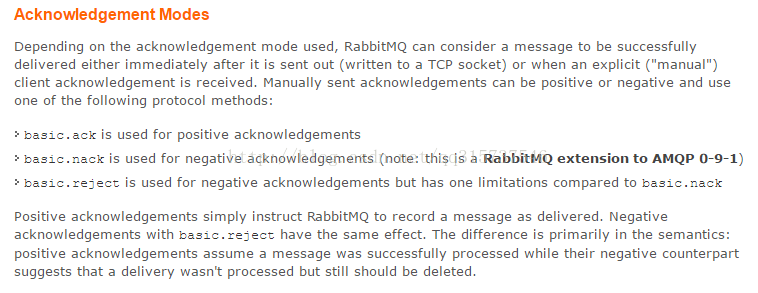
意思如下:

确认消息不能路由时(exchange确认不能路由到任何queue),进行确认操作(确认失败).如果发送方设置了mandatory模式,则会先调用basic.return方法.

消息可以路由时,当需要发送的队列都发送成功后,进行消息确认.对于持久化的队列,意味着已经写入磁盘,对于镜像队列,意味着所有镜像都接受成功.

至于如何确认的问题,上面已经写了 basic.ack方法

4.消费方如何告知rabbitmq消息消费成功或失败?



如图可知,根据消费方不同的确认模式,确认时机也不同.

自动确认会在消息发送给消费者后立即确认,如果手动则当消费者调用ack,nack,reject几种方法时进行确认.

一般会设置手动模式,业务失败后可以进行一些操作.

5.使用spring的代码示例

下面是一个使用spring整合的代码示例:

首先是rabbitmq的配置文件:

**[html]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

**<?xml** version="1.0" encoding="UTF-8"**?>**

**<beans** xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rabbit="http://www.springframework.org/schema/rabbit"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans.xsd

    http://www.springframework.org/schema/rabbit

    http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd"**>**

    <!-- spring-rabbit.xsd的版本要注意,很1.4以前很多功能都没有,要用跟jar包匹配的版本 -->

    **<bean** id="jsonMessageConverter" class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter" **/>**

    **<rabbit:connection-factory**

        id="connectionFactory"

        host="${rabbit.host}"

        port="${rabbit.port}"

        username="${rabbit.username}"

        password="${rabbit.password}"

        publisher-confirms="true"

    **/>**

    **<rabbit:admin** connection-factory="connectionFactory" **/>**

    <!-- 给模板指定转换器 **connection-factory 中** publisher-confirms="true" 才能回调  -->

<!-- mandatory必须设置true,return callback才生效 -->

    **<rabbit:template** id="amqpTemplate"   connection-factory="connectionFactory"

        confirm-callback="confirmCallBackListener"

        return-callback="returnCallBackListener"

        mandatory="true"

    **/>**

    **<rabbit:queue** name="CONFIRM\_TEST" **/>**

    **<rabbit:direct-exchange** name="DIRECT\_EX" id="DIRECT\_EX" **>**

        **<rabbit:bindings>**

            **<rabbit:binding** queue="CONFIRM\_TEST" **/>**

        **</rabbit:bindings>**

    **</rabbit:direct-exchange>**

    <!-- 配置consumer, 监听的类和queue的对应关系 -->

    **<rabbit:listener-container**

        connection-factory="connectionFactory" acknowledge="manual" **>**

        **<rabbit:listener** queues="CONFIRM\_TEST" ref="receiveConfirmTestListener" **/>**

    **</rabbit:listener-container>**

**</beans>**

然后发送方:

**[java]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

**import** org.springframework.amqp.core.AmqpTemplate;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

@Service("publishService")

**public** **class** PublishService {

    @Autowired

    **private** AmqpTemplate amqpTemplate;

    **public** **void** send(String exchange, String routingKey, Object message) {

        amqpTemplate.convertAndSend(exchange, routingKey, message);

    }

}

消费方:

**[java]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

**import** org.springframework.amqp.core.Message;

**import** org.springframework.amqp.rabbit.core.ChannelAwareMessageListener;

**import** org.springframework.stereotype.Service;

**import** com.rabbitmq.client.Channel;

@Service("receiveConfirmTestListener")

**public** **class** ReceiveConfirmTestListener **implements** ChannelAwareMessageListener {

    @Override

    **public** **void** onMessage(Message message, Channel channel) **throws** Exception {

        **try**{

            System.out.println("consumer--:"+message.getMessageProperties()+":"+**new** String(message.getBody()));

            channel.basicAck(message.getMessageProperties().getDeliveryTag(), **false**);

        }**catch**(Exception e){

            e.printStackTrace();//TODO 业务处理

            channel.basicNack(message.getMessageProperties().getDeliveryTag(), **false**,**false**);

        }

    }

}

确认后回调:

**[java]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

**import** org.springframework.amqp.rabbit.core.RabbitTemplate.ConfirmCallback;

**import** org.springframework.amqp.rabbit.support.CorrelationData;

**import** org.springframework.stereotype.Service;

@Service("confirmCallBackListener")

**public** **class** ConfirmCallBackListener **implements** ConfirmCallback{

    @Override

    **public** **void** confirm(CorrelationData correlationData, **boolean** ack, String cause) {

        System.out.println("confirm--:correlationData:"+correlationData+",ack:"+ack+",cause:"+cause);

    }

}

失败后return回调:

**[java]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

**import** org.springframework.amqp.core.Message;

**import** org.springframework.amqp.rabbit.core.RabbitTemplate.ReturnCallback;

**import** org.springframework.stereotype.Service;

@Service("returnCallBackListener")

**public** **class** ReturnCallBackListener **implements** ReturnCallback{

    @Override

    **public** **void** returnedMessage(Message message, **int** replyCode, String replyText, String exchange, String routingKey) {

        System.out.println("return--message:"+**new** String(message.getBody())+",replyCode:"+replyCode+",replyText:"+replyText+",exchange:"+exchange+",routingKey:"+routingKey);

    }

}

测试类:

**[java]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.test.context.ContextConfiguration;

**import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

**import** com.dingcheng.confirms.publish.PublishService;

@RunWith(SpringJUnit4ClassRunner.**class**)

@ContextConfiguration(locations = {"classpath:application-context.xml"})

**public** **class** TestConfirm {

    @Autowired

    **private** PublishService publishService;

    **private** **static** String exChange = "DIRECT\_EX";

    @Test

    **public** **void** test1() **throws** InterruptedException{

        String message = "currentTime:"+System.currentTimeMillis();

        System.out.println("test1---message:"+message);

        //exchange,queue 都正确,confirm被回调, ack=true

        publishService.send(exChange,"CONFIRM\_TEST",message);

        Thread.sleep(1000);

    }

    @Test

    **public** **void** test2() **throws** InterruptedException{

        String message = "currentTime:"+System.currentTimeMillis();

        System.out.println("test2---message:"+message);

        //exchange 错误,queue 正确,confirm被回调, ack=false

        publishService.send(exChange+"NO","CONFIRM\_TEST",message);

        Thread.sleep(1000);

    }

    @Test

    **public** **void** test3() **throws** InterruptedException{

        String message = "currentTime:"+System.currentTimeMillis();

        System.out.println("test3---message:"+message);

        //exchange 正确,queue 错误 ,confirm被回调, ack=true; return被回调 replyText:NO\_ROUTE

        publishService.send(exChange,"",message);

//        Thread.sleep(1000);

    }

    @Test

    **public** **void** test4() **throws** InterruptedException{

        String message = "currentTime:"+System.currentTimeMillis();

        System.out.println("test4---message:"+message);

        //exchange 错误,queue 错误,confirm被回调, ack=false

        publishService.send(exChange+"NO","CONFIRM\_TEST",message);

        Thread.sleep(1000);

    }

}

测试结果:

**[html]** [view plain](http://blog.csdn.net/qq315737546/article/details/54176560#) [copy](http://blog.csdn.net/qq315737546/article/details/54176560#)

 [print?](http://blog.csdn.net/qq315737546/article/details/54176560#)

test1---message:currentTime:1483786948506

test2---message:currentTime:1483786948532

consumer--:MessageProperties [headers={spring\_return\_correlation=445bc7ca-a5bd-47e2-8ba3-f0448420e441}, timestamp=null, messageId=null, userId=null, appId=null, clusterId=null, type=null, correlationId=null, replyTo=null, contentType=text/plain, contentEncoding=UTF-8, contentLength=0, deliveryMode=PERSISTENT, expiration=null, priority=0, redelivered=false, receivedExchange=DIRECT\_EX, receivedRoutingKey=CONFIRM\_TEST, deliveryTag=1, messageCount=0]:currentTime:1483786948506

test3---message:currentTime:1483786948536

confirm--:correlationData:null,ack:false,cause:channel error; protocol method: #method**<channel.close>**(reply-code=404, reply-text=NOT\_FOUND - no exchange 'DIRECT\_EXNO' in vhost '/', class-id=60, method-id=40)

confirm--:correlationData:null,ack:false,cause:Channel closed by application

[ERROR] 2017-01-07 19:02:28 org.springframework.amqp.rabbit.connection.CachingConnectionFactory.shutdownCompleted(CachingConnectionFactory.java:281):--**>** Channel shutdown: channel error; protocol method: #method**<channel.close>**(reply-code=404, reply-text=NOT\_FOUND - no exchange 'DIRECT\_EXNO' in vhost '/', class-id=60, method-id=40)

 return--message:currentTime:1483786948536,replyCode:312,replyText:NO\_ROUTE,exchange:DIRECT\_EX,routingKey:

confirm--:correlationData:null,ack:true,cause:null

test4---message:currentTime:1483786948546

confirm--:correlationData:null,ack:false,cause:channel error; protocol method: #method**<channel.close>**(reply-code=404, reply-text=NOT\_FOUND - no exchange 'DIRECT\_EXNO' in vhost '/', class-id=60, method-id=40)

[ERROR] 2017-01-07 19:02:28 org.springframework.amqp.rabbit.connection.CachingConnectionFactory.shutdownCompleted(CachingConnectionFactory.java:281):--**>** Channel shutdown: channel error; protocol method: #method**<channel.close>**(reply-code=404, reply-text=NOT\_FOUND - no exchange 'DIRECT\_EXNO' in vhost '/', class-id=60, method-id=40)

**代码和配置里面,已经都有注释,就不在多说明了.(callback是异步的,所以测试中sleep1秒钟等待下)**

**总结下就是:**

**如果消息没有到exchange,则confirm回调,ack=false**

**如果消息到达exchange,则confirm回调,ack=true**

**exchange到queue成功,则不回调return**

**exchange到queue失败,则回调return(需设置mandatory=true,否则不回回调,消息就丢了)**

**注意：只有加上 mandatory，才能做到和事物一样的安全。**

备注:需要说明,spring-rabbit和原生的rabbit-client ,表现是不一样的.

测试的时候,原生的client,exchange错误的话,直接就报错了,是不会到confirmListener和returnListener的

**如下介绍事物与confirm**

在交换机简介一篇中，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。但是上一篇所有的代码都存在如下两个问题：

1、服务器宕机，消息将丢失。

2、发送的消息如果没有正确路由到队列，例如生产者先运行消费者未运行的情况，那么消息将被丢弃，且生产者没有收到任何反馈。

下面的介绍将会解决这些问题。

一、消息持久化

以前我的消息都是临时消息，服务器宕机重启后，没有被处理的消息将会被丢弃掉，通过设置交换机持久化、队列持久化、消息持久化可以解决这一个问题。

生产者：

//声明持久化交换机，类型为direct
boolean durable=true;
channel.exchangeDeclare(EXCHANGE\_NAME, "direct", durable, false, null);
String message="hello world";
//发送消息，并设置为持久化
channel.basicPublish(EXCHANGE\_NAME, "blue", MessageProperties.PERSISTENT\_TEXT\_PLAIN, message.getBytes());

//声明持久化交换机
boolean durable=true;
channel.exchangeDeclare(EXCHANGE\_NAME, "direct", durable, false, null);
//声明持久化队列
channel.queueDeclare(queueName, durable, false, false, null);
channel.queueBind(queueName, EXCHANGE\_NAME, "blue");
System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");
QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(queueName, true, consumer);
while (true)
{
QueueingConsumer.Delivery delivery = consumer.nextDelivery();
String message = new String(delivery.getBody());
String routingKey = delivery.getEnvelope().getRoutingKey();
System.out.println(" [x] Received routingKey = " + routingKey + ",msg = " + message + ".");
}

即使交换机和队列都是持久化的，消息是否持久化取决于发送方发送消息时指定的properties，MessageProperties类中有一些常用的Properties对象，例如持久化消息MessageProperties.PERSISTENT\_TEXT\_PLAIN。

二、事务和消息确认

消息持久化，可以保证已经发送到队列中的消息被持久化到磁盘上，服务器重启后可以从磁盘恢复这些消息。那么对于正在发送还没有成功的消息呢？两种解决方案：

1、事务

RabbitMQ中与事务机制有关的方法有三个，分别是Channel里面的txSelect()，txCommit()以及txRollback()，txSelect用于将当前Channel设置成是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，则消息一定是到达broker了，如果在txCommit执行之前broker异常奔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。

channel.txSelect();
try{
channel.basicPublish(EXCHANGE\_NAME, "blue", MessageProperties.PERSISTENT\_TEXT\_PLAIN, message.getBytes());
System.out.println("commit");
channel.txCommit();
}catch(Exception e){
System.out.println("rollback");
channel.txRollback();
}

2、PublisherConfirm

Publisher Confirm机制（又称为Confirms或Publisher Acknowledgements）是作为解决事务机制性能开销大（导致吞吐量下降）而提出的另外一种保证消息不会丢失的方式。

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 。

Confirm机制在性能上要比事务优越很多。但是Confirm机制，无法进行回滚，就是一旦服务器崩溃，生产者无法得到Confirm信息，生产者其实本身也不知道该消息吃否已经被持久化，只有继续重发来保证消息不丢失，但是如果原先已经持久化的消息，并不会被回滚，这样队列中就会存在两条相同的消息，系统需要支持去重。

Channel对象提供的ConfirmListener()回调方法只包含deliveryTag（当前Chanel发出的消息序号），我们需要自己为每一个Channel维护一个unconfirm的消息序号集合，每publish一条数据，集合中元素加1，每回调一次handleAck方法，unconfirm集合删掉相应的一条（multiple=false）或多条（multiple=true）记录。从程序运行效率上看，这个unconfirm集合最好采用有序集合SortedSet存储结构。

SortedSet<Long> confirmSet = Collections.synchronizedSortedSet(new TreeSet<Long>());
channel.confirmSelect();
channel.addConfirmListener(new ConfirmListener() {
public void handleAck(long deliveryTag, boolean multiple) throws IOException {
if (multiple) {
confirmSet.headSet(deliveryTag + 1).clear();
} else {
confirmSet.remove(deliveryTag);
}
}
public void handleNack(long deliveryTag, boolean multiple) throws IOException {
System.out.println("Nack, SeqNo: " + deliveryTag + ", multiple: " + multiple);
if (multiple) {
confirmSet.headSet(deliveryTag + 1).clear();
} else {
confirmSet.remove(deliveryTag);
}
}
});
while (true) {
long nextSeqNo = channel.getNextPublishSeqNo();
channel.basicPublish(ConfirmConfig.exchangeName, ConfirmConfig.routingKey, MessageProperties.PERSISTENT\_TEXT\_PLAIN, ConfirmConfig.msg\_10B.getBytes());
confirmSet.add(nextSeqNo);

三、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。

关键代码：

//声明持久化交换机，类型为direct
boolean durable=true;
channel.exchangeDeclare(EXCHANGE\_NAME, "direct", durable, false, null);
//添加回调监听，用于处理basic.return命令返回的消息
channel.addReturnListener(new ReturnListener() {
public void handleReturn(int replyCode,String replyText,String exchange,String routingKey,AMQP.BasicProperties properties,byte[] body)
throws IOException{
System.out.println("handleReturn");
System.out.println("replyCode:"+replyCode);
System.out.println("replyText:"+replyText);
System.out.println("messagebody:"+new String(body));
}
});
String message="hello world";
//设置mandatory为true
boolean mandatory=true;
channel.basicPublish(EXCHANGE\_NAME, "blue", mandatory, false, MessageProperties.PERSISTENT\_TEXT\_PLAIN, message.getBytes());

运行结果：

[x] Sent ‘hello world’

handleReturn

replyCode:312

replyText:NO\_ROUTE

messagebody:hello world