RabbitMQ的消费与发送的消息确认

<https://www.jianshu.com/p/2c5eebfd0e95>

**消息接收确认**

**消息消费者如何通知 Rabbit 消息消费成功？**

* 消息**通过 ACK 确认是否被正确接收**，每个 Message 都要被确认（acknowledged），可以手动去 ACK 或自动 ACK
* 自动确认会在消息发送给消费者后立即确认，但存在丢失消息的可能，如果消费端消费逻辑抛出异常，也就是消费端没有处理成功这条消息，那么就相当于丢失了消息
* 如果消息已经被处理，但后续代码抛出异常，使用 Spring 进行管理的话消费端业务逻辑会进行回滚，这也同样造成了实际意义的消息丢失
* 如果手动确认则当消费者调用 ack、nack、reject 几种方法进行确认，手动确认可以在业务失败后进行一些操作，如果消息未被 ACK 则会发送到下一个消费者
* **如果某个服务忘记 ACK 了，则 RabbitMQ 不会再发送数据给它，因为 RabbitMQ 认为该服务的处理能力有限**
* ACK 机制还可以起到限流作用，比如在接收到某条消息时休眠几秒钟
* 消息确认模式有：
  + AcknowledgeMode.NONE：自动确认
  + AcknowledgeMode.AUTO：根据情况确认
  + AcknowledgeMode.MANUAL：手动确认

### 确认消息（局部方法处理消息）

* 默认情况下消息消费者是自动 ack （确认）消息的，如果要手动 ack（确认）则需要修改确认模式为 manual

spring: rabbitmq: listener: simple: acknowledge-mode: manual

* 或在 RabbitListenerContainerFactory 中进行开启手动 ack

@Bean public RabbitListenerContainerFactory<?> rabbitListenerContainerFactory(ConnectionFactory connectionFactory){ SimpleRabbitListenerContainerFactory factory = new SimpleRabbitListenerContainerFactory(); factory.setConnectionFactory(connectionFactory); factory.setMessageConverter(new Jackson2JsonMessageConverter()); factory.setAcknowledgeMode(AcknowledgeMode.MANUAL); //开启手动 ack return factory; }

* 确认消息

@RabbitHandler public void processMessage2(String message,Channel channel,@Header(AmqpHeaders.DELIVERY\_TAG) long tag) { System.out.println(message); try { channel.basicAck(tag,false); // 确认消息 } catch (IOException e) { e.printStackTrace(); } }

* 需要注意的 basicAck 方法需要传递两个参数
  + **deliveryTag（唯一标识 ID）**：当一个消费者向 RabbitMQ 注册后，会建立起一个 Channel ，RabbitMQ 会用 basic.deliver 方法向消费者推送消息，这个方法携带了一个 delivery tag， **它代表了 RabbitMQ 向该 Channel 投递的这条消息的唯一标识 ID**，是一个单调递增的正整数，delivery tag 的范围仅限于 Channel
  + **multiple**：为了减少网络流量，手动确认可以被批处理，**当该参数为 true 时，则可以一次性确认 delivery\_tag 小于等于传入值的所有消息**

**手动否认、拒绝消息**

* 发送一个 header 中包含 error 属性的消息

hducA.png

* 消费者获取消息时检查到头部包含 error 则 nack 消息

@RabbitHandler public void processMessage2(String message, Channel channel,@Headers Map<String,Object> map) { System.out.println(message); if (map.get("error")!= null){ System.out.println("错误的消息"); try { channel.basicNack((Long)map.get(AmqpHeaders.DELIVERY\_TAG),false,true); //否认消息 return; } catch (IOException e) { e.printStackTrace(); } } try { channel.basicAck((Long)map.get(AmqpHeaders.DELIVERY\_TAG),false); //确认消息 } catch (IOException e) { e.printStackTrace(); } }

* 此时控制台重复打印，说明该消息被 nack 后一直重新入队列然后一直重新消费

hello 错误的消息 hello 错误的消息 hello 错误的消息 hello 错误的消息

* 也可以拒绝该消息，消息会被丢弃，不会重回队列

channel.basicReject((Long)map.get(AmqpHeaders.DELIVERY\_TAG),false); //拒绝消息

**确认消息（全局处理消息）**

* 自动确认涉及到一个问题就是如果在处理消息的时候抛出异常，消息处理失败，但是因为自动确认而导致 Rabbit 将该消息删除了，造成消息丢失

@Bean public SimpleMessageListenerContainer messageListenerContainer(ConnectionFactory connectionFactory){ SimpleMessageListenerContainer container = new SimpleMessageListenerContainer(); container.setConnectionFactory(connectionFactory); container.setQueueNames("consumer\_queue"); // 监听的队列 container.setAcknowledgeMode(AcknowledgeMode.NONE); // NONE 代表自动确认 container.setMessageListener((MessageListener) message -> { //消息监听处理 System.out.println("====接收到消息====="); System.out.println(new String(message.getBody())); //相当于自己的一些消费逻辑抛错误 throw new NullPointerException("consumer fail"); }); return container; }

* 手动确认消息

@Bean public SimpleMessageListenerContainer messageListenerContainer(ConnectionFactory connectionFactory){ SimpleMessageListenerContainer container = new SimpleMessageListenerContainer(); container.setConnectionFactory(connectionFactory); container.setQueueNames("consumer\_queue"); // 监听的队列 container.setAcknowledgeMode(AcknowledgeMode.MANUAL); // 手动确认 container.setMessageListener((ChannelAwareMessageListener) (message, channel) -> { //消息处理 System.out.println("====接收到消息====="); System.out.println(new String(message.getBody())); if(message.getMessageProperties().getHeaders().get("error") == null){ channel.basicAck(message.getMessageProperties().getDeliveryTag(),false); System.out.println("消息已经确认"); }else { //channel.basicNack(message.getMessageProperties().getDeliveryTag(),false,false); channel.basicReject(message.getMessageProperties().getDeliveryTag(),false); System.out.println("消息拒绝"); } }); return container; }

* AcknowledgeMode 除了 NONE 和 MANUAL 之外还有 AUTO ，它会根据方法的执行情况来决定是否确认还是拒绝（是否重新入queue）
  + 如果消息成功被消费（成功的意思是在消费的过程中没有抛出异常），则自动确认
  + 当抛出 AmqpRejectAndDontRequeueException 异常的时候，则消息会被拒绝，且 requeue = false（不重新入队列）
  + 当抛出 ImmediateAcknowledgeAmqpException 异常，则消费者会被确认
  + 其他的异常，则消息会被拒绝，且 requeue = true（如果此时只有一个消费者监听该队列，则有发生死循环的风险，多消费端也会造成资源的极大浪费，这个在开发过程中一定要避免的）。可以通过 setDefaultRequeueRejected（默认是true）去设置

@Bean public SimpleMessageListenerContainer messageListenerContainer(ConnectionFactory connectionFactory){ SimpleMessageListenerContainer container = new SimpleMessageListenerContainer(); container.setConnectionFactory(connectionFactory); container.setQueueNames("consumer\_queue"); // 监听的队列 container.setAcknowledgeMode(AcknowledgeMode.AUTO); // 根据情况确认消息 container.setMessageListener((MessageListener) (message) -> { System.out.println("====接收到消息====="); System.out.println(new String(message.getBody())); //抛出NullPointerException异常则重新入队列 //throw new NullPointerException("消息消费失败"); //当抛出的异常是AmqpRejectAndDontRequeueException异常的时候，则消息会被拒绝，且requeue=false //throw new AmqpRejectAndDontRequeueException("消息消费失败"); //当抛出ImmediateAcknowledgeAmqpException异常，则消费者会被确认 throw new ImmediateAcknowledgeAmqpException("消息消费失败"); }); return container; }

**消息可靠总结**

* 持久化
  + exchange要持久化
  + queue要持久化
  + message要持久化
* 消息确认
  + 启动消费返回（@ReturnList注解，生产者就可以知道哪些消息没有发出去）
  + 生产者和Server（broker）之间的消息确认
  + 消费者和Server（broker）之间的消息确认

补充说明：

<https://www.jianshu.com/p/19e0927315da>