课程目标

1、消息可靠性投递业务场景分析及解决方案；

2、RabbitMQ 在项目中的应用；

3、生产环境是如何应用 MQ 的；

4、RabbitMQ 经典面试题分析；

1、可靠性投递

上次课讲完之后，很多同学有疑问？我呢也分别作了解答。

非常好，在自己思考了。大部分问题呢，也是我们今天要讲的内容。

在使用 MQ 实现异步通信的过程中，有消息丢了怎么办？或者 MQ 消息重复了怎么办？死信队列满了怎么办？既然消息都产生了，为什么不消费？

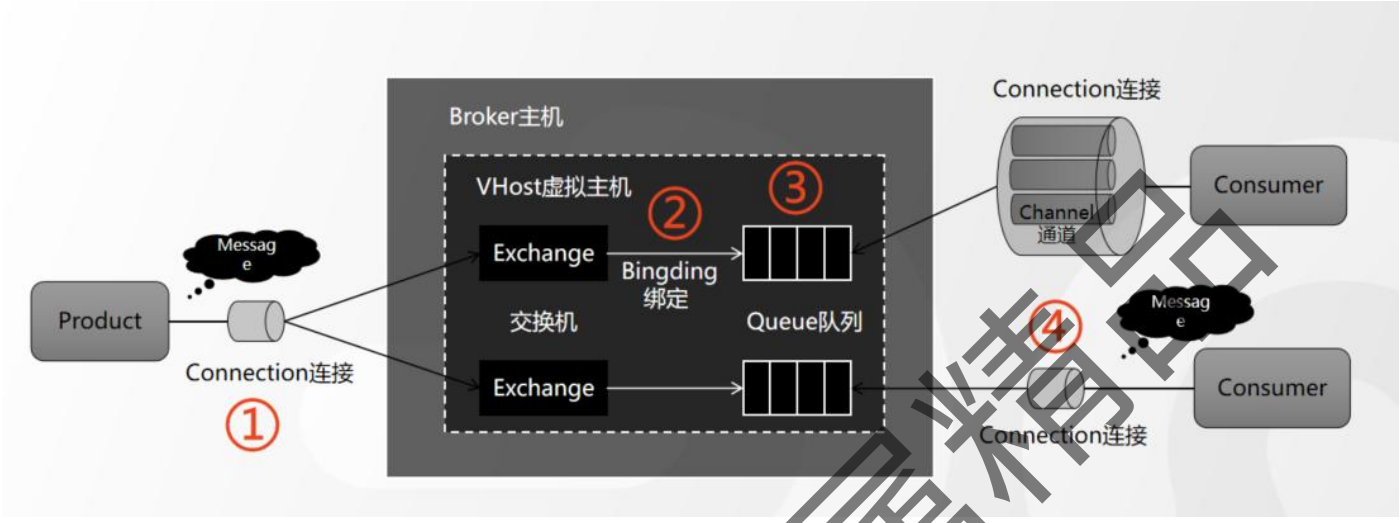
这个就是我们今天讲的 RabbitMQ 的可靠性投递。当然，RabbitMQ 在设计的时候其实就考虑了这一点，提供了很多保证消息可靠投递的机制。这个可以说是 RabbitMQ 比较突出的一个特性。

这里要先跟大家交个底，可靠性只是问题的一个方面，发送消息的效率同样是我们需要考虑的问题，而这两个因素是无法兼得的。如果在发送消息的每一个环节都采取相关措施来保证可靠性，势必会对消息的收发效率造成影响。

所以，这些手段大家都可以用，但并不是一定要用。

例如：一些业务实时一致性要求不是特别高的场合，可以牺牲一些可靠性来换取效率。比如发送通知或者记录日志的这种场景，如果用户没有收到通知，不会造成很大的影响，就不需要严格保证所有的消息都发送成功。如果失败了，只要再次发送就可以了。

下面，我们会根据之前学习过的 RabbitMQ 的工作模型，来分析一下 RabbitMQ 为我们提供了哪些可靠性措施。



在我们使用 RabbitMQ 收发消息的时候，有几个主要环节：

① 代表消息从生产者发送到 Broker

生产者把消息发到 Broker 之后，怎么知道自己的消息有没有被 Broker 成功接收？如果 Broker 不给应答，生产者不断地发送，那有可能是一厢情愿，消息全部进了黑洞。

② 代表消息从 Exchange 路由到 Queue

Exchange 是一个绑定列表，它的职责是分发消息。如果它没有办法履行它的职责怎么办？也就是说，找不到队列或者找不到正确的队列，怎么处理？

③ 代表消息在 Queue 中存储

队列有自己的数据库（Mnesia），它是真正用来存储消息的。如果还没有消费者来消费，那么消息要一直存储在队列里面。你的信件放在邮局，如果邮局内部出了问题，比如起火，信件肯定会丢失。怎么保证消息在队列稳定地存储呢？

④ 代表消费者订阅 Queue 并消费消息

队列的特性是什么？FIFO。队列里面的消息是一条一条的投递的，也就是说，只有上一条消息被消费者接收以后，才能把这一条消息从数据库删掉，继续投递下

一条消息。

或者反过来说，如果消费者不签收，我是不能去派送下一个快件的，总不能丢在门口就跑吧?

问题来了，Broker（快递总部）怎么知道消费者已经接收了消息呢?

下面我们就从这四个环节入手，分析如何保证消息的可靠性。

1.1 消息发送到 RabbitMQ 服务器

第一个环节是生产者发送消息到 Broker。先来说一下什么情况下会发送消息失败?

可能因为网络连接或者 Broker 的问题（比如硬盘故障、硬盘写满了）导致消息发送失败，生产者不能确定 Broker 有没有正确的接收。

如果我们去设计，肯定要给生产者发送消息的接口一个应答，生产者才可以明确知道消息有没有发送成功。

在 RabbitMQ 在服务端提供了两种**确认机制**，也就是在生产者发送消息给 RabbitMQ 的服务端的时候，服务端会通过某种方式返回一个应答，只要生产者收到了这个应答，就知道消息发送成功了。

第一种是 Transaction（事务）模式，第二种 Confirm（确认）模式。

1.1.1 Transaction（事务）模式

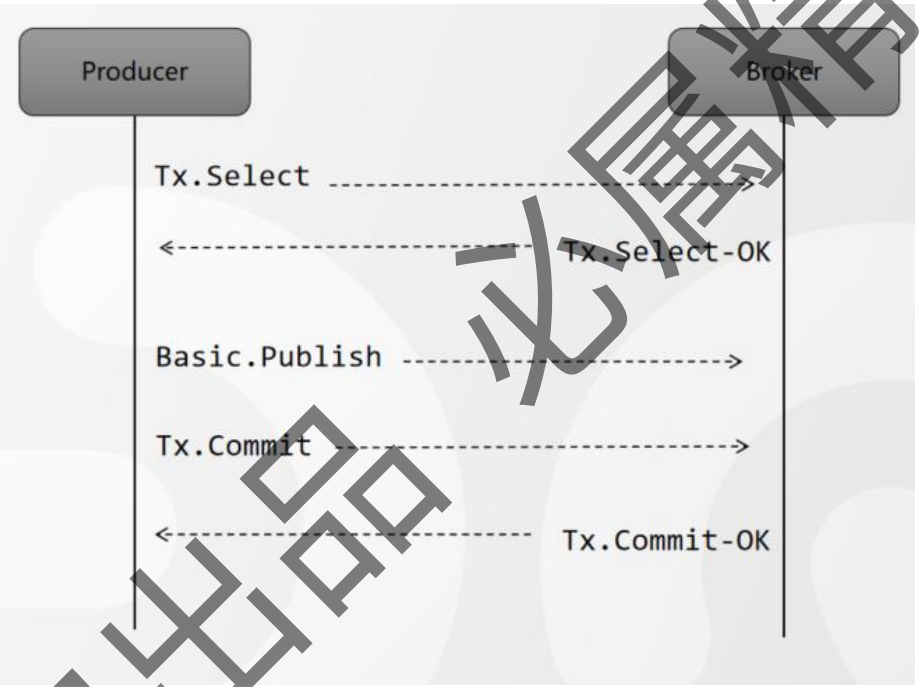
事务模式怎么使用呢？它在创建 channel 的时候，可以把信道设置成事务模式，然后就可以发布消息给 RabbitMQ 了。如果 channel.txCommit();的方法调用成功，就说明事务提交成功，则消息一定到达了 RabbitMQ 中。

代码位置：`com.gupaoedu.vip.mq.rabbit.javaapi.transaction`

```
try {
   channel.txSelect();
```

```
    // 发送消息
    // String exchange, String routingKey, BasicProperties props, byte[] body
    channel.basicPublish("", QUEUE_NAME, null, (msg).getBytes());
    // int i =1/0;
    channel.txCommit();
    System.out.println("消息发送成功");
} catch (Exception e) {
    channel.txRollback();
    System.out.println("消息已经回滚");
}
```

如果在事务提交执行之前由于 RabbitMQ 异常崩溃或者其他原因抛出异常，这个时候我们便可以将其捕获，进而通过执行 channel.txRollback()方法来实现事务回滚。



AMQP 协议抓包示意

在事务模式里面，只有收到了服务端的 Commit-OK 的指令，才能提交成功。所以可以解决生产者和服务端确认的问题。但是事务模式有一个缺点，它是阻塞的，一条消息没有发送完毕，不能发送下一条消息，它会榨干 RabbitMQ 服务器的性能。所以不建议大家在生产环境使用。

Spring Boot 中的设置：

```
rabbitTemplate.setChannelTransacted(true);
```

那么有没有其他可以保证消息被 Broker 接收，但是又不大量消耗性能的方式呢？

这个就是第二种模式，叫做确认（Confirm）模式。

1.1.2 Confirm（确认）模式

确认模式有三种，一种是普通确认模式。

在生产者这边通过调用 channel.confirmSelect()方法将信道设置为 Confirm 模式，然后发送消息。一旦消息被投递到交换机之后（跟是否路由到队列没有关系），RabbitMQ 就会发送一个确认（Basic.Ack）给生产者，也就是调用 channel.waitForConfirms()返回 true，这样生产者就知道消息被服务端接收了。

如果网络错误，会抛出连接异常。如果交换机不存在，会抛出 404 错误。

```
// 开启发送方确认模式
channel.confirmSelect();

channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
// 普通 Confirm，发送一条，确认一条
if (channel.waitForConfirms()) {
   System.out.println("消息发送成功" );
}
```

这种发送 1 条确认 1 条的方式，发送消息的效率不太高，所以我们还有一种批量确认的方式。批量确认，就是在开启 Confirm 模式后，先发送一批消息。

```
try {
   channel.confirmSelect();
   for (int i = 0; i < 5; i++) {
       // 发送消息
       // String exchange, String routingKey, BasicProperties props, byte[] body
       channel.basicPublish("", QUEUE_NAME, null, (msg +"-"+ i).getBytes());
  }
   // 批量确认结果，ACK 如果是 Multiple=True，代表 ACK 里面的 Delivery-Tag 之前的消息都被确认了
   // 比如 5 条消息可能只收到 1 个 ACK，也可能收到 2 个（抓包才看得到）
   // 直到所有信息都发布，只要有一个未被 Broker 确认就会 IOException
   channel.waitForConfirmsOrDie();
   System.out.println("消息发送完毕，批量确认成功");
} catch (Exception e) {
   // 发生异常，可能需要对所有消息进行重发
   e.printStackTrace();
}
```

只要 channel.waitForConfirmsOrDie();方法没有抛出异常，就代表消息都被服务端接收了。

批量确认的方式比单条确认的方式效率要高，但是也有两个问题：

第一个就是批量的数量的确定。对于不同的业务，到底发送多少条消息确认一次？数量太少，效率提升不上去。数量多的话，又会带来另一个问题，比如我们发 1000 条消息才确认一次，如果前面 999 条消息都被服务端接收了，如果第 1000 条消息被拒绝了，那么前面所有的消息都要重发。

有没有一种方式，可以一边发送一边确认的呢？这个就是异步确认模式。

异步确认模式需要添加一个 ConfirmListener，并且用一个 SortedSet 来维护一个批次中没有被确认的消息。

```
// 用来维护未确认消息的 deliveryTag
final SortedSet<Long> confirmSet = Collections.synchronizedSortedSet(new
TreeSet<Long>());
```

```
// 这里不会打印所有响应的 ACK；ACK 可能有多个，有可能一次确认多条，也有可能一次确认一条
// 异步监听确认和未确认的消息
// 如果要重复运行，先停掉之前的生产者，清空队列
channel.addConfirmListener(new ConfirmListener() {
    public void handleNack(long deliveryTag, boolean multiple) throws IOException
{
        System.out.println("Broker 未确认消息，标识：" + deliveryTag);
        if (multiple) {
            // headSet 表示后面参数之前的所有元素，全部删除
            confirmSet.headSet(deliveryTag + 1L).clear();
        } else {
            confirmSet.remove(deliveryTag);
        }
        // 这里添加重发的方法
    }
    public void handleAck(long deliveryTag, boolean multiple) throws IOException {
        // 如果 true 表示批量执行了 deliveryTag 这个值以前（小于 deliveryTag 的）的所有消
息，如果为 false 的话表示单条确认
        System.out.println(String.format("Broker 已确认消息，标识：%d，多个消息：%b",
deliveryTag, multiple));
        if (multiple) {
            // headSet 表示后面参数之前的所有元素，全部删除
            confirmSet.headSet(deliveryTag + 1L).clear();
        } else {
            // 只移除一个元素
            confirmSet.remove(deliveryTag);
        }
        System.out.println("未确认的消息:"+confirmSet);
    }
});

// 开启发送方确认模式
channel.confirmSelect();
for (int i = 0; i < 10; i++) {
    long nextSeqNo = channel.getNextPublishSeqNo();
    // 发送消息
    // String exchange, String routingKey, BasicProperties props, byte[] body
    channel.basicPublish("", QUEUE_NAME, null, (msg +"-"+ i).getBytes());
    confirmSet.add(nextSeqNo);
}
System.out.println("所有消息:"+confirmSet);
```

Spring Boot：

Confirm 模式是在 Channel 上开启的，RabbitTemplate 对 Channel 进行了封装。

```
rabbitTemplate.setConfirmCallback(new RabbitTemplate.ConfirmCallback() {
    @Override
```

```
    public void confirm(CorrelationData correlationData, boolean ack, String cause)
{
        if (!ack) {
            System.out.println("发送消息失败: " + cause);
            throw new RuntimeException("发送异常: " + cause);
        }
    }
});
```

参考：

gupaoedu-vip-rabbitmq-javaapi：com.gupaoedu.confirm

gupaoedu-vip-rabbitmq-springbootapi com.gupaoedu.amqp.template.TemplateConfig

1.2 消息从交换机路由到队列

第二个环节就是消息从交换机路由到队列。大家来思考一下，在什么情况下，消息会无法路由到正确的队列?

可能因为 routingkey 错误，或者队列不存在（但是生产环境基本上不会出现这两种问题）。

我们有两种方式处理无法路由的消息，一种就是让服务端重发给生产者，一种是让交换机路由到另一个备份的交换机。

1、消息回发

```
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("监听器收到了无法路由，被返回的消息======");
        System.out.println("replyText:"+replyText);
        System.out.println("exchange:"+exchange);
        System.out.println("routingKey:"+routingKey);
        System.out.println("message:"+new String(body));
```

```
  }
});
```

Spring Boot 消息回发的方式：使用 mandatory 参数和 ReturnListener（在 Spring AMQP 中是 ReturnCallback）。

```
rabbitTemplate.setMandatory(true);
rabbitTemplate.setReturnCallback(new RabbitTemplate.ReturnCallback(){
          public void returnedMessage(Message message,
                                      int replyCode,
                                      String replyText,
                                      String exchange,
                                      String routingKey){
              System.out.println("回发的消息：");
              System.out.println("replyCode: "+replyCode);
              System.out.println("replyText: "+replyText);
              System.out.println("exchange: "+exchange);
              System.out.println("routingKey: "+routingKey);
          }
      });
```

2、消息路由到备份交换机的方式。

在创建交换机的时候，从属性中指定备份交换机。

```
Map<String,Object> arguments = new HashMap<String,Object>();
arguments.put("alternate-exchange","ALTERNATE_EXCHANGE"); // 指定交换机的备份交换机

channel.exchangeDeclare("TEST_EXCHANGE","topic", false, false, false, arguments);
```

（注意区别，队列可以指定死信交换机；交换机可以指定备份交换机）

参考：

gupaoedu-vip-rabbitmq-javaapi：com.gupaoedu.returnlistener

gupaoedu-vip-springboot-amqp：com.gupaoedu.amqp.template.TemplateConfig

1.3 消息在队列存储

第三个环节是消息在队列存储，如果没有消费者的话，队列一直存在在数据库中。

如果 RabbitMQ 的服务或者硬件发生故障，比如系统宕机、重启、关闭等等，可能会导致内存中的消息丢失，所以我们要把消息本身和元数据（队列、交换机、绑定）都保存到磁盘。

解决方案：

1、队列持久化

com.gupaoedu.vip.mq.rabbit.javaapi.simple.MyConsumer

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
```

durable：没有持久化的队列，保存在内存中，服务重启后队列和消息都会消失（钱和人一起没了）。

autoDelete：没有消费者连接的时候，自动删除。

exclusive：排他性队列的特点是：

1）只对首次声明它的连接（Connection）可见

2）会在其连接断开的时候自动删除。

2、交换机持久化

```
@Bean("GpExchange")
public DirectExchange exchange() {
    // exchangeName, durable, exclusive, autoDelete, Properties
    return new DirectExchange("GP_TEST_EXCHANGE", true, false, new HashMap<>());
}
```

3、消息持久化

com.gupaoedu.vip.mq.rabbit.javaapi.ttl.TTLProducer

```
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
      .deliveryMode(2)   // 2 代表持久化
      .contentEncoding("UTF-8")  // 编码
      .expiration("10000")  // TTL，过期时间
```

```
        .headers(headers) // 自定义属性
        .priority(5) // 优先级，默认为 5，配合队列的 x-max-priority 属性使用
        .messageId(String.valueOf(UUID.randomUUID()))
        .build();
```

如果消息没有持久化，保存在内存中，队列还在 ，但是消息在重启后会消失（人生最最痛苦的事情：人还在，钱没了）。

4、集群部署

如果只有一个 RabbitMQ 的节点，即使交换机、队列、消息做了持久化，如果服务崩溃或者硬件发生故障（比如机房起火、地震、或者机房被炸了我们先不讲……），RabbitMQ 的服务一样是不可用的。

所以为了提高 MQ 服务的可用性，保障消息的传输，我们需要有多个 RabbitMQ 的节点。

1.4 消息投递到消费者

如果消费者收到消息后没来得及处理即发生异常，或者处理过程中发生异常，会导致④失败。服务端应该以某种方式得知消费者对消息的接收情况，并决定是否重新投递这条消息给其他消费者。

RabbitMQ 提供了消费者的消息确认机制（message acknowledgement），消费者可以自动或者手动地发送 ACK 给服务端。

如果没有 ACK 会怎么办？永远等待下去？也不会。

没有收到 ACK 的消息，消费者断开连接后，RabbitMQ 会把这条消息发送给其他消费者。如果没有其他消费者，消费者重启后会重新消费这条消息，重复执行业务逻辑（如果代码修复好了还好）。

消费者怎么给 Broker 应答呢？有两种方式，一种是自动 ACK，一种是手动 ACK。

首先是自动 ACK，这个也是默认的情况。也就是我们没有在消费者处编写 ACK 的代码，消费者会在收到消息的时候就自动发送 ACK，而不是在方法执行完毕的时候发送 ACK（并不关心你有没有正常消息）。

如果想要等消息消费完毕或者方法执行完毕才发送 ACK，需要先把自动 ACK 设置成手动 ACK。把 autoAck 设置成 false。

com.gupaoedu.vip.mq.rabbit.javaapi.ack.AckConsumer

```
channel.basicConsume(QUEUE_NAME, false, consumer);
```

这个时候 RabbitMQ 会等待消费者显式地回复 ACK 后才从队列中移去消息。

```
channel.basicAck(envelope.getDeliveryTag(), true);
```

在 Spring Boot 中：

application.properties

```
spring.rabbitmq.listener.direct.acknowledge-mode=manual
spring.rabbitmq.listener.simple.acknowledge-mode=manual
```

SimpleRabbitListenerContainer 或者 SimpleRabbitListenerContainerFactory

```
factory.setAcknowledgeMode(AcknowledgeMode.MANUAL);
```

注意这三个值的区别：

NONE：自动 ACK

MANUAL： 手动 ACK

AUTO：如果方法未抛出异常，则发送 ack。如果方法抛出异常，并且不是 AmqpRejectAndDontRequeueException 则发送 nack，并且重新入队列。如果抛出异常时 AmqpRejectAndDontRequeueException 则发送 nack 不会重新入队列。

消费者又怎么调用 ACK，或者说怎么获得 Channel 参数呢?

引入 com.rabbitmq.client.Channel。

参考：gupaoedu-vip-rabbitmq-springbootapi ：com.gupaoedu.consumer.SecondConsumer

```
    @RabbitHandler
    public void process(String msg, Channel channel, Message message) throws IOException {
        System.out.println(" second queue received msg : " + msg);
        channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);
    }
```

如果消费出了问题，确实是不能发送 ACK 告诉服务端成功消费了怎么办？当然也有拒绝消息的指令，而且还可以让消息重新入队给其他消费者消费。

如果消息无法处理或者消费失败，也有两种拒绝的方式，Basic.Reject()拒绝单条，Basic.Nack()批量拒绝。

```
if (msg.contains("拒收")){
    // 拒绝消息
    // requeue：是否重新入队列，true：是；false：直接丢弃，相当于告诉队列可以直接删除掉
    // TODO 如果只有这一个消费者，requeue 为 true 的时候会造成消息重复消费
    channel.basicReject(envelope.getDeliveryTag(), false);
} else if (msg.contains("异常")){
    // 批量拒绝
    // requeue：是否重新入队列
    // TODO 如果只有这一个消费者，requeue 为 true 的时候会造成消息重复消费
    channel.basicNack(envelope.getDeliveryTag(), true, false);
}
```

如果 requeue 参数设置为 true，可以把这条消息重新存入队列，以便发给下一个消费者（当然，只有一个消费者的时候，这种方式可能会出现无限循环重复消费的情况。可以投递到新的队列中，或者只打印异常日志）。

简单地总结一下：

从生产者到 Broker、交换机到队列，队列本身，队列到消费者，我们都有相应的方法知道消费有没有正常流转，或者说当消息没有正常流转的时候采取相关措施。

思考：服务端收到了 ACK 或者 NACK，生产者会知道吗？即使消费者没有接收到消息，或者消费时出现异常，生产者也是完全不知情的。这个是符合解耦思想的，不然你用 MQ 干嘛?

但是如果现在为了保证一致性，生产者必须知道消费者有没有成功消费，怎么办？

例如，我们寄出去一个快递，是怎么知道收件人有没有收到的?

因为有物流跟踪和签收反馈，所以寄件人可以知道。

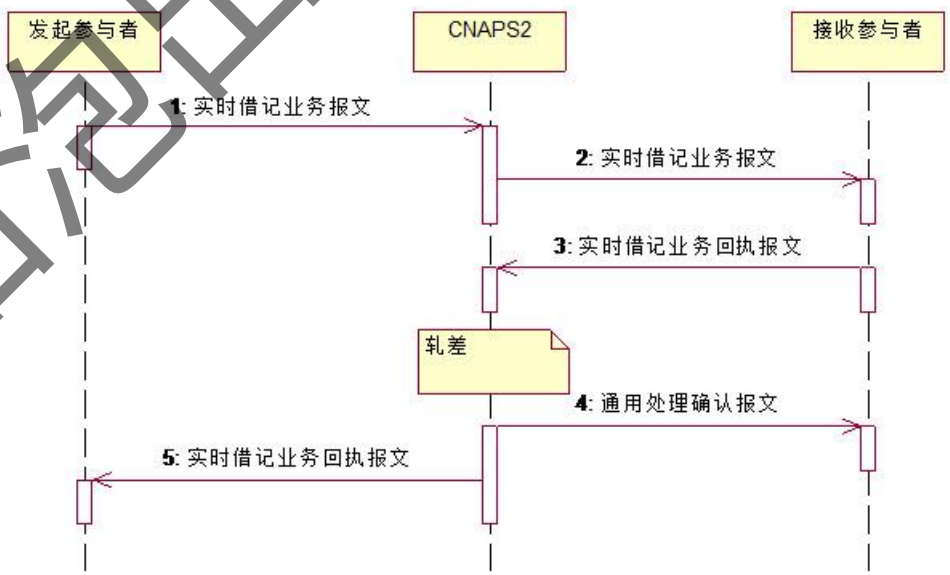
但是，在没有用上电话的年代，我们寄出去一封信，是怎么知道收信人有没有收到信件？只有收到回信，才知道寄出的信被收到了。

所以，这个是生产者最终确定消费者有没有消费成功的两种方式：

1） 消费者收到消息，处理完毕后，调用生产者的 API（思考：是否破坏解耦？）

2） 消费者收到消息，处理完毕后，发送一条响应消息给生产者。

例如人民银行二代支付系统，除了通知类的消息，所有的消息都需要有回执。



1.5 消费者回调

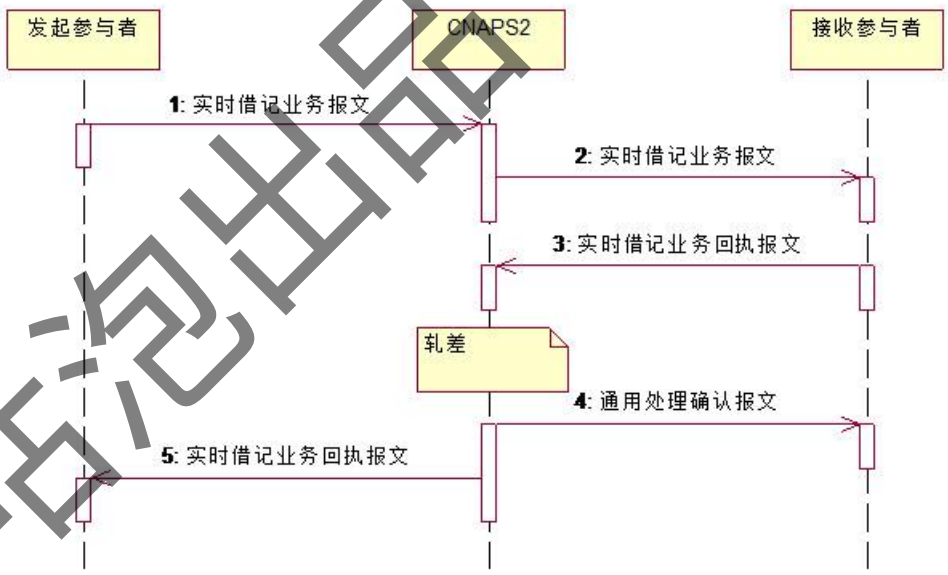
1） 调用生产者 API

例如：提单系统给其他系统发送了保险消息后（通知通知！发生了一笔保险），其他系统必须在处理完消息后调用提单系统提供的 API，来修改提单系统中这笔数据的状态。只要 API 没有被调用，数据状态没有被修改，提单系统就认为下游系统没有收到这条消息。

2） 发送响应消息给生产者

例如：商业银行与人民银行二代支付通信（使用 IBM MQ），无论是人行收到了商业银行的消息，还是商业银行收到了人行的消息，都必须发送一条响应消息（叫做回执报文）。

整个通信的流程设计得非常复杂，但是对于金融场景下的消息可靠性保证，是很有用的。



1.6 补偿机制

如果生产者的 API 就是没有被调用，也没有收到消费者的响应消息，怎么办？不要着急，先等等，可能是消费者处理时间太长或者网络超时。

你发微信消息给朋友去吃烧烤，他没有立即回复消息，别担心，可能是路上出车祸了。但是如果一直不回复消息就不行了。

生产者与消费者之间应该约定一个超时时间，对于超出这个时间没有得到响应的消息，才确定为消费失败，比如 5 分钟。

5 分钟，对于临时性故障的处理，比如网络恢复，或者重启应用，重启数据库，应该够了。

过了 5 分钟依然没有得到回复的消息，我们才判断为消费失败。

确定消费失败以后怎么办呢？肯定要重发消息了。

不过这里面有几个问题：

1、谁来重发?

假设这个消息是由业务人员操作产生的，对于异步的操作来说，他只要提交了请求就 OK 了，后面成不成功是不归他管的。所以肯定是后台的代码重发的。不可能让业务人员重新做一笔交易。

先创建一个定时任务，比如每 30 秒跑一次，找到业务表里面的这条业务状态是中间状态的记录，查询出来，构建为 MQ 消息，重新发送。

也可以单独设计设计一张消息表，把本系统所以发送出去的消息全部异步地登记起来，找出状态是未回复的消息发送（注意注意：这种做法毫无疑问会消耗性能、消耗数据库存储空间）。

2、隔多久重发一次?

假如消费者一直没有回复，比如它重启要 20 分钟，你 5 分钟之内尝试重发，肯定还不能正常消费。所以重发肯定不只发一次，要尝试多次，但是又不能发得太频繁，给它一点恢复的时间。比如可以设置为 1 分钟重发一次。也可以设置衰减机制，第一

次隔一分钟，第二次隔两分钟（谈恋爱的时候，发消息不回复，开始一天联系一次，后来一周联系一次，慢慢地失去了信心）。

时间由定时任务的执行时间决定。

3、一共重发几次?

好了，终极的问题来了，消费者真的是死了！你跟对方项目经理反馈了这个问题，他说暂时恢复不了，明天才能修复这个 bug。而你的程序 10 分钟重发一次，一个小时 6 条消息，一天就重发了 100 多条消息，后面绝大部分时间都是在做无用功，还无端造成了服务端的 MQ 消息堆积。

所以，重发消息务必要控制次数，比如设置成 3 次。

这个要在消息表里面记录次数来实现，发一次就加 1。

4、重发什么内容?

重发，是否发送一模一样的消息?

参考:

ATM 机上运行的系统叫 C 端（ATMC），银行的前置系统或者渠道系统叫 P 端（ATMP），它接收 ATMC 的消息，再转发给卡系统或者核心系统。

1）如果客户存款，没有收到核心系统的应答。怎么处理?

因为不知道有没有记账成功，不能给客户吐钞，否则会造成银行短款。因为已经吞钞了，所以要保证成功。最多发送 3 次存款确认报文;

2）如果客户取款，ATMC 未得到核心系统的应答时，怎么处理?

因为没有吐钞，所以要保证失败。最多发送 3 次存款冲正报文。

到这里故事讲完了吗？还没有。

1.7 消息幂等性

什么叫消息幂等性？用户对于同一操作发起的一次或者多次请求，最后的结果都是相同的，这就是幂等性。

假设消费者状态是正常的，每一条消息都可以正常处理。只是在响应或者调用 API 的时候出了问题，会不会出现消息的重复处理？例如：存款 1000 元，ATMC 重发了 3 次存款消息，核心系统一共处理了 4 次，账户余额增加了 4000 元。

所以，为了避免相同消息的重复处理，必须要采取一定的措施。RabbitMQ 服务端是没有这种控制的（同一批的消息有个递增的 DeliveryTag），它并不知道对于你的业务来说什么才是重复的消息。所以这个只能在消费端控制。

如何避免消息的重复消费？

消息出现重复可能会有三个原因：

1、生产者的问题，环节①重复发送消息，比如在开启了 Confirm 模式但未收到确认，消费者重复投递。

2、消费环节④出了问题，由于消费者未发送 ACK 或者其他原因，消息重复消费。

3、生产者代码或者网络问题。

对于重复发送的消息，可以对每一条消息生成一个唯一的业务 ID，通过日志或者消息落库来做重复控制。

例如：在金融系统中有一个叫流水号的东西。不管你在柜面汇款，还是 ATM 取款，或者信用卡消费，都会有一个唯一的序号。通过这个序号就可以找到唯一的一笔消息。

参考：银行的重账控制环节，对于进来的每一笔交易，第一件要做的事情就是查询是否重复。

大家有没有用微信支付的时候被提示可能是重复支付？

业务要素一致（付款人 ID、商户 ID、交易类型、金额、交易地 点、交易时间），

可能是同一笔消息。

1.8 最终一致

如果确实是消费者宕机了，或者代码出现了 BUG 导致无法正常消费，在我们尝试多次重发以后，消息最终也没有得到处理，怎么办？

刚刚我们说了，如果对方项目经理很屌，他说今天修复不了，那怎么办？不可能我们这边的消息一直是未回复的状态吧？

例如存款的场景，客户的钱已经被吞了，但是余额没有增加，这个时候银行出现了长款，应该怎么处理？（那还用说，到了我的机器里面不就是我的？）

如果客户没有主动通知银行，他也没有及时查询余额，这个问题是怎么发现的？银行最终怎么把这个账务做平？

在金融系统中，会有双方对账或者多方对账的操作，通常是在一天的业务结束之后，第二天营业之前。金融系统里面，多一分钱少一分钱都是非常严重的问题。

金融系统会约定一个标准，比如 ATM 跟核心系统对账，肯定是以核心系统的账务为准。ATMC 获取到核心系统的对账文件，然后解析，登记成数据，然后跟自己记录的流水比较，找出核心系统有 ATM 没有，或者 ATM 有核心系统没有，或者两边都有但是金额不一致的数据。

对账之后，我们再手工平账。比如取款记了账但是没吐钞的，做一笔冲正。存款吞了钞没记账的，要么把钱退给客户，要么补一笔账。

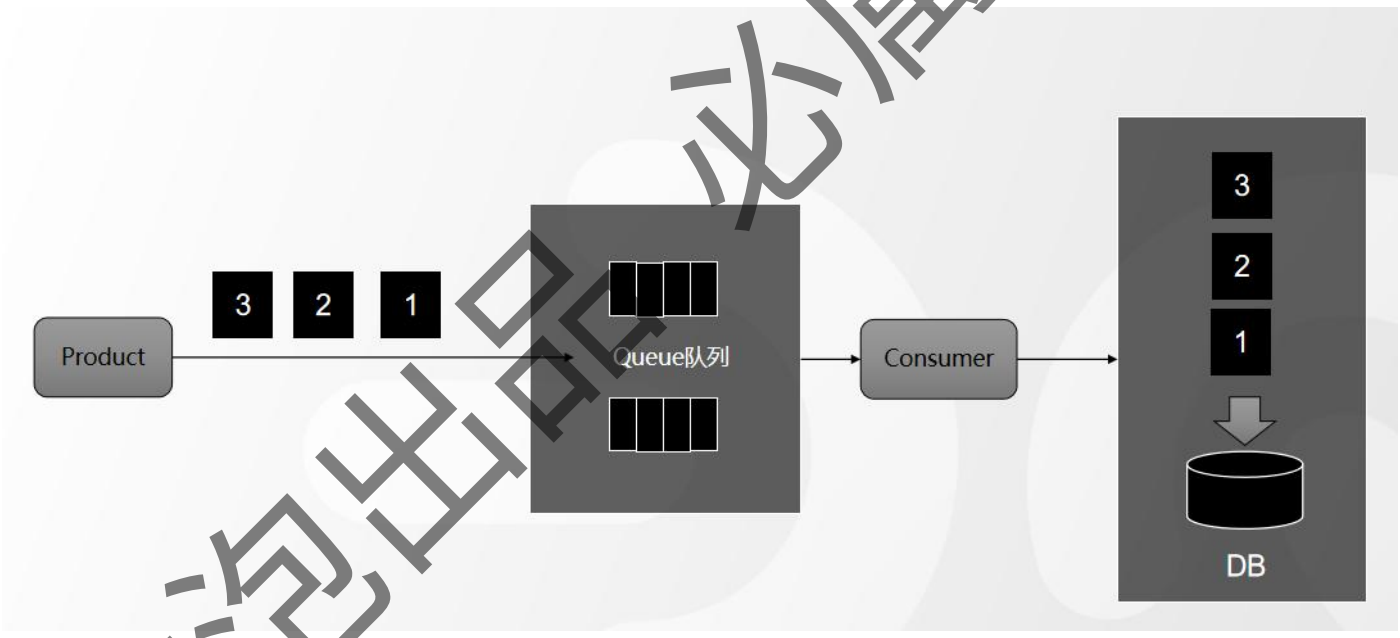
1.9 消息的顺序性

消息的顺序性指的是消费者消费消息的顺序跟生产者生产消息的顺序是一致的。

例如：商户信息同步到其他系统，有三个业务操作：1、新增门店 2、绑定产品 3、激活门店，这种情况下消息消费顺序不能颠倒（门店不存在时无法绑定产品和激活）。

又比如：1、发表微博；2、发表评论；3、删除微博。顺序不能颠倒。

在 RabbitMQ 中，一个队列有多个消费者时，由于不同的消费者消费消息的速度是不一样的，顺序无法保证。只有一个队列仅有一个消费者的情况才能保证顺序消费（不同的业务消息发送到不同的专用的队列）。



除非负载的场景，不要用多个消费者消费消息。消费端捕获异常。

2、实践经验分享

2.1 资源管理

到底在消费者创建还是在生产者创建?

如果 A 项目和 B 项目有相互发送和接收消息，应该创建几个 Vhost，几个

Exchange?

一般来说遵循谁使用谁管理，谁污染谁治理的原则。

交换机和队列，实际上是作为资源，由运维管理员创建的。

为什么仍然需要在代码中定义？重复创建不报错吗？

RabbitMQ 生 产 者 资 源 申 请 单

申请人姓名		申请人联系方式		部门	
申请时间		岗位		期望完成时间	
创建位置	☐[redacted]IDC ☐[redacted]IDC ☐[redacted] ☐阿里云				
申请类型	☐PRD环境	☐UAT环境		☐SIT环境	
申请原因					
需求描述					
项目名称					
Virtual-host					
消息模式	☐单一模式 ☐普通模式 ☐镜像模式				
Queue名称					
Exchange模式	☐Direct exchange ☐Topic exchange ☐Fanout exchange ☐Headers exchange				
Exchange名称					
Bingding key					
其他要求					
开通周期	年 月 日 至 年 月 日				
需求审批					
申请部门经理审批			中间件管理部门审批		
年 月 日			年 月 日		
（以下由Rabbitmq管理员填写并交付）					
IP地址					
PORT端口					
用户账号					

RabbitMQ 消 费 者 资 源 申 请 单					
申请人姓名		申请人联系方式		部门	
申请时间		岗位		期望完成时间	
创建位置	□ IDC □ IDC □ □阿里云				
申请类型	□PRD环境	□UAT环境		□SIT环境	
申请原因					
需求描述					
项目名称					
Queue名称					
Exchange模式	□Direct exchange □Topic exchange □Fanout exchange □Headers exchange				
Exchange名称					
Bingding key					
其他要求					
开通周期	年 月 日 至 年 月 日				
需求审批					
申请部门经理审批		中间件管理部门审批			
年 月 日		年 月 日			
（以下由Rabbitmq管理员填写并交付）					
IP地址					
PORT端口					
用户账号					
Virtual-host					

2.2 配置文件与命名规范

1、元数据的命名集中放在 properties 文件中，不要用硬编码。如果有多个系统，可以配置多个 xxx_mq.properties。

2、命名体现元数据的类型

虚拟机命名：XXX_VHOST

交换机命名：XXX_EXCHANGE

队列命名：_QUEUE

3、命名体现数据来源和去向

例如：销售系统发往产品系统的交换机：SALE_TO_PRODUCT_EXCHANGE。做

到见名知义，不用去查文档（当然注释是必不可少的）。

2.3 调用封装

在项目中可以对 Template 做进一步封装，简化消息的发送。

例如：如果交换机、路由键是固定的，封装之后就只需要一个参数：消息内容。

另外，如果想要平滑地迁移不同的 MQ（如果有这种需求的话），也可以再做一层简单的封装。

```
gpSendMsg(Objetc msg){
    JmsTemplate.send(destination,msg);
}
```

这时，如果要把 ActiveMQ 替换为 RabbitMQ，只需要修改：

```
gpSendMsg(Object msg){
    RabbitTemplate.send(exchange,routingKey,msg);
}
```

2.4 信息落库+定时任务

将需要发送的消息保存在数据库中，可以实现消息的可追溯和重复控制，需要配合定时任务来实现。

1） 将需要发送的消息登记在消息表中。

2） 定时任务一分钟或半分钟扫描一次，将未发送的消息发送到 MQ 服务器，并且修改状态为已发送。

如果需要重发消息，将指定消息的状态修改为未发送即可。

2.5 日志追踪

RabbitMQ 可以通过 Firehose 功能来记录消息流入流出的情况，用于调试，排错。

它是通过创建一个 TOPIC 类型的交换机（amq.rabbitmq.trace），把生产者发送给 Broker 的消息或者 Broker 发送给消费者的消息发到这个默认的交换机上面来实现的。

另外 RabbitMQ 也提供了一个 Firehose 的 GUI 版本，就是 Tracing 插件。

启用 Tracing 插件后管理界面右侧选项卡会多一个 Tracing，可以添加相应的策略。

RabbitMQ 还提供了其他的插件来增强功能。

https://www.rabbitmq.com/firehose.html
https://www.rabbitmq.com/plugins.html

2.6 如何减少连接数

在发送大批量消息的情况下，创建和释放连接依然有不小的开销。我们可以跟接收方约定批量消息的格式，比如支持 JSON 数组的格式，通过合并消息内容，可以减少生产者/消费者与 Broker 的连接。

比如：活动过后，要全范围下线产品，通过 Excel 导入模板，通常有几万到几十万条解绑数据，合并发送的效率更高。

建议单条消息不要超过 4M（4096KB），一次发送的消息数需要合理地控制。

```
    /**
     * 将MQ消息分成10个一条记录，并批量登记数据库
     * @param list
     * @param dType
     */
    private void partAndInsertMq(List<StoreProdToAlsBean> list,String dType) {
        // 1条记录包含10个MQ消息
        List<List<StoreProdToAlsBean>> partLists = getAverRange(list,10);
        List<MQTask> mqList = new ArrayList<MQTask>();
        for( List<SimsStoreProdToAlsBean> partList: partLists ){
            JSONArray jsonArry = JSONArray.fromObject(partList);
            JSONObject pdDelJson = new JSONObject();
            pdDelJson.put("data", jsonArry);
            pdDelJson.put("title", "storeProd");
            pdDelJson.put("dType", dType);
            MQTask mqMsg = new MQTask();
            mqMsg.setRoutingKey(Constant.MQRoutingKeys.MQ_CXF.getRoutingKey());
            mqMsg.setJsonDatastr(pdDelJson.toString());
            Long regDay = Long.parseLong(DatesUtil.dateToString(new Date(), DatesUtil.SIMPLE_TIME));
            mqMsg.setRegDay(regDay);
            mqMsg.setLastUpTms(regDay);
            mqMsg.setMqStatus("0");

            mqList.add(mqMsg);
        }
        NewBatchImportDao.batachInsertMQTask(mqList);
    }

/**
 * 把一个数组按每份N个划分，返回一个大的数组
 * @param list 数组
 * @param average 每份包含多少个
 * @return 新的数组
 */
public <T> List<List<T>> getAverRange( List<T> list, int average ){
    List<List<T>> newArrays = new ArrayList<List<T>>();
    int start = 0;
    int end = 0;
    for (int i = 0; i <= list.size() / average; i++) {
        List<T> partList = new ArrayList<T>();
        end = start + average;
        if (start >= list.size()) {
            break;
        }
        if (end >= list.size()) {
            end = list.size() ;
        }
        partList = list.subList(start, end);
        newArrays.add(partList);
        start = end;
    }
    return newArrays;
}
```

```
msgContent = {
   "type":"add",
   "num":3,
   "detail": [
```

```
      { "merchName":"黄金手机店", "address":"黄金路 999 号" ] },
      { "merchName":"尖山手机店", "address":"尖山路 168 号" ] } }
  ]
}
```

3、在生产环境中不适用的策略

3.1 vhost

在生产中，如果 rabbitmq 只为单个系统提供服务的时候，我们使用默认的(/)是可以的。在为多个系统提供的服务时，我们建议使用单独的 vhost.

3.2 user

对于生产环境，请删除默认用户(guest),默认用户只能从 localhost 连接。

我们可以创建指定权限的单独用户为每个应用提供服务。对于开启权限用户来说，我们可以使用证书，和源 ip 地址过滤，和身份验证。来加强安全性。

3.3 最大打开文件限制

在生产环境我们可能需要调整一些系统的默认限制，以便处理大量的并发连接和队列。

需要调整的值有打开的最大文件数。在生产环境为 rabbitmq 运行的用户设定为 65536，但是对于大多数开发环境来说，4096 就已经足够了。

查看默认的打开文件的最大数量。

```
ulimit -n
```

更改方式：

1 临时修改

```
ulimit -n 65536
```

2 永久修改

1)如果是 systemed 来进行管理的话我们可以编辑 systemed 配置文件来进行控制

```
[service]LimitNOFILE=300000
```

2)如果不是 systemed 来进行管理的话，我们可以更改 rabbitmq 的启动加载的环境配置文件 rabbitmq-env.conf。在里面开头添加 ulimit -S -n 4096，但该值不能超过系统的默认值的最大值。

```
ulimit -S -n 4096
```

3) 系统级别更改

更改配置文件：/etc/security/limits.conf

在文件末尾前面加入

```
rabbitmq(启动的用户名) - nofile 65536
```

如果更改前用户已经登录的话，需要重新登录下才能生效。

3.4 连接

少使用短连接，使用连接池或者长连接。

3.5 TLS

建议尽可能使用 TLS 连接，使用 TLS 会对传输的数据加密，但是对系统的吞吐量产生很大的影响

3.6 更改默认端口

我们常用的 web 界面的端口 15672 和 AMQP 0-9-1 协议端口 5672 ，建议更改，web 界面更改，配置参数 management.listener.port ， AMQP 0-9-1 协议端口配置参数 listeners.tcp.default。

4、面试题

4.1 消息队列的作用与使用场景？

要点：关键词+应用场景

4.2 Channel 和 vhost 的作用是什么？

Channel：减少 TCP 资源的消耗。也是最重要的编程接口。

Vhost：提高硬件资源利用率，实现资源隔离。

4.3 RabbitMQ 的消息有哪些路由方式？适合在什么业务场景使用？

Direct、Topic、Fanout

4.4 交换机与队列、队列与消费者的绑定关系是什么样的？

4.5 多个消费者监听一个队列时（比如一个服务部署多个实例），消息会重复消费吗?

多对多；

轮询（平均分发）

4.6 无法被路由的消息，去了哪里？

直接丢弃。可用备份交换机接收。

4.7 消息在什么时候会变成 Dead Letter（死信）？

消息过期；消息超过队列长度或容量；消息被拒绝并且未设置重回队列

4.8 如果一个项目要从多个服务器接收消息，怎么做？

定义多个 ConnectionFactory，注入到消费者监听类/Temaplate。

4.9 RabbitMQ 如何实现延迟队列？

基于数据库+定时任务；

或者消息过期+死信队列；

或者延迟队列插件。

4.10 哪些情况会导致消息丢失？怎么解决？

4.11 一个队列最多可以存放多少条消息？

MaxLength

4.12 可以用队列的 x-max-length 最大消息数来实现限流吗？例如秒杀场景。

不能，因为会删除先入队的消息，不公平。

4.13 如何提高消息的消费速率？

创建多个消费者。

4.14 AmqpTemplate 和 RabbitTemplate 的区别？

Spring AMQP 是 Spring 整合 AMQP 的一个抽象。Rabbit 是一个实现。

4.15 如何动态地创建队列和消费者？

通过 ListenerContainer

`com.gupaoedu.vip.mq.rabbit.springbootapi.amqp.container.ContainerConfig`

```
container.setQueues(getSecondQueue(), getThirdQueue()); //监听的队列
```

4.16 Spring AMQP 中消息怎么封装？用什么转换？

Message，MessageConvertor

4.17 如何保证消息的顺序性？

一个队列只有一个消费者

4.18 RabbitMQ 的集群节点类型？

磁盘节点和内存节点

4.19 如何保证 RabbitMQ 的高可用？

HAProxy（LVS）+Keepalived

4.20 大量消息堆积怎么办？

1） 重启（滑稽）

2） 多创建几个消费者同时消费

3） 直接清空队列，重发消息

4.21 设计一个 MQ，你的思路是什么？

存储与转发。

存储：内存：用什么数据结构?

磁盘：文件系统？数据库?

通信：通信协议（TCP HTTP AMQP ）？一对一？一对多?

推模式？拉模式？

其他特性……