# 第三版：RabbitMQ 50 道

## RabbitMQ 是什么?

RabbitMQ 是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件）。RabbitMQ 服务器是用 Erlang 语言编写的，而群集和故障转移是构建在开放电信平台框架上的。所有主要的编程语言均有与代理接口通讯的客户端库。

## 使用 RabbitMQ 有什么好处?

**1、** 解耦，系统 A 在代码中直接调用系统 B 和系统 C 的代码，如果将来 D 系统接入，系统 A 还需要修改代码，过于麻烦！

**2、** 异步，将消息写入消息队列，非必要的业务逻辑以异步的方式运行，加快响应速度

**3、** 削峰，并发量大的时候，所有的请求直接怼到数据库，造成数据库连接异常

## RabbitMQ 的特点?

**1、** 可靠性: RabbitMQ 使用一些机制来保证可靠性， 如持久化、传输确认及确认等。

**2、** 灵活的路由 ：在消息进入队列之前，通过交换器来路由消息。对于典型的路由功能， RabbitMQ 己经提供了一些内置的交换器来实现。针对更复杂的路由功能，可以将多个 交换器绑定在一起， 也可以通过插件机制来实现自己的交换器。

**3、** 扩展性：多个 RabbitMQ 节点可以组成一个集群，也可以根据实际业务情况动态地扩展 集群中节点。

**4、** 高可用性 ：队列可以在集群中的机器上设置镜像，使得在部分节点出现问题的情况下队 列仍然可用。

**5、** 多种协议：RabbitMQ 除了原生支持 AMQP 协议，还支持 STOMP， MQTT 等多种消息 中间件协议。

**6、** 多语言客户端 ：RabbitMQ 几乎支持所有常用语言，比如 Java、 Python、 Ruby、 PHP、 C#、 JavaScript 等。

**7、** 管理界面 ：RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息、集 群中的节点等。

**8、** 令插件机制：RabbitMQ 提供了许多插件 ， 以实现从多方面进行扩展，当然也可以编写自 己的插件

## AMQP 是什么?

RabbitMQ 就是 AMQP 协议的 Erlang 的实现(当然 RabbitMQ 还支持 STOMP2、 MQTT3 等协议 ) AMQP 的模型架构 和 RabbitMQ 的模型架构是一样的，生产者将消息发送给交换器，交换器和队列绑定 。

RabbitMQ 中的交换器、交换器类型、队列、绑定、路由键等都是遵循的 AMQP 协议中相 应的概念。目前 RabbitMQ 最新版本默认支持的是 AMQP 0-9-1。

## AMQP 协议 3 层?

1、 Module Layer：协议最高层，主要定义了一些客户端调用的命令，客户端可以用这些命令实现自己的业务逻辑。

2、 Session Layer：中间层，主要负责客户端命令发送给服务器，再将服务端应答返回客户端，提供可靠性同步机制和错误处理。

3、 TransportLayer：最底层，主要传输二进制数据流，提供帧的处理、信道服用、错误检测和数据表示等。

## AMQP 模型的几大组件?

1、 交换器（Exchange）：消息代理服务器中用于把消息路由到队列的组件。

2、 队列（Queue）：用来存储消息的数据结构，位于硬盘或内存中。

3、 绑定（Binding）：一套规则，告知交换器消息应该将消息投递给哪个队列。

## RabbitMQ 有那些基本概念?

1、 Broker：简单来说就是消息队列服务器实体

2、 Exchange：消息交换机，它指定消息按什么规则，路由到哪个队列

3、 Queue：消息队列载体，每个消息都会被投入到一个或多个队列

4、 Binding：绑定，它的作用就是把 exchange 和 queue 按照路由规则绑定起来

5、 Routing Key：路由关键字，exchange 根据这个关键字进行消息投递

6、 VHost：vhost 可以理解为虚拟 broker ，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。

7、 Producer：消息生产者，就是投递消息的程序

8、 Consumer：消息消费者，就是接受消息的程序

9、 Channel：消息通道，在客户端的每个连接里，可建立多个 channel，每个 channel 代表一个会话任务

由 Exchange、Queue、RoutingKey 三个才能决定一个从 Exchange 到 Queue 的唯一的线路。

## 什么是生产者 Producer?

消息生产者，就是投递消息的一方。

消息一般包含两个部分：消息体（payload)和标签(Label)。

## 什么是消费者 Consumer?

消费消息，也就是接收消息的一方。

消费者连接到 RabbitMQ 服务器，并订阅到队列上。消费消息时只消费消息体，丢弃标签。

## 什么是 Broker 服务节点?

Broker 可以看做 RabbitMQ 的服务节点。一般请下一个 Broker 可以看做一个 RabbitMQ 服务器。

## 什么是 Queue 队列?

Queue 是 RabbitMQ 的内部对象，用于存储消息。多个消费者可以订阅同一队列，这时队列中的消息会被平摊（轮询）给多个消费者进行处理。

## 什么是 Exchange 交换器?

Exchange:生产者将消息发送到交换器，有交换器将消息路由到一个或者多个队列中。当路由不到时，或返回给生产者或直接丢弃。

## 什么是 RoutingKey 路由键?

生产者将消息发送给交换器的时候，会指定一个 RoutingKey,用来指定这个消息的路由规则，这个 RoutingKey 需要与交换器类型和绑定键(BindingKey)联合使用才能最终生效。

## 什么是 Binding 绑定?

通过绑定将交换器和队列关联起来，一般会指定一个 BindingKey,这样 RabbitMq 就知道如何正确路由消息到队列了。

## RabbitMQ 中的 Broker 是指什么? Cluster 又是指什么?

broker 是指一个或多个 erlang node 的逻辑分组，且 node 上运行着 RabbitMQ 应用程序。cluster 是在 broker 的基础之上，增加了 node 之间共享元数据的约束。

## vhost 是什么? 起什么作用?

vhost 可以理解为虚拟 broker ，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。

## RabbitMQ 的工作模式有几种?

**simple 模式（即最简单的收发模式）**

1、 消息产生消息，将消息放入队列

2、 消息的消费者(consumer) 监听 消息队列,如果队列中有消息,就消费掉,消息被拿走后,自动从队列中删除(隐患 消息可能没有被消费者正确处理,已经从队列中消失了,造成消息的丢失，这里可以设置成手动的 ack,但如果设置成手动 ack，处理完后要及时发送 ack 消息给队列，否则会造成内存溢出)。

**work 工作模式(资源的竞争)**

**1、** 消息产生者将消息放入队列消费者可以有多个,消费者 1,消费者 2 同时监听同一个队列,消息被消费。C1 C2 共同争抢当前的消息队列内容,谁先拿到谁负责消费消息(隐患:高并发情况下,默认会产生某一个消息被多个消费者共同使用,可以设置一个开关(syncronize) 保证一条消息只能被一个消费者使用)。

**publish/subscribe 订阅(共享资源)**

**1、** 每个消费者监听自己的队列;

**2、** 生产者将消息发给 broker，由交换机将消息转发到绑定此交换机的每个队列，每个绑定交换机的队列都将接收到消息。

四、routing 路由模式![122_4.png][122_4.png]

**1、** 消息生产者将消息发送给交换机按照路由判断,路由是字符串(info) 当前产生的消息携带路由字符(对象的方法),交换机根据路由的 key,只能匹配上路由 key 对应的消息队列,对应的消费者才能消费消息;

**2、** 根据业务功能定义路由字符串

**3、** 从系统的代码逻辑中获取对应的功能字符串,将消息任务扔到对应的队列中。

**4、** 业务场景:error 通知;EXCEPTION;错误通知的功能;传统意义的错误通知;客户通知;利用 key 路由,可以将程序中的错误封装成消息传入到消息队列中,开发者可以自定义消费者,实时接收错误;

**topic 主题模式(路由模式的一种)**

**1、** 星号井号代表通配符

2、 星号代表多个单词,井号代表一个单词

3、 路由功能添加模糊匹配

4、 消息产生者产生消息,把消息交给交换机

5、 交换机根据 key 的规则模糊匹配到对应的队列,由队列的监听消费者接收消息消费（在我的理解看来就是 routing 查询的一种模糊匹配，就类似 sql 的模糊查询方式）

## 消息基于什么传输?

由于 TCP 连接的创建和销毁开销较大，且并发数受系统资源限制，会造成性能瓶颈。RabbitMQ 使用信道的方式来传输数据。信道是建立在真实的 TCP 连接内的虚拟连接，且每条 TCP 连接上的信道数量没有限制。

## RabbitMQ 中消息可能有的几种状态?

1、 alpha: 消息内容(包括消息体、属性和 headers) 和消息索引都存储在内存中 。

2、 beta: 消息内容保存在磁盘中，消息索引保存在内存中。

3、 gamma: 消息内容保存在磁盘中，消息索引在磁盘和内存中都有 。

4、 delta: 消息内容和索引都在磁盘中 。

## 如何确保消息正确地发送至 RabbitMQ?

RabbitMQ 使用发送方确认模式，确保消息正确地发送到 RabbitMQ。发送方确认模式：将信道设置成 confirm 模式（发送方确认模式），则所有在信道上的消息都会被指派一个唯一的 ID。一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个确认给生产者（包含消息唯一 ID）。如果 RabbitMQ 发生内部错误从而导致消息丢失，会发送一条 nack（not acknowledged，未确认）消息。发送方确认模式是异步的，生产者应用程序在等待确认的同时，可以继续发送消息。当确认消息到达生产者应用程序，生产者应用程序的回调方法就会被触发来处理确认消息。

## 生产者消息如何运转?

1、 Producer 先连接到 Broker,建立连接 Connection,开启一个信道(Channel)。

2、 Producer 声明一个交换器并设置好相关属性。

3、 Producer 声明一个队列并设置好相关属性。

4、 Producer 通过路由键将交换器和队列绑定起来。

5、 Producer 发送消息到 Broker,其中包含路由键、交换器等信息。

6、 相应的交换器根据接收到的路由键查找匹配的队列。

7、 如果找到，将消息存入对应的队列，如果没有找到，会根据生产者的配置丢弃或者退回给生产者。

8、 关闭信道。

9、 管理连接。

## RabbitMQ 队列结构?

**通常由以下两部分组成：**

rabbit_amqqueue_process：负责协议相关的消息处理，即接收生产者的消息、向消费者交付消息、处理消息的确认(包括生产端的 confirm 和消费端的 ack)等。

backing_queue：是消息存储的具体形式和引擎，并向 rabbit amqqueue process 提供相关的接口以供调用。

## 消费者获取消息的方式?

推

拉

## 消息如何分发?

若该队列至少有一个消费者订阅，消息将以循环（round-robin）的方式发送给消费者。每条消息只会分发给一个订阅的消费者（前提是消费者能够正常处理消息并进行确认）。

## 消息怎么路由?

从概念上来说，消息路由必须有三部分：交换器、路由、绑定。生产者把消息到交换器上；绑定决定了消息如何从路由器路由到特定的队列；消息最终到达队列，并被消费者接收。

消息到交换器时，消息将拥有一个路由键（routing key），在消息创建时设定。通过队列路由键，可以把队列绑定到交换器上。消息到达交换器后，RabbitMQ 会将消息的路由键与队列的路由键进行匹配（针对不同的交换器有不同的路由规则）。如果能够匹配到队列，则消息会投递到相应队列中；如果不能匹配到任何队列，消息将进入 “黑洞”。

**常用的交换器主要分为一下三种：**

1、 direct：如果路由键完全匹配，消息就被投递到相应的队列

2、 fanout：如果交换器收到消息，将会广播到所有绑定的队列上

3、 topic：可以使来自不同源头的消息能够到达同一个队列。使用 topic 交换器时，可以使用通配符。比如：“*” 匹配特定位置的任意文本，“.” 把路由键分为了几部分，“#” 匹配所有规则等。特别注意：发往 topic 交换器的消息不能随意的设置选择键（routing_key），必须是由"."隔开的一系列的标识符组成。

## 消息传输保证层级?

1、 At most once：最多一次。消息可能会丢失，单不会重复传输。

2、 At least once：最少一次。消息觉不会丢失，但可能会重复传输。

3、 Exactly once：恰好一次，每条消息肯定仅传输一次。

## 消费者接收消息过程?

1、 Producer 先连接到 Broker,建立连接 Connection,开启一个信道(Channel)。

2、 向 Broker 请求消费响应的队列中消息，可能会设置响应的回调函数。

3、 等待 Broker 回应并投递相应队列中的消息，接收消息。

4、 消费者确认收到的消息,ack。

5、 RabbitMq 从队列中删除已经确定的消息。

6、 关闭信道。

7、 关闭连接

## 如何确保消息接收方消费了消息?

接收方消息确认机制：消费者接收每一条消息后都必须进行确认（消息接收和消息确认是两个不同操作）。只有消费者确认了消息，RabbitMQ 才能安全地把消息从队列中删除。这里并没有用到超时机制，RabbitMQ 仅通过 Consumer 的连接中断来确认是否需要重新发送消息。也就是说，只要连接不中断，RabbitMQ 给了 Consumer 足够长的时间来处理消息。

下面罗列几种特殊情况：

- 如果消费者接收到消息，在确认之前断开了连接或取消订阅，RabbitMQ 会认为消息没有被分发，然后重新分发给下一个订阅的消费者。（ 可能存在消息重复消费的隐患，需要根据 bizId 去重）
- 如果消费者接收到消息却没有确认消息，连接也未断开，则 RabbitMQ 认为该消费者繁忙，将不会给该消费者分发更多的消息。

## 如何避免消息重复投递或重复消费?

在消息生产时，MQ 内部针对每条生产者发送的消息生成一个 inner-msg-id，作为去重和幂等的依据（消息投递失败并重传），避免重复的消息进入队列；在消息消费时，要求消息体中必须要有一个 bizId（对于同一业务全局唯一，如支付 ID、订单 ID、帖子 ID 等）作为去重和幂等的依据，避免同一条消息被重复消费。

这个问题针对业务场景来答分以下几点：

**1、** 拿到这个消息做数据库的 insert 操作。然后给这个消息做一个唯一主键，那么就算出现重复消费的情况，就会导致主键冲突，避免数据库出现脏数据。

**2、** 拿到这个消息做 Redis 的 set 的操作，因为你无论 set 几次结果都是一样的，set 操作本来就算幂等操作。

**3、** 如果上面两种情况还不行。准备一个第三方介质,来做消费记录。以 Redis 为例，给消息分配一个全局 id，只要消费过该消息，将<id,message>以 K-V 形式写入 Redis。那消费者开始消费前，先去 Redis 中查询有没消费记录即可。

## 消费者某些原因无法处理当前接受的消息如何来拒绝?

channel.basicNack

channel.basicReject

## 如何解决 RabbitMQ 丢数据的问题?

**1、** 生产者丢数据生产者的消息没有投递到 MQ 中怎么办？从生产者弄丢数据这个角度来看，RabbitMQ 提供 transaction 和 confirm 模式来确保生产者不丢消息。

transaction 机制就是说，发送消息前，开启事物(channel.txSelect())，然后发送消息，如果发送过程中出现什么异常，事物就会回滚(channel.txRollback())，如果发送成功则提交事物(channel.txCommit())。

然而缺点就是吞吐量下降了。因此，生产上用 confirm 模式的居多。一旦 channel 进入 confirm 模式，所有在该信道上面的消息都将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，rabbitMQ 就会发送一个 Ack 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 rabiitMQ 没能处理该消息，则会发送一个 Nack 消息给你，你可以进行重试操作。

**2、** 消息队列丢数据处理消息队列丢数据的情况，一般是开启持久化磁盘的配置。这个持久化配置可以和 confirm 机制配合使用，你可以在消息持久化磁盘后，再给生产者发送一个 Ack 信号。这样，如果消息持久化磁盘之前，rabbitMQ 阵亡了，那么生产者收不到 Ack 信号，生产者会自动重发。

**那么如何持久化呢，这里顺便说一下吧，其实也很容易，就下面两步：**

这样设置以后，rabbitMQ 就算挂了，重启后也能恢复数据。在消息还没有持久化到硬盘时，可能服务已经死掉，这种情况可以通过引入 mirrored-queue 即镜像队列，但也不能保证消息百分百不丢失（整个集群都挂掉）

**3、** 将 queue 的持久化标识 durable 设置为 true,则代表是一个持久的队列发送消息的时候将 deliveryMode=2

**4、** 消费者丢数据启用手动确认模式可以解决这个问题

**5、** 自动确认模式，消费者挂掉，待 ack 的消息回归到队列中。消费者抛出异常，消息会不断的被重发，直到处理成功。不会丢失消息，即便服务挂掉，没有处理完成的消息会重回队列，但是异常会让消息不断重试。

手动确认模式，如果消费者来不及处理就死掉时，没有响应 ack 时会重复发送一条信息给其他消费者；如果监听程序处理异常了，且未对异常进行捕获，会一直重复接收消息，然后一直抛异常；如果对异常进行了捕获，但是没有在 finally 里 ack，也会一直重复发送消息(重试机制)。

不确认模式，acknowledge="none" 不使用确认机制，只要消息发送完成会立即在队列移除，无论客户端异常还是断开，只要发送完就移除，不会重发。

## 如何保证消息的可靠性投递?

发送方确认模式：将信道设置成 confirm 模式（发送方确认模式），则所有在信道上的消息都会被指派一个唯一的 ID。

一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个确认给生产者（包含消息唯一 ID）。如果 RabbitMQ 发生内部错误从而导致消息丢失，会发送一条 nack（not acknowledged，未确认）消息。

发送方确认模式是异步的，生产者应用程序在等待确认的同时，可以继续发送消息。当确认消息到达生产者应用程序，生产者应用程序的回调方法就会被触发来处理确认消息。

接收方确认机制：消费者接收每一条消息后都必须进行确认（消息接收和消息确认是两个不同操作）。只有消费者确认了消息，RabbitMQ 才能安全地把消息从队列中删除。

这里并没有用到超时机制，RabbitMQ 仅通过 Consumer 的连接中断来确认是否需要重新发送消息。也就是说，只要连接不中断，RabbitMQ 给了 Consumer 足够长的时间来处理消息。保证数据的最终一致性；下面罗列几种特殊情况：

如果消费者接收到消息，在确认之前断开了连接或取消订阅，RabbitMQ 会认为消息没有被分发，然后重新分发给下一个订阅的消费者。（可能存在消息重复消费的隐患，需要去重）

如果消费者接收到消息却没有确认消息，连接也未断开，则 RabbitMQ 认为该消费者繁忙，将不会给该消费者分发更多的消息。

## 消息如何保证幂等性?

生产者方面：可以对每条消息生成一个 msgID，以控制消息重复投递

```
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
porperties.messageId(String.valueOF(UUID.randomUUID()))
```

消费者方面：消息体中必须携带一个业务 ID，如银行流水号，消费者可以根据业务 ID 去重，避免重复消费

## 消息如何被优先消费?

生产者

```
Map<String, Object> argss = new HashMap<String, Object>();
argss.put("x-max-priority",10);
```

消费者

```
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
        .priority(5) // 优先级，默认为 5，配合队列的 x-max-priority 属性使用
```

## 如何保证消息的顺序性?

一个队列只有一个消费者的情况下才能保证顺序，否则只能通过全局 ID 实现（每条消息都一个 msgId，关联的消息拥有一个 parentMsgId。可以在消费端实现前一条消息未消费，不处理下一条消息；也可以在生产端实现前一条消息未处理完毕，不下一条消息）

## 多个消费者监听一个队列时，消息如何分发?

轮询：默认的策略，消费者轮流，平均地接收消息

公平分发：根据消费者的能力来分发消息，给空闲的消费者发送更多消息

**当消费者有 x 条消息没有响应 ACK 时，不再给这个消费者发送消息**

```
channel.basicQos(int x)
```

## 无法被路由的消息去了哪里?

mandatory：true 返回消息给生产者。

mandatory：false 直接丢弃。

## 死信队列和延迟队列的使用?

**1、** 死信消息：消息被拒绝（Basic.Reject 或 Basic.Nack）并且设置 requeue 参数的值为 false 消息过期了 队列达到最大的长度

**2、** 过期消息：在 rabbitmq 中存在 2 种方可设置消息的过期时间，第一种通过对队列进行设置，这种设置后，该队列中所有的消息都存在相同的过期时间，第二种通过对消息本身进行设置，那么每条消息的过期时间都不一样。如果同时使用这 2 种方法，那么以过期时间小的那个数值为准。当消息达到过期时间还没有被消费，那么那个消息就成为了一个 死信 消息。

**3、** 队列设置：在队列申明的时候使用 x-message-ttl 参数，单位为 毫秒

**4、** 单个消息设置：是设置消息属性的 expiration 参数的值，单位为 毫秒

**5、** 延时队列：在 rabbitmq 中不存在延时队列，但是我们可以通过设置消息的过期时间和死信队列来模拟出延时队列。消费者监听死信交换器绑定的队列，而不要监听消息发送的队列。

场景演示：需求：用户在系统中创建一个订单，如果超过时间用户没有进行支付，那么自动取消订单。

**分析：**

**1、** 上面这个情况，我们就适合使用延时队列来实现，那么延时队列如何创建

**2、** 延时队列可以由 过期消息+死信队列 来时间

**3、** 过期消息通过队列中设置 x-message-ttl 参数实现

**4、** 死信队列通过在队列申明时，给队列设置 x-dead-letter-exchange 参数，然后另外申明一个队列绑定 x-dead-letter-exchange 对应的交换器。

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("127.0.0.1");
factory.setPort(AMQP.PROTOCOL.PORT);
factory.setUsername("guest");
factory.setPassword("guest");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
//
// 声明一个接收被删除的消息的交换机和队列
String EXCHANGE_DEAD_NAME = "exchange.dead";
String QUEUE_DEAD_NAME = "queue_dead";
channel.exchangeDeclare(EXCHANGE_DEAD_NAME,
BuiltinExchangeType.DIRECT);
channel.queueDeclare(QUEUE_DEAD_NAME, false, false, false, null);
channel.queueBind(QUEUE_DEAD_NAME, EXCHANGE_DEAD_NAME,
"routingkey.dead");
//
String EXCHANGE_NAME = "exchange.fanout";
String QUEUE_NAME = "queue_name";
channel.exchangeDeclare(EXCHANGE_NAME,
BuiltinExchangeType.FANOUT);
Map<String, Object> arguments = new HashMap<String, Object>();
// 统一设置队列中的所有消息的过期时间
arguments.put("x-message-ttl", 30000);
// 设置超过多少毫秒没有消费者来访问队列，就删除队列的时间
```

```
arguments.put("x-expires", 20000);
// 设置队列的最新的 N 条消息，如果超过 N 条，前面的消息将从队列中移除掉
arguments.put("x-max-length", 4);
// 设置队列的内容的最大空间，超过该阈值就删除之前的消息
arguments.put("x-max-length-bytes", 1024);
// 将删除的消息推送到指定的交换机，一般 x-dead-letter-exchange 和
x-dead-letter-routing-key 需要同时设置
arguments.put("x-dead-letter-exchange", "exchange.dead");
// 将删除的消息推送到指定的交换机对应的路由键
arguments.put("x-dead-letter-routing-key", "routingkey.dead");
// 设置消息的优先级，优先级大的优先被消费
arguments.put("x-max-priority", 10);
channel.queueDeclare(QUEUE_NAME, false, false, false, arguments);
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");
String message = "Hello RabbitMQ: ";
//
for(int i = 1; i <= 5; i++) {
 // expiration: 设置单条消息的过期时间
 AMQP.BasicProperties.Builder properties = new
AMQP.BasicProperties().builder().priority(i).expiration( i * 1000 + "");
 channel.basicPublish(EXCHANGE_NAME, "", properties.build(),
(message + i).getBytes("UTF-8"));
}
channel.close();
connection.close();
```

## 消息在什么时候会变成死信?

1、 消息拒绝并且没有设置重新入队

2、 消息过期

3、 消息堆积，并且队列达到最大长度，先入队的消息会变成 DL

## RabbitMQ 如何实现延时队列?

利用 TTL（队列的消息存活时间或者消息存活时间），加上死信交换机

```
// 设置属性，消息 10 秒钟过期
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
.expiration("10000") // TTL

// 指定队列的死信交换机
Map<String,Object> arguments = new HashMap<String,Object>();
arguments.put("x-dead-letter-exchange","DLX_EXCHANGE");
```

## RabbitMQ 事务机制?

**RabbitMQ 客户端中与事务机制相关的方法有三个:**

1、 channel.txSelect 用于将当前的信道设置成事务模式。

2、 channel.txCommit 用于提交事务 。

3、 channel.txRollback 用于事务回滚,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,通过 txRollback 来回滚。

## RabbitMQ 的集群模式有几种?

RabbitMQ 是比较有代表性的，因为是基于主从（非分布式）做高可用性的，我们就以 RabbitMQ 为例子讲解第一种 MQ 的高可用性怎么实现。RabbitMQ 有三种模式：单机模式、普通集群模式、镜像集群模式。

**1、** 单机模式，就是 Demo 级别的，一般就是你本地启动了玩玩儿的?，没人生产用单机模式

**2、** 普通模式：以两个节点（rabbit01，rabbit02）为例来进行说明，对于 Queue 来说，消息实体只存在于其中一个节点 rabbit01（或者 rabbit02），rabbit01 和 rabbit02 两个节点仅有相同的元数据，即队列结构。当消息进入 rabbit01 节点的 Queue 后，consumer 从 rabbit02 节点消费时，RabbitMQ 会临时在 rabbit01，rabbit02 间进行消息传输，把 A 中的消息实体取出并经过 B 发送给 consumer，所以 consumer 应尽量连接每一个节点，从中取消息。即对于同一个逻辑队列，要在多个节点建立物理 Queue。否则无论 consumer 连 rabbit01 或 rabbit02，出口总在 rabbit01，会产生瓶颈。当 rabbit01 节点故障后，rabbit02 节点无法取到 rabbit01 节点中还未消费的消息实体。如果做了消息持久化，那么等到 rabbit01 节点恢复，然后才可被消费。如果没有消息持久化，就会产生消息丢失的现象。

**3、** 镜像模式：把需要的队列做成镜像队列，存在与多个节点属于 RabibitMQ 的 HA 方案，该模式解决了普通模式中的问题，其实质和普通模式不同之处在于，消息体会主动在镜像节点间同步，而不是在客户端取数据时临时拉取，该模式带来的副作用也很明显，除了降低系统性能外，如果镜像队列数量过多，加之大量的消息进入，集群内部的网络带宽将会被这种同步通讯大大消耗掉，所以在对可靠性要求比较高的场合中适用

## 集群节点类型有几种?

内存节点：保存状态到内存，但持久化的队列和消息还是会保存到磁盘；

磁盘节点：保存状态到内存和磁盘，一个集群中至少需要一个磁盘节点

## 如何自动删除长时间没有消费的消息?

```
// 通过队列属性设置消息过期时间
Map<String, Object> argss = new HashMap<String, Object>();
argss.put("x-message-ttl",6000);

// 对每条消息设置过期时间
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()
      .expiration("10000") // TTL
```

## 如何确保消息不丢失?

消息持久化，当然前提是队列必须持久化

RabbitMQ 确保持久性消息能从服务器重启中恢复的方式是，将它们写入磁盘上的一个持久化日志文件，当一条持久性消息到持久交换器上时，RabbitMQ 会在消息提交到日志文件后才发送响应。

一旦消费者从持久队列中消费了一条持久化消息，RabbitMQ 会在持久化日志中把这条消息标记为等待垃圾收集。如果持久化消息在被消费之前 RabbitMQ 重启，那么 RabbitMQ 会自动重建交换器和队列（以及绑定），并重新持久化日志文件中的消息到合适的队列。