第三版：RabbitMQ 45 道

什么是 rabbitmq

采用 AMQP 高级消息队列协议的一种消息队列技术,最大的特点就是消费并不需要确保提供方存在,实现了服务之间的高度解耦

为什么要使用 rabbitmq

1、 在分布式系统下具备异步,削峰,负载均衡等一系列高级功能;

2、 拥有持久化的机制，进程消息，队列中的信息也可以保存下来。

3、 实现消费者和生产者之间的解耦。

4、 对于高并发场景下，利用消息队列可以使得同步访问变为串行访问达到一定量的限流，利于数据库的操作。

5.可以使用消息队列达到异步下单的效果，排队中，后台进行逻辑下单。

使用 rabbitmq 的场景

1、 服务间异步通信

2、 顺序消费

3、 定时任务

4、 请求削峰

如何确保消息正确地发送至 RabbitMQ？ 如何确保消息接收方消费了消息?

发送方确认模式

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

2、 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）。

3、 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。

4、 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

接收方消息确认机制

1、 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。

2、 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

1、 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）

2、 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）作为去重的依据，避免同一条消息被重复消费。

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。

消息如何分发?

1、 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）。

2、 通过路由可实现多消费的功能

消息怎么路由?

1、 消息提供方->路由->一至多个队列

2、 消息发布到交换器时，消息将拥有一个路由键（routing key），在消息创建时设定。

3、 通过队列路由键，可以把队列绑定到交换器上。

4、 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）；

常用的交换器主要分为一下三种

1、 fanout：如果交换器收到消息，将会广播到所有绑定的队列上

2、 direct：如果路由键完全匹配，消息就被投递到相应的队列

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

如何确保消息不丢失?

1、 消息持久化，当然前提是队列必须持久化

2、 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，当发布一条持久性消息到持久交换器上时，Rabbit 会在消息提交到日志文件后才发送响应。

3、 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 Rabbit 会自动重建交换器和队列（以及绑定），并重新发布持久化日志文件中的消息到合适的队列。

使用 RabbitMQ 有什么好处?

1、 服务间高度解耦

2、 异步通信性能高

3、 流量削峰

rabbitmq 的集群

镜像集群模式

你创建的 queue，无论元数据还是 queue 里的消息都会存在于多个实例上，然后每次你写消息到 queue 的时候，都会自动把消息到多个实例的 queue 里进行消息同步。

好处在于，你任何一个机器宕机了，没事儿，别的机器都可以用。坏处在于，第一，这个性能开销也太大了吧，消息同步所有机器，导致网络带宽压力和消耗很重！第二，这么玩儿，就没有扩展性可言了，如果某个 queue 负载很重，你加机器，新增的机器也包含了这个 queue 的所有数据，并没有办法线性扩展你的 queue

mq 的缺点

系统可用性降低

系统引入的外部依赖越多，越容易挂掉，本来你就是 A 系统调用 BCD 三个系统的接口就好了，人 ABCD 四个系统好好的，没啥问题，你偏加个 MQ 进来，万一 MQ 挂了咋整？MQ 挂了，整套系统崩溃了，你不就完了么。

系统复杂性提高

硬生生加个 MQ 进来，你怎么保证消息没有重复消费？怎么处理消息丢失的情况？怎么保证消息传递的顺序性？头大头大，问题一大堆，痛苦不已

一致性问题

1、 A 系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是，要是 BCD 三个系统那里，BD 两个系统写库成功了，结果 C 系统写库失败了，咋整？你这数据就不一致了。

2、 所以消息队列实际是一种非常复杂的架构，你引入它有很多好处，但是也得针对它带来的坏处做各种额外的技术方案和架构来规避掉，最好之后，你会发现，妈呀，系统复杂度提升了一个数量级，也许是复杂了 10 倍。但是关键时刻，用，还是得用的

RabbitMQ 包括哪些要素?

1. 生产者 ： 消息的创建者，发送到 RabbitMQ
2. 消费者 ： 连接到 RabbitMQ，订阅到队列上，消费消息，持续订阅（basicConsumer）和单条订阅（basicGet）
3. 消息 ：包含有效载荷和标签，有效载荷指要传输的数据，标签描述了有效载荷，并且 RabbitMQ 用它来决定谁获得消息，消费者只能拿到有效载荷，并不知道生产者是谁。

RabbitMQ 什么是信道?

信道：是生产者、消费者与 RabbitMQ 通信的渠道，生产者 publish 或是消费者 subscribe 一个队列都是通过信道来通信的。信道是建立在 TCP 连接上的虚拟连接。就是说 RabbitMQ 在一条 TCP 上建立成百上千个信道来达到多个线程处理，这个 TCP 被多个线程共享，每个线程对应一个信道，信道在 RabbitMQ 都有一个唯一的 ID，保证了信道私有性，对应上唯一的线程使用。

疑问：为什么不建立多个 TCP 连接?

原因是 RabbitMQ 需要保证性能，系统为每个线程开辟一个 TCP 是非常消耗性能的，美妙成百上千的建立销毁 TCP 会严重消耗系统性能；所以 RabbitMQ 选择建立多个信道（建立在 TCP 的虚拟连接）连接到 RabbitMQ 上

RabbitMQ 概念里的 channel、exchange 和 queue 是逻辑概念，还是对应着进程实体？作用分别是什么?

queue 具有自己的 erlang 进程；

exchange 内部实现为保存 binding 关系的查找表；

channel 是实际进行路由工作的实体，负责按照 routing_key 将 message 投递给 queue。

由 AMQP 协议描述可知，channel 是真实 TCP 连接之上的 虚拟连接 ，所有 AMQP 命令都是通过 channel 发送的，且每一个 channel 有 唯一的 ID 。一个 channel 只能被单独一个操作系统线程使用，所以投递到特定的 channel 上的 message 是有顺序的。单一个操作系统线程上允许使用多个 channel。

RabbitMQ 消息是如何路由的?

消息路由必须有三部分：交换器、路由、绑定。

生产者把消息发布到交换器上，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。

1. 消息发布到交换器时，消息将拥有一个 路由键（routing key） ， 在消息创建时设定。
2. 通过队列路由键，可以把队列绑定到交换器上。
3. 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以下三种：

1. direct ：如果路由键完全匹配，消息就会被投递到相应的队列；每个 AMQP 的实现都必须有一个 direct 交换器，包含一个空白字符串名称的默认交换器。声明一个队列时，会自动绑定到默认交换器，并且以队列名称作为路由键：channel -> basic_public($msg, '', 'queue-name')

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，"." 把路由键分为了几个标识符， "#" 匹配所有规则等。

4. 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），必须是有"."隔开的一系列的标识符组成。

RabbitMQ 消息确认过程?

1. 消费者收到的每一条消息都必须进行确认（自动确认和自行确认）

2. 消费者在声明队列时，可以置顶 autoAck 参数，当 autoAck = false 时，RabbitMQ 会等待消费者显式发送回 ack 信号后才从内存（和磁盘，如果是持久化消息的话）中删除消息，否则 RabbitMQ 会在队列中消息被消费后立即删除它。

3. 采用消息确认机制后，只要使 autoAck = false，消费者就有足够的时间处理消息（任务），不用担心处理消息过程中消费者进程挂掉后消息丢失的问题，因为 RabbitMQ 会一直持有消息直到消费者显式调用 basicAck 为止。

4. 当 autoAck = false 时，对于 RabbitMQ 服务器端而言，队列中的消息分成了两部分：一部分是等待投递给消费者的消息；一部分是已经投递给消费者，但是还没有收到消费者 ack 信号的消息。如果服务器端一直没有收到消费者的 ack

信号，并且消费此消息的消费者已经断开连接，则服务器端会安排该消息 重新进入队列，等待投递给下一个消费者（也可能还是原来的那个消费者）。

5. RabbitMQ 不会为 ack 消息设置超时时间，它判断此消息是否需要重新投递给消费者的唯一依据是消费该消息的消费者连接是否已经断开。这么设计的原因是 RabbitMQ 允许消费者消费一条消息的时间可以很久很久。

如何保证 RabbitMQ 不被重复消费?

1. 正常情况下，消费者在消费消息的时候，消费完毕后，会发送一个确认信息给消息队列，消息队列就知道该消息被消费了，就会将该消息从消息队列中删除。
2. 但是因为网络传输等故障，确认信息没有传送到消息队列，导致消息队列不知道自己已经消费过该消息了，再次将消息分发给其他的消费者。

解决思路：

1. 保证消息的唯一性，就算是多次传输，不要让消息的多次消费带来影响；
2. 保证消息幂等性；
3. 比如：在写入消息队列的数据做唯一标识，消费消息时，根据唯一标识判断该消息是否被消费过。

如何保证 RabbitMQ 消息的可靠传输?

消息不可靠的情况可能是消息丢失，劫持等原因；

丢失可能又分为：

1. 生产者丢失消息
2. 消息队列丢失消息
3. 消费者丢失消息

生产者丢失消息：

1. 从生产者弄丢数据来看，RabbitMQ 提供了 transaction 机制 和 confirm 模式 来确保生产者不丢失消息；
2. transaction 机制： 发送消息前，开启事务（channel.exSelect()），然后发送消息，如果发送过程中出现异常，事务就会回滚（channel.txRollback()），如果发送成功则提交事务（channel.txCommit()）。
3. confirm 模式：一般这种模式居多，一旦 channel 进入 confirm 模式，所有在该信道上发布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 RabbitMQ 没能处理该消息，则会发送一个 Nack 消息回来，这样可以进行重试操作。

消息队列丢失消息：

1. 针对消息队列丢失数据的情况，一般是开启持久化磁盘的配置：

2. 将队列的持久化标识 durable 设置为 true ， 则代表是一个持久的队列，发送消息的时候讲 deliveryMode=2 这样设置以后，即使 RabbitMQ 挂了，重启后也能恢复数据。

消费者丢失消息：

1. 消费者丢失消息一般是因为采用了自动确认消息模式，改为手动确认消息即可。
2. 消费者在收到消息之后，处理消息之前，会自动回复 RabbitMQ 已收到消息；如果这时候处理消息失败，就会丢失该消息；
3. 解决方案：处理消息成功后，手动回复确认消息。

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。

什么是元数据？元数据分为哪些类型？包括哪些内容？与 cluster 相关的元数据有哪些？元数据是如何保存的？元数据在 cluster 中是如何分布的？

在非 cluster 模式下，元数据主要分为 Queue 元数据（queue 名字和属性等）、Exchange 元数据（exchange 名字、类型和属性等）、Binding 元数据（存放路由关系的查找表）、Vhost 元数据（vhost 范围内针对前三者的名字空间约束

和安全属性设置）。

在 cluster 模式下，还包括 cluster 中 node 位置信息和 node 关系信息。元数据按照 erlangnode 的类型确定是仅保存于 RAM 中，还是同时保存在 RAM 和 disk 上。元数据在 cluster 中是全 node 分布的。

RAM node 和 disk node 的区别?

RAM node 仅将 fabric（即 queue、exchange 和 binding 等 RabbitMQ 基础构件）相关元数据保存到内存中，但 disk node 会在内存和磁盘中均进行存储。RAM node 上唯一会存储到磁盘上的元数据是 cluster 中使用的 disk node 的地址。要求在 RabbitMQ cluster 中至少存在一个 disk node 。

RabbitMQ 上的一个 queue 中存放的 message 是否有数量限制?

可以认为是无限制，因为限制取决于机器的内存，但是消息过多会导致处理效率的下降。

RabbitMQ 概念里的 channel、exchange 和 queue 这些东东是逻辑概念，还是对应着进程实体？这些东东分别起什么作用?

queue 具有自己的 erlang 进程；exchange 内部实现为保存 binding 关系的查找表；channel 是实际进行路由工作的实体，即负责按照 routing_key 将 message 投递给 queue 。由 AMQP 协议描述可知，channel 是真实 TCP 连

接之上的虚拟连接，所有 AMQP 命令都是通过 channel 发送的，且每一个 channel 有唯一的 ID。

一个 channel 只能被单独一个操作系统线程使用，故投递到特定 channel 上的 message 是有顺序的。但一个操作系统线程上允许使用多个 channel 。channel 号为 0 的 channel 用于处理所有对于当前 connection 全局有效的帧，而 1-65535 号 channel 用于处理和特定 channel 相关的帧。AMQP 协议给出的 channel ，其中每一个 channel 运行在一个独立的线程上，多线程共享同一个 socket。

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）。

在单 node 系统和多 node 构成的 cluster 系统中声明 queue、exchange ，以及进行 binding 会有什么不同？

当你在单 node 上声明 queue 时，只要该 node 上相关元数据进行了变更，你就会得到 Queue.Declare-ok 回应；而在 cluster 上声明 queue ，则要求 cluster 上的全部 node 都要进行元数据成功更新，才会得到 Queue.Declare-ok 回应。

另外，若 node 类型为 RAM node 则变更的数据仅保存在内存中，若类型为 disk node 则还要变更保存在磁盘上的数据。

客户端连接到 cluster 中的任意 node 上是否都能正常工作?

是的。客户端感觉不到有何不同。

问题九：若 cluster 中拥有某个 queue 的 owner node 失效了，且该 queue 被声明具有 durable 属性,是否能够成功从其他 node 上重新声明该 queue ？不能，在这种情况下，将得到 404 NOT_FOUND 错误。只能等 queue 所属的 node 恢复后才能使用该 queue 。但若该 queue 本身不具有 durable 属性，则可在其他 node 上重新声明。

cluster 中 node 的失效会对 consumer 产生什么影响? 若是在 cluster 中创建了 mirrored queue ，这时 node 失效会对 consumer 产生什么影响?

若是 consumer 所连接的那个 node 失效（无论该 node 是否为 consumer 所订阅 queue 的 owner node），则 consumer 会在发现 TCP 连接断开时，按标准行为执行重连逻辑，并根据“Assume Nothing”原则重建相应的 fabric 即可。

若是失效的 node 为 consumer 订阅 queue 的 owner node，则 consumer 只能通过 Consumer CancellationNotification 机制来检测与该 queue 订阅关系的终止，否则会出现傻等却没有任何消息来到的问题。

能够在地理上分开的不同数据中心使用 RabbitMQ cluster 么?

不能。第一，你无法控制所创建的 queue 实际分布在 cluster 里的哪个 node 上（一般使用 HAProxy + cluster 模型时都是这样），这可能会导致各种跨地域访问时的常见问题；第二，Erlang 的 OTP 通信框架对延迟的容忍度有限，这可能会触发各种超时，导致业务疲于处理；第三，在广域网上的连接失效问题将导致经典的“脑裂”问题，而 RabbitMQ 目前无法处理（该问题主要是说 Mnesia）。

为什么 heavy RPC 的使用场景下不建议采用 disk node ？

heavy RPC 是指在业务逻辑中高频调用 RabbitMQ 提供的 RPC 机制，导致不断创建、销毁 reply queue ，进而造成 disk node 的性能问题（因为会针对元数据不断写盘）。所以在使用 RPC 机制时需要考虑自身的业务场景。

向不存在的 exchange 发 publish 消息会发生什么? 向不存在的 queue 执行 consume 动作会发生什么?

都会收到 Channel.Close 信令告之不存在（内含原因 404 NOT_FOUND）。

routing_key 和 binding_key 的最大长度是多少?

255 字节。

RabbitMQ 允许发送的 message 最大可达多大?

根据 AMQP 协议规定，消息体的大小由 64-bit 的值来指定，所以你就可以知道到底能发多大的数据了。

什么情况下 producer 不主动创建 queue 是安全的?

1.message 是允许丢失的；2.实现了针对未处理消息的 republish 功能（例如采用 Publisher Confirm 机制）。

“dead letter” queue 的用途?

当消息被 RabbitMQ server 投递到 consumer 后，但 consumer 却通过 Basic.Reject 进行了拒绝时（同时设置 requeue=false），那么该消息会被放入“dead letter” queue 中。该 queue 可用于排查 message 被 reject 或 undeliver 的原因。

为什么说保证 message 被可靠持久化的条件是 queue 和 exchange 具有 durable 属性，同时 message 具有 persistent 属性才行?

binding 关系可以表示为 exchange – binding – queue 。从文档中我们知道，若要求投递的 message 能够不丢失，要求 message 本身设置 persistent 属性，要求 exchange 和 queue 都设置 durable 属性。

其实这问题可以这么想，若 exchange 或 queue 未设置 durable 属性，则在

其 crash 之后就会无法恢复，那么即使 message 设置了 persistent 属性，仍然存在 message 虽然能恢复但却无处容身的问题；同理，若 message 本身未设置 persistent 属性，则 message 的持久化更无从谈起。

什么情况下会出现 blackholed 问题?

blackholed 问题是指，向 exchange 投递了 message ，而由于各种原因导致该 message 丢失，但发送者却不知道。可导致 blackholed 的情况：1.向未绑定 queue 的 exchange 发送 message；2.exchange 以 binding_key key_A 绑定了 queue queue_A，但向该 exchange 发送 message 使用的 routing_key 却是 key_B。

如何防止出现 blackholed 问题?

没有特别好的办法，只能在具体实践中通过各种方式保证相关 fabric 的存在。另外，如果在执行 Basic.Publish 时设置 mandatory=true ，则在遇到可能出现 blackholed 情况时，服务器会通过返回 Basic.Return 告之当前 message 无法被正确投递（内含原因 312NO_ROUTE）。

Consumer Cancellation Notification 机制用于什么场景?

用于保证当镜像 queue 中 master 挂掉时，连接到 slave 上的 consumer 可以收到自身 consume 被取消的通知，进而可以重新执行 consume 动作从新选出的 master 出获得消息。若不采用该机制，连接到 slave 上的 consumer 将不会感知 master 挂掉这个事情，导致后续无法再收到新 master 广播出来的 message 。另外，因为在镜像 queue 模式下，存在将 message 进行 requeue 的可能，所以实现 consumer 的逻辑时需要能够正确处理出现重复 message 的情况。

Basic.Reject 的用法是什么?

该信令可用于 consumer 对收到的 message 进行 reject 。若在该信令中设置 requeue=true，则当 RabbitMQ server 收到该拒绝信令后，会将该 message 重新发送到下一个处于 consume 状态的 consumer 处（理论上仍可能将该消息发送给当前 consumer）。若设置 requeue=false ，则 RabbitMQ server 在收到拒绝信令后，将直接将该 message 从 queue 中移除。

另外一种移除 queue 中 message 的小技巧是，consumer 回复 Basic.Ack 但不对获取到的 message 做任何处理。而 Basic.Nack 是对 Basic.Reject 的扩展，以支持一次拒绝多条 message 的能力。