面试必备(背)-RabbitMQ八股文系列！

mp.weixin.qq.com/s/REbttMhvruCk8OcsNGzM0w

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
- 可以用它来：解耦、异步、削峰。

什么是 AMQP ？

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。

说下 AMQP 协议三层？

- 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。
- 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。
- 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？

- 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。
- 队列 (Queue)：用来存储消息的数据结构，位于硬盘或内存中。
- 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。

为什么要使用rabbitmq

- 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
- 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
- 实现消费者和生产者之间的解耦
- 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
- 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

- 服务间异步通信
- 顺序消费

- 定时任务
- 请求削峰

RabbitMQ 的优缺点？

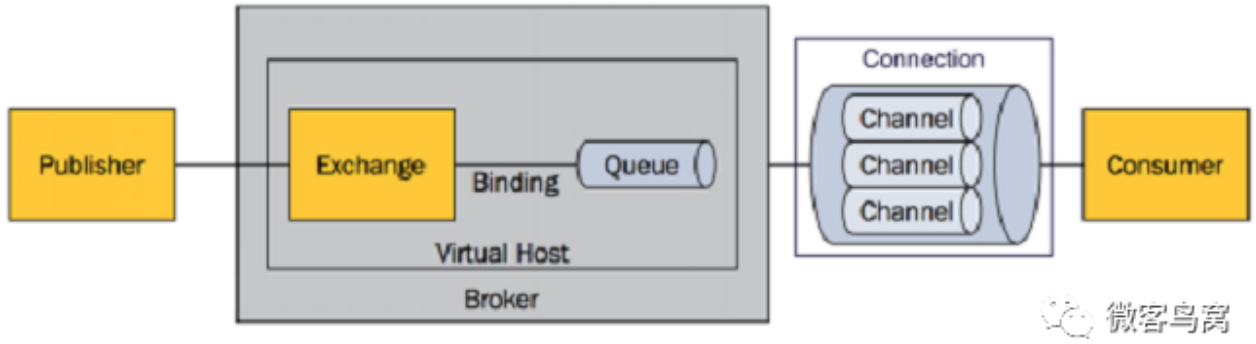
优点：

- 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！
- 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。
- 削峰
  并发量大的时候，所有的请求直接到数据库，造成数据库连接异常。

缺点：

- 降低了系统的稳定性
  本来系统运行好好的，现在你非要加入个消息队列进去，那消息队列挂了，你的系统不是呵呵了。因此，系统可用性会降低。
- 增加了系统的复杂性
  加入了消息队列，要多考虑很多方面的问题，比如：一致性问题、如何保证消息不被重复消费、如何保证消息可靠性传输等。因此，需要考虑的东西更多，复杂性增大。

RabbitMQ 的构造



（1）生产者Publisher：生产消息，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）。

（2）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。

（3）Broker服务节点：表示消息队列服务器实体。一般情况下一个Broker可以看做一个RabbitMQ服务器。

（4）Queue：消息队列，用来存放消息。一个消息可投入一个或多个队列，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。

（5）Exchange：交换器，接受生产者发送的消息，根据路由键将消息路由到绑定的队列上。

（6）Routing Key：路由关键字，用于指定这个消息的路由规则，需要与交换器类型和绑定键(Binding Key)联合使用才能最终生效。

（7）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，通过BindingKey，交换器就知道将消息路由给哪个队列了。

（8）Connection ：网络连接，比如一个TCP连接，用于连接到具体broker。

（9）Channel：信道，AMQP 命令都是在信道中进行的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接，一个TCP连接可以用多个信道。客户端可以建立多个channel，每个channel表示一个会话任务。

（10）Message：消息，由消息头和消息体组成。消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括routing-key（路由键）、priority（相对于其他消息的优先权）、delivery-mode（指出该消息可能需要持久性存储）等。

（11）Virtual host：虚拟主机，用于逻辑隔离，表示一批独立的交换器、消息队列和相关对象。一个Virtual host可以有若干个Exchange和Queue，同一个Virtual host不能有同名的Exchange或Queue。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。

RabbitMQ 中的 broker、cluster 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

RabbitMQ 概念里的 channel、exchange 和 queue 是逻辑概念，还是对应着进程实体？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

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？

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）

生产者消息运转？

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。

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。

交换器无法根据自身类型和路由键找到符合条件队列时，有哪些处理？

- mandatory ：true 返回消息给生产者。
- mandatory: false 直接丢弃。

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），则所有在信道上发布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

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

下面列出几种特殊情况：

> - 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

如何保证 RabbitMQ 的高可用？

没有哪个项目会搭建一台 RabbitMQ 服务器提供服务，风险太大；

如何保证消息的可靠性？

消息到MQ的过程中搞丢，MQ自己搞丢，MQ到消费过程中搞丢。

- 生产者到RabbitMQ：事务机制和Confirm机制，注意：事务机制和 Confirm 机制是互斥的，两者不能共存，会导致 RabbitMQ 报错。
- RabbitMQ自身：持久化、集群、普通模式、镜像模式。
- RabbitMQ到消费者：basicAck机制、死信队列、消息补偿机制。

如何保证 RabbitMQ 不被重复消费？

先说为什么会重复消费：正常情况下，消费者在消费消息的时候，消费完毕后，会发送一个确认消息给消息队列，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；

但是因为网络传输等等故障，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

- 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
- 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；保证消息幂等性
- 在写入消息队列的数据做唯一标示，消费消息时，根据唯一标识判断是否消费过

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；

丢失又分为：

- 生产者丢失消息
- 消息列表丢失消息
- 消费者丢失消息

生产者丢失消息：

> 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，开启事务（channel.txSelect()）,然后发送消息，如果发送过程中出现什么异常，事务就会回滚（channel.txRollback()）
- 如果发送成功则提交事务（channel.txCommit()）
- 这种方式有个缺点：吞吐量下降

confirm 模式用的居多：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rabbitMQ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。

消息列表丢失消息：

- 消息持久化。
- 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？

- 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`

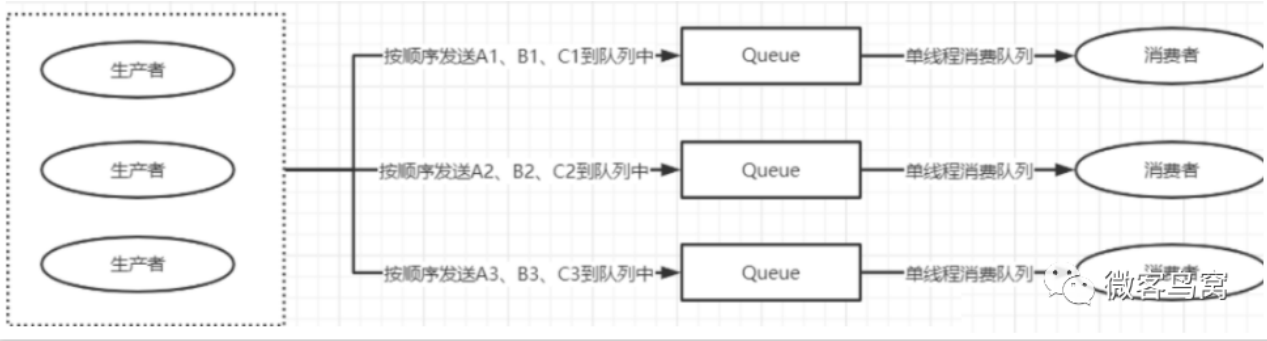
  这样设置以后，即使 rabbitMQ 挂了，重启后也能恢复数据

消费者丢失消息：

- 消费者丢数据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！
- 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；
- 如果这时处理消息失败，就会丢失该消息；
- 解决方案：处理消息成功后，手动回复确认消息。

如何保证RabbitMQ消息的顺序性？

解决方法就是保证生产者入队的顺序是有序的，出队后的顺序消费则交给消费者去保证。



- 方法一：拆分queue，使得一个queue只对应一个消费者。由于MQ一般都能保证内部队列是先进先出的，所以把需要保持先后顺序的一组消息使用某种算法都分配到同一个消息队列中。然后只用一个消费者单线程去消费该队列，这样就能保证消费者是按照顺序进行消费的了。但是消费者的吞吐量会出现瓶颈。如果多个消费者同时消费一个队列，还是可能会出现顺序错乱的情况，这就相当于是多线程消费了

- 方法二：对于多线程的消费同一个队列的情况，可以使用重试机制：比如有一个微博业务场景的操作，发微博、写评论、删除微博，这三个异步操作。如果一个消费者先执行了写评论的操作，但是这时微博都还没发，写评论一定是失败的，等一段时间。等另一个消费者，先执行发微博的操作后，再执行，就可以成功。

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。

- 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。

  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以下三种：

- direct：如果路由键完全匹配，消息就被投递到相应的队列
- fanout：如果交换器收到消息，将会广播到所有绑定的队列上
- 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。

什么是元数据？元数据分为哪些类型？

1. 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）

2. 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 erlang node 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。

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？

- 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；
- 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。

死信队列&死信交换器

DLX 全称（Dead-Letter-Exchange）,称之为死信交换器
当消息变成一个死信之后，如果这个消息所在的队列存在x-dead-letter-exchange参数，那么它会被发送到x-dead-letter-exchange对应值的交换器上，这个交换器就称之为死信交换器，与这个死信交换器绑定的队列就是死信队列

消息在什么时候会变成死信?

- 消息拒绝并且没有设置重新入队
- 消息过期
- 消息堆积，并且队列达到最大长度，先入队的消息会变成DL

延迟队列？

存储对应的延迟消息，指当消息被发送以后，并不想让消费者立刻拿到消息，而是等待特定时间后，消费者才能拿到这个消息进行消费。

优先级队列？

- 优先级高的队列会先被消费。
- 可以通过 x-max-priority 参数来实现。
- 当消费速度大于生产速度且 Broker 没有堆积的情况下，优先级显得没有意义。

事务机制？

RabbitMQ 客户端中与事务机制相关的方法有三个:

- channel.txSelect 用于将当前的信道设置成事务模式。
- channel.txCommit 用于提交事务 。
- 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。

如何解决消息队列的延时以及过期失效问题？

RabbtiMQ 是可以设置过期时间的，也就是 TTL。如果消息在 queue 中积压超过一定的时间就会被 RabbitMQ 给清理掉，这个数据就没了。这就不是说数据会大量积压在 mq 里，而是大量的数据会直接搞丢。我们可以进行批量重导。

就是大量积压的时候，直接丢弃数据，然后等过了高峰期以后，比如晚上12点以后，将丢失的那批数据，写个临时程序，然后重新灌入 mq 里面去，手动发到 mq 里去再补一次。

消息传输保证层级？

- At most once：最多一次。消息可能会丢失，但不会重复传输。
- At least once：最少一次。消息绝不会丢失，但可能会重复传输。
- Exactly once：恰好一次，每条消息肯定仅传输一次。

Virtual Host

每一个 RabbitMQ 服务器都能创建虚拟的消息服务器，也叫虚拟主机(virtual host)，简称 vhost。
默认为“/”。

RabbitMQ中消息可能有的几种状态？

- 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 。
- beta: 消息内容保存在磁盘中，消息索引保存在内存中。
- gamma: 消息内容保存在磁盘中，消息索引在磁盘和内存中都有 。
- delta: 消息内容和索引都在磁盘中 。

消息队列满了以后该怎么处理？

临时写程序，接入数据来消费，消费一个丢弃一个，都不要了，快速消费掉所有的消息。然后等过了高峰期以后再补数据。

有几百万消息持续积压几小时，怎么办？

消息积压处理办法：临时紧急扩容：

- 先修复 consumer 的问题，确保其恢复消费速度，然后将现有 consumer 都停掉。新建一个 topic，partition 是原来的 10 倍，临时建立好原先 10 倍的 queue 数量。
- 然后写一个临时的分发数据的 consumer 程序，这个程序部署上去消费积压的数据，消费之后不做耗时的处理，直接均匀轮询写入临时建立好的 10 倍数量的 queue。
- 接着临时征用 10 倍的机器来部署 consumer，每一批 consumer 消费一个临时 queue 的数据。
- 这种做法相当于是临时将 queue 资源和 consumer 资源扩大 10 倍，以正常的 10 倍速度来消费数据。
- 等快速消费完积压数据之后，得恢复原先部署的架构，重新用原先的 consumer 机器来消费消息。

八股文系列文章: