2019
版

治愈系 Java 工程师面试指导课程

[Part8・消息中间件 ActiveMQ]

[本教程梳理了 ActiveMQ 部分的复习脉络，并且涵盖了该部分高频面试题；建议大家将目录结构打开，对照目录结构做一个复习的、提纲准备面试。]

1 消息队列

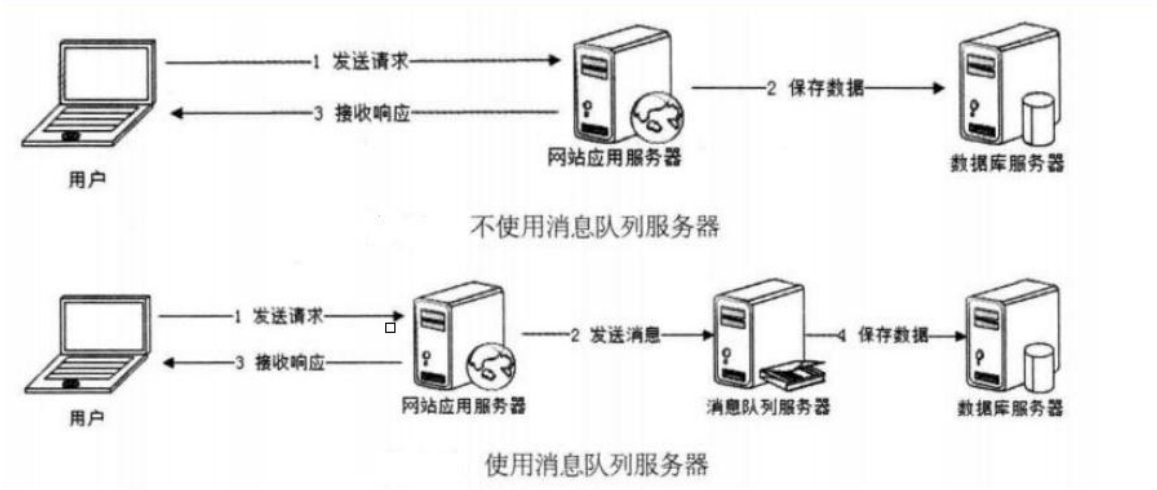
1.1 什么是消息队列

我们可以把消息队列比作是一个存放消息的容器，当我们需要使用消息的时候可以取出消息供自己使用。消息队列是分布式系统中重要的组件，使用消息队列主要是为了通过异步处理提高系统性能和削峰、降低系统耦合性。目前使用较多的消息队列有 ActiveMQ，RabbitMQ，Kafka，RocketMQ。

1.2 为什么要用消息队列

使用消息队列主要有两点好处：
1.通过异步处理提高系统性能（削峰、减少响应所需时间;
2.降低系统耦合性。【结合你自己的项目来回答】

1.2.1 通过异步处理提高系统性能（削峰、减少响应所需时间）

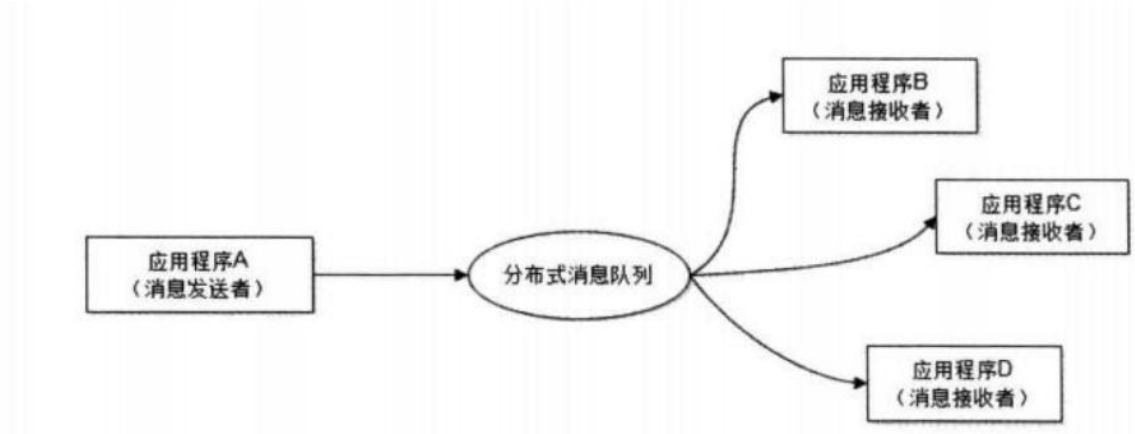


消息队列具有很好的削峰作用的功能——即通过异步处理，将短时间高并发产生的事务消息存储在消息队列中，从而削平高峰期的并发事务。 举例：在电子商务一些秒杀、促销活动中，合理使用消息队列可以有效抵御促销活动刚开始大量订单涌入对系统的冲击。

1.2.2 降低系统耦合性

我们知道如果模块之间不存在直接调用，那么新增模块或者修改模块就对其他模块影响较小，这样系统的可扩展性无疑更好一些。

我们最常见的事件驱动架构类似生产者消费者模式，在大型网站中通常用利用消息队列实现事件驱动结构。如下图所示：



消息队列使利用发布-订阅模式工作，消息发送者（生产者）发布消息，一个或多个消息接受者（消费者）订阅消息。

从上图可以看到消息发送者（生产者）和消息接受者（消费者）之间没有直接耦合，消息发送者将消息发送至分布式消息队列即结束对消息的处理，消息接受者从分布式消息队列获取该消息后进行后续处理，并不需要知道该消息从何而来。**对新增业务，只要对该类消息感兴趣，即可订阅该消息，对原有系统和业务没有任何影响，从而实现网站业务的可扩展性设计**。

1.3 使用消息队列带来的一些问题

1.3.1 系统可用性降低

系统可用性在某种程度上降低，为什么这样说呢？在加入 MQ 之前，你不用考虑消息丢失或者说 MQ 挂掉等等的情况，但是，引入 MQ 之后你就需要去考虑了！

1.3.2 系统复杂性提高

加入 MQ 之后，你需要保证消息没有被重复消费、处理消息丢失的情况、保证消息传递的顺序性等等问题！

1.3.3 一致性问题

我上面讲了消息队列可以实现异步，消息队列带来的异步确实可以提高系统响应速度。但是，万一消息的真正消费者并没有正确消费消息怎么办？这样就会导致数据不一致的情况了！

2 JMS VS AMQP

2.1 JMS

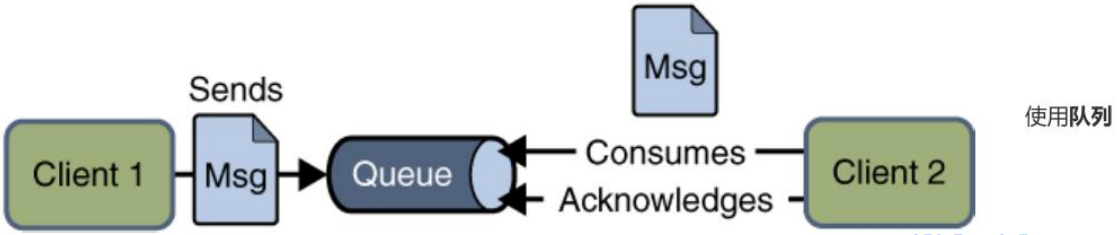
JMS（JAVA Message Service,java 消息服务）是 java 的消息服务，JMS 的客户端之间可以通过 JMS 服务进行异步的消息传输。**JMS（JAVA Message Service,Java 消息服务）API 是一个消息服务的标准或者说是规范**，允许应用

程序组件基于 JavaEE 平台创建、发送、接收和读取消息。它使分布式通信耦合度更低，消息服务更加可靠以及异步性。**ActiveMQ 就是基于 JMS 规范实现的。**

2.1.1 JMS 两种消息模型

- 点到点（P2P）模型

使用队列（Queue）作为消息通信载体；**满足生产者与消费者模式**，一条消息只能被一个消费者使用，未被消费的消息在队列中保留直到被消费或超时。



比如：我们生产者发送 100 条消息的话，两个消费者来消费一般情况下两个消费者会按照消息发送的顺序各自消费一半（也就是你一个我一个的消费。）

- 发布/订阅（Pub/Sub）模型

发布订阅模型（Pub/Sub） 使用主题（Topic）作为消息通信载体，类似于**广播模式**；发布者发布一条消息，该消息通过主题传递给所有的订阅者，在一条消息广播之后才订阅的用户则是收不到该条消息的。

2.1.2 JMS 五种不同的消息正文格式

JMS 定义了五种不同的消息正文格式，以及调用的消息类型，允许你发送并接收以一些不同形式的数据，提供现有消息格式的一些级别的兼容性。

- StreamMessage -- Java 原始值的数据流
- MapMessage--一套名称-值对
- TextMessage--一个字符串对象
- ObjectMessage--一个序列化的 Java 对象
- BytesMessage--一个字节的数据流

2.2 AMQP

AMQP，即 Advanced Message Queuing Protocol,一个提供统一消息服务的应用层标准 高级消息队列协议（二进制应用层协议），是应用层协议的一个开放标准,为面向消息的中间件设计，兼容 JMS。基于此协议的客户端与

消息中间件可传递消息，并不受客户端/中间件同产品，不同的开发语言等条件的限制。

RabbitMQ 就是基于 AMQP 协议实现的。

2.3 JMS vs AMQP

对比方向	JMS	AMQP
定义	Java API	协议
跨语言	否	是
跨平台	否	是
支持消息类型	提供两种消息模型：①Peer-2-Peer;②Pub/sub	提供了五种消息模型：①direct exchange；②fanout exchange；③topic change；④headers exchange；⑤system exchange。本质来讲，后四种和JMS的pub/sub模型没有太大差别，仅是在路由机制上做了更详细的划分；
支持消息类型	支持多种消息类型，我们在上面提到过	byte[]（二进制）

总结：

- AMQP 为消息定义了线路层（wire-level protocol）的协议，而 JMS 所定义的是 API 规范。在 Java 体系中，多个 client 均可以通过 JMS 进行交互，不需要应用修改代码，但是其对跨平台的支持较差。而 AMQP 天然具有跨平台、跨语言特性。
- JMS 支持 TextMessage、MapMessage 等复杂的消息类型；而 AMQP 仅支持 byte[] 消息类型（复杂的类型可序列化后发送）。
- 由于 Exchange 提供的路由算法，AMQP 可以提供多样化的路由方式来传递消息到消息队列，而 JMS 仅支持 队列 和 主题/订阅 方式两种。

2.4 市面上比较常用的消息队列的对比

对比方向	概要
吞吐量	万级的 ActiveMQ 和 RabbitMQ 的吞吐量（ActiveMQ 的性能最差）要比 十万级甚至是百万级的 RocketMQ 和 Kafka 低一个数量级。
可用性	都可以实现高可用。ActiveMQ 和 RabbitMQ 都是基于主从架构实现高可用性。RocketMQ 基于分布式架构。 kafka 也是分布式的，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
时效性	RabbitMQ 基于erlang开发，所以并发能力很强，性能极其好，延时很低，达到微秒级。其他三个都是 ms 级。
功能支持	除了 Kafka，其他三个功能都较为完备。 Kafka 功能较为简单，主要支持简单的MQ功能，在大数据领域的实时计算以及日志采集被大规模使用，是事实上的标准
消息丢失	ActiveMQ 和 RabbitMQ 丢失的可能性非常低， RocketMQ 和 Kafka 理论上不会丢失。

3 ActiveMQ 的高可用

ActiveMQ 提供了 master-slave、broker cluste 两种部署方式结合来满足分布式和高可用的需求。

3.1 master-slave+消息持久化解决高可用

使用 ZooKeeper（集群）注册所有的 ActiveMQ Broker。只有其中的一个 Broker 可以提供服务，被视为 Master，其他的 Broker 处于待机状态，被视为 Slave。如果 Master 因故障而不能提供服务，Zookeeper 会从 Slave 中选举出一个 Broker 充当 Master。

Slave 连接 Master 并同步他们的存储状态，Slave 不接受客户端连接。所有的存储操作都将被复制到 连接至 Master 的 Slaves。如果 Master 宕了，得到了最新更新的 Slave 会成为 Master。故障节点在恢复后会重新加入到集群中并连接 Master 进入 Slave 模式。

是不是觉得和 Redis Sentinel 主从高可用的方式很像，这里的 zookeeper 起到的作用和 reids 里的 sentinel 作用差不多。

ActiveMQ 有三种持久化方式（在 activemq.xml 可配）：分别是基于共享文件，基于 JDBC，基于 LevelDB

3.2 Broker-Cluster 解决负载和分布式

Master-Slave 的方式虽然能解决多服务热备的高可用问题，但无法解决负载均衡和分布式的问题。Broker-Cluster 的部署方式就可以解决负载均衡的问题。

Broker-Cluster 部署方式中，各个 broker 通过网络互相连接，并共享 queue。当 broker-A 上面指定的 queue-A 中接收到一个 message 处于 pending 状态，而此时没有 consumer 连接 broker-A 时。如果 cluster 中的 broker-B 上面由一个 consumer 在消费 queue-A 的消息，那么 broker-B 会先通过内部网络获取到 broker-A 上面的 message，并通知自己的 consumer 来消费。

3.3 MQ 异常导致的问题

发送方异常：使用消息的持久化机制保障重启即可
接收方异常：可能会出现重复消费问题，只要我们能搞高正保证幂等性，那么重复消费也无所谓。

给你举个例子吧。假设你有个系统，消费一条往数据库里插入一条，要是你一个消息重复两次，你不就插入了两条，这数据不就错了？但是你要是消费到第二次的时候，自己判断一下已经消费过了，直接扔了，不就保留了一条数据？

一条数据重复出现两次，数据库里就只有一条数据，这就保证了系统的幂等性

幂等性，我通俗点说，就一个数据，或者一个请求，给你重复来多次，你得确保对应的数据是不会改变的，不能出错。
其实还是得结合业务来思考，我这里给几个思路：

（1）比如你拿个数据要写库，你先根据主键查一下，如果这数据都有了，你就别插入了，update 一下好吧
（2）比如你是写 redis，那没问题了，反正每次都是 set，天然幂等性
（3）比如你不是上面两个场景，那做的稍微复杂一点，你需要让生产者发送每条数据的时候，里面加一个全局唯一的 id，类似订单 id 之类的东西，然后你这里消费到了之后，先根据这个 id 去比如 redis 里查一下，之前消费过吗？如果没有消费过，你就处理，然后这个 id 写 redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。

还有比如基于数据库的唯一键来保证重复数据不会重复插入多条，我们之前线上系统就有这个问题，就是拿到数据的时候，每次重启可能会有重复，因为 kafka 消费者还没来得及提交 offset，重复数据拿到了以后我们插入的时候，因为有唯一键约束了，所以重复数据只会插入报错，不会导致数据库中出现脏数据

如何保证 MQ 的消费是幂等性的，需要结合具体的业务来看

3.1 队列积压如何解决

3.1.1 没有设置过期时间

一般这个时候，只能操作临时紧急扩容了，具体操作步骤和思路如下：

1）先修复 consumer 的问题，确保其恢复消费速度，然后将现有 cnosumer 都停掉

2）新建一个 topic，partition 是原来的 10 倍，临时建立好原先 10 倍或者 20 倍的 queue 数量
3）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
4）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
5）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
6）等快速消费完积压数据之后，得恢复原先部署架构，重新用原先的 consumer 机器来消费消息

3.1.2 设置了过期时间

假设你用的是 ActiveMQ 是可以设置过期时间的，就是 TTL，

1）message 过期则客户端不能接收

2）ttlCeiling：表示过期时间上限（程序写的过期时间不能超过此时间，超过则以此时间为准）

3）zeroExpirationOverride：表示过期时间（给未分配过期时间的消息分配过期时间

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。那这就是第二个坑了。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。

这个情况下，就不是说要增加 consumer 消费积压的消息，因为实际上没啥积压，而是丢了大量的消息。我们可以采取一个方案，就是批量重导，这个我们之前线上也有类似的场景干过。就是大量积压的时候，我们当时就直接丢弃数据了，然后等过了高峰期以后再重新导入。

4 常见面试题

4.1 什么是 activeMQ

MQ 简称消息队列，他是一个不同项目之间进行通讯时，对消息进行管理的组件。有了 MQ，可以使项目之间交互由同步转换成异步，解耦了消息的处理过程。把消息统一管理起来，按照顺序，根据客户端的处理能力一个一个的进行操作，MQ 具有送达保证、排序保证、峰值处理、异步通信几大特性。在高并发时，对于减轻数据库压力非常有效。MQ 一般有点对点和发布订阅两种方式，点对点就是一个消息只允许接收一次，发布订阅模式，一个消息可以被多次接收。目前主流的产品有 RabbitMQ、ActiveMQ 和 Kafka；ActiveMq 是 Java 语言开发的，RabbitMQ 是 Erlang 语言开发的，理论上，RabbitMQ 的性能比 ActiveMq 更强，是非 Java 系统的首选，ActiveMq 是 Java 的，整套系统如果本来就是 Java 的，配合的默契更佳。Kafka 相当于一个分布式的 MQ，传统的 MQ，消息被消化掉后会被 mq 删除，而 kafka 中消息被消化后不会被删除，而是到配置的 expire 时间后，才删除，他把写入压力均摊到各个节点。可以通过增加节点降低压力；

ActiveMQ 是当前非常流行，能力非常强劲的消息中间件。可以在分布式系统的不同服务之间进行消息的发送和接收；它是 Apache 的开源产品，支持跨平台跨语言。完美的实现了 JMS java 消息规范，我们在项目可以直接使用 jms 的 api 来进行开发。

遵循了 JMS 消息协议，支持多种消息格式（字符串、Map、Object、字节数组、数据流）包含两个角色（Producer 生产者 Consumer 消费者），(broker)包含两种消费模式 队列模式（queue）点对点 主题模式(topic)

4.2 为什么要用 activeMQ

可以让系统解耦。异步处理，应对高并发。比如，使用消息中间件，某一个服务，可能依赖了其他好几个服务。比如我们电商项目中，商品的增加和删除都依赖其他项目，如：在我们上架一款商品时 需要依赖搜索引擎更新索引库，同时还需要页面静态化服务生成 html 页面，如果不用 MQ 我们代码就会和这两个服务耦合，使用 MQ 就可以直接通知 MQ 不和其他服务联系。

4.3 如何使用 activeMQ

在项目中，我们使用的是 SpringJMS 操作 activeMQ，已 maven 操作为例,首先先引用 SpringJMS 的依赖 及 activeMQ 的依赖，然后在 spring 的配置中，如果要配置消息的生产者的话，需要配置 springjms 的连接工厂，通过连接工厂配置 jmsTemplate 实例，我们可以使用 jmsTemplate 进行消息的相关操作，另外我们也需要配置消息的目的地，也是在 spring 的配置文件中配置配置队列 或者 主题。

消息的消费者和生产者配置差不多，不同的是需要配置一个消息的监听器，监听器里面实现 onMessage 方法，在这个方法里面处理消息

4.4 activeMQ 的消息确认机制

客户端消费消息的风格有两种：同步和异步【同步是在消费者端调用 receive()方法，异步是使用 messageListener. 在同一个消费者中只能使用一种风格】。

- AUTO_ACKNOWLEDGE=1 自动确认（自动确认的消息同步方法中 receive 方法一调用消息就会被确认消费。在异步方法中是调用完 onMessage 方法后消息被确认，如果方法抛出异常消息会重发，重发有一定次数，次数过多会进入死信队列。我们通常会使用 try catch 避免抛出异常）
- CLIENT_ACKNOWLEDGE=2 客户端手动确认（客户端手动确认需要我们在代码中使用 acknowledge 方法进行确认。）
- DUPS_OK_ACKNOWLEDGE=3 自动批量确认 SESSION_TRANSACTED=0 事务提交并确认（如果 session 是事务类型，则消息确认机制必须是这个，如果 session 是非事务的，则不可以是这个消息确认机制）session.commit();
- INDIVIDUAL_ACKNOWLEDGE=4 单条消息确认

4.5 消息持久化和事务的关系

默认的情况下，非持久化的消息是异步发送的，持久化的消息是同步发送的，遇到慢一点的硬盘，发送消息的速度是无法忍受的。但是在开启事务的情况下，消息都是异步发送的，效率会有 2 个数量级的提升。所以在发送持久化消息时，请务必开启事务模式。其实发送非持久化消息时也建议开启事务，因为根本不会影响性能。

4.6 持久化消息和持久化消息的区别

ActiveMQ 的储存机制：在通常的情况下，非持久化消息是存储在内存中的，持久化消息是存储在文件中的，它们的最大限制在配置文件的<systemUsage>节点中配置。但是，在非持久化消息堆积到一定程度，内存告急的时候，ActiveMQ 会将内存中的非持久化消息写入临时文件中，以腾出内存。虽然都保存到了文件里，但它和持久化消息的区别是，重启后持久化消息会从文件中恢复，非持久化的临时文件会直接删除。

那如果文件增大到达了配置中的最大限制的时候，生产者产生大量的持久化消息直到文件达到最大限制，此时生产者阻塞，但消费者可正常连接并消费消息，等消息消费掉一部分，文件删除又腾出空间之后，生产者又可继续发送消息，服务自动恢复正常。

生产者产生大量的非持久化消息并写入临时文件，在达到最大限制时，生产者阻塞，消费者可正常连接但不能消费消息，或者原本慢速消费的消费者，消费突然停止。整个系统可连接，但是无法提供服务，就这样挂了。

解决方案：尽量不要用非持久化消息，非要用的话，将临时文件限制尽可能的调大。

4.7 activeMQ 丢消息怎么办

这得从 java 的 java.net.SocketException 异常说起。简单点说就是当网络发送方发送一堆数据，然后调用 close 关闭连接之后。这些发送的数据都在接收者的缓存里，接收者如果调用 read 方法仍旧能从缓存中读取这些数据，尽管对方已经关闭了连接。但是当接收者尝试发送数据时，由于此时连接已关闭，所以会发生异常，这个很好理解。不过需要注意的是，当发生 SocketException 后，原本缓存区中数据也作废了，此时接收者再次调用 read 方法去读取缓存中的数据，就会报 Software caused connection abort: recv failed 错误。

通过抓包得知，ActiveMQ 会每隔 10 秒发送一个心跳包，这个心跳包是服务器发送给客户端的，用来判断客户端死没死。如果你看过上面第一条，就会知道非持久化消息堆积到一定程度会写到文件里，这个写的过程会阻塞所有动作，而且会持续 20 到 30 秒，并且随着内存的增大而增大。当客户端发完消息调用 connection.close()时，会期待服务器对于关闭连接的回答，如果超过 15 秒没回答就直接调用 socket 层的 close 关闭 tcp 连接了。这时客户端发出的消息其实还在服务器的缓存里等待处理，不过由于服务器心跳包的设置，导致发生了 java.net.SocketException 异常，把缓存里的数据作废了，没处理的消息全部丢失。

解决方案：用持久化消息，或者非持久化消息及时处理不要堆积，或者启动事务，启动事务后，commit()方法会负责任的等待服务器的返回，也就不会关闭连接导致消息丢失了。

4.8 消息的不均匀消费

有时在发送一些消息之后，开启 2 个消费者去处理消息。会发现一个消费者处理了所有的消息，另一个消费者根本没收到消息。原因在于 ActiveMQ 的 prefetch 机制。当消费者去获取消息时，不会一条一条去获取，而是一次性获取一批，默认是 1000 条。这些预获取的消息，在还没确认消费之前，在管理控制台还是可以看见这些消息的，但是不会再分配给其他消费者，此时这些消息的状态应该算作“已分配未消费”，如果消息最后被消费，则会在服务器端被删除，如果消费者崩溃，则这些消息会被重新分配给新的消费者。但是如果消费者既不消费确认，又不崩溃，那这些消息就永远躺在消费者的缓存区里无法处理。更通常的情况是，消费这些消息非常耗时，你开了 10 个消费者去处理，结果发现只有一台机器吭哧吭哧处理，另外 9 台啥事不干。

解决方案：将 prefetch 设为 1，每次处理 1 条消息，处理完再去取，这样也慢不了多少。

4.9 activeMQ 中死信队列

如果你想在消息处理失败后，不被服务器删除，还能被其他消费者处理或重试，可以关闭 AUTO_ACKNOWLEDGE，将 ack 交由程序自己处理。那如果使用了 AUTO_ACKNOWLEDGE，消息是什么时候被确认的，还有没有阻止消息确认的方法？有！

消费消息有 2 种方法，一种是调用 consumer.receive()方法，该方法将阻塞直到获得并返回一条消息。这种情况下，消息返回给方法调用者之后就自动被确认了。另一种方法是采用 listener 回调函数，在有消息到达时，会调用 listener 接口的 onMessage 方法。在这种情况下，在 onMessage 方法执行完毕后，消息才会被确认，此时只要在方法中抛出异常，该消息就不会被确认。那么问题来了，如果一条消息不能被处理，会被退回服务器重新分配，如果只

有一个消费者，该消息又会重新被获取，重新抛异常。就算有多个消费者，往往在一个服务器上不能处理的消息，在另外的服务器上依然不能被处理。难道就这么退回--获取--报错死循环了吗？

在重试 6 次后，ActiveMQ 认为这条消息是“有毒”的，将会把消息丢到死信队列里。如果你的消息不见了，去 ActiveMQ.DLQ 里找找，说不定就躺在那里。

4.10 修改重发间隔和重发次数

4.10.1 消息重发的场景

在哪些情况下，ActiveMQ 服务器会将消息重发给消费者，这里为简单起见，假定采用的消息发送模式为队列（即消息发送者和消息接收者）。

- 如果消息接收者在处理完一条消息的处理过程后没有对消息中间件(MOM)进行应答，则该消息将由 MOM 重发.
- 如果我们队某个队列设置了预读参数（consumer.prefetchSize），如果消息接收者在处理第一条消息时（没向 MOM 发送消息接收确认）就宕机了，则预读数量的所有消息都将被重发!
- 如果 Session 是事务的，则只要消息接收者有一条消息没有确认，或发送消息期间 MOM 或客户端某一方突然宕机了，则该事务范围中的所有消息 MOM 都将重发。

ActiveMQ 消息服务器怎么知道消费者客户端到底是消息正在处理中还没来得急对消息进行应答还是已经处理完成了没有应答或是宕机了根本没机会应答呢？其实在所有的客户端机器上，内存中都运行着一套客户端的 ActiveMQ 环境，该环境负责缓存发来的消息，负责维持着和 ActiveMQ 服务器的消息通讯，负责失效转移（fail-over）等，所有的判断和处理都是由这套客户端环境来完成的。

4.10.2 消息重发策略的配置

我们可以来对 ActiveMQ 的重发策略（Redelivery Policy）来进行自定义配置，其中的配置参数主要有以下几个：
可用的属性属性 默认值 说明

- collisionAvoidanceFactor：默认值 0.15，设置防止冲突范围的正负百分比，只有启用 useCollisionAvoidance 参数时才生效。
- lmaximumRedeliveries：默认值 6，最大重传次数，达到最大重连次数后抛出异常。为-1 时不限制次数，为 0 时表示不进行重传。

maximumRedeliveryDelay 默认值-1，最大传送延迟，只在 useExponentialBackOff 为 true 时有效（V5.5），假设首次重连间隔为 10ms，倍数为 2，那么第二次重连时间间隔为 20ms，第三次重连时间间隔为 40ms，当重连时间间隔大的最大重连时间间隔时，以后每次重连时间间隔都为最大重连时间间隔。

- initialRedeliveryDelay 默认值 1000L，初始重发延迟时间
- redeliveryDelay 默认值 1000L，重发延迟时间，当 initialRedeliveryDelay=0 时生效（v5.4）
- useCollisionAvoidance 默认值 false，启用防止冲突功能，因为消息接收时是可以使用多线程并发处理的，应该是为了重发的安全性，避开所有并发线程都在同一个时间点进行消息接收处理。所有线程在同一个时间点处理时会发生什么问题呢？应该没有问题，只是为了平衡 broker 处理性能，不会有时很忙，有时很空闲。
- useExponentialBackOff 默认值 false，启用指数倍数递增的方式增加延迟时间。
- backOffMultiplier 默认值 5，重连时间间隔递增倍数，只有值大于 1 和启用 useExponentialBackOff 参数时才生效。

4.11 遇到的问题

4.12 activemq 信息优先级

JMS 标准中约定 priority 可以为 0~9 的数值，值越大表示权重越高，默认值为 4。不过 activeMQ 中各个存储器对 priority 的支持并非完全一样。比如 JDBC 存储器可以支持 0~9，因为 JDBC 存储器可以基于 priority 对消息进行排序和索引化；但是对于 kahadb/levelDB 等这种基于日志文件的存储器而言，priority 支持相对较弱，只能识别三种优先级(LOW: < 4,NORMAL: =4,HIGH: > 4)。在 broker 端，默认是不支持 priority 排序的，我们需要手动开启:

5 rabbitMQ

5.1 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

5.2 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？

5.2.1 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。

5.2.2 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；

两种况：

- ✓ 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。

5.3 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；

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。

5.4 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。

5.5 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）。通过路由可实现多消费的功能

5.6 消息怎么路由？

消息提供方->路由->一至多个队列
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）；
常用的交换器主要分为以下三种：

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

5.7 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。abbitMQ 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。

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。