1. **activemq 的几种通信方式**

publish(发布)-subscribe(订阅)(发布-订阅方式)

发布/订阅方式用于多接收客户端的方式.作为发布订阅的方式，可能存在多个接收客户端，并且接收端客户端与发送客户端存在时间上的依赖。一个接收端只能接收他创建以后发送客户端发送的信息。作为 subscriber ,在接收消息时有两种方法，destination 的 receive 方法，和实现 message listener 接口的 onMessage 方法。

p2p(point-to-point)(点对点)

p2p 的过程则理解起来比较简单。它好比是两个人打电话，这两个人是独享这一条通信链路的。一方发送消息，另外一方接收，就这么简单。在实际应用中因为有多个用户对使用 p2p 的链路。

在 p2p 的场景里，相互通信的双方是通过一个类似于队列的方式来进行交流。和前面 pub-sub 的区别在于一个 topic 有一个发送者和多个接收者，而在 p2p 里一个 queue 只有一个发送者和一个接收者。

2.**activemq 如果数据提交不成功怎么办(消息丢失)**

1. **publish(发布)-subscribe(订阅)方式的处理**

发布订阅模式的通信方式， 默认情况下只通知一次， 如果接收不到此消息就没有了。 这种场景只适用于对消息送达率要求不高的情况。 如果要求消息必须送达不可以丢失的话， 需要配置持久订阅。 每个订阅端定义一个 id，

<property name="clientId" 在订阅是向 activemq 注册。 发布消息

<**property** name="subscriptionDurable" value="true"**/>**和接收消息时需要配置发送模式为持久化 template.setDeliveryMode(DeliveryMode.***PERSISTENT***);。 此时如果客户端接收不到消息， 消息会持久化到服务端(就是硬盘上)， 直到客户端正常接收后为止。

1. **4.2p - p(点对点)方式的处理**

   点对点模式的话， 如果消息发送不成功此消息默认会保到 activemq 服务端直到有消费者将其消费， 所以此时消息是不会丢失的。

3.如何解决消息重复问题

所谓消息重复,就是消费者接收到了重复的消息,一般来说我们对于这个问题的处理要把握下面几点,

①.消息不丢失(上面已经处理了)

②.消息不重复执行

一般来说我们可以在业务段加一张表,用来存放消息是否执行成功,每次业务事物 commit 之后,告知服务端,已经处理过该消息,

这样即使你消息重发了,也不会导致重复处理

大致流程如下:

业务端的表记录已经处理消息的 id,每次一个消息进来之前先判断该消息是否执行过,如果执行过就放弃,如果没有执行就开始执行消息,消息执行完成之后存入这个消息的 id

4.大量的消息每页被消费，能否发生 oom 异常？

可以控制每个消息队列中数据的大小，不允许无线填充数据，避免该队列多大，导致过度消耗系统资源问题； 可以控制队列的内存大小；

5.activeMQ 发送消息的方式有哪些？

消息通信的基本方式有两种：

1、同步方式

两个通信应用服务之间必须要进行同步，两个服务之间必须都是正常运行的。发送程序和接收程序都必须一直处于运行状态，并且随时做好相互通信的准备。

发送程序首先向接收程序发起一个请求，称之为发送消息，发送程序紧接着就会堵塞当前自身的进程，不与其他应用进行任何的通信以及交互，等待接收程序的响应，待发送消息得到接收程序的返回消息之后会继续向下运行，进行下一步的业务处理。

2、异步方式

两个通信应用之间可以不用同时在线等待，任何一方只需各自处理自己的业务，比如发送方发送消息以后不用登录接收方的响应，可以接着处理其他的任务。也就是说发送方和接收方都是相互独立存在的，发送方只管方，接收方只能接收，无须去等待对方的响应。

Java 中 JMS 就是典型的异步消息处理机制，JMS 消息有两种类型：点对点、发布/订阅。

6. activeMQ 如何调优

1. 使用非持久化消息;

2. 需要确保消息发送成功时使用事务来将消息分批组合.

```
public void sendTransacted() throws JMSException {
        ActiveMQConnectionFactory cf = new
ActiveMQConnectionFactory();
        Connection connection = cf.createConnection();
        connection.start();
        Session session = connection.createSession(true,
Session.SESSION_TRANSACTED);
        Topic topic = session.createTopic("Test.Transactions");
        MessageProducer producer = session.createProducer(topic);
        int count = 0;
        for (int i = 0; i < 1000; i++) {
            Message message = session.createTextMessage("message " +
i);

            producer.send(message);
            if (i != 0 && i % 10 == 0) {
```

```
                session.commit();
            }
        }

    }

    public void sendNonTransacted() throws JMSException {
        ActiveMQConnectionFactory cf = new
ActiveMQConnectionFactory();
        Connection connection = cf.createConnection();
        connection.start();
        // create a default session (no transactions)
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWELDGE);
        Topic topic = session.createTopic("Test.Transactions");
        MessageProducer producer = session.createProducer(topic);
        int count = 0;
        for (int i = 0; i < 1000; i++) {
            Message message = session.createTextMessage("message " +
i);
            producer.send(message);
```

```
        }
    }
```

7.什么是死信队列？

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。

8.Basic.Reject 的用法是什么？

答：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。

9.为什么不应该对所有的 message 都使用持久化机制？

答：首先，必然导致性能的下降，因为写磁盘比写 RAM 慢的多，message 的吞吐量可能有 10 倍的差距。其次，message 的持久化机制用在 RabbitMQ 的内置 cluster 方案时会出现“坑爹”问题。矛盾点在于，若 message 设置了 persistent 属性，但 queue 未设置 durable 属性，那么当该 queue 的 owner node 出现异常后，在未重建该 queue 前，发往该 queue 的 message 将被 blackholed ；若 message 设置了 persistent 属性，同时 queue 也设置了 durable 属性，那么当 queue 的 owner node 异常且无法重启的情况下，则该 queue 无法在其他 node 上重建，只能等待其 owner node 重启后，才能恢复该 queue 的使用，而在这段时间

内发送给该 queue 的 message 将被 blackholed 。所以，是否要对 message 进行持久化，需要综合考虑性能需要，以及可能遇到的问题。若想达到 100,000 条/秒以上的消息吞吐量（单 RabbitMQ 服务器），则要么使用其他的方式来确保 message 的可靠 delivery ，要么使用非常快速的存储系统以支持全持久化（例如使用 SSD）。另外一种处理原则是：仅对关键消息作持久化处理（根据业务重要程度），且应该保证关键消息的量不会导致性能瓶颈。

10.为什么 heavy RPC 的使用场景下不建议采用 disk node ？

答：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。

11.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？

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

12.什么情况下 producer 不主动创建 queue 是安全的？

答：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）。

13. “dead letter” queue 的用途？

答：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。

14.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？

答：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。