消息中间件面试题-参考回答

面试官：RabbitMQ-如何保证消息不丢失

候选人：

嗯！我们当时MYSQL和Redis的数据双写一致性就是采用RabbitMQ实现同步的，这里面就要求了消息的高可用性，我们要保证消息的不丢失。主要从三个层面考虑

第一个是开启生产者确认机制，确保生产者的消息能到达队列，如果报错可以先记录到日志中，再去修复数据

第二个是开启持久化功能，确保消息未消费前在队列中不会丢失，其中的交换机、队列、和消息都要做持久化

第三个是开启消费者确认机制为auto，由spring确认消息处理成功后完成ack，当然也需要设置一定的重试次数，我们当时设置了3次，如果重试3次还没有收到消息，就将失败后的消息投递到异常交换机，交由人工处理

面试官：RabbitMQ消息的重复消费问题如何解决的

候选人：

嗯，这个我们还真遇到过，是这样的，我们当时消费者是设置了自动确认机制，当服务还没来得及给MQ确认的时候，服务宕机了，导致服务重启之后，又消费了一次消息。这样就重复消费了

因为我们当时处理的支付（订单|业务唯一标识），它有一个业务的唯一标识，我们再处理消息时，先到数据库查询一下，这个数据是否存在，如果不存在，说明没有处理过，这个时候就可以正常处理这个消息了。如果已经存在这个数据了，就说明消息重复消费了，我们就不需要再消费了

面试官：那你还知道其他的解决方案吗？

候选人：

嗯，我想想~

其实这个就是典型的幂等的问题，比如，redis分布式锁、数据库的锁都是可以的

面试官：RabbitMQ中死信交换机？（RabbitMQ延迟队列有了解过嘛）

候选人：

嗯！了解过！

我们当时的xx项目有一个xx业务，需要用到延迟队列，其中就是使用RabbitMQ来实现的。

延迟队列就是用到了死信交换机和TTL（消息存活时间）实现的。

如果消息超时未消费就会变成死信，在RabbitMQ中如果消息成为死信，队列可以绑定一个死信交换机，在死信交换机上可以绑定其他队列，在我们发消息的时候可以按照需求指定TTL的时间，这样就实现了延迟队列的功能了。

我记得RabbitMQ还有一种方式可以实现延迟队列，在RabbitMQ中安装一个死信插件，这样更方便一些，我们只需要在声明交互机的时候，指定这个就是死信交换机，然后在发送消息的时候直接指定超时时间就行了，相对于死信交换机+TTL要省略了一些步骤

面试官：如果有100万消息堆积在MQ , 如何解决？

候选人：

我在实际的开发中，没遇到过这种情况，不过，如果发生了堆积的问题，解决方案也所有很多的

第一:提高消费者的消费能力 ,可以使用多线程消费任务

第二：增加更多消费者，提高消费速度

使用工作队列模式, 设置多个消费者消费消费同一个队列中的消息

第三：扩大队列容积，提高堆积上限

可以使用RabbitMQ惰性队列，惰性队列的好处主要是

①接收到消息后直接存入磁盘而非内存

②消费者要消费消息时才会从磁盘中读取并加载到内存

③支持数百万条的消息存储

面试官：RabbitMQ的高可用机制有了解过嘛

候选人：

嗯，熟悉的~

我们当时项目在生产环境下，使用的集群，当时搭建是镜像模式集群，使用了3台机器。

镜像队列结构是一主多从，所有操作都是主节点完成，然后同步给镜像节点，如果主节点宕机后，镜像节点会替代成新的主节点，不过在主从同步完成前，主节点就已经宕机，可能出现数据丢失

面试官：那出现丢数据怎么解决呢？

候选人：

我们可以采用仲裁队列，与镜像队列一样，都是主从模式，支持主从数据同步，主从同步基于Raft协议，强一致。

并且使用起来也非常简单，不需要额外的配置，在声明队列的时候只要指定这个是仲裁队列即可

面试官：Kafka是如何保证消息不丢失

候选人：

嗯，这个保证机制很多，在发送消息到消费者接收消息，在每个阶段都有可能会丢失消息，所以我们解决的话也是从多个方面考虑

第一个是生产者发送消息的时候，可以使用异步回调发送，如果消息发送失败，我们可以通过回调获取失败后的消息信息，可以考虑重试或记录日志，后边再做补偿都是可以的。同时在生产者这边还可以设置消息重试，有的时候是由于网络抖动的原因导致发送不成功，就可以使用重试机制来解决

第二个在broker中消息有可能会丢失，我们可以通过kafka的复制机制来确保消息不丢失，在生产者发送消息的时候，可以设置一个acks，就是确认机制。我们可以设置参数为all，这样的话，当生产者发送消息到了分区之后，不仅仅只在leader分区保存确认，在follwer分区也会保存确认，只有当所有的副本都保存确认以后才算是成功发送了消息，所以，这样设置就很大程度了保证了消息不会在broker丢失

第三个有可能是在消费者端丢失消息，kafka消费消息都是按照offset进行标记消费的，消费者默认是自动按期提交已经消费的偏移量，默认是每隔5s提交一次，如果出现重平衡的情况，可能会重复消费或丢失数据。我们一般都会禁用掉自动提价偏移量，改为手动提交，当消费成功以后再报告给broker消费的位置，这样就可以避免消息丢失和重复消费了

面试官：Kafka中消息的重复消费问题如何解决的

候选人：

kafka消费消息都是按照offset进行标记消费的，消费者默认是自动按期提交已经消费的偏移量，默认是每隔5s提交一次，如果出现重平衡的情况，可能会重复消费或丢失数据。我们一般都会禁用掉自动提价偏移量，改为手动提交，当消费成功以后再报告给broker消费的位置，这样就可以避免消息丢失和重复消费了

为了消息的幂等，我们也可以设置唯一主键来进行区分，或者是加锁，数据库的锁，或者是redis分布式锁，都能解决幂等的问题

面试官：Kafka是如何保证消费的顺序性

候选人：

kafka默认存储和消费消息，是不能保证顺序性的，因为一个topic数据可能存储在不同的分区中，每个分区都有一个按照顺序的存储的偏移量，如果消费者关联了多个分区不能保证顺序性

如果有这样的需求的话，我们是可以解决的，把消息都存储同一个分区下就行了，有两种方式都可以进行设置，第一个是发送消息时指定分区号，第二个是发送消息时按照相同的业务设置相同的key，因为默认情况下分区也是通过key的hashcode值来选择分区的，hash值如果一样的话，分区肯定也是一样的

面试官：Kafka的高可用机制有了解过嘛

候选人：

嗯，主要是有两个层面，第一个是集群，第二个是提供了复制机制

kafka集群指的是由多个broker实例组成，即使某一台宕机，也不耽误其他broker继续对外提供服务

复制机制是可以保证kafka的高可用的，一个topic有多个分区，每个分区有多个副本，有一个leader，其余的是follower，副本存储在不同的broker中；所有的分区副本的内容是都是相同的，如果leader发生故障时，会自动将其中一个follower提升为leader，保证了系统的容错性、高可用性

面试官：解释一下复制机制中的ISR

候选人：

ISR的意思是in-sync replica，就是需要同步复制保存的follower

其中分区副本有很多的follower，分为了两类，一个是ISR，与leader副本同步保存数据，另外一个普通的副本，是异步同步数据，当leader挂掉之后，会优先从ISR副本列表中选取一个作为leader，因为ISR是同步保存数据，数据更加的完整一些，所以优先选择ISR副本列表

面试官：Kafka数据清理机制了解过嘛

候选人：

嗯，了解过~~

Kafka中topic的数据存储在分区上，分区如果文件过大会分段存储segment

每个分段都在磁盘上以索引(xxxx.index)和日志文件(xxxx.log)的形式存储，这样分段的好处是，第一能够减少单个文件内容的大小，查找数据方便，第二方便kafka进行日志清理。

在kafka中提供了两个日志的清理策略：

第一，根据消息的保留时间，当消息保存的时间超过了指定的时间，就会触发清理，默认是168小时（ 7天）

第二是根据topic存储的数据大小，当topic所占的日志文件大小大于一定的阈值，则开始删除最久的消息。这个默认是关闭的

这两个策略都可以通过kafka的broker中的配置文件进行设置

面试官：Kafka中实现高性能的设计有了解过嘛

候选人：

Kafka 高性能，是多方面协同的结果，包括宏观架构、分布式存储、ISR 数据同步、以及高效的利用磁盘、操作系统特性等。主要体现有这么几点：

消息分区：不受单台服务器的限制，可以不受限的处理更多的数据

顺序读写：磁盘顺序读写，提升读写效率

页缓存：把磁盘中的数据缓存到内存中，把对磁盘的访问变为对内存的访问

零拷贝：减少上下文切换及数据拷贝

消息压缩：减少磁盘IO和网络IO

分批发送：将消息打包批量发送，减少网络开销