# 消息队列面试题

## 说说对消息队列的理解

**类似问题：**

* **为什么要用消息队列、消息队列的作用**

通常情况下消息队列用来解决，流量削峰，服务解耦，异步通讯，分布式事务一致性方案。

**一个可靠的消息队列实现起来有3种不同的语义：**

1. 至多消费一次，适用于可容忍数据丢失的场景，且不会重试，比如非关键日志采集。
2. 至少消费一次，不容忍数据丢失的场景，消息链路上的每一步都必须收到确定反馈，存在详细幂等问题，防止二次消费。
3. 仅消费一次，至少消费一次实现去重后，就是仅消费一次，可以通过二阶段提交来实现。

**使用消息队列带来的问题：**

1. 系统可用性降低，增加程序的不稳定性
2. 数据不一致性，增加消息队列会出现数据一致性问题，特别是网络抖动比较大的时候，容易产生数据丢失，或者数据重发，二次消费，持久化失败等等。
3. 复杂性上升，提高了集群维护的代码复杂度。

当然了所有的中间件都会存在这种类似的问题。

## 你用消息队列做过什么？

异步通讯：

1. 对外部提供的第三方接口，没有采用grpc的方式，而是用消息队列进行通讯。利用消息队列获取的数据来驱动下一步执行。

解耦：

1. 上下级服务间的发布订阅，上级服务修改了某一处内容，需要向下级关联服务进行广播通讯，告知其需要一并更改。

削峰：

1. 某些高并发接口瞬间有几万的流量打进来，需要用消息队列进行一个缓冲处理，数据库扛不住那么大的压力。根据业务允许的处理效率，一批一批的消费数据。

## 超热partition 如何解决数据倾斜？

1. 合理设置key值，为key值加上一些随机后缀，或者使用轮询方式，避免一直发送一个partition。
2. 数据压缩，防止大数据量，尽量保证发送消息量级一致。
3. 牺牲掉部分的串行特性、将消息打入到全局partition中。

## 消息积压如何解决

消息积压的原因，消息生产太快，消费速度太慢。要么控制生产者发送速度，要么提高消费者消费效率。一般更倾向于提高消费者效率。

要么就是集群规模和分配资源预估容量不对。

1. **提高消费单条消息的效率**。

原本同步消费的变成异步消费，优化耗时步骤。

1. **增加消费者的数量。**

增加消费者，增加集群规模，启用多线程消费同一个partition。

1. **如果是突然激增的消息**

临时扩充集群规模，应对峰值，如果消息实时性不强可以在中间件上保存起来，等待峰值下降后逐步消费。

1. **如果是经常出现**

生产者速率经常大于消费速率，说明容量预估 与 消费者集群规模设计不对，需要调整集群规模，并增加消费者。

## 线上消息积压如何实际解决？

线上救场的首要任务是将消息缓冲下来，不要堆在那里，找出堆积的partition和producer然后让数据分流。

1. **建立临时的partition，假设10个**
2. **写一个临时的分发消息的 producer程序，用这个producer专门处理处理积压的消息，轮询写入创建好的10partition中。**
3. **开辟了多少partition，就增加多少个consumer节点，每个consumer消费一个partition数据。**

# Kafka

## Kafka 高性能是用什么保证的

Kafka 的高性能最关键点是 顺序I/O + 零拷贝。其它机制：page Cache、批量操作、数据压缩。

**顺序I/O + 零拷贝**

Kafka 高度依赖文件系统存储和缓存消息。每一台broker接收到的数据会不断的写入partition文件末尾，但是不允许修改。顺序I/O避免了重复寻道的时间，从硬件上层面降低了寻址的时间。

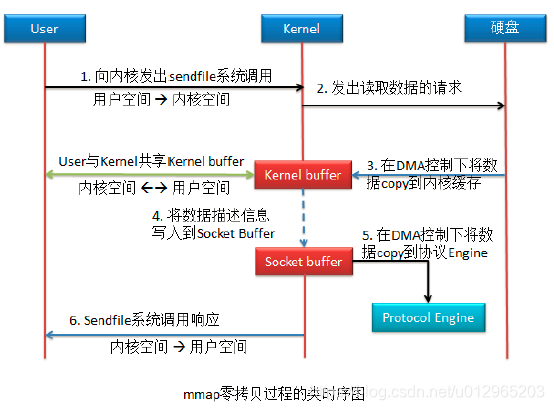
零拷贝减少了CPU消耗和内存带宽占用，减少用户空间与内核空间的拷贝过程，用户上下文和内核上下文的切换，从而提高了系统效率。(mmap - producer ，sendfile - consumer)

**批量操作、数据压缩**

Producer 批量发送，broker批量罗盘，批量可以放大顺序写的优势。同时采用数据压缩，减少数据传输量，提高效率。

**延申知识，如果面试官问到了能说两句。**

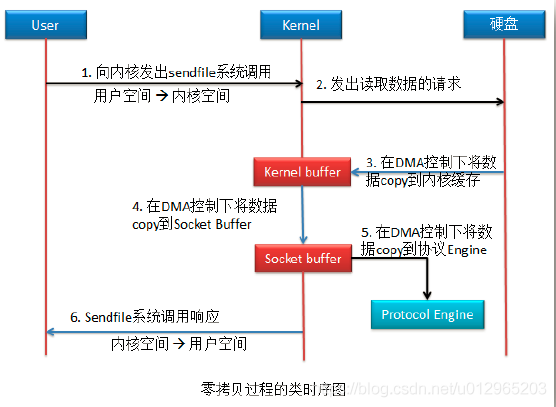
Mmap 零拷贝



Mmap主要用来解决broker网络数据落盘，通过内存映射（与用户共享kernel buffer），完成了网络缓冲区到磁盘缓冲区的写入。

Sendfile 零拷贝

Kafka 用sendfile 实现consumer的拉取数据，直接将数据从内核空间copy到 socket-buffer 一次调用获取数据，避免了频繁的切换与拷贝。



## Kafka 的ISR 是如何工作的？ 如何维护ISR，什么情况会导致partition出入ISR

ISR 是kafka 保证集群partiton可靠性的重要机制。

### 什么是ISR,ISR的作用

ISR ( in-sync-replica ) 是partiton在多broker间保持分区同步的一个列表。每个Leader分区中都维护了一个ISR，leader每次做数据分发复制的时候，都会优先从ISR列表中选择 follower进行同步。

ISR 的引入主要用来解决“当副本消息落后主副本时，重新选举出现的数据丢失问题”。只有Leader 与consumer进行交互，只有Leader负责读写。

### ISR 的工作流程，partition写入消息的过程

1. 当producer 写入消息后，需要等ISR中主分区确认所有Follower分区同步成功。Follower 拉取消息。
2. ISR 里面的分区会定时从主分区拉取消息保持同步，如果长时间未拉取，数据落后太多就会移除ISR列表。
3. ISR 中已经同步的偏移量被称为 LEO（log End Offset），也是消费者可以消费的最新消息。
4. 当主分区挂掉的时候，会从ISR中选举一个新的主分区。

**Producer 写入消息**

1. 生产者从 kafka 集群获取主分区leader的信息。
2. 生产者将消息发送给leader主分区。
3. 主分区leader 将消息写入本地磁盘。
4. Follower 从 leader 拉取消息数据。
5. Follower 将消息写入本地磁盘后向 leader 发送ACK。
6. Leader 收到所有的 follower 的 ACK之后向生产者发送ACK。

**Producer的写入机制等级可以配置成三种：**

1. Producer发出去就算成功，无需ACK应答。
2. Producer发出去，主分区所在磁盘写入成功。
3. Producer发出去，ISR中所有分区都写入成功。

**延申知识点说明**

ISR 的同步机制跟其它带有主从的中间件机制类似，可以归纳为：

1. 主写入就代表成功
2. 主写入，多数从写入代表成功。
3. 著写入，所有从写入代表成功。

**具体什么才算是真正的“写入”。**

1. 中间件写入缓存算写入。
2. 中间件写入到系统磁盘缓冲区算写入。
3. 中间件强制刷新缓冲区到磁盘 fsync 就认为是写入了。

**类似ISR这种机制，不同级别之间可以做一个可靠性 和 性能的取舍。**

### Kafka GC过长会发生什么？ 什么时候分区被移除ISR

面试官想问的是kafka 如何维护 ISR 的。其实就是两个参数控制。一个是落后多少消息，一个是多长时间内没有同步数据。

1. 消息落后太多

Replica.lag.max.message （0.9版本被移除了）

1. 分区多久没同步（同步副本）

分区同步请求 fetch ,由参数 replica.lag.time.max.ms 控制，默认10秒

存在的问题：

1. 两个参数设置的过大会导致消息可靠性降低，存在数据丢失；设置的过小会导致ISR频繁变化。
2. 一般出现太久没有同步，或者同步失败都是由网络、磁盘I/O，负载过高导致的。

## Kafka 的负载均衡策略，生产者分区策略

Kafka 消息产生算法，就2种，1. 轮询策略 2. Hash取余，用key进行取余。简单说一下怎么实现的：

**轮询策略：**

没有指明partition，没有key，就会自动生成一个随机整数每次递增1，将这个值与分区数取余，得到partition。

**消息建保序策略：**

没有指明partition，但是有key，将key的hash值与分区数取余，得到partition值。

**Key的选取 与 partition消息堆积：**

通常情况下，会将某一类业务特征拼接成一个key，通过key取余命中到一个partition上进行顺序消费，保证了针对该业务的顺序消费，但是会出现数据倾斜问题。

可以聊一聊consumer负载的问题，如果某些consumer的内容消费特别慢，有些特别快。假设非常慢的消息都落在了一个parititon中等待消费，就会出现消息堆积的情况。

## 为什么kafka 的从Partition 不能读取

一般的主从模式，都是主负责写入，从负责读取。但是kafka违背了这个规律。Kafka 自身的限制，强制一个partition只能有一个 Consumer，因此Consumer 只需要消费主partition就可以了。

从另一个角度来说，mysql读是因为需要复杂的查询逻辑，和写操作放在一起会增加mysql压力，但是kafka的消息只是单纯的获取字节，所以没必要延用这一套从读机制。

**假设 kafka 允许 follower节点读取**

比如现在有多个consumer，分别从主Partition 和 从Partition上读取数据，那么会出现偏移量如何同步的问题。假设 Consumer从A-parition 读取了 0-100 offset 消息， 另一个Consumer 从B-partition 读取，就只能读取到100 offset以后的消息。Kafka 就无需在不同的 partition 之间协调已读取LEO偏移量了。

会出现不同partition分区偏移量不一致的情况。导致所有分区数据不一致。

## Kafka 的推拉模型，为什么消费者使用PULL ?

明确kafka推拉模型：

Producer 是推 push，Consumer 是拉 pull。

**Producer 和 broker之间为什么不用拉？**

反向思维，如果broker拉取消息，就需要producer在本地通过某种方式将消息保存并等待broker的拉取。这种pull模式一般都是用长轮询间断性的获取数据，如果有很多个生产者长连接数不可控数，长轮询浪费资源，数据可靠性无法保证。

**Producer 为什么要用 push 模型呢？**

1. Producer 对生产消息可控。
2. 更利于broker 进行分区存储，而无需producer保存数据。
3. 避免了持续不断的长连接占用系统资源。

**Consumer 为什么使用 Pull 模式？**

首先pull 拉取模型的核心是可控的消费，kafka不同消费者的速率不同，在pull模型下每个消费者的拉取速率是自己控制的。而broker 到 consumer之间如果是主动推送，这个消费速率是无法保证的。

另外，pull模型还存在竞争问题，所以kafka 强制要求一个Partition 只能有一个consumer，从根本上解决了竞争问题。将优势发挥，将缺点避免。

**延申知识点**

拉模式：

1. 实时性强，即时推送，生产消息可控
2. 容易产生消息积压，因为消费者的消费速率不同。

推模式：

1. 需要长连接保持，基于长轮询来实现。
2. 保证了数据的实时性，不会造成消息积压，消费速率可控。

## Kafka 分区数量过多会怎么样？ 分区是不是越多越好

这道题考查的kafka的分区性能。

**类似问题：**

* Kafka 分区是不是越多越好？ 不是
* 分区数量越高，吞吐性能越强？不是
* 能不能通过增加分区来提高kafka性能？可以，但不是无限增加
* Topic过多会引起什么问题？Topic多分区就多，过多的topic会导致分区数据量不大。

**Producer角度来说**

1. Partition 分区数量过多的话会消耗大量的内存来维护这些分区
2. Partition 分区数量 > broker 数量，会导致不同broker上分布的partition分区数量不均衡，有的节点负载量大，有的负载量小，吞吐性能就会下降。
3. 分区越多，数据丢失的风险就越大，ISR队列拉取同步。

**Consumer角度来说**

1. 因为一个consumer只能消费一个分区，所以partition多，consumer就要多。Consumer要么是多实例对象，要么是多线程消费，无论哪一种开销都很巨大。

## 如何解决 Topic 分区数量过多的问题？

1. 按照业务领域 和 服务来划分topic。最好是某一个业务一个topic。根据业务中的不同功能来划分分区。
2. 增加broker，partition负责计算，broker负责存储。如果有更多的节点，会带来更多的存储通道，I/O负载提高，从而提高吞吐。
3. 让broker的分区数量和 partition持平，每个broker 的partition数量被均分，避免某些broker文件句柄过多/过少。
4. 消费者也不要过多，最好和partition持平，刚好能够全面消费，又不至于浪费资源。

## Kafka 能不能重复消费？

**类似问题、关联问题：**

* **消费组的 commit offset**
* **Kafka 重平衡机制**
* kafka 如何准确的知道message偏移量？
* Kafka 的数据文件格式，分别是用来干什么的？

这个问题是想问“能不能主动重复消费”，消费者能不能消费历史上的消息。从而引出 consumer offset 和 message offset、 message保留策略。

简单说一下不同offset的作用，然后想要重复消费可以根据时间戳找到offset，再从指定的offset进行重复消费。

另外，如果kafka消费者触发了重平衡机制，会导致原来的消费状态过期，所有的消费者暂停，恢复后从上一次提交的offset位置开始重复消费。

**1.consumer offset 消费组offset**

Partition 会为每一个consumer group 保存一个偏移量，记录group消费到的位置。

消费端的offset 信息保存在 consumer中，每一条消费的日志格式喂 topic partition + offsetMetaData，该日志记录着消费的topic、partition、消费组、以及偏移量，偏移量提交时间，提交信息等。

1. **Kafka 的offset 偏移量机制**

Kafka 的message包含3个属性，offset，mesagesize，data。offset偏移量，可以找到消息的 .index索引文件，然后根据索引文件找到对应的数据文件。**如果消费者挂掉后恢复，可以通过offset来确定消费位置继续消费。**

1. **Message 保留策略**

Kafka 可以配置消息的保留时长，如果超过保留时长就会被丢弃再也无法重复消费。

## Kafka Rebalance 重平衡机制

### 重平衡机制发生时机？ kafka 有没有出现消费者暂停的情况 ?

Ranbalance 的核心功能是给消费组 的 消费者从新分配任务的过程。那么每次触发ranbalance 其关联的topic、partition、consumer都会受到影响。

1. **Topic 分区的数量发生变化（从来没出现过，但是如果数量变化会触发）**。一般业务系统中不会增加partition数量，如果需要也是不可避免的重平衡。
2. **消费者数量发生变化，细分如下**

**消费超时：（消费超时 跟 每次消费的数据条目数有关系）**

- max.poll.interval.ms 每隔多长时间就拉取消息。

- max.poll.records 一次拉取出的数据条数。可以根据业务处理耗时合理设置，如果时间设 置的较短，interval.ms 就可以也设置小点，防止超时。

1. **心跳超时（session.timeout.ms）**

- session.timeout.ms 一次session的连接超时时间

- heartbeat.interval.ms 心跳时间，一般为超时时间的1/3，Consumer在被判定为死亡之前，能够发送至少 3 轮的心跳请求

### Consumer如何均分partition ？ Rabalance 过程

消费者通过定期发送心跳，到一个作为组协调者 Group Coordinator 的Broker来保持在消费者组内存活。

消费者通过超时判断session是否过期，如果session过期broker会认为该消费者宕机然后触发重平衡重新均分partition。

以新的消费者加入为例，步骤如下:

1. 新的消费者像协调者上报自己的订阅信息。
2. 协调者强制别的消费者发起一轮 rabalance，上报消费者信息。
3. 协调者从消费者中挑选一个新的 leader。
4. 协调者将订阅信息发送给leader。
5. 协调者同步所有其它消费者。

### Rabalance 带来的影响

1. 在 rebalance 的过程中，kafka所有消费者都不能消费。】
2. 重复消费：如果消费者已经消费了，但还没来得及提交就宕机了。这个时候会发生ranbalance，导致别的消费者再会再一次进行消。消费。
3. 影响性能：rebalance 的过程，一般是在几十毫秒到上百毫秒。这个过程会导致集群处于一种不稳定状态中，影响消费者的吞吐量；

## Kafka 消息如何保证不丢失？

消息可能丢失的三个点：

1. 生产者发送消息异常导致。
2. Broker 写入磁盘异常导致。
3. 消费者 消费异常导致。

### 生产者 消息丢失异常：

1. Kafka生产者异步发送消息后，不管broker是否响应都会立即返回。导致消息可能出现失败。
2. Kafka生产者发送的消息超过大小限制，broker收到后无法进行存储。

**生产者异常的解决方案：**

1. 生产者异步回调，告诉生产者消息成功/失败。
2. 生产者ACK应答机制，acks =all 只有partition的leader 和 follower 副本都同步了之后，才认为消息发送成功了。
3. 生产者设置重试次数，增加重试次数，以确保消不会丢失。
4. 定义本地消息日志表，定义定时任务扫描这个表的自动补偿，并做好监控。

### 2.Broker 消息丢失异常

1. Broker 接收到数据后会将数据进行持久化存储到磁盘，
2. 消息先写入操作系统缓存页，然后由操作系统负责具体的刷盘任务或者使用 fsync强制刷盘。

**Broker 异常的解决方案:**

**Replicatition 多副本机制**

1. Unclean.leader.election.enable = false 数据丢失太多的副本不能选举为leader，防止落后太多的消息数据而引起丢失。
2. Replicatiton.factor >=3 消息分区的副本个数，进来设置为与broker相同的数量，>=3。
3. Min.insync.replicas > 1 消息写入多少副本才算提交，必须大于1，要求leader至少能感知到一个 follower还能保持联系。
4. 以上第 3 点分区副本数 必须 > 写入副本数

### 消费者 消息丢失异常：

1. 消费者配置了 offset 自动提交参数：enable.auto.commit = true
2. 消费者收到了消息，进行了自动提交offset。Kafka以为消费者已经消费了这个消息，但其实刚刚准备处理，还没有处理完成就宕机了。实际没完成，但是已提交。

**消费者解决方法：**

1. 消费者关闭自动提交，采用手动提交offset。Enable.auto.commit = false
2. 消费者多线程处理业务逻辑，等待所有线程处理完之后，手工提交offset。
3. 消费者，消费消息要采用幂等性处理，防止重复消费。

# Nats