# Kafka面试题

## 1、kafka的消费者是pull(拉)还是push(推)模式，这种模式有什么好处？

Kafka 遵循了一种大部分消息系统共同的传统的设计：producer 将消息推送到 broker，consumer 从broker 拉取消息。

优点：pull模式消费者自主决定是否批量从broker拉取数据，而push模式在无法知道消费者消费能力情况下，不易控制推送速度，太快可能造成消费者奔溃，太慢又可能造成浪费。

缺点：如果 broker 没有可供消费的消息，将导致 consumer 不断在循环中轮询，直到新消息到到达。为了避免这点，Kafka 有个参数可以让 consumer阻塞知道新消息到达(当然也可以阻塞知道消息的数量达到某个特定的量这样就可以批量发送)。

## 2、kafka维护消息状态的跟踪方法

Kafka中的Topic 被分成了若干分区，每个分区在同一时间只被一个 consumer 消费。然后再通过offset进行消息位置标记，通过位置偏移来跟踪消费状态。相比其他一些消息队列使用“一个消息被分发到consumer 后 broker 就马上进行标记或者等待 customer 的通知后进行标记”的优点是，避免了通信消息发送后，可能出现的程序奔溃而出现消息丢失或者重复消费的情况。同时也无需维护消息的状态，不用加锁，提高了吞吐量。

## 3、zookeeper对于kafka的作用是什么?

Zookeeper 主要用于在集群中不同节点之间进行通信，在 Kafka 中，它被用于提交偏移量，因此如果节点在任何情况下都失败了，它都可以从之前提交的偏移量中获取，除此之外，它还执行其他活动，如: leader 检测、分布式同步、配置管理、识别新节点何时离开或连接、集群、节点实时状态等等。

## 4、kafka判断一个节点还活着的有那两个条件？

（1）节点必须维护和 ZooKeeper 的连接，Zookeeper 通过心跳机制检查每个节点的连接

（2）如果节点是个 follower,他必须能及时的同步 leader 的写操作，延时不能太久

## 5、讲一讲 kafka 的 ack 的三种机制

request.required.acks 有三个值 0 1 -1(all)，具体如下：

0：生产者不会等待 broker 的 ack，这个延迟最低但是存储的保证最弱当 server 挂掉的时候就会丢数据。

1：服务端会等待 ack 值 leader 副本确认接收到消息后发送 ack 但是如果 leader挂掉后他不确保是否复制完成新 leader 也会导致数据丢失。

-1(all)：服务端会等所有的 follower 的副本受到数据后才会受到 leader 发出的ack，这样数据不会丢失。

## 6、kafka 分布式（不是单机）的情况下，如何保证消息的顺序消费?

Kafka 中发送 1 条消息的时候，可以指定(topic, partition, key) 3 个参数，partiton 和 key 是可选的。

Kafka 分布式的单位是 partition，同一个 partition 用一个 write ahead log 组织，所以可以保证FIFO 的顺序。不同 partition 之间不能保证顺序。因此你可以指定 partition，将相应的消息发往同 1个 partition，并且在消费端，Kafka 保证1 个 partition 只能被1 个 consumer 消费，就可以实现这些消息的顺序消费。

另外，你也可以指定 key（比如 order id），具有同 1 个 key 的所有消息，会发往同 1 个partition，那这样也实现了消息的顺序消息。

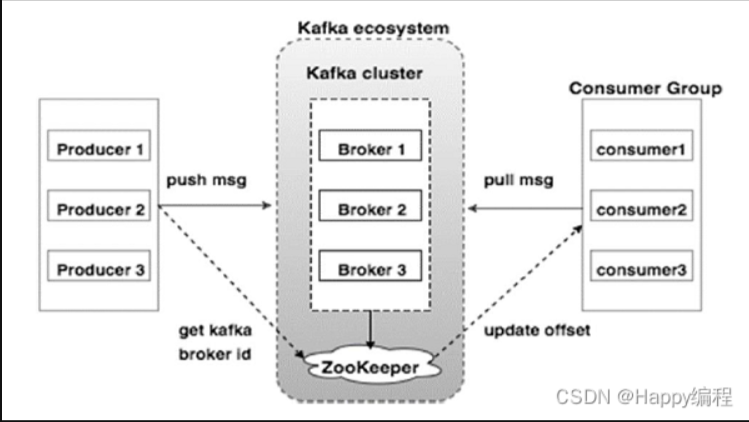
## 7、kafka 如何不消费重复数据？比如扣款，我们不能重复的扣。

这个问题换种问法，就是kafka如何保证消息的幂等性。对于消息队列来说，出现重复消息的概率还是挺大的，不能完全依赖消息队列，而是应该在业务层进行数据的一致性幂等校验。

比如你处理的数据要写库（mysql，redis等），你先根据主键查一下，如果这数据都有了，你就别插入了，进行一些消息登记或者update等其他操作。另外，数据库层面也可以设置唯一健，确保数据不要重复插入等 。一般这里要求生产者在发送消息的时候，携带全局的唯一id。

## 8、讲一下kafka集群的组成？

kafka的集群图如下：



Broker（代理）

Kafka集群通常由多个代理组成以保持负载平衡。 Kafka代理是无状态的，所以他们使用ZooKeeper来维护它们的集群状态。 一个Kafka代理实例可以每秒处理数十万次读取和写入，每个Broker可以处理TB的消息，而没有性能影响。 Kafka经纪人领导选举可以由ZooKeeper完成。

ZooKeeper

ZooKeeper用于管理和协调Kafka代理。 ZooKeeper服务主要用于通知生产者和消费者Kafka系统中存在任何新代理或Kafka系统中代理失败。 根据Zookeeper接收到关于代理的存在或失败的通知，然后生产者和消费者采取决定并开始与某些其他代理协调他们的任务。

Producers（生产者

生产者将数据推送给经纪人。 当新代理启动时，所有生产者搜索它并自动向该新代理发送消息。 Kafka生产者不等待来自代理的确认，并且发送消息的速度与代理可以处理的一样快。

Consumers（消费者）

因为Kafka代理是无状态的，这意味着消费者必须通过使用分区偏移来维护已经消耗了多少消息。 如果消费者确认特定的消息偏移，则意味着消费者已经消费了所有先前的消息。 消费者向代理发出异步拉取请求，以具有准备好消耗的字节缓冲区。 消费者可以简单地通过提供偏移值来快退或跳到分区中的任何点。 消费者偏移值由ZooKeeper通知。

## 9、kafka是什么？

Kafka是一种高吞吐量、分布式、基于发布/订阅的消息系统，最初由LinkedIn公司开发，使用Scala语言编写，目前是Apache的开源项目。

broker： Kafka服务器，负责消息存储和转发

topic：消息类别，Kafka按照topic来分类消息

partition： topic的分区，一个topic可以包含多个partition， topic 消息保存在各个partition上4. offset：消息在日志中的位置，可以理解是消息在partition上的偏移量，也是代表该消息的唯一序号

Producer：消息生产者

Consumer：消息消费者

Consumer Group：消费者分组，每个Consumer必须属于一个group

Zookeeper：保存着集群 broker、 topic、 partition等meta 数据；另外，还负责broker故障发现， partition leader选举，负载均衡等功能

## 10、partition的数据文件（offffset，MessageSize，data）

partition中的每条Message包含了以下三个属性： offset，MessageSize，data，其中offset表示Message在这个partition中的偏移量，offset不是该Message在partition数据文件中的实际存储位置，而是逻辑上一个值，它唯一确定了partition中的一条Message，可以认为offset是partition中Message的 id； MessageSize表示消息内容data的大小；data为Message的具体内容。

## 11、kafka如何实现数据的高效读取？（顺序读写、分段命令、二分查找）

Kafka为每个分段后的数据文件建立了索引文件，文件名与数据文件的名字是一样的，只是文件扩展名为index。 index文件中并没有为数据文件中的每条Message建立索引，而是采用了稀疏存储的方式，每隔一定字节的数据建立一条索引。这样避免了索引文件占用过多的空间，从而可以将索引文件保留在内存中。

## 12、 Kafka 消费者端的 Rebalance 操作什么时候发生？

同一个 consumer 消费者组 group.id 中，新增了消费者进来，会执行 Rebalance 操作

消费者离开当期所属的 consumer group组。比如宕机

分区数量发生变化时(即 topic 的分区数量发生变化时)

消费者主动取消订阅

Rebalance的过程如下：

第一步：所有成员都向coordinator发送请求，请求入组。一旦所有成员都发送了请求，coordinator会从中选择一个consumer担任leader的角色，并把组成员信息以及订阅信息发给leader

第二步：leader开始分配消费方案，指明具体哪个consumer负责消费哪些topic的哪些partition。一旦完成分配，leader会将这个方案发给coordinator。coordinator接收到分配方案之后会把方案发给各个consumer，这样组内的所有成员就都知道自己应该消费哪些分区了。

所以对于Rebalance来说，Coordinator起着至关重要的作用

## 13、Kafka 中的ISR(InSyncRepli)、OSR(OutSyncRepli)、AR(AllRepli)代表什么？

答：kafka中与leader副本保持一定同步程度的副本（包括leader）组成ISR。与leader滞后太多的副本组成OSR。分区中所有的副本通称为AR。

ISR : 速率和leader相差低于10秒的follower的集合

OSR : 速率和leader相差大于10秒的follower

AR : 全部分区的follower

## 14、Kafka 中的HW、LEO等分别代表什么？

答：HW：高水位，指消费者只能拉取到这个offset之前的数据

LEO：标识当前日志文件中下一条待写入的消息的offset，大小等于当前日志文件最后一条消息的offset+1.

## 15、Kafka的那些设计让它有如此高的性能?

1.kafka是分布式的消息队列

2.对log文件进行了segment,并对segment创建了索引

3.(对于单节点)使用了顺序读写,速度能够达到600M/s

4.引用了zero拷贝,在os系统就完成了读写操做

## 16、Kafka为什么不支持读写分离？

1、 这其实是分布式场景下的通用问题，因为我们知道CAP理论下，我们只能保证C（一致性）和A（可用性）取其一，如果支持读写分离，那其实对于一致性的要求可能就会有一定折扣，因为通常的场景下，副本之间都是通过同步来实现副本数据一致的，那同步过程中肯定会有时间的消耗，如果支持了读写分离，就意味着可能的数据不一致，或数据滞后。

2、 Leader/Follower模型并没有规定Follower副本不可以对外提供读服务。很多框架都是允许这么做的，只是 Kafka最初为了避免不一致性的问题，而采用了让Leader统一提供服务的方式。

3、 不过，自Kafka 2.4之后，Kafka提供了有限度的读写分离，也就是说，Follower副本能够对外提供读服务。

## 17、分区Leader选举策略有几种？

分区的Leader副本选举对用户是完全透明的，它是由Controller独立完成的。你需要回答的是，在哪些场景下，需要执行分区Leader选举。每一种场景对应于一种选举策略。

1、 OfflinePartition Leader选举：每当有分区上线时，就需要执行Leader选举。所谓的分区上线，可能是创建了新分区，也可能是之前的下线分区重新上线。这是最常见的分区Leader选举场景。

2、 ReassignPartition Leader选举：当你手动运行Kafka-reassign-partitions命令，或者是调用Admin的alterPartitionReassignments方法执行分区副本重分配时，可能触发此类选举。假设原来的AR是[1，2，3]，Leader是1，当执行副本重分配后，副本集合AR被设置成[4，5，6]，显然，Leader必须要变更，此时会发生Reassign Partition Leader选举。

3、 PreferredReplicaPartition Leader选举：当你手动运行Kafka-preferred-replica-election命令，或自动触发了Preferred Leader选举时，该类策略被激活。所谓的Preferred Leader，指的是AR中的第一个副本。比如AR是[3，2，1]，那么，Preferred Leader就是3。

4、 ControlledShutdownPartition Leader选举：当Broker正常关闭时，该Broker上的所有Leader副本都会下线，因此，需要为受影响的分区执行相应的Leader选举。

这4类选举策略的大致思想是类似的，即从AR中挑选首个在ISR中的副本，作为新Leader。

## 18、请简述下你在哪些场景下会选择 Kafka？

•日志收集：一个公司可以用Kafka可以收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer，例如hadoop、HBase、Solr等。

•消息系统：解耦和生产者和消费者、缓存消息等。

•用户活动跟踪：Kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后订阅者通过订阅这些topic来做实时的监控分析，或者装载到hadoop、数据仓库中做离线分析和挖掘。

•运营指标：Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告。

•流式处理：比如spark streaming和 Flink

## 19、请谈一谈 Kafka 数据一致性原理

一致性就是说不论是老的 Leader 还是新选举的 Leader，Consumer 都能读到一样的数据。

假设分区的副本为3，其中副本0是 Leader，副本1和副本2是 follower，并且在 ISR 列表里面。虽然副本0已经写入了 Message4，但是 Consumer 只能读取到 Message2。因为所有的 ISR 都同步了 Message2，只有 High Water Mark 以上的消息才支持 Consumer 读取，而 High Water Mark 取决于 ISR 列表里面偏移量最小的分区，对应于上图的副本2，这个很类似于木桶原理。

这样做的原因是还没有被足够多副本复制的消息被认为是“不安全”的，如果 Leader 发生崩溃，另一个副本成为新 Leader，那么这些消息很可能丢失了。如果我们允许消费者读取这些消息，可能就会破坏一致性。试想，一个消费者从当前 Leader（副本0） 读取并处理了 Message4，这个时候 Leader 挂掉了，选举了副本1为新的 Leader，这时候另一个消费者再去从新的 Leader 读取消息，发现这个消息其实并不存在，这就导致了数据不一致性问题。

当然，引入了 High Water Mark 机制，会导致 Broker 间的消息复制因为某些原因变慢，那么消息到达消费者的时间也会随之变长（因为我们会先等待消息复制完毕）。延迟时间可以通过参数 replica.lag.time.max.ms 参数配置，它指定了副本在复制消息时可被允许的最大延迟时间。

## 20、Kafka 缺点？

由于是批量发送，数据并非真正的实时；

•对于mqtt协议不支持；

•不支持物联网传感数据直接接入；

•仅支持统一分区内消息有序，无法实现全局消息有序；

•监控不完善，需要安装插件；

•依赖zookeeper进行元数据管理；