**架构师面试题之Kafka专题篇**

**1、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 groupZookeeper：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

图示

描述已自动生成

**2.Kafka 的设计时什么样的呢**

Kafka 将消息以 topic 为单位进行归纳 将向 Kafka topic 发布消息的程序成为 producers. 将预订 topics 并消费消息的程序成为 consumer. Kafka 以集群的方式运行，可以由一个或多个服务组成，每个服务叫做一个 broker. producers 通过网络将消息发送到 Kafka 集群，集群向消费者提供消息。

**3.为什么要使用 kafka，为什么要使用消息队列?**

**缓冲和削峰** ：上游数据时有突发流量，下游可能扛不住，或者下游没有足够多的机器来保证冗余，kafka在中间可以起到一个缓冲的作用，把消息暂存在kafka中，下游服务就可以按照自己的节奏进行慢慢处理。  
**解耦和扩展性** ：项目开始的时候，并不能确定具体需求。消息队列可以作为一个接口层，解耦重要的业务流程。只需要遵守约定，针对数据编程即可获取扩展能力。**冗余** ：可以采用一对多的方式，一个生产者发布消息，可以被多个订阅topic的服务消费到，供多个毫无关联的业务使用。**健壮性** ：消息队列可以堆积请求，所以消费端业务即使短时间死掉，也不会影响主要业务的正常进行。**异步通信** ：很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

**4.数据传输的事物定义有哪三种？**

数据传输的事务定义通常有以下三种级别：（ 1）最多一次: 消息不会被重复发送，最多被传输一次，但也有可能一次不传输 （ 2）最少一次: 消息不会被漏发送，最少被传输一次，但也有可能被重复传输. （ 3）精确的一次（ Exactly once） : 不会漏传输也不会重复传输,每个消息都传输被一次而且仅仅被传输一次，这是大家所期望的

**5.Kafka 判断一个节点是否还活着有那两个条件？**

（ 1）节点必须可以维护和 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

**6.Kafka中的ISR、AR又代表什么？ISR的伸缩又指什么**

ISR:In-Sync Replicas 副本同步队列  
AR:Assigned Replicas 所有副本ISR是由leader维护，follower从leader同步数据有一些延迟（包括延迟时间replica.lag.time.max.ms和延迟条数replica.lag.max.messages两个维度, 当前最新的版本0.10.x中只支持replica.lag.time.max.ms这个维度），任意一个超过阈值都会把follower剔除出ISR, 存入OSR（Outof-Sync Replicas）列表，新加入的follower也会先存放在OSR中。AR=ISR+OSR。

**7.kafka中的broker 是干什么的**

broker 是消息的代理，Producers往Brokers里面的指定Topic中写消息，Consumers从Brokers里面拉取指定Topic的消息，然后进行业务处理，broker在中间起到一个代理保存消息的中转站。

**8.producer 是否直接将数据发送到broker 的 leader(主节点)？**

 producer 直接将数据发送到 broker 的 leader(主节点)，不需要在多个节点进行分发，为了帮助 producer 做到这点，所有的 Kafka 节点都可以及时的告知:哪些节点是活动的，目标topic 目标分区的 leader 在哪。这样 producer 就可以直接将消息发送到目的地了

**9.什么情况下一个 broker 会从 isr中踢出去**

leader会维护一个与其基本保持同步的Replica列表，该列表称为ISR(in-sync Replica)，每个Partition都会有一个ISR，而且是由leader动态维护 ，如果一个follower比一个leader落后太多，或者超过一定时间未发起数据复制请求，则leader将其重ISR中移除 。

**10、 Kafa consumer 是否可以消费指定分区消息？**

 Kafka consumer 消费消息时，向 broker 发出"fetch"请求去消费特定分区的消息， consumer指定消息在日志中的偏移量（ offset），就可以消费从这个位置开始的消息， customer 拥有了 offset 的控制权，可以向后回滚去重新消费之前的消息，这是很有意义的

**11、 Kafka 消息是采用 Pull 模式，还是 Push 模式？**

Kafka 最初考虑的问题是， customer 应该从 brokes 拉取消息还是 brokers 将消息推送到consumer，也就是 pull 还 push。在这方面， 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 一些消息系统比如 Scribe 和 Apache Flume 采用了 push 模式，将消息推送到下游的consumer。这样做有好处也有坏处：由 broker 决定消息推送的速率，对于不同消费速率的consumer 就不太好处理了。消息系统都致力于让 consumer 以最大的速率最快速的消费消息，但不幸的是， push 模式下，当 broker 推送的速率远大于 consumer 消费的速率时，consumer 恐怕就要崩溃了。最终 Kafka 还是选取了传统的 pull 模式 Pull 模式的另外一个好处是 consumer 可以自主决定是否批量的从 broker 拉取数据。Push模式必须在不知道下游 consumer 消费能力和消费策略的情况下决定是立即推送每条消息还是缓存之后批量推送。如果为了避免 consumer 崩溃而采用较低的推送速率，将可能导致一次只推送较少的消息而造成浪费。Pull 模式下， consumer 就可以根据自己的消费能力去决定这些策略 Pull 有个缺点是，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 t 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)

**12.Kafka 存储在硬盘上的消息格式是什么？**

 消息由一个固定长度的头部和可变长度的字节数组组成。头部包含了一个版本号和 CRC32校验码。

1. 消息长度: 4 bytes (value: 1+4+n)
2. 版本号: 1 byte 3CRC
3. 校验码: 4 bytes
4. 具体的消息: n bytes

**13.Kafka 高效文件存储设计特点：**

(1).Kafka 把 topic 中一个 parition 大文件分成多个小文件段，通过多个小文件段，就容易定期清除或删除已经消费完文件，减少磁盘占用。(2).通过索引信息可以快速定位 message 和确定 response 的最大大小。(3).通过 index 元数据全部映射到 memory，可以避免 segment file 的 IO 磁盘操作。(4).通过索引文件稀疏存储，可以大幅降低 index 文件元数据占用空间大小。

**14.Kafka 与传统消息系统之间有三个关键区别**

(1).Kafka 持久化日志，这些日志可以被重复读取和无限期保留(2).Kafka 是一个分布式系统：它以集群的方式运行，可以灵活伸缩，在内部通过复制数据提升容错能力和高可用性(3).Kafka 支持实时的流式处理

**15.Kafka 创建 Topic 时如何将分区放置到不同的 Broker 中**

(1).副本因子不能大于 Broker 的个数；(2).第一个分区（编号为 0）的第一个副本放置位置是随机从 brokerList 选择的；(3).其他分区的第一个副本放置位置相对于第 0 个分区依次往后移。也就是如果我们有 5 个Broker， 5 个分区，假设第一个分区放在第四个 Broker 上，那么第二个分区将会放在第五个 Broker 上；第三个分区将会放在第一个 Broker 上；第四个分区将会放在第二个Broker 上，依次类推；(4).剩余的副本相对于第一个副本放置位置其实是由 nextReplicaShift 决定的，而这个数也是随机产生的

**16.Kafka 新建的分区会在哪个目录下创建**

在启动 Kafka 集群之前，我们需要配置好 log.dirs 参数，其值是 Kafka 数据的存放目录，这个参数可以配置多个目录，目录之间使用逗号分隔，通常这些目录是分布在不同的磁盘上用于提高读写性能。当然我们也可以配置 log.dir 参数，含义一样。只需要设置其中一个即可。如果 log.dirs 参数只配置了一个目录，那么分配到各个 Broker 上的分区肯定只能在这个目录下创建文件夹用于存放数据。但是如果 log.dirs 参数配置了多个目录，那么 Kafka 会在哪个文件夹中创建分区目录呢？答案是：Kafka 会在含有分区目录最少的文件夹中创建新的分区目录，分区目录名为 Topic名+分区 ID。注意，是分区文件夹总数最少的目录，而不是磁盘使用量最少的目录！也就是说，如果你给 log.dirs 参数新增了一个新的磁盘，新的分区目录肯定是先在这个新的磁盘上创建直到这个新的磁盘目录拥有的分区目录不是最少为止。

**17.partition 的数据如何保存到硬盘**

topic 中的多个 partition 以文件夹的形式保存到 broker，每个分区序号从 0 递增，且消息有序 Partition 文件下有多个 segment（ xxx.index， xxx.log） segment 文件里的 大小和配置文件大小一致可以根据要求修改 默认为 1g 如果大小大于 1g 时，会滚动一个新的 segment 并且以上一个 segment 最后一条消息的偏移量命名

**18、讲讲 kafka 维护消费状态跟踪的方法**

大部分消息系统在 broker 端的维护消息被消费的记录：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。这样也可以在消息在消费后立马就删除以减少空间占用。但是这样会不会有什么问题呢？如果一条消息发送出去之后就立即被标记为消费过的， 一旦 consumer 处理消息时失败了（比如程序崩溃）消息就丢失了。为了解决这个问题， 很多消息系统提供了另外一个个功能：当消息被发送出去之后仅仅被标记为已发送状态，当接到 consumer 已经消费成功的通知后才标记为已被消费的状态。这虽然解决了消息丢失的问题，但产生了新问题，首先如果 consumer处理消息成功了但是向 broker 发送响应时失败了，这条消息将被消费两次。第二个问题时，broker 必须维护每条消息的状态，并且每次都要先锁住消息然后更改状态然后释放锁。这样麻烦又来了，且不说要维护大量的状态数据，比如如果消息发送出去但没有收到消费成功的通知，这条消息将一直处于被锁定的状态，Kafka 采用了不同的策略。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。这意味着每个分区被消费的消息在日志中的位置仅仅是一个简单的整数：oﬀset。这样就很容易标记每个分区消费状态就很容易了，仅仅需要一个整数而已。这样消费状态的跟踪就很简单了。这带来了另外一个好处：consumer 可以把 oﬀset 调成一个较老的值，去重新消费老的消息。这对传统的消息系统来说看起来有些不可思议，但确实是非常有用的，谁规定了一条消息只能被消费一次呢？

**19.kafka 的 ack 机制**

request.required.acks 有三个值 0 1 -1 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：同样在 1 的基础上 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

**20.Kafka 的消费者如何消费数据**

消费者每次消费数据的时候，消费者都会记录消费的物理偏移量（ offset）的位置等到下次消费时，他会接着上次位置继续消费

**21.消费者负载均衡策略**

一个消费者组中的一个分片对应一个消费者成员，他能保证每个消费者成员都能访问，如果组中成员太多会有空闲的成员

**22.数据有序**

一个消费者组里它的内部是有序的 消费者组与消费者组之间是无序的

**23.kafaka 生产数据时数据的分组策略**

生产者决定数据产生到集群的哪个 partition 中 每一条消息都是以（ key， value）格式 Key是由生产者发送数据传入 所以生产者（ key）决定了数据产生到集群的哪个 partition

**24.Kafka中的消息是否会丢失和重复消费？**

要确定Kafka的消息是否丢失或重复，从两个方面分析入手：消息发送和消息消费。  
1、消息发送         Kafka消息发送有两种方式：同步（sync）和异步（async），默认是同步方式，可通过producer.type属性进行配置。Kafka通过配置request.required.acks属性来确认消息的生产：0---表示不进行消息接收是否成功的确认；1---表示当Leader接收成功时确认；-1---表示Leader和Follower都接收成功时确认；综上所述，有6种消息生产的情况，下面分情况来分析消息丢失的场景：（1）acks=0，不和Kafka集群进行消息接收确认，则当网络异常、缓冲区满了等情况时，消息可能丢失；（2）acks=1、同步模式下，只有Leader确认接收成功后但挂掉了，副本没有同步，数据可能丢失；2、消息消费Kafka消息消费有两个consumer接口，Low-level API和High-level API：Low-level API：消费者自己维护offset等值，可以实现对Kafka的完全控制；High-level API：封装了对parition和offset的管理，使用简单；如果使用高级接口High-level API，可能存在一个问题就是当消息消费者从集群中把消息取出来、并提交了新的消息offset值后，还没来得及消费就挂掉了，那么下次再消费时之前没消费成功的消息就“诡异”的消失了；解决办法：        针对消息丢失：同步模式下，确认机制设置为-1，即让消息写入Leader和Follower之后再确认消息发送成功；异步模式下，为防止缓冲区满，可以在配置文件设置不限制阻塞超时时间，当缓冲区满时让生产者一直处于阻塞状态；        针对消息重复：将消息的唯一标识保存到外部介质中，每次消费时判断是否处理过即可。

**25.Kafka中是怎么体现消息顺序性的？**

kafka每个partition中的消息在写入时都是有序的，消费时，每个partition只能被每一个group中的一个消费者消费，保证了消费时也是有序的。整个topic不保证有序。如果为了保证topic整个有序，那么将partition调整为1.

**26.kafka如何实现延迟队列？**

Kafka并没有使用JDK自带的Timer或者DelayQueue来实现延迟的功能，而是基于时间轮自定义了一个用于实现延迟功能的定时器（SystemTimer）。JDK的Timer和DelayQueue插入和删除操作的平均时间复杂度为O(nlog(n))，并不能满足Kafka的高性能要求，而基于时间轮可以将插入和删除操作的时间复杂度都降为O(1)。时间轮的应用并非Kafka独有，其应用场景还有很多，在Netty、Akka、Quartz、Zookeeper等组件中都存在时间轮的踪影。底层使用数组实现，数组中的每个元素可以存放一个TimerTaskList对象。TimerTaskList是一个环形双向链表，在其中的链表项TimerTaskEntry中封装了真正的定时任务TimerTask.Kafka中到底是怎么推进时间的呢？Kafka中的定时器借助了JDK中的DelayQueue来协助推进时间轮。具体做法是对于每个使用到的TimerTaskList都会加入到DelayQueue中。Kafka中的TimingWheel专门用来执行插入和删除TimerTaskEntry的操作，而DelayQueue专门负责时间推进的任务。再试想一下，DelayQueue中的第一个超时任务列表的expiration为200ms，第二个超时任务为840ms，这里获取DelayQueue的队头只需要O(1)的时间复杂度。如果采用每秒定时推进，那么获取到第一个超时的任务列表时执行的200次推进中有199次属于“空推进”，而获取到第二个超时任务时有需要执行639次“空推进”，这样会无故空耗机器的性能资源，这里采用DelayQueue来辅助以少量空间换时间，从而做到了“精准推进”。Kafka中的定时器真可谓是“知人善用”，用TimingWheel做最擅长的任务添加和删除操作，而用DelayQueue做最擅长的时间推进工作，相辅相成。