第三版：Kafka 40 道

什么是 Kafka?

Kafka 是**分布式-订阅消息系统**，它最初是由 LinkedIn 公司开发的，之后成为 Apache 项目的一部分，Kafka 是一个分布式，可划分的，冗余备份的持久性的日志服务，它主要用于处理流式数据。

Kafka 中有哪几个组件?

主题(Topic)：Kafka 主题是一堆或一组消息。

生产者(Producer)：在 Kafka，生产者通信以及向 Kafka 主题消息。

消费者(Consumer)：Kafka 消费者订阅了一个主题，并且还从主题中读取和处理消息。

经纪人(Brokers)：在管理主题中的消息存储时，我们使用 Kafka Brokers。

什么是消费者或用户?

Kafka 消费者订阅一个主题，并读取和处理来自该主题的消息。此外，有了消费者组的名字，消费者就给自己贴上了标签。换句话说，在每个订阅使用者组中，到主题的每个记录都传递到一个使用者实例。确保使用者实例可能位于单独的进程或单独的计算机上。

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。

什么是生产者?

生产者的主要作用是将数据到他们选择的主题上。基本上，它的职责是选择要分配给主题内分区的记录。

什么是消费者组?

消费者组的概念是 Apache Kafka 独有的。基本上，每个 Kafka 消费群体都由一个或多个共同消费一组订阅主题的消费者组成。

偏移的作用是什么?

给分区中的消息提供了一个顺序 ID 号，我们称之为偏移量。因此，为了唯一地识别分区中的每条消息，我们使用这些偏移量。

Kafka 系统工具有哪些类型?

1、 Kafka 迁移工具：它有助于将代理从一个版本迁移到另一个版本。

2、 Mirror Maker：Mirror Maker 工具有助于将一个 Kafka 集群的镜像提供给另一个。

3、 消费者检查:对于指定的主题集和消费者组，它显示主题，分区，所有者。

Kafka 为什么那么快?

1. Cache Filesystem Cache PageCache 缓存
2. 顺序写 由于现代的操作系统提供了预读和写技术，磁盘的顺序写大多数情况下比随机写内存还要快。
3. Zero-copy 零拷技术减少拷贝次数
4. Batching of Messages 批量量处理。合并小的请求，然后以流的方式进行交互，直顶网络上限。
5. Pull 拉模式 使用拉模式进行消息的获取消费，与消费端处理能力相符。

Kafka 的 message 格式是什么?

一个 Kafka 的 Message 由一个**固定长度的 header** 和一个**变长的消息体 body** 组成

1. header 部分 由一个字节的 magic(文件格式)和四个字节的 CRC32(用于判断 body 消息体是否正常)构成。
2. 当 magic 的值为 1 的时候，会在 magic 和 crc32 之间多一个字节的数据：attributes(保存一些相关属性，比如是否压缩、压缩格式等等);

3. 如果 magic 的值为 0，那么不存在 attributes 属性
4. body 部分 由 N 个字节构成的一个消息体，包含了具体的 key/value 消息

Kafka 可以接收的消息最大为多少?

Kafka 可以接收的最大消息大小约为 1000000 字节。

Kafka 的优点有那些?

1. 高吞吐量：我们在 Kafka 中不需要任何大型硬件，因为它能够处理高速和大容量数据。此外，它还可以支持每秒数千条消息的消息吞吐量。
2. 低延迟：Kafka 可以轻松处理这些消息，具有毫秒级的极低延迟，这是大多数新用例所要求的。
3. 容错：Kafka 能够抵抗集群中的节点/机器故障。
4. 耐久性：由于 Kafka 支持消息复制，因此消息永远不会丢失。这是耐久性背后的原因之一。
5. 可扩展性：Kafka 可以扩展，而不需要通过添加额外的节点而在运行中造成任何停机。

为什么要使用 Kafka? 为什么要使用消息队列?

1. 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。

2. 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。

3. 冗余：可以采用一对多的方式，一个生产者消息，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。

4. 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。

5. 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。

Kafka 存在那些局限性?

1、 没有完整的监控工具集

2、 消息调整的问题

3、 不支持通配符主题选择

4、 速度问题

Leader 和 Follower 的概念是什么?

在 Kafka 的每个分区中，都有一个服务器充当 leader，0 到多个服务器充当 follower 的角色。

为什么要使用 Apache Kafka 集群?

为了克服收集大量数据和分析收集数据的挑战，我们需要一个消息队列系统。因此 Apache Kafka 应运而生。其好处是：只需存储/发送事件以进行实时处理，就可以跟踪 Web 活动。通过这一点，我们可以发出警报并报告操作指标。此外，我们可以将数据转换为标准格式。此外，它允许对主题的流数据进行连续处理。由于它的广泛使用，它秒杀了竞品，如 ActiveMQ，RabbitMQ 等。

Kafka 集群中保留期的目的是什么?

保留期限保留了 Kafka 群集中的所有已记录。它不会检查它们是否已被消耗。此外，可以通过使用保留期的配置设置来丢弃记录。而且，它可以释放一些空间。

Kafka 和 Flume 之间的主要区别是什么?

工具类型

Apache Kafka 是面向多个生产商和消费者的通用工具。
Apache Flume 是特定应用程序的专用工具。

复制功能

Apache Kafka 可以复制事件；
Apache Flume 不复制事件。

Apache Kafka 是分布式流处理平台吗？如果是，你能用它做什么？

Kafka 是一个流处理平台。它可以完成以下工作：

1、 轻松推送记录

2、 可以存储大量记录，而不会出现任何存储问题

3、 它还可以在记录进入时对其进行处理。

流 API 的作用是什么？

一种允许应用程序充当流处理器的 API，它还使用一个或多个主题的输入流，并生成一个输出流到一个或多个输出主题，此外，有效地将输入流转换为输出流，我们称之为流 API。

消费者 API 的作用是什么？

允许应用程序订阅一个或多个主题并处理生成给它们的记录流的 API，我们称之为消费者 API。

连接器 API 的作用是什么？

一个允许运行和构建可重用的生产者或消费者的 API，将 Kafka 主题连接到现有的应用程序或数据系统，我们称之为连接器 API。

Kafka 中的 zookeeper 起到什么作用？可以不用 zookeeper 吗？

zookeeper 是一个分布式的协调组件，早期版本的 Kafka 用 zk 做 meta 信息存储，consumer 的消费状态，group 的管理以及 offset 的值。考虑到 zookeeper 本身的一些因素以及整个架构较大概率存在单点问题，新版本中逐渐弱化了 zookeeper 的作用。新的 consumer 使用了 Kafka 内部的 group coordination 协议，也减少了对 zookeeper 的依赖，

但是 broker 依然依赖于 zookeeper，zookeeper 在 Kafka 中还用来选举 controller 和检测 broker 是否存活等等。

没有 zookeeper 可以使用 Kafka 吗？

绕过 Zookeeper 并直接连接到 Kafka 服务器是不可以的，所以答案是否定的。如果以某种方式，使 ZooKeeper 关闭，则无法为任何客户端请求提供服务。

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；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，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。

AR：Assigned Replicas 所有副本；

副本和 ISR 扮演什么角色?

基本上，复制日志的节点列表就是副本。特别是对于特定的分区。但是，无论他们是否扮演 leader 的角色，他们都是如此。此外，ISR 指的是同步副本。在定义 ISR 时，它是一组与 leader 同步的消息副本。

Kafka Follower 如何与 Leader 同步数据?

Kafka 的复制机制既不是完全的同步复制，也不是单纯的异步复制。完全同步复制要求 All Alive Follower 都复制完，这条消息才会被认为 commit，这种复制方式极大的影响了吞吐率。而异步复制方式下，Follower 异步的从 Leader 复制数据，数据只要被 Leader 写入 log 就被认为已经 commit，这种情况下，如果 leader 挂掉，会丢失数据，Kafka 使用 ISR 的方式很好的均衡了确保数据不丢失以及吞吐率。Follower 可以批量的从 Leader 复制数据，而且 Leader 充分利用磁盘顺序读以及 send file(zero copy) 机制，这样极大的提高复制性能，内部批量写磁盘，大幅减少了 Follower 与 Leader 的消息量差。

为什么 Kafka 的复制至关重要?

由于复制，我们可以确保的消息不会丢失，并且可以在发生任何机器错误、程序错误或频繁的软件升级时使用。

什么是 Kafka 中的地域复制?

对于我们的集群，Kafka MirrorMaker 提供地理复制。基本上，消息是通过 MirrorMaker 跨多个数据中心或云区域复制的。因此，它可以在主动/被动场景中用于备份和恢复；也可以将数据放在离用户更近的位置，或者支持数据位置要求。

什么是多租户?

我们可以轻松地将 Kafka 部署为多租户解决方案。但是，通过配置主题可以生成或使用数据，可以启用多租户。此外，它还为配额提供操作支持。

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 。

如果 Leader Crash 时，ISR 为空怎么办

Kafka 在 Broker 端提供了一个配置参数：unclean.leader.election,这个参数有两个值：

true（默认）：允许不同步副本成为 leader，由于不同步副本的消息较为滞后，此时成为 leader，可能会出现消息不一致的情况。

false：不允许不同步副本成为 leader，此时如果发生 ISR 列表为空，会一直等待旧 leader 恢复，降低了可用性。

副本长时间不在 ISR 中，这意味着什么?

意味着 follower 不能像 leader 收集数据那样快速地获取数据。

Kafka Producer 如何优化写入速度?

1. 增加线程
2. 提高 batch.size
3. 增加更多 producer 实例
4. 增加 partition 数
5. 设置 acks=-1 时，如果延迟增大：可以增大 num.replica.fetchers（follower 同步数据的线程数）来调解；
6. 跨数据中心的传输：增加 socket 缓冲区设置以及 OS tcp 缓冲区设置。

Kafka Producer API 的作用是什么?

允许应用程序将记录流到一个或多个 Kafka 主题的 API 就是我们所说的 Producer API。

生产者中，什么情况下会发生 QueueFullException?

每当 Kafka 生产者试图以代理的身份在当时无法处理的速度发送消息时，通常都会发生 QueueFullException。但是，为了协作处理增加的负载，用户需要添加足够的代理，因为生产者不会阻止。

Kafka Producer 写数据，ACK 为 0，1，-1 时分别代表什么？

1（默认） 数据发送到 Kafka 后，经过 leader 成功接收消息的的确认，就算是发送成功了。在这种情况下，如果 leader 宕机了，则会丢失数据。

0 生产者将数据发送出去就不管了，不去等待任何返回。这种情况下数据传输效率最高，但是数据可靠性确是最低的。

-1 producer 需要等待 ISR 中的所有 follower 都确认接收到数据后才算一次发送完成，可靠性最高。

当 ack 为-1 时，什么情况下，Leader 认为一条消息 Commit 了？

当 ISR 中所有 Replica 都向 Leader 发送 ACK 时，leader 才 commit，这时候 producer 才能认为一个请求中的消息都 commit 了。

Kafka Unclean 配置代表什么？会对 spark streaming 消费有什么影响？

unclean.leader.election.enable 为 true 的话，意味着非 ISR 集合的 broker 也可以参与选举，这样有可能就会丢数据，spark streaming 在消费过程中拿到的

end offset 会突然变小，导致 spark streaming job 挂掉。如果 unclean.leader.election.enable 参数设置为 true，就有可能发生数据丢失和数据不一致的情况，Kafka 的可靠性就会降低；而如果 unclean.leader.election.enable 参数设置为 false，Kafka 的可用性就会降低。

Kafka 中 Consumer Group 是什么概念?

同样是逻辑上的概念，是 Kafka 实现单播和广播两种消息模型的手段。同一个 topic 的数据，会广播给不同的 group；同一个 group 中的 worker，只有一个 worker 能拿到这个数据。换句话说，对于同一个 topic，每个 group 都可以拿到同样的所有数据，但是数据进入 group 后只能被其中的一个 worker 消费。group 内的 worker 可以使用多线程或多进程来实现，也可以将进程分散在多台机器上，worker 的数量通常不超过 partition 的数量，且二者最好保持整数倍关系，因为 Kafka 在设计时假定了一个 partition 只能被一个 worker 消费（同一 group 内）。

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。

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：

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：
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；

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；

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；

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。

为什么 Kafka 不支持读写分离?

在 Kafka 中，生产者写入消息、消费者读取消息的操作都是与 leader 副本进行交互的，从 而实现的是一种**主写主读**的生产消费模型。

Kafka 并不支持主写从读，因为主写从读有 2 个很明 显的缺点:

数据一致性问题。数据从主节点转到从节点必然会有一个延时的时间窗口，这个时间 窗口会导致主从节点之间的数据不一致。某一时刻，在主节点和从节点中 A 数据的值都为 X， 之后将主节点中 A 的值修改为 Y，那么在这个变更通知到从节点之前，应用读取从节点中的 A 数据的值并不为最新的 Y，由此便产生了数据不一致的问题。

延时问题。类似 Redis 这种组件，数据从写入主节点到同步至从节点中的过程需要经 历网络→主节点内存→网络→从节点内存这几个阶段，整个过程会耗费一定的时间。而在 Kafka 中，主从同步会比 Redis 更加耗时，它需要经历网络→主节点内存→主节点磁盘→网络→从节 点内存→从节点磁盘这几个阶段。对延时敏感的应用而言，主写从读的功能并不太适用。

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.

消费者提交消费位移时提交的是当前消费到的最新消息的 offset 还是 offset+1?

offset+1

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。