Kafka是什么？主要应用场景有哪些?

Kafka是一个分布式流失处理平台

流平台的三个关键功能：

- **消息队列**：发布和订阅消息流，这个功能类似于消息队列，这也是 Kafka 也被归类为消息队列的原因。
- **容错的持久方式存储记录消息流**：Kafka 会把消息持久化到磁盘，有效避免了消息丢失的风险。
- **流式处理平台：** 在消息发布的时候进行处理，Kafka 提供了一个完整的流式处理类库。

两大应用场景：

- 消息队列：建立实时流数据管道，以可靠地在系统或应用程序之间获取数据。
- 数据处理：构建实时的流数据处理程序来转换或处理数据流。

和其他消息队列相比，Kafka的优势在哪里?

- **极致的性能**：设计中大量使用了批量处理和异步的思想，最高可以每秒处理千万级别的消息。
- **生态系统兼容性无可匹敌**：Kafka 与周边生态系统的兼容性是最好的没有之一，尤其在大数据和流计算领域。

Kafka 都有哪些特点?

- 高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个topic可以分多个partition, consumer group 对partition进行consume操作。
- 可扩展性：kafka集群支持热扩展。
- 持久性、可靠性：消息被持久化到本地磁盘，并且支持数据备份防止数据丢失。
- 容错性：允许集群中节点失败。
- 高并发：支持千个客户端同时读写。

在哪些场景下会选择Kafka?

- 日志收集
- 消息系统：解耦生产者和消费者、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。
- 运营分析：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

队列模型是什么?

队列模型是早期的消息模型，使用队列作为消息通信载体，满足生产者与消费者模式，一条消息只能被一个消费者使用，未被消费的消息再队列中保留直到被消费或者超时。

队列模型存在的问题?

如果我们需要生产者将生产的消息发给多个消费者，且消费者拿到的都是完整的信息，则这种情况就会无法解决。如果我们为每个消费者都创建一个单独的队列，然后生产者发送多份信息，这种方法就太浪费资源了。

Kafka的消息模型是什么?

发布-订阅模型

发布订阅模型使用Topic（主题）作为消息通信载体，类似于广播模式；发布者发布一条消息，该消息通过主题传递给所有的订阅者，在一条消息广播之后才订阅的用户是收不到该条消息的。

在发布 - 订阅模型中，如果只有一个订阅者，那它和队列模型就基本是一样的了。所以说，发布 - 订阅模型在功能层面上是可以兼容队列模型的。

说一说Kafka的架构

- Producer： 消息生产者，就是向 kafka broker 发消息的客户端。
- Consumer： 消息消费者，向 kafka broker 取消息的客户端。
- Topic： 可以理解为一个队列，一个 Topic 又分为一个或多个分区，
- Consumer Group：这是 kafka 用来实现一个 topic 消息的广播（发给所有的 consumer）和单播（发给任意一个 consumer）的手段。一个 topic 可以有多个 Consumer Group。
- Broker： 一台 kafka 服务器就是一个 broker。一个集群由多个 broker 组成。一个 broker 可以容纳多个 topic。
- Partition：为了实现扩展性，一个非常大的 topic 可以分布到多个 broker上，每个 partition 是一个有序的队列。partition 中的每条消息都会被分配一个有序的id（offset）。将消息发给 consumer，kafka 只保证按一个 partition 中的消息的顺序，不保证一个 topic 的整体（多个 partition 间）的顺序。
- Offset：kafka 的存储文件都是按照 offset.kafka 来命名，用 offset 做名字的好处是方便查找。例如你想找位于 2049 的位置，只要找到 2048.kafka 的文件即可。

Kafka的多副本机制是什么?

kafka为分区引入多副本机制。分区中的多个副本之间会有一个叫做leader的家伙，其他副本称为follower。我们发送的消息会被发送到 leader 副本，然后 follower 副本才能从 leader 副本中拉取消息进行同步。

> 生产者和消费者只与 leader 副本交互。你可以理解为其他副本只是 leader 副本的拷贝，它们的存在只是为了保证消息存储的安全性。当 leader 副本发生故障时会从 follower 中选举出一个 leader,但是 follower 中如果有和 leader 同步程度达不到要求的参加不了 leader 的竞选。

Kafka多分区以及多副本机制有什么好处?

- kafka通过给特定的topic指定多个patition，而各个patition可以分布在不同的Broker上，这样便能提供比较好的并发能力（负载均衡）。
- patition可以指定对应的Replica数，这也极大提高了消息存储的安全性，提高了容灾能力，不过也相应的增加了所需要的存储空间。

Zookeeper在Kafka中的作用是什么?

Zookeeper主要是为Kafka提供元数据管理功能

- **Broker 注册**：在 Zookeeper 上会有一个专门**用来进行 Broker 服务器列表记录**的节点。每个 Broker 在启动时，都会到 Zookeeper 上进行注册。
- **Topic 注册**：在 Kafka 中，同一个**Topic 的消息会被分成多个分区**并将其分布在多个 Broker 上，**这些分区信息及与 Broker 的对应关系**也都是由 Zookeeper 在维护。
- **负载均衡**：上面也说过了 Kafka 通过给特定 Topic 指定多个 Partition, 而各个 Partition 可以分布在不同的 Broker 上, 这样便能提供比较好的并发能力。 对于同一个 Topic 的不同 Partition，Kafka

会尽力将这些 Partition 分布到不同的 Broker 服务器上。当生产者产生消息后也会尽量投递到不同 Broker 的 Partition 里面。当 Consumer 消费的时候，Zookeeper 可以根据当前的 Partition 数量以及 Consumer 数量来实现动态负载均衡。

使用Kafka能否不引入Zookeeper?

在 Kafka 2.8 之前，Kafka 最被大家诟病的就是其重度依赖于 Zookeeper。在 Kafka 2.8 之后，引入了基于 Raft 协议的 KRaft 模式，不再依赖 Zookeeper，大大简化了 Kafka 的架构，让你可以以一种轻量级的方式来使用 Kafka。

Kafka如何做到消息的有序性?

kafka 中的每个 partition 中的消息在写入时都是有序的，而且单独一个 partition 只能由一个消费者去消费，可以在里面保证消息的顺序性。但是分区之间的消息是不保证有序的。

Kafka如何保证消息的消费顺序?

- 1个topic只对应一个patition
- 发送消息的时候指定key/patition

Kafka如何保证消息不丢失?

- 生产者丢失消息的情况：生产者(Producer) 调用`send`方法发送消息之后，消息可能因为网络问题并没有发送过去。
    - 发送后通过get()方法来获取调用send()的调用结果，send其实是异步操作，但是后面调用get方法的话就是同步操作了，如果发送失败，重新发送即可。
    - 使用回调函数来判断消息是否发送成功，此时没有破坏异步发送，失败重新发送即可。
    - 将retries设置一个比较合理的值，一般是3，可以设置的大一些，当出现网络问题之后能够自动重试消息发送，避免消息丢失。另外，建议还要设置重试间隔，因为间隔太小的话重试的效果就不明显了。
- 消费者丢失消息的情况：当消费者拉取到了分区的某个消息之后，消费者会自动提交了 offset。自动提交的话会有一个问题，试想一下，当消费者刚拿到这个消息准备进行真正消费的时候，突然挂掉了，消息实际上并没有被消费，但是 offset 却被自动提交了。
    - 手动关闭自动提交 offset，每次在真正消费完消息之后再自己手动提交 offset 。
- Kafka弄丢了消息：假如 leader 副本所在的 broker 突然挂掉，那么就要从 follower 副本重新选出一个 leader ，但是 leader 的数据还有一些没有被 follower 副本的同步的话，就会造成消息丢失
    - 解决办法就是我们设置 **acks = all**，acks 的默认值即为 1，代表我们的消息被 leader 副本接收之后就算被成功发送。当我们配置 **acks = all** 表示只有所有 ISR 列表的副本全部收到消息时，生产者才会接收到来自服务器的响应. 这种模式是最高级别的，也是最安全的，可以确保不止一个 Broker 接收到了消息. 该模式的延迟会很高
    - **设置 replication.factor >= 3**，为了保证 leader 副本能有 follower 副本能同步消息，可以保证每个 分区(partition) 至少有 3 个副本。虽然造成了数据冗余，但是带来了数据的安全性。
    - **设置 min.insync.replicas > 1**，代表消息至少要被写入到 2 个副本才算是被成功发送。
    - **设置 unclean.leader.election.enable = false**，当 leader 副本发生故障时就不会从 follower 副本中和 leader 同步程度达不到要求的副本中选择出 leader ，这样降低了消息丢失的可能性。

Kafka如何保证消息不重复消费?

出现消息重复消费的原因

- 服务端侧已经消费的数据没有成功提交 offset（根本原因）。
- Kafka 侧 由于服务端处理业务时间长或者网络链接等等原因让 Kafka 认为服务假死，触发了分区 rebalance。

  解决方案

- 消费消息服务做幂等校验
- 将 `enable.auto.commit` 参数设置为 false，关闭自动提交，开发者在代码中手动提交 offset。

  > 什么时候提交offset合适?

  - 处理完消息再提交：依旧有消息重复消费的风险，和自动提交一样。
  - 拉取到消息即提交：会有消息丢失的风险。

Kafka重试机制

当消息消费失败时，重试多次后会跳过当前消息，继续进行后续消息的消费，不会一直卡在当前消息。Kafka 消费者在默认配置下会进行最多 10 次 的重试，每次重试的时间间隔为 0，即立即进行重试。如果在 10 次重试后仍然无法成功消费消息，则不再进行重试，消息将被视为消费失败。

重试失败后的数据如何再次处理?

当消息进入队列后，消费者会尝试处理它。如果处理失败，或者超过一定的重试次数仍无法被成功处理，消息可以发送到死信队列中，而不是被永久性地丢弃。在死信队列中，可以进一步分析、处理这些无法正常消费的消息，以便定位问题、修复错误，并采取适当的措施

死信队列（Dead Letter Queue，简称 DLQ） 是消息中间件中的一种特殊队列。它主要用于处理无法被消费者正确处理的消息，通常是因为消息格式错误、处理失败、消费超时等情况导致的消息被"丢弃"或"死亡"的情况。