分布式消息系统 Kafka
知识点总结暨面试题

第 1 次直播课

【知识点 01】请简述 Kafka 中 Broker、Topic、Partition 及 Segment 间的关系。
【解答参考】Kafka 中的 Broker 指的就是 Kafka 集群中的一个节点，就是一台 Kafka 主机。Broker 中可以存放很多的消息，但这些消息是分类存放的，一类就是一个 Topic。每一类的消息在一台 Broker 中都被集中存放在了一起，被放在了一个目录中，这个目录称为 Partition。但消息本身并不是文件，其需要存放到文件中。用于存放消息的文件称为 Segment。

一个 Topic 的消息可以被存放到多个 Broker 中，一个 Broker 中可以存放一个 Topic 的多个 Partition，而一个 Partition 中可以存放很多的 Segment，一个 Segment 中可以存放很多的消息。

【知识点 02】Kafka 与其它 MQ 相比，其最大的特点就是高吞吐率。Kafka 中消息存放的顺序存放是实现高吞吐率的重要特征。请简述一下 Kafka 中消息的顺序存放特性。
【解答参考】Kafka 中消息的顺序存放指的是一个 Segment 中消息的存放是顺序存放的。为了能够存放大量的消息，Kafka 是将消息直接存放在了磁盘。由于磁盘是一个低速 IO 设备，为了提高 IO 速度，就想将 Broker 中同一 Topic 的消息顺序存放在磁盘。但一个 Topic 的消息量是非常庞大的，但到底有多大，并不确定，因为其是一直在增长的。为了便于连续磁盘空间的分配，就将一个 Topic 的消息写入到一个文件中，这个文件的最大大小通过配置可以固定。即在磁盘中查找到一块连接的指定大小的空间是可以完成的。这个文件称为 Segment。随着消息量的不断增长，Segment 文件越来越多，为了便于管理，将同一 Topic 的 Segment 文件都存放到一个或多个目录中，这些目录就是 Partition。通过以上叙述可知，Segment 中的消息是顺序存放的，而 Partition 中的这些 Segment 文件的存储并不是顺序存放的。

【知识点 03】Kafka 中的 Segment 实际是一套文件，其中最为重要的是.log 与.index 文件。这两个文件的文件名是相同的。请简述文件名的意义及这两个文件的关系。
【解答参考】Kafka 中的 segment 是一个逻辑概念，其包含两类重要的物理文件，分别为“.index”文件和“.log”文件。“.log”文件中存放的是消息，而“.index”文件中存放的是“.log”文件中消息的索引。

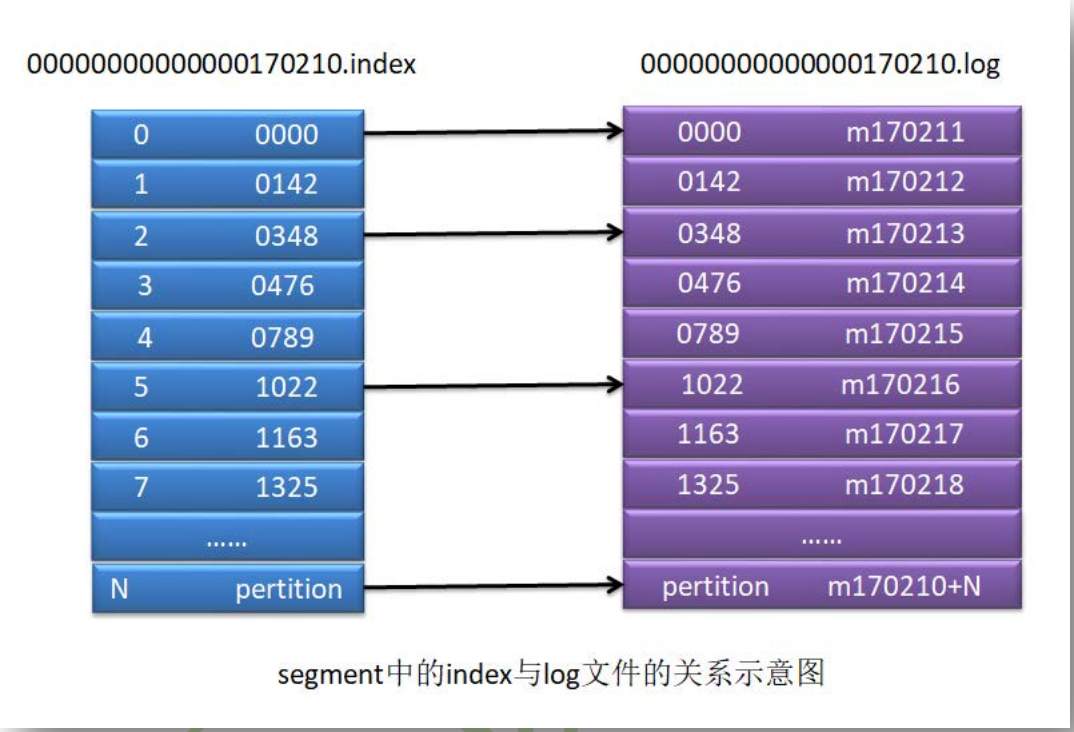
这两个文件的文件名是成对出现的，即会出现相同文件名的 log 与 index 文件。文件名由 20 位数字字符组成，其要表示一个 64 位长度的数值（2 的 64 次方是一个长度为 20 的数字）。但作为文件名，其数值长度不足 20 位的全部用 0 填补。

这个 64 位长度的数值表示当前 segment 文件之前有多少条消息。那么，第一个 segment 文件的文件名就应是由 20 个 0 组成，因为其前面没有消息了。

- 第 0 号 segment 文件，表示其前面没有消息。
  00000000000000000000.index
  00000000000000000000.log
- 第 170210 号 segment 文件，表明其前面有 170210 个消息。
  00000000000000170210.index
  00000000000000170210.log

- 第 239430 号 segment 文件，表明其前面有 239430 个消息。
  00000000000000239430.index
  00000000000000239430.log

【知识点 04】消费者准备消费编号为 170213 的消息，这个消费者是如何找到这条消息的？请简述这个查找过程。



segment中的index与log文件的关系示意图

【解答参考】Kafka 中的消息是存放在相应相应 partition 的相应 segment 中的。对于消费者来说，每个消费者其消费的 partition 是系统自动分配好的，所以其不用查找 partition。但一个 partition 中的 segment 会有很多，而 segment 中包含 index 与 log 两个重要文件。简单来说，消费者是通过 index 文件找到 log 文件中包含的消息的。

具体过程是这样的：170213 这个称为消息在 partition 中的偏移量 offset。首先拿 170213 这个 offset 通过二分法在所有 segment 文件名中查找对应文件。当定位到 segment 文件名为 00000000000000170210 的文件后，进行减法运算：170213 – 170210 =3。然后再在该名称的 index 文件中定位到 2 号（编号从 0 开始），而 2 号对应的偏移地址为 0348。最后在当前 log 文件中直接定位到 0348 地址处即可找到该消息。

不过，这里还有个问题：消费者并不知道这个消息的大小，而 log 中的消息是顺序存放的。消费者是如何知道这条消息已经读取完毕了呢？存放在 log 文件中的消息并不仅仅是一个单纯的消息，其是由若干元数据构成的、具有固定格式的一个消息体。即每个消息体都有相应的开始标识位。当读到下一个消息的开始标识位时，表示当前消息已经结束。

【知识点 05】Kafka 中为了减轻单台 broker 的压力，一般会为 Topic 设置多个 partition。那么，一个 Topic 设置多少个 partition 合适呢？

【解答参考】假设某 topic 中有 N 个 partition，集群中有 M 个 broker，则 broker 与 partition

间的关系如下所述：

- 若 N>M，且 N 是 M 的整数倍（N%M=0），那么每个 broker 会平均存储该 topic 的多个 partition。
- 若 N>M，但 N 不是 M 的整数倍（N%M!=0），那么就会出现 broker 中 partition 分布不平均的情况。应尽量避免这种情况的发生，这种情况容易导致 Kafka 集群消息不均衡，各个 broker 的任务压力不均衡。
- 若 N<M，则会有 N 个 broker 存储该 topic 的一个 partition，剩下的（M-N）个 broker 将不存储该 topic 的 partition 消息。

通过以上分析可知，partition 的数量最好是 broker 数量的整数倍。

【知识点 06】请总结一下 Kafka 中 Consumer Group 中的 Consumer 与其要消费的 Topic 中的 partition 间的数量关系。

【解答参考】Consumer Group 中组内 consumer 与其要消息的 Topic 的 partition 的关系是 1:n，partition 与组内 consumer 的关系则是 1:1。也就是说，在稳定状态下，一旦为某组内 consumer 分配了某一个或几个 partition 后，就不会发生变化了；反过来说，一旦为某 partition 分配了组内 consumer，就不会再为其分配其它组内 consumer 了。

【知识点 07】Kafka 中一个 partition 只允许一个 Consumer Group 中的一个组内 Consumer 进行消费。请简述这样设计的优劣。

【解答参考】Kafka 中一个 partition 只允许一个 Consumer Group 中的一个组内 Consumer 进行消费。而一个组内 Consumer 可以消费同一 Topic 的多个 parittion。这样设计的好处是，对于每个 parition 中消费消费的偏移量控制简单。不足是，无法让同一个组内的 consumer 均匀消费消息，因为消息在同一 Topic 的 parition 中的存放并不是平均的。一旦组内 consumer 与 parition 的消费关系确立，则可能会导致某些组内 consumer 需要消费的消息量很大，有的组内 consumer 可能无消息可消费。

【知识点 08】我们知道，Kafka 中 partition 的数量最好是 broker 数量的整数倍。但具体来说，是一倍、两倍，还是三倍，如何选择呢？

【解答参考】Kafka 中 partition 的数量最好是 broker 数量的整数倍。但应该设置为几倍，需要根据实际需求中一个消费者组包含消费者的数量。我们知道，Kafka 的设计要求，同一消费者组中两个消费者是不能同时消费同一个 partition 的。若 Consumer Group 中包含的消费者数量较多，而其消费的 Topic 的 partition 数量较少，则会导致很多消息者是闲置的。降低了 Consumer Group 的消费能力。所以，一个 Topic 中包含的 partition 数量，与其消费者组中包含的消费者数量要相匹配，这样可以充分发挥消费者组的消费能力。

【知识点 09】为了保障消息的安全性，我们会为 Kafka 的 parition 设置副本。每个 partition 可能有多个副本，但有且仅有一个作为 Leader，其余的都是 Follower。partition Leader 与 Follower 间是主从还是主备关系？

【解答参考】Kafka 中 partition Leader 与 Follower 间是主备关系。Leader 负责对外提供读写服务，即 producer 与 consumer 操作的都是parititon leader。所有 Follower 都需要从 Leader 同步消息，但平时对外是不提供服务的。一旦 leader 出现问题，会从 follower 马上再选举出一个新的 leader。

【知识点 10】在 Kafka 的副本中有 AR、ISR 与 ORS 概念，它们都是什么意思，它们间是什么关系？

【解答参考】Kafka 中某个 partition 的所有的副本统称为 Assigned Replicas，即 AR。初始时 leader 与所有 follower 都在 ISR(In-Sync Replicas)列表，即初始时 ISR = AR。ISR 中的 partition 是要从 leader 同步消息的。但同步会有延迟，只要延迟超过了阈值 replica.lag.time.max.ms，就会把 follower 剔除出 ISR，移入 OSR(Outof-Sync Replicas)列表。故 AR=ISR+OSR。

ISR 由 leader 负责维护，leader 会对 OSR 中的 follower 进行定期检测，以查看其是否适合重新进入到 ISR。

第 2 次直播课

【知识点 01】在 Kafka 中有一个 Rebalance 的概念，请简述一下。

【解答参考】在 Kafka 中，当消费者组中消费者数量发生变化，或 Topic 中的 partition 数量发生了变化时， partition 的所有权会在消费者间转移，即 partition 会重新分配，这个过程称为再均衡 Rebalance。

再均衡能够给消费者组及 broker 集群带来高可用性和伸缩性，但在再均衡期间消费者是无法读取消息的，即整个 broker 集群有一小段时间是不可用的。因此要避免不必要的再均衡。

【知识点 02】在 Kafka 中当消费者消费完消息后需要提交其消费过消息的 offset，这个 offset 保存在哪里？

【解答参考】在 Kafka 中，老版本的消费 offset 是存储的 zookeeper 中的，但是 zookeeper 并不适合频繁的读/写操作，所以在新版本的中消费 offset 存放到了__consumer_offsets 内置 topic 的 partition 中了。该主题的 partition 默认有 50 个，那么 Consumer Group 消费的 offset 存放的分区索引可以通过如下公式计算：Math.abs(groupID.hashCode()) % 50。

【知识点 03】Kafka 中的 partition leader 与 broker controller 是什么，是由谁选举出来的？

【解答参考】Kafka 中为了保障消息的高可用性，一般会为 partition 创建副本。副本中 partition leader 负责对外提供服务，而 partition follower 则仅同步 leader 中的数据。当 partition leader 出现 ，则立马由 broker controller 从 follower 中选举出来一个新的 leader。其实所谓选举，其实就是“按资排辈”。从 ISR 列表中找到第一个 follower 作为新的 leader。

broker controller 负责管理分区与副本，例如 partition leader 的选举。broker controller 由 zk 负责选举。其选举算法就是 zk 中“Master 的选举”应用场景，即由 kafka 集群中的 broker 在 zk 中创建一个临时节点，谁先创建了谁就是 broker controller。一般情况下就是，哪个 broker 先启动了，哪个就是 broker controller。

【知识点 04】Kafka 中的生产者生产的消息被写入到了哪个 partition？消费者将 offset 提交到了哪里？这两个问题有什么联系？

【解答参考】前两个问题其实都是 Kafka 中生产者的消息路由策略。只不过，消费者提交的 offset 是一种特殊的消息，其是提交到了一个名称为__consumer_offsets 的特殊主题的相应 partition。__consumer_offsets 主题默认有 50 个分区。

对于普通消息的路由，其路由规则如下：

1) 若指定了消息要写入的 partition，则直接写入到指定的 partition；
2) 若未指定 partition 但指定了消息的 key，则通过对 key 的 hash 值与 partition 数量取模，

该取模结果就是要选出的 partition 索引；

3) 若 partition 和 key 都未指定，则使用轮询算法选出一个 partition。

对于消费者提交 offset，这个 offset 是一种特殊的消息，该消息的 key 为消费者组的 Id。offset 要写入到__consumer_offsets 主题的哪个 partition，其索引的计算方式与前面说的相同：通过 groupId 的 hashCode 与 50 取模。最终这个 offset 被提交到了这个 partition。

【知识点 05】Kafka 中消息的生产者会将消息写入到 broker 中，这个过程是一个相对比较复杂的过程。请详细描述一下这个写入过程。

【解答参考】消息生产者将消息发送给 broker，并形成最终的可供消费者消费的 log，是一个比较复杂的过程。具体过程如下：

1) producer 向 broker 集群提交连接请求，其所连接上的任意 broker 都会向其发送 broker controller 的通信 URL，即 broker controller 主机配置文件中的 listeners 地址
2) 当 producer 指定了要生产消息的 topic 后，其会向 broker controller 发送请求，请求当前 topic 中所有 partition 的 leader 列表地址
3) broker controller 在接收到请求后，会从 zk 中查找到指定 topic 的所有 partition 的 leader，并返回给 producer
4) producer 在接收到 leader 列表地址后，根据消息路由策略找到当前要发送消息所要发送的 partition leader，然后将消息发送给该 leader
5) leader 将消息写入本地 log，并通知 ISR 中的 followers
6) ISR 中的 followers 从 leader 中同步消息后向 leader 发送 ACK
7) leader 收到所有 ISR 中的 followers 的 ACK 后，增加 HW，表示消费者已经可以消费到该位置了
8) 当然，若 leader 在等待的 followers 的 ACK 超时了，发现还有 follower 没有发送 ACK，则会将该 follower 从 ISR 中清除，然后增加 HW。

【知识点 06】Kafka 中的 HW 机制是什么？（或 Kafka 中的 HW 截断机制是什么？）

【解答参考】HW，HighWatermark，高水位，表示 Consumer 可以消费到的最高 partition 偏移量。在 Kafka 中与 HW 相关的机制有两种：HW 机制与 HW 截断机制。它们都是为了保证 partition leader 与 follower 间数据的一致性。只不过它们处理的场景不同。

- HW 机制：该机制在 Kafka 集群正常运行状态下可以防止 partition leader 与 follower 间出现数据不一致。该机制要求，对于 partition leader 新写入的消息，consumer 不能立刻消费。leader 会等待该消息被所有 ISR 中的 partition follower 同步后才会更新 HW，此时该消息才能被 consumer 消费。
- HW 截断机制：该机制在 Kafka 中 partition leader 出现宕机情况然后又恢复时，可以防止 partition leader 与 follower 间出现数据不一致。当原 Leader 宕机后又恢复时，将其 LEO 回退到其宕机时的 HW，然后再与新的 Leader 进行数据同步，这种机制称为 HW 截断机制。

【知识点 07】Kafka 是如何保证消息发送的可靠性的？

【解答参考】Kafka 的消息生产者有一个配置属性 acks，用于指定消息发送的可靠性级别的。

- 0 值：异步发送。生产者向 kafka 发送消息但不需要 kafka 反馈成功 ack。该方式效率最高，但可靠性最低。其可能会存在消息丢失的情况。
- 1 值：同步发送，默认值。生产者发送消息给 kafka，broker 的 partition leader 在收到消

息后马上发送成功 ack，生产者收到后才会再发送消息。如果一直未收到 kafka 的 ack，则生产者会认为消息发送失败，会重发消息。

- -1 值：同步发送。其值等同于 all。生产者发送消息给 kafka，kafka 收到消息后要等到 ISR 列表中的所有副本都同步消息完成后，才向生产者发送成功 ack。如果一直未收到 kafka 的 ack，则认为消息发送失败，会自动重发消息。

【知识点 08】Kafka 的消息生产者设置 acks 属性值为 1 时，能否使生产者确认它发送的消息发送成功或失败呢？

【解答参考】Kafka 的消息生产者设置 acks 属性值为 1 时表示，生产者发送消息给 kafka，broker 的 partition leader 在收到消息后马上发送成功 ack，生产者收到后才会再发送消息。如果一直未收到 kafka 的 ack，则生产者会认为消息发送失败，会重发消息。

该方式不能使生产者确认其发送的消息一定成功。例如，当 partition leader 收到消息后向生产者发送了 ACK，在 ISR 中的 Follower 还未同步时 leader 挂了，此时，leader 先前收到的消息对于 kafka 来说就不存在，但对于生产者来说，kafka 已经接收成功。但实际上该消息已经丢失。所以，无法保证消息不丢失。

但该方式可以使生产者确认其发送的消息失败。只要在超时时限内生产者没有收到 ACK 就表示消息发送失败。发送失败，生产者会重新发送。

【知识点 09】Kafka 的消息生产者设置 acks 属性值为 all 时，基本可以保证消息的不丢失。但却可能会引发消息的重复接收。为什么？

【解答参考】Kafka 的消息生产者设置 acks 属性值为 all 时表示，生产者发送消息给 kafka，kafka 收到消息后要等到 ISR 列表中的所有副本都同步消息完成后，才向生产者发送成功 ack。如果一直未收到 kafka 的 ack，则认为消息发送失败，会自动重发消息。

这种模式下可靠性很高，很少会出现消息丢失的情况。但可能会出现部分 Follower 重复接收消息的情况。例如生产者发送消息给 partition leader，然后 ISR 中的 follower 要同步消息。当 follower 同步还未完成时 partition leader 挂了，此时不会发送 ack 给生产者。由于发送者没有收到 ack，所以生产者会再次发送消息给新的 Leader。若新的 Leader 曾经同步过一部分原来的消息，那么此时又接收到相同的消息，此时 leader 中的消息就会有重复消息。

该模式下 kafka 对消息的重复接收问题无法直接解决。但可以想办法对于重复接收的消息不进行重复消费：为消息指定唯一标识 key，然后在消费者端定义去重机制。当然，消费者端的去重，是需要开发人员自己定义的。

【知识点 10】Kafka 中 partition leader 若挂了，一般会选择 ISR 中的 follower 作为新的 leader。这个新 leader 的选举是由谁完成的？若 ISR 中没有其它 follower 能否选举出新的 leader？

【解答参考】kafka 中 partition leader 若挂了，broker controller 会选择 ISR 中的 follower 作为新的 leader。若 ISR 中没有其它 follower 能否选举出新的 leader，可以通过 broker 的属性设置 unclean.leader.election.enable 的值来确定。

- 若该值为 false，则只能从 ISR 中选择。
- 若该值为 true，在 ISR 中没有副本的情况下可以选择任何一个该 Topic 的 partition 作为新的 leader，该策略可用性高，但可靠性没有保证。

【知识点 11】Kafka 中 partition leader 若挂了，通过设置允许其从任意没有宕机的主机的 partition 中选举新的 leader，这种情况可能会导致大量数据的丢失，为什么？

【解答参考】在 ISR 中没有副本的情况下可以选择的 partition 只能是 OSR 中的。一个 partition

只所以能够进入到 OSR，就是因为其与原来的 leader 的通信出现了问题。若该 partition 成为了新的 leader，则可能会出现这个新的 leader 与其它所有副本均无法通信的情况。但由于其为 leader，所以其只会认为是其它副本出现了问题，从而导致该 leader 的 ISR 中没有 follower。而没有 follower 的风险是很高的，这个 leader 的失效将会导致大量消息的丢失。

【知识点 12】Kafka 中，当一个 Consumer Group 中仅包含一个 Consumer 时，若其消费能力较低，则可能会引发重复消费。为什么？怎么解决？

【解答参考】正常情况下，consumer 在“自动提交超时时限”内消费完一批消息后会自动提交 offset。但当 consumer 消费能力比较低时，其取出的一批消息在阈值时间内没有消费完毕，此时 consumer 会向 broker 提交一个异常。此时 broker 不会认为该 consumer 宕机，因为当前 consumer 向 broker 提交了信息。

对于 consumer 来说，由于本次消费过程在时限内没有完成，即没有成功，所以该 consumer 会再从该 partition 中拉取消息。由于前面没有提交 offset，所以这次拉取的消息与上次的是相同的。该 consumer 又重新消费之前的那一批消息，然后就又出现了消费超时，所以会造成死循环，一直消费相同的消息。

对于这种情况下的重复消费的解决方案是，延长 offset 自动提交时间，即增加自动提交超时时限 auto.commit.interval.ms 的值。或将自动提交改为手动提交，真正消费完毕后再进行提交。