分布式协调服务器 Zookeeper
面试题暨知识点总结

第 1 次直播课

【Q-01】对于 zk 功能的开发，或者说是对于 zk 在具体应用场景中的解决方案的设计中，我们要着重考虑对哪两个 zk 特性的灵活使用？请谈一下你的看法。

【RA】对于 zk 功能的开发，或者说是对于 zk 在具体应用场景中的解决方案的设计中，我们要着重考虑 zk 的临时节点及 watcher 监听机制的灵活使用。

- 临时节点：其生命周期与创建它的客户端会话是绑定在一起的。客户端消失，则临时节点消失。
- watcher 机制：客户端可以对节点进行监听，监听其数据内容的变更、子节点列表的变更等。

【Q-02】对于 zk 的节点类型，谈一下你的认识。

【RA】每个 znode 根据节点类型的不同，具有不同的生命周期与特性。

- 持久节点：节点被创建后会一直保存在 zk 中，直到将其删除。
- 持久顺序节点：一个父节点可以为它的第一级子节点维护一份顺序，用于记录每个子节点创建的先后顺序。其在创建子节点时，会在子节点名称后添加数字后辍，作为该子节点的完整节点名。序号由 10 位数字组成，由这些子节点的父节点维护，从 0 开始计数。
- 临时节点：临时节点的生命周期与客户端的会话绑定在一起，会话消失则该节点就会被自动清理。
- 临时顺序节点：添加了创建序号的临时节点。

【Q-03】查看 Zookeeper 的节点属性，哪个属性表示节点的类型？请谈一下你的认识。

【RA】zookeeper 的节点属性 ephemeralOwner 反应了节点的类型。若当前 znode 是持久节点，则其值为 0；若为临时节点，则其值为创建该节点的会话的 SessionID。当会话消失后，会根据 SessionID 来查找与该会话相关的临时节点进行删除。

【Q-04】对于 zk 来说 watcher 机制非常重要，watcher 机制的工作原理是怎样的？谈一下你的认识。

【RA】当客户端想要监听 zk 中某节点的状态变化时，需要向该节点注册 watcher 监听。其首先会在客户端创建一个 watcher 对象，并为其添加相应的回调。当 zk 中对应的节点发生了相应的 watcher 事件后，zk 会向客户端发送事件通知，触发该 watcher 回调的执行。执行完毕，该 watcher 对象销毁。若要再次监听，则需再次注册。

【Q-05】对于 zk 官方给出了四种最典型的应用场景，配置维护就是之一。什么是配置维护？请谈一下你的看法。

【RA】分布式系统中，很多服务都是部署在集群中的，即多台服务器中部署着完全相同的应用，起着完全相同的作用。当然，集群中的这些服务器的配置文件是完全相同的。

若集群中服务器的配置文件需要进行修改，那么我们就需要逐台修改这些服务器中的配置文件。如果我们集群服务器比较少，那么这些修改还不是太麻烦，但如果集群服务器特别

多，比如某些大型互联网公司的 Hadoop 集群有数千台服务器，那么纯手工的更改这些配置文件几乎就是一件不可能完成的任务。即使使用大量人力进行修改可行，但过多的人员参与，出错的概率大大提升，对于集群所形成的危险是很大的。

zk 可以通过一种发布订阅模式对集群中的配置文件进行统一管理，这就是配置维护。其作用与 Spring Cloud Config 的作用相同。

【Q-06】对于 zk 官方给出了四种最典型的应用场景，命名服务就是之一。不过，像 UUID、GUID 等就可以非常方便地生成几乎不会重复的 id，为什么还要那么麻烦的使用 zk 实现呢？
【RA】不错，UUID、GUID 可以非常方便地生成几乎不会重复的 id，但也存在问题。例如，id 过长，占用空间；虽然那么长，但还没有语义，无法从 id 中解析出业务数据。而使用 zk 生成的 id 一定是不重复的，且可以根据业务自定义 id 长度和语义。这是 UUID、GUID 无法替代的。

【Q-07】对于分布式日志系统，无论是日志源主机还是日志收集主机，都需要监控其存活状态。请设计一种方案，可以方便地监控日志收集主机的存活状态。
【RA】对于分布式日志系统中日志源主机存活状态的监控是比较简单的，只需要让日志源主机在 zk 中其对应的日志收集主机节点下创建一个临时节点，系统只需向这个日志收集主机注册子节点列表变更 watcher 监听即可。

但是对于日志收集主机就不同了，因为其是持久节点，主机宕机也不会引发持久节点的消失。此时可以这样实现：在日志收集主机对应的持久节点创建完毕后，马上在其下创建一个临时节点，然后让系统为该节点注册一个节点删除 watcher 监听。一旦日志收集主机宕机，该临时节点就会消失，然后就会触发该节点删除 watcher 的回调，然后就可以进行 Rebalance。

【Q-08】使用 DBMS 可以实现 Master 选举，实现原理是什么？存在什么问题？请谈一下你的看法。
【RA】使用 DBMS 实现 Master 选举的原理是，集群中所有主机都向关系型数据库中插入一条相同主键的记录，数据库会自动进行主键冲突检查，然后只允许一台主机插入成功。那么这个插入成功的主机就是 Master。

但该方案存在一个问题：若 Master 宕机，DBMS 无法通知其它 Slave，让其它 Slave 重新发起 Master 的再次选举。

【Q-09】使用 zk 可以实现 Master 选举，实现原理是什么？请谈一下你的看法。
【RA】使用 zk 实现 Master 选举的原理是，集群中所有主机都向 zk 中创建相同路径下的某持久节点注册子节点列表变更 watcher 监听，并在该节点下持久相同名称的临时节点，谁创建成功谁就是 Master。

当 Master 宕机，该临时节点消失，此时会触发其他主机 watcher 回调的执行。watcher 回调会重新抢注该节点下的临时节点，谁注册成功谁就是 Master。即可以实现 Master 宕机后的自动重新选举。

【Q-10】“数据复制总线”的功能可以由 MySQL 主备集群完成吗？数据复制总线与 zk 有什么关系。请谈一下你的认识。
【RA】MySQL 主备集群完成的是相同数据库间的数据备份，而“数据复制总线”则可以完成不同 DBMS 中异构 DB 间数据的复制、变更。

数据复制总线中复制任务是由复制器 replicator 完成的。为了保证复制任务的顺利完成，

解决 replicator 的单点问题，replicator 采用了主备模式。即只有一个是可以工作的 RUNNING 状态，其它均为热备的 STANDBY 状态。谁是 RUNNING 状态？RUNNING 与 STANDBY 间如何实现协调同步的工作？RUNNING 状态的 replicator 宕机，哪个 STANDBY 的 replicator 变为 RUNNING 状态？这些都需要由 zk 来负责。

第 2 次直播课

【Q-01】我们可以通过上层业务向关系型数据库中添加额外的行锁、表锁或事务处理等来实现分布式锁。为什么还需要通过 zk 来实现分布式锁功能？请谈一下你的看法。
【RA】我们可以通过上层业务向关系型数据库中添加额外的行锁、表锁或事务处理等来实现分布式锁。但目前绝大多数分布式系统的性能瓶颈都集中在数据库操作引发的数据库性能上，若再给数据库添加更多的额外功能，会势必会增加数据库的负担，降低整个系统的性能。此时可以使用第三方的 zk 来实现分布式锁，以减轻 DBMS 的压力。当然，zk 实现的分布式锁还可以实现非 DBMS 中共享数据的读写访问控制。

【Q-02】什么是锁机制？锁都有哪些分类？请谈一下你的认识。
【RA】锁机制是一种对共享数据读写访问的控制机制，其有两种类型：

- 排他锁（eXclusive locks）：也称为 X 锁、写锁。若事务 T 对数据对象 O 加了排他锁，那么在整个加锁期间，只允许事务 T 对 O 进行读写操作，其他任何事务都不能再对该数据对象进行任何操作，直到 T 释放了排他锁。
- 共享锁（Shared locks）：又称为 S 锁、读锁、若事务 T 对数据对象 O 加了共享锁，那么该事务只能对 O 进行读操作。当然，其它事务同时也可以对数据对象 O 再添加共享锁，且可以对 O 进行读操作。只有当所有共享锁全部释放后，才可对 O 添加排他锁。

【Q-03】zk 可以实现分布式锁，Redis 也可以实现。由它们实现的分布式锁有什么区别？请谈一下你的认识。
【RA】使用 zk 实现的分布式锁是 CP 的分布式锁。因为 zk 是 CP 的。在某客户端向 zk 集群中的某节点写入数据后，会等待超过半数的其它节点完成同步后，才会响应该客户端。

使用 Redis 实现的分布式锁是 AP 的分布式锁。因为 Redis 是 AP 的。在某客户端向 Redis 集群中的某节点写入数据后，会立即响应该客户端，之后在 Redis 集群中会以异步的方式来同步数据。

对于 AP 的分布式锁，需要注意可能会出现的问题：一个客户端 a 在 Redis 集群的某节点 A 写入数据后，另一个节点 B 在还未同步时，另一个客户端 b 从 B 节点读取数据，没有发现 a 写入的数据。此时可能会出现问题。所以，如果某共享资源要求必须严格按照锁机制进行访问，那么就使用 zk 实现的 CP 锁。

【Q-04】什么是 Barrier 队列？请谈一下你的认识。
【RA】Barrier 队列是分布式系统中的一种同步协调器，规定了一个队列中的元素必须全部集聚齐后才能继续执行后面的任务，否则一直等待。其常见于大规模分布式并行计算的应用场景中：最终的合并计算需要基于很多并行计算的子结果来进行。

【Q-05】一个 zk 客户端到底连接的是 zk 列表中的哪台 Server？请谈一下你的看法。
【RA】客户端在获取到 zk 列表后，首先将 zk 列表打散，然后再进行轮询方式获取到一个由主机名与 port 构成的地址，然后再根据主机名获取到其对应的所有 ip，再对这些 ip 进行

shuffle，获取 shuffle 后的第一个地址，进行连接尝试，直到连接成功，或不满足重试策略。

【Q-06】（追问）为什么要打散？请谈一下你的看法。
【RA】若不进行打散操作，则一定都是按照原来设置好的顺序进行的连接尝试，都是从第一台开始连接，而一般情况下都会连接成功。这样的话每一个客户端连接的将都是第一台 Server，不利于负载均衡。

【Q-07】（再追问，看来是打破砂锅了🤔）zk 客户端指定的要连接的 zk 集群地址，会被 shuffle 几次？请谈一下你的看法。
【RA】zk 客户端指定的要连接的 zk 集群地址可能会被 shuffle 两次。第一次是对给出地址的直接 shuffle，shuffle 后会通过轮询方式获取到一个地址。如果这个地址是以主机名 hostName 方式给出的，则会再根据 hostName 获取到其对应的所有 IP，然后会对这些 IP 再进行一次 shuffle，shuffle 后会通过轮询方式获取到一个地址，这个地址就是客户端真正要连接的 zk 地址。

【Q-08】第一个客户端将 zk 列表打散后，在打散的列表上采取轮询方式尝试连接。那么，第二个客户端又来连接 zk 集群，其是在前面打散的基础上采用轮询方式选择 Server，还是又重新打散后再进行轮询连接尝试？
【RA】首先要清楚，这个打散操作发生在客户端，所以是每个客户端都会进行打散，然后再进行连接尝试。

【Q-09】zk 客户端维护着会话超时管理，请谈一下你对此的认识。
【RA】zk 客户端维护着会话超时管理，主要管理的超时有两类：读超时与连接超时。当客户端长时间没有收到服务端的请求响应或心跳响应时，会发生读超时；当客户端发出连接请求后，长时间没有收到服务端的连接成功 ACK，此时发生连接超时。无论哪类超时，都会抛出 SessionTimeoutException 异常。但若是读超时，则会发出 Ping 连接请求，Ping 失败则会关闭连接；若是连接超时，则会再次连接，直到重连策略不满足，关闭连接。

【Q-10】zk 时 CP 的，zk 集群在数据同步或 leader 选举时是不对外提供服务的，那岂不是用户体验非常不好？请谈一下你对此的看法。
【RA】对于 zk 的 CP，其实对于客户端来说，一般是感知不到的。因为当客户端连接 zk 集群时，若集群恰好由于数据同步或 leader 选举而不对外提供服务，那么，客户端的此次连接是失败的。所以，其就会尝试着按照重试策略再连接。只要不超时就会一直连。而 zk 集群中的数据同步与 leader 选举是很快的，在客户端重试连接过程中已经完成。此时客户端再连就会连接成功。所以，对于客户端来说，zk 的 CP 是感知不到的。

第 3 次直播课

【Q-01】zk Client 在连接 zk 时会发生连接丢失事件，什么是连接丢失？请谈一下你的认识。
【RA】因为网络抖动等原因导致客户端长时间收不到服务端的心跳回复，客户端就会引发连接丢失。连接丢失会引发客户端自动从 zk 地址列表中逐个尝试重新连接，直到重连成功，或按照指定的重试策略终止。

【Q-02】zk Client 在连接 zk 时会发生会话转移事件，什么是会话转移？请谈一下你的认识。

【RA】当发生连接丢失后，客户端又以原来的 sessionId 重新连接上了 zk 服务器。若重连上的服务器不是原来的服务器，那么客户端就需要更新本地 zk 对象中的相关信息。这就是会话转移，即回话从一个服务端转移到了另一个服务器。

服务端会由于长时间没有收到某客户端的心跳，即该客户端会话在服务端出现长时间的空闲状态时，服务器会认为该客户端已经挂了，然后会将该会话从服务器中删除。不过，在空闲超时时间范围内，该客户端又重新连接上了服务器，此时服务器并不会删除该会话，且 sessionId 仍是原来的。

【Q-03】zk Client 在连接 zk 时会发生会话失效事件，什么是会话失效？请谈一下你的认识。
【RA】若客户端连接丢失后，在会话空闲超时范围内没有连接上服务器，则服务器会将该会话从服务器中删除。

由于客户端的重连与服务端的会话删除是两个独立运行于不同主机的进程，所以客户端的重连与服务端的会话删除没有关系。

若在服务端将某客户端的会话删除后，而该客户端仍使用原来的 sessionId 又重新连接上了服务器。那么这个会话是失效的，因为服务端根本就没有该会话的信息。此时服务端会向客户端发送关闭该连接的响应。这也是客户端知道其与服务端连接失效的途径。

【Q-04】zk 中的会话空闲超时管理采用的是分桶策略。请谈一下你对分桶策略的认识。
【RA】分桶策略是一种查找空闲超时会话的方式，是将空闲超时时间相近的会话放到同一个会话桶中来进行管理，以减少管理的复杂度。在检查超时时，只需要检查桶中剩下的会话即可，因为在该桶的时间范围内没有超时的会话已经被移出了桶，而桶中存在的会话就是超时的会话。

【Q-05】zk 中的会话空闲超时管理采用的是分桶策略。什么是会话桶？里面存放的是什么？从源码角度请谈一下你的认识。
【RA】zk 中的会话空闲超时管理中分桶策略中存在会话桶的概念，从源码角度来说，会话桶就是一个实例，其中维护着一个 Set 集合，Set 集合中存放的是会话实例。

当 zk 集群启动后，其会将时间按照固定长度划分为若干段，每一段生成一个会话桶。这些会话桶被存放在一个 Map 集合中，Map 的 value 为会话桶，key 为这个会话桶的标识。这个标识就是这个会话桶对应时间段的最大边界值，即是一个时间。

当有客户端会话连接 Server 成功后，Server 首先会根据会话连接成功的时间及设置的空闲超时时长计算出其空闲超时时间点，然后会根据该时间点计算出该会话所对应的会话桶标识，然后根据标识找到会话桶，将该会话对象放入到该会话桶中。

【Q-06】zk 中的会话空闲超时管理采用的是分桶策略。会话桶中的会话会发生换桶，什么时候会进行换桶？如何换桶呢？从源码角度请谈一下你的认识。
【RA】zk 中的会话空闲超时管理中分桶策略中存在会话桶的概念，从源码角度来说，会话发生换桶说明会话没有发生空闲过期，只有当会话与服务器发生了交互时才不会过期。所以，当某会话与服务器发生了一次交互时，就会马上判断这个会话是否需要换桶。

如何判断是否需要换桶呢？首先会重新计算会话的空闲超时时间点，然后根据这个时间点再计算出其应该对应的会话桶标识 expireTime。再然后比较 expireTime 与当前会话所在的会话桶标识 tickTime（包含在会话实例中）的大小关系。若 expireTime 等于 tickTime，则说明当前会话不需要换桶；若 expireTime 大于 tickTime，则说明该换桶了；不可能出现 expireTime 小于 tickTime 的情况，因为时间值只会增大。

如何进行换桶呢？从会话当前所在桶中删除当前会话，然后更新当前会话所在的会话桶标识 tickTime，然后再根据这个 tickTime 从会话 Map 中查找响应的会话桶。若存在，则直接将会话放入桶中。若没有，则创建一个，并放入到会话 Map 中，然后再将会话放入桶中。

【Q-07】zk 中的会话空闲超时管理采用的是分桶策略。该分桶策略中会话空闲超时判断发生在哪里？超时发生后的处理发生在哪里？都做了哪些处理呢？请谈一下你的认识。

【RA】zk 中的会话空闲超时管理中分桶策略中存在会话桶的概念。当服务器启动时会创建并启动一个不会停止的线程，专门用于查找过期的会话桶。然后将过期的会话桶从会话桶 Map 中删除，并关闭该会话桶中的所有会话。