<https://mp.weixin.qq.com/s?__biz=MzIyNDU2ODA4OQ==&mid=2247484047&idx=1&sn=101f940479a708ab4ade6116ec1ec381&chksm=e80db4f9df7a3def7e4546389db9a6a1503e4e9bbaf78e7a10dd528d8c193b2479d4de28ffd1&scene=21#wechat_redirect>

# ZooKeeper面试那些事儿

### 1.ZooKeeper是什么？

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，它是集群的管理者，监视着集群中各个节点的状态根据节点提交的反馈进行下一步合理操作。最终，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

客户端的读请求可以被集群中的任意一台机器处理，如果读请求在节点上注册了监听器，这个监听器也是由所连接的zookeeper机器来处理。对于写请求，这些请求会同时发给其他zookeeper机器并且达成一致后，请求才会返回成功。因此，随着zookeeper的集群机器增多，读请求的吞吐会提高但是写请求的吞吐会下降。

有序性是zookeeper中非常重要的一个特性，所有的更新都是全局有序的，每个更新都有一个唯一的时间戳，这个时间戳称为zxid（Zookeeper Transaction Id）。而读请求只会相对于更新有序，也就是读请求的返回结果中会带有这个zookeeper最新的zxid。

### 2.ZooKeeper提供了什么？

1、文件系统  
2、通知机制

### 3.Zookeeper文件系统

Zookeeper提供一个多层级的节点命名空间（节点称为znode）。与文件系统不同的是，这些节点都可以设置关联的数据，而文件系统中只有文件节点可以存放数据而目录节点不行。Zookeeper为了保证高吞吐和低延迟，在内存中维护了这个树状的目录结构，这种特性使得Zookeeper不能用于存放大量的数据，每个节点的存放数据上限为1M。

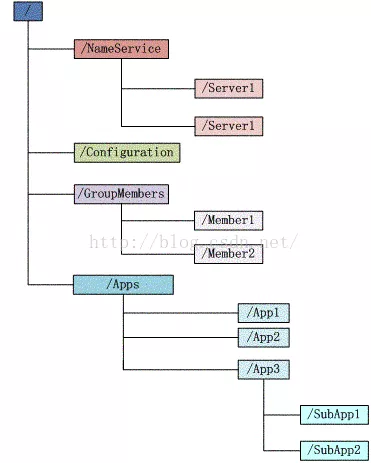
### 4.四种类型的znode

**1、PERSISTENT-持久化目录节点**  
客户端与zookeeper断开连接后，该节点依旧存在

**2、PERSISTENT\_SEQUENTIAL-持久化顺序编号目录节点**  
客户端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进行顺序编号

**3、EPHEMERAL-临时目录节点**  
客户端与zookeeper断开连接后，该节点被删除

**4、EPHEMERAL\_SEQUENTIAL-临时顺序编号目录节点**  
客户端与zookeeper断开连接后，该节点被删除，只是Zookeeper给该节点名称进行顺序编号



### 5.Zookeeper通知机制

client端会对某个znode建立一个watcher事件，当该znode发生变化时，这些client会收到zk的通知，然后client可以根据znode变化来做出业务上的改变等。

### 6.Zookeeper做了什么？

1、命名服务  
2、配置管理  
3、集群管理  
4、分布式锁  
5、队列管理

### 7.zk的命名服务（文件系统）

命名服务是指通过指定的名字来获取资源或者服务的地址，利用zk创建一个全局的路径，即是唯一的路径，这个路径就可以作为一个名字，指向集群中的集群，提供的服务的地址，或者一个远程的对象等等。

### 8.zk的配置管理（文件系统、通知机制）

程序分布式的部署在不同的机器上，将程序的配置信息放在zk的znode下，当有配置发生改变时，也就是znode发生变化时，可以通过改变zk中某个目录节点的内容，利用watcher通知给各个客户端，从而更改配置。

### 9.Zookeeper集群管理（文件系统、通知机制）

所谓集群管理无在乎两点：是否有机器退出和加入、选举master。

对于第一点，所有机器约定在父目录下创建临时目录节点，然后监听父目录节点的子节点变化消息。一旦有机器挂掉，该机器与 zookeeper的连接断开，其所创建的临时目录节点被删除，所有其他机器都收到通知：某个兄弟目录被删除，于是，所有人都知道：它上船了。

新机器加入也是类似，所有机器收到通知：新兄弟目录加入，highcount又有了，对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为master就好。

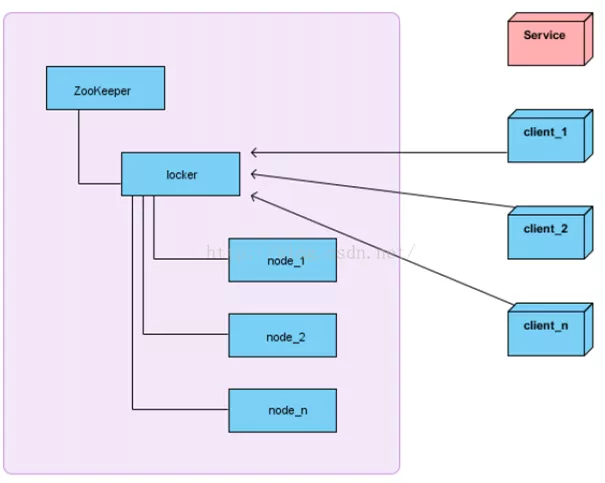
### 10.Zookeeper分布式锁（文件系统、通知机制）

有了zookeeper的一致性文件系统，锁的问题变得容易。锁服务可以分为两类，一个是保持独占，另一个是控制时序。

对于第一类，我们将zookeeper上的一个znode看作是一把锁，通过createznode的方式来实现。所有客户端都去创建 /distribute\_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。用完删除掉自己创建的distribute\_lock 节点就释放出锁。

对于第二类， /distribute\_lock 已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选master一样，编号最小的获得锁，用完删除，依次方便。

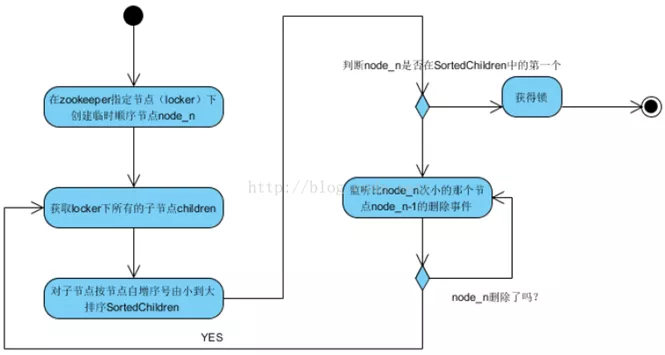
### 11.获取分布式锁的流程



在获取分布式锁的时候在locker节点下创建临时顺序节点，释放锁的时候删除该临时节点。客户端调用createNode方法在locker下创建临时顺序节点，

然后调用getChildren(“locker”)来获取locker下面的所有子节点，注意此时不用设置任何Watcher。客户端获取到所有的子节点path之后，如果发现自己创建的节点在所有创建的子节点序号最小，那么就认为该客户端获取到了锁。

如果发现自己创建的节点并非locker所有子节点中最小的，说明自己还没有获取到锁，此时客户端需要找到比自己小的那个节点，然后对其调用exist()方法，同时对其注册事件监听器。之后，让这个被关注的节点删除，则客户端的Watcher会收到相应通知，此时再次判断自己创建的节点是否是locker子节点中序号最小的，如果是则获取到了锁，如果不是则重复以上步骤继续获取到比自己小的一个节点并注册监听。当前这个过程中还需要许多的逻辑判断。



代码的实现主要是基于互斥锁，获取分布式锁的重点逻辑在于BaseDistributedLock，实现了基于Zookeeper实现分布式锁的细节。

### 12.Zookeeper队列管理（文件系统、通知机制）

两种类型的队列：

* 同步队列，当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达。
* 队列按照 FIFO 方式进行入队和出队操作。

第一类，在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。

第二类，和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。在特定的目录下创建PERSISTENT\_SEQUENTIAL节点，创建成功时Watcher通知等待的队列，队列删除序列号最小的节点用以消费。此场景下Zookeeper的znode用于消息存储，znode存储的数据就是消息队列中的消息内容，SEQUENTIAL序列号就是消息的编号，按序取出即可。由于创建的节点是持久化的，所以不必担心队列消息的丢失问题。

### 13.Zookeeper数据复制

Zookeeper作为一个集群提供一致的数据服务，自然，它要在所有机器间做数据复制。数据复制的好处：

* 容错：一个节点出错，不致于让整个系统停止工作，别的节点可以接管它的工作；
* 提高系统的扩展能力 ：把负载分布到多个节点上，或者增加节点来提高系统的负载能力；
* 提高性能：让客户端本地访问就近的节点，提高用户访问速度。

从客户端读写访问的透明度来看，数据复制集群系统分下面两种：

* 写主(WriteMaster) ：对数据的修改提交给指定的节点。读无此限制，可以读取任何一个节点。这种情况下客户端需要对读与写进行区别，俗称读写分离；
* 写任意(Write Any)：对数据的修改可提交给任意的节点，跟读一样。这种情况下，客户端对集群节点的角色与变化透明。

对zookeeper来说，它采用的方式是写任意。通过增加机器，它的读吞吐能力和响应能力扩展性非常好，而写，随着机器的增多吞吐能力肯定下降（这也是它建立observer的原因），而响应能力则取决于具体实现方式，是延迟复制保持最终一致性，还是立即复制快速响应。

### 14.Zookeeper工作原理

Zookeeper 的核心是原子广播，这个机制保证了各个Server之间的同步。实现这个机制的协议叫做Zab协议。Zab协议有两种模式，它们分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数Server完成了和 leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和Server具有相同的系统状态。

### 15.zookeeper是如何保证事务的顺序一致性的？

zookeeper采用了递增的事务Id来标识，所有的proposal（提议）都在被提出的时候加上了zxid，zxid实际上是一个64位的数字，高32位是epoch（时期; 纪元; 世; 新时代）用来标识leader是否发生改变，如果有新的leader产生出来，epoch会自增，低32位用来递增计数。当新产生proposal的时候，会依据数据库的两阶段过程，首先会向其他的server发出事务执行请求，如果超过半数的机器都能执行并且能够成功，那么就会开始执行。

### 16.Zookeeper 下 Server工作状态

每个Server在工作过程中有三种状态：

* LOOKING：当前Server不知道leader是谁，正在搜寻
* LEADING：当前Server即为选举出来的leader
* FOLLOWING：leader已经选举出来，当前Server与之同步

### 17.zookeeper是如何选取主leader的？

当leader崩溃或者leader失去大多数的follower，这时zk进入恢复模式，恢复模式需要重新选举出一个新的leader，让所有的Server都恢复到一个正确的状态。Zk的选举算法有两种：一种是基于basic paxos实现的，另外一种是基于fast paxos算法实现的。系统默认的选举算法为fast paxos。

**1、Zookeeper选主流程(basic paxos)**

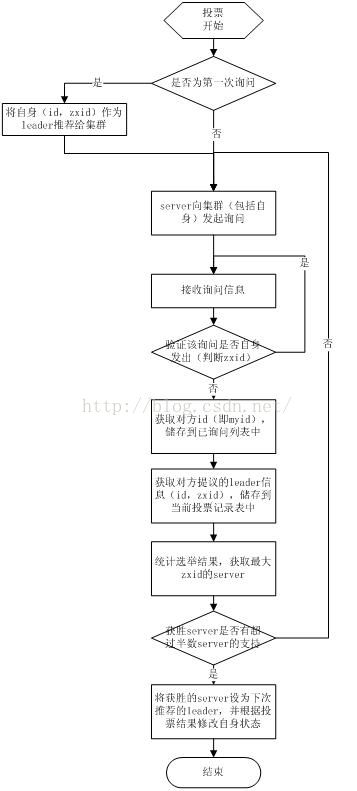
1.选举线程由当前Server发起选举的线程担任，其主要功能是对投票结果进行统计，并选出推荐的Server；

2.选举线程首先向所有Server发起一次询问(包括自己)；

3.选举线程收到回复后，验证是否是自己发起的询问(验证zxid是否一致)，然后获取对方的id(myid)，并存储到当前询问对象列表中，最后获取对方提议的leader相关信息(id,zxid)，并将这些信息存储到当次选举的投票记录表中；

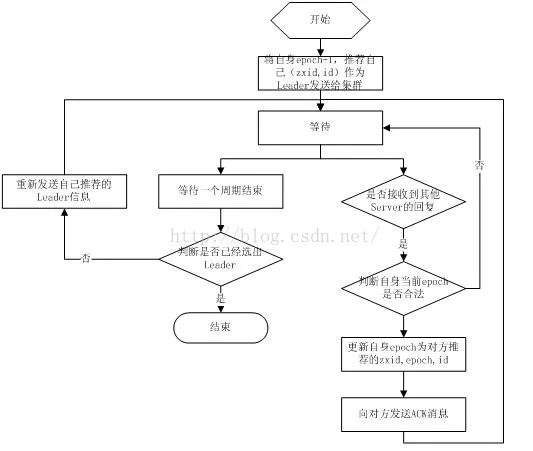
4.收到所有Server回复以后，就计算出zxid最大的那个Server，并将这个Server相关信息设置成下一次要投票的Server；

5.线程将当前zxid最大的Server设置为当前Server要推荐的Leader，如果此时获胜的Server获得n/2 + 1的Server票数，设置当前推荐的leader为获胜的Server，将根据获胜的Server相关信息设置自己的状态，否则，继续这个过程，直到leader被选举出来。通过流程分析我们可以得出：要使Leader获得多数Server的支持，则Server总数必须是奇数2n+1，且存活的Server的数目不得少于n+1. 每个Server启动后都会重复以上流程。在恢复模式下，如果是刚从崩溃状态恢复的或者刚启动的server还会从磁盘快照中恢复数据和会话信息，zk会记录事务日志并定期进行快照，方便在恢复时进行状态恢复。



**2、Zookeeper选主流程(basic paxos)**

fast paxos流程是在选举过程中，某Server首先向所有Server提议自己要成为leader，当其它Server收到提议以后，解决epoch和 zxid的冲突，并接受对方的提议，然后向对方发送接受提议完成的消息，重复这个流程，最后一定能选举出Leader。



### 18.Zookeeper同步流程

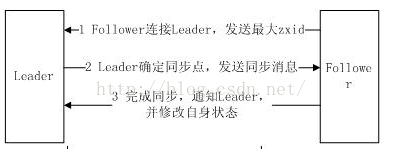
选完Leader以后，zk就进入状态同步过程。

1. Leader等待server连接；
2. Follower连接leader，将最大的zxid发送给leader；
3. Leader根据follower的zxid确定同步点；
4. 完成同步后通知follower 已经成为uptodate状态；
5. Follower收到uptodate消息后，又可以重新接受client的请求进行服务了。

### 19.分布式通知和协调

对于系统调度来说：操作人员发送通知实际是通过控制台改变某个节点的状态，然后zk将这些变化发送给注册了这个节点的watcher的所有客户端。

对于执行情况汇报：每个工作进程都在某个目录下创建一个临时节点。并携带工作的进度数据，这样汇总的进程可以监控目录子节点的变化获得工作进度的实时的全局情况。



### 20.机器中为什么会有leader？

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，于是就需要进行leader选举。

### 21.zk节点宕机如何处理？

Zookeeper本身也是集群，推荐配置不少于3个服务器。Zookeeper自身也要保证当一个节点宕机时，其他节点会继续提供服务。

如果是一个Follower宕机，还有2台服务器提供访问，因为Zookeeper上的数据是有多个副本的，数据并不会丢失；

如果是一个Leader宕机，Zookeeper会选举出新的Leader。

ZK集群的机制是只要超过半数的节点正常，集群就能正常提供服务。只有在ZK节点挂得太多，只剩一半或不到一半节点能工作，集群才失效。

所以

* 3个节点的cluster可以挂掉1个节点(leader可以得到2票>1.5)
* 2个节点的cluster就不能挂掉任何1个节点了(leader可以得到1票<=1)

### 22.zookeeper负载均衡和nginx负载均衡区别

zk的负载均衡是可以调控，nginx只是能调权重，其他需要可控的都需要自己写插件；但是nginx的吞吐量比zk大很多，应该说按业务选择用哪种方式。

### 23.zookeeper watch机制

Watch机制官方声明：一个Watch事件是一个一次性的触发器，当被设置了Watch的数据发生了改变的时候，则服务器将这个改变发送给设置了Watch的客户端，以便通知它们。

**Zookeeper机制的特点：**

1、一次性触发数据发生改变时，一个watcher event会被发送到client，但是client只会收到一次这样的信息。

2、watcher event异步发送watcher的通知事件从server发送到client是异步的，这就存在一个问题，不同的客户端和服务器之间通过socket进行通信，由于网络延迟或其他因素导致客户端在不通的时刻监听到事件，由于Zookeeper本身提供了ordering guarantee，即客户端监听事件后，才会感知它所监视znode发生了变化。所以我们使用Zookeeper不能期望能够监控到节点每次的变化。Zookeeper只能保证最终的一致性，而无法保证强一致性。

3、数据监视Zookeeper有数据监视和子数据监视getdata() and exists()设置数据监视，getchildren()设置了子节点监视。

4、注册watcher getData、exists、getChildren

5、触发watcher create、delete、setData

6、setData()会触发znode上设置的data watch（如果set成功的话）。一个成功的create() 操作会触发被创建的znode上的数据watch，以及其父节点上的child watch。而一个成功的delete()操作将会同时触发一个znode的data watch和child watch（因为这样就没有子节点了），同时也会触发其父节点的child watch。

7、当一个客户端连接到一个新的服务器上时，watch将会被以任意会话事件触发。当与一个服务器失去连接的时候，是无法接收到watch的。而当client重新连接时，如果需要的话，所有先前注册过的watch，都会被重新注册。通常这是完全透明的。只有在一个特殊情况下，watch可能会丢失：对于一个未创建的znode的exist watch，如果在客户端断开连接期间被创建了，并且随后在客户端连接上之前又删除了，这种情况下，这个watch事件可能会被丢失。

8、Watch是轻量级的，其实就是本地JVM的Callback，服务器端只是存了是否有设置了Watcher的布尔类型