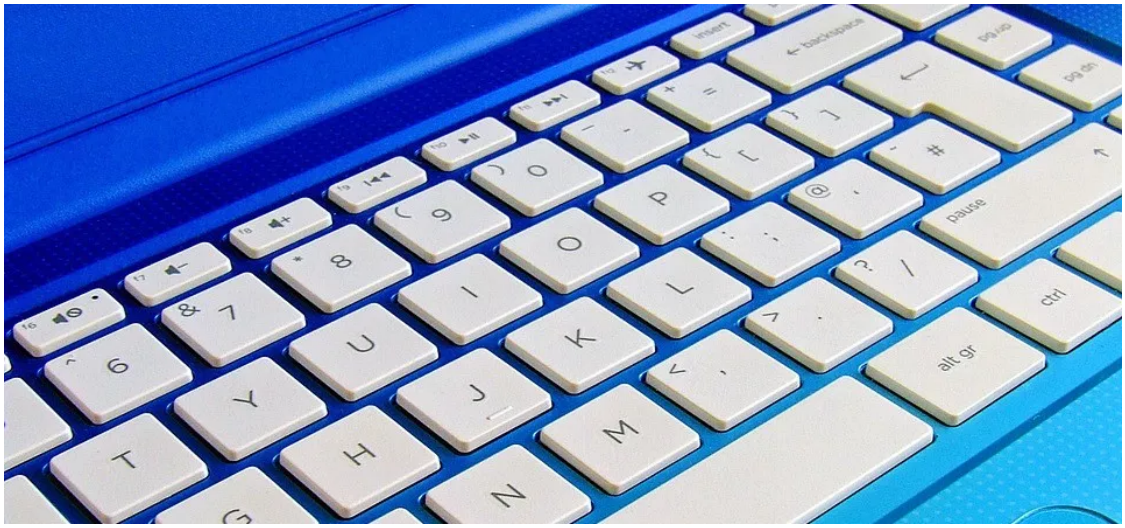
ZooKeeper基本原理

大神解析 民工哥技术之路 1周前

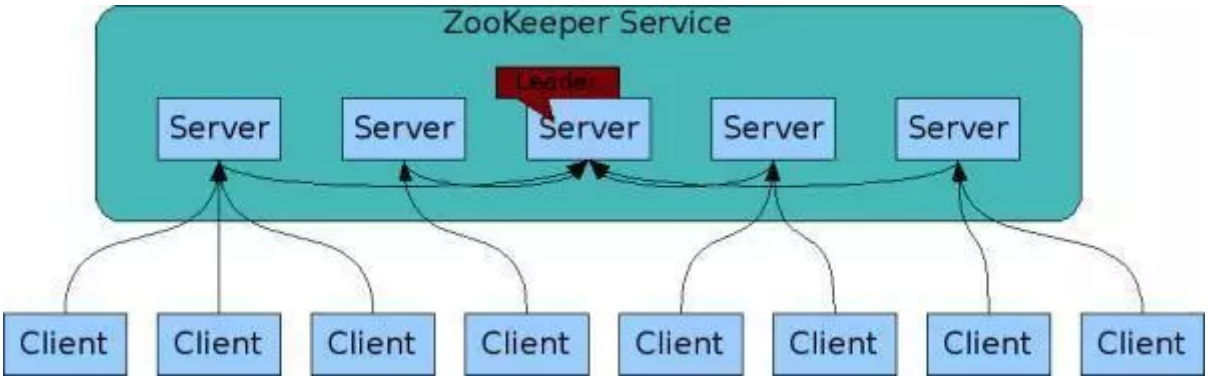
点击上方“民工哥技术之路”，选择“置顶公众号”

有趣有内涵的文章第一时间送达！



ZooKeeper简介

ZooKeeper是一个开放源码的分布式应用程序协调服务，它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务，配置维护和命名服务等。



ZooKeeper设计目的

1.最终一致性：client不论连接到哪个Server，展示给它都是同一个视图，这是zookeeper最重要的性能。

2.可靠性：具有简单、健壮、良好的性能，如果消息m被到一台服务器接受，那么它将被所有的服务器接受。

3.实时性：Zookeeper保证客户端将在一个时间间隔范围内获得服务器的更新信息，或者服务器失效的信息。但由于网络延时等原因，Zookeeper不能保证两个客户端能同时

得到刚更新的数据，如果需要最新数据，应该在读数据之前调用sync()接口。

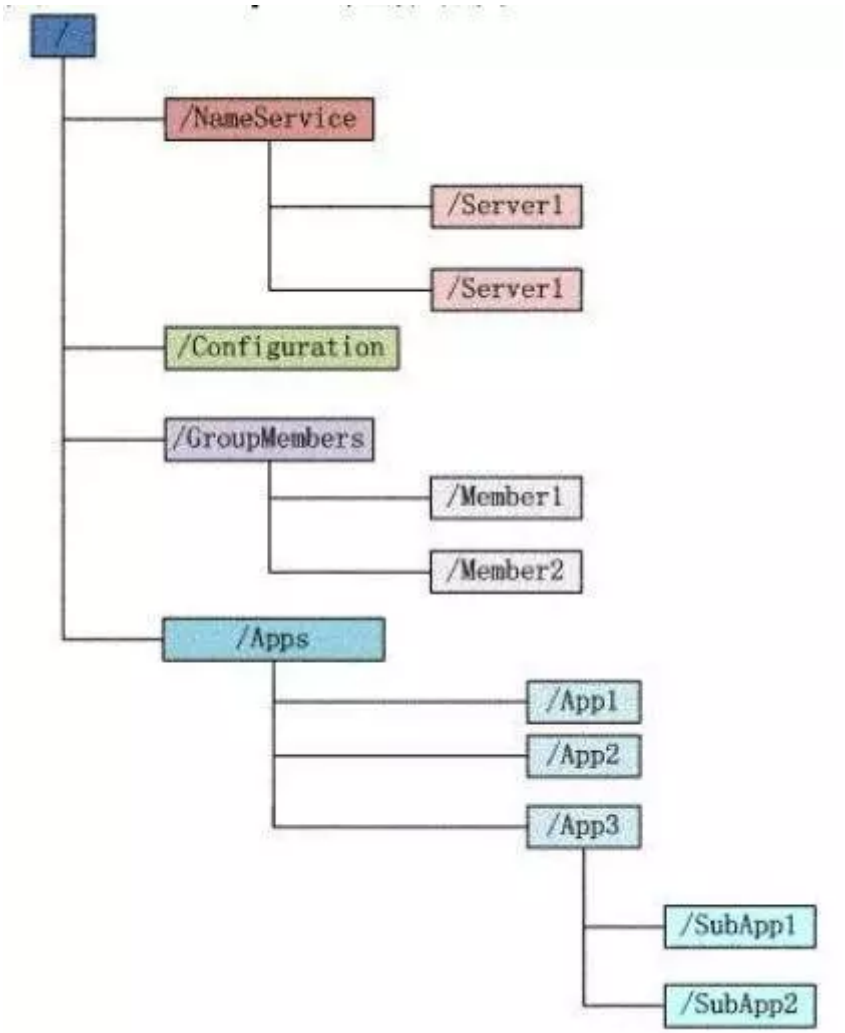
4.等待无关（wait-free）：慢的或者失效的client不得干预快速的client的请求，使得每个client都能有效的等待。

5.原子性：更新只能成功或者失败，没有中间状态。

6.顺序性：包括全局有序和偏序两种：全局有序是指如果在一台服务器上消息a在消息b前发布，则在所有Server上消息a都将在消息b前被发布；偏序是指如果一个消息b在消息a后被同一个发送者发布，a必将排在b前面。

ZooKeeper数据模型

Zookeeper会维护一个具有层次关系的数据结构，它非常类似于一个标准的文件系统，如图所示：



Zookeeper这种数据结构有如下这些特点：

1）每个子目录项如NameService都被称作为znode，这个znode是被它所在的路径唯一标识，如Server1这个znode的标识为/NameService/Server1。

2）znode可以有子节点目录，并且每个znode可以存储数据，注意EPHEMERAL（临时的）类型的目录节点不能有子节点目录。

3）znode是有版本的（version），每个znode中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据，version号自动增加。

4）znode的类型：

Persistent 节点，一旦被创建，便不会意外丢失，即使服务器全部重启也依然存在。每个 Persist 节点即可包含数据，也可包含子节点。

Ephemeral 节点，在创建它的客户端与服务器间的 Session 结束时自动被删除。服务器重启会导致 Session 结束，因此 Ephemeral 类型的 znode 此时也会自动删除。

Non-sequence 节点，多个客户端同时创建同一 Non-sequence 节点时，只有一个可创建成功，其它匀失败。并且创建出的节点名称与创建时指定的节点名完全一样。

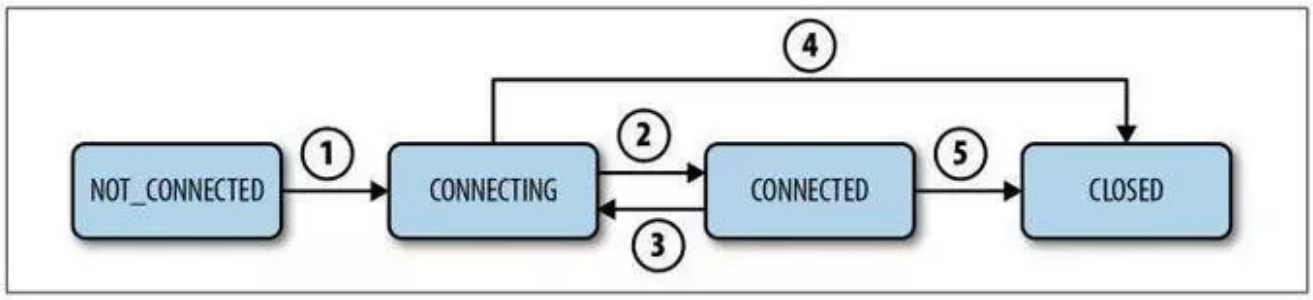
Sequence 节点，创建出的节点名在指定的名称之后带有10位10进制数的序号。多个客户端创建同一名称的节点时，都能创建成功，只是序号不同。

5）znode可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是Zookeeper的核心特性，Zookeeper的很多功能都是基于这个特性实现的。

6）ZXID：每次对Zookeeper的状态的改变都会产生一个zxid（ZooKeeper Transaction Id），zxid是全局有序的，如果zxid1小于zxid2，则zxid1在zxid2之前发生。

ZooKeeper Session

Client和Zookeeper集群建立连接，整个session状态变化如图所示：



如果Client因为Timeout和Zookeeper Server失去连接，client处在CONNECTING状态，会自动尝试再去连接Server，如果在session有效期内再次成功连接到某个Server，则回到CONNECTED状态。

注意：如果因为网络状态不好，client和Server失去联系，client会停留在当前状态，会尝试主动再次连接Zookeeper Server。client不能宣称自己的session expired，session expired是由Zookeeper Server来决定的，client可以选择自己主动关闭session。

ZooKeeper Watch

Zookeeper watch是一种监听通知机制。Zookeeper所有的读操作getData(), getChildren()和 exists()都可以设置监视(watch)，监视事件可以理解为一次性的触发器，官方定义如下： a watch event is one-time trigger, sent to the client that set the watch, whichoccurs when the data for which the watch was set changes。Watch的三个关键点：

*（一次性触发）One-time trigger

当设置监视的数据发生改变时，该监视事件会被发送到客户端，例如，如果客户端调用了 getData("/znode1", true) 并且稍后 /znode1 节点上的数据发生了改变或者被删除了，客户端将会获取到 /znode1 发生变化的监视事件，而如果 /znode1 再一次发生了变化，除非客户端再次对/znode1 设置监视，否则客户端不会收到事件通知。

*（发送至客户端）Sent to the client

Zookeeper客户端和服务端是通过 socket 进行通信的，由于网络存在故障，所以监视事件很有可能不会成功地到达客户端，监视事件是异步发送至监视者的，Zookeeper 本身提供了顺序保证(ordering guarantee)：即客户端只有首先看到了监视事件后，才会感知到它所设置监视的znode发生了变化(a client will never see a change for which it has set a watch until it first sees the watch event)。网络延迟或者其他因素可能导致不同的客户端在不同的时刻感知某一监视事件，但是不同的客户端所看到的一切具有一致的顺序。

*（被设置 watch 的数据）The data for which the watch was set

这意味着znode节点本身具有不同的改变方式。你也可以想象 Zookeeper 维护了两条监视链表：数据监视和子节点监视(data watches and child watches) getData() 和 exists()设置数据监视，getChildren()设置子节点监视。或者你也可以想象 Zookeeper 设置的不同监视返回不同的数据，getData() 和 exists() 返回znode节点的相关信息，而getChildren() 返回子节点列表。因此，setData() 会触发设置在某一节点上所设置的数据监视（假定数据设置成功），而一次成功的create() 操作则会出发当前节点上所设置的数据监视以及父节点的子节点监视。一次成功的 delete操作将会触发当前节点的数据监视和子节点监视事件，同时也会触发该节点父节点的child watch。

Zookeeper 中的监视是轻量级的，因此容易设置、维护和分发。当客户端与Zookeeper 服务器失去联系时，客户端并不会收到监视事件的通知，只有当客户端重新连接后，若在必要的情况下，以前注册的监视会重新被注册并触发，对于开发人员来说这通常是透明的。只有一种情况会导致监视事件的丢失，即：通过exists()设置了某个znode节点的监视，但是如果某个客户端在此znode节点被创建和删除的时间间隔内与zookeeper服务器失去了联系，该客户端即使稍后重新连接 zookeeper服务器后也得不到事件通知。

Consistency Guarantees

Zookeeper是一个高效的、可扩展的服务，read和write操作都被设计为快速的，read比write操作更快。

顺序一致性（Sequential Consistency）：从一个客户端来的更新请求会被顺序执行。

原子性（Atomicity）：更新要么成功要么失败，没有部分成功的情况。

唯一的系统镜像（Single System Image）：无论客户端连接到哪个Server，看到系统镜像是一致的。

可靠性（Reliability）：更新一旦有效，持续有效，直到被覆盖。

时间线（Timeliness）：保证在一定的时间内各个客户端看到的系统信息是一致的。

ZooKeeper的工作原理

在zookeeper的集群中，各个节点共有下面3种角色和4种状态：

角色：leader,follower,observer

状态：leading,following,observing,looking

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（ZooKeeper Atomic Broadcast protocol）。Zab协议有两种模式，它们分别是恢复模式（Recovery选主）和广播模式（Broadcast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。

为了保证事务的顺序一致性，zookeeper采用了递增的事务id号（zxid）来标识事务。所有的提议（proposal）都在被提出的时候加上了zxid。实现中zxid是一个64位的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch，标识当前属于那个leader的统治时期。低32位用于递增计数。

每个Server在工作过程中有4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。

OBSERVING：observer的行为在大多数情况下与follower完全一致，但是他们不参加选举和投票，而仅仅接受(observing)选举和投票的结果。

Leader Election

当leader崩溃或者leader失去大多数的follower，这时候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。先介绍basic paxos流程：

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。

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。

Leader工作流程

Leader主要有三个功能：

1.恢复数据；

2.维持与follower的心跳，接收follower请求并判断follower的请求消息类型；

3.follower的消息类型主要有PING消息、REQUEST消息、ACK消息、REVALIDATE消息，根据不同的消息类型，进行不同的处理。

PING消息是指follower的心跳信息；REQUEST消息是follower发送的提议信息，包括写请求及同步请求；

ACK消息是follower的对提议的回复，超过半数的follower通过，则commit该提议；

REVALIDATE消息是用来延长SESSION有效时间。

Follower工作流程

Follower主要有四个功能：

1. 向Leader发送请求（PING消息、REQUEST消息、ACK消息、REVALIDATE消息）；

2.接收Leader消息并进行处理；

3.接收Client的请求，如果为写请求，发送给Leader进行投票；

4.返回Client结果。

Follower的消息循环处理如下几种来自Leader的消息：

1.PING消息：心跳消息

2.PROPOSAL消息：Leader发起的提案，要求Follower投票

3.COMMIT消息：服务器端最新一次提案的信息

4.UPTODATE消息：表明同步完成

5.REVALIDATE消息：根据Leader的REVALIDATE结果，关闭待revalidate的session还是允许其接受消息

6.SYNC消息：返回SYNC结果到客户端，这个消息最初由客户端发起，用来强制得到最新的更新。

Zab: Broadcasting State Updates

Zookeeper Server接收到一次request，如果是follower，会转发给leader，Leader执行请求并通过Transaction的形式广播这次执行。Zookeeper集群如何决定一个Transaction是否被commit执行？通过“两段提交协议”（a two-phase commit）：

Leader给所有的follower发送一个PROPOSAL消息。

一个follower接收到这次PROPOSAL消息，写到磁盘，发送给leader一个ACK消息，告知已经收到。

当Leader收到法定人数（quorum）的follower的ACK时候，发送commit消息执行。

Zab协议保证：

1） 如果leader以T1和T2的顺序广播，那么所有的Server必须先执行T1，再执行T2。

2） 如果任意一个Server以T1、T2的顺序commit执行，其他所有的Server也必须以T1、T2的顺序执行。

“两段提交协议”最大的问题是如果Leader发送了PROPOSAL消息后crash或暂时失去连接，会导致整个集群处在一种不确定的状态（follower不知道该放弃这次提交还是执行提交）。Zookeeper这时会选出新的leader，请求处理也会移到新的leader上，不同的leader由不同的epoch标识。切换Leader时，需要解决下面两个问题：

1. Never forget delivered messages

Leader在COMMIT投递到任何一台follower之前crash，只有它自己commit了。新Leader必须保证这个事务也必须commit。

2. Let go of messages that are skipped

Leader产生某个proposal，但是在crash之前，没有follower看到这个proposal。该server恢复时，必须丢弃这个proposal。

Zookeeper会尽量保证不会同时有2个活动的Leader，因为2个不同的Leader会导致集群处在一种不一致的状态，所以Zab协议同时保证：

1） 在新的leader广播Transaction之前，先前Leader commit的Transaction都会先执行。

2） 在任意时刻，都不会有2个Server同时有法定人数（quorum）的支持者。

这里的quorum是一半以上的Server数目，确切的说是有投票权力的Server（不包括Observer）。

总结：简单介绍了Zookeeper的基本原理，数据模型，Session，Watch机制，一致性保证，Leader Election，Leader和Follower的工作流程和Zab协议。

更多精彩文章，请点击**“阅读原文”**

推荐阅读

[强烈推荐] 精心整理 | 公众号文章目录大全

民工哥的十年故事续集：杭漂十年，今撤霸都！

图解分布式架构的演进过程！

看看80万程序员怎么评论：程序员工资为什么这么高？

如何提高程序员（技术人）自我生产力？

面试中有哪些经典的数据库问题？

如何伪装成一个年薪 50 万刀以上的码农？

趣文 | 20幅程序员才能看懂的趣图

Redis面试总结

前腾讯员工（创业者）离世、安全牛创始人于昨日猝然辞世！

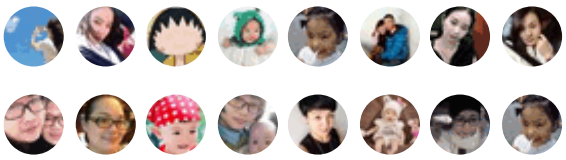
·end·

—写文不易，你的转发就是对我最大的支持—

我们一起愉快的玩耍吧

喜欢这篇文章

目前40000+人已关注加入我们



关注公众号点击菜单**“微信群”**入群与小伙伴一起交流吧！

阅读原文