链滴

Zookeeper 详解

作者：wlgzs-sjl

原文链接：https://ld246.com/article/1605778335001

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20200706.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="1--概述">1、 概述</h2>
Zookeeper 是一个开源的分布式协调服务，从设计模式角度来理解：是一个基于观察者模式设
的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数
的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应

<img src="https://ld246.com/images/img-loading.svg" alt="01.png" data-src="https://b3
ogfile.com/file/2020/11/01-c9bb2d36.png?imageView2/2/interlace/1/format/jpg">

服务端启动时去注册信息（创建都是临时节点）
获取到当前在线服务器列表，并且注册监听
服务器节点下线
服务器节点上下线事件通知
重新再去获取服务器列表，并注册监听

ZooKeeper = 文件系统 + 通知机制
<h2 id="2-特点">2、特点</h2>
<img src="https://ld246.com/images/img-loading.svg" alt="02.png" data-src="https://b3
ogfile.com/file/2020/11/02-ecdd3257.png?imageView2/2/interlace/1/format/jpg">

Zookeeper：一个领导者（Leader），多个跟随者（Follower)组成的集群。
集群中只要有半数以上节点存活，Zookeeper 集群就能正常服务(节点多
奇数)。
全局数据一致：每个 Server 保存一份相同的数据副本，Client 无论连接到哪个 Server，数据都
一致的。
更新请求顺序进行，来自同一个 Client 的更新请求按其发送顺序依次执行。
数据更新原子性，一次数据更新要么成功，要么失败。
实时性，在一定时间范围内，Client 能读到最新数据。

<h2 id="3-数据结构">3、数据结构</h2>
Zookeeper 数据模型的结构与 Unix 文件系统很类似，整体上可以看作是一棵树， 每个节点称
一个 ZNode。每一个 ZNode 默认能够存储 1MB 的数据， 每个 ZNode 都可以通过其路径唯一标
。
<img src="https://ld246.com/images/img-loading.svg" alt="03.png" data-src="https://b3
ogfile.com/file/2020/11/03-36c8f8c4.png?imageView2/2/interlace/1/format/jpg">
和文件系统一样，我们能够自由的增加、删除 znode，在一个 znode 下增加、删除子 znode
唯一的不同在于 znode 是可以存储数据的。
有四种类型的 znode：

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
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</l
>
EPHEMERAL-临时目录节点
客户端与 zookeeper 断开连接后，该节点被删除
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

说明：创建 znode 时设置顺序标识，znode 名称后会附加一个值，顺序号是一个单调递增
计数器，由父节点维护
注意：在分布式系统中，顺序号可以被用于为所有的事件进行全局排序， 这样客户端可以通
顺序号推断事件的顺序

<h2 id="4-应用场景">4、应用场景</h2>
<h3 id="4-1-统一命名服务">4.1 统一命名服务</h3>
在分布式环境下，经常需要对应用/服务进行统一命名，便于识别。
例如：IP 不容易记住，而域名容易记住。

<h3 id="4-2-统一配置管理">4.2 统一配置管理</h3>

分布式环境下，配置文件同步非常常见
（1）一般要求一个集群中，所有节点的配置信息是一致的，比如 Kafka 集群
（2）对配置文件修改后，希望能够快速同步到各个节点上

配置管理可交由 ZooKeeper 实现
（1）可将配置信息写入 ZooKeeper 上的一个 Znode
（2）各个客户端服务器监听这个 Znode
（3）一旦 Znode 中的数据被修改， ZooKeeper 将通知各个客户端服务器

假设我们的程序是分布式部署在多台机器上，如果我们要改变程序的配置文件，需要逐台机器修改，非常麻烦，现在把这些配置全部放到 zookeeper 上去，保存在 zookeeper 的某个目录节点中然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 zookeeper 的通知，然后从 zookeeper 获取新的配置信息应用到系统中。

<h3 id="4-3-统一集群管理">4.3 统一集群管理</h3>

分布式环境中，实时掌握每个节点的状态是必要的
（1）可根据节点实时状态做出一些调整

ZooKeeper 可以实现实时监控节点状态变化
（1）可将节点信息写入 ZooKeeper 上的一个 ZNode
（2）监听这个 ZNode 可获取它的实时状态变化

<h3 id="4-4-软负载均衡">4.4 软负载均衡</h3>
在 Zookeeper 中记录每台服务器的访问数， 让访问数最少的服务器去处理最新的客户端请求

<h2 id="5-选举机制">5、选举机制</h2>

半数机制：集群中半数以上机器存活，集群可用。所以 Zookeeper 适合安装奇数台服务器。例，5 台服务器有 3 台存活，集群可用，而只有 2 台存活，集群不可用。
Zookeeper 虽然在配置文件中并没有指定 Master 和 Slave。 但是， Zookeeper 工作时，是有个节点为 Leader，其他则为 Follower， Leader 是通过内部的选举机制临时产的。

假设有五台服务器组成的 Zookeeper 集群，它们的 id 从 1-5，同时它们都是最新启动的，也

是没有历史数据，在存放数据量这一点上，都是一样的。假设这些服务器依序启动：
<img src="https://ld246.com/images/img-loading.svg" alt="08.png" data-src="https://b3logfile.com/file/2020/11/08-ce00901d.png?imageView2/2/interlace/1/format/jpg">

服务器 1 启动， 发起一次选举。服务器 1 投自己一票。此时服务器 1 票数一票，不够半数以上 3 票），选举无法完成，服务器 1 状态保持为 LOOKING；
服务器 2 启动， 再发起一次选举。服务器 1 和 2 分别投自己一票并交换选票信息：此时服务器 1 发现服务器 2 的 ID 比自己目前投票推举的（服务器 1）大，更改选票为推举服务器 2。此时服务器 1 票数 0 票，服务器 2 票数 2 票，没有半数以上结果，选举无法完成，服务器 1， 2 状态保持 LOOKING
服务器 3 启动， 发起一次选举。此时服务器 1 和 2 都会更改选票为服务器 3。此次投票结果：务器 1 为 0 票，服务器 2 为 0 票，服务器 3 为 3 票。此时服务器 3 的票数已经超过半数，服务器 3 当选 Leader。服务器 1， 2 更改状态为 FOLLOWING，服务器 3 更改状态为 LEADING；
服务器 4 启动， 发起一次选举。此时服务器 1， 2， 3 已经不是 LOOKING 状态，不会更改选信息。交换选票信息结果：服务器 3 为 3 票，服务器 4 为 1 票。此时服务器 4 服从多数，更改选票息为服务器 3，并更改状态为 FOLLOWING；
服务器 5 启动，同 4 一样更改状态为 FOLLOWING。

<h2 id="6-监听器原理">6、监听器原理</h2>

首先要有一个 main()线程
在 main 线程中创建 Zookeeper 客户端， 这时就会创建两个线程， 一个负责网络连接通信（ connet ）， 一个负责监听（ listener ）。
通过 connect 线程将注册的监听事件发送给 Zookeeper。
在 Zookeeper 的注册监听器列表中将注册的监听事件添加到列表中。
Zookeeper 监听到有数据或路径变化， 就会将这个消息发送给 listener 线程。
listener 线程内部调用了 process()方法。

<img src="https://ld246.com/images/img-loading.svg" alt="10.png" data-src="https://b3logfile.com/file/2020/11/10-96b10185.png?imageView2/2/interlace/1/format/jpg">
常见的监听:

监听节点数据的变化 <code>get path [watch]</code>
监听子节点增减的变化 <code>ls path [watch]</code>

<h2 id="7-写数据流程">7、写数据流程</h2>
<img src="https://ld246.com/images/img-loading.svg" alt="11.png" data-src="https://b3logfile.com/file/2020/11/11-71bddca8.png?imageView2/2/interlace/1/format/jpg">

Client 向 ZooKeeper 的 Server1 上写数据，发送一个写请求。
如果 Server1 不是 Leader，那么 Server1 会把接受到的请求进一步转发给 Leader，因为每个 ZooKeeper 的 Server 里面有一个是 Leader。这个 Leader 会将写请求广播给各个 Server， 比如 Server1 和 Server2，各个 Server 会将该写请求加入待写队列，并向 Leader 发送成信息。
当 Leader 收到半数以上 Server 的成功信息， 说明该写操作可以执行。Leader 会向各个 Server 发送提交信息，各个 Server 收到信息后会落实队列里的写请求， 此时写成功。/li>
Server1 会进一步通知 Client 数据写成功了，这时就认为整个写操作成功。
