黑客派

Zookeeper的功能以及工作原理

作者：hadoop

原文链接：https://hacpai.com/article/1471831066891

来源网站：黑客派

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
2.ZooKeeper 提供了什么？
1)文件系统
2)通知机制
3.Zookeeper 文件系统
每个子目录项如 NameService 都被称作为 znode，和文件系统一样，我们能够自由的增加、删 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。
 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

4.Zookeeper 通知机制
客户端注册监听它关心的目录节点，当目录节点发生变化（数据改变、被删除、子目录节点增加除）时，zookeeper 会通知客户端。
5.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

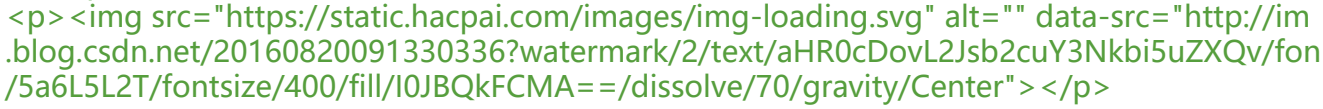
6.Zookeeper 命名服务
在 zookeeper 的文件系统里创建一个目录，即有唯一的 path。在我们使用 tborg 无法确定上程序的部署机器时即可与下游程序约定好 path，通过 path 即能互相探索发现。
7.Zookeeper 的配置管理
程序总是需要配置的，如果程序分散部署在多台机器上，要逐个改变配置就变得困难。现在把这配置全部放到 zookeeper 上去，保存在 Zookeeper 的某个目录节点中，然后所有相关应用程序对这目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中就好

8.Zookeeper 集群管理
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，所有机约定在父目录 GroupMembers 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，所有机器创建临顺序编号目录节点，每次选取编号最小的机器作为 master 就好。

<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://im
.blog.csdn.net/20160820090042896?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/fon
/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
9.Zookeeper 分布式锁
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
，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 c
eateznode 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
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， /dist
ibute_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
最小的获得锁，用完删除，依次方便。
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://im
.blog.csdn.net/20160820090209489?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/fon
/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
10.Zookeeper 队列管理
两种类型的队列：
 1、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
等待所有成员到达。
 2、队列按照 FIFO 方式进行入队和出队操作。
 第一类，在约定目录
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
序场景基本原理一致，入列有编号，出列按编号。
11.分布式与数据复制
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
的好处：
 1、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；
br> 2、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；<b
> 3、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：
 1、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
与写进行区别，俗称读写分离；
 2、写任意(Write Any)：对数据的修改可提交给任意的节点，
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
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
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。
12.Zookeeper 角色描述
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src=
http://img.blog.csdn.net/20160820090655858?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5
ZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
 </s
rong>
13.Zookeeper 与客户端
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src=
http://img.blog.csdn.net/20160820090755672?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5
ZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
 </s
rong>
14.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
要的性能。
 2.可靠性：具有简单、健壮、良好的性能，如果消息被到一台服务器接受，那么它
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
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
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。<b

> 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：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。

15.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 lea er 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。<b > 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（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，标识当前于那个 leader 的统治时期。低 32 位用于递增计数。
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，正搜寻
 LEADING：当前 Server 即为选举出来的 leader
 FOLLOWING：leader 已经选举出，当前 Server 与之同步
17.Zookeeper 选主流程(basic paxos)
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 paxo 。
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 会录事务日志并定期进行快照，方便在恢复时进行状态恢复。选主的具体流程图所示：



18.Zookeeper 选主流程（fast 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的消息，重复这个流程，最后一定能选举出 Leader。
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://img.blog.csdn.net/20160820091510556?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5ZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
19.Zookeeper 同步流程
选完 Leader 以后，zk 就进入状态同步过程。

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。

<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://im.blog.csdn.net/20160820091727354?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
20.Zookeeper 工作流程-Leader
1 .恢复数据；
 2 .维持与 Learner 的心跳，接收 Learner 请求并判断 Learner 的请求消息型；
 3 .Learner 的消息类型主要有 PING 消息、REQUEST 消息、ACK 消息、REVALIDATE 消，根据不同的消息类型，进行不同的处理。
 PING 消息是指 Learner 的心跳信息；
 REQUST 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 的对提的回复，超过半数的 Follower 通过，则 commit 该提议；
 REVALIDATE 消息是用来延长 SESSON 有效时间。
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://im.blog.csdn.net/20160820091954922?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">
21.Zookeeper 工作流程-Follower
Follower 主要有四个功能：
 1.向 Leader 发送请求（PING 消息、REQUEST 消息、ACK 息、REVALIDATE 消息）；
 2.接收 Leader 消息并进行处理；
 3.接收 Client 的请求，如为写请求，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 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。
<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="http://im.blog.csdn.net/20160820092442280?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center">

<a href="https://link.hacpai.com/forward?goto=http%3A%2F%2Fblog.csdn.net%2Fxqb_56148978%2Farticle%2Fdetails%2F52259381" target="_blank" rel="nofollow ugc">原文链接<a>