ZooKeeper 基础

作者：XinyiZhang
原文链接：https://ld246.com/article/1601261159130
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<p><img src="https://ld246.com/images/img-loading.svg" alt="Apache ZooKeeper" data-sr
="https://b3logfile.com/file/2020/06/solofetchupload946965565474417169-8c798185.png?i
ageView2/2/interlace/1/format/jpg"></p>
<h2 id="应用场景">应用场景</h2>

分布式协调:对 Zookeeper 中的数据做监听,一旦数据发生变动都会感知,为客户端进行选举;
元数据管理:存放客户端需要的元数据信息,Dubbo、Kafka 等中间件都有用到;
高可用:利用分布式锁实现高可用,多个节点往 ZK 上注册,注册成功后成为 active,没有注册成功的
点阻塞;
分布式锁:可以搞,但高并发下性能差,建议用 Redis;

<h2 id="基础知识">基础知识</h2>
<h2 id="数据模型">数据模型</h2>
<h3 id="树形结构">树形结构</h3>
<p><img src="https://ld246.com/images/img-loading.svg" alt="ZooKeeper 数据结构" data-sr
="https://b3logfile.com/file/2020/06/solofetchupload3011769259206555708-323390d1.png?
mageView2/2/interlace/1/format/jpg"></p>
<h3 id="节点类型与特性">节点类型与特性</h3>

持久节点
这种节点也是在 ZooKeeper 最为常用的，几乎所有业务场景中都会包含持久节点的创建。之所以叫
持久节点是因为一旦将节点创建为持久节点，该数据节点会一直存储在 ZooKeeper 服务器上，即使
建该节点的客户端与服务端的会话关闭了，该节点依然不会被删除。如果我们想删除持久节点，就要
式调用 delete 函数进行删除操作。
临时节点
从名称上我们可以看出该节点的一个最重要的特性就是临时性。所谓临时性是指，如果将节点创建为
时节点，那么该节点数据不会一直存储在 ZooKeeper 服务器上。当创建该临时节点的客户端会话因
时或发生异常而关闭时，该节点也相应在 ZooKeeper 服务器上被删除。同样，我们可以像删除持久
点一样主动删除临时节点。
有序节点
其实有序节点并不算是一种单独种类的节点，而是在之前提到的持久节点和临时节点特性的基础上，
加了一个节点有序的性质。所谓节点有序是说在我们创建有序节点的时候，ZooKeeper 服务器会自
使用一个单调递增的数字作为后缀，追加到我们创建节点的后边。例如一个客户端创建了一个路径为
orks/task- 的有序节点，那么 ZooKeeper 将会生成一个序号并追加到该节点的路径后，最后该节点
路径为 works/task-1。通过这种方式我们可以直观的查看到节点的创建顺序。

<p><img src="https://ld246.com/images/img-loading.svg" alt="节点状态结构解释" data-src=
https://b3logfile.com/file/2020/06/solofetchupload665270369991945746-aebf709e.png?ima
eView2/2/interlace/1/format/jpg"></p>
<h3 id="Watch机制">Watch 机制</h3>

客户端、服务端分别有 ZKWatchManager 个 WatchManager，用来存放对应的观察者列表</li

客户端工作内容

当发送一个带有 Watch 事件的请求，客户端首先将该会话标记为 Watch 事件，之后通过 Data
atchRegistration 类保存 Watch 事件和节点的对应关系
客户端将请求封装成一个 Packet 对象，将该对象添加到等待发送队列 outgoingQueue 中，最
将请求逐个发送给服务端
最后调用负责处理响应的 SendThread 线程类中的 readResponse 方法接收服务端的回调。最
调用 finishPacket 方法将 Watch 注册到 ZKWatchManager 中

服务端工作内容：

当 zookeeper 服务端收到请求时，会判断请求中是否包含 Watch 事件（底层通过 FinalRequest rocessor 类中的 processRequest 方法实现）
当 getDataRequest.getWatch 为 True 时，表明该请求需要进行 Watch 监控注册通过 zks.get KDatabase().getData 将 Watch 事件注册到服务端的 WatchManager 中

<h3 id="服务端Watch事件触发过程">服务端 Watch 事件触发过程</h3>

在 setData 方法中执行完对节点数据的变更后会调用 WatchManager.triggerWatch 方法触发据变更事件
triggerWatch 方法内容

首先封装了一个具有会话状态、事件类型、数据节点 3 种属性的 WatchedEvent 对象；
查询该节点注册的 Watch 事件，如果为空说明没有注册 Watch 事件，存在则将 Watch 事件添到 Watchers 集合中
将 WatchManager 中的 Watch 事件删除，最后通过 process 方法向客户端发送通知

客户端回调处理过程

SendThread.readResponse() 方法来统一处理服务端的响应
反序列化服务器发送请求头信息 replyHdr.deserialize(bbia, “header”)，并判断相属性字段 xi 的值为 -1，表示该请求响应为通知类型
在处理通知类型时，先将已收到的字节流反序列化为 WatcherEvent 对象
判断客户端是否配置了 chrootPath ，如果配置了 chrootPath 属性，需要对接收到的节点路径行 chrootPath 处理
调用 eventThread.queueEvent() 方法将收到的事件交给 EventThread 线程处理
ventThread.queueEvent() 方法

按照通知事件类型，会从 ZKWatchManager 中查询在客户端注册过的 Watch 事件信息，查询后将 Watch 信息从 ZKWatchManager 中删除
然后在获取到 Watch 事件信息之后，将查询到的 Watch 存储到 waitingEvents 队列中，调用 E entThread 类中的 run 方法循环取出 Watch 事件进行处理，最后调用 processEvent(event) 方法来终执行实现了 Watcher 接口的 process（）方法。

<h3 id="ACL机制">ACL 机制</h3>

授权方式

IP 方式：使用的授权对象可以是一个 IP 地址或 IP 地址段
Digest 或 Super 方式：对应于一个用户名
World 方式：授权系统中所有的用户

授权内容

数据节点（create）创建权限，授予权限的对象可以在数据节点下创建子节点；
数据节点（wirte）更新权限，授予权限的对象可以更新该数据节点；

数据节点（read）读取权限，授予权限的对象可以读取该节点的内容以及子节点的信息；
数据节点（delete）删除权限，授予权限的对象可以删除该数据节点的子节点；
数据节点（admin）管理者权限，授予权限的对象可以对该数据节点体进行 ACL 权限设置。

实现原理

客户端在 ACL 权限请求发送过程的步骤

封装该请求的类型
将权限信息封装到 request 中并发送给服务端

服务器的实现

分析请求类型是否是权限相关操作
根据不同的权限模式（scheme）调用不同的实现类验证权限最后存储权限信息

<h3 id="集群架构">集群架构</h3>
<p><img src="https://ld246.com/images/img-loading.svg" alt="ZooKeeper 集群架构" data-sr
="https://b3logfile.com/file/2020/06/solofetchupload1289658074468507709-c37f1003.png?
mageView2/2/interlace/1/format/jpg"></p>

leader(领导者) -- 为客户端提供读和写的功能,负责投票的发起和决议,只有 leader 才能接受写的
务;
follower(跟随者) -- 为客户端提供读和写的功能，负责投票的发起和决议;
observer(观察者) -- 为客户端提供读服务，如果是写服务就转发个 leader。不参与 leader 的选
投票。也不参与写的过半原则机制。在不影响写的前提下，提高集群读的性能，为 zookeeper3.3 系
新增的角色;
client：连接 zookeeper 集群的使用者，请求的发起者，独立于 zookeeper 集群的角色;

<h3 id="选举机制">选举机制</h3>

概念

Serverid：服务器 ID,编号越大在选择算法中的权重越大;
Zxid：数据 ID,服务器中存放的最大数据 ID,值越大说明数据越新，在选举算法中数据越新权重越
;
Epoch：逻辑时钟，或者叫投票的次数，同一轮投票过程中的逻辑时钟值是相同的。每投完一次
这个数据就会增加，然后与接收到的其它服务器返回的投票信息中的数值相比，根据不同的值做出不
的判断。
Server 状态：选举状态

LOOKING，竞选状态；
FOLLOWING，随从状态，同步 leader 状态，参与投票；
OBSERVING，观察状态,同步 leader 状态，不参与投票；
LEADING，领导者状态。

触发结点

集群启动
leader 挂掉
follower 挂掉后 leader 发现已经没有过半的，leader 发现怎么集群不能对外提供服务了，会将
己的状态改为挂掉，重新进行 leader 选举

选举流程

<p><img src="https://ld246.com/images/img-loading.svg" alt="ZooKeeper 选举流程" data-sr
="https://b3logfile.com/file/2020/06/solofetchupload2578265284769679494-f43c642c.png?i
ageView2/2/interlace/1/format/jpg"></p>

选举状态

<p><img src="https://ld246.com/images/img-loading.svg" alt="ZooKeeper 选举状态" data-sr
="https://b3logfile.com/file/2020/06/solofetchupload3882221959281500167-e9767e8e.png?
mageView2/2/interlace/1/format/jpg"></p>
<h3 id="过半机制">过半机制</h3>
<pre><code class="highlight-chroma"><span class="highlight
cl">public class QuorumMaj implements QuorumVerifier {
   private static fi
al Logger LOG = LoggerFactory.getLogger(QuorumMaj.class);
   int half;
   // n表示集群中z
Server的个数（准确的说是参与者的个数，参与者不包括观察者节点）
   public Quoru
Maj(int n){
   this.half = n/
;
   }
   // 验证是否符
过半机制
   public boolean
ontainsQuorum(Set<Long> set){
   // half是在
造方法里赋值的
   // set.size()
示某台zkServer获得的票数
   return (set.si
e() > half);
   }
}
</code></pre>
<h3 id="奇数台">奇数台</h3>

既然要大于 n/2，那么当 n 为奇数的时候，必然有一边是大于 n/2 的，这样就能选举出 Leader

减少没必要的机器

<h3 id="读写流程">读写流程</h3>

写数据流程

当 Client 向 ZooKeeper 写入数据时，判断节点是不是 Leader，是直接写入；不是转发给 Leade

Leader 收到写请求时，不会直接将数据写入到 ZooKeeper 中，而是将数据写入到事务日志中（
似 Mysql 的 binlog）。
当 Leader 将事务日志写完后，会将写请求发送给所有节点（包括自己），收到请求后各个节点
始写自己的事务日志，日志写完后，会给 Leader 回复一个 ACK，表示写日志完成。
当 Leader 收到集群中半数以上的 Follower 回复 ACK 后，Leader 发送一个 commit 消息。</li

各个节点收到 commit 消息就会把内容放到内存中（保证数据可见）

<p><img src="https://ld246.com/images/img-loading.svg" alt="ZooKeeper 写数据流程" data
src="https://b3logfile.com/file/2020/06/solofetchupload3948658523186325394-107f8096.pn
?imageView2/2/interlace/1/format/jpg"></p>

读数据流程

读请求属于非事务性请求。无论在 leader、follower 还是 observer 上都可以直接读取。非事务
求还有 exist

<h3 id="ZAB协议">ZAB 协议</h3>

将数据都复制到 Follwer 中

<p><img src="https://ld246.com/images/img-loading.svg" alt="复制数据到 Follower" data-sr
="https://b3logfile.com/file/2020/06/solofetchupload161510471868258031-59271c95.png?i
ageView2/2/interlace/1/format/jpg"></p>

等待 Follwer 回应 Ack，最低超过半数即成功

<p><img src="https://ld246.com/images/img-loading.svg" alt="等待 Follower 回应" data-src
"https://b3logfile.com/file/2020/06/solofetchupload3088789680115811696-db308283.png?i
ageView2/2/interlace/1/format/jpg"></p>

当超过半数成功回应，则执行 commit ，同时提交自己

<p><img src="https://ld246.com/images/img-loading.svg" alt="执行 commit" data-src="htt
s://b3logfile.com/file/2020/06/solofetchupload807802129039252441-43b9b8e3.png?imageVi
w2/2/interlace/1/format/jpg"></p>