Zookeeper

——Learning Report

ACADEMIC YEAR: 2019-2020

SEMESTER 1

张磊

UCAS 中国科学院大学

(zhanglei171@mails.ucas.ac.cn)

目录

0. 声明

大家好，我于 2019 年秋季参加中国科学院大学“面向的对象程序设计”课程，并选择 ZooKeeper 作为源码学习环节的目标项目，本篇文章主要分为三个部分：功能分析与建模、核心流程设计分析、高级设计意图分析。

本次源码分析使用的版本号为 ZooKeeper 3.4.13，主要关注第二部分的 Leader 选举的核心流程分析，可能会弱化第一部分功能分析和第三部分高级设计意图分析的部分内容，文中摘录了许多我认为总结的比较好的对 ZooKeeper 中重要概念的介绍，感兴趣的读者可以自行查阅，参考文献会在每一部分最后的小结部分中给出，如果是网上的文章会给出相应的网址，项目源码地址：

https://github.com/apache/zookeeper

由于本人初学 java，对 Zookeeper 难免有理解错误的地方，欢迎大家批评指正，共同进步。

张磊 2019 年 11 月 4 日 星期一

www.zhanglei171@mails.ucas.ac.cn

第一部分

重要功能和概念简介

一. ZooKeeper 的诞生

从计算机诞生之后的几十年里，大多数我们平时使用的应用服务都是在一个单独的单处理器计算机上运行的程序。但是今天，应用服务已经发生了很大的变化。今天的应用服务都是由很多个独立的程序组成，并且这些独立的程序分别运行在不同的计算机上。

开发一个程序时，要想协同一个应用中的多个程序工作是一件非常棘手的事。开发这样的应用，很容易让很多开发人员陷入如何使多个程序协同工作的逻辑中，最后导致没有时间更好地思考和实现他们自己的应用程序逻辑；又或者开发人员对协同逻辑关注不够，只是用很少的时间开发了一个简单脆弱的主协调器，导致不可靠的单一失效点。[1]

所以，ZooKeeper 诞生了。

ZooKeeper 的设计目标是将那些复杂且容易出错的分布式一致性服务封装起来，构成一个高效可靠的原语集，并以一系列简单易用的接口提供给用户使用。这样便使得应用开发人员可以更多关注应用本身的逻辑，而不是协同工作。

简单地说，ZooKeeper 的功能就是：**在分布式系统中协作多个任务。**

[1] ZooKeeper 分布式过程协同技术讲解 机械工业出版社

二. ZooKeeper 的特点

ZooKeeper 主要有以下 4 个特点：

- 顺序一致性：从同一客户端发起的事务请求，最终将会严格地按照顺序被应用到 ZooKeeper 中去。
- 原子性：所有事务请求的处理结果在整个集群中所有机器上的应用情况是一致的，也就是说，要么整个集群中所有的机器都成功应用了某一个事务，要么都没有应用。
- 单一系统映像 ：无论客户端连到哪一个 ZooKeeper 服务器上，其看到的服务端数据模型都是一致的。
- 可靠性：一旦一次更改请求被应用，更改的结果就会被持久化，直到被下一次更改覆盖。

三. ZooKeeper 的重要概念

3.1 会话（session）

Session 指的是 ZooKeeper 服务器与客户端会话。在 ZooKeeper 中，一个客户端连接是指客户端和服务器之间的一个 TCP 长连接。客户端启动的时候，首先会与服务器建立一个 TCP 连接，从第一次连接建立开始，客户端会话的生命周期也开始了。通过这个连接,客户端能够通过心跳检测与服务器保持有效的会话,也能够向 Zookeeper 服务器发送请求并接受响应，同时还能够通过该连接接收来自服务器的 Watch 事件通知。Session 的 sessionTimeout 值用来设置一个客户端会话的超时时间。当由于服务器压力太大、网络故障或是客户端主动断开连接等各种原因导致客户端连接断开时，只要在 sessionTimeout 规定的时间内能够重新连接上集群中任意一台服务器，那么之前创建的会话仍然有效。

在为客户端创建会话之前，服务端首先会为每个客户端都分配一个 sessionID。由于 sessionID 是 Zookeeper 会话的一个重要标识,许多与会话相关的运行机制都是基于这个 sessionID 的,因此,无论是哪台服务器为客户端分配的 sessionID，都务必保证全局唯一。

3.2 Znode

在谈到分布式的时候，我们通常说的“节点"是指组成集群的每一台机器。然而，在 Zookeeper 中，“节点"分为两类，第一类同样

是指构成集群的机器，我们称之为机器节点；第二类则是指数据模型中的数据单元，我们称之为数据节点——Znode。

Zookeeper 将所有数据存储在内存中，数据模型是一棵树（Znode Tree)，其中由斜杠（/）进行分割的路径，就是一个 Znode，例如/foo/path1。每个上都会保存自己的数据内容，同时还会保存一系列属性信息。

在 Zookeeper 中，node 可以分为持久节点和临时节点两类。所谓持久节点是指一旦这个 Znode 被创建了,除非主动进行 Znode 的移除操作，否则这个 Znode 将一直保存在 Zookeeper 上。而临时节点就不一样了，它的生命周期和客户端会话绑定，一旦客户端会话失效，那么这个客户端创建的所有临时节点都会被移除。另外，ZooKeeper 还允许用户为每个节点添加一个特殊的属性：SEQUENTIAL.一旦节点被标记上这个属性，那么在这个节点被创建的时候,Zookeeper 会自动在其节点名后面追加上一个整型数字，这个整型数字是一个由父节点维护的自增数字。

3.3 版本

在前面我们已经提到，ZooKeeper 的每个 Znode 上都会存储数据，对应于每个 Znode，ZooKeeper 都会为其维护一个叫做 Stat 的数据结构，Stat 中记录了这个 Znode 的三个数据版本，分别是 version（当前 Znode 的版本）、cversion（当前 Znode 子节点的版本）和 cversion（当前 Znode 的 ACL 版本）。

3.4 Watcher

Watcher（事件监听器），是 ZooKeeper 中的一个很重要的特性。ZooKeeper 允许用户在指定节点上注册一些 Watcher，并且在一些特定事件触发的时候，ZooKeeper 服务端会将事件通知到感兴趣的客户端上去，该机制是 ZooKeeper 实现分布式协调服务的重要特性。

3.5 ACL

ZooKeeper 采用 ACL（AccessControlLists）策略来进行权限控制，类似于 UNIX 文件系统的权限控制。ZooKeeper 定义了如下 5 种权限。

- **CREATE** ： 创建子节点的权限；
- **READ** ： 获取节点数据和子节点列表的权限；
- **WRITE** ： 更新节点数据的权限；
- **DELETE** ： 删除子节点的权限；
- **ADMIN** ： 设置节点 ACL 的权限；

其中需要特别注意的是，CREATE 和 DELETE 这两种权限都是针对子节点的权限控制。

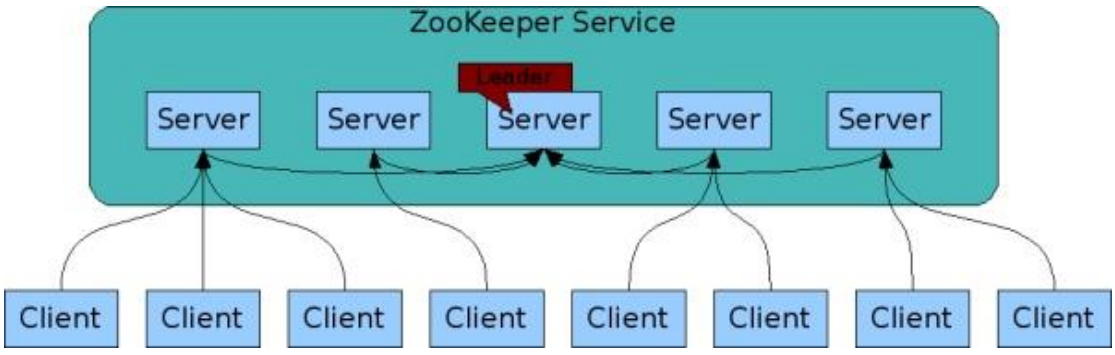
四. ZooKeeper 的设计目标

4.1 简单的数据模型

ZooKeeper 允许分布式进程通过共享的层次结构命名空间进行相互协调，这与标准文件系统类似。名称空间由 ZooKeeper 中的数据寄存器组成——称为 znode，这些类似于文件和目录。与为存储设计的典型文件系统不同，ZooKeeper 数据保存在内存中，这意味着 ZooKeeper 可以实现高吞吐量和低延迟。

4.2 可构建集群

为了保证高可用，最好是以集群形态来部署 ZooKeeper，这样只要集群中大部分机器是可用的（能够容忍一定的机器故障），那么 ZooKeeper 本身仍然是可用的。客户端在使用 ZooKeeper 时，需要知道集群机器列表，通过与集群中的某一台机器建立 TCP 连接来使用服务，客户端使用这个 TCP 链接来发送请求、获取结果、获取监听事件以及发送心跳包。如果这个连接异常断开了，客户端可以连接到另外的机器上。



上图中每一个 Server 代表一个安装 Zookeeper 服务的服务器。组成 ZooKeeper 服务的服务器都会在内存中维护当前的服务器状

态，并且每台服务器之间都互相保持着通信。集群间通过 Zab 协议（Zookeeper Atomic Broadcast）来保持数据的一致性。

4.3 顺序访问

对于来自客户端的每个更新请求，ZooKeeper 都会分配一个全局唯一的递增编号，这个编号反应了所有事务操作的先后顺序，应用程序可以使用 ZooKeeper 这个特性来实现更高层次的同步原语。这个编号也叫做时间戳——zxid（Zookeeper Transaction Id）

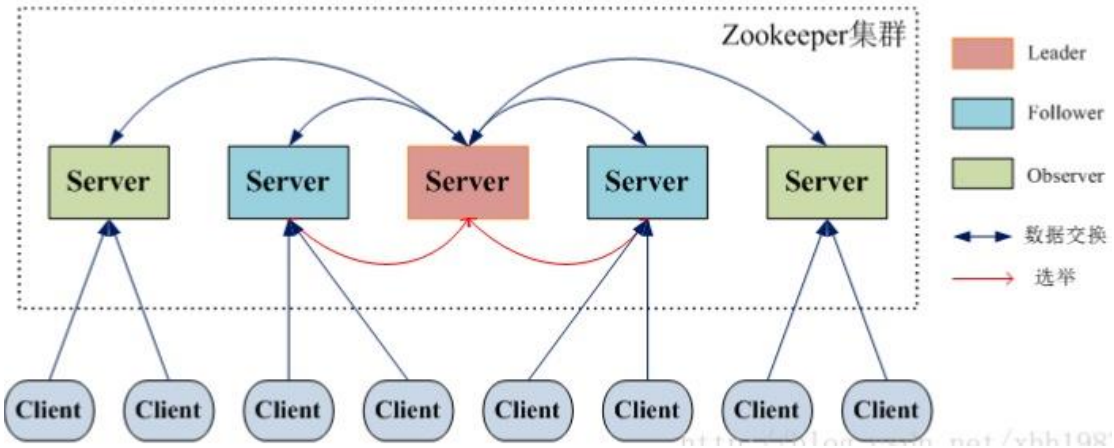
4.4 高性能

ZooKeeper 是高性能的。在“读”多于“写”的应用程序中尤其地高性能，因为“写”会导致所有的服务器间同步状态。(“读”多于“写”是协调服务的典型场景。)

五. ZooKeeper 的集群角色

最典型集群模式： Master/Slave 模式（主备模式）。在这种模式中，通常 Master 服务器作为主服务器提供写服务，其他的 Slave 服务器从服务器通过异步复制的方式获取 Master 服务器最新的数据提供读服务。

但是，在 ZooKeeper 中没有选择传统的 Master/Slave 概念，而是引入了 Leader、Follower 和 Observer 三种角色。



ZooKeeper 集群中的所有机器通过一个 Leader 选举过程来选定一台称为 “Leader” 的机器，Leader 既可以为客户端提供写服务又能提供读服务。除了 Leader 外，Follower 和 Observer 都只能提供读服务。Follower 和 Observer 唯一的区别在于 Observer 机器不参与 Leader 的选举过程，也不参与写操作的“过半写成功”策略，因此 Observer 机器可以在不影响写性能的情况下提升集群的读性能。

角色		主要工作描述
领导者		1. 事务请求的唯一调度和处理者，保证集群事务处理的顺序性； 2. 集群内部各服务器的调度者
学习者（Learner）	跟随者（Follower）	1. 处理客户端非事务请求，转发事务请求给Leader服务器 2. 参与事务请求Proposal的投票 3. 参与Leader选举的投票
	观察者（Observer）	Follower 和 Observer 唯一的区别在于 Observer 机器不参与 Leader 的选举过程，也不参与写操作的"过半写成功"策略，因此 Observer 机器可以在不影响写性能的情况下提升集群的读性能。
客户端（Client）		请求发起方

当 Leader 服务器出现网络中断、崩溃退出与重启等异常情况时，ZAB 协议就会进人恢复模式并选举产生新的 Leader 服务器。这个过程大致是这样的：

1. **Leader election（选举阶段）**：节点在一开始都处于选举阶段，只要有一个节点得到超半数节点的票数，它就可以当选准 leader。
2. **Discovery（发现阶段）**：在这个阶段，followers 跟准 leader 进行通信，同步 followers 最近接收的事务提议。
3. **Synchronization（同步阶段）**:同步阶段主要是利用 leader 前一阶段获得的最新提议历史，同步集群中所有的副本。同步完成之后 准 leader 才会成为真正的 leader。
4. **Broadcast（广播阶段）**：到了这个阶段，Zookeeper 集群才能正式对外提供事务服务，并且 leader 可以进行消息广播。同时如果有新的节点加入，还需要对新节点进行同步。

六. ZooKeeper & ZAB 协议 & Paxos 算法

6.1 ZAB 协议 & Paxos 算法

ZooKeeper 并没有完全采用 Paxos 算法 ，而是使用 ZAB 协议作为其保证数据一致性的核心算法。另外，在 ZooKeeper 的官方文档中也指出，ZAB 协议并不像 Paxos 算法那样，是一种通用的分布式一致性算法，它是一种特别为 Zookeeper 设计的崩溃可恢复的原子消息广播算法。

6.2 ZAB 协议

ZAB（ZooKeeper Atomic Broadcast 原子广播） 协议是为分布式协调服务 ZooKeeper 专门设计的一种支持崩溃恢复的原子广播协议。 在 ZooKeeper 中，主要依赖 ZAB 协议来实现分布式数据一致性，基于该协议，ZooKeeper 实现了一种主备模式的系统架构来保持集群中各个副本之间的数据一致性。

6.3 崩溃恢复和消息广播

ZAB 协议包括两种基本的模式,分别是 **崩溃恢复和消息广播**。当整个服务框架在启动过程中，或是当 Leader 服务器出现网络中断、崩溃退出与重启等异常情况时，ZAB 协议就会进入恢复模式并选举产生新的 Leader 服务器。

当选举产生了新的 Leader 服务器，同时集群中已经有过半的机器与该 Leader 服务器完成了状态同步之后，ZAB 协议就会退出

恢复模式。其中，**所谓的状态同步是指数据同步，用来保证集群中存在过半的机器能够和 Leader 服务器的数据状态保持一致。**

当集群中已经有过半的 Follower 服务器完成了和 Leader 服务器的状态同步，那么整个服务框架就可以进人消息广播模式了。当一台同样遵守 ZAB 协议的服务器启动后加入到集群中时，如果此时集群中已经存在一个 Leader 服务器在负责进行消息广播，那么新加入的服务器就会自觉地进人数据恢复模式：找到 Leader 所在的服务器，并与其进行数据同步，然后一起参与到消息广播流程中去。

如上所述，ZooKeeper 设计成只允许唯一的一个 Leader 服务器来进行事务请求的处理。Leader 服务器在接收到客户端的事务请求后，会生成对应的事务提案并发起一轮广播协议；而如果集群中的其他机器接收到客户端的事务请求，那么这些非 Leader 服务器会首先将这个事务请求转发给 Leader 服务器。

七. 小结

通过这段时间的阅读了解，我基本了解了 ZooKeeper 的由来、功能作用、特点、重要的概念、设计目标、集群角色以及 ZAB 协议和 Paxos 算法总算有了初步的了解。

但是由于没有亲自使用过 ZooKeeper，所以我对 ZooKeeper 的具体功能还是处于一种懵懂的状态，而且本部分摘取了很多前人的文章，自己的东西非常少，在之后的学习过程中我会尝试亲自使用 ZooKeeper，然后在使用中总结出自己的东西。

参考文献：

1. ZooKeeper：分布式过程协同技术详解 ZooKeeper：Distributed Process Coordination [美] 荣凯拉（Junqueira, F.）[美] 里德（Reed， B.）著
2. https://github.com/Snailclimb/JavaGuide/blob/master/docs/system-design/framework/ZooKeeper.md

第二部分

功能分析与需求建模

(Leader Election)

一. ZooKeeper 简单示例

ZooKeeper 最基本的一个功能就是数据同步。当 ZooKeeper 处于集群模式的时候（也就是有多个服务器的时候），此时假设我们有两个服务器 A 和 B，有两个客户端 C1 和 C2，C1 与 A 连接，C2 与 B 连接。

此时，客户端 C1 向服务器 A 发送数据，比如要创建一个/test 结点，并写入数据 666，写入成功后，客户端 C2 访问/test 结点的数据，那么此时，服务器 B 应该返回客户端 C1 写入的数据 666。

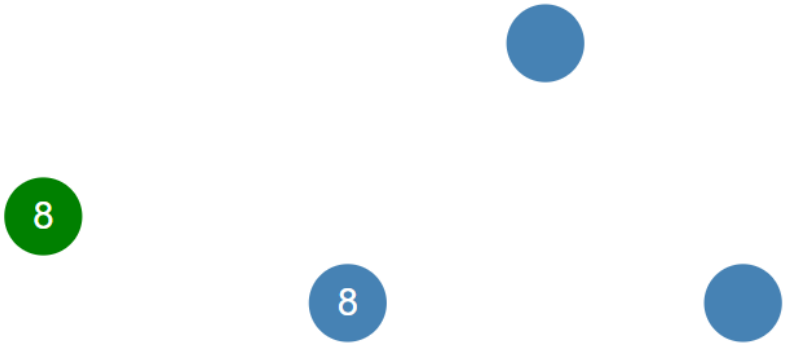
基于 watcher 的功能，ZooKeeper 还可以实现发布和订阅功能。

二. 数据同步

数据的同步在只有一个服务器的时候是非常简单地：

绿色客户端发送信息数据给服务器，不需要同步：

但是在集群模式下，数据同步就变得不那么容易了：



那么，如何同步多个服务器之间的数据呢？这就是我们这一部分需要讨论的重点了。

三. 人类的选举

在讨论这个问题之前我们需要先来研究一下选举，在后面我们会看到服务器的选举其实和人类的选举是十分类似的。

1. 人类选举的基本原理

小到选班长、组长，大到选地方领导，我们应该都经历过投票，如果你是一个非常具有正义感的人,那么你在投票时一定会把票投给自己认为能力比较强的人。如果你已经选好了一个人，但是又突然出现了能力更强的人，那么你可能会改选这个更强的人。选票将会放在投票中,最后从投票箱中进行统计,获得票数最多的人当选。

在这样一个选举过程中我们提炼出四个基本概念：

1. **个人能力：**投我认为能力最强的人，这是投票的基本规则；
2. **改票：**能力最强的人是逐渐和其他人沟通之后的结果，类似改票，先投给 A，但是后来发现 B 更厉害，则改为投 B；
3. **投票箱：**所有人公用一个投票箱；
4. **领导者：**获得投票数最多的人为领导者；

有了这个概念，我们就可以来看看 ZooKeeper 的选举了。

四. ZooKeeper 的选举

首先 ZooKeeper 只有在集群模式下才需要进行选举。

在选举中 Zookeeper 需要实现上面人类选举的四个基本概念：

1. **个人能力：**我们说 Zookeeper 可以看做是一个数据库，那么集群中节点的数据越新就代表此节点能力越强，而在 Zookeeper 中可以通过事务 id(zxid)来表示数据的新旧，一个节点的 zxid 越大则该节点的数据越新。所以 Zookeeper 选举时会根据 zxid 的大小来作为投票的基本规则。

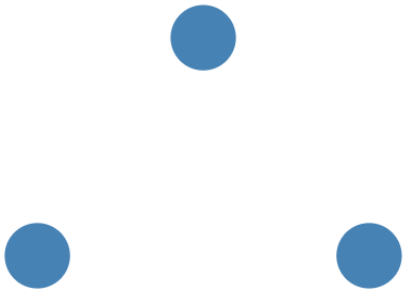
2. **改票：**Zookeeper 集群中的某一个节点在开始进行选举时，首先认为自己的数据是最新的，会先投自己一票，并且把这张选票发送给其他服务器，这张选票里包含了两个重要信息：zxid 和 sid，sid 表示这张选票投的服务器 id，zxid 表示这张选票投的服务器上最大的事务 id，同时也会接收到其他服务器的选票。
   接收到其他服务器的选票后，可以根据选票信息中的 zxid 来与自己当前所投的服务器上的最大 zxid 来进行比较，如果其他服务器的选票中的 zxid 较大，则表示自己当前所投的机器数据没有接收到的选票所投的服务器上的数据新，所以本节点需要改票，改成投给和刚刚接收到的选票一样。

3. **投票箱：**Zookeeper 集群中会有很多节点，和人类选举不一样，Zookeeper 集群并不会单独去维护一个投票箱应用，而是在每个节点内存里利用一个数组来作为投票箱。每个节点里都有一个投票箱，节点会将自己的选票以及从其他服务器接收到的选票放在这个投票箱中。因为集群节点是相互交互的，并且选票的 PK 规则是一致的，所以每个节点里的这个投票箱所存储的选票都会是一样的，这样也可以达到公用一个投票箱的目的。

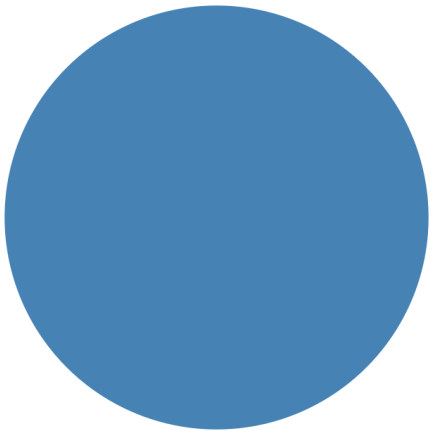
4. **领导者：**Zookeeper 集群中的每个节点，开始进行领导选举后，会不断的接收其他节点的选票，然后进行选票 PK，将自己的选票修改为投给数据最新的节点，这样就保证了，每个节点自己的选票代表的都是自己暂时所认为的数据最新的节点，再因为其他服务器的选票都会存储在投票箱内，所以可以根据投票箱里去统计是否有超过一半的选票和自己选择的是同一个节点，都认为这个节点的数据最新，一旦整个集群里超过一半的节点都认为某一个节点上的数据最新，则该节点就是领导者。

五. 具体选举过程说明

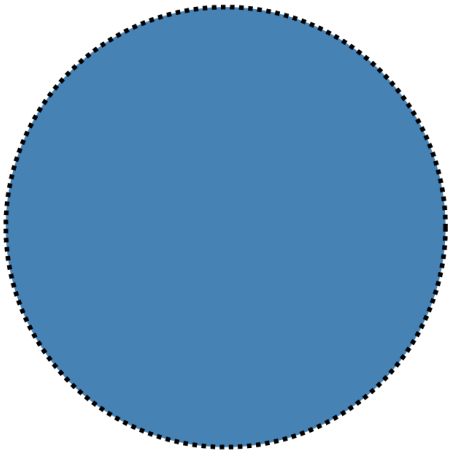
我们来看 3 台服务器的情况：



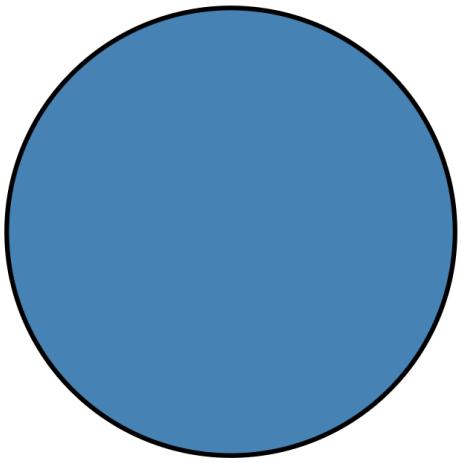
在 raft 中每台服务器存在三种可能的状态：



The *Follower* state,

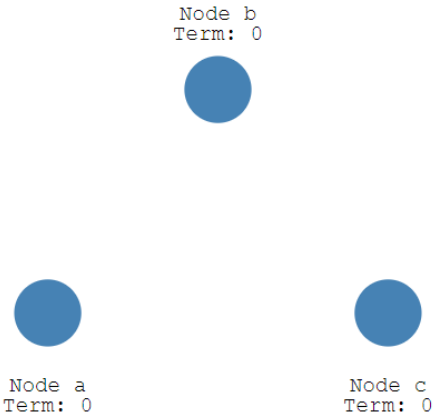


the *Candidate* state,

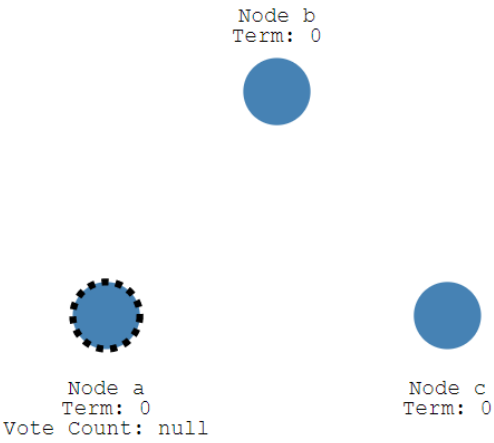


or the *Leader* state.

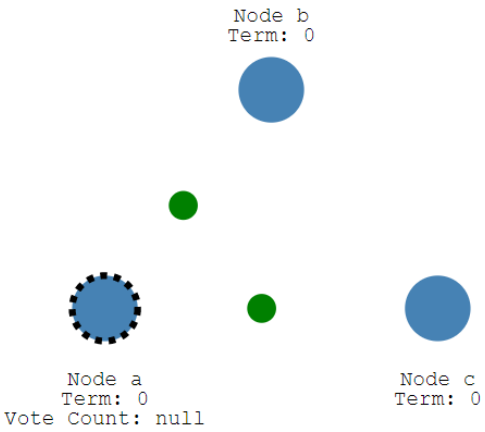
下面我们把刚才 3 个结点服务器命名如下：



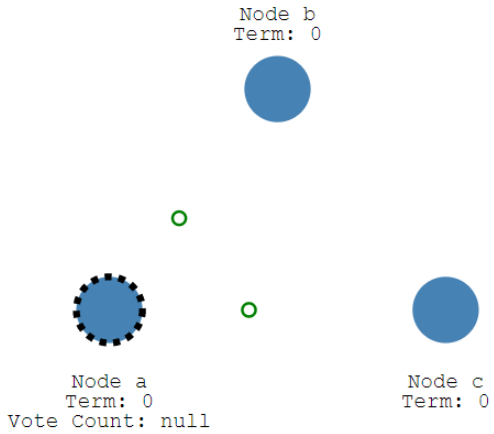
如果集群中没有 leader，那么 follower 就会变成 candidate 状态，



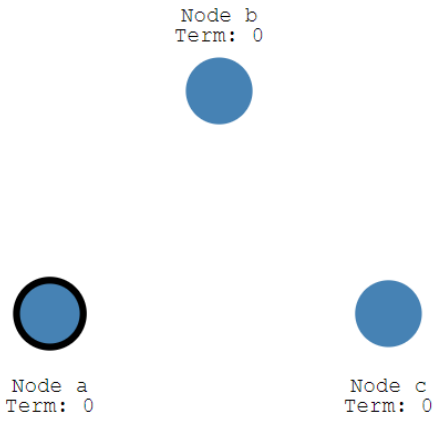
Candidate 会发送一个请求询问其他成员的投票：



其他成员也会回复 Candidate 的询问，



然后按照少数服从多数的原则，选出 Leader，



这个过程就是我们要讨论的重点：Leader Election。

六. 小结

在这一部分中，我们通过 ZooKeeper 的一个最基本的数据同步功能引出了 Leader 选举，然后通过与人类社会的选举类比，将 ZooKeeper 的 Leader 选举给抽象了出来，大致理解了整个 Leader 选举的过程。

但是在服务器运行的过程当中可能会出现一些其他的情况，导致 Leader 的变更，此时又要重新进行选举，但是大致的流程都是一样的，我就不再细讲了。有需要的同学可以通过下方的参考文献中的链接去看下一下选举的过程，以及出现问题了应该怎么办。

下一部分，我们将会深入到函数和类当中，看一下，ZooKeeper 是如何运用面向对象的方法，完成 Leader 选举的。

参考文献

1. http://thesecretlivesofdata.com/raft/

第三部分

核心流程设计分析

未完待续