1.介绍

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一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

来自〈https://juejin.im/post/5baf7db75188255c3d11622e?utm_source=gold_browser_extension#heading-18〉

2.ZooKeeper的设计目标

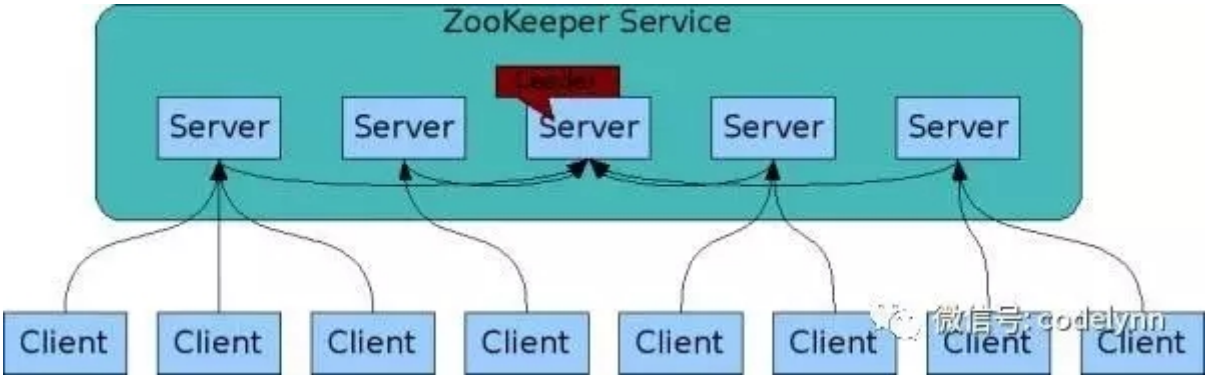
- 简单的数据结构：Zookeeper 使得分布式程序能够通过一个共享的树形结构的名字空间来进行相互协调，即Zookeeper 服务器内存中的数据模型由一系列被称为ZNode的数据节点组成，Zookeeper 将全量的数据存储在内存中，以此来提高服务器吞吐、减少延迟的目的。

- 可以构建集群：Zookeeper 集群通常由一组机器构成，组成 Zookeeper 集群的而每台机器都会在内存中维护当前服务器状态，并且每台机器之间都相互通信。
- 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，这个编号反映了所有事务操作的先后顺序。
- 高性能：Zookeeper 和Redis一样全量数据存储在内存中，100%读请求压测 QPS 12-13W

3.关于 ZooKeeper 的一些重要概念

3.1 Zookeeper 集群:

Zookeeper 是一个由多个 server 组成的集群,一个 leader，多个 follower。（这个不同于我们常见的Master/Slave模式）leader 为客户端服务器提供读写服务，除了leader外其他的机器只能提供读服务。 每个 server 保存一份数据副本全数据一致，分布式读 follower，写由 leader 实施更新请求转发，由 leader 实施更新请求顺序进行，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。全局唯一数据视图，client 无论连接到哪个 server，数据视图都是一致的实时性，在一定事件范围内，client 能读到最新数据。



3.2 集群角色

①.Leader：是整个 Zookeeper 集群工作机制中的核心 。Leader 作为整个 ZooKeeper 集群的主节点，负责响应所有对 ZooKeeper 状态变更的请求。 主要工作：

- 事务请求的唯一调度和处理，保障集群处理事务的顺序性。
- 集群内各服务器的调度者。

Leader 选举是 Zookeeper 最重要的技术之一，也是保障分布式数据一致性的关键所在,选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪 来自<https://blog.lqdev.cn/2018/09/09/SpringCloud/chapter-three/> 。我们以三台机器为例，在服务器集群初始化阶段，当有一台服务器Server1启动时候是无法完成选举的，当第二台机器

Server2 启动后两台机器能互相通信，每台机器都试图找到一个leader，于是便进入了leader 选举流程.

1. 每个 server 发出一个投票 投票的最基本元素是（SID-服务器id,ZXID-事物id）
2. 接受来自各个服务器的投票
3. 处理投票 优先检查 ZXID(数据越新ZXID越大),ZXID比较大的作为leader，ZXID一样的情况下比较SID
4. 统计投票 这里有个**过半**的概念，大于集群机器数量的一半，即大于或等于（n/2+1）,我们这里的由三台，大于等于2即为达到“过半”的要求。 这里也有引申到为什么 Zookeeper 集群推荐是单数。

集群数量	至少正常运行数量	允许挂掉的数量
2	2的半数为1，半数以上最少为2	0
3	3的半数为1.5，半数以上最少为2	1
4	4的半数为2，半数以上最少为3	1
5	5的半数为2.5，半数以上最少为3	2
6	6的半数为3，半数以上最少为4	2

通过以上可以发现，3台服务器和4台服务器都最多允许1台服务器挂掉，5台服务器和6台服务器都最多允许2台服务器挂掉,明显4台服务器成本高于3台服务器成本，6台服务器成本高于5服务器成本。这是由于半数以上投票通过决定的。

5.改变服务器状态 一旦确定了 leader，服务器就会更改自己的状态，且一般不会再发生变化，比如新机器加入集群、非 leader 挂掉一台。

②.Follower ：是 Zookeeper 集群状态的跟随者。他的逻辑就比较简单。除了响应本服务器上的读请求外，follower 还要处理leader 的提议，并在 leader 提交该提议时在本地也进行提交。另外需要注意的是，leader 和 follower 构成ZooKeeper 集群的法定人数，也就是说，只有他们才参与新 leader的选举、响应 leader 的提议。

③.Observer ：服务器充当一个观察者的角色。如果 ZooKeeper 集群的读取负载很高，或者客户端多到跨机房，可以设置一些 observer 服务器，以提高读取的吞吐量。Observer 和 Follower 比较相似，只有一些小区别：首先 observer 不属于法定人数，即不参加选举也不响应提议，也不参与写操作的“过半写成功”策略；其次是 observer 不需要将事务持久化到磁盘，一旦 observer 被重启，需要从 leader 重新同步整个名字空间。

④.Learner：Observer和Follower统称为学习者,因为同步数据时他们都会接受leader的同步

3.3.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。

3.3.1 会话（Session)

在Zookeeper客户端与服务端成功完成连接创建后，就创建了一个会话，Zookeeper会话在整个运行期间的生命周期中，会在不同的会话状态中之间进行切换，这些状态可以分为CONNECTING、CONNECTED、RECONNECTING、RECONNECTED、CLOSE等。

一旦客户端开始创建Zookeeper对象，那么客户端状态就会变成CONNECTING状态，同时客户端开始尝试连接服务端，

连接成功后，客户端状态变为CONNECTED，

通常情况下，由于断网或其他原因，客户端与服务端之间会出现断开情况，这时，Zookeeper客户端会自动进行重连服务，同时客户端状态再次变成CONNCTING，

直到重新连上服务端后，状态又变为CONNECTED，

在通常情况下，客户端的状态总是介于CONNECTING 和CONNECTED 之间。

但是，如果出现诸如会话超时、权限检查或是客户端主动退出程序等情况，客户端的状态就会直接变更为CLOSE状态。

3.3.2 会话创建

Session是Zookeeper中的会话实体，代表了一个客户端会话，其包含了如下四个属性

1. sessionID。会话ID，唯一标识一个会话，每次客户端创建新的会话时，Zookeeper都会为其分配一个全局唯一的sessionID。

2. TimeOut。会话超时时间，客户端在构造Zookeeper实例时，会配置sessionTimeout参数用于指定会话的超时时间，Zookeeper客户端向服务端发送这个超时时间后，服务端会根据自己的超时时间限制最终确定会话的超时时间。

3. TickTime。下次会话超时时间点，为了便于Zookeeper对会话实行”分桶策略”管理，同时为了高效低耗地实现会话的超时检查与清理，Zookeeper会为每个会话标记一个下次会话超时时间点，其值大致等于当前时间加上TimeOut。

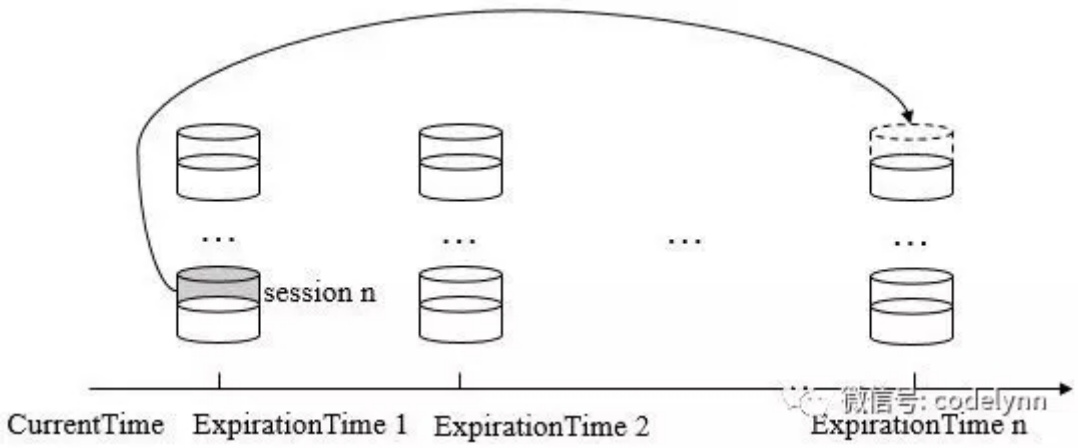
4. isClosing。标记一个会话是否已经被关闭，当服务端检测到会话已经超时失效时，会将该会话的isClosing标记为”已关闭”，这样就能确保不再处理来自该会话的心情求了。

Zookeeper为了保证请求会话的全局唯一性，在SessionTracker初始化时，调用initializeNextSession方法生成一个sessionID，之后在Zookeeper运行过程中，会在该sessionID的基础上为每个会话进行分配，初始化算法如下

```
public static long initializeNextSession(long id) {
    long nextSid = 0;
    // 无符号右移8位是为了避免左移24后，再右移8位出现负数而无法通过高8位确定sid值
    nextSid = (System.currentTimeMillis() << 24) >>> 8;
    nextSid = nextSid | (id << 56);
    return nextSid;
  }
```

3.3.3 会话管理

Zookeeper的会话管理主要是通过SessionTracker来负责，其采用了分桶策略（将类似的会话放在同一区块中进行管理）进行管理，以便Zookeeper对会话进行不同区块的隔离处理以及同一区块的统一处理。

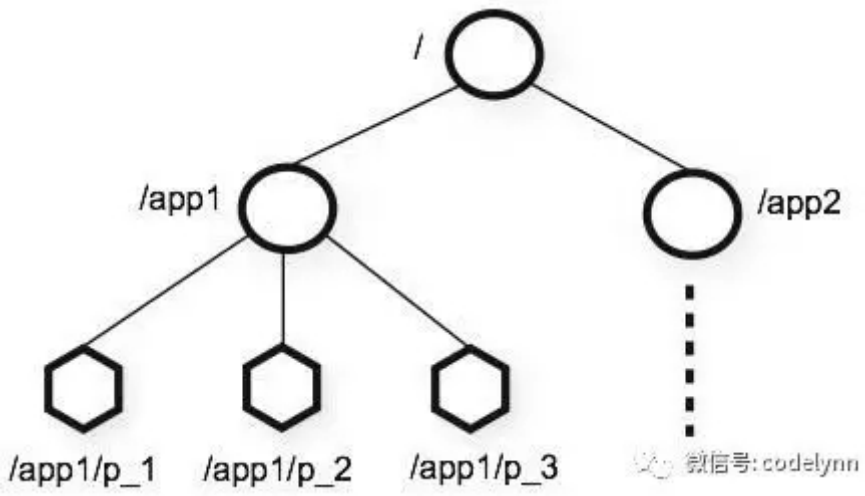


3.4 数据节点 Znode

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），由斜杠（/）的进行分割的路径，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。



3.4.1 节点类型

在Zookeeper中，node可以分为持久节点 和 临时节点 和 顺序节点 三大类。
可以通过组合生成如下四种类型节点

1. PERSISTENT 持久节点,节点创建后便一直存在于Zookeeper服务器上，直到有删除操作来主动清楚该节点。
2. PERSISTENT_SEQUENTIAL 持久顺序节点,相比持久节点，其新增了顺序特性，每个父节点都会为它的第一级子节点维护一份顺序，用于记录每个子节点创建的先后顺序。在创建节点时，会自动添加一个数字后缀，作为新的节点名，该数字后缀的上限是整形的最大值。
3. EPEMERAL 临时节点，临时节点的生命周期与客户端会话绑定，客户端失效，节点会被自动清理。同时，Zookeeper规定不能基于临时节点来创建子节点，即临时节点只能作为叶子节点。

3.5 版本——保证分布式数据原子性操作

每个数据节点都具有三种类型的版本信息，对数据节点的任何更新操作都会引起版本号的变化。

- version– 当前数据节点数据内容的版本号
- cversion– 当前数据子节点的版本号

- aversion– 当前数据节点ACL变更版本号

上述各版本号都是表示修改次数，如version为1表示对数据节点的内容变更了一次。即使前后两次变更并没有改变数据内容，version的值仍然会改变。version可以用于写入验证，类似于CAS。

3.6 watcher事件监听器

ZooKeeper允许用户在指定节点上注册一些Watcher，当数据节点发生变化的时候，ZooKeeper服务器会把这个变化的通知发送给感兴趣的客户端,客户端是先收到变更通知,后收到修改后的数据