Zookeeper Developer Guide

For Zookeeper3.4.6 version

Arranged by jiangzz

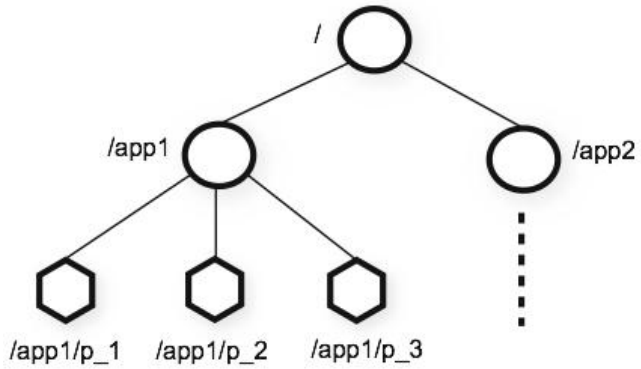
ZooKeeper 简介

ZooKeeper 是一个为分布式应用所设计的分布的、开源的协调服务。分布式的应用可以建立在同步配置管理、选举、分布式锁、分组和命名等服务的更高级别的实现的基础之上。 ZooKeeper 意欲设计一个易于编程的环境，它的文件系统使用我们所熟悉的目录树结构。ZooKeeper 使用 Java 所编写，但是支持 Java 和 C 两种编程语言。

协调服务非常容易出错，但是却很难恢复正常，例如，协调服务很容易处于竞争状态以至于出现死锁。设计 ZooKeeper 的目的是为了减轻分布式应用程序所承担的协调任务，借助于 zookeeper 服务开发出高可靠的分布式协同程序。

ZooKeeper 数据模型

提供的命名空间与标准的文件系统非常相似。一个名称是由通过斜线分隔开的路径名序列所组成的。 ZooKeeper 中的每一个节点是都通过路径来识别。



ZooKeeper 节点

ZooKeeper 的节点是通过像树一样的结构来进行维护的，并且每一个节点通过路径来标示以及访问。除此之外，每一个节点还拥有自身的一些信息，包括：数据、数据长度、创建时间、修改时间等等。从这样一类既含有数据，又作为路径表标示的节点的特点中，可以看出， ZooKeeper 的节点既可以被看做是一个文件，又可以被看做是一个目录，它同时具有二者的特点。为了便于表达，今后将使用Znode 来表示所讨论的 ZooKeeper 节点。

具体地说， Znode 维护着数据、 ACL（ access control list，访问控制列表）、时间戳等交换版本号等数据结构，它通过对这些数据的管理来让缓存生效并且令协调更新。每当 Znode 中的数据更新后它所维护的版本号将增加，这非常类似于数据库中计数器时间戳的操作方式。

另外 Znode 还具有原子性操作的特点：命名空间中，每一个 Znode 的数据将被原子地读写。读操作将读取与 Znode 相关的所有数据，写操作将替换掉所有的数据。除此之外，每一个节点都有一个访问控制列表，这个访问控制列表规定了用户操作的权限。ZooKeeper 中同样存在临时节点。这些节点与 session 同时存在，当 session生命周期结束，这些临时节点也将被删除。临时节点在某些场合也发挥着非常重要的作用。

ZooKeeper 节点是有生命周期的，这取决于节点的类型。在 ZooKeeper 中，节点类型可以分为持久节点（PERSISTENT ）、临时节点（EPHEMERAL），以及时序节点（SEQUENTIAL ），具体在节点创建过程中，一般是组合使用，可以生成以下 4 种节点类型。

持久节点（PERSISTENT）

持久节点，是指在节点创建后，就一直存在，直到有删除操作来主动清除这个节点——不会因为创建该节点的客户端会话失效而消失。

持久顺序节点（PERSISTENT_SEQUENTIAL）

这类节点的基本特性和上面的节点类型是一致的。额外的特性是，在 ZK 中，每个父节点会为他的第一级子节点维护一份时序，会记录每个子节点创建的先后顺序。基于这个特性，在创建子节点的时候，可以设置这个属性，那么在创建节点过程中，ZK 会自动为给定节点名加上一个数字后缀，作为新的节点名。这个数字后缀的范围是整型的最大值。

临时节点（EPHEMERAL）

和持久节点不同的是，临时节点的生命周期和客户端会话绑定。也就是说，如果客户端会话失效，那么这个节点就会自动被清除掉。注意，这里提到的是会话失效，而非连接断开。另外，在临时节点下面不能创建子节点。

临时顺序节点（EPHEMERAL_SEQUENTIAL）

分布式锁

第一步：客户端调用 create()方法创建“_locknode_/guid-lock-”节点，需要注意的是，这里节点的创建类型设置 EPHEMERAL_SEQUENTIAL。

第二步：客户端调用 getChildren(“_locknode_”)方法来获取所有已经创建的子节点，注意，这里不注册任何 Watcher。

第三步：客户端获取到所有子节点 path 之后，如果发现自己在步骤 1 中创建的节点序号最小，那么就认为这个客户端获得了锁。

如果在步骤 3 中发现自己并非所有子节点中最小的，说明自己还没有获取到锁。此时客户端需要找到比自己小的那个节点，然后对其调用 exist()方法，同时

注册事件监听。之后当这个被关注的节点被移除了，客户端会收到相应的通知。这个时候客户端需要再次调用 getChildren(“_locknode_”)方法来获取所有已经创建的子节点，确保自己确实是最小的节点了，然后进入步骤 3。

监测

客户端可以监测 znode 节点的变化。Znode 节点的变化触发相应的事件，然后清除对该节点的监测。当监测一个 znode 节点时候，Zookeeper 会发送通知给监测节点。

ZooKeeper 安装

ZooKeeper 的安装模式分为三种，分别为：单机模式（ stand-alone）、集群模式和集群伪分布模式。 ZooKeeper 单机模式的安装相对比较简单，如果第一次接触 ZooKeeper 建议安装 ZooKeeper 单机模式或者集群伪分布模式。

单机模式

- 从 Apache 官方网站下载一个 ZooKeeper 的最近稳定版本。
- ZooKeeper 要求 JAVA 的环境才能运行，并且需要 JAVA6 以上的版本，可以从 SUN 官网上下载，并对 JAVA 环境变量进行设置。除此之外，为了今后操作的方便，我们需要对 ZooKeeper 的环境变量进行配置。
- 在 Zookeeper 安装目录下 conf 目录添加 zoo.cfg 配置文件。

```
tickTime=2000
dataDir=/var/zookeeper
clientPort=2181
```

说明:

在这个文件中，我们需要指定 dataDir 的值，它指向了一个目录，这个目录在开始的时候需要为空。下面是每个参数的含义：

tickTime ：基本事件单元，以毫秒为单位。它用来指示心跳，最小的 session 过期时间为两倍的 tickTime。

dataDir ：存储内存中数据库快照的位置，如果不设置参数，更新事务日志将被存储到默认位置。

clientPort ：监听客户端连接的端口。

使用单机模式时用户需要注意：这种配置方式下没有 ZooKeeper 副本，所以如果 ZooKeeper 服务器出现故障， ZooKeeper 服务将会停止。

集群模式

为了获得可靠的 ZooKeeper 服务，用户应该在一个集群上部署 ZooKeeper 。只要集群上大多数的 ZooKeeper 服务启动了，那么总的 ZooKeeper 服务将是可用的。另外，最好使用奇数台机器。 如果 zookeeper 拥有 5 台机器，那么它就能处理 2 台机器的故障了。N-1/2

之后的操作和单机模式的安装类似，我们同样需要对 JAVA 环境进行设置，下载最新的 ZooKeeper 稳定版本并配置相应的环境变量。不同之处在于每台机器上 conf/zoo.cfg 配置文件的参数设置，参考下面的配置：

```
tickTime=2000
dataDir=/var/zookeeper/
clientPort=2181
initLimit=5
syncLimit=2
server.1=zoo1:2887:3887
server.2=zoo2:2888:3888
server.3=zoo3:2889:3889
```

“server.id=host:port:port”指出了不同的 Zookeeper 的服务器的自身的标示，作为集群中一部分机器应该知道集群中其他的机器。用户可以从”server.id=host:port:port”中读取相关信息。在 dataDir 目录创建一个文件名为 myid 的文件，这个文件仅仅含有一行内容，指定的是自身的 id 值。比如服务器 id 是 1 就在这个文件写 1.第一个 port 是保持和主机通信，第二个 port 是做选举的。启动 Zookeeper 的集群

```
./bin/zkServer.sh start zoo1.cfg
```

连接集群

```
./bin/zkCli.sh -server 127.0.0.1:2181
```

ZooKeeper 四字命令

ZooKeeper 支持某些特定的四字命令字母与其的交互。它们大多是查询命令，用来获取 ZooKeeper 服务的当前状态及相关信息。用户在客户端可以通过 telnet 或 nc 向 ZooKeeper 提交相应的命令。（备注 CentOS minal 版本不支持 nc 和 telnet 需要使用 yum 安装）。

ZooKeeper 四字命令	功能描述
conf	输出相关服务配置的详细信息。
cons	列出所有连接到服务器的客户端的完全的连接 / 会话的详细信息。包括“接受 / 发送”的包数量、会话 id 、操作延迟、最后的操作执行等等信息。
dump	列出未经处理的会话和临时节点。
envi	输出关于服务环境的详细信息（区别于 conf 命令）。

ZooKeeper 四字命令	功能描述
reqs	列出未经处理的请求
ruok	测试服务是否处于正确状态。如果确实如此，那么服务返回“ imok ”，否则不做任何相应。
stat	输出关于性能和连接的客户端的列表。
wchs	列出服务器 watch 的详细信息。
wchc	通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表。
wchp	通过路径列出服务器 watch 的详细信息。它输出一个与 session 相关的路径。

使用案例：

```
[root@CentOS zookeeper-3.4.6]# echo ruok |nc 127.0.0.1 2181
Imok
[root@CentOS zookeeper-3.4.echo conf |nc 127.0.0.1 2181
clientPort=2181
dataDir=/data/d1/version-2
dataLogDir=/data/d1/version-2
tickTime=2000
maxClientCnxns=60
minSessionTimeout=4000
maxSessionTimeout=40000
serverId=1
initLimit=10
syncLimit=5
electionAlg=3
electionPort=3887
quorumPort=2887
peerType=0
```

ZooKeeper 命令行工具

当启动 ZooKeeper 服务成功之后，输入下述命令，连接到 ZooKeeper 服务：
zkCli.sh –server 10.77.20.23:2181
输入 help 之后，屏幕会输出可用的 ZooKeeper 命令，如下图所示
[zk: 127.0.0.1:2182(CONNECTED) 0] help
ZooKeeper -server host:port cmd args
connect host:port
get path [watch]
ls path [watch]
set path data [version]
rmr path
delquota [-n|-b] path
quit
printwatches on|off
create [-s] [-e] path data acl
stat path [watch]
close
ls2 path [watch]
history
listquota path
setAcl path acl
getAcl path
sync path
redo cmdno
addauth scheme auth
delete path [version]
setquota -n|-b val path

ZooKeeper 简单操作

- 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容：

  [zk: 127.0.0.1:2182(CONNECTED) 1] ls /
  [zookeeper]
- 创建一个新的 znode ，使用 create /zk myData 。这个命令创建了一个新的 znode 节点“ zk ”以及与它关联的字符串：
  [zk: 127.0.0.1:2182(CONNECTED) 9] create /jiangzz "helloworld"
  Created /jiangzz
  [zk: 127.0.0.1:2182(CONNECTED) 10] ls /
  [jiangzz, zookeeper]

- 运行 get 命令来确认第二步中所创建的 znode 是否包含我们所创建的字符串：

```
[zk: 127.0.0.1:2182(CONNECTED) 11] get /jiangzz
"helloworld"
cZxid = 0x10000000b
ctime = Thu Aug 13 05:58:13 CST 2015
mZxid = 0x10000000b
mtime = Thu Aug 13 05:58:13 CST 2015
pZxid = 0x10000000b
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 12
numChildren = 0
```

- 通过 set 命令来对 zk 所关联的字符串进行设置：

```
[zk: 127.0.0.1:2182(CONNECTED) 14] set /jiangzz "lovejava"
cZxid = 0x10000000b
ctime = Thu Aug 13 05:58:13 CST 2015
mZxid = 0x10000000c
mtime = Thu Aug 13 06:01:25 CST 2015
pZxid = 0x10000000b
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 10
numChildren = 0
```

- 下面我们将刚才创建的 znode 删除

```
[zk: 127.0.0.1:2182(CONNECTED) 15] delete /jiangzz
[zk: 127.0.0.1:2182(CONNECTED) 16] ls /
[zookeeper]
```

查看/jiangzz 已经被删除

Zookeeper ACL 控制

传统的文件系统中，ACL 分为两个维度，一个是属组，一个是权限，子目录/文件默认继承父目录的 ACL。而在 Zookeeper 中，znode 的 ACL 是没有继承关系的，是独立控制的。Zookeeper 的 ACL，可以从三个维度来理解：一是 scheme; 二是 user; 三是 permission，通常表示为 scheme:id:permissions, 下面从这三个方面分别来介绍：

scheme: scheme 对应于采用哪种方案来进行权限管理，zookeeper 实现了一个 pluggable 的 ACL 方案，可以通过扩展 scheme，来扩展 ACL 的机制。zookeeper-3.4.4 缺省支持下面几种 scheme:

world: 它下面只有一个 id, 叫 anyone, world:anyone 代表任何人，zookeeper 中对所有人有权限的结点就是属于 world:anyone 的

auth: 它不需要 id, 只要是通过 authentication 的 user 都有权限（zookeeper 支持通过 kerberos 来进行 authencation, 也支持 username/password 形式的 authentication)

digest: 它对应的 id 为 username:BASE64(SHA1(password))，它需要先通过 username:password 形式的 authentication

ip: 它对应的 id 为客户机的 IP 地址，设置的时候可以设置一个 ip 段，比如 ip:192.168.1.0/16, 表示匹配前 16 个 bit 的 IP 段

super: 在这种 scheme 情况下，对应的 id 拥有超级权限，可以做任何事情(cdrwa)

id: id 与 scheme 是紧密相关的，具体的情况在上面介绍 scheme 的过程都已介绍，这里不再赘述。

permission: zookeeper 目前支持下面一些权限：

CREATE(c): 创建权限，可以在在当前 node 下创建 child node

DELETE(d): 删除权限，可以删除当前的 node

READ(r): 读权限，可以获取当前 node 的数据，可以 list 当前 node 所有的 child nodes

WRITE(w): 写权限，可以向当前 node 写数据

ADMIN(a): 管理权限，可以设置当前 node 的 permission

实现

如前所述，在 zookeeper 中提供了一种 pluggable 的 ACL 机制。具体来说就是每种 scheme 对应于一种 ACL 机制，可以通过扩展 scheme 来扩展 ACL 的机制。在具体的实现中，每种 scheme 对应一种 AuthenticationProvider。每种 AuthenticationProvider 实现了当前机制下 authentication 的检查，通过了 authentication 的检查，然后再进行统一的 permission 检查，如此便实现了 ACL。所有的 AuthenticationProvider 都注册在 ProviderRegistry 中，新扩展的 AuthenticationProvider 可以通过配置注册到 ProviderRegistry 中去。下面是实施检查的具体实现.

```
void checkACL(ZooKeeperServer zks, List<acl> acl,
    int perm, List<id> ids) throws KeeperException.NoAuthException {
    if (skipACL)return;
    if (acl == null || acl.size() == 0) return;
    for (Id authId : ids) {
        if (authId.getScheme().equals("super")) {
            return;
        }
    }
    for (ACL a : acl) {
        Id id = a.getId();
        if ((a.getPerms() & perm) != 0) {
            if (id.getScheme().equals("world")
                && id.getId().equals("anyone")) {
                return;
            }
            AuthenticationProvider ap =
                ProviderRegistry.getProvider(id .getScheme());
            if (ap != null) {
                for (Id authId : ids) {
                    if (authId.getScheme().equals(id.getScheme())
                        && ap.matches(authId.getId(), id.getId())) {
                        return;
                    }
                }
            }
        }
    }
    throw new KeeperException.NoAuthException();
}
```

管理 **ACL**

案例：

1.使用 **schema**

创建一个节点，任何人对该节点具有创建、读取、修改、删除、管理权限

[zk: 127.0.0.1:2182(CONNECTED) 32] create -s /jiangzz hello world:anyone:crwda

Created /jiangzz0000000004

[zk: 127.0.0.1:2182(CONNECTED) 33] getAcl /jiangzz0000000004

'world,'anyone

: cdrwa

2.使用 **digest**

create /node digest:jiangzz:zOL5mkauOP5kV9xnATsWSNNqEdw=:cdrwa