

# zookeeper系统知识

## DEMO项目地址

<https://github.com/ShouZhiDuan/zookeeper>

## 下载地址

<https://apache.osuosl.org/zookeeper/zookeeper-3.7.0/apache-zookeeper-3.7.0-bin.tar.gz>

## 保存目录

D:\download\apache-zookeeper-3.7.0-bin.tar.gz

## 启动命令

- bin/zkServer.sh start

```
root@master-k8s:/usr/local/zk/apache-zookeeper-3.7.0-bin# bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /usr/local/zk/apache-zookeeper-3.7.0-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

- 如果启动出现

```
2021-07-19 07:39:32,969 [myid:1] - INFO  [main:SnappyStream@611] - zookeeper snapshot compression method = CHECKED
2021-07-19 07:39:32,982 [myid:1] - ERROR [main:ZooKeeperServerMain@91] - Unexpected exception, exiting abnormally
java.io.IOException: No snapshot found, but there are log entries. Something is broken!
    at org.apache.zookeeper.server.persistence.FileTxnSnapLog.restore(FileTxnSnapLog.java:290)
    at org.apache.zookeeper.server.ZKDatabase.loadDataBase(ZKDatabase.java:286)
    at org.apache.zookeeper.server.ZooKeeperServer.loadData(ZooKeeperServer.java:516)
    at org.apache.zookeeper.server.ZooKeeperServer.startData(ZooKeeperServer.java:683)
    at org.apache.zookeeper.server.NIOServerCnxnFactory.startup(NIOServerCnxnFactory.java:744)
    at org.apache.zookeeper.server.ServerCnxnFactory.startup(ServerCnxnFactory.java:130)
    at org.apache.zookeeper.server.ZooKeeperServerMain.runFromConfig(ZooKeeperServerMain.java:161)
    at org.apache.zookeeper.server.ZooKeeperServerMain.initializeAndRun(ZooKeeperServerMain.java:113)
    at org.apache.zookeeper.server.ZooKeeperServerMain.main(ZooKeeperServerMain.java:68)
    at org.apache.zookeeper.server.quorum.QuorumPeerMain.initializeAndRun(QuorumPeerMain.java:141)
```

有可能这台机器原来安装过zookeeper，清除配置文件中的目录dataDir=/tmp/zookeeper,或者重新创建一个dataDir=/tmp/resource-zookeeper，在启动就好了。

## 常用命令

- bin/zkServer.sh status/stop/restart
- 创建类型
  - create [-s] [-e] [-c] [-t ttl] path [data] [acl]
- 删除类型
  - delete [-v version] path
- 查询类型
  - get [-s] [-w] path

## 常用命令参考

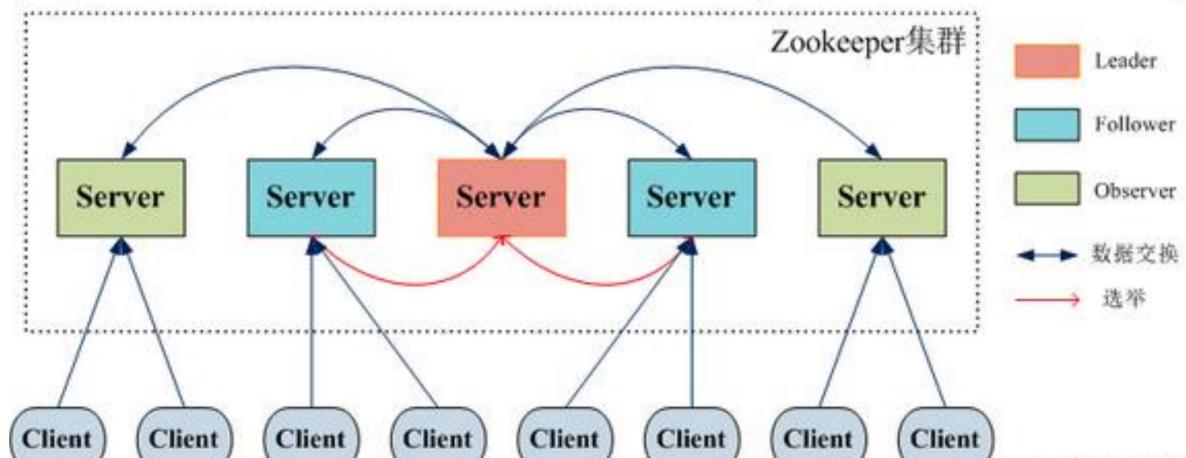
- 增删改查、监听
  - <https://www.cnblogs.com/dalianpai/p/12682368.html>

- ACL操作
  - <https://www.cnblogs.com/jkin/p/14778342.html>

## 客户端连接zk服务端

- bin/zkCli.sh -timeout 3000 -r -server 192.168.10.33:2185

## 集群架构



## ZK数据类型

- 1、持久节点

一直存在。

- 2、持久有序节点

同级节点存在顺序关系。

- 3、临时节点

临时节点会跟客户端会话绑定，客户端离线则当前临时节点就会清除。

- 4、临时有序节点

在临时节点的基础上多了一个顺序。

- 5、CONTAINER节点

当子节点都被删除后，container也会被删除。

- 6、PERSISTENT\_WITH\_TTL

客户端断开连接后不会自动删除znode，如果该znode没有子znode且在给定TTL时间内无修改，该znode将会被删除。TTL单位是毫秒，必须大于0且小于或等于EphemeralType.MAX\_TTL。

## 数据节点信息

- 创建节点并且赋值

```
create /name "shouzhi@duan"
```

- 查看节点详情

```
get -s /name
```

状态属性	说 明
czxid	即 Created ZXID，表示该数据节点被创建时的事务 ID
mzxid	即 Modified ZXID，表示该节点最后一次被更新时的事务 ID
ctime	即 Created Time，表示节点被创建的时间
mtime	即 Modified Time，表示该节点最后一次被更新的时间
version	数据节点的版本号。关于 ZooKeeper 中版本相关的内容，将在 7.1.3 节中做详细讲解
cversion	子节点的版本号
aversion	节点的 ACL 版本号
ephemeralOwner	创建该临时节点的会话的 sessionID。如果该节点是持久节点，那么这个属性值为 0
dataLength	数据内容的长度
numChildren	当前节点的子节点个数
pzxid	表示该节点的子节点列表最后一次被修改时的事务 ID。注意，只有子节点列表变更了才会变更 pzxid，子节点内容变更不会影响 pzxid

```
[zk: 192.168.10.33:2185(CONNECTED) 38] get -s /name
shouzhi@duan 值
czxid = 0xd    创建时候的事务操作ID
ctime = Mon Jul 19 11:40:08 UTC 2021 创建时间
mzxid = 0xd    修改时候的事务操作ID
mtime = Mon Jul 19 11:40:08 UTC 2021 修改时间
pzxid = 0xd 记录子节点列表变更时候的事务ID，注意只有子节点成员数量变了才会更新，子节点的值变动
不会更新。
cversion = 0    当前节点的子节点的版本号，版本不同说明子节点有被变更过。
dataVersion = 0 当前节点自身数据版本。
aclVersion = 0  当前节点权限版本号
ephemeralOwner = 0x0 临时节点会话sessionID，持久节点的这个属性值则为0
dataLength = 12 值的长度
numChildren = 0 子节点的数量
```

## SpringBoot操作ZK

### 1、maven依赖

```
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>2.7.1</version>
</dependency>
```

### 2、增删改查

```
package com.example.zk.testmain;

import org.apache.curator.framework.CuratorFramework;
```

```
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.data.Stat;

/**
 * @Auther: Shouzhi@Duan
 * @Description: 增删改查操作
 */
public class CRUDClientTest {

    private static String CONNECTION_STR="192.168.10.33:2185";

    public static void main(String[] args) throws Exception {
        //CuratorFramework curatorFramework=
        CuratorFrameworkFactory.newClient("");
        CuratorFramework curatorFramework =
            CuratorFrameworkFactory
                .builder()
                .connectString(CONNECTION_STR)
                .sessionTimeoutMs(5000).retryPolicy(new
        ExponentialBackoffRetry(1000,3))
                .namespace("curator1") //工作空间，不同的空间可以存在相同的数据名称
                .build();
        //ExponentialBackoffRetry
        //RetryOneTime 仅仅只重试一次
        //RetryUntilElapsed
        //RetryNTimes
        curatorFramework.start(); //启动
        createData(curatorFramework);
        //      updateData(curatorFramework);
        //      deleteData(curatorFramework);
        //CRUD
        //      curatorFramework.create();
        //      curatorFramework.setData(); //修改
        //      curatorFramework.delete(); //删除
        //      curatorFramework.getData(); //查询
    }

    private static void createData(CuratorFramework curatorFramework) throws
Exception {

    curatorFramework.create().creatingParentsIfNeeded().withMode(CreateMode.PERSISTENT).forPath("/data/program","test".getBytes());
}

    private static void updateData(CuratorFramework curatorFramework) throws
Exception {
    curatorFramework.setData().forPath("/data/program","up".getBytes());
}

    private static void deleteData(CuratorFramework curatorFramework) throws
Exception {
    Stat stat = new Stat();
    //设置stat
    curatorFramework.getData().storingStatIn(stat).forPath("/data/program");
}
```

```
//乐观锁执行删除

curatorFramework.delete().withVersion(stat.getVersion()).forPath("/data/program");
}
}
```

### 3、数据目录权限操作

```
package com.example.zk.testmain;

import org.apache.curator.framework.AuthInfo;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.ZooDefs;
import org.apache.zookeeper.data.ACL;
import org.apache.zookeeper.data.Id;
import org.apache.zookeeper.server.auth.DigestAuthenticationProvider;
import org.junit.Test;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;

/**
 * @Auther: shouzhi@Duan
 * @Description: 权限操作
 */
public class ACLClientTest {
    private final static String CONNECT_STR = "192.168.10.33:2185";
    private final static String PWD_1 = "123456";
    private final static String PWD_2 = "666666";
    private final static String NAME_SPACE = "namespace";

    private static CuratorFramework curatorFramework;
    private static CuratorFramework curatorFrameworkQuery;

    static {
        curatorFramework = CuratorFrameworkFactory.builder().
            connectString(CONNECT_STR)
            .sessionTimeoutMs(5000)
            .retryPolicy(new ExponentialBackoffRetry(1000,3))
            .namespace(NAME_SPACE) //工作空间
            .build();
    }
    static {
        //设置授权认证信息
        AuthInfo authInfo = new AuthInfo("digest",PWD_1.getBytes());
        List<AuthInfo> authInfos=new ArrayList<>();
        authInfos.add(authInfo);
        curatorFrameworkQuery = CuratorFrameworkFactory.builder().
            connectString(CONNECT_STR)
            .sessionTimeoutMs(5000)
            .retryPolicy(new ExponentialBackoffRetry(1000,3))
```

```

        .authorization(authInfos) //授权
        .namespace(NAME_SPACE) //工作空间
        .build();
    }

    private List<ACL> acls() throws NoSuchAlgorithmException {
        List<ACL> acl = new ArrayList<>();
        Id id1 = new Id("digest",
DigestAuthenticationProvider.generateDigest(PWD_1));
        Id id2 = new Id("digest",
DigestAuthenticationProvider.generateDigest(PWD_2));
        ACL acl1 = new ACL(ZooDefs.Perms.ALL, id1);
        ACL acl2 = new ACL(ZooDefs.Perms.DELETE | ZooDefs.Perms.READ, id2);
        acl.add(acl1);
        acl.add(acl2);
        return acl;
    }

    /**
     * 添加节点
     */
    @Test
    public void testDoACL() throws Exception {
        List<ACL> acls = acls();
        curatorFramework.start();
        curatorFramework.create()
            .withMode(CreateMode.PERSISTENT)//数据模式
            .withACL(acls) //设置权限
            .forPath("/acls-test-2","测试目录权限".getBytes());
        curatorFramework.close();
    }

    /**
     * 查询节点
     */
    @Test
    public void test2() throws Exception {
        curatorFramework.start();
        byte[] bytes = curatorFramework
            .getData()
            .forPath("/acls-test-2");
        String s = new String(bytes);
        System.out.printf(s);
    }
}

```

## 权限以及验证方式

推荐参考：

- <https://www.cnblogs.com/jkin/p/14778342.html>
- [https://blog.csdn.net/weixin\\_42073629/article/details/107872159](https://blog.csdn.net/weixin_42073629/article/details/107872159)

# Curator 三种 Watcher

---

- PathChildCache：监视一个路径下孩子结点的创建、删除、更新。
- NodeCache：监视当前结点的创建、更新、删除，并将结点的数据缓存在本地。
- TreeCache：PathChildCache 和 NodeCache 的“合体”，监视路径下的创建、更新、删除事件，并缓存路径下所有孩子结点的数据。

# Curator 分布式锁

---

## 1、原理

- 节点唯一性
- 节点有序性：每个客户端只会监听比自己小一个的加点删除事件。

## 2、锁的几种类型

- 分布式可重入排它锁
  - InterProcessMutex
- 分布式排它锁
  - InterProcessSemaphoreMutex
- 分布式读写锁
  - InterProcessReadWriteLock

# Leader选举

---

- leader latch 不得再次参与master选举
  - spark
  - kafka
- leader selector 可以再次参与master选举