zookeeper

- zookeeper 开源客户端curator介绍
- zookeeper四字监控命令
- zookeeper图形化的客户端工具(ZooInspector)
- taokeeper监控工具的使用

1.zookeeper 开源客户端curator介绍

1.1 curator简介

curator是Netflix公司开源的一个zookeeper客户端，后捐献给apache，curator框架在zookeeper原生API接口上进行了包装，解决了很多zooKeeper客户端非常底层的细节开发。提供zooKeeper各种应用场景(比如：分布式锁服务、集群领导选举、共享计数器、缓存机制、分布式队列等)的抽象封装，实现了Fluent风格的API接口，是最好用，最流行的zookeeper的客户端。

原生zookeeperAPI的不足：

- 连接对象异步创建，需要开发人员自行编码等待
- 连接没有自动重连超时机制
- watcher一次注册生效一次
- 不支持递归创建树形节点

  curator特点：

- 解决session会话超时重连
- watcher反复注册
- 简化开发api
- 遵循Fluent风格的API
- 提供了分布式锁服务、共享计数器、缓存机制等机制

  maven依赖:

```xml
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.7</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-framework</artifactId>
            <version>2.6.0</version>
            <type>jar</type>
            <exclusions>
                <exclusion>
                    <groupId>org.apache.zookeeper</groupId>
                    <artifactId>zookeeper</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.4.10</version>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>org.apache.curator</groupId>
            <artifactId>curator-recipes</artifactId>
            <version>2.6.0</version>
            <type>jar</type>
        </dependency>
```

1.2 连接到ZooKeeper

案例：

```
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.apache.curator.retry.RetryOneTime;

public class CuratorConnection {
    public static void main(String[] args) {
        // session重连策略
        /*
            3秒后重连一次，只重连1次
            RetryPolicy retryPolicy = new RetryOneTime(3000);
        */
        /*
            每3秒重连一次，重连3次
            RetryPolicy retryPolicy = new RetryNTimes(3,3000);
        */
       /*
            每3秒重连一次，总等待时间超过10秒后停止重连
            RetryPolicy retryPolicy=new RetryUntilElapsed(10000,3000);
       */
        // baseSleepTimeMs * Math.max(1, random.nextInt(1 << (retryCount
+ 1)))
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);

        // 创建连接对象
        CuratorFramework client= CuratorFrameworkFactory.builder()
                // IP地址端口号

.connectString("192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:21
83")
                // 会话超时时间
                .sessionTimeoutMs(5000)
                // 重连机制
                .retryPolicy(retryPolicy)
                // 命名空间
                .namespace("create")
                // 构建连接对象
                .build();

        // 打开连接
        client.start();
        System.out.println(client.isStarted());
        // 关闭连接
        client.close();
    }
}
```

1.3 新增节点

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
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.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.util.ArrayList;
import java.util.List;

public class CuratorCreate {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString(IP)
                .sessionTimeoutMs(5000)
                .retryPolicy(retryPolicy)
                .namespace("create")
                .build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }


    @Test
    public void create1() throws Exception {
        // 新增节点
        client.create()
                // 节点的类型
                .withMode(CreateMode.PERSISTENT)
                // 节点的权限列表 world:anyone:cdrwa
                .withACL(ZooDefs.Ids.OPEN_ACL_UNSAFE)
                // arg1:节点的路径
                // arg2:节点的数据
                .forPath("/node1", "node1".getBytes());
        System.out.println("结束");
    }


    @Test
    public void create2() throws Exception {
        // 自定义权限列表
        // 权限列表
        List<ACL> list = new ArrayList<ACL>();
        // 授权模式和授权对象
        Id id = new Id("ip", "192.168.60.130");
        list.add(new ACL(ZooDefs.Perms.ALL, id));

client.create().withMode(CreateMode.PERSISTENT).withACL(list).forPath("/n
ode2", "node2".getBytes());
        System.out.println("结束");
    }

    @Test
    public void create3() throws Exception {
        // 递归创建节点树
        client.create()
                // 递归节点的创建
                .creatingParentsIfNeeded()
                .withMode(CreateMode.PERSISTENT)
                .withACL(ZooDefs.Ids.OPEN_ACL_UNSAFE)
                .forPath("/node3/node31", "node31".getBytes());
        System.out.println("结束");
    }
    @Test
    public void create4() throws Exception {
        // 异步方式创建节点
        client.create()
                .creatingParentsIfNeeded()
                .withMode(CreateMode.PERSISTENT)
                .withACL(ZooDefs.Ids.OPEN_ACL_UNSAFE)
                // 异步回调接口
                .inBackground(new BackgroundCallback() {
                     public void processResult(CuratorFramework
curatorFramework, CuratorEvent curatorEvent) throws Exception {
                          // 节点的路径
                          System.out.println(curatorEvent.getPath());
                          // 时间类型
                          System.out.println(curatorEvent.getType());
                     }
        })
                .forPath("/node4","node4".getBytes());
        Thread.sleep(5000);
        System.out.println("结束");
    }
}
```

1.4 更新节点

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CuratorSet {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString(IP)
                .sessionTimeoutMs(5000)
                .retryPolicy(retryPolicy)
                .namespace("set").build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }

    @Test
    public void set1() throws Exception {
        // 更新节点
        client.setData()
                // arg1:节点的路径
                // arg2:节点的数据
                .forPath("/node1", "node11".getBytes());

        System.out.println("结束");
    }

    @Test
    public void set2() throws Exception {
        client.setData()
                // 指定版本号
                .withVersion(2)
                .forPath("/node1", "node1111".getBytes());
        System.out.println("结束");
    }

    @Test
    public void set3() throws Exception {
        // 异步方式修改节点数据
        client.setData()
                .withVersion(-1).inBackground(new BackgroundCallback() {
            public void processResult(CuratorFramework curatorFramework,
CuratorEvent curatorEvent) throws Exception {
                // 节点的路径
                System.out.println(curatorEvent.getPath());
                // 事件的类型
                System.out.println(curatorEvent.getType());
            }
        }).forPath("/node1", "node1".getBytes());
        Thread.sleep(5000);
        System.out.println("结束");
    }
}
```

1.5 删除节点

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
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.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.util.ArrayList;
import java.util.List;

public class CuratorDelete {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
            .connectString(IP)
                .sessionTimeoutMs(10000)
                .retryPolicy(retryPolicy)
                .namespace("delete").build();
        client.start();
}

    @After
    public void after() {
        client.close();
    }

    @Test

    public void delete1() throws Exception {
        // 删除节点
        client.delete()
                // 节点的路径
                .forPath("/node1");
        System.out.println("结束");
    }


    @Test
    public void delete2() throws Exception {
        client.delete()
                // 版本号
                .withVersion(0)
                .forPath("/node1");
        System.out.println("结束");
    }

    @Test
    public void delete3() throws Exception {
        //删除包含字节点的节点
        client.delete()
                .deletingChildrenIfNeeded()
                .withVersion(-1)
                .forPath("/node1");
        System.out.println("结束");
    }

    @Test
    public void delete4() throws Exception {
        // 异步方式删除节点
        client.delete()
                .deletingChildrenIfNeeded()
                .withVersion(-1)
                .inBackground(new BackgroundCallback() {
                    public void processResult(CuratorFramework
curatorFramework, CuratorEvent curatorEvent) throws Exception {
                        // 节点路径
                        System.out.println(curatorEvent.getPath());
                        // 事件类型
                        System.out.println(curatorEvent.getType());

                    }
                })
                .forPath("/node1");
        Thread.sleep(5000);
        System.out.println("结束");
    }
}
```

1.6 查看节点

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.apache.zookeeper.data.Stat;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CuratorGet {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString(IP)
                .sessionTimeoutMs(10000).retryPolicy(retryPolicy)
                .namespace("get").build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }

    @Test
    public void get1() throws Exception {
        // 读取节点数据
        byte [] bys=client.getData()
                 // 节点的路径
                .forPath("/node1");
        System.out.println(new String(bys));

    }
    @Test
    public void get2() throws Exception {
        // 读取数据时读取节点的属性
        Stat stat=new Stat();
        byte [] bys=client.getData()
                // 读取属性
                .storingStatIn(stat)
                .forPath("/node1");
        System.out.println(new String(bys));
        System.out.println(stat.getVersion());
    }

    @Test
    public void get3() throws Exception {
        // 异步方式读取节点的数据
        client.getData()
                .inBackground(new BackgroundCallback() {
                    public void processResult(CuratorFramework
curatorFramework, CuratorEvent curatorEvent) throws Exception {
                        // 节点的路径
                        System.out.println(curatorEvent.getPath());
                        // 事件类型
                        System.out.println(curatorEvent.getType());
                        // 数据
                        System.out.println(new
String(curatorEvent.getData()));
                    }
                })
                .forPath("/node1");
        Thread.sleep(5000);
        System.out.println("结束");
    }
}
```

1.7 查看子节点

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.apache.zookeeper.data.Stat;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.util.List;

public class CuratorGetChild {


    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString(IP)
                .sessionTimeoutMs(10000).retryPolicy(retryPolicy)
                .build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }

    @Test
    public void getChild1() throws Exception {
        // 读取子节点数据
        List<String> list = client.getChildren()

                // 节点路径
                .forPath("/get");
        for (String str : list) {
            System.out.println(str);
        }
    }

    @Test
    public void getChild2() throws Exception {
        // 异步方式读取子节点数据
        client.getChildren()
                .inBackground(new BackgroundCallback() {
                    public void processResult(CuratorFramework
curatorFramework, CuratorEvent curatorEvent) throws Exception {
                        // 节点路径
                        System.out.println(curatorEvent.getPath());
                        // 事件类型
                        System.out.println(curatorEvent.getType());
                        // 读取子节点数据
                        List<String> list=curatorEvent.getChildren();
                        for (String str : list) {
                            System.out.println(str);
                        }
                    }
                })
                .forPath("/get");
        Thread.sleep(5000);
        System.out.println("结束");
    }
}
```

1.8 检查节点是否存在

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.apache.zookeeper.data.Stat;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CuratorExists {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString(IP)
                .sessionTimeoutMs(10000).retryPolicy(retryPolicy)
                .namespace("get").build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }

    @Test
    public void exists1() throws Exception {
        // 判断节点是否存在
       Stat stat= client.checkExists()
                 // 节点路径
                .forPath("/node2");
        System.out.println(stat.getVersion());

    }
    @Test
    public void exists2() throws Exception {
        // 异步方式判断节点是否存在
        client.checkExists()
                 .inBackground(new BackgroundCallback() {
                     public void processResult(CuratorFramework
curatorFramework, CuratorEvent curatorEvent) throws Exception {
                         // 节点路径
                         System.out.println(curatorEvent.getPath());
                         // 事件类型
                         System.out.println(curatorEvent.getType());

System.out.println(curatorEvent.getStat().getVersion());
                     }
                 })
                .forPath("/node2");
        Thread.sleep(5000);
        System.out.println("结束");
    }
}
```

1.9 watcherAPI

curator提供了两种Watcher(Cache)来监听结点的变化

- Node Cache : 只是监听某一个特定的节点，监听节点的新增和修改
- PathChildren Cache : 监控一个ZNode的子节点. 当一个子节点增加， 更新，删除时， Path Cache会改变它的状态， 会包含最新的子节点， 子节点的数据和状态

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
import org.apache.curator.framework.recipes.cache.*;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CuratorWatcher {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory
                .builder()
                .connectString(IP)
                .sessionTimeoutMs(10000)
                .retryPolicy(retryPolicy)
                .build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }


    @Test
    public void watcher1() throws Exception {
        // 监视某个节点的数据变化
        // arg1:连接对象

        // arg2:监视的节点路径
       final NodeCache nodeCache=new NodeCache(client,"/watcher1");
        // 启动监视器对象
        nodeCache.start();
        nodeCache.getListenable().addListener(new NodeCacheListener() {
            // 节点变化时回调的方法
            public void nodeChanged() throws Exception {
                System.out.println(nodeCache.getCurrentData().getPath());
                System.out.println(new
String(nodeCache.getCurrentData().getData()));
            }
        });
        Thread.sleep(100000);
        System.out.println("结束");
        //关闭监视器对象
        nodeCache.close();
    }

    @Test
    public void watcher2() throws Exception {
         // 监视子节点的变化
        // arg1:连接对象
        // arg2:监视的节点路径
        // arg3:事件中是否可以获取节点的数据
        PathChildrenCache pathChildrenCache=new
PathChildrenCache(client,"/watcher1",true);
        // 启动监听
        pathChildrenCache.start();
        pathChildrenCache.getListenable().addListener(new
PathChildrenCacheListener() {
            // 当子节点方法变化时回调的方法
            public void childEvent(CuratorFramework curatorFramework,
PathChildrenCacheEvent pathChildrenCacheEvent) throws Exception {
                // 节点的事件类型
                System.out.println(pathChildrenCacheEvent.getType());
                // 节点的路径

System.out.println(pathChildrenCacheEvent.getData().getPath());
                // 节点数据
                System.out.println(new
String(pathChildrenCacheEvent.getData().getData()));

            }
        });
        Thread.sleep(100000);
        System.out.println("结束");
        // 关闭监听
        pathChildrenCache.close();

    }
}
```

1.10 事务

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.api.BackgroundCallback;
import org.apache.curator.framework.api.CuratorEvent;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CuratorTransaction {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory.builder()
                .connectString(IP)
                .sessionTimeoutMs(10000).retryPolicy(retryPolicy)
                .namespace("create").build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }

    @Test
    public void tra1() throws Exception {
        client.inTransaction().
                create().forPath("node1", "node1".getBytes()).
                and().
                create().forPath("node2", "node2".getBytes()).
                and().
                commit();

    }
}
```

1.11 分布式锁

InterProcessMutex：分布式可重入排它锁

InterProcessReadWriteLock：分布式读写锁

案例：

```java
import org.apache.curator.RetryPolicy;
import org.apache.curator.framework.CuratorFramework;
import org.apache.curator.framework.CuratorFrameworkFactory;
import org.apache.curator.framework.recipes.cache.*;
import org.apache.curator.framework.recipes.locks.InterProcessLock;
import org.apache.curator.framework.recipes.locks.InterProcessMutex;
import
org.apache.curator.framework.recipes.locks.InterProcessReadWriteLock;
import
org.apache.curator.framework.recipes.locks.InterProcessSemaphoreMutex;
import org.apache.curator.retry.ExponentialBackoffRetry;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CuratorLock {

    String IP =
"192.168.60.130:2181,192.168.60.130:2182,192.168.60.130:2183";
    CuratorFramework client;

    @Before
    public void before() {
        RetryPolicy retryPolicy = new ExponentialBackoffRetry(1000, 3);
        client = CuratorFrameworkFactory
                .builder()
                .connectString(IP)
                .sessionTimeoutMs(10000)
                .retryPolicy(retryPolicy)
                .build();
        client.start();
    }

    @After
    public void after() {
        client.close();
    }



    @Test
    public void lock1() throws Exception {
        // 排他锁
        // arg1:连接对象
        // arg2:节点路径
        InterProcessLock interProcessLock = new InterProcessMutex(client,
"/lock1");
        System.out.println("等待获取锁对象!");
        // 获取锁
        interProcessLock.acquire();
        for (int i = 1; i <= 10; i++) {
            Thread.sleep(3000);
            System.out.println(i);
        }
        // 释放锁
        interProcessLock.release();
        System.out.println("等待释放锁!");
    }

    @Test
    public void lock2() throws Exception {
        // 读写锁
        InterProcessReadWriteLock interProcessReadWriteLock=new
InterProcessReadWriteLock(client, "/lock1");
        // 获取读锁对象
        InterProcessLock
interProcessLock=interProcessReadWriteLock.readLock();
        System.out.println("等待获取锁对象!");
        // 获取锁
        interProcessLock.acquire();
        for (int i = 1; i <= 10; i++) {
            Thread.sleep(3000);
            System.out.println(i);
        }
        // 释放锁
        interProcessLock.release();
        System.out.println("等待释放锁!");
    }

    @Test
    public void lock3() throws Exception {
        // 读写锁
        InterProcessReadWriteLock interProcessReadWriteLock=new
InterProcessReadWriteLock(client, "/lock1");
        // 获取写锁对象
        InterProcessLock
interProcessLock=interProcessReadWriteLock.writeLock();
        System.out.println("等待获取锁对象!");
        // 获取锁
        interProcessLock.acquire();
        for (int i = 1; i <= 10; i++) {
            Thread.sleep(3000);
            System.out.println(i);
        }
        // 释放锁
        interProcessLock.release();
        System.out.println("等待释放锁!");
    }

}
```

2.zookeeper四字监控命令

zooKeeper支持某些特定的四字命令与其的交互。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。 zooKeeper常用四字命令见下表 所示：

命令	描述
conf	输出相关服务配置的详细信息。比如端口、zk数据及日志配置路径、最大连接数，session超时时间、serverId等
cons	列出所有连接到这台服务器的客户端连接/会话的详细信息。包括“接受/发送”的包数量、session id 、操作延迟、最后的操作执行等信息
crst	重置当前这台服务器所有连接/会话的统计信息
dump	列出未经处理的会话和临时节点
envi	输出关于服务器的环境详细信息
ruok	测试服务是否处于正确运行状态。如果正常返回"imok"，否则返回空
stat	输出服务器的详细信息：接收/发送包数量、连接数、模式（leader/follower）、节点总数、延迟。 所有客户端的列表
srst	重置server状态
wchs	列出服务器watches的简洁信息：连接总数、watching节点总数和watches总数
wchc	通过session分组，列出watch的所有节点，它的输出是一个与 watch 相关的会话的节点列表
mntr	列出集群的健康状态。包括“接受/发送”的包数量、操作延迟、当前服务模式（leader/follower）、节点总数、watch总数、临时节点总数

nc命令工具安装:

```
#root用户安装
#下载安装包
wget http://vault.centos.org/6.6/os/x86_64/Packages/nc-1.84-22.el6.x86_64.rpm
#rpm安装
rpm -iUv nc-1.84-22.el6.x86_64.rpm
```

使用方式，在shell终端输入：echo mntr | nc localhost 2181

2.1 conf命令

conf：输出相关服务配置的详细信息

shell终端输入：echo conf| nc localhost 2181

属性	含义
clientPort	客户端端口号
dataDir	数据快照文件目录 默认情况下100000次事务操作生成一次快照
dataLogDir	事物日志文件目录，生产环境中放在独立的磁盘上
tickTime	服务器之间或客户端与服务器之间维持心跳的时间间隔(以毫秒为单位)
maxClientCnxns	最大连接数
minSessionTimeout	最小session超时 minSessionTimeout=tickTime*2
maxSessionTimeout	最大session超时 maxSessionTimeout=tickTime*20
serverId	服务器编号
initLimit	集群中的follower服务器(F)与leader服务器(L)之间初始连接时能容忍的最多心跳数
syncLimit	集群中的follower服务器(F)与leader服务器(L)之间 请求和应答之间能容忍的最多心跳数
electionAlg	0:基于UDP的LeaderElection 1:基于UDP的FastLeaderElection 2:基于UDP和认证的FastLeaderElection 3:基于TCP的FastLeaderElection 在3.4.10版本中，默认值为3另外三种算法已经被弃用，并且有计划在之后的版本中将它们彻底删除而不再支持
electionPort	选举端口
quorumPort	数据通信端口
peerType	是否为观察者 1为观察者

2.2 cons命令

cons:列出所有连接到这台服务器的客户端连接/会话的详细信息

shell终端输入：echo cons| nc localhost 2181

属性	含义
ip	ip地址
port	端口号
queued	等待被处理的请求数，请求缓存在队列中
received	收到的包数
sent	发送的包数
sid	会话id
lop	最后的操作 GETD-读取数据 DELE-删除数据 CREA-创建数据
est	连接时间戳
to	超时时间
lcxid	当前会话的操作id
lzxid	最大事务id
lresp	最后响应时间戳
llat	最后/最新 延时
minlat	最小延时
maxlat	最大延时
avglat	平均延时

2.3 crst命令

crst:重置当前这台服务器所有连接/会话的统计信息

shell终端输入：echo crst| nc localhost 2181

2.4 dump命令

dump:列出未经处理的会话和临时节点

shell终端输入：echo dump| nc localhost 2181

属性	含义
session id	znode path(1对多 ，处于队列中排队的session和临时节点)

2.5 envi命令

envi:输出关于服务器的环境配置信息

shell终端输入：echo envi| nc localhost 2181

属性	含义
zookeeper.version	版本
host.name	host信息
java.version	java版本
java.vendor	供应商
java.home	运行环境所在目录
java.class.path	classpath
java.library.path	第三方库指定非java类包的位置（如：dll，so）
java.io.tmpdir	默认的临时文件路径
java.compiler	JIT 编译器的名称
os.name	Linux
os.arch	amd64
os.version	3.10.0-514.el7.x86_64
user.name	zookeeper
user.home	/home/zookeeper
user.dir	/home/zookeeper/zookeeper2181/bin

2.6 ruok命令

ruok:测试服务是否处于正确运行状态

shell终端输入：echo ruok| nc localhost 2181

2.7 stat命令

stat:输出服务器的详细信息与srvr相似，但是多了每个连接的会话信息

shell终端输入：echo stat| nc localhost 2181

属性	含义
Zookeeper version	版本
Latency min/avg/max	延时
Received	收包
Sent	发包
Connections	连接数
Outstanding	堆积数
Zxid	最大事物id
Mode	服务器角色
Node count	节点数

2.8 srst命令

srst:重置server状态

shell终端输入：echo srst| nc localhost 2181

2.9 wchs命令

wchs:列出服务器watches的简洁信息

shell终端输入：echo wchs| nc localhost 2181

属性	含义
connectsions	连接数
watch-paths	watch节点数
watchers	watcher数量

2.10 wchc命令

wchc:通过session分组，列出watch的所有节点，它的输出的是一个与 watch 相关的会话的节点列表

问题:

```
wchc is not executed because it is not in the whitelist.
```

解决方法：

```
# 修改启动指令 zkServer.sh

# 注意找到这个信息
else
    echo "JMX disabled by user request" >&2
    ZOOMAIN="org.apache.zookeeper.server.quorum.QuorumPeerMain"
fi

# 下面添加如下信息
ZOOMAIN="-Dzookeeper.4lw.commands.whitelist=* ${ZOOMAIN}"
```

shell终端输入：echo wchc| nc localhost 2181

2.11 wchp命令

wchp:通过路径分组，列出所有的 watch 的session id信息

问题:

```
wchp is not executed because it is not in the whitelist.
```

解决方法：

```
# 修改启动指令 zkServer.sh

# 注意找到这个信息
else
    echo "JMX disabled by user request" >&2
    ZOOMAIN="org.apache.zookeeper.server.quorum.QuorumPeerMain"
fi

# 下面添加如下信息
ZOOMAIN="-Dzookeeper.4lw.commands.whitelist=* ${ZOOMAIN}"
```

shell终端输入：echo wchp| nc localhost 2181

2.12 mntr命令

mntr:列出服务器的健康状态

属性	含义
zk_version	版本
zk_avg_latency	平均延时
zk_max_latency	最大延时
zk_min_latency	最小延时
zk_packets_received	收包数
zk_packets_sent	发包数
zk_num_alive_connections	连接数
zk_outstanding_requests	堆积请求数
zk_server_state	leader/follower 状态
zk_znode_count	znode数量
zk_watch_count	watch数量
zk_ephemerals_count	临时节点（znode）
zk_approximate_data_size	数据大小
zk_open_file_descriptor_count	打开的文件描述符数量
zk_max_file_descriptor_count	最大文件描述符数量

shell终端输入：echo mntr| nc localhost 2181

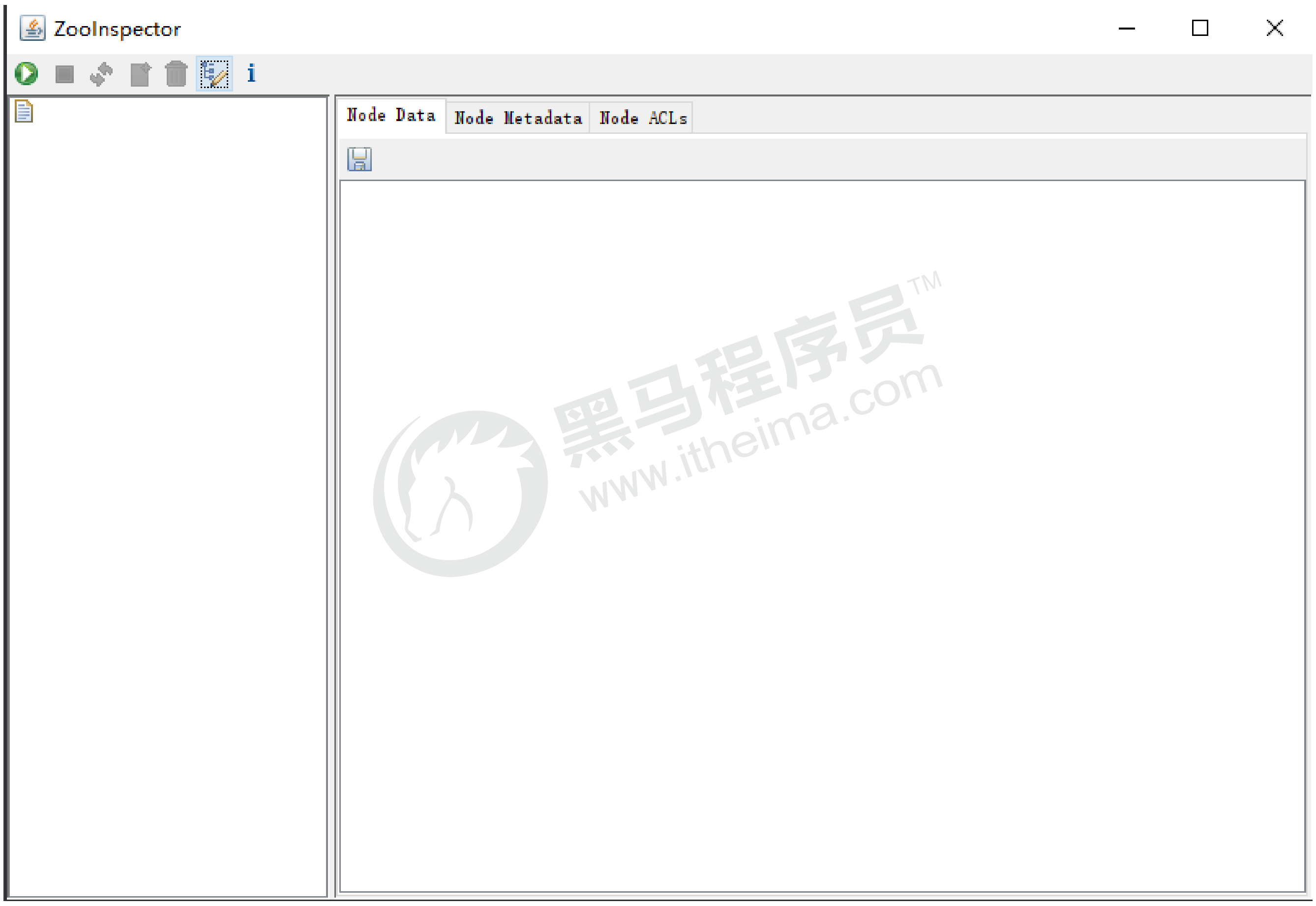
3.zookeeper图形化的客户端工具(ZooInspector)

ZooInspector下载地址：

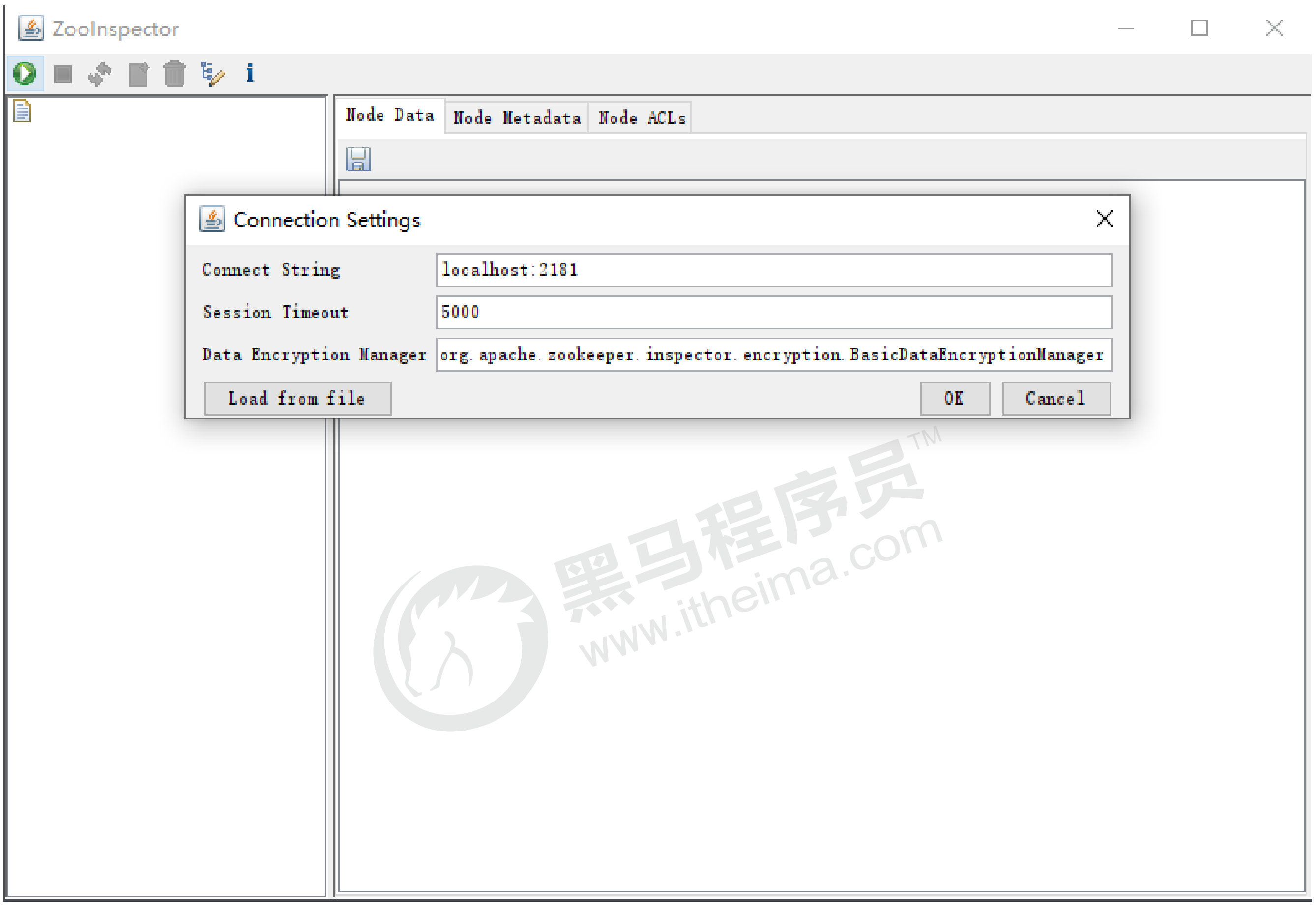
```
https://issues.apache.org/jira/secure/attachment/12436620/ZooInspector.zip
```

解压后进入目录ZooInspector\build，运行zookeeper-dev-ZooInspector.jar

```
#执行命令如下
java -jar zookeeper-dev-ZooInspector.jar
```

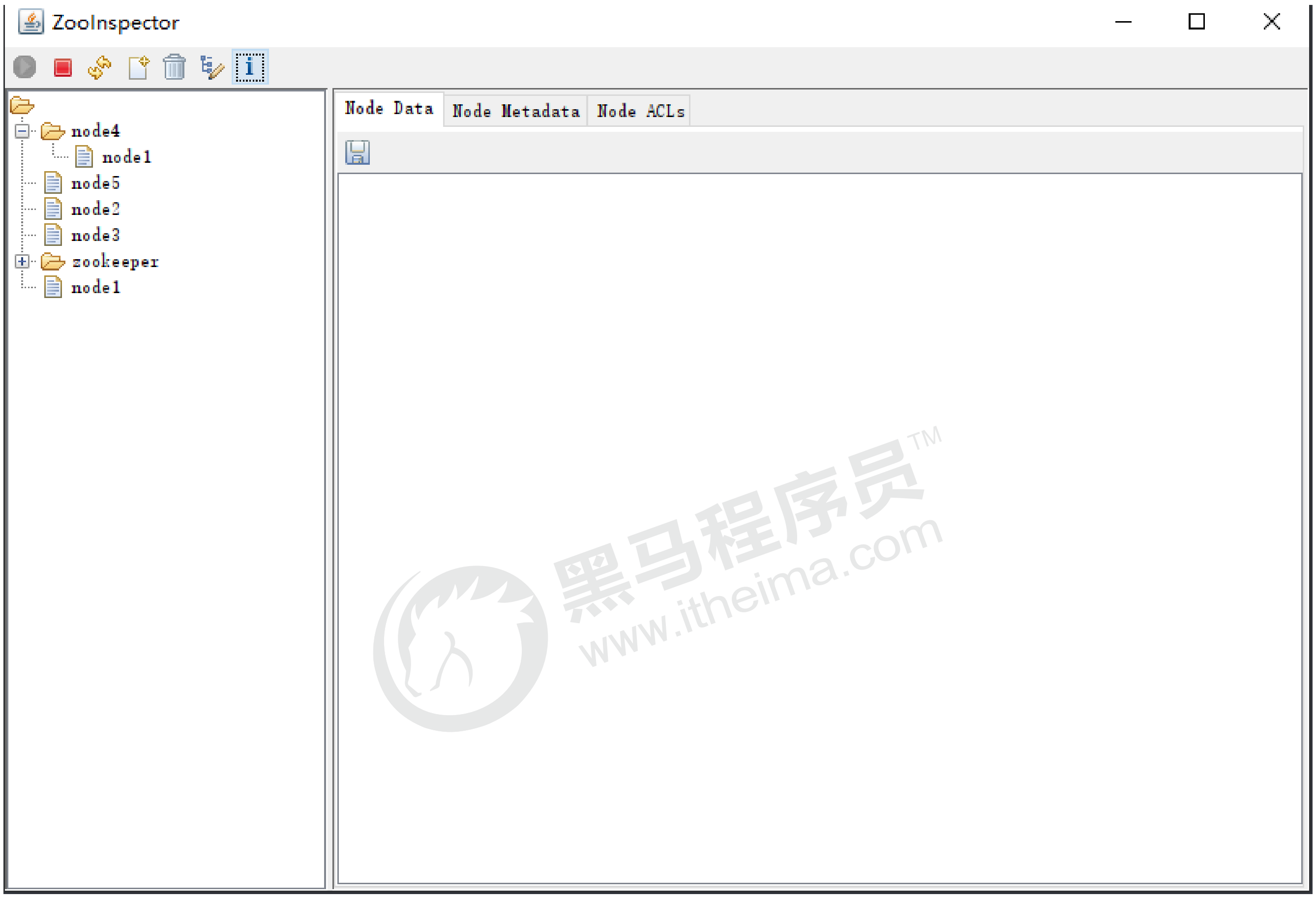


点击左上角连接按钮，输入zk服务地址：ip或者主机名:2181



Connect String: localhost:2181
Session Timeout: 5000
Data Encryption Manager: org.apache.zookeeper.inspector.encryption.BasicDataEncryptionManager

点击OK，即可查看ZK节点信息



4.taokeeper监控工具的使用

基于zookeeper的监控管理工具taokeeper，由淘宝团队开源的zk管理中间件，安装前要求服务前先配置nc 和 sshd

1.下载数据库脚本

```
wget https://github.com/downloads/alibaba/taokeeper/taokeeper.sql
```

2.下载主程序

```
wget https://github.com/downloads/alibaba/taokeeper/taokeeper-monitor.tar.gz
```

3.下载配置文件

```
wget https://github.com/downloads/alibaba/taokeeper/taokeeper-monitor-config.properties
```

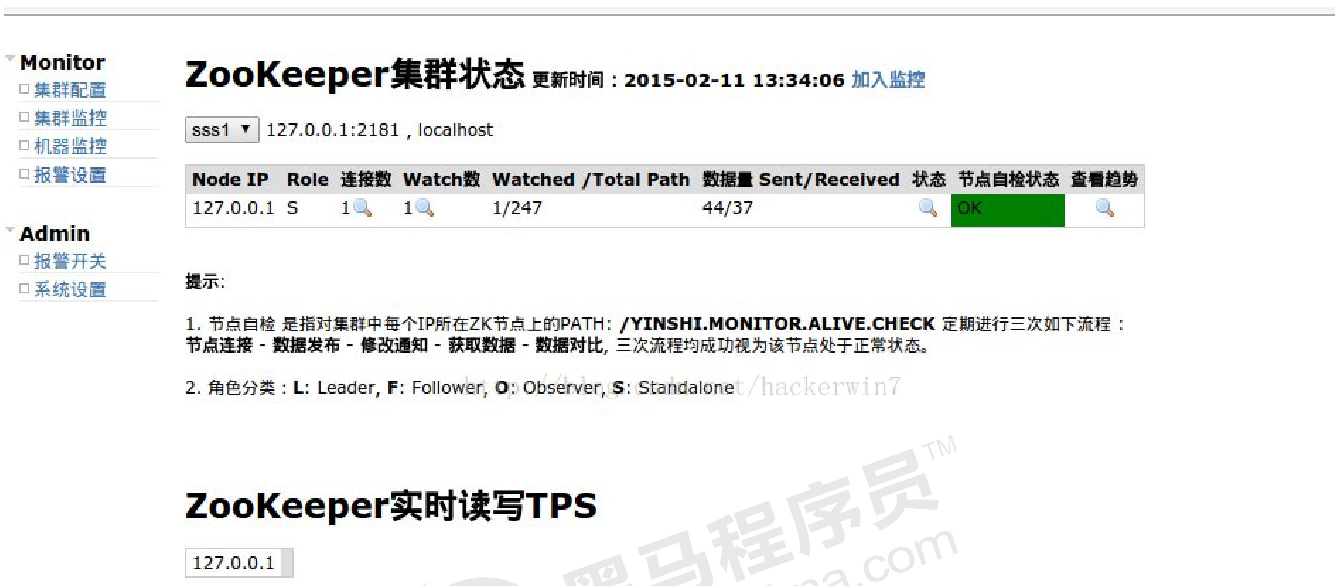
4.配置 taokeeper-monitor-config.properties

```
#Daily
systemInfo.envName=DAILY
#DBCP
dbcp.driverClassName=com.mysql.jdbc.Driver
#mysql连接的ip地址端口号
dbcp.dbJDBCUrl=jdbc:mysql://192.168.60.130:3306/taokeeper
dbcp.characterEncoding=GBK
#用户名
dbcp.username=root
#密码
dbcp.password=root
dbcp.maxActive=30
dbcp.maxIdle=10
dbcp.maxWait=10000
#SystemConstant
#用户存储内部数据的文件夹
#创建/home/zookeeper/taokeeperdata/ZooKeeperClientThroughputStat
SystemConstent.dataStoreBasePath=/home/zookeeper/taokeeperdata
#ssh用户
SystemConstant.userNameOfSSH=zookeeper
#ssh密码
SystemConstant.passwordOfSSH=zookeeper
#Optional
SystemConstant.portOfSSH=22
```

5.安装配置 tomcat，修改catalina.sh

```
#指向配置文件所在的位置
JAVA_OPTS=-DconfigFilePath="/home/zookeeper/taokeeper-monitor-tomcat/webapps/ROOT/conf/taokeeper-monitor-config.properties"
```

6.部署工程启动



Monitor
- 集群配置
- 集群监控
- 机器监控
- 报警设置

Admin
- 报警开关
- 系统设置

ZooKeeper集群状态 更新时间：2015-02-11 13:34:06 加入监控

sss1 ▼ 127.0.0.1:2181 , localhost

Node IP	Role	连接数	Watch数	Watched /Total Path	数据量 Sent/Received	状态	节点自检状态	查看趋势
127.0.0.1	S	1🔍	1🔍	1/247	44/37	🔍	🟩 OK	🔍

提示:

1. 节点自检 是指对集群中每个IP所在ZK节点上的PATH: **/YINSHI.MONITOR.ALIVE.CHECK** 定期进行三次如下流程：**节点连接 - 数据发布 - 修改通知 - 获取数据 - 数据对比,** 三次流程均成功视为该节点处于正常状态。

2. 角色分类：**L**: Leader, **F**: Follower, **O**: Observer, **S**: Standalone

ZooKeeper实时读写TPS

127.0.0.1