**探访zookeeper**

## 背景

随着信息化水平延绵不断地提高，企业级系统发展得越来越庞大和臃肿，性能和体验急剧下降，客户频频抱怨和吐槽。拆分系统是目前可行的有效解决系统的伸缩性和性能问题的方法。但是系统的拆分也带来了系统的复杂性问题----每一个子系统不是单独存在的，他们之间需要需要协作交互、协调配合，为此需要一种统一的服务机制来协调各个子系统的运作，由此衍生了zookeeper即动物园管理员。

## 介绍

本质上zookeeper是一个树形的分布式小文件存储系统。

设计目标：（1）简单化；（2）有序性；（3）健壮性；（4）速度效率优势.

## zookeeper集群

zookeeper集群通常由2n+1台server组成，服务器角色分为leader（领导者）和learner（学习者），learner又分为Observer（观察者）和Fllower(跟随者)。

1.leader(领导者)，负责进行投票的发起和决议，更新系统状态

2.learner(学习者)，包括follower（跟随者）和observer（观察者），

3.follower用于接受客户端请求并向客户端返回结果，在选主过程中参与投票

4.Observer可以接受客户端请求，将写请求转发给leader，但observer不参加投票过程，只同步leader的状态，observer的目的是为了扩展系统，提高读取速度。

5. client(客户端)，请求发起方

zookeeper数据模型

znode具有文件和目录两种特点，既像文件那样有数据信息也像目录拥有路径标识可以有子znode。znode具有原子性操作。

用一颗树状图来表示，每个节点称为一个znode. 每个znode由3部分组成:

1.stat：此为状态信息, 描述该znode的版本, 权限等信息.

2.data：与该znode关联的数据.

3.children：该znode下的子节点.

### Znode是客户端访问ZooKeeper的主要实体对象，包含以下特征：

（1）Watches

客户端可在节点上设置监视器(watch)。当节点状态发生改变时(数据的增、删、改)将触发watch对应的方法事件并执行该代码块，期间watch的方法事件只被触发一次。

（2）数据访问

znode需要被原子性的操作。读操作获取与znode相关的所有数据，写操作也会替换znode的所有数据。每个znode拥有自己的ACL(访问控制列表)，ACL列表规定了用户的权限，限定用户对目标节点可执行的操作。

（3）节点类型

znode分为临时节点和永久节点。节点的类型在创建时即被确定，并且不能改变。  
　　临时节点：生命周期依赖于创建它们的会话。一旦会话结束，临时节点将被自动删除，也可以手动删除。临时节点不允许拥有子节点。  
　　永久节点：生命周期不依赖于会话，只有客户端执行删除操作的后，他们才被删除。

（4）顺序节点（唯一性的保证）

当创建Znode的时候，用户可请求在ZooKeeper的路径结尾添加一个递增的计数。这个计数对于此节点的父节点来说是唯一的，它的格式为“%10d”(10位数字，没有数值的数位用0补充，例如“0000000001”)。当计数值大于232-1时，计数器将溢出。

org.apache.zookeeper.CreateMode中定义了四种节点类型，分别为：

PERSISTENT：永久节点

EPHEMERAL：临时节点

PERSISTENT\_SEQUENTIAL：永久节点、序列化

EPHEMERAL\_SEQUENTIAL：临时节点、序列化

## **搭建zookeeper的单机，集群和伪集群**

参考：

<http://blog.csdn.net/shatelang/article/details/7596007> 八步（介绍和搭建和操作）

<http://blog.csdn.net/tlk20071/article/details/52028945>

<http://blog.csdn.net/top_code/article/details/51348409>

问题与解决：

遇到启动错误

闪退

<http://blog.csdn.net/ljyhust/article/details/72782313>

Myid找不到

<https://zhidao.baidu.com/question/811879934729920332.html>

Zoo.cfg文件的dataDir=路径\改为/

Connection refused and time out

<https://www.cnblogs.com/simple-focus/p/6143495.html>

查看状态

<http://blog.csdn.net/ceasadan/article/details/52343734>

查看状态zkCLI.cmd

Cmd报错。解决：三个cmd各自对应

## **基于zookeeper的分布式锁**

在蜂拥的网络服务消费下，有一些服务或者程序是需要排队来一个一个地单独执行的，比如抢票、抢红包、抢单... ...面对这些类型的需求就催生了各种各样的分布式锁。

*/\*\*  
 \* zookeeper锁,入门版  
 \*/***public class** ZookeeperLock **implements** Lock {  
**private static** ZkClient *zkClient*=**new** ZkClient(**"127.0.0.1:2181"**);  
 **private static final** String ***lock***=**"/lock"**;  
 **private** CountDownLatch **countDownLatch**=**null**;  
  
 */\*\*阻塞式的加锁\*/* @Override  
 **public void** lock() {  
 **if** (tryLock()){  
 System.***out***.println(**"==============get lock success============="**);  
}**else**{  
 waitForLock();  
 lock();  
 }  
 }  
  
 */\*\*暂时挂起,等待解锁后再激活\*/* **private void** waitForLock(){  
 *//1.创建一个监听器* IZkDataListener listener=**new** IZkDataListener() {  
 @Override  
 **public void** handleDataChange(String dataPath, Object data) **throws** Exception {  
 *//3.线程解锁后，抛出事件，让其他线程重新竞争锁* System.***out***.println(**"=====get lock deleted event====="**);  
 **if**(**countDownLatch**!=**null**){  
 **countDownLatch**.countDown();  
 }  
 }  
  
 @Override  
 **public void** handleDataDeleted(String dataPath) **throws** Exception {  
  
 }  
 };  
 *//2.订阅监听器;节点存在时，让线程挂起;  
 zkClient*.subscribeDataChanges(***lock***,listener);  
 **if** (*zkClient*.exists(***lock***)){  
 **countDownLatch**=**new** CountDownLatch(1);  
 **try** {  
 **countDownLatch**.await();  
 }**catch** (InterruptedException e){  
 e.printStackTrace();  
 *//log.error("挂起时countDownLatch出现异常:"+e.getMessage());* }  
 }  
 *//4.退订卸载监听器  
 zkClient*.unsubscribeDataChanges(***lock***,listener);  
 }  
  
 */\*\*非阻塞式加锁;在根节点创建lock节点文件夹\*/* @Override  
 **public boolean** tryLock() {  
 **try** {  
 *//createPersistent持久化节点  
 zkClient*.createPersistent(***lock***);  
 **return true**;  
 }**catch** (ZkNodeExistsException e) {  
 System.***out***.println(e.getMessage());  
 **return false**;  
 }  
 }  
  
 */\*\*解锁\*/* @Override  
 **public void** unlock() {  
 *zkClient*.delete(***lock***);  
 }