Zookeeper常用功能

Zookeeper是一个分布式协调框架，主要用于协调多个进程之间对资源的访问控制和数据共享等。其常用功能如下

1. 作为多进程之间的配置中心
2. 作为命名服务中心
3. 作为分布式集群机的管理中心

针对上面的使用场景，zookeeper都可以提供支持，不过由于zookeeper提供的这些接口太过底层，本身协议就不太好理解，所以很难驾驭，为此Curator框架解决了这一切，使用curator框架前引入依赖

<**dependency**>  
 <**groupId**>org.apache.curator</**groupId**>  
 <**artifactId**>curator-recipes</**artifactId**>  
 <**version**>2.7.0</**version**>  
</**dependency**>

下面是我们几种常用场景下的使用示例

**public class** App {  
 **static** String *ZK\_ADDRESS* = **"localhost:2181"**;  
  
  
 **public static void** main(String[] args) **throws** Exception {  
  
  
 *pathChildenCacheTest*(*ZK\_ADDRESS*, **"/john"**);  
*// nodeCacheTest();  
// treeCacheTest();  
// setPathData();  
// getPathData();  
// getChildrenPath();  
 // getLock(ZK\_PATH);  
  
// for (int i = 0; i < 100; i++) {  
// Thread.sleep(3000);  
// new Thread() {  
// public void run() {  
// try {  
// getMultiLock("/john", "/john2");  
// } catch (Exception e) {  
// e.printStackTrace();  
// }  
// };  
// }.start();  
// }  
  
 // client.delete().forPath(ZK\_PATH);  
  
  
// for (int i = 0; i <10 ; i++) {  
// new Thread(){  
// @Override  
// public void run() {  
// try {  
// getWriteLock("/john/abc");  
// } catch (Exception e) {  
// e.printStackTrace();  
// }  
// }  
// }.start();  
// }  
// for (int i = 0; i <10 ; i++) {  
// new Thread(){  
// @Override  
// public void run() {  
// try {  
// getReadLock("/john/abc");  
// } catch (Exception e) {  
// e.printStackTrace();  
// }  
// }  
// }.start();  
// }  
  
  
// for (int i = 0; i <10 ; i++) {  
// new Thread(){  
// @Override  
// public void run() {  
// try {  
// getSemaphore("/john/abc",4,15L,TimeUnit.SECONDS);  
// } catch (Exception e) {  
// e.printStackTrace();  
// }  
// }  
// }.start();  
// }* }  
  
 */\*\*  
 \* （1）永久监听指定节点下的节点  
 \* （2）只能监听指定节点下一级节点的变化，比如说指定节点”/example”, 在下面添加”node1”可以监听到，但是添加”/example/node1/n1”就不能被监听到了  
 \* （3）可以监听到的事件：节点创建、节点数据的变化、节点删除等  
 \*  
 \** ***@throws*** *Exception  
 \*/* **public static void** pathChildenCacheTest(String zkAddress, String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(zkAddress, **new** RetryNTimes(10, 5000));  
 client.start();  
 PathChildrenCache watcher = **new** PathChildrenCache(client, path, **true** *// if  
 // cache  
 // data* );  
  
 watcher.getListenable().addListener(**new** PathChildrenCacheListener() {  
  
 **public void** childEvent(CuratorFramework framework, PathChildrenCacheEvent event) **throws** Exception {  
 System.***out***.println(event.getType() + **" "** + event.getData().getPath() + **" "** + **new** String(event.getData().getData()));  
  
 }  
 });  
 watcher.start();  
 Thread.*sleep*(Integer.***MAX\_VALUE***);  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 */\*\*  
 \* 监听某一个节点的数据变化  
 \*  
 \** ***@throws*** *Exception  
 \*/* **public static void** nodeCacheTest(String zkAddress, String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(zkAddress, **new** RetryNTimes(10, 5000));  
 client.start();  
 **final** NodeCache watcher = **new** NodeCache(client, path);  
 watcher.start();  
 watcher.getListenable().addListener(**new** NodeCacheListener() {  
  
 **public void** nodeChanged() **throws** Exception {  
 System.***out***.println(**"路径："** + watcher.getCurrentData().getPath());  
 System.***out***.println(**"数据："** + **new** String(watcher.getCurrentData().getData()));  
 System.***out***.println(**"状态："** + watcher.getCurrentData().getStat());  
  
 }  
 });  
 Thread.*sleep*(Integer.***MAX\_VALUE***);  
 watcher.close();  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* 监听某一个节点下所有结点的变化，包括子节点，子节点的节点等，没有级数限制  
 \* 监听的事件有节点的添加，删除，更新  
 \*  
 \** ***@throws*** *Exception  
 \*/* **public static void** treeCacheTest(String zkAddress, String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(zkAddress, **new** RetryNTimes(10, 5000));  
 client.start();  
 TreeCache watcher = **new** TreeCache(client, path);  
 watcher.start();  
 watcher.getListenable().addListener(**new** TreeCacheListener() {  
  
 **public void** childEvent(CuratorFramework client, TreeCacheEvent event) **throws** Exception {  
 System.***out***.println(**"命名空间："** + client.getNamespace());  
 System.***out***.println(**"路径："** + **new** String(event.getData().getPath()));  
 System.***out***.println(**"数据："** + **new** String(event.getData().getData()));  
 System.***out***.println(**"类型："** + event.getType());  
 System.***out***.println(**"状态："** + event.getData().getStat());  
  
 }  
 });  
 Thread.*sleep*(Integer.***MAX\_VALUE***);  
 watcher.close();  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
  
 */\*\*  
 \* 获取某一个路径对应的值  
 \*  
 \** ***@throws*** *Exception  
 \*/* **public static void** getPathData(String zkAddress, String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(zkAddress, **new** RetryNTimes(10, 5000));  
 client.start();  
 **byte**[] bytes = client.getData().forPath(path);  
 System.***out***.println(**new** String(bytes));  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 */\*\*  
 \* 设置某一个路径的值  
 \*  
 \** ***@throws*** *Exception  
 \*/* **public static void** setPathData(String zkAddress, String path, **byte**[] data) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(zkAddress, **new** RetryNTimes(10, 5000));  
 client.start();  
 client.setData().forPath(path, data);  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 */\*\*  
 \* 获取某一个路径下的所有子路径  
 \*  
 \** ***@throws*** *Exception  
 \*/* **public static void** getChildrenPath(String zkAddress, String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(zkAddress, **new** RetryNTimes(10, 5000));  
 client.start();  
 List<String> pathList = client.getChildren().forPath(path);  
 **for** (**int** i = 0; i < pathList.size(); i++) {  
 System.***out***.println(pathList.get(i));  
 }  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* 独占锁，用于分布式多个进程之间的协调  
 \*  
 \** ***@param path*** *\** ***@throws*** *Exception  
 \*/* **public static void** getLock(String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(*ZK\_ADDRESS*, **new** RetryNTimes(10, 5000));  
 client.start();  
 InterProcessMutex lock = **new** InterProcessMutex(client, path);  
 **try** {  
 **if** (lock.acquire(10 \* 1000, TimeUnit.***SECONDS***)) {  
 System.***out***.println(Thread.*currentThread*().getName() + **" hold lock"**);  
 Thread.*sleep*(15000L);  
 System.***out***.println(Thread.*currentThread*().getName() + **" release lock"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 lock.release();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 获取写锁  
 \*  
 \** ***@param path*** *\** ***@throws*** *Exception  
 \*/* **public static void** getReadLock(String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(*ZK\_ADDRESS*, **new** RetryNTimes(10, 5000));  
 client.start();  
 InterProcessReadWriteLock lock = **new** InterProcessReadWriteLock(client, path);  
 InterProcessMutex readLock = lock.readLock();  
 **try** {  
 **if** (readLock.acquire(10 \* 1000, TimeUnit.***SECONDS***)) {  
 System.***out***.println(Thread.*currentThread*().getName() + **" hold readLock"**);  
 Thread.*sleep*(10000L);  
 System.***out***.println(Thread.*currentThread*().getName() + **" release readLock"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 readLock.release();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 获得写锁  
 \*  
 \** ***@param path*** *\** ***@throws*** *Exception  
 \*/* **public static void** getWriteLock(String path) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(*ZK\_ADDRESS*, **new** RetryNTimes(10, 5000));  
 client.start();  
 InterProcessReadWriteLock lock = **new** InterProcessReadWriteLock(client, path);  
 InterProcessMutex writeLock = lock.writeLock();  
 **try** {  
 **if** (writeLock.acquire(10 \* 1000, TimeUnit.***SECONDS***)) {  
 System.***out***.println(Thread.*currentThread*().getName() + **" hold writeLock"**);  
 Thread.*sleep*(10000L);  
 System.***out***.println(Thread.*currentThread*().getName() + **" release writeLock"**);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 writeLock.release();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 获得信号量  
 \*  
 \** ***@param path*** *\** ***@throws*** *Exception  
 \*/* **public static void** getSemaphore(String path, Integer permit, Long time, TimeUnit timeUnit) **throws** Exception {  
  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(*ZK\_ADDRESS*, **new** RetryNTimes(10, 5000));  
 client.start();  
 InterProcessSemaphoreV2 semophore = **new** InterProcessSemaphoreV2(client, path, permit);  
 Lease lease = **null**;  
 **try** {  
 lease = semophore.acquire(time, timeUnit);  
 **if** (lease != **null**) {  
 System.***out***.println(Thread.*currentThread*().getName() + **" hold semophore"**);  
 Thread.*sleep*(10000L);  
 System.***out***.println(Thread.*currentThread*().getName() + **" release semophore"**);  
 } **else** {  
 System.***out***.println(Thread.*currentThread*().getName() + **" can not get semophore"**);  
 }  
  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **if** (lease != **null**) {  
 semophore.returnLease(lease);  
 }  
 } **finally** {  
 **try** {  
 semophore.returnLease(lease);  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }  
  
 */\*\*  
 \* 获得多个锁  
 \*  
 \** ***@param path1*** *\** ***@param path2*** *\** ***@throws*** *Exception  
 \*/* **public static void** getMultiLock(String path1, String path2) **throws** Exception {  
 CuratorFramework client = CuratorFrameworkFactory.*newClient*(*ZK\_ADDRESS*, **new** RetryNTimes(10, 5000));  
 client.start();  
  
 InterProcessLock lock1 = **new** InterProcessMutex(client, path1);  
 InterProcessLock lock2 = **new** InterProcessMutex(client, path2);  
 InterProcessMultiLock lock = **new** InterProcessMultiLock(Arrays.*asList*(lock1, lock2));  
 **try** {  
 **if** (!lock.acquire(10000, TimeUnit.***MICROSECONDS***)) {  
 System.***out***.println(Thread.*currentThread*().getName() + **"不能获得锁"**);  
 } **else** {  
 System.***out***.println(Thread.*currentThread*().getName() + **"获得锁"**);  
 Thread.*sleep*(5000);  
 System.***out***.println(Thread.*currentThread*().getName() + **"是否获取第lock1锁:"** + lock1.isAcquiredInThisProcess());  
 System.***out***.println(Thread.*currentThread*().getName() + **"是否获取第lock2锁:"** + lock2.isAcquiredInThisProcess());  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 **if** (lock.isAcquiredInThisProcess()) {  
 lock.release();  
 System.***out***.println(Thread.*currentThread*().getName() + **"释放多个锁"**);  
  
 }  
 **try** {  
 client.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
}