**ZooKeeper的应用**[](#zookeeper)

* 分布式锁
* 分布式配置中心

**分布式锁**[](#_1)

非分布式场景下：多线程锁实现多线程共享资源同步

**public** **class** **DistributeLock{**

**public** **static** Lock lock **=** **new** ReentrantLock**();**

**public** **static** **void** **main(**String**[]** args**)** **{**

**}**

**static** **class** **UserThread** **implements** Runnable**{**

**public** **void** **run(){**

**new** Order**().**createOrder**();**

lock**.**lock**();**

Boolean result **=** **new** Stock**().**reduceStock**();**

lock**.**unlock**();**

**if(**result**){**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()** **+** "减库存成功"**);**

**new** Pay**().**pay**();**

**}else{**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()+**"减库存失败"**);**

**}**

**}**

**}**

**}**

使用ZooKeeper实现分布式锁，其实就是自己实现ZKLock()用来替代 new ReentrantLock();

**public** **class** **DistributeLock{**

**public** **static** Lock lock **=** **new** ZkLock**();**

**public** **static** **void** **main(**String**[]** args**)** **{**

**}**

**static** **class** **UserThread** **implements** Runnable**{**

**public** **void** **run(){**

**new** Order**().**createOrder**();**

lock**.**lock**();**

Boolean result **=** **new** Stock**().**reduceStock**();**

lock**.**unlock**();**

**if(**result**){**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()** **+** "减库存成功"**);**

**new** Pay**().**pay**();**

**}else{**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()+**"减库存失败"**);**

**}**

**}**

**}**

**}**

**lock()及trylock()加锁方法的实现**[](#locktrylock)

**unlock()方法的实现**[](#unlock)

**public** **class** **ZkLock** **implements** Lock**{**

**private** ThreadLocal**<**ZooKeeper**>** zk **=** **new** ThreadLocal**<**ZooKeeper**>();**

**private** String LOCK\_NAME **=** "/LOCK"**;**

**private** ThreadLocal**<**String**>** CURRENT\_NODE **=** **new** ThreadLocal**<**String**>();**

**public** **void** **lock(){**

init**();**

**if(**tryLock**){**

System**.**out**.**println**(**"拿到锁了"**);**

**}**

**}**

**private** **void** **init(){**

**if(**zk**.**get**()==null){**

zk**.**set**(new** Zookeeper**(**"localhost:2181"**,**3000**,new** Watcher**(){**

**public** **void** **process(**WatchedEvent watchedEvent**){**

System**.**out**.**println**(**"初始化ZooKeeper"**);**

**}**

**}));**

**}**

**}**

**private** **boolean** **tryLock(){**

String nodeName **=** LOCK\_NAME **+** "/zk\_"**;**

*//创建临时顺序节点 /LOCK/zk\_1*

String result **=** zk**.**get**().**create**(**nodeName**,new** **byte[**0**],** ZooDefs**.**Ids**.**OPEN\_ACL\_UNSAFE**,** CreateNode**.**EPHEMERAL\_SEQUENTIAL**);**

CURRENT\_NODE**.**set**(**result**);**

*//取出LOCAL\_NAME的所有子节点 //zk\_1, zk\_2,ZK\_3*

List**<**String**>** list **=** zk**.**get**().**getChildren**(**LOCK\_NAME**,false);**

Collections**.**sort**(**list**);**

*//取出第一个锁节点Znode*

String minNode **=** list**.**get**(**0**);**

**if((**LOCK\_NAME**+**"/"**+**minNode**).**equals**(**CURRENT\_NODE**.**get**())){**

**return** **true;**

**}else{**

*//等待锁*

*//watch*

*//找出CURRENT\_NODE的前一个节点，对其进行监听*

Integer currentIndex **=** list**.**indexOf**(**CURRENT\_NODE**.**get**().**substring**(**CURRENT\_NODE**.**get**().**lastIndexOf**(**"/"**)+**1**));**

*//前一个节点就是cucrentIndex-1*

String prevNodeName **=** list**.**get**(**cucrentIndex**-**1**);**

**final** CountDownLatch countDownLatch **=** **new** CountDownlatch**(**1**);**

zk**.**get**().**exists**(**LOCK\_NAME**+**"/"**+**prevNodeName**,new** Watcher**(){**

**public** **void** **process(**WatchedEvent watchedEvent**){**

**if(**Event**.**EventType**.**NodeDeleted**.**equals**(**watchedEvent**.**getType**())){**

countDownLatch**.**countDown**();**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()+**"唤醒锁了"**);**

**}**

**}**

**});**

System**.**out**.**println**(**Thread**.**currentThread**().**getName**()+**"等待锁"**);**

countDownLatch**.**await**();**

**}**

**return** **false;**

**}**

**private** **void** **unlock(){**

zk**.**get**().**delete**(**CURRENT\_NODE**.**get**(),-**1**);**

CURRENT\_NODE**.**remove**();**

zk**.**get**().**close**();**

**}**

**}**

**分布式配置中心**[](#_2)

分布式配置中心其实就是将配置存储与ZooKeeper的节点上，比如都存储在/CONFIG的子目录下，当节点出现删除或者修改之后或者创建新节点之后，都会通知客户端，然后客户端去Zookeeper服务端获取最新的值。

为了提高本地配置的效率，将从Zookeeper中的配置存储于Map中

**初始化配置init()方法**[](#init)

**保存配置的save()方法**[](#save)

**ZookeeperServer配置中心的配置修改后watcher()监听方法**[](#zookeeperserverwatcher)

**获取指定配置值的get()方法**[](#get)

**public** **class** **Config** **{**

**private** **static** **final** String CONFIG\_PREFIX **=** "/CONFIG"**;**

**private** CuratorFramework client**;**

**private** Map**<**String**,** String**>** cache **=** **new** HashMap**<>();**

**public** **Config(**String address**)** **{**

client **=** CuratorFrameworkFactory**.**newClient**(**address**,**

**new** RetryNTimes**(**3**,** 1000**));**

**;**

client**.**start**();**

init**();**

**}**

**public** **void** **init()** **{**

**try** **{**

List**<**String**>** childrenNames **=** client**.**getChildren**().**forPath**(**CONFIG\_PREFIX**);**

**for** **(**String name **:** childrenNames**)** **{**

String value **=** **new** String**(**client**.**getData**().**forPath**(**getConfigFullName**(**name**)));**

cache**.**put**(**name**,** value**);**

**}**

*// 监听*

PathChildrenCache watcher **=** **new** PathChildrenCache**(**client**,** CONFIG\_PREFIX**,** **true);**

watcher**.**getListenable**().**addListener**(new** PathChildrenCacheListener**()** **{**

@Override

**public** **void** **childEvent(**CuratorFramework client**,** PathChildrenCacheEvent event**)** **throws** Exception **{**

String path **=** event**.**getData**().**getPath**();**

System**.**out**.**println**(**event**);**

**if** **(**path**.**startsWith**(**CONFIG\_PREFIX**))** **{**

String key **=** path**.**replace**(**CONFIG\_PREFIX **+** "/"**,** ""**);**

**if** **(**PathChildrenCacheEvent**.**Type**.**CHILD\_ADDED**.**equals**(**event**.**getType**())** **||**

PathChildrenCacheEvent**.**Type**.**CHILD\_UPDATED**.**equals**(**event**.**getType**()))** **{**

String value **=** **new** String**(**event**.**getData**().**getData**());**

cache**.**put**(**key**,** value**);**

**}** **else** **if** **(**PathChildrenCacheEvent**.**Type**.**CHILD\_REMOVED**.**equals**(**event**.**getType**()))** **{**

cache**.**remove**(**key**);**

**}**

**}**

**}**

**});**

watcher**.**start**();**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

*// 新增或更新配置，一个配置项对应一个zookeeper节点，节点内容为配置项值*

**public** **void** **save(**String name**,** String value**)** **{**

**try** **{**

String configFullName **=** getConfigFullName**(**name**);**

Stat stat **=** client**.**checkExists**().**forPath**(**configFullName**);**

**if** **(**stat **!=** **null)** **{**

*// update*

client**.**setData**().**forPath**(**configFullName**,** value**.**getBytes**());**

**}** **else** **{**

*// create*

client**.**create**().**creatingParentsIfNeeded**().**withMode**(**CreateMode**.**PERSISTENT**).**forPath**(**configFullName**,** value**.**getBytes**());**

**}**

cache**.**put**(**name**,** value**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**private** String **getConfigFullName(**String name**)** **{**

**return** CONFIG\_PREFIX **+** "/" **+** name**;**

**}**

**public** String **get(**String name**)** **{**

**return** cache**.**get**(**name**);**

**}**

**}**

**public** **class** **Main** **{**

**public** **static** **void** **main(**String**[]** args**)** **throws** InterruptedException **{**

Config config **=** **new** Config**(**"localhost:2181"**);**

config**.**save**(**"timeout"**,** "1"**);**

**for** **(int** i**=**0**;** i**<**100**;** i**++)** **{**

System**.**out**.**println**(**"====="**+**config**.**get**(**"timeout"**));**

System**.**out**.**println**(**"====="**+**config**.**get**(**"grade"**));**

TimeUnit**.**SECONDS**.**sleep**(**5**);**

**}**

**}**

**}**