# 分布式利器Zookeeper（二）:分布式锁

张丰哲 (/u/cb569cce501b) 已关注
2017.05.12 07:49 字数 3162 阅读 1030 评论 4 喜欢 22 赞赏 1
(/u/cb569cce501b)

在《分布式利器Zookeeper（一）》(http://www.jianshu.com/p/3dfd63811e20)中对ZK进行了初步的介绍以及搭建ZK集群环境，本篇博客将涉及的话题是：基于原生API方式操作ZK，Watch机制，分布式锁思路探讨等。

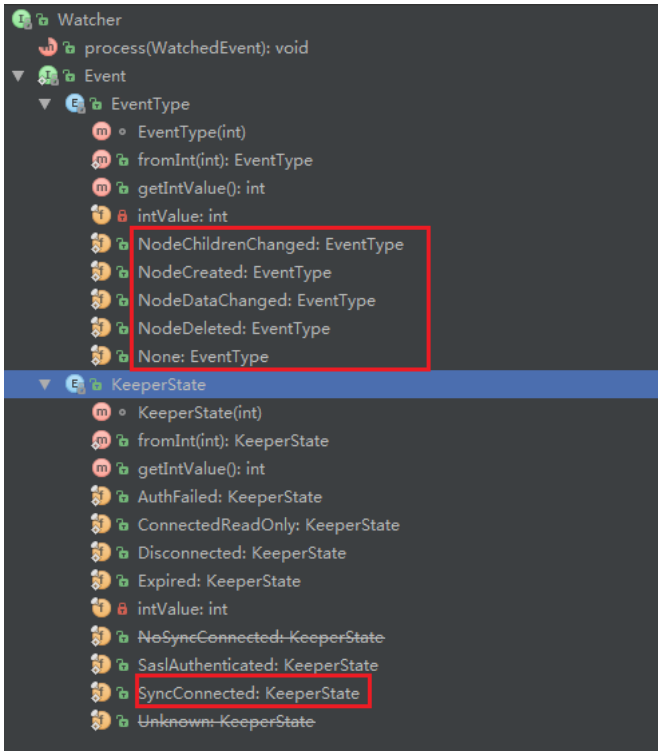
## 原生API操作ZK

> **什么叫原生API操作ZK呢？实际上，利用zookeeper.jar这样的就是基于原生的API方式操作ZK，因为这个原生API使用起来并不是让人很舒服，于是出现了zkclient这种方式，以至到后来基于Curator框架，让人使用ZK更加方便。有一句话，Guava is to JAVA what Curator is to Zookeeper。**

```
private static final String connectString = "192.168.99.121:2181,192.168.99.122:2181,192.168.99.123:2181";
private static final int sessionTimeout = 5000;
private static final CountDownLatch zkCountDownLatch = new CountDownLatch(1);
public static void main(String[] args) throws IOException, InterruptedException {
    ZooKeeper zooKeeper = new ZooKeeper(connectString, sessionTimeout, new Watcher() {
        @Override
        public void process(WatchedEvent event) {

            Event.KeeperState state = event.getState();
            Event.EventType type = event.getType();
            if(state == Event.KeeperState.SyncConnected && type == Event.EventType.None){

                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("connected zk server!");
                zkCountDownLatch.countDown();
            }


        }
    });
    zkCountDownLatch.await();
    System.out.println("run..." + zooKeeper);
    zooKeeper.close();
```

连接ZK的方式

> **说明：**
>
> **在初始化Zookeeper时，有多种构造方法可以选择，有3个参数是必备的：connectionString（多个ZK SERVER之间以,分隔），sessionTimeout（就是zoo.cfg中的tickTime），Watcher（事件处理通知器）。**
>
> **需要注意的是ZK的连接是异步的，因此我们需要CountDownLatch来帮助我们确保ZK初始化完成。**
>
> **对于事件（WatchedEvent）而言，有状态以及类型。**




事件状态及事件类型

下面，我们来看一看基于原生API方式的增删改查：

```
//创建节点 [要写全路径,ACL不必关注,持久化节点]
//重复创建会出现异常
//创建的路径的父路径都必须存在,否则异常
zooKeeper.create("/node/n2","n2".getBytes("UTF-8"), ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
zooKeeper.create("/node/n3","n3".getBytes("UTF-8"), ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
zooKeeper.create("/node/n3/child","child".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE,CreateMode.PERSISTENT);

//临时节点  连接断开,删除节点  临时节点是不允许有子节点的
zooKeeper.create("/node/n3/tmp","tmp".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE,CreateMode.EPHEMERAL);
//异常
zooKeeper.create("/node/n3/tmp/t1","t1".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE,CreateMode.EPHEMERAL);
```

create

> **注意，节点有2大类型，持久化节点、临时节点。在此基础上，又可以分为持久化顺序节点（PERSISTENT_SEQUENTIAL）、临时顺序节点（EPHEMERAL_SEQUENTIAL）。**
>
> **节点类型只支持byte[]，也就是说我们是无法直接给一个对象给ZK，让ZK帮助我们完成序列化操作的！**

```
//获取节点数据信息
byte[] data = zooKeeper.getData("/node/n1",false,null);
System.out.println(new String(data));

System.out.println("--------------------");
//getChildren并不会递归获取,而且获取到的是子路径,并不是全路径
List<String> children = zooKeeper.getChildren("/node", false);
for(String child : children){
    System.out.println(child);
}

//演示异步操作
//要知道删除只能是删除子节点
//原生API可没有rmr方法！
zooKeeper.delete("/node/n3", -1, new AsyncCallback.VoidCallback() {
    @Override
    public void processResult(int rc, String path, Object ctx) {
        System.out.println("rc : " + rc);
        System.out.println("path : " + path);
        System.out.println("ctx : " + ctx);
    }
},"info");
```

get/delete

> **这里需要注意的是，原生API对于ZK的操作其实是分为同步和异步2种方式的。**
>
> **rc表示return code，就是返回码，0即为正常。**
>
> **path是传入API的参数，ctx也是传入的参数。**
>
> **注意在删除过程中，是需要版本检查的，所以我们一般提供-1跳过版本检查机制。**

## Watch机制

> **ZK有watch事件，是一次性触发的。当watch监控的数据发生变化，会通知设置了该监控的client，即watcher。Zookeeper的watch是有自己的一些特性的：**
>
> **一次性：请牢记，just watch one time! 因为ZK的监控是一次性的，所以每次必须设置监控。**
>
> **轻量：WatchedEvent是ZK进行watch通知的最小单元，整个数据结构包含：事件状态、事件类型、节点路径。注意ZK只是通知client节点的数据发生了变化，而不会直接提供具体的数据内容。**
>
> **客户端串行执行机制：注意客户端watch回调的过程是一个串行同步的过程，这为我们保证了顺序，我们也应该意识到不能因一个watch的回调处理逻辑而影响了整个客户端的watch回调。**

下面我们来直接看代码：

```java
public class ZookeeperWatcher implements Watcher{

    //watch计数器
    private AtomicInteger atomicInteger = new AtomicInteger(0);
    private CountDownLatch countDownLatch = new CountDownLatch(1);
    private ZooKeeper zooKeeper = null;

    public ZooKeeper getZooKeeper() {
        return zooKeeper;
    }

    public ZookeeperWatcher(String connectionString , int sessionTimeout){

        try {
            zooKeeper = new ZooKeeper(connectionString,sessionTimeout,this);

            countDownLatch.await();

        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

提供Watcher的实现

```java
@Override
public void process(WatchedEvent event) {
    //对于WATCH事件而言，需要关注  事件类型/状态/PATH路径  这3个重点参数
    Event.EventType type = event.getType();
    Event.KeeperState state = event.getState();
    String path = event.getPath();
    System.out.println("Watch type : " + type.toString() + " , state : " + state.toString() + " , path : " + path);


    if(state == Event.KeeperState.SyncConnected){

        if(type == Event.EventType.None){
            countDownLatch.countDown();
            System.out.println("[Watch事件" + atomicInteger.incrementAndGet() + "]:ZK 建立连接" );
        }else if(type == Event.EventType.NodeCreated){
            System.out.println("[Watch事件" + atomicInteger.incrementAndGet() + "]:节点创建:" + path);
        }else if(type == Event.EventType.NodeDataChanged){
            System.out.println("[Watch事件" + atomicInteger.incrementAndGet() + "]:节点数据改变:" + path);
        }else if(type == Event.EventType.NodeChildrenChanged){
            System.out.println("[Watch事件" + atomicInteger.incrementAndGet() + "]:子节点改变:" + path);
        }else if(type == Event.EventType.NodeDeleted){
            System.out.println("[Watch事件" + atomicInteger.incrementAndGet() + "]:节点删除:" + path);
        }
    }
```

提供process方法

```java
ZookeeperWatcher zookeeperWatcher = new ZookeeperWatcher("192.168.99.121:2181,192.168.99.122:2181,192.168.99.123:2181",5000);
if(zookeeperWatcher.getZooKeeper() == null){
    System.out.println("ZK初始化失败!");
    return ;
}
//监控节点创建 true/false表示是否继续用上下文环境中的WATHCER进行监控  又或者NEW一个WATCHER进行监控
zookeeperWatcher.getZooKeeper().exists("/watcher",true);
System.out.println("MAIN create :  " + zookeeperWatcher.getZooKeeper().create("/watcher","watcher".getBytes(),
        ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT));
//watcher的一次性机制
zookeeperWatcher.getZooKeeper().delete("/watcher",-1);
zookeeperWatcher.getZooKeeper().create("/watcher","watcher".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);

System.out.println("-----------------1");
zookeeperWatcher.getZooKeeper().exists("/watcher",true);
zookeeperWatcher.getZooKeeper().delete("/watcher",-1);
System.out.println("-----------------2");
zookeeperWatcher.getZooKeeper().exists("/watcher",true);
zookeeperWatcher.getZooKeeper().create("/watcher","watcher".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
System.out.println("-----------------3");
zookeeperWatcher.getZooKeeper().exists("/watcher",true);
zookeeperWatcher.getZooKeeper().setData("/watcher","world".getBytes(),-1);
System.out.println("-----------------4");
zookeeperWatcher.getZooKeeper().exists("/watcher/child",true);
zookeeperWatcher.getZooKeeper().getChildren("/watcher",zookeeperWatcher);
zookeeperWatcher.getZooKeeper().create("/watcher/child","child".getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE,CreateMode.PERSISTENT);
```

main

> **一定得注意的是，监控该节点和监控该节点的子节点是2码子事。**
>
> **比如exists(path,true)监控的就是该path节点的create/delete/setData；getChildren(path,watcher)监控的就是该path节点下的子节点的变化（子节点的创建、修改、删除都会监控到，而且事件类型都是一样的，想一想如何区分呢？给一个我的思路，就是我们得先有该path下的子节点的列表，然后watch触发后，我们对比下该path下面的子节点SIZE大小及内容，就知道是增加的是哪个子节点，删除的是哪个子节点了！）**
>
> **getChildren(path,true)和getChildren(path,watcher)有什么区别？前者是沿用上下文中的Watcher，而后者则是可以设置一个新的Watcher的！（因此，要想做到一直监控，那么就有2种方式，一个是注意每次设置成true，或者干脆每次设置一个新的Watcher）**
>
> **从上面的讨论中，你大概能了解到原生的API其实功能上还不是很强大，有些还得我们去操心，到后面为大家介绍Curator框架，会有更好的方式进行处理。**

## 分布式锁思路

> **首先，我们不谈Zookeeper是如何帮助我们处理分布式锁的，而是先来想一想，什么是分布式锁？为什么需要分布式锁？有哪些场景呢？分布式锁的使用又有哪些注意的？分布式锁有什么特性呢？**
>
> **说起锁，我们自然想到Java为我们提供的synchronized/Lock，但是这显然不够，因为这只能针对一个JVM中的多个线程对共享资源的操作。那么对于多台机器，多个进程对同一类资源进行操作的话，就是所谓分布式场景下的锁。**
>
> **各个电商平台经常搞的“秒杀”活动需要对商品的库存进行保护、12306火车票也不能多卖，更不允许一张票被多个人买到、这样的场景就需要分布式锁对共享资源进行保护！**
>
> **既然，Java在分布式场景下的锁已经无能为力，那么我们只能借助其他东西了！**

### 我们的老朋友：DB

对，没错，我们能否借助DB来实现呢？要知道DB是有一些特点供我们利用的，比如DB本身就存在锁机制（表锁、行锁），唯一约束等等。

> **假设，我们的DB中有一张表T（id，methodname，ip，threadname，.....），其中id为主键，methodname为唯一索引。**
>
> **对于多台机器，每台机器上的多个线程而言，对一个方法method进行操作前，先select下T表中是否存在method这条记录，如果没有，就插入一条记录到T中。当然可能并发select，但是由于T表的唯一约束，使得只有一个请求能插入成功，即获得锁。至于释放锁，就是方法执行完毕后delete这条记录即可。**
>
> **考虑一些问题：如果DB挂了，怎么办？如果由于一些因素，导致delete没有执行成功，那么这条记录会导致该方法再也不能被访问！为什么要先select，为什么不直接insert呢？性能如何呢？**
>
> **为了避免单点，可以主备之间实现切换；为了避免死锁的产生，那么我们可以有一个定时任务，定期清理T表中的记录；先select后insert，其实是为了保证锁的可重入性，也就是说，如果一台IP上的某个线程获取了锁，那么它可以不用在释放锁的前提下，继续获得锁；性能上，如果大量的请求，将会对DB考验，这将成为瓶颈。**
>
> **到这里，还有一个明显的问题，需要我们考虑：上述的方案，虽然保证了只会有一个请求获得锁，但其他请求都获取锁失败返回了，而没有进行锁等待！当然，我们可以通过重试机制，来实现阻塞锁，不过数据库本身的锁机制可以帮助我们完成。别忘了select ... for update这种阻塞式的行锁机制，commit进行锁的释放。而且对于for update这种独占锁，如果长时间不提交释放，会一直占用DB连接，连接爆了，就跪了！**
>
> **不说了，老朋友也只能帮我们到这里了！**

### 我们的新朋友：Redis or 其他分布式缓存(Tair/...)

> **既然说是缓存，相较DB，有更好的性能；既然说是分布式，当然避免了单点问题；**
>
> **比如，用Redis作为分布式锁的setnx，这里我就不细说了，总之分布式缓存需要特别注意的是缓存的失效时间。（有效时间过短，搞不好业务还没有执行完毕，就释放锁了；有效时间过长，其他线程白白等待，浪费了时间，拖慢了系统处理速度）**

### 看Zookeeper是如何帮助我们实现分布式锁

> **Zookeeper中临时顺序节点的特性：**
>
> **第一，节点的生命周期和client回话绑定，即创建节点的客户端回话一旦失效，那么这个节点就会被删除。（临时性）**
>
> **第二，每个父节点都会维护子节点创建的先后顺序，自动为子节点分配一个整形数值，以后缀的形式自动追加到节点名称中，作为这个节点最终的节点名称。（顺序性）**

那么，基于临时顺序节点的特性，Zookeeper实现分布式锁的一般思路如下：

> **1.client调用create()方法创建“/root/lock_”节点，注意节点类型是EPHEMERAL_SEQUENTIAL**
>
> **2.client调用getChildren("/root/lock_",watch)来获取所有已经创建的子节点，并同时在这个节点上注册子节点变更通知的Watcher**
>
> **3.客户端获取到所有子节点Path后，如果发现自己在步骤1中创建的节点是所有节点中最小的，那么就认为这个客户端获得了锁**
>
> **4.如果在步骤3中，发现不是最小的，那么等待，直到下次子节点变更通知的时候，在进行子节点的获取，判断是否获取到锁**
>
> **5.释放锁也比较容易，就是删除自己创建的那个节点即可**

上面的这种思路，在集群规模很大的情况下，会出现“羊群效应”（Herd Effect）：

> **在上面的分布式锁的竞争中，有一个细节，就是在getChildren上注册了子节点变更通知Watcher，这有什么问题么？这其实会导致客户端大量重复的运行，而且绝大多数的运行结果都是判断自己并非是序号最小的节点，从而继续等待下一次通知，也就是很多客户端做了很多无用功。更加要命的是，在集群规模很大的情况下，这显然会对Server的性能造成影响，而且一旦同一个时间，多个客户端断开连接，服务器会向其余客户端发送大量的事件通知，这就是所谓的羊群效应！**

出现这个问题的根源，其实在于，上述的思路并没有找准客户端的“痛点”：

> **客户端的核心诉求在于判断自己是否是最小的节点，所以说每个节点的创建者其实不用关心所有的节点变更，它真正关心的应该是比自己序号小的那个节点是否存在！**
>
> **1.client调用create()方法创建“/root/lock_”节点，注意节点类型是EPHEMERAL_SEQUENTIAL**
>
> **2.client调用getChildren("/root/lock_",false)来获取所有已经创建的子节点，这里并不注册任何Watcher**
>
> **3.客户端获取到所有子节点Path后，如果发现自己在步骤1中创建的节点是所有节点中最小的，那么就认为这个客户端获得了锁**
>
> **4.如果在步骤3中，发现不是最小的，那么找到比自己小的那个节点，然后对其调用exist()方法注册事件监听**
>
> **5.之后一旦这个被关注的节点移除，客户端会收到相应的通知，这个时候客户端需要再次调用getChildren("/root/lock_",false)来确保自己是最小的节点，然后进入步骤3**

**OK,talk is cheap show me the code，下一篇文章会为大家带来Zookeeper实现分布式锁的代码。不早啦，上班去啦！**



张丰哲 (/u/cb569cce501b) ♂
写了 63893 字，被 2144 人关注，获得了 1674 个喜欢
(/u/cb569cce501b)

资深Java工程师 51CTO博客【2014-2016】：http://zhangfengzhe.blog.51cto.com/

好好学习，天天赞赏~

赞赏支持

♡ 喜欢 | 22

更多分享

(http://cwb.assets.jianshu.io/notes/images/121082

写下你的评论...

**4条评论** 只看作者 按喜欢排序 按时间正序 按时间倒序

程序员爸爸 (/u/2ee6a7de7013)
2楼 · 2017.05.12 09:19
(/u/2ee6a7de7013)
redis做分布式锁，个人理解，一次请求获取锁，用完马上释放锁delete，间隔时间很短的，小于失效时间

赞 回复

> 张丰哲 (/u/cb569cce501b)： 谢谢分享~
> 是这样的，为了保证锁释放的安全性，不能仅仅依靠delete，比如考虑一种情况，如果刚刚获取到锁，系统发生异常，重启或者宕机，那么意味着出现死锁。所以对于分布式缓存，设置过期时间，是非常必要的。其次，在考虑一种情况，设置了超时时间，但是客户端执行业务操作比平时多了些时间，导致锁提前过期释放，此时其他客户端获取到了锁，这样就导致多个客户端在同一段时间都获取到了锁，此外也需要考虑delete应该确保删除的是自己的锁，而不是其他客户端的。还有对于Redis集群而言，实质上存在master同步数据到slave上的过程，假设客户端在master上获取到了锁之后，master发生故障还没有来得及同步到slave上，那么意味着slave升级成master后，锁就消失了。所以对于分布式缓存来实现分布式锁，看似简单，其实要考虑的东西特别多。
>
> 
> 2017.05.14 20:59 回复
>
> 程序员爸爸 (/u/2ee6a7de7013)： @张丰哲 (/users/cb569cce501b) 厉害了
> 2017.05.14 22:00 回复
>
> Kevin_Zhan (/u/915f1433e136)： 好厉害啊
> 2017.08.13 21:29 回复
>
> 添加新评论

被以下专题收入，发现更多相似内容

+ 收入我的专题

Java学习笔记 (/c/04cb7410c597?utm_source=desktop&utm_medium=notes-included-collection)

java进阶干货 (/c/addfce4ca518?utm_source=desktop&utm_medium=notes-included-collection)

Java 杂谈 (/c/0b39448c4e08?utm_source=desktop&utm_medium=notes-included-collection)

程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)

分布式系列 (/c/da0570866697?utm_source=desktop&utm_medium=notes-included-collection)

Zookeeper (/c/057bb7fb1243?utm_source=desktop&utm_medium=notes-included-collection)

推荐阅读 更多精彩内容 > (/)

**玩转Redis集群(下) (/p/ee18dbd45d02?utm_campaig…** (/p/ee18dbd45d02?utm_campaign=maleskine&utm_content=note&utm

接上一篇《玩转Redis集群（上）》，我们来继续玩~ Redis集群操作实践 数据的分布性 从上面的操作，你可以看到，当存储某一个数据的时候，会分配一…

张丰哲 (/u/cb569cce501b?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

**写一个迷你版的Tomcat (/p/dce1ee01fb90?utm_cam…** (/p/dce1ee01fb90?utm_campaign=maleskine&utm_content=note&utm

前言 Tomcat，这只3脚猫，大学的时候就认识了，直到现在工作中，也常会和它打交道。这是一只神奇的猫，今天让我来抽象你，实现你！ Write MyTomc…

张丰哲 (/u/cb569cce501b?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

**手把手教你，如何用手机修图软件做出逼格超高的照片…** (/p/e1eb13a137aa?utm_campaign=maleskine&utm_content=note&utm

大家好哇，我是小丸子~ 国庆放假，很多同学都选择出去游玩，所以这篇文章我想给大家分享一些关于拍照修图的小技巧，让你用手机就能修出超高逼格的照…

小丸子的杂物集 (/u/24bca2bb387d?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

**后来，我再也不找你聊天了 (/p/f5d2e7f11fc3?utm_ca…** (/p/f5d2e7f11fc3?utm_campaign=maleskine&utm_content=note&utm

时间真的是个神奇的东西，它可以让原本两个陌生的人变得熟悉，然后，又让两个已经熟悉的人变得陌生。 01 我与阿泽相识于一场通话。 那时候我在一家公…

丁小谢 (/u/5c446bff04a7?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

**赵丽颖不配代言迪奥，环卫工不配送女儿出国留学，穷…** (/p/6f240d3248d6?utm_campaign=maleskine&utm_content=note&utm

最近，赵丽颖因为英语发音不够好而被嘲，认为她不配为迪奥代言。 我想这种新闻也只能发生在中国。假如我们找一个法国女星（比如说苏菲玛索）来代言…

罗衣一时聚散 (/u/2409fd4dfc9a?utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)