**zookeeper分布式锁的实现**

# 一、什么是分布式锁

## 1、单机情况下

在单进程的系统中，当存在多个线程同时改变某个共享变量时，就需要对变量或代码块做同步，我么可以使用synchronize或者Lock等锁住代码块，使多个线程在修改这种变量时能够顺序执行防止并发修改变量造成的数据不一致问题。

单机情况下线程之间共享内存，只要使用线程锁就可以解决并发问题。可以简单的采取内存作为锁标记存储位置，这个标记每个线程都能看到，当标记不存在时可以设置该标记，后续线程发现已经有标记了则等待拥有标记的线程结束同步代码块取消标记后再去尝试设置标记。

## 2、分布式情况下

目前几乎很多大型网站及应用都是分布式部署的，分布式与单机情况下最大的不同在于其不是多线程而是多进程。多线程由于可以共享堆内存，进程之间甚至可能都不在同一台物理机上，线程A和线程B很可能不是在同一JVM中，这样线程锁就无法起到作用了，因此需要将标记存储在一个所有进程都能看到的地方，比如数据库、redis、zk等，这时候就要用到分布式锁来解决。

# 二、分布式锁的需求

1、要保证加锁的方案同一时刻只能被一台机器上的一个线程访问。

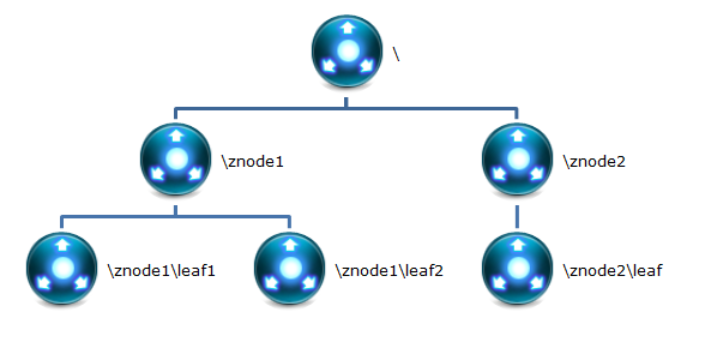
2、获取锁的客户端若因为宕机而未能释放锁，其它客户端也能获取到该锁。

3、锁只能被持有该锁的客户端删除，不能由其它客户端删除。

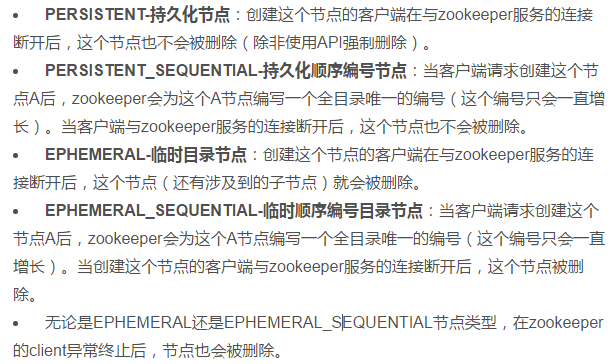
4、最好是阻塞锁，即让线程进入阻塞状态以等待获取锁，等待时不占用CPU 。

# 三、ZK基本介绍

## 1、zk节点特性



zookeeper中的数据节点（ZNode）是按照“树”结构进行存储的，类似于文件目录结构。znode是由客户端创建的，分为4中不同的类型：



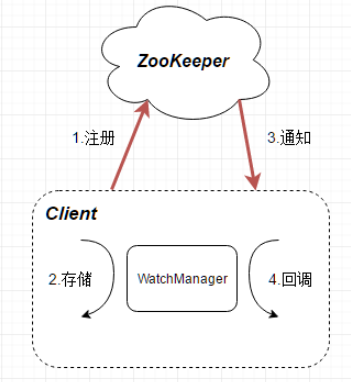
## 2、节点操作常用API

zookeeper不支持级联创建节点，需存在父节点再创建子节点；父节点为临时节点时不能为其创建子节点。



## 3、watcher机制

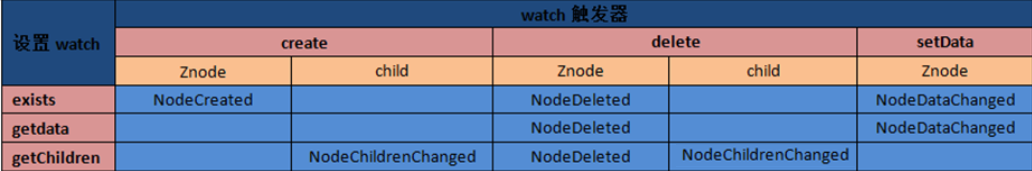
### （1）、watcher原理

ZooKeeper 的 Watcher 机制主要包括客户端进程、客户端 WatchManager 和 ZooKeeper 服务器三部分。 在具体流程上，客户端在向 ZooKeeper 服务器注册 Watcher 的同时，会将 Watcher 对象存储在客户端的 WatchManager 中。当ZooKeeper 服务器触发 Watcher 事件后，会向客户端发送通知，客户端线程从 WatchManager 的实现类中取出对应的 Watcher 对象来执行回调逻辑，统一在重载的process方法针对不同Watcher事件做不同处理。

### （2）、watcher事件

ZooKeeper可以为exists()、getChildren()及getData()这些读操作设置watcher，也就是这类函数除了**函数本身的功能之外，还能够注册watcher事件**。





### （3）、watcher特性

* 一次触发

当设置监视的数据发生改变时，该监视事件会被发送到客户端，例如，如果客户端调用了 getData("/znode1", true) 并且稍后 /znode1 节点上的数据发生了改变或者被删除了，客户端将会获取到 /znode1 发生变化的监视事件，而如果 /znode1 再一次发生了变化，除非客户端再次对 /znode1 设置监视，否则客户端不会收到事件通知。

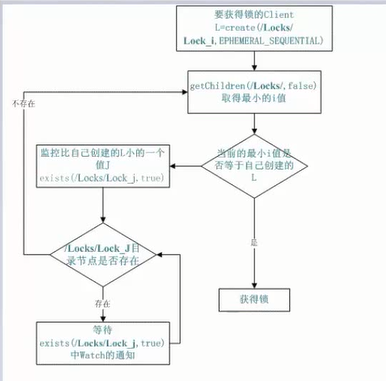
* 顺序执行

Zookeeper 客户端和服务端是通过 socket 进行通信，客户端 Watcher 回调的过程是一个串行同步的过程，为我们保证了顺序。

* 轻量级

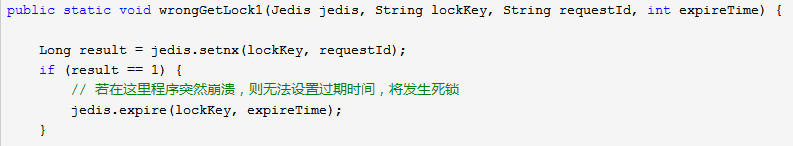
Watcher 通只会告诉客户端发生了事件，而不会说明事件的具体内容。例如针对 NodeDataChanged 事件，ZooKeeper 的Watcher 只会通知客户端指定数据节点的数据内容发生了变更，而对于原始数据以及变更后的新数据都无法从这个事件中直接获取到，而是需要客户端主动重新去获取数据。

# 四、zk分布式锁实现思路

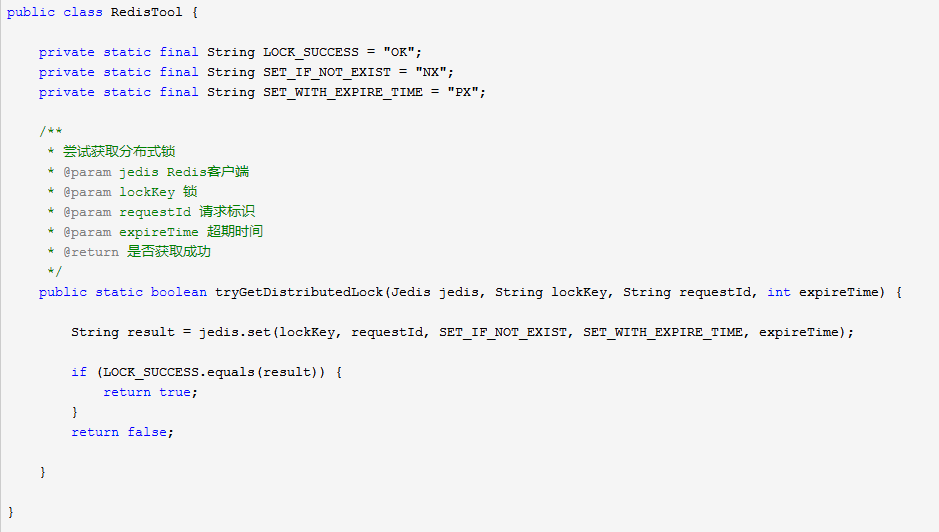


# 五、redis分布式锁

## 1、获得锁



setnx()方法作用就是SET IF NOT EXIST，expire()方法就是给锁加一个过期时间。锁必须要设置一个过期时间，否则当一个客户端获取锁成功之后，假如它崩溃了，或者由于发生了网络分割导致它再也无法和Redis节点通信了，那么它就会一直持有这个锁，而其它客户端永远无法获得锁了。这个过期时间称为锁的有效时间，获得锁的客户端必须在这个时间之内完成对共享资源的访问。然而由于这是两条Redis命令，不具有原子性，如果程序在执行完setnx()之后突然崩溃，导致锁没有设置过期时间。那么将会发生死锁。

从 Redis 2.6.12 版本开始，jedis的一个set命令就能原子的完成加锁和设置过期时间。

加锁就一行代码：jedis.set(String key, String value, String nxxx, String expx, int time)，这个set()方法一共有五个形参：

第一个为key，我们使用key来当锁，因为key是唯一的。

第二个为value，我们传的是requestId，通过给value赋值为唯一的requestId，我们就知道这把锁是哪个请求加的了，在解锁的时候就可以有依据。

第三个为nxxx，这个参数我们填的是NX，意思是SET IF NOT EXIST，即当key不存在时，我们进行set操作；若key已经存在，则不做任何操作；

第四个为expx，这个参数我们传的是PX，意思是我们要给这个key加一个过期的设置，具体时间由第五个参数决定。

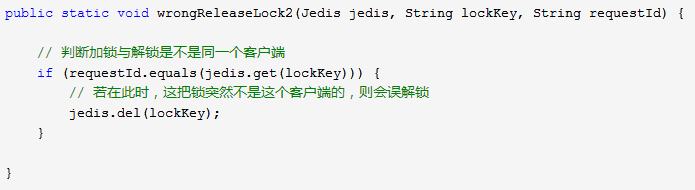
第五个为time，与第四个参数相呼应，代表key的过期时间。

首先，set()加入了NX参数，可以保证如果已有key存在，则函数不会调用成功，也就是只有一个客户端能持有锁，满足互斥性。

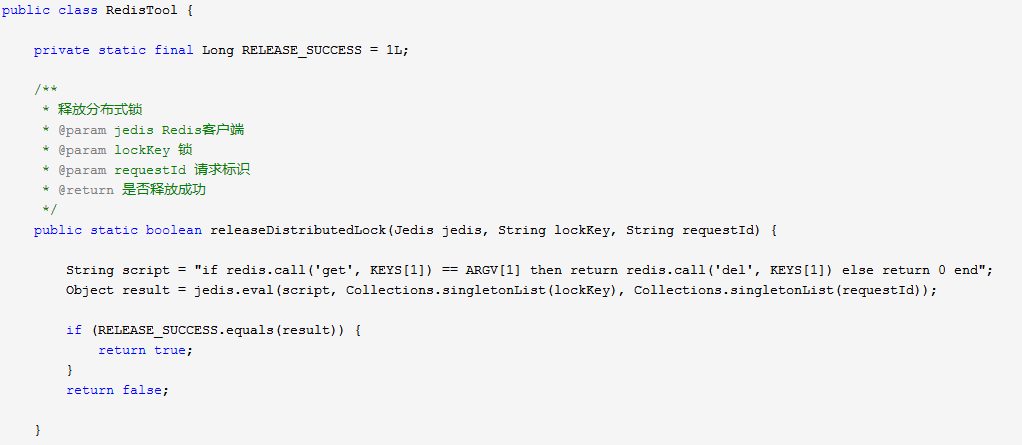
其次，由于我们对锁设置了过期时间，即使锁的持有者后续发生崩溃而没有解锁，锁也会因为到了过期时间而自动解锁（即key被删除），不会发生死锁。

最后，因为我们将value赋值为requestId，代表加锁的客户端请求标识，那么在客户端在解锁的时候就可以进行校验是否是同一个客户端。例如客户端A获取锁成功，在某个操作上阻塞了很长时间，过期时间到了，锁自动释放了；客户端B获取到了对应同一个资源的锁，客户端A从阻塞中恢复过来，释放掉了客户端B持有的锁，客户端B在访问共享资源的时候，就没有锁为它提供保护了

## 2、解锁



如果调用jedis.del()方法的时候，这把锁已经不属于当前客户端的时候会解除他人加的锁。比如客户端A加锁，一段时间之后客户端A解锁，在执行jedis.del()之前，锁突然过期了，此时客户端B尝试加锁成功，然后客户端A再执行del()方法，则将客户端B的锁给解除了。



一个Lua脚本代码，首先获取锁对应的value值，检查是否与requestId相等，如果相等则删除锁（解锁）。eval命令执行Lua代码的时候，Lua代码将被当成一个命令去执行，并且直到eval命令执行完成，Redis才会执行其他命令。

# 六、zk分布式锁优缺点

## 优点：

1、zookeeper的数据可以支持临时节点的概念，即客户端写入的数据是临时数据，在客户端宕机后，临时数据会被删除，这样就实现了锁的异常释放。使用这样的方式，就不需要给锁增加超时自动释放的特性了。而redis超时时间不好控制。

2、使用Zookeeper可以实现阻塞的锁，客户端可以通过在ZK中创建顺序节点，并且在节点上绑定监听器，一旦节点有变化，Zookeeper会通知客户端，客户端可以检查自己创建的节点是不是当前所有节点中序号最小的，如果是，那么自己就获取到锁，便可以执行业务逻辑了。Redis需要自己实现轮循。

## 缺点：

性能上可能并没有缓存服务那么高。因为每次在创建锁和释放锁的过程中，都要动态创建、销毁瞬时节点来实现锁功能。ZK中创建和删除节点只能通过Leader服务器来执行，然后将数据同不到所有的Follower机器上。