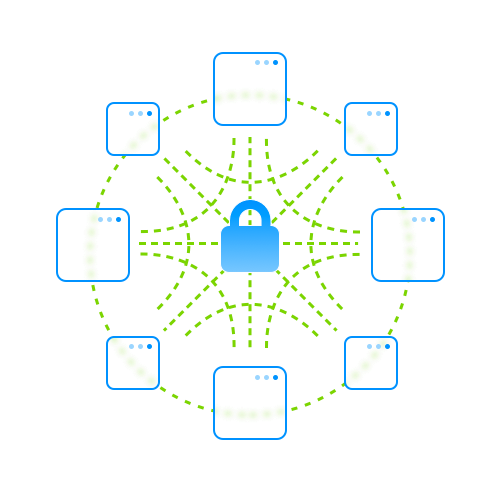
https://www.cnblogs.com/mushroom/p/4752499.html

[Redis分布式锁服务(八)](https://www.cnblogs.com/mushroom/p/4752499.html)



**阅读目录：**

1. [概述](https://www.cnblogs.com/mushroom/p/4752499.html#one)
2. [分布式锁](https://www.cnblogs.com/mushroom/p/4752499.html#two)
3. [多实例分布式锁](https://www.cnblogs.com/mushroom/p/4752499.html#three)
4. [总结](https://www.cnblogs.com/mushroom/p/4752499.html#four)

**概述**

在多线程环境下，通常会使用锁来保证有且只有一个线程来操作共享资源。比如：

object obj = new object();

lock (obj)

{

//操作共享资源

}

利用操作系统提供的锁机制，可以确保多线程或多进程下的并发唯一操作。但如果在多机环境下就不能满足了，当A，B两台机器同时操作C机器的共享资源时，就需要第三方的锁机制来保证在分布式环境下的资源协调，也称分布式锁。

Redis有三个最基本属性来保证分布式锁的有效实现：

* 安全性： 互斥，在任何时候，只有一个客户端能持有锁。
* 活跃性A：没有死锁，即使客户端在持有锁的时候崩溃，最后也会有其他客户端能获得锁，超时机制。
* 活跃性B：故障容忍，只有大多数Redis节点时存活的，客户端仍可以获得锁和释放锁。

**分布式锁**

由于Redis是单线程模型，命令操作原子性，所以利用这个特性可以很容易的实现分布式锁。 获得一个锁

SET key uuid NX PX timeout

SET resource\_name uniqueVal NX PX 30000

命令中的NX表示如果key不存在就添加，存在则直接返回。PX表示以毫秒为单位设置key的过期时间，这里是30000ms。 设置过期时间是防止获得锁的客户端突然崩溃掉或其他异常情况，导致redis中的对象锁一直无法释放，造成死锁。  
Key的值需要在所有请求锁服务的客户端中，确保是个唯一值。 这是为了保证拿到锁的客户端能安全释放锁，防止这个锁对象被其他客户端删除。  
举个例子：

1. A客户端拿到对象锁，但在因为一些原因被阻塞导致无法及时释放锁。
2. 因为过期时间已到，Redis中的锁对象被删除。
3. B客户端请求获取锁成功。
4. A客户端此时阻塞操作完成，删除key释放锁。
5. C客户端请求获取锁成功。
6. 这时B、C都拿到了锁，因此分布式锁失效。

要避免例子中的情况发生，就要保证key的值是唯一的，只有拿到锁的客户端才能进行删除。 基于这个原因，普通的del命令是不能满足要求的，我们需要一个能判断客户端传过来的value和锁对象的value是否一样的命令。遗憾的是Redis并没有这样的命令，但可以通过Lua脚本来完成：

if redis.call("get",KEYS[1]) == ARGV[1] then

return redis.call("del",KEYS[1])

else   
 return 0

end

逻辑很简单，获取key中的值和参数中的值相比较，相等删除，不相等返回0。

**多实例分布式锁**

上面是在单个Redis实例实现分布式锁的，这存在一个问题就是，如果这台实例因某些原因崩溃掉，那么所有客户端的锁服务全部失效。  
Redis本身支持Master-Slave结构，可以一主多从，采用高可用方法，可以保证在master挂的时候自动切换到slave。 但是由于主从之间是异步同步数据的，所以redis并不能完全的实现锁的安全性。 举个例子来说：

1. A客户端在master实例上获得一个锁。
2. 在对象锁key传送到slave之前，master崩溃掉。
3. 一个slave被选举成master。
4. B客户端可以获取到同个key的锁，但A也已经拿到锁，导致锁失效。

在多台master情况下实现这个算法，并保证锁的安全性。 步骤如下：

1. 客户端以毫秒为单位获取当前时间。
2. 使用同样key和值，循环在多个实例中获得锁。 为了获得锁，客户端应该设置个偏移时间，它小于锁自动释放时间(即key的过期时间)。 举个例子来说，如果一个锁自动释放时间是10秒，那偏移时间应该设置在5~50毫秒的范围。 防止因为某个实例崩溃掉或其他原因，导致client在获取锁时耗时过长。
3. 计算获取所有锁的耗时，即当前时间减去开始时间，得到a值。 用锁自动释放时间减去a值，在减去偏移时间，得到c值，如果获取锁成功的实例数量大于实际的数量一半，并且c大于0，那么锁就被获取成功。
4. 锁获取成功，锁对象的有效时间是上面的c值。
5. 若是客户端因为一些原因获取失败，原因可能是上面的c值为负数或者锁成功的数量小于实例数，以用N/2+1当标准(N为实例数)。 那么会释放所有实例上的锁。

上面描述可能不方便理解，用代码表示如下：

[复制代码](javascript:void(0);)

//锁自动释放时间

TimeSpan ttl=new TimeSpan(0,0,0,30000)

//获取锁成功的数量

int n = 0;

//记录开始时间

var startTime = DateTime.Now;

//在每个实例上获取锁

for\_each\_redis(

redis =>

{

if (LockInstance(redis, resource, val, ttl)) n += 1;

}

);

//偏移时间是锁自动释放时间的1%，根据上面10s是5-50毫秒推出。

var drift = Convert.ToInt32(ttl.TotalMilliseconds \* 0.01);

//锁对象的有效时间=锁自动释放时间-(当前时间-开始时间)-偏移时间

var validity\_time = ttl - (DateTime.Now - startTime) - new TimeSpan(0, 0, 0, 0, drift);

//判断成功的数量和有效时间c值是否大于0 if (n >= (N/2+1) && validity\_time.TotalMilliseconds > 0) { }

[复制代码](javascript:void(0);)

**总结**

用Redis做分布式锁相比其他分布式锁(zookeeper)实现更简单，速度更快。  
在ServiceStack.Redis客户端组件上是直接支持锁实现的。  
或者用stackexchange客户端组件，锁实现及示例代码：https://github.com/kidfashion/redlock-cs。  
官方介绍文档：http://redis.io/topics/distlock。