ZooKeeper实现分布式锁

|  |  |  |  |
| --- | --- | --- | --- |
| 编写人 | 编写时间 | 版本 | 备注 |
| 张贵东 | 2017年8月 | V1.0 |  |

目 录

[1. 介绍 2](#_Toc490596844)

[1.1. 介绍 2](#_Toc490596845)

[1.2. 一般实现分布式锁的方式有以下几种 2](#_Toc490596846)

[1.2.1. 数据库方式 2](#_Toc490596847)

[1.2.2. redis实现 3](#_Toc490596848)

[1.2.3. ZooKeeper实现 4](#_Toc490596849)

1. 介绍
   1. 介绍

在分布式应用, 往往存在多个进程提供同一服务. 这些进程有可能在相同的机器上, 也有可能分布在不同的机器上. 如果这些进程共享了一些资源, 可能就需要分布式锁来锁定对这些资源的访问.

* 1. 一般实现分布式锁的方式有以下几种

数据库方式

（1）数据库：利用表索引的唯一性来实现

创建一个表， 通过索引唯一的方式

create table (id , methodname …) methodname增加唯一索引

insert 一条数据XXX delete 语句删除这条记录

mysql for update

首先性能不是特别高。

通过数据库的锁来实现多进程之间的互斥，但是这貌似也有一个问题：就是sql超时异常的问题

jdbc超时具体有3种超时，具体见深入理解JDBC的超时设置

框架层的事务超时

jdbc的查询超时

Socket的读超时

这里只涉及到后2种的超时，jdbc的查询超时还好（mysql的jdbc驱动会向服务器发送kill query命令来取消查询），如果一旦出现Socket的读超时，对于如果是同步通信的Socket连接来说(底层实现Connection的可能是同步通信也可能是异步通信)，该连接基本上不能使用了，需要关闭该连接，从新换用新的连接，因为会出现请求和响应错乱的情况，比如jedis出现的类型转换异常，详见Jedis的类型转换异常深究

redis实现

（1）redis：setNX 、存在则会返回0， 不存在

setnx来创建一个key，如果key不存在则创建成功返回1，如果key已经存在则返回0。依照上述来判定是否获取到了锁

获取到锁的执行业务逻辑，完毕后删除lock\_key，来实现释放锁

其他未获取到锁的则进行不断重试，直到自己获取到了锁

改进方案一：

上述逻辑在正常情况下是OK的，但是一旦获取到锁的客户端挂了，没有执行上述释放锁的操作，则其他客户端就无法获取到锁了，所以在这种情况下有2种方式来解决：

为lock\_key设置一个过期时间

对lock\_key的value进行判断是否过期

改进方案二：

问题1： lock timeout的存在也使得失去了锁的意义，即存在并发的现象。一旦出现锁的租约时间，就意味着获取到锁的客户端必须在租约之内执行完毕业务逻辑，一旦业务逻辑执行时间过长，租约到期，就会引发并发问题。所以有lock timeout的可靠性并不是那么的高。

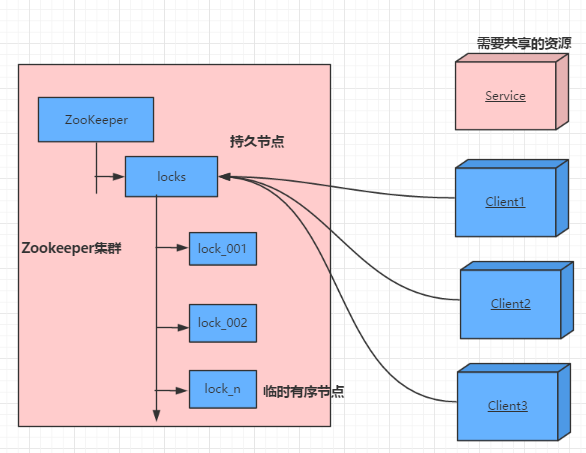
问题2： 上述方式仅仅是redis单机情况下，还存在redis单点故障的问题。如果为了解决单点故障而使用redis的sentinel或者cluster方案，则更加复杂，引入的问题更多。

ZooKeeper实现

进程需要访问共享数据时, 就在"/lock"节点下创建一个sequence类型的子节点, 称为thisPath. 当thisPath在所有子节点中最小时, 说明该进程获得了锁. 进程获得锁之后, 就可以访问共享资源了. 访问完成后, 需要将thisPath删除. 锁由新的最小的子节点获得.

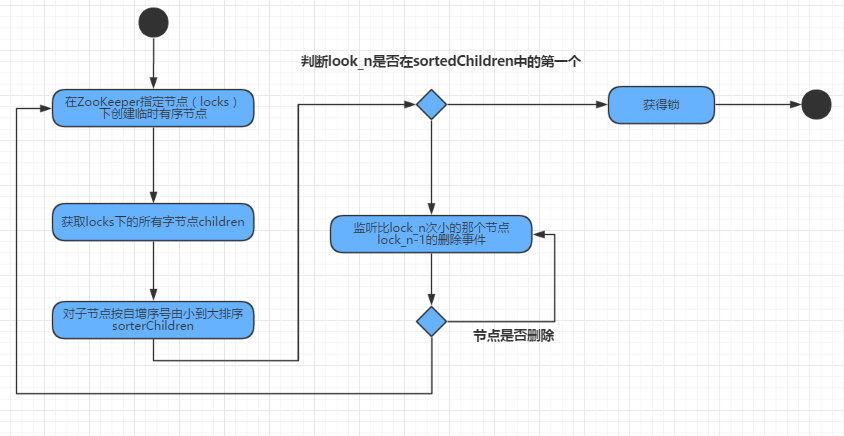
有了清晰的思路之后, 还需要补充一些细节. 进程如何知道thisPath是所有子节点中最小的呢? 可以在创建的时候, 通过getChildren方法获取子节点列表, 然后在列表中找到排名比thisPath前1位的节点, 称为waitPath, 然后在waitPath上注册监听, 当waitPath被删除后, 进程获得通知, 此时说明该进程获得了锁.

原理图如下：



ZooKeeper分布式锁原理图

算法流程图如下：



算法流程图

代码实现见github地址：<https://git.oschina.net/BJZGD/zookeeper-demo>