LUA脚本改造redis分布式锁

原创 机跃 于 2024-07-29 13:42:38 发布 阅读量913 收藏 9 点赞数 7

文章标签： redis 分布式

在redis集群模式下，我们会启动多个tomcat实例，每个tomcat实例都有一个JVM，且不共享。而synchronize锁的作用范围仅仅是当前JVM，所以我们需要
群下的锁，也就是 分布式 锁。（就是不能用JVM自带的锁了，需要一个第三方应用实现锁）

在redis集群中我们是用setnx实现互斥锁来实现分布式锁。

setnx key value NX EX 时间 是往redis中创建一个key-value键值对，如果redis中没有该key，则创建成功，返回true；如果redis已经有了该key，则创建
null。NX是互斥，EX是超时时间，后跟具体时间+单位（s），EX用于过期时间，过了过期时间自动删除该key-value键值对。

我们用setnx命令实现分布式锁时，key和value都是String类型，key一般是特定前缀+该锁的名字，

value为线程id。

为什么value要存线程id呢?

因为存在这样情况：线程一获取锁，去执行逻辑，过程中遇到网络动荡，线程一卡住了，然后一段时间后，线程一获取的锁的过期时间到了，线程一的锁
后线程二来获取锁，线程二获取成功，去执行逻辑，而在这个过程中线程一的网络动荡恢复了，线程一继续执行，线程一先于线程二执行完，线程一去释
线程一创建的锁已经因超时而自动释放了，所以此时线程一会去错误的释放线程二的锁，所以我们要把线程id存入加以判断，防止误删其他线程的id。

代码实现获取锁和释放锁：

```
package com.hmdp.utils;

import cn.hutool.core.lang.UUID;
import org.springframework.core.io.ClassPathResource;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.core.script.DefaultRedisScript;
import org.springframework.data.redis.core.script.RedisScript;

import java.util.Collections;
import java.util.concurrent.TimeUnit;

public class SimpleRedisLock implements ILock {

    private String name;
    private StringRedisTemplate stringRedisTemplate;

    public SimpleRedisLock(String name, StringRedisTemplate stringRedisTemplate) {
        this.name = name;
        this.stringRedisTemplate = stringRedisTemplate;
    }

    private static final String KEY_PREFIX = "lock:";

    //randomUUID生成的数字会带一个横线，toString（true）方法就是去掉横线
    //因为不同JVM中，可能存在线程号相同的情况，所以需要用UUID来区分不同的JVM
    private static final String ID_PREFIX = UUID.randomUUID().toString(true) + "-";

    @Override
    public boolean tryLock(long timeoutSec) {
        // 获取线程标示
        String threadId = ID_PREFIX + Thread.currentThread().getId();
        // 获取锁
        Boolean success = stringRedisTemplate.opsForValue()
                .setIfAbsent(KEY_PREFIX + name, threadId, timeoutSec, TimeUnit.SECONDS);
        //不能直接返回success，因为会有自动拆箱的风险，如果success是null，就会返回true，返回错误数据
        return Boolean.TRUE.equals(success);
    }

    @Override
    public void unlock() {
        // 获取线程标示
        String threadId = ID_PREFIX + Thread.currentThread().getId();
        // 获取锁中的标示
        String id = stringRedisTemplate.opsForValue().get(KEY_PREFIX + name);
        // 判断标示是否一致
```

```
        if(threadId.equals(id)) {
                                47              // 释放锁
            stringRedisTemplate.delete(KEY_PREFIX + name);
        }
    }
}
```

我们需要新建一个类实现ILock接口，然后去重写其中的tryLock获取锁、unlock释放锁方法。

往SimpleRedisLock的构造方法传入name变量，方便我们在实例化SimpleRedisLock类中设置该锁的名字。我们后面用setnx命令创建锁时的key值就是特
KEY_PREFIX加上这个name；value值是randomUUID生成的唯一一串数字加上该线程id，因为不同JVM中，可能存在线程号相同的情况，所以需要用UU
的JVM。

问题：

当本线程1在进入这个if判断后（释放锁之前），突然阻塞（比如full GC，该JVM上全服务堵塞）阻塞时间过长，锁超时释放，这时另一个jvm的线程2获取
程1继续执行释放锁
这样就又会出现同一个用户会有俩个线程在同时运行,所以需要保证判断和释放锁这俩步为一个原子性，同成功或同失败
这样很容易想到事务，redis也有事务，但redis的事务可以保证原子性，但不能保证一致性，而且redis的事务，是最后事务中的步骤同时完成。并不是一步
所以只能用 乐观锁 但不建议用乐观锁，推荐用lua脚本

一个lua脚本中可以编写多条redis命令，确保多条命令的原子性。即实现判断和释放锁一起执行的原子性。

我们需要把这个脚本写在resources目录下

释放锁的lua脚本：

```
local key=KEYS[1]  -- 锁的key
local threadId=ARGV[1]  --线程唯一标识
-- 比较线程标示与锁中的标示是否一致
if(redis.call('get', key) ==  threadId) then
    -- 释放锁 del key
    return redis.call('del', key)
end
return 0;
```

if() then end 相当于Java命令中的 if( ){ } 。

KEYS[1]:为需要传入的key值，当需要传入多个key值时，声明KEYS[2]、KEYS[3]...等就行。

ARGV[1]:为需要传入的其他非key的变量，声明方法与key一样。例如：

```
local key= KEYS[1]
local key2= KEYS[2]
local threadId=ARGV[1]
local releaseTime=ARGV[2]
```

redis.call(' ', ...)是执行的redis命令，命令中单引号' '中写要执行的redis操作，后面的参数为执行该redis命令所需的参数，例如，get命令，需要知道ke

然后改写我们的分布式锁：

我们需要先加载我们的lua脚本：

```
private static final DefaultRedisScript<Long> UNLOCK_SCRIPT;
    static {
        UNLOCK_SCRIPT = new DefaultRedisScript<>();
        UNLOCK_SCRIPT.setLocation(new ClassPathResource("unlock.lua")); //加载的脚本的位置，如果脚本文件在resources目录下，则只需写脚本名
        UNLOCK_SCRIPT.setResultType(Long.class); //返回值的类型
    }
@Override
    public void unlock() {
        // 调用Lua脚本  ,原来的多行代码变成了现在的单行代码就保证了原子性
        stringRedisTemplate.execute(
                UNLOCK_SCRIPT,
                Collections.singletonList(KEY_PREFIX + name), //生成单元素的集合，即脚本
                ID_PREFIX + Thread.currentThread().getId()); //即脚本中需要的ARVG[1]参数
```

```
    }
```

完整代码：

```
package com.hmdp.utils;

import cn.hutool.core.lang.UUID;
import org.springframework.core.io.ClassPathResource;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.core.script.DefaultRedisScript;
import org.springframework.data.redis.core.script.RedisScript;

import java.util.Collections;
import java.util.concurrent.TimeUnit;

public class SimpleRedisLock implements ILock {

    private String name;
    private StringRedisTemplate stringRedisTemplate;

    public SimpleRedisLock(String name, StringRedisTemplate stringRedisTemplate) {
        this.name = name;
        this.stringRedisTemplate = stringRedisTemplate;
    }

    private static final String KEY_PREFIX = "lock:";

    //randomUUID生成的数字会带一个横线，toString（true）方法就是去掉横线
    //因为不同JVM中，可能存在线程号相同的情况，所以需要用UUID来区分不同的JVM
    private static final String ID_PREFIX = UUID.randomUUID().toString(true) + "-";

    private static final DefaultRedisScript<Long> UNLOCK_SCRIPT;
    static {
        UNLOCK_SCRIPT = new DefaultRedisScript<>();
        UNLOCK_SCRIPT.setLocation(new ClassPathResource("unlock.lua")); //加载的脚本的位置
        UNLOCK_SCRIPT.setResultType(Long.class); //返回值的类型
    }

    @Override
    public boolean tryLock(long timeoutSec) {
        // 获取线程标示
        String threadId = ID_PREFIX + Thread.currentThread().getId();
        // 获取锁
        Boolean success = stringRedisTemplate.opsForValue()
                .setIfAbsent(KEY_PREFIX + name, threadId, timeoutSec, TimeUnit.SECONDS);
        //不能直接返回success，因为会有自动拆箱的风险，如果success是null，就会返回true，返回错误数据
        return Boolean.TRUE.equals(success);
    }

    @Override
    public void unlock() {
        // 调用lua脚本  ,原来的多行代码变成了现在的单行代码就保证了原子性
        stringRedisTemplate.execute(
                UNLOCK_SCRIPT,
                Collections.singletonList(KEY_PREFIX + name), //生成单元素的集合，即脚本中的需要的KETS[1]参数
                ID_PREFIX + Thread.currentThread().getId()); //即脚本中需要的ARVG[1]参数
    }

}
```


使用redis+lua脚本实现分布式锁

分布式锁的应用场景 当多个机器（多个进程）会对同一条数据进行修改时，并且要求这个修改是原子性的。这里有[

redis实现分布式锁,自旋式加锁,lua原子性解锁
本文将深入探讨如何使用Redis实现分布式锁，以及如何利用自旋式加锁和Lua脚本实现原子性解锁。 首先，我们来理解分布式锁的基本概念。分布式锁是在多节点之间共享资

spring整合的redis分布式锁_spring-integration-redis
获取锁的本质就是执行一段lua脚本加锁,首先锁key是在构造RedisLockRegistry.RedisLock生成的RedisLockRegistry.this.registryKey + ":" + path;,path就是我们要操作的redis中

spring cloud 实现分布式锁--黑马程序员笔记_springcloud分布式锁-CSDN...
zookeeper的结构类似的文件目录,多线程向zookeeper创建一个子目录(节点)只会有一个创建成功,利用此特点可以实现分布式锁,谁创建该结点成功谁就获得锁。 3.3 基于redis

Redis分布式锁--java实现 A
在单体redis中通过SETNX + EXPIRE方式可以为多个JVM加一个分布式锁，但是由于操作的非原子性会导致并发问题，因此出现了几种原子性解决方法，包括SETNX+时间va

java中利用Redis整合Lua脚本 最新发布 m0_7359890
Lua 编程语言编写的脚本程序。Lua 是一种轻量级的脚本语言，由巴西里约热内卢天主教大学的 Luiz Henrique de Figueiredo 教授及其团队于 1993 年创建。它被设计为一种

...和 原子性问题_springcloud reids 分布式锁原子性
Redis提供了Lua脚本功能,在一个脚本中编写多条Redis命令,确保多条命令执行时的原子性。Lua是一种编程语言,它的基本语法大家可以参考网站:https://www.runoob.com/lua/

redis 分布式锁原理解析及使用教程详细_redis分布式锁
packagecom.crazymaker.springcloud.standard.lock;@Slf4j@Data@AllArgsConstructorpublicclassJedisCommandLock{privateRedisTemplateredisTemplate;privatestaticfinalS

redis分布式锁-lua脚本 王洪
package com.tuling.jedis; import redis.clients.jedis.Jedis; import redis.clients.jedis.JedisPool; import redis.clients.jedis.JedisPoolConfig; import java.io.IOException; import jav

分布式锁2：基于redis实现分布式锁 setnx+lua脚本 健康平安的活
redis中setnx+expire是非原子操作, 除了用LUA脚本保证实现原子操作, 其实可以直接使用redis自带的set方法直接实现.setnx+expire+del 命令实现redis的分布式锁；其中 setn

redis+lua脚本 分布式锁初步学习_lua脚本怎么连接redis带密码-CSDN博...
cat springcloud/lua/redis-lua/test.lua | redis-cli -a 123456 script load --pipe script load缓存lua脚本 并将返回SHA1校验和 复制到java中evalsha参数里 /** * @Description: v2.2

Spring Cloud:基于Redisson的分布式锁实现_springcloud redisson-CSDN...
将业务逻辑进行封装在lua脚本中,然后发送给redis,而且由于redis是单线程的,从而保证业务逻辑执行的原子性。 整合Spring Boot 为了在项目中可以自由引用,我创建了一个独立

Redis 实现分布式锁+执行lua脚本 qq_3428555
本篇主要一步步演进手写redis分布式锁的实现, 包括 setnx -> set -> 过期时间 -> 误删锁 -> uuid控制锁误删-> lua脚本控制删锁的原子性等等.. 其实目前还有问题, 包括锁续期问

Redis实现分布式锁 莫等闲 白了少年头
我们一路走来，利用添加过期时间，防止死锁问题的发生，但是有了过期时间之后，可能出现误删别人锁的问题，这个问题我们开始是利用删之前 通过拿锁，比锁，删锁这个

redisson分布式锁底层原理_redisson 底层是lua脚本
redisson分布式锁底层原理 redisson底层是lua脚本+自动续期,并且支持可重入锁 1.在加锁的时候,判断锁是否存在 2.如果锁不存在,那么插入key,并设置锁的计数器为1,设置过期

分布式锁实现方案(二):基于Redis+Lua脚本的分布式锁
我们在 《分布式锁实现方案(一):基于数据库实现分布式锁》 中曾经说过,xxxx 我们知道,Redis 保证以一种原子性的方式来执行 Lau 脚本,当 Lua脚本执行的时候,不会有其他的

基于redis和lua脚本的分布式锁的实现
“基于Redis和Lua脚本的分布式锁的实现” 基于Redis和Lua脚本的分布式锁的实现是使用Redis和Lua脚本来实现分布式锁的技术。分布式锁是指在分布式系统中，多个节点之间

redis分布式锁工具类
现在很多项目单机版已经不满足了，分布式变得越受欢迎，同时也带来很多问题，分布式锁也变得没那么容易实现，分享一个redis分布式锁工具类，里面的加锁采用lua脚本

使用Lua基本实现分布式锁并自动续期_resttemplate lua 实现分布式锁...
基于RedisTemplate的redis分布式锁的LUA实现 top_explore的博客 433 import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annot

分布式锁-基于Redis实现_innerthreadlocal
1.客户端获取锁,通过setnx(lockkey,currenttime+timeout)命令,将key为lockkey的value设置为当前时间+锁超时时间 2.如果setnx(lockkey,currenttime+timeout)设置后返回值为1

redis分布式锁.zip
Redis 分布式锁是分布式系统中解决并发控制和数据一致性问题的一种常见机制。在大型分布式应用中，单机锁无法满足需求，因为它们局限于单个服务器。Redis 的高可用性

Redis分布式锁实现方式及超时问题解决
解决思路是使用 Lua脚本实现多条指令的原子性执行，并对释放锁进行身份校验，使用随机数作为value的唯一标识。 七、结论 本文详细介绍了分布式锁的实现方式和超时问题

利用Java代码调用Lua脚本改造分布式锁 fishniu
笔者总结：我们一路走来，利用添加过期时间，防止死锁问题的发生，但是有了过期时间之后，可能出现误删别人锁的问题，这个问题我们开始是利用删之前 通过拿锁，比锁

Redis分布式锁—SETNX+Lua脚本实现 weixin_448669
使用redis实现分布式锁，就是利用redis中的setnx，如果key不存在则进行set操作返回1，key已经存在则直接返回0。

使用 Redis 和 Lua 实现分布式锁
分布式锁是一种用于多台服务器上处理同一资源的并发访问的锁机制。它用于协调分布式系统中的各个节点，确保它们的操作不会相互干扰。Redis （Remote Dictionary Serv

Redis系列学习文章分享---第六篇（Redis实战篇--Redis分布式锁+实现思路+误删问题+原子性+lua脚本+Redisson功能介绍+… 热门推荐 weixin_4497669
Redisson是一个基于Redis的Java驻内存数据网格（In-Memory Data Grid）和分布式锁服务的框架，提供了丰富的分布式对象和服务支持。它封装了Redis的分布式对象和服务

lua脚本实现redis分布式锁(脚本解析) 怀着一颗
Lua 是一种轻量小巧的脚本语言，用标准C语言编写并以源代码形式开放， 其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。设计目的 其设计

使用lua实现redis分布式锁 C182981825
Lua官网 http://doc.redisfans.com/script/eval.html 问题重现：预约功能，用jemter测试，30个不同用户同时预约，2个用户重复发送请求，即重复发送消息场景，结果导致重复

Redis_配合Lua做Java分布式事务锁 pseudony
Redis作Java分布式锁 我们都知道redis现在成为越来越多人作为缓存工具的选择，redis在性能、操作等方面上有着独特的优势。 1. 检查reids版本 因为redis是在2.6版本

redis分布式锁+lua脚本代码实现
下面是一个使用Lua脚本实现Redis分布式锁的代码示例： ```lua -- Lua脚本实现Redis分布式锁 local lockKey = 'lock' local uuid = ARGV…

机跃
码龄1年 暂无认证

83	106万+	2万+	3万+	
原创	周排名	总排名	访问	等级

1421	436	585	23	542
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

热门文章

最新评论

JVM之运行时数据区
普通网友: 学到了，细节很到位！【我也写了一些相关领域的文章，希望能够得到 ...

JVM之运行时数据区
CSDN-Ada助手: 推荐 Java 技能树：https://edu.csdn.net/skill/java?utm_source= ...

机跃 关注

Java虚拟机（JVM）之字节码文件
天`南: 写的真不错，受益匪浅

springboot结合elasticJob
普通网友: 优质好文，细节很到位！【我也写了一些相关领域的文章，希望能够得到 ...

springboot结合elasticJob
全栈小5: 干货很多，文章内容实用性很好，技术点讲解的很到位。期待大佬的持续更 ...

最新文章

Seata解决分布式事务

Java代码实现elasticSearch的DSL复合查询

elasticSearch的索引库文档的增删改查

2024

07月	06月	05月	04月
5篇	6篇	5篇	2篇
03月	02月	01月	
7篇	7篇	9篇	

2023年 42篇