Ruthless
J2EE+ANDROID+jQuery交流群：158560018

昵称：Ruthless
园龄：7年2个月
粉丝：3002
关注：29
+加关注

< 2018年4月 >

日	一	二	三	四	五	六
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

搜索

找找看

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

java 多线程(22)
redis(19)
JAVA(13)
android(12)
docker(11)
nginx(8)
jQuery(7)
OSCache(7)
oracle系统包(5)
jpa(5)
更多

随笔分类

ActiveMQ(7)
android菜单与对话框(8)
android常用控件(16)
android高级应用(51)
android基础知识(17)
APP(8)
Backbone(5)
Bootstrap 教程(7)
css(3)
django(14)
Docker系列教程(19)
dubbo(7)
ext(9)
freemarker(3)
ftp(1)
Hadoop(8)
hadoop源码分析(1)
Hbase(4)
HDFS(1)
hibernate(2)
Hive(3)
html5(3)
ibatis(2)
j2ee案例(72)
javascript(14)
Java多线程编程(40)
java高级特性(44)
Java后端架构(11)

博客园 首页 新随笔 联系 订阅 管理

随笔-894 评论-813 文章-0

Redis分布式锁的正确实现方式

前言

分布式锁一般有三种实现方式：1. 数据库乐观锁；2. 基于Redis的分布式锁；3. 基于ZooKeeper的分布式锁。本篇博客将介绍第二种方式，基于Redis实现分布式锁。虽然网上已经有各种介绍Redis分布式锁实现的博客，然而他们的实现却有着各种各样的问题，为了避免误人子弟，本篇博客将详细介绍如何正确地实现Redis分布式锁。

可靠性

首先，为了确保分布式锁可用，我们至少要确保锁的实现同时满足以下四个条件：

1. 互斥性。在任意时刻，只有一个客户端能持有锁。
2. 不会发生死锁。即使有一个客户端在持有锁的期间崩溃而没有主动解锁，也能保证后续其他客户端能加锁。
3. 具有容错性。只要大部分的Redis节点正常运行，客户端就可以加锁和解锁。
4. 解铃还须系铃人。加锁和解锁必须是同一个客户端，客户端自己不能把别人加的锁给解了。

代码实现

组件依赖

首先我们要通过Maven引入Jedis开源组件，在pom.xml文件加入下面的代码：

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.9.0</version>
</dependency>
```

加锁代码

正确姿势

Talk is cheap, show me the code。先展示代码，再带大家慢慢解释为什么这样实现：

```
public class RedisTool {

    private static final String LOCK_SUCCESS = "OK";
    private static final String SET_IF_NOT_EXIST = "NX";
    private static final String SET_WITH_EXPIRE_TIME = "PX";

    /**
     * 尝试获取分布式锁
     * @param jedis Redis客户端
     * @param lockKey 锁
     * @param requestId 请求标识
     * @param expireTime 超期时间
     * @return 是否获取成功
     */
    public static boolean tryGetDistributedLock(Jedis jedis, String lockKey, String requestId, int
expireTime) {

        String result = jedis.set(lockKey, requestId, SET_IF_NOT_EXIST, SET_WITH_EXPIRE_TIME,
expireTime);

        if (LOCK_SUCCESS.equals(result)) {
            return true;
        }
        return false;

    }
```

```
}
```

可以看到，我们加锁就一行代码：`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，很多童鞋可能不明白，有key作为锁不就够了吗，为什么还要用到value？原因就是我们在上面讲到可靠性时，分布式锁要满足第四个条件解铃还须系铃人，通过给value赋值为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。requestId可以使用`UUID.randomUUID().toString()`方法生成。
- 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。

总的来说，执行上面的set()方法就只会导致两种结果：1. 当前没有锁（key不存在），那么就进行加锁操作，并对锁设置个有效期，同时value表示加锁的客户端。2. 已有锁存在，不做任何操作。

心细的童鞋就会发现了，我们的加锁代码满足我们可靠性里描述的三个条件。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。由于我们只考虑Redis单机部署的场景，所以容错性我们暂不考虑。

错误示例1

比较常见的错误示例就是使用`jedis.setnx()`和`jedis.expire()`组合实现加锁，代码如下：

```
public static void wrongGetLock1(Jedis jedis, String lockKey, String requestId, int expireTime) {

    Long result = jedis.setnx(lockKey, requestId);
    if (result == 1) {
        // 若在这里程序突然崩溃，则无法设置过期时间，将发生死锁
        jedis.expire(lockKey, expireTime);
    }

}
```

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。乍一看好像和前面的set()方法结果一样，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。网上之所以有人这样实现，是因为低版本的jedis并不支持多参数的set()方法。

错误示例2

```
public static boolean wrongGetLock2(Jedis jedis, String lockKey, int expireTime) {

    long expires = System.currentTimeMillis() + expireTime;
    String expiresStr = String.valueOf(expires);

    // 如果当前锁不存在，返回加锁成功
    if (jedis.setnx(lockKey, expiresStr) == 1) {
        return true;
    }

    // 如果锁存在，获取锁的过期时间
    String currentValueStr = jedis.get(lockKey);
    if (currentValueStr != null && Long.parseLong(currentValueStr) < System.currentTimeMillis()) {
        // 锁已过期，获取上一个锁的过期时间，并设置现在锁的过期时间
        String oldValueStr = jedis.getSet(lockKey, expiresStr);
        if (oldValueStr != null && oldValueStr.equals(currentValueStr)) {
            // 考虑多线程并发的情况，只有一个线程的设置值和当前值相同，它才有权利加锁
            return true;
        }
    }

    // 其他情况，一律返回加锁失败
    return false;

}
```

这一种错误示例就比较难以发现问题，而且实现也比较复杂。实现思路：使用`jedis.setnx()`命令实现加锁，其中key是锁，value是锁的过期时间。执行过程：1. 通过setnx()方法尝试加锁，如果当前锁不存在，返回加锁成功。2. 如果锁已经存在则获取锁的过期时间，和当前时间比较，如果锁已经过期，则设置新的过期时间，返回加锁成功。代码如下：

那么这段代码问题在哪里？1. 由于是客户端自己生成过期时间，所以需要强制要求分布式下每个客户端的时间必须同步。 2. 当锁过期的时候，如果多个客户端同时执行`jedis.getSet()`方法，那么虽然最终只有一个客户端可以加锁，但是这个客户端的锁的过期

2017年8月 (16)
2017年7月 (9)
2017年6月 (11)
2017年5月 (14)
2017年4月 (4)
2017年3月 (12)
2017年2月 (13)
2017年1月 (8)
2016年12月 (5)
2016年11月 (8)
2016年10月 (4)
2016年9月 (7)
2016年8月 (6)
2016年7月 (3)
2016年6月 (4)
2016年5月 (16)
2016年4月 (3)
2016年3月 (8)
2016年2月 (2)
2016年1月 (14)
2015年12月 (8)
2015年10月 (8)
2015年9月 (10)
2015年8月 (3)
2015年7月 (1)
2015年6月 (3)
2015年5月 (10)
2015年4月 (5)
2015年3月 (1)
2015年2月 (1)
2015年1月 (3)
2014年12月 (2)
2014年11月 (2)
2014年9月 (7)
2014年8月 (1)
2014年7月 (5)
2014年6月 (7)
2014年5月 (16)
2014年4月 (8)
2014年3月 (20)
2014年2月 (28)
2014年1月 (8)
2013年12月 (2)
2013年11月 (28)
2013年10月 (7)
2013年9月 (5)
2013年8月 (6)
2013年7月 (38)
2013年6月 (84)
2013年5月 (21)
2013年4月 (1)
2013年3月 (25)
2012年6月 (4)
2012年5月 (5)
2012年4月 (8)
2012年3月 (5)
2012年2月 (26)
2012年1月 (8)
2011年12月 (14)
2011年11月 (11)
2011年10月 (4)
2011年9月 (9)
2011年8月 (12)
2011年7月 (19)
2011年6月 (24)
2011年5月 (17)
2011年4月 (24)
2011年3月 (49)

时间可能被其他客户端覆盖。3. 锁不具备拥有者标识，即任何客户端都可以解锁。

解锁代码

正确姿势

还是先展示代码，再带大家慢慢解释为什么这样实现：

```
public class RedisTool {

    private static final Long RELEASE_SUCCESS = 1L;

    /**
     * 释放分布式锁
     * @param jedis Redis客户端
     * @param lockKey 锁
     * @param requestId 请求标识
     * @return 是否释放成功
     */
    public static boolean releaseDistributedLock(Jedis jedis, String lockKey, String requestId) {

        String script = "if redis.call('get', KEYS[1]) == ARGV[1] then return redis.call('del',
KEYS[1]) else return 0 end";
        Object result = jedis.eval(script, Collections.singletonList(lockKey),
Collections.singletonList(requestId));

        if (RELEASE_SUCCESS.equals(result)) {
            return true;
        }
        return false;

    }

}
```

可以看到，我们解锁只需要两行代码就搞定了！第一行代码，我们写了一个简单的Lua脚本代码，上一次见到这个编程语言还是在《黑客与画家》里，没想到这次居然用上了。第二行代码，我们将Lua代码传到`jedis.eval()`方法里，并使参数KEYS[1]赋值为lockKey，ARGV[1]赋值为requestId。eval()方法是将Lua代码交给Redis服务端执行。

那么这段Lua代码的功能是什么呢？其实很简单，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）。那么为什么要使用Lua语言来实现呢？因为要确保上述操作是原子性的。关于非原子性会带来什么问题，可以阅读【解锁代码-错误示例2】。那么为什么执行eval()方法可以确保原子性，源于Redis的特性，下面是官网对eval命令的部分解释：

> ## Atomicity of scripts
>
> Redis uses the same Lua interpreter to run all the commands. Also R
> guarantees that a script is executed in an atomic way: no other script
> Redis command will be executed while a script is being executed. Thi
> semantic is similar to the one of MULTI / EXEC. From the point of viev
> the other clients the effects of a script are either still not visible or alr
> completed.
>
> However this also means that executing slow scripts is not a good ide
> not hard to create fast scripts, as the script overhead is very low, but
> are going to use slow scripts you should be aware that while the scrip
> running no other client can execute commands.

简单来说，就是在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。

错误示例1

最常见的解锁代码就是直接使用`jedis.del()`方法删除锁，这种不先判断锁的拥有者而直接解锁的方式，会导致任何客户端都可以随时进行解锁，即使这把锁不是它的。

```
public static void wrongReleaseLock1(Jedis jedis, String lockKey) {
    jedis.del(lockKey);
}
```

错误示例2

2011年2月 (45)
2011年1月 (1)

积分与排名
积分 - 1298845
排名 - 46

最新评论
1. Re:二、oracle pctfree和pctused详解
楼主你好~pctused的例子有点疑问：一共存放100个数据，现在是90，而pctused为40%，是指当快内数据低于40个数据的时候，才可以插入。这时，应该至少删除51个数据，快内数据是39个，才可.....
--Fairy帆帆
2. Re:Redis分布式锁的正确实现方式
集大成的一篇文章，最简单明了的说清楚了
--张万帆
3. Re:ActiveMQ开发注意要点
mark，写的很好。
--问题大白
4. Re:Redis分布式锁的正确实现方式
@vision_08021. lock 方法不支持重入。####同一时刻只有一个线程获取锁，这个锁是排他性的；获取不到锁的同学只能排队等候。2. unlock解锁时，若锁过期，get拿不到值，此时也是.....
--Ruthless
5. Re:Redis分布式锁的正确实现方式
@Ruthless您这里的客户端指的是？还有自动过期时间是redis设置锁的过期时间吗？还有超时时间5~50ms是在哪儿设置？没太搞明白您的意思，还麻烦您解释下。...
--westboy

阅读排行榜
1. jQuery Validate验证框架详解(293961)
2. Linux启动/停止/重启Mysql数据库的方法(269227)
3. java枚举使用详解(214850)
4. 数据库设计三大范式(183093)
5. java.lang.OutOfMemoryError: Java heap space解决方法(172565)

评论排行榜
1. 二十二、startActivityForResult用法详解(35)
2. 数据库设计三大范式(33)
3. jQuery Validate验证框架详解(18)
4. Bootstrap 栅格系统(16)
5. jpa+spring配置多数据源(15)

推荐排行榜
1. 数据库设计三大范式(61)
2. 二十二、startActivityForResult用法详解(30)
3. jQuery Validate验证框架详解(29)
4. java枚举使用详解(25)
5. 十二、ContentProvider和Uri详解(25)

这种解锁代码乍一看也是没问题，甚至我之前也差点这样实现，与正确姿势差不多，唯一区别的是分成两条命令去执行，代码如下：

```
public static void wrongReleaseLock2(Jedis jedis, String lockKey, String requestId) {

    // 判断加锁与解锁是不是同一个客户端
    if (requestId.equals(jedis.get(lockKey))) {
        // 若在此时，这把锁突然不是这个客户端的，则会误解锁
        jedis.del(lockKey);
    }

}
```

如代码注释，问题在于如果调用`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。那么是否真的有这种场景？答案是肯定的，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。

总结

本文主要介绍了如何使用Java代码正确实现Redis分布式锁，对于加锁和解锁也分别给出了两个比较经典的错误示例。其实想要通过Redis实现分布式锁并不难，只要保证能满足可靠性里的四个条件。互联网虽然给我们带来了方便，只要有问题就可以google，然而网上的答案一定是对的吗？其实不然，所以我们更应该时刻保持着质疑精神，多想多验证。

如果你的项目中Redis是多机部署的，那么可以尝试使用`Redisson`实现分布式锁，这是Redis官方提供的Java组件，链接在参考阅读章节已经给出。

Ruthless
关注 - 29
粉丝 - 3002
+加关注

3 0

« 上一篇：Redis+Jedis封装工具类
» 下一篇：Git 的4个阶段的撤销更改

posted on 2017-12-08 09:11 Ruthless 阅读(3911) 评论(6) 编辑 收藏

评论：

#1楼 2018-03-24 18:41 | vision_0802
你好。 请教下：
1. lock 方法不支持重入。
2. unlock解锁时，若锁过期，get拿不到值，此时也是解锁失败吗？我的理解应该是不存在时也是解锁成功，不知道是否准确。
3. lock方法没有获取到锁，下一次什么时候在去获取锁？意思是线程需要sleep多久？我有一个想法是lock 返回ttl时间，避免频繁sleep，引发线程上下文频繁切换。
不好意思，问题有点多。请有时间时看看。谢谢！
支持(0) 反对(0)

#2楼 2018-03-27 18:10 | westboy
不错，但是还有一个问题，当我获取到锁之后，执行业务代码，业务代码的执行时间大于设置的过期时间时，业务还没有执行完就释放掉锁，是不是会有问题呢？
支持(0) 反对(0)

#3楼[楼主] 2018-03-28 16:08 | Ruthless
@ westboy
客户端需要一个比过期时间小很多的超时时间，例如，如果自动过期时间为10s，那么超时时间大概是5~50ms，这样可以避免客户端一直被阻塞，而不能继续请求下一个实例。
支持(0) 反对(0)

#4楼 2018-03-28 16:14 | westboy
@ Ruthless
您这里的客户端指的是？还有自动过期时间是redis设置锁的过期时间吗？还有超时时间5~50ms是在哪儿设置？没太搞明白您的意思，还麻烦您解释下。
支持(0) 反对(0)

#5楼[楼主] 2018-03-28 16:16 | Ruthless
@ vision_0802
1. lock 方法不支持重入。
####同一时刻只有一个线程获取锁，这个锁是排他性的；获取不到锁的同学只能排队等候。

2. unlock解锁时，若锁过期，get拿不到值，此时也是解锁失败吗？我的理解应该是不存在时也是解锁成功，不知道是否准确。

####避免解锁过期，引入超时时间，如：客户端需要一个比过期时间小很多的超时时间，例如，如果自动过期时间为10s，那么超时时间大概是5~50ms，这样可以避免客户端一直被阻塞，而不能继续请求下一个实例。

3. lock方法没有获取到锁，下一次什么时候在去获取锁？意思是线程需要sleep多久？我有一个想法是lock 返回ttl时间，避免频繁sleep，引发线程上下文频繁切换。
####这个根据具体业务具体处理，如：获取不到锁重试N次或直接返回无法获取锁。

支持(0) 反对(0)

#6楼 2018-04-07 21:34 | 张万帆
集大成的一篇文章，最简单明了的说清楚了

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· IPO热潮带动更多公司进入 今年将是科技行业并购年
· 乐视危机的又一个受伤者 毅昌股份去年净利同比降2575%
· 马斯克拟进行超级高铁加速和刹车测试 速度可达半音速
· 彭蕾将卸任蚂蚁金服董事长，CEO井贤栋将兼任董事长一职
· 小米河南营销团队23名员工遭解聘 新零售之路坎坷前行
» 更多新闻...

最新知识库文章:
· 写给自学者的入门指南
· 和程序员谈恋爱
· 学会学习
· 优秀技术人的管理陷阱
· 作为一个程序员，数学对你到底有多重要
» 更多知识库文章...