链滴

[转]java 多线程读写锁 ReadWriteLock

作者：boolean-dev

原文链接：https://ld246.com/article/1566357461591

来源网站：链滴

Java多线程中读写锁ReadWriteLock的使用

该博客转载自lavimer的Java多线程中读写锁ReadWriteLock的使用

1. 概念

读写锁分为读锁和写锁，多个读锁之间是不需要互斥的(读操作不会改变数据，如果上了锁，反而会影效率)，写锁和写锁之间需要互斥，也就是说，如果只是读数据，就可以多个线程同时读，但是如果你写数据，就必须互斥，使得同一时刻只有一个线程在操作。

2. 案例

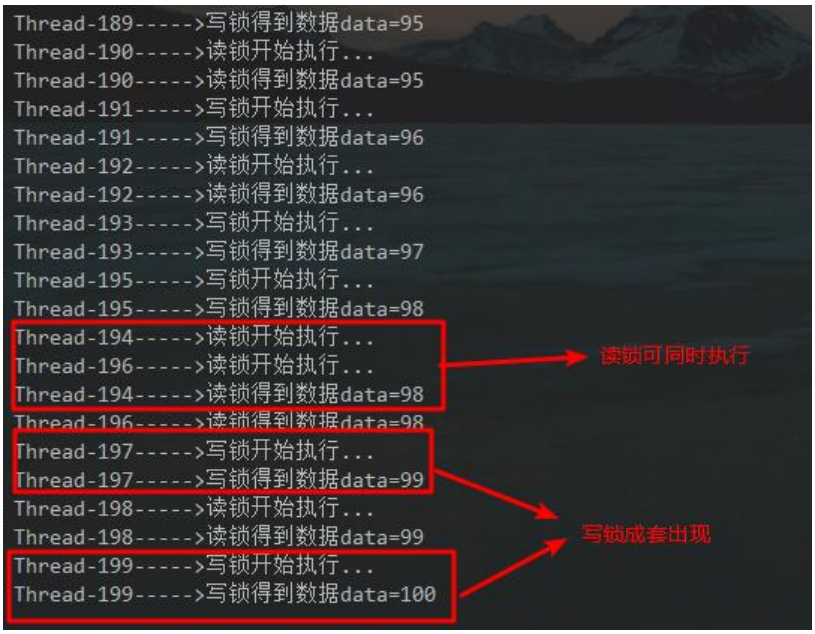
2.1 三线程读数据，三线程写数据

```
package com.tao.leanrn.thread.lock;

import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

/**
 * @ClassName ReadWrite
 * @Descriiption 读写锁
 * @Author yanjiantao
 * @Date 2019/8/21 10:42
 **/
public class ReadWrite {

    /**
     * 共享数据
```

```
     */
    private int data = 0;

    /**
     * 读写锁
     */
    private ReadWriteLock lock = new ReentrantReadWriteLock();

    public void get() {

        // 上读锁
        lock.readLock().lock();

        try {
            System.out.println(Thread.currentThread().getName() + "----->读锁开始执行...");
            Thread.sleep((long)(Math.random() * 10));
            System.out.println(Thread.currentThread().getName() + "----->读锁得到数据data=" +
ata);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            // 释放锁
            lock.readLock().unlock();
        }
    }

    public void set() {

        // 上写锁
        lock.writeLock().lock();

        try {
            System.out.println(Thread.currentThread().getName() + "----->写锁开始执行...");
            Thread.sleep((long)(Math.random() * 10));
            data++;
            System.out.println(Thread.currentThread().getName() + "----->写锁得到数据data=" +
ata);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            // 释放写锁
            lock.writeLock().unlock();
        }

    }
}

/**
 * 测试类
 */
class Main{

    public static void main(String[] args) {
        ReadWrite readWrite = new ReadWrite();
```

```
        // 创建100个读写线程
        for (int i = 0; i < 100; i++) {
            new Thread(() -> {
                readWrite.get();
            }).start();

            new Thread(() -> {
                readWrite.set();
            }).start();
        }
    }

}
```

程序运行结果



2.2 模拟Hibernate缓存

```
public class HibernateCache {

    /* 定义一个Map来模拟缓存 */
    private Map<String, Object> cache = new HashMap<String, Object>();

    /* 创建一个读写锁 */
    private ReadWriteLock rwLock = new ReentrantReadWriteLock();

    /**
     * 模拟Hibernate缓存
     * @param key
     * @return
     */
    private Object getData(String key) {

        /* 上读锁 */
```

```
        rwLock.readLock().lock();
        /* 定义从缓存中读取的对象 */
        Object value = null;

        try {
            /* 从缓存中读取数据 */
            value = cache.get(key);

            if (value == null) {
                /* 如果缓存中没有数据，我们就把读锁关闭，直接上写锁【让一个线程去数据库中取数据】
*/
                rwLock.readLock().unlock();
                /* 上写锁 */
                rwLock.writeLock().lock();

                try {
                    /* 上了写锁之后再判断一次【我们只让一个线程去数据库中取值即可，当第二个线程过
的时候，发现value不为空了就去缓存中取值】 */
                    if (value == null) {
                        /* 模拟去数据库中取值 */
                        value = "hello";
                    }
                } finally {
                    /* 写完之后把写锁关闭 */
                    rwLock.writeLock().unlock();
                }
                /* 缓存中已经有了数据，我们再把已经 关闭的读锁打开 */
                rwLock.readLock().lock();
            }
        } finally {
            /* 最后把读锁也关闭 */
            rwLock.readLock().unlock();
        }

        return value;
    }

}
```