链滴

读写锁 / 阻塞队列（Juc-05）

作者：yscxy

原文链接：https://ld246.com/article/1605963032766

来源网站：链滴

读写锁简介

```
public class ReentrantLock
extends Object
implements Lock, Serializable
```

一个可重入的互斥 Lock具有相同的基本行为和语义为隐式监控锁使用 synchronized方法和报表访问，但扩展功能。

一个ReentrantLock是由线程最后成功锁定，但尚未解锁它。一个线程调用lock将返回，成功获取锁，当锁不是由另一个线程拥有。如果当前线程已经拥有锁，该方法将立即返回。这可以使用方法isHeldByCurrentThread()检查，并getHoldCount()。

此类的构造函数接受一个可选的公平性参数。当设置true，争，锁青睐授予访问最长等待线程。否则，此锁不保证任何特定的访问顺序。使用许多线程访问的公平锁的程序可能会显示较低的整体吞吐量（即，比那些使用默认设置慢，往往要慢得多），但有较小的差异，在时间获得锁和保证缺乏饥饿。请注意，锁的公平性并不能保证线程调度的公平性。因此，使用一个公平锁的许多线程之一可能会获得它的连续多次，而其他活动线程不进展，而不是目前持有的锁。还要注意，不定时的tryLock()方法不尊重公平设置。如果锁是可用的，即使其他线程正在等待，它也会成功的。

这是推荐的做法总是紧跟一个电话lock与try块，最典型的是在前/后建设等：

代码实现

大致意思就是可以被多线程同时读写的时候只能有一个线程去写！

加入读锁是为了允许别人一起读，防止其他线程写

```
package net.yscxy.rw;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;

/**
 * @Author WangFuKun
```

```
 * @create 2020/11/21 16:07
 */
/*
 *独占锁（写锁）只能一个线程可以占有
 * 共享锁（读锁）多个线程可以 同时占有
 * 读-读 可以共存
 * 读-写 不能共存
 * 写-写 不能共存
 * */
public class ReentrantLockDemo {

    public static void main(String[] args) {
        MyCacheLock myCache = new MyCacheLock();

        for (int i = 1; i < 5; i++) {
            final int temp = i;
            new Thread(() -> {
                myCache.put(String.valueOf(temp), temp);
            }, String.valueOf(i)).start();
        }
        for (int i = 1; i < 5; i++) {
            final int temp = i;
            new Thread(() -> {
                myCache.get(String.valueOf(temp));
            }, String.valueOf(i)).start();
        }
    }
}

class MyCacheLock {
    private volatile Map<String, Object> map = new HashMap<>();
    //这是一把读写锁
    private ReadWriteLock readWriteLock = new ReentrantReadWriteLock();

    //储存，也就是写入，我们写入的时候只希望只有一个线程写
    public void put(String key, Object value) {
        readWriteLock.writeLock().lock();
        try {
            System.out.println(Thread.currentThread().getName() + "写入" + key);
            map.put(key, value);
            System.out.println(Thread.currentThread().getName() + "写入OK");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            readWriteLock.writeLock().unlock();
        }
    }

    //取，读 ，读的时候我们希望所有人都可以读
    public Object get(String key) {
        readWriteLock.readLock().lock();
        Object o = null;
        try {
            System.out.println(Thread.currentThread().getName() + "读取" + key);
```

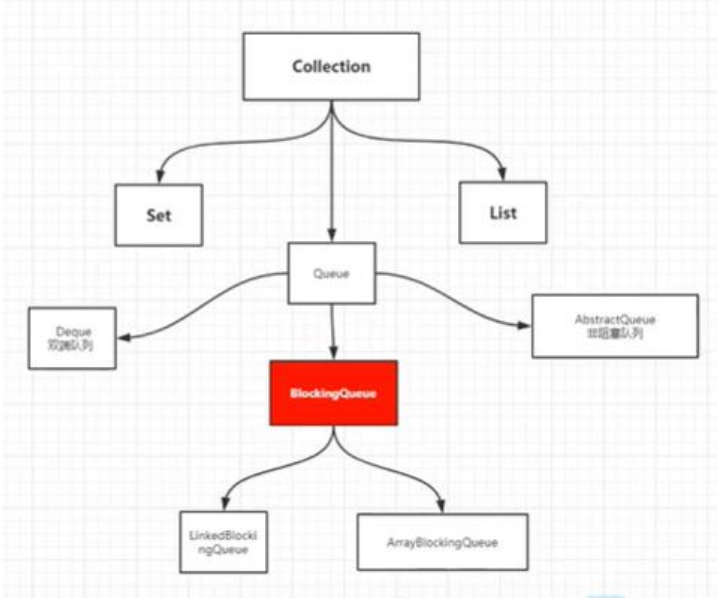
```
            o = map.get(key);
            System.out.println(Thread.currentThread().getName() + "读取OK");
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            readWriteLock.readLock().unlock();
        }
        return o;
    }
}

class MyCache {
    private volatile Map<String, Object> map = new HashMap<>();

    //储存
    public void put(String key, Object value) {
        System.out.println(Thread.currentThread().getName() + "写入" + key);
        map.put(key, value);
        System.out.println(Thread.currentThread().getName() + "写入OK");
    }

    //取，读
    public Object get(String key) {
        System.out.println(Thread.currentThread().getName() + "读取" + key);
        Object o = map.get(key);
        System.out.println(Thread.currentThread().getName() + "读取OK");
        return o;
    }
}
```

阻塞队列



阻塞队列的四组API

方式	抛出异常	有返回值，不抛出异常	塞等待	超时等待
添加	add	offer（）	put	ffer（，，）
移除	remove	pull（）	take	oll（，）
监测队列首	element	peek（）		

1. 抛出异常
2. 不会抛出异常
3. 阻塞，等待
4. 超时等待

代码实现

```
package net.yscxy.bq;

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

/**
 * @Author WangFuKun
 * @create 2020/11/21 17:37
 */
/*
 *阻塞队列
 * 那么什么时候会用到阻塞队列呢?
 * 多线程A要调用B，但是B还没有执行完成
 * 也就是多线程并发处理，线程池
```

```
 * */
public class Test {
    public static void main(String[] args) throws InterruptedException {
        test4();
    }

    /*抛出异常*/
    public static void test1() {
        ArrayBlockingQueue arrayBlockingQueue = new ArrayBlockingQueue<>(3);
        System.out.println(arrayBlockingQueue.add("a"));
        //查看队首元素是谁
        System.out.println(arrayBlockingQueue.element());

        System.out.println(arrayBlockingQueue.add("b"));
        System.out.println(arrayBlockingQueue.add("c"));
        //ava.lang.IllegalStateException: Queue full
        //System.out.println(arrayBlockingQueue.add("d"));

        System.out.println(arrayBlockingQueue.remove());
        System.out.println(arrayBlockingQueue.remove());
        System.out.println(arrayBlockingQueue.remove());
        //java.util.NoSuchElementException
        //System.out.println(arrayBlockingQueue.remove());
    }

    /*
     * 没有异常的方式
     * */
    public static void test2() {
        ArrayBlockingQueue arrayBlockingQueue = new ArrayBlockingQueue<>(3);
        System.out.println(arrayBlockingQueue.offer("a"));
        //查看队首元素
        System.out.println(arrayBlockingQueue.peek());
        System.out.println(arrayBlockingQueue.offer("a"));
        System.out.println(arrayBlockingQueue.offer("a"));
        //这样就不抛出异常，返回false
        System.out.println(arrayBlockingQueue.offer("a"));
        System.out.println(arrayBlockingQueue.poll());
        System.out.println(arrayBlockingQueue.poll());
        System.out.println(arrayBlockingQueue.poll());
        //返回null，但是不报错
        System.out.println(arrayBlockingQueue.poll());
    }

    /*
     * 等待，阻塞（一直阻塞）
     * */
    public static void test3() throws InterruptedException {
        ArrayBlockingQueue arrayBlockingQueue = new ArrayBlockingQueue<>(3);
        arrayBlockingQueue.put("a");
        arrayBlockingQueue.put("a");
        arrayBlockingQueue.put("a");
        // arrayBlockingQueue.put("a");一直等待
        System.out.println(arrayBlockingQueue.take());
```

```
        System.out.println(arrayBlockingQueue.take());
        System.out.println(arrayBlockingQueue.take());
        //System.out.println(arrayBlockingQueue.take());一直阻塞状态
    }

    public static void test4() throws InterruptedException {
        ArrayBlockingQueue arrayBlockingQueue = new ArrayBlockingQueue<>(3);
        arrayBlockingQueue.offer("a");
        arrayBlockingQueue.offer("b");
        arrayBlockingQueue.offer("c");
        //超时退出
        //arrayBlockingQueue.offer("d", 2, TimeUnit.SECONDS);
        System.out.println(arrayBlockingQueue.poll());
        System.out.println(arrayBlockingQueue.poll());
        System.out.println(arrayBlockingQueue.poll());
        //超时退出
        System.out.println(arrayBlockingQueue.poll(2, TimeUnit.SECONDS));
    }
}
```

同步队列

SynchronousQueue

他和其他的队列不一样，它不存储元素，put了一个元素，必须从里面先take出来，否则不能再put进去

```
package net.yscxy.bq;

import java.util.concurrent.SynchronousQueue;
import java.util.concurrent.TimeUnit;

/**
 * @Author WangFuKun
 * @create 2020/11/21 20:36
 */
/*
 * 同步队列
 * */
public class SynchronousQueueDemo {
    public static void main(String[] args) {
        SynchronousQueue<String> queue = new SynchronousQueue<>();
        new Thread(() -> {
            try {
                System.out.println(Thread.currentThread().getName() + "put 1");
                queue.put("1");
                System.out.println(Thread.currentThread().getName() + "put 2");
                queue.put("2");
                System.out.println(Thread.currentThread().getName() + "put 3");
                queue.put("3");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
        new Thread(() -> {
```

```
            try {
                TimeUnit.SECONDS.sleep(3);
                System.out.println(Thread.currentThread().getName() + "->" + queue.take());
                TimeUnit.SECONDS.sleep(3);
                System.out.println(Thread.currentThread().getName() + "->" + queue.take());
                TimeUnit.SECONDS.sleep(3);
                System.out.println(Thread.currentThread().getName() + "->" + queue.take());
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
    }
}
```