深入浅出 Java Concurrency (12): 锁机制 part 7 信号量（Semaphore）

Semaphore 是一个计数信号量。从概念上讲，信号量维护了一个许可集。如有必要，在许可可用前会阻塞每一个 acquire()，然后再获取该许可。每个 release() 添加一个许可，从而可能释放一个正在阻塞的获取者。但是，不使用实际的许可对象，Semaphore 只对可用许可的号码进行计数，并采取相应的行动。

说白了，Semaphore是一个计数器，在计数器不为0的时候对线程就放行，一旦达到0，那么所有请求资源的新线程都会被阻塞，包括增加请求到许可的线程，也就是说Semaphore不是可重入的。每一次请求一个许可都会导致计数器减少1，同样每次释放一个许可都会导致计数器增加1，一旦达到了0，新的许可请求线程将被挂起。

缓存池整好使用此思想来实现的，比如链接池、对象池等。

清单1 对象池

```
package xylz.study.concurrency.lock;

import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ObjectCache<T> {

    public interface ObjectFactory<T> {

        T makeObject();
    }

    class Node {

        T obj;

        Node next;
    }

    final int capacity;

    final ObjectFactory<T> factory;

    final Lock lock = new ReentrantLock();

    final Semaphore semaphore;

    private Node head;

    private Node tail;

    public ObjectCache(int capacity, ObjectFactory<T>
factory) {
        this.capacity = capacity;
        this.factory = factory;
        this.semaphore = new Semaphore(this.capacity);
        this.head = null;
```

2012年2月 (3)
2012年1月 (2)
2011年12月 (6)
2011年11月 (1)
2011年10月 (1)
2011年7月 (2)
2011年6月 (3)
2011年4月 (1)
2011年2月 (2)
2011年1月 (5)
2010年12月 (4)
2010年11月 (4)
2010年8月 (3)
2010年7月 (24)
2010年6月 (2)
2010年1月 (5)
2009年12月 (12)
2009年11月 (2)
2009年9月 (1)
2009年7月 (1)

文章分类(12)
Eclipse (rss)
Java (rss)
Python(12) (rss)

文章档案(12)
2010年6月 (6)
2010年5月 (3)
2010年4月 (3)

友情链接
imxylz (rss)
jafka
a fast distributed publish-subscribe messaging system

搜索
搜索

积分与排名
积分 - 1918335
排名 - 4

最新评论 XML
1. re: 深入浅出 Java Concurrency (9): 锁机制 part 4
评论内容较长,点击标题查看
--guanzhisong
2. 提供参考链接
评论内容较长,点击标题查看
--33
3. re: 处理Zookeeper的session过期问题
评论内容较长,点击标题查看
--codewarrior
4. re: 当Ajax遭遇GBK编码 (完全解决方案)
前面说的都同意 就是解决方法说的太抽象 没看懂 给个具体的方法
--穆

```
        this.tail = null;
    }

    public T getObject() throws InterruptedException {
        semaphore.acquire();
        return getNextObject();
    }

    private T getNextObject() {
        lock.lock();
        try {
            if (head == null) {
                return factory.makeObject();
            } else {
                Node ret = head;
                head = head.next;
                if (head == null) tail = null;
                ret.next = null;//help GC
                return ret.obj;
            }
        } finally {
            lock.unlock();
        }
    }

    private void returnObjectToPool(T t) {
        lock.lock();
        try {
            Node node = new Node();
            node.obj = t;
            if (tail == null) {
                head = tail = node;
            } else {
                tail.next = node;
                tail = node;
            }

            } finally {
            lock.unlock();
        }
    }

    public void returnObject(T t) {
        returnObjectToPool(t);
        semaphore.release();
    }

    }
```

清单1描述了一个基于信号量Semaphore的对象池实现。此对象池最多支持capacity个对象，这在构造函数中传入。对象池有一个基于FIFO的队列，每次从对象池的头结点开始取对象，如果头结点为空就直接构造一个新的对象返回。否则将头结点对象取出，并且头结点往后移动。特别要说明的如果对象的个数用完了，那么新的线程将被阻塞，直到有对象被返回回来。返还对象时将对象加入FIFO的尾节点并且释放一个空闲的信号量，表示对象池中增加一个可用对象。

实际上对象池、线程池的原理大致上就是这样的，只不过真正的对象池、线程池要处理比较复杂的逻辑，所以实现起来还需要做很多的工作，例如超时机制，自动回

收机制，对象的有效期等等问题。

这里特别说明的是信号量只是在信号不够的时候挂起线程，但是并不能保证信号量足够的时候获取对象和返还对象是线程安全的，所以在清单1中仍然需要锁Lock来保证并发的正确性。

将信号量初始化为 1，使得它在使用时最多只有一个可用的许可，从而可用作一个相互排斥的锁。这通常也称为*二进制信号量*，因为它只能有两种状态：一个可用的许可，或零个可用的许可。按此方式使用时，二进制信号量具有某种属性（与很多 Lock 实现不同），即可以由线程释放“锁”，而不是由所有者（因为信号量没有所有权的概念）。在某些专门的上下文（如死锁恢复）中这会很有用。

上面这段话的意思是说当某个线程A持有信号量数为1的信号量时，其它线程只能等待此线程释放资源才能继续，这时候持有信号量的线程A就相当于持有了“锁”，其它线程的继续就需要这把锁，于是线程A的释放才能决定其它线程的运行，相当于扮演了“锁”的角色。

另外同公平锁非公平锁一样，信号量也有公平性。如果一个信号量是公平的表示线程在获取信号量时按FIFO的顺序得到许可，也就是按照请求的顺序得到释放。这里特别说明的是：所谓请求的顺序是指在请求信号量而进入FIFO队列的顺序，有可能某个线程先请求信号而后进去请求队列，那么次线程获取信号量的顺序就会晚于其后请求但是先进入请求队列的线程。这个在公平锁和非公平锁中谈过很多。

除了acquire以外，Semaphore还有几种类似的acquire方法，这些方法可以更好的处理中断和超时或者异步等特性，可以参考JDK API。

按照同样的学习原则，下面对主要的实现进行分析。Semaphore的acquire方法实际上访问的是**AQS**的*acquireSharedInterruptibly(arg)*方法。这个可以参考**CountDownLatch**一节或者**AQS**一节。

所以Semaphore的await实现也是比较简单的。与CountDownLatch不同的是，Semaphore区分公平信号和非公平信号。

清单2 公平信号获取方法

```
protected int tryAcquireShared(int acquires) {
Thread current = Thread.currentThread();
for (;;) {
   Thread first = getFirstQueuedThread();
   if (first != null && first != current)
      return -1;
   int available = getState();
   int remaining = available - acquires;
   if (remaining < 0 ||
      compareAndSetState(available, remaining))
      return remaining;
  }
 }
```

清单3 非公平信号获取方法

```
protected int tryAcquireShared(int acquires) {
return nonfairTryAcquireShared(acquires);
}

final int nonfairTryAcquireShared(int acquires) {
for (;;) {
```

```
            int available = getState();
            int remaining = available - acquires;
            if (remaining < 0 ||
                compareAndSetState(available, remaining))
                return remaining;
        }
    }
```

对比清单2和清单3可以看到，公平信号和非公平信号在于第一次尝试能否获取信号时，公平信号量总是将当前线程进入AQS的CLH队列进行排队（因为第一次尝试时队列的头结点线程很有可能不是当前线程，当然不排除同一个线程第二次进入信号量），从而根据AQS的CLH队列的顺序FIFO依次获取信号量；而对于非公平信号量，第一次立即尝试能否拿到信号量，一旦信号量的剩余数available大于请求数（acquires通常为1），那么线程就立即得到了释放，而不需要进行AQS队列进行排队。只有remaining<0的时候（也就是信号量不够的时候）才会进入AQS队列。

所以非公平信号量的吞吐量总是要比公平信号量的吞吐量要大，但是需要强调的是非公平信号量和非公平锁一样存在"饥渴死"的现象，也就是说活跃线程可能总是拿到信号量，而非活跃线程可能难以拿到信号量。而对于公平信号量由于总是靠请求的线程的顺序来获取信号量，所以不存在此问题。

参考资料：

1. 信号量(Semaphore)在生产者和消费者模式的使用
2. What is mutex and semaphore in Java ? What is the main difference ?
3. 关于 java.util.concurrent 您不知道的 5 件事，第 2 部分
4. Semahores

锁机制 part 6 目 录
CyclicBarrier 锁机制 part 8 读写锁 (1)

posted on 2010-07-13 22:41 imxylz 阅读(20862) 评论(2) 编辑 收藏 所属分类: J2EE

评论

 # re: 深入浅出 Java Concurrency (12): 锁机制 part 7 信号量（Semaphore）[未登录] 2010-07-14 00:53 行云流水

留下成长的足迹。 回复 更多评论

 # re: 深入浅出 Java Concurrency (12): 锁机制 part 7 信号量（Semaphore）[未登录] 2013-06-20 21:28 flying

写得太好了，一口气读到这。 回复 更多评论

新用户注册 刷新评论列表

@高帆
jstack 打印出来 线程栈信息，能看到 线程栈目前运行在那个地方，等待什么资源
--shaw
22. re: 《深入浅出 Java Concurrency》目录[未登录]
谢谢楼主！
--luke
23. re: 世界邦旅行网(北京)招聘Java高级/资深工程师前端工程师/移动开发工程师等_20150616更新
评论内容较长,点击标题查看
--songxin
24. re: 深入浅出 Java Concurrency (9): 锁机制 part 4
评论内容较长,点击标题查看
--imxylz
25. re: 深入浅出 Java Concurrency (9): 锁机制 part 4
评论内容较长,点击标题查看
--mitisky
26. re: 一次简单却致命的错误
请交大侠！查看java线程是怎么看的
--高帆
27. re: 深入浅出 Java Concurrency (20): 并发容器 part 5 ConcurrentLinkedQueue
@mashiguang
我猜测应该是ArrayBlockingQueue用一把锁而LinkedBlockingQueue用两把锁的原因
--liubey
28. John
Thanksamundo for the post.Really thank you! Awesome. edebdbceaekadffd
--Smithc667
29. re: 捕获Java线程池执行任务抛出的异常
@imxylz
谢谢指点，你这种方式更优雅些，我自己是new了个exceptionQueue，new线程的时候set进去，然后执行完子线程后查看这个Queue
--liubey
30. re: 使用Nginx的proxyCache缓存功能

浏览器收到302之后，就直接去源链接下载了，消息都不经过nginx了，然后怎么缓存呢?
--wd
31. re: 使用Nginx的proxyCache缓存功能
nginx支持302这种非200状态码，但是当有302经过nginx，它缓存的是什么？是源文件还是链接？我从浏览器输入的时候还是会收到302啊。
--wd