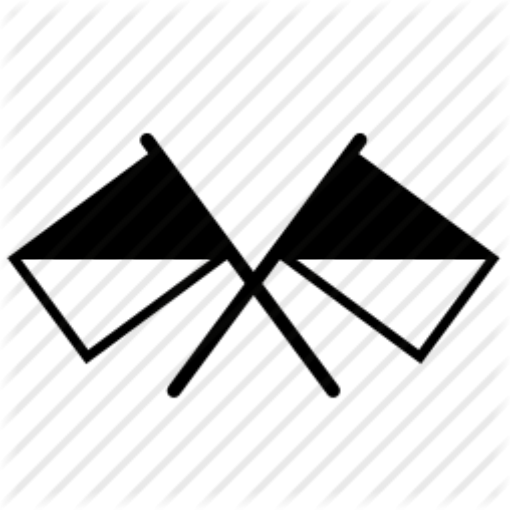
信号量

原文地址 By Jakob Jenkov 翻译：寒桐 校对：方腾飞



Semaphore（信号量）是一个线程同步结构，用于在线程间传递信号，以避免出现信号丢失（译者注：下文会具体介绍），或者像锁一样用于保护一个关键区域。自从5.0开始，jdk在java.util.concurrent包里提供了Semaphore 的官方实现，因此大家不需要自己去实现Semaphore。但是还是很有必要去熟悉如何使用Semaphore及其背后的原理

本文的涉及的主题如下：

1. 简单的Semaphore实现
2. 使用Semaphore来发出信号
3. 可计数的Semaphore
4. 有上限的Semaphore
5. 把Semaphore当锁来使用

一、简单的Semaphore实现

下面是一个信号量的简单实现：

```
01 public class Semaphore {
02
03 private boolean signal = false;
04
05 public synchronized void take() {
06
07 this.signal = true;
08
09 this.notify();
10
11 }
12
13 public synchronized void release() throws InterruptedException{
14
15 while(!this.signal) wait();
16
17 this.signal = false;
18
19 }
20
21 }
```

Take方法发出一个被存放在Semaphore内部的信号，而Release方法则等待一个信号，当其接收到信号后，标记位signal被清空，然后该方法终止。

使用这个semaphore可以避免错失某些信号通知。用take方法来代替notify，release方法来代替wait。如果某线程在调用release等待之前调用take方法，那么调用release方法的线程仍然知道take方法已经被某个线程调用过了，因为该Semaphore内部保存了take方法发出的信号。而wait和notify方法就没有这样的功能。

当用semaphore来产生信号时，take和release这两个方法名看起来有点奇怪。这两个名字来源于后面把semaphore当做锁的例子，后面会详细介绍这个例子，在该例子中，take和release这两个名字会变得很合理。

二、使用Semaphore来产生信号

下面的例子中，两个线程通过Semaphore发出的信号来通知对方

```
Semaphore semaphore = new Semaphore();

SendingThread sender = new SendingThread(semaphore);

ReceivingThread receiver = new ReceivingThread(semaphore);

receiver.start();

sender.start();

public class SendingThread {

Semaphore semaphore = null;

public SendingThread(Semaphore semaphore){

this.semaphore = semaphore;

}

public void run(){

while(true){

//do something, then signal

this.semaphore.take();

}

}

}

public class RecevingThread {

Semaphore semaphore = null;

public ReceivingThread(Semaphore semaphore){

this.semaphore = semaphore;

```

```
}

public void run(){

while(true){

this.semaphore.release();

//receive signal, then do something...

}

}

}
```

三、可计数的Semaphore

上面提到的Semaphore的简单实现并没有计算通过调用take方法所产生信号的数量。可以把它改造成具有计数功能的Semaphore。下面是一个可计数的Semaphore的简单实现。

```
public class CountingSemaphore {

private int signals = 0;

public synchronized void take() {

this.signals++;

this.notify();

}

public synchronized void release() throws InterruptedException{

while(this.signals == 0) wait();

this.signals--;

}

}
```

四、有上限的Semaphore

上面的CountingSemaphore并没有限制信号的数量。下面的代码将CountingSemaphore改造成一个信号数量有上限的BoundedSemaphore。

```
public class BoundedSemaphore {

private int signals = 0;

private int bound   = 0;

public BoundedSemaphore(int upperBound){

this.bound = upperBound;

}

public synchronized void take() throws InterruptedException{

```

```
15 while(this.signals == bound) wait();
16
17 this.signals++;
18
19 this.notify();
20
21 }
22
23 public synchronized void release() throws InterruptedException{
24
25 while(this.signals == 0) wait();
26
27 this.signals--;
28
29 this.notify();
30
31 }
32
33 }
```

在BoundedSemaphore中，当已经产生的信号数量达到了上限，take方法将阻塞新的信号产生请求，直到某个线程调用release方法后，被阻塞于take方法的线程才能传递自己的信号。

五、把Semaphore当锁来使用

当信号量的数量上限是1时，Semaphore可以被当做锁来使用。通过take和release方法来保护关键区域。请看下面的例子：

```
01 BoundedSemaphore semaphore = new BoundedSemaphore(1);
02
03 ...
04
05 semaphore.take();
06
07 try{
08
09 //critical section
10
11 } finally {
12
13 semaphore.release();
14
15 }
```

在前面的例子中，Semaphore被用来在多个线程之间传递信号，这种情况下，take和release分别被不同的线程调用。但是在锁这个例子中，take和release方法将被同一线程调用，因为只允许一个线程来获取信号（允许进入关键区域的信号），其它调用take方法获取信号的线程将被阻塞，知道第一个调用take方法的线程调用release方法来释放信号。对release方法的调用永远不会被阻塞，这是因为任何一个线程都是先调用take方法，然后再调用release。

通过有上限的Semaphore可以限制进入某代码块的线程数量。设想一下，在上面的例子中，如果BoundedSemaphore 上限设为5将会发生什么？意味着允许5个线程同时访问关键区域，但是你必须保证，这个5个线程不会互相冲突。否则你的应用程序将不能正常运行。

必须注意，release方法应当在finally块中被执行。这样可以保在关键区域的代码抛出异常的情况下，信号也一定会被释放。

阳光天鹅
2014/12/20 4:38下午

怎么感觉这个文章里信号量的take()与release()方法的含义跟java.util.concurrent.Semaphore中acquire()与release()的含义相反呢，很别扭。

yxrswx
2018/03/07 2:31下午

确实，含义反了

zhili
2018/06/12 3:15下午

> 阳光天鹅 :
>
> 怎么感觉这个文章里信号量的take()与release()方法的含义跟java.util.concurrent.Semaphore中acquire()与release()的含义相反呢，很别扭。

意思确实反了, 原文也是这样 …. 不知道是有什么含义 = =

zhili
2018/06/12 3:56下午

take() -> currentThread() 获取锁

release() -> currentThread() 释放锁

这么理解如何?

WSZ1102Tree
2018/08/15 6:21下午

感觉这代码块：

semaphore.take();

try{

//critical section

} finally {

semaphore.release();

}

有种java.util.concurrent.Lock的感觉

WSZ1102Tree
2018/08/15 6:13下午

在 四、有上限的Semaphore，如果将BoundedSemaphore的bound不设置初始值，此时bound的值为0，此时调用take和release方法，会产生？？多个线程处在等待（wait）状态，而没有唤醒； 产生死锁？？

WSZ1102Tree
2018/08/15 6:14下午

所以是不是bound的初始值就设置为1，这样作为一个非重入锁