黑客派

线程基础-----详细介绍线程状态转换（一）

作者：oldcaptain

原文链接：https://hacpai.com/article/1376567030681

来源网站：黑客派

<h2 id="toc_h2_0">一、场景：</h2>

      接触到java，如果要继续深入学习，那么就必须要了解线程，这个过程比较长的。如果在实际工作中没有接触到，自己摸索的话，始终不能得其道，当然这并不妨碍，我们线程基础学习一翻。以后有了实际的需求驱动，那么就会事半功倍。
     在学习的时候，线程的状态转换，线程调度，同步异步，线程的通信这基本的知识当然要牢牢掌握。在摸索中前进。
    在这里为大家推荐一个网站 并发编程
<h2 id="toc_h2_1">二、线程基础</h2>
<h3 id="toc_h3_2">   1、线程状态转换</h3>
<img src="https://static.hacpai.com/images/img-loading.svg" alt="thradstate.PNG" width="610" height="244" data-src="http://www.oldcaptain.cc:7080/blog_imaes/1376566929843/threadstate.PNG">
图示说明：
     1、新建状态：新建了一个线程对象。
     2、就绪状态(Runnable)：线程对象创建后，其他线程对象调用了该线对象的start()方法，该状态的线程位于可运行的线程池中，变得可以运行，等待获得cpu使用权。

     3、运行状态(running)：就绪状态的线程获得cpu，开始执行该线程的码。
     4、阻塞状态(Blocked)：阻塞状态是因为，线程由于某种原因放弃cpu用权，暂时停止运行。知道线程进入就绪状态，才有机会转到运行状态。阻塞状态情况分为三种：(<san>该状态为停止当前线程，当时并不释放其所占用的资源)
           4.1等待阻塞：运行的线程执行了wait，jvm把该线程放入等待的池中。
           4.2同步阻塞：运行的线程在获取对象同步锁，若该同步锁被别人的线程占用，则该jvm会把线程放入锁定池中。
           4.3其他阻塞：运行的线程执行执行sleep()或者oin()方法，或者发出io请求时，jvm会将该线程设置为阻塞。当sleep()状态超时，join等待线程终止超时，或者io处理完毕，线程重新转为就绪状态。
     5、死亡状态（dead）：线程执行完了，或者因异常退出了run方法，线程生命周期结束。
<h3 id="toc_h3_3">关于java线程调度的几个方法的解释说明：</h3>
      （一）当线程调用了自身的sleep()或者其他线程的join()方法时，就会入阻塞状态，当sleep()或者join方法结束后，该线程转为可运行状态，继续等待os分配cpu时间片

      （二）线程调用了yield方法，意思是放弃当前的cpu时间分片，回到运行状态，这时与其他可运行状态，这时与其他进程处于同等竞争状态，os有可能会接着让这个线程入运行状态；(注：yield并不是让线程进入就绪状态，而是运行状态，但是不占有cpu时间片已)
      （三）当线程刚进入可运行状态（即就绪状态），发现将要调用的资被synchroniza（同步），获取不到锁标记，将会立即进入锁池状态，等待获取锁标记（这时的锁定

里也许已经有了其他线程在等待获取锁标记，这时他们处于队列状态，即先到先得），一旦线程获得
程锁标记后，就可转入可运行状态，等待os分配cpu时间片；</p>
<p>   （四）当线程调用wait()方法后进入等待队列（进入到
个状态会释放所占有的资源，与阻塞不同），这个状态是不能够被自动唤醒的，必须依赖于
他线程调用notify（）或者notifyall（）方法才唤醒（wait（1000）可以自动唤醒）（由于notify（
只是唤醒一个线程，但我们不能确定具体唤醒的是那个线程，也许我们需要唤醒的线程不能够被唤醒
因此在实际使用时，一般都使用notifyall（）方法，唤醒所有的线程），线程被唤醒后会进入锁池，
待获取锁标记</p>
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></scr
pt>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342"
data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in
>
<script>
  (adsbygoogle = window.adsbygoogle || []).push({});
</script>
<p>   （五）wait()和notify()方法：当一个线程执行到wait()方法时，
就进入到一个和对象相关的等待池中，同时失去了对象锁。当他被一个notify()方法唤醒的时，等待
中的线程就被放到了锁定池中。该线程从池中获的锁，然后回到wait()方法前的中断现场</sp
n>。</p>
<p>   调用sleep，join时，不会释放所占用的资源，所以会进入到阻塞状态。
/p>
<p>   调用wait时，会释放所占用的资源，所以会进入到等待队列。</p>
<p>更加细致的图示：</p>
<p>   <a class="fancybox" href="https://link.hacpai.com/forward?goto=
ttp%3A%2F%2Fwww.oldcaptain.cc%3A7080%2Fblog_images%2F1376566929843%2Fthread7s
ate-4.PNG" target="_blank" rel="nofollow ugc"><img src="https://static.hacpai.com/images/
mg-loading.svg" alt="" width="610" height="294" data-src="http://www.oldcaptain.cc:7080/b
og_images/1376566929843/thread7state-4.PNG"></p>
<hr>
<p> 注：几个线程方法的介绍：</p>
<p>1、wait（）</p>
<p>   public final void wait() throws Interru
tedException,IllegalMonitorStateException</p>
<p>   该方法用来将当前线程置入休眠状态，直到接到通知或
中断为止。在调用wait（）之前，线程必须要获得该对象的对象级别锁，即只能在同步方法或同步块
调用wait（）方法。进入wait（）方法后，当前线程释放锁。在从wait（）返回前，线程与其他线程
争重新获得锁。如果调用wait（）时，没有持有适当的锁，则抛出IllegalMonitorStateException，
是RuntimeException的一个子类，因此，不需要try-catch结构。</p>
<p>2、notify（）</p>
<p>   public final native void notify() throws IllegalMonito
StateException</p>
<p>   该方法也要在同步方法
同步块中调用，即在调用前，线程也必须要获得该对象的对象级别锁，的如果调用notify（）时没有
有适当的锁，也会抛出IllegalMonitorStateException。</p>
<p>   该方法用来通知那些可能等待该对象的对象锁的其他线
。如果有多个线程等待，则线程规划器任意挑选出其中一个wait（）状态的线程来发出通知，并使它
待获取该对象的对象锁（notify后，当前线程不会马上释放该对象锁，wait所在的线程并不
马上获取该对象锁，要等到程序退出synchronized代码块后，当前线程才会释放锁，wait所在的线程
才可以获取该对象锁），但不惊动其他同样在等待被该对象notify的线程们。当第
个获得了该对象锁的wait线程运行完毕以后，它会释放掉该对象锁，此时如果该对象没有再次使用noti
y语句，则即便该对象已经空闲，其他wait状态等待的线程由于没有得到该对象的通知，会继续阻塞在
ait状态，直到这个对象发出一个notify或notifyAll。<span>这里需要注意：它们等待的是

notify或notifyAll，而不是锁。这与下面的notifyAll（）方法执行后的情况不同。
nbsp;
3、notifyAll（）
     public final native void notifyAll() throws IllegalMonitorStateException
      该方法与notify（）方法的工作式相同，重要的一点差异是：
      notifyAll使所有原来在该对象上wait的程统统退出wait的状态（即全部被唤醒，不再等待notify或notifyAll，但由于此时还没有获取到该对锁，因此还不能继续往下执行），变成等待获取该对象上的锁，一旦该对象锁被释放（notifyAll线程出调用了notifyAll的synchronized代码块的时候），他们就会去竞争。如果其中一个线程获得了该对锁，它就会继续往下执行，在它退出synchronized代码块，释放锁后，其他的已经被唤醒的线程将会续竞争获取该锁，一直进行下去，直到所有被唤醒的线程都执行完毕。
<script async src="https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js"></script>
<!-- 黑客派PC帖子内嵌-展示 -->
<ins class="adsbygoogle" style="display:block" data-ad-client="ca-pub-5357405790190342" data-ad-slot="8316640078" data-ad-format="auto" data-full-width-responsive="true"></in>
<script>
    (adsbygoogle = window.adsbygoogle || []).push({});
</script>

     4、wait（long）和wait（long,int）</pan>
     显然，这两个方法是设置等待超时时间的，后者超值时间上加上ns，精度也难以达到，因此，该方法很少使用。对于前者，如果在等待线程接到通知被中断之前，已经超过了指定的毫秒数，则它通过竞争重新获得锁，并从wait（long）返回。<span另外，需要知道，如果设置了超时时间，当wait（）返回时，我们不能确定它是因为接到了通知还是为超时而返回的，因为wait（）方法不会返回任何相关的信息。但一般可以通过设置标志位来判断，notify之前改变标志位的值，在wait（）方法后读取该标志位的值来判断，当然为了保证notify不被漏，我们还需要另外一个标志位来循环判断是否调用wait（）方法。
       深入理解：
   如果线程调用了对象的wait（）方法，那么线程便处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。

   当有线程调用了对象的notifyAll（）方法（唤醒所有ait线程）或notify（）方法（只随机唤醒一个wait线程），被唤醒的的线程便会进入该对象的锁池中锁池中的线程会去竞争该对象锁。
   优先级高的线程竞争到对象锁的概率大，假若某线没有竞争到该对象锁，它还会留在锁池中，唯有线程再次调用wait（）方法，它才会重新回到等待池。而竞争到对象锁的线程则继续往下执行，直到执行完了synchronized代码块，它会释放掉该对象锁这时锁池中的线程会继续竞争该对象锁。
几个线程方法介绍转载于“http://blog.csdn.net/ns_code/article/details/17225469”已经写得够清楚了，没必要我再赘述一次，再根据我文章里面画的线程状态转换图，会很清晰的理解程状态转换。
注：以下图是我照着网友的图画的。
<h3 id="toc_h3_4">下面的图为运行状态到锁池状态的转换</h3>
<img src="https://static.hacpai.com/images/img-loading.svg" alt="threa-1.PNG" width="610" height="324" data-src="http://www.oldcaptain.cc:7080/blog_images/176566929843/thread-1.PNG">

-2.PNG" width="610" height="324" data-src="http://www.oldcaptain.cc:7080/blog_images/1376566929843/thread-1.PNG">

-3.PNG" width="610" height="341" data-src="http://www.oldcaptain.cc:7080/blog_images/1376566929843/thread-3.PNG">
注：借鉴网上资料进行整理，再加上自己的理解，将图重新画了一遍。目的是能够更加清晰的理线程状态转换，线程状态转换是写好多线程的代码的基础。
再贴其他类似的文章，http://my.oshina.net/zhdkn/blog/114301，http://my.oschina.net/mingdongcheng/blog/139263