LockSupport：一个很灵活的线程工具类

愚公要移山1
发布时间: 20-05-13 12:34

LockSupport是一个编程工具类，主要是为了阻塞和唤醒线程用的。使用它我们可以实现很多功能，今天主要就是对这个工具类的讲解，希望对你有帮助：

一、LockSupport简介

1、LockSupport是什么

刚刚开头提到过，LockSupport是一个线程工具类，所有的方法都是静态方法，可以让线程在任意位置阻塞，也可以在任意位置唤醒。

它的内部其实两类主要的方法：park（停车阻塞线程）和unpark（启动唤醒线程）。

```
//（1）阻塞当前线程
public static void park(Object blocker);
//（2）暂停当前线程，有超时时间
public static void parkNanos(Object blocker, long nanos);
//（3）暂停当前线程，直到某个时间
public static void parkUntil(Object blocker, long deadline);
//（4）无期限暂停当前线程
public static void park();
//（5）暂停当前线程，不过有超时时间的限制
public static void parkNanos(long nanos);
//（6）暂停当前线程，直到某个时间
public static void parkUntil(long deadline);
//（7）恢复当前线程
public static void unpark(Thread thread);
public static Object getBlocker(Thread t);
```

注意上面的123方法，都有一个blocker，这个blocker是用来记录线程被阻塞时被谁阻塞的。用于线程监控和分析工具来定位原因的。

作者最新文章

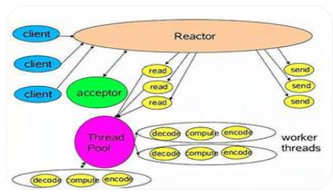
这么多的bean在容器中，Spring是如何区分的?

spring专题系列之IOC的理解和分析

spring专题系列之AOP的理解和分析

相关文章

Netty之线程模型



Java——面试官：讲一下String、StringBuilder及…

2、与wait/notify对比

这里假设你已经了解了wait/notify的机制，如果不了解，可以在网上一搜，很简单。相信你既然学到了这个LockSupport，相信你已经提前已经学了wait/notify。

我们先来举一个使用案例：

先park再unpark和先unpark再park都不会死锁

```
public class LockSupportTest {
    public static class MyThread extends Thread {
        @Override
        public void run() {
            System.out.println(getName() + " 进入线程");
            LockSupport.park();
            System.out.println("t1线程运行结束");
        }
    }
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();
        System.out.println("t1已经启动，但是在内部进行了park");
        LockSupport.unpark(t1);
        System.out.println("LockSupport进行了unpark");
    }
}
```

上面这段代码的意思是，我们定义一个线程，但是在内部进行了park，因此需要unpark才能唤醒继续执行，不过上面，我们在MyThread进行的park，在main线程进行的unpark。

这样来看，好像和wait/notify没有什么区别。那他的区别到底是什么呢？这个就需要仔细的观察了。这里主要有两点：

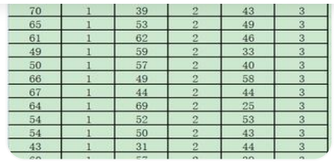
（1）wait和notify都是Object中的方法,在调用这两个方法前必须先获得锁对象，但是park不需要获取某个对象的锁就可以锁住线程。

（2）notify只能随机选择一个线程唤醒，无法唤醒指定的线程，unpark却可以唤醒一个指定的线程。

区别就是这俩，还是主要从park和unpark的角度来解释的。既然这个LockSupport这么强，我们就深入一下他的源码看看。

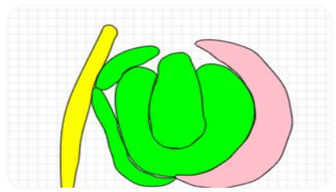
二、源码分析（基于jdk1.8）

1、park方法



Mybatis如何清晰的解决出现「多对一模型」和「一对多模…

LeetCode-055-跳跃游戏



```
    setBlocker(t, null);
}
```

```
public static void park(Object blocker) {
        Thread t = Thread.currentThread();
        setBlocker(t, blocker);
        UNSAFE.park(false, 0L);
        setBlocker(t, null);
}
```

blocker是用来记录线程被阻塞时被谁阻塞的。用于线程监控和分析工具来定位原因的。setBlocker(t, blocker)方法的作用是记录t线程是被broker阻塞的。因此我们只关注最核心的方法，也就是UNSAFE.park(false, 0L)。

UNSAFE是一个非常强大的类，他的的操作是基于底层的，也就是可以直接操作内存，因此我们从JVM的角度来分析一下：

每个java线程都有一个Parker实例：

```
class Parker : public os::PlatformParker {
private:
  volatile int _counter ;
  ...
public:
  void park(bool isAbsolute, jlong time);
  void unpark();
  ...
}
class PlatformParker : public CHeapObj<mtInternal> {
  protected:
    pthread_mutex_t _mutex [1] ;
    pthread_cond_t  _cond  [1] ;
    ...
}
```

我们换一种角度来理解一下park和unpark，可以想一下，unpark其实就相当于一个许可，告诉特定线程你可以停车，特定线程想要park停车的时候一看到有许可，就可以立马停车继续运行了。因此其执行顺序可以颠倒。

现在有了这个概念，我们体会一下上面JVM层面park的方法，这里面counter字段，就是用来记录所谓的“许可”的。

本文部分总结来源于：https://www.jianshu.com/p/1f16b838ccd8

当调用park时，先尝试直接能否直接拿到“许可”，即_counter>0时，如果成功，则把_counter设置为0,并返回。

```
void Parker::park(bool isAbsolute, jlong time) {
  // Ideally we'd do something useful while spinning, such
  // as calling unpackTime().
```

_counter往下走许可证：初始化=0，只有0和1两种状态，（1才可以继续往下走）

```
park():
if(_counter == 1){
  _counter = 0
  return//继续往下走
}
阻塞住，当超时了或者中断了，再继续往下走

unpark()：
_counter=1,
判断之前是否为0，是唤醒线程
```

案例1：先park再unpark
1、park-->阻塞住
2、unpark --> _counter从0变为1，唤醒线程
3、线程拿到cpu后继续执行
案例2：先unpark再park
1、unpark --> _counter置为1，唤醒线程
2、park --> 不阻塞，_counter置为0，继续往下走

案例3、unpark unpark parkpark
1、unpark --> _counter置为1，唤醒线程
2、unpark --> _counter再次置为1
3、park --> _counter置为0，不阻塞
4、park --> 阻塞

小结：
unpark获取park通行凭证，做多只有一张
park消耗通行凭证，没了就阻塞

1、park要么消耗通行证直接往下走
2、park阻塞住，直到中断或者被unpark

```
if (Atomic::xchg(0, &_counter) > 0) return;
```

如果不成功，则构造一个ThreadBlockInVM，然后检查_counter是不是>0，如果是，则把_counter设置为0，unlock mutex并返回:

```
ThreadBlockInVM tbivm(jt);
// no wait needed
if (_counter > 0)  {
  _counter = 0;
  status = pthread_mutex_unlock(_mutex);
```

否则，再判断等待的时间，然后再调用pthread_cond_wait函数等待，如果等待返回，则把_counter设置为0，unlock mutex并返回：

```
if (time == 0) {
  status = pthread_cond_wait (_cond, _mutex) ;
}
_counter = 0 ;
status = pthread_mutex_unlock(_mutex) ;
assert_status(status == 0, status, "invariant") ;
OrderAccess::fence();
```

这就是整个park的过程，总结来说就是消耗“许可”的过程。

2、unpark

还是先来看一下JDK源码：

```
/**
 * Makes available the permit for the given thread, if it
 * was not already available.  If the thread was blocked on
 * {@code park} then it will unblock.  Otherwise, its next call
 * to {@code park} is guaranteed not to block. This operation
 * is not guaranteed to have any effect at all if the given
 * thread has not been started.
 *
 * @param thread the thread to unpark, or {@code null}, in which case
 *        this operation has no effect
 */
public static void unpark(Thread thread) {
    if (thread != null)
        UNSAFE.unpark(thread);
```

上面注释的意思是给线程生产许可证。

当unpark时，则简单多了，直接设置_counter为1，再unlock mutext返回。如果_counter之前的值是0，则还要调用pthread_cond_signal唤醒在park中等待的线程：

```
void Parker::unpark() {
  int s, status ;
  status = pthread_mutex_lock(_mutex);
  assert (status == 0, "invariant") ;
  s = _counter;
  _counter = 1;
  if (s < 1) {
     if (WorkAroundNPTLTimedWaitHang) {
        status = pthread_cond_signal (_cond) ;
        assert (status == 0, "invariant") ;
        status = pthread_mutex_unlock(_mutex);
        assert (status == 0, "invariant") ;
     } else {
        status = pthread_mutex_unlock(_mutex);
        assert (status == 0, "invariant") ;
        status = pthread_cond_signal (_cond) ;
        assert (status == 0, "invariant") ;
     }
  } else {
    pthread_mutex_unlock(_mutex);
    assert (status == 0, "invariant") ;
  }
}
```

调用park()方法dump线程：

```
"main" #1 prio=5 os_prio=0 tid=0x02cdcc00 nid=0x2b48 waiting on condition
[0x00d6f000]
   java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:
304)
        at learn.LockSupportDemo.main(LockSupportDemo.java:7)
```

调用park(Object blocker)方法dump线程

```
"main" #1 prio=5 os_prio=0 tid=0x0069cc00 nid=0x6c0 waiting on condition
[0x00dcf000]
   java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x048c2d18> (a java.lang.String)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:
175)
        at learn.LockSupportDemo.main(LockSupportDemo.java:7)
```

带Object的park方法相较于无参的park方法会增加 parking to wait for <0x048c2d18> (a java.lang.String) 的信息

synchronzed致使线程阻塞，线程会进入到BLOCKED状态，而调用LockSupprt方法阻塞线程会致使线程进入到WAITING状态。

ok，现在我们已经对源码进行了分析，整个过程其实就是生产许可和消费许可的过程。而且这个生产过程可以反过来。也就是先生产再消费。下面我们使用几个例子验证一波。

三、LockSupport使用

1、不可重入（不像重入锁那样）
2、可响应中断

1、先interrupt再park

```
public class LockSupportTest {
    public static class MyThread extends Thread {
        @Override
        public void run() {
            System.out.println(getName() + " 进入线程");
            LockSupport.park();
            System.out.println(" 运行结束");
            System.out.println("是否中断: " + Thread.currentThread().isInterrupted());
        }
    }
    public static void main(String[] args) {
```

线程被中断后park结束，中断状态还是中断，不会被还原

```
    }
}
```

我们看一下结果：

2、先unpark再park

```
public static class MyThread extends Thread {
        @Override
        public void run() {
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(getName() + " 进入线程");
            LockSupport.park();
            System.out.println(" 运行结束");
        }
    }
```

我们只需在park之前先休眠1秒钟，这样可以确保unpark先执行。