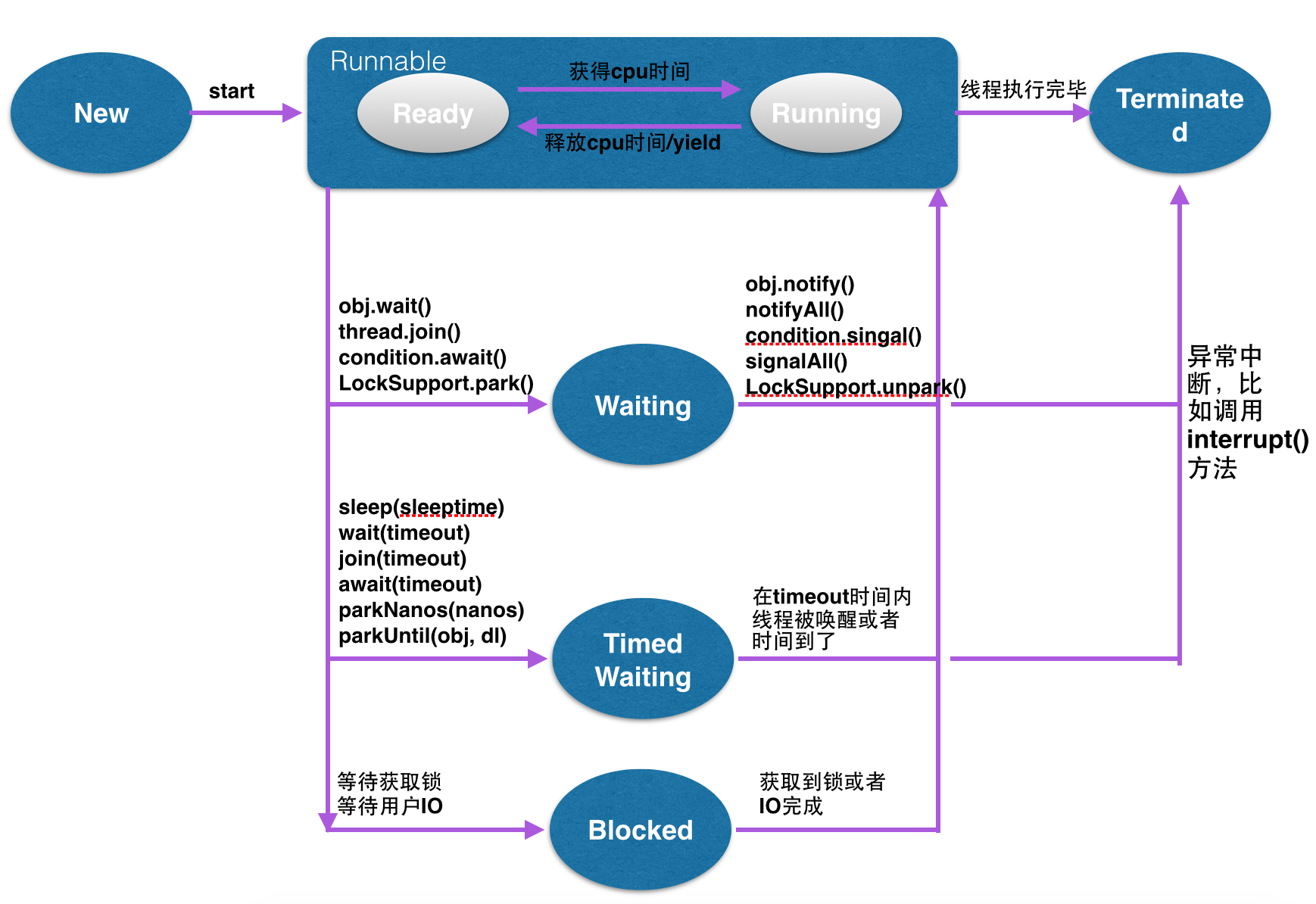
**基础题目**

**Java线程的状态**

 以下内容主要总结自《Java多线程编程核心技术》，不定时补充更新。

**一、线程的状态**

    Java中，线程的状态有以下6类：NEW, RUNNABLE, BLOCKED, WAITING, TIMED\_WAITING, TERMINATED。各状态之间的关系可用下图表示：



**二、常用方法介绍**

1、thread.start()和thread.run()的区别

1 public static void main(String[] args) {

2 Thread t = new Thread();

3 t.start();

4 System.out.println("main end");

5 }

调用start()方法启动线程t，t线程的状态会从New -> Runnable，t线程和main主线程同时执行。

如果把t.start()改成t.run()，则是普通的调用方法，同步执行，System.out.println("main end")语句必须等t.run()方法执行完毕之后才能执行。

注意：t.run()方法不会改变线程t的状态，也就是说线程没有启动。

2、object.wait()和thread.sleep()的区别

private Object obj = new Object();

public void testMethod() throws InterruptedException {

synchronized (obj) {

obj.wait();

System.out.println("testMethod end");

}

}

wait()方法主要用在synchronized同步方法或者同步块中，意味着调用object.wait()之前必须先获取锁，调用wait()方法之后释放锁，线程进入waiting状态。常见用法如上所示。如果有其他线程通过调用object.notify()或者object.notifyAll()方法时，线程必须再次获取到obj锁，然后才能继续执行obj.wait()后的语句，即打印 "testMethod end"。obj.wait(timeout)方法类似，也需要先释放锁。

wait()方法是Object类的方法，而sleep(timeout)方法是Thread类的方法。线程调用sleep(timeout)方法，状态从runnable -> timed\_waiting，但是不释放锁。

3、interrupt()方法

    当线程调用interrupt()方法时，只是设置了线程的中断状态。 也就是说如果线程处于runnable或者blocked状态的时候，调用interrupt()方法并不会终止线程。于是，我想当然的理解如果线程处于waiting或者timed\_waiting状态时，调用interrupt方法会抛出异常，从而终止线程。

    然后发现错了。见如下代码：

private ReentrantLock lock = new ReentrantLock();

private Condition condition = lock.newCondition();

public void testMethod() {

try {

lock.lock();

System.out.println("wait begin");

condition.awaitUninterruptibly();

System.out.println("wait end");

} finally {

lock.unlock();

}

}    condition.awaitUninterruptibly()方法不需要捕获InterruptedException异常，意味着如果线程通过调用awaitUninterruptibly从而使得线程状态为waiting，并不会因为调用interrupt()方法而中断。实际测试，线程状态不响应interrupt方法，只有通过condition.singal或者singalAll才能唤醒线程。

    实际测试，wait(), wait(timeout), join(), sleep(timeout), await(), await(timeout)等方法都是可以被interrupt()方法中断的。

**三、synchronized关键字**

1、synchronized(object): 同步方法或者代码块，锁是一个对象。

2、synchronized(this): this指的是当前对象。

3、针对静态方法，比如synchronized public static void testMethod()，锁是当前的Class类。

4、如果代码抛出异常，锁自动释放。

# java 线程的几种状态

java thread的运行周期中, 有几种状态, 在 java.lang.Thread.State 中有详细定义和说明:

**NEW**状态是指线程刚创建, 尚未启动

**RUNNABLE** 状态是线程正在正常运行中, 当然可能会有某种耗时计算/IO等待的操作/CPU时间片切换等, 这个状态下发生的等待一般是其他系统资源, 而不是锁, Sleep等

**BLOCKED** 这个状态下, 是在多个线程有同步操作的场景, 比如正在等待另一个线程的synchronized 块的执行释放, 或者可重入的 synchronized块里别人调用wait() 方法, 也就是这里是线程在等待进入临界区

**WAITING** 这个状态下是指线程拥有了某个锁之后, 调用了他的wait方法, 等待其他线程/锁拥有者调用 notify / notifyAll 一遍该线程可以继续下一步操作, 这里要区分 BLOCKED 和 WATING 的区别, 一个是在临界点外面等待进入, 一个是在理解点里面wait等待别人notify, 线程调用了join方法 join了另外的线程的时候, 也会进入WAITING状态, 等待被他join的线程执行结束

**TIMED\_WAITING** 这个状态就是有限的(时间限制)的WAITING, 一般出现在调用wait(long), join(long)等情况下, 另外一个线程sleep后, 也会进入TIMED\_WAITING状态

**TERMINATED** 这个状态下表示 该线程的run方法已经执行完毕了, 基本上就等于死亡了(当时如果线程被持久持有, 可能不会被回收)

## 下面谈谈如何让线程进入以上几种状态:

1. **NEW**, 这个最简单了,

**static** **void** NEW() {

          Thread t = **new** Thread ();

         System. out.println(t.getState());

    }

输出NEW

2. **RUNNABLE**, 也简单, 让一个thread start, 同时代码里面不要sleep或者wait等

**private** **static** **void** RUNNABLE() {

         Thread t = **new** Thread(){

**public** **void** run(){

**for**(**int** i=0; i<Integer.MAX\_VALUE; i++){

                      System. out.println(i);

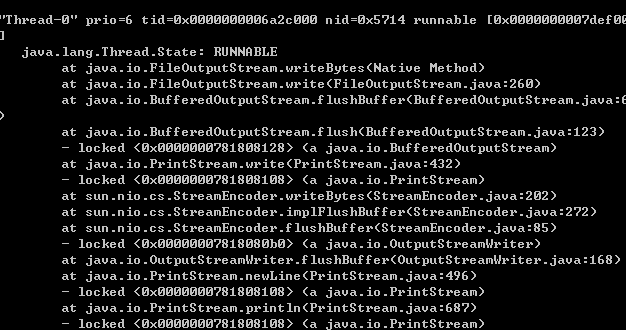
                 }

             }

         };

         t.start();

    }

[](http://www.jiacheo.org/wp-content/uploads/2013/04/71e94764e28c7f8bbd3ef91c1c0088b4.png)

3. **BLOCKED**, 这个就必须至少两个线程以上, 然后互相等待synchronized 块

**private** **static** **void** BLOCKED() {

**final** Object lock = **new** Object();

         Runnable run = **new** Runnable() {

              @Override

**public** **void** run() {

**for**(**int** i=0; i<Integer.MAX\_VALUE; i++){

**synchronized** (lock) {

                          System. out.println(i);

                      }

                 }

             }

         };

         Thread t1 = **new** Thread(run);

         t1.setName( “t1”);

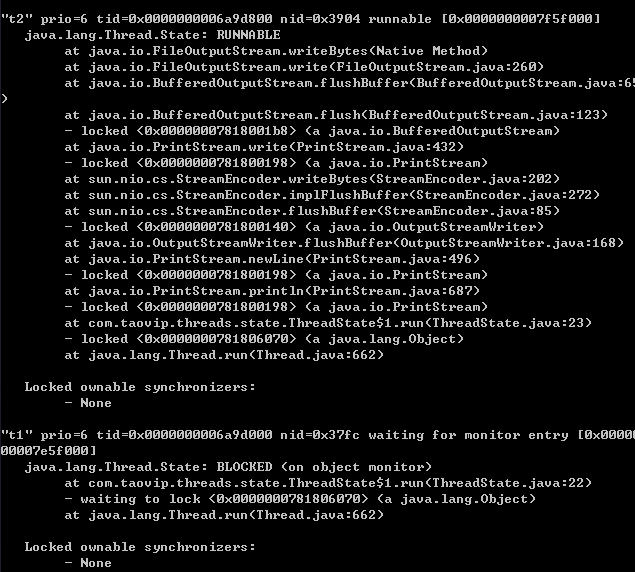
         Thread t2 = **new** Thread(run);

         t2.setName( “t2”);

         t1.start();

         t2.start();

    }

[](http://www.jiacheo.org/wp-content/uploads/2013/04/8e9ad1eadf9d38c0b6c8cb024cb36c0c.png)

这时候, 一个在RUNNABLE, 另一个就会在BLOCKED (等待另一个线程的 System.out.println.. 这是个IO操作, 属于系统资源, 不会造成WAITING等)

4. **WAITING**, 这个需要用到生产者消费者模型, 当生产者生产过慢的时候, 消费者就会等待生产者的下一次notify

**private** **static** **void** WAITING() {

**final** Object lock = **new** Object();

         Thread t1 = **new** Thread(){

              @Override

**public** **void** run() {

**int** i = 0;

**while**(**true** ){

**synchronized** (lock) {

**try** {

                               lock.wait();

                          } **catch** (InterruptedException e) {

                          }

                          System. out.println(i++);

                      }

                 }

             }

         };

         Thread t2 = **new** Thread(){

              @Override

**public** **void** run() {

**while**(**true** ){

**synchronized** (lock) {

**for**(**int** i = 0; i< 10000000; i++){

                              System. out.println(i);

                          }

                          lock.notifyAll();

                      }

                 }

             }

         };

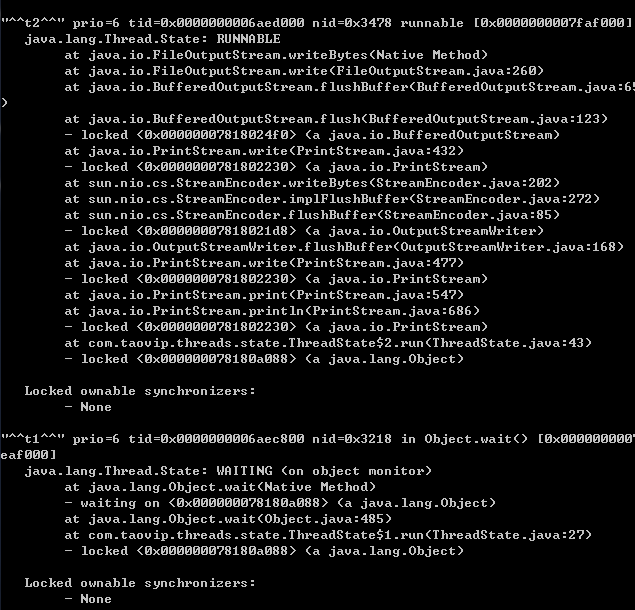
         t1.setName( “^^t1^^”);

         t2.setName( “^^t2^^”);

         t1.start();

         t2.start();

    }

[](http://www.jiacheo.org/wp-content/uploads/2013/04/b43a3d9b67bab266ffea4537fb043bba.png)

5. **TIMED\_WAITING**, 这个仅需要在4的基础上, 在wait方法加上一个时间参数进行限制就OK了.

把4中的synchronized 块改成如下就可以了.

**synchronized** (lock) {

**try** {

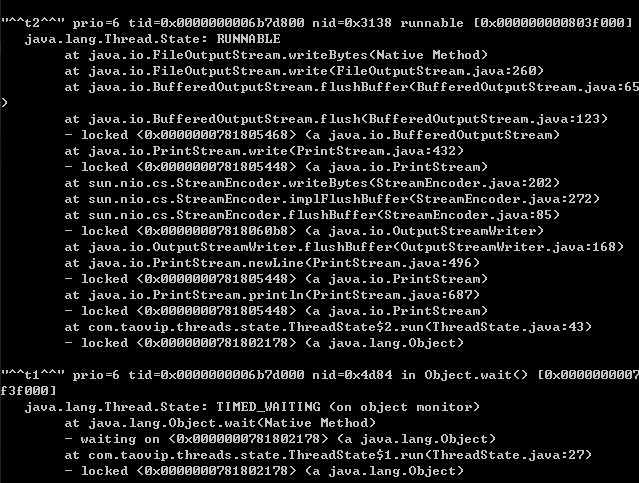
      lock.wait(60 \* 1000L);

   } **catch** (InterruptedException e) {

   }

   System. out .println(i++);

 }

[](http://www.jiacheo.org/wp-content/uploads/2013/04/88d9047d8a709c2d63c695bcf58a0297.png)

另外看stack的输出,  他叫 TIMED\_WAITING(on  object monitor) , 说明括号后面还有其他的情况, 比如sleep, 我们直接把t2的for循环改成sleep试试:

**synchronized** (lock) {

**try** {

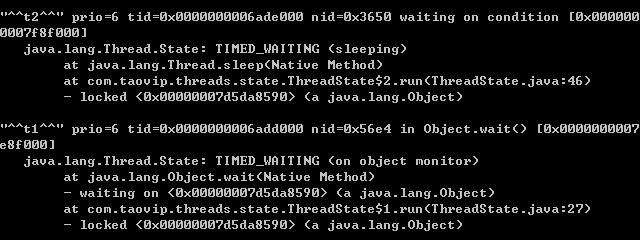
          sleep(30\*1000L);

    } **catch** (InterruptedException e) {

    }

    lock.notifyAll();

}

[](http://www.jiacheo.org/wp-content/uploads/2013/04/a37ef4c72c00e793f8b6c746d74fd4d9.png)

看到了吧, t2的state是 TIMED\_WAITING( sleeping),  而t1依然是on object monitor , 因为t1还是wait在等待t2 notify, 而t2是自己sleep

另外, join操作也是进入 on object monitor

6. **TERMINATED**, 这个状态只要线程结束了run方法, 就会进入了…

**private** **static** **void** TERMINATED() {

         Thread t1 = **new** Thread();

         t1.start();

         System. out.println(t1.getState());

**try** {

             Thread. sleep(1000L);

         } **catch** (InterruptedException e) {

         }

         System. out.println(t1.getState());

    }

输出:

RUNNABLE

TERMINATED

由于线程的start方法是异步启动的, 所以在其执行后立即获取状态有可能才刚进入RUN方法且还未执行完毕

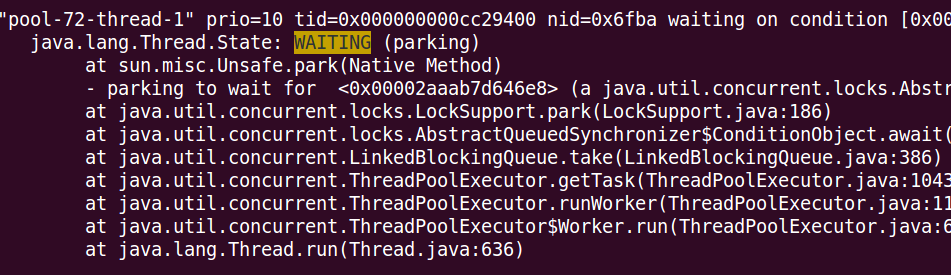
废话了这么多, **了解线程的状态究竟有什么用**?

所以说这是个钓鱼贴么…

好吧, 一句话, 在找到系统中的潜在性能瓶颈有作用.

当java系统运行慢的时候, 我们想到的应该先找到性能的瓶颈, 而**jstack**等工具, 通过jvm当前的stack可以看到当前整个vm所有线程的状态, 当我们看到一个线程状态经常处于

WAITING 或者 BLOCKED的时候, 要小心了, 他可能在等待资源经常没有得到释放(当然, 线程池的调度用的也是各种队列各种锁, 要区分一下, 比如下图)

[](http://www.jiacheo.org/wp-content/uploads/2013/04/6db341bbd7680bbc2e6ae37a66329397.png)

这是个经典的并发包里面的线程池, 其调度队列用的是LinkedBlockingQueue, 执行take的时候会block住, 等待下一个任务进入队列中, 然后进入执行, 这种理论上不是系统的性能瓶颈, 找瓶颈一般**先找自己的代码**stack,**再去排查那些开源的组件**/JDK的问题

排查问题的几个思路:

0. 如何跟踪一个线程?

看到上面的stack输出没有, 第一行是内容是 threadName priority tid nid desc

更过跟踪tid, nid 都可以唯一找到该线程.

1. 发现有线程进入BLOCK, 而且持续好久, 这说明性能瓶颈存在于synchronized块中, 因为他一直block住, 进不去, 说明另一个线程一直没有处理好, 也就这个synchronized块中处理速度比较慢, 然后再深入查看. 当然也有可能同时block的线程太多, 排队太久造成.

2. 发现有线程进入WAITING, 而且持续好久, 说明性能瓶颈存在于触发notify的那段逻辑. 当然还有就是同时WAITING的线程过多, 老是等不到释放.

3. 线程进入TIME\_WAITING 状态且持续好久的, 跟2的排查方式一样.

上面的黑底白字截图都是通过jstack打印出来的, 可以直接定位到你想知道的线程的执行栈, 这对java性能瓶颈的分析是有极大作用的.

NOTE: 上面所有代码都是为了跟踪线程的状态而写的, 千万不要在线上应用中这么写…