黄小斜学Java

微信公众号【黄小斜学Java】 作者是蚂蚁金服Java工程师，专注分享Java领域干货，不限于BAT面试，算法、计算机基础、数据库、分布式、spring全家桶、微服务、高并发、JVM、Docker容器，ELK、大数据等相关知识，希望我们可以一起进步。

订阅 管理

java 线程的几种状态

```
Object.wait with no timeout    Thread.sleep
Thread.join with no timeout    Object.wait with timeout
LockSupport.park               Thread.join with timeout
                               LockSupport.parkNanos
                               LockSupport.parkUntil
```

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:

公告

昵称： 黄小斜学Java
园龄： 4年7个月
粉丝： 202
关注： 2
+加关注

阅读排行榜

1. Java工程师修炼之路（校招总结）(53417)
2. java 线程的几种状态(19347)
3. MySQL数据库事务详解(16044)
4. 不得不提的volatile及指令重排序(happen-before)(15420)

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();
    }
```

```
"Thread-0" prio=6 tid=0x0000000006a2c000 nid=0x5714 runnable [0x0000000007def00
0]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileOutputStream.writeBytes(Native Method)
        at java.io.FileOutputStream.write(FileOutputStream.java:260)
        at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:8
2)
        at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:123)
        - locked <0x0000000781808128> (a java.io.BufferedOutputStream)
        at java.io.PrintStream.write(PrintStream.java:432)
        - locked <0x0000000781808108> (a java.io.PrintStream)
        at sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:202)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:272)
        at sun.nio.cs.StreamEncoder.flushBuffer(StreamEncoder.java:85)
        - locked <0x00000007818080b0> (a java.io.OutputStreamWriter)
        at java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:168)
        at java.io.PrintStream.newLine(PrintStream.java:496)
        - locked <0x0000000781808108> (a java.io.PrintStream)
        at java.io.PrintStream.println(PrintStream.java:687)
        - locked <0x0000000781808108> (a java.io.PrintStream)
```

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

5. 我的秋招经验分享（已拿BAT头条网易滴滴）(14838)

```
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();

}
```

```
"t2" prio=6 tid=0x0000000006a9d800 nid=0x3904 runnable [0x0000000007f5f000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileOutputStream.writeBytes(Native Method)
        at java.io.FileOutputStream.write(FileOutputStream.java:260)
        at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:65
)
        at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:123)
        - locked <0x00000007818001b8> (a java.io.BufferedOutputStream)
        at java.io.PrintStream.write(PrintStream.java:432)
        - locked <0x0000000781800198> (a java.io.PrintStream)
        at sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:202)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:272)
        at sun.nio.cs.StreamEncoder.flushBuffer(StreamEncoder.java:85)
        - locked <0x0000000781800140> (a java.io.OutputStreamWriter)
        at java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:168)
        at java.io.PrintStream.newLine(PrintStream.java:496)
        - locked <0x0000000781800198> (a java.io.PrintStream)
        at java.io.PrintStream.println(PrintStream.java:687)
        - locked <0x0000000781800198> (a java.io.PrintStream)
        at com.taovip.threads.state.ThreadState$1.run(ThreadState.java:23)
        - locked <0x0000000781806070> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:662)

   Locked ownable synchronizers:
        - None

"t1" prio=6 tid=0x0000000006a9d000 nid=0x37fc waiting for monitor entry [0x00000
00007e5f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.taovip.threads.state.ThreadState$1.run(ThreadState.java:22)
        - waiting to lock <0x0000000781806070> (a java.lang.Object)
        at java.lang.Thread.run(Thread.java:662)

   Locked ownable synchronizers:
        - None
```

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();
}
```

```
"^^t2^^" prio=6 tid=0x0000000006aed000 nid=0x3478 runnable [0x0000000007faf000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileOutputStream.writeBytes(Native Method)
        at java.io.FileOutputStream.write(FileOutputStream.java:260)
        at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:65
)
        at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:123)
        - locked <0x00000007818024f0> (a java.io.BufferedOutputStream)
        at java.io.PrintStream.write(PrintStream.java:432)
        - locked <0x0000000781802230> (a java.io.PrintStream)
        at sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:202)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:272)
        at sun.nio.cs.StreamEncoder.flushBuffer(StreamEncoder.java:85)
        - locked <0x00000007818021d8> (a java.io.OutputStreamWriter)
        at java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:168)
        at java.io.PrintStream.write(PrintStream.java:477)
        - locked <0x0000000781802230> (a java.io.PrintStream)
        at java.io.PrintStream.print(PrintStream.java:547)
        at java.io.PrintStream.println(PrintStream.java:686)
        - locked <0x0000000781802230> (a java.io.PrintStream)
        at com.taovip.threads.state.ThreadState$2.run(ThreadState.java:43)
        - locked <0x000000078180a088> (a java.lang.Object)

   Locked ownable synchronizers:
        - None

"^^t1^^" prio=6 tid=0x0000000006aec800 nid=0x3218 in Object.wait() [0x000000000
eaf000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x000000078180a088> (a java.lang.Object)
        at java.lang.Object.wait(Object.java:485)
        at com.taovip.threads.state.ThreadState$1.run(ThreadState.java:27)
        - locked <0x000000078180a088> (a java.lang.Object)

   Locked ownable synchronizers:
        - None
```

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++);
}
```

```
"^^t2^^" prio=6 tid=0x0000000006b7d800 nid=0x3138 runnable [0x000000000803f000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileOutputStream.writeBytes(Native Method)
        at java.io.FileOutputStream.write(FileOutputStream.java:260)
        at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:65
)
        at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:123)
        - locked <0x0000000781805468> (a java.io.BufferedOutputStream)
        at java.io.PrintStream.write(PrintStream.java:432)
        - locked <0x0000000781805448> (a java.io.PrintStream)
        at sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:202)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:272)
        at sun.nio.cs.StreamEncoder.flushBuffer(StreamEncoder.java:85)
        - locked <0x00000007818060b8> (a java.io.OutputStreamWriter)
        at java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:168)
        at java.io.PrintStream.newLine(PrintStream.java:496)
        - locked <0x0000000781805448> (a java.io.PrintStream)
        at java.io.PrintStream.println(PrintStream.java:687)
        - locked <0x0000000781805448> (a java.io.PrintStream)
        at com.taovip.threads.state.ThreadState$2.run(ThreadState.java:43)
        - locked <0x0000000781802178> (a java.lang.Object)

"^^t1^^" prio=6 tid=0x0000000006b7d000 nid=0x4d84 in Object.wait() [0x0000000007
f3f000]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x0000000781802178> (a java.lang.Object)
        at com.taovip.threads.state.ThreadState$1.run(ThreadState.java:27)
        - locked <0x0000000781802178> (a java.lang.Object)
```

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();
}
```

```
"^^t2^^" prio=6 tid=0x0000000006ade000 nid=0x3650 waiting on condition [0x000000
0007f8f000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
        at com.taovip.threads.state.ThreadState$2.run(ThreadState.java:46)
        - locked <0x00000007d5da8590> (a java.lang.Object)

"^^t1^^" prio=6 tid=0x0000000006add000 nid=0x56e4 in Object.wait() [0x0000000007
e8f000]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x00000007d5da8590> (a java.lang.Object)
        at com.taovip.threads.state.ThreadState$1.run(ThreadState.java:27)
        - locked <0x00000007d5da8590> (a java.lang.Object)
```

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)

```
"pool-72-thread-1" prio=10 tid=0x000000000cc29400 nid=0x6fba waiting on condition [0x00
   java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00002aaab7d646e8> (a java.util.concurrent.locks.Abstr
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:186)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(
        at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:386)
        at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1043
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:11
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:6
        at java.lang.Thread.run(Thread.java:636)
```

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...

javajstackthread

原创文章转载请注明：转载自：java 线程的几种状态

> 微信公众号【黄小斜】大厂程序员，互联网行业新知，终身学习践行者。关注后回复「Java」、「Python」、「C++」、「大数据」、「机器学习」、「算法」、「AI」、「Android」、「前端」、「iOS」、「考研」、「BAT」、「校招」、「笔试」、「面试」、「面经」、「计算机基础」、「LeetCode」 等关键字可以获取对应的免费学习资料。

标签: java

好文要顶 关注我 收藏该文

黄小斜学Java
关注 - 2
粉丝 - 202
+加关注

1 0

« 上一篇： C++中 引用&与取地址&的区别
» 下一篇： Java线程之 InterruptedException 异常

posted @ 2017-02-19 11:15 黄小斜学Java 阅读(19347) 评论(2) 编辑 收藏 举报

刷新评论 刷新页面 返回顶部

登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

【推荐】百度智能云2021普惠上云节：新用户首购云服务器低至0.7折
【推荐】阿里云云大使特惠：新用户购ECS服务器1核2G最低价87元/年

【推荐】大型组态、工控、仿真、CAD\GIS 50万行VC++源码免费下载!
【推广】园子与爱卡汽车爱宝险合作，随手就可以买一份的百万医疗保险

编辑推荐:
· 技术从业者的未来（三）
· .NET 6 全新指标 System.Diagnostics.Metrics 介绍
· [WPF] 使用 MVVM Toolkit 构建 MVVM 程序
· 使用 C# 下载文件的十八般武艺
· 我用 10000 张图片合成我们美好的瞬间

最新新闻:
· 虚拟更衣室火了！Revery.ai 用计算机视觉来增强购物体验（2021-08-24 20:40）
· 最新 AI 可发现伪造艺术品（2021-08-24 20:30）
· 重估百度智能交通野心（2021-08-24 20:20）
· 腾讯云联合慧眼科技发布腾慧飞瞳AI质检仪，攻克质检自动化、智能化难题（2021-08-24 20:05）
· 腾讯在重庆打造首个100%采用可再生能源大型数据中心（2021-08-24 19:53）
» 更多新闻...

Copyright © 2021 黄小斜学Java
Powered by .NET 5.0 on Kubernetes