使用 [TDA 工具](http://java.net/projects/tda/" \t "http://www.cnblogs.com/zhengyun_ustc/archive/2013/03/18/_blank)，看到大量 Java Thread State 的第一反应是：

**1，线程状态为“waiting for monitor entry”：**

意味着它 **在等待进入一个临界区**，所以它在”Entry Set“队列中等待。

此时线程状态一般都是 Blocked：

* java.lang.Thread.State: **BLOCKED** (on object monitor)

**2，线程状态为“waiting on condition”：**

说明它**在等待另一个条件的发生，来把自己唤醒**，或者干脆它是调用了 sleep(N)。

此时线程状态大致为以下几种：

* java.lang.Thread.State: **WAITING** (parking)：一直等那个条件发生；
* java.lang.Thread.State: **TIMED\_WAITING** (parking或sleeping)：定时的，那个条件不到来，也将定时唤醒自己。

**3，如果大量线程在“waiting for monitor entry”：**

可能是一个全局锁阻塞住了大量线程。

如果短时间内打印的 thread dump 文件反映，随着时间流逝，waiting for monitor entry 的线程越来越多，没有减少的趋势，可能意味着**某些线程在临界区里呆的时间太长了，以至于越来越多新线程迟迟无法进入临界区**。

**4，如果大量线程在“waiting on condition”：**

可能是它们又跑去获取第三方资源，**尤其是第三方网络资源，迟迟获取不到Response**，导致大量线程进入等待状态。

所以如果你发现有大量的线程都处在 Wait on condition，从线程堆栈看，正等待网络读写**，这可能是一个网络瓶颈的征兆**，因为网络阻塞导致线程无法执行。

**线程状态为“in Object.wait()”：**

说明它**获得了监视器之后，又调用了 java.lang.Object.wait() 方法**。

每个 Monitor在某个时刻，只能被一个线程拥有，该线程就是 “Active Thread”，而其它线程都是 “Waiting Thread”，分别在两个队列 “ Entry Set”和 “Wait Set”里面等候。在 “Entry Set”中等待的线程状态是 “Waiting for monitor entry”，而在 “Wait Set”中等待的线程状态是 “in Object.wait()”。

当线程获得了 Monitor，如果发现线程继续运行的条件没有满足，它则调用对象（一般就是被 synchronized 的对象）的 wait() 方法，放弃了 Monitor，进入 “Wait Set”队列。

此时线程状态大致为以下几种：

* java.lang.Thread.State: **TIMED\_WAITING** (on object monitor)；
* java.lang.Thread.State: **WAITING** (on object monitor)；

一般都是RMI相关线程（RMI RenewClean、 GC Daemon、RMI Reaper），GC线程（Finalizer），引用对象垃圾回收线程（Reference Handler）等系统线程处于这种状态。

图1 A Java Monitor

示范一：

下面这个线程在等待这个锁 **0x00000000fe7e3b50**，等待进入临界区：

"RMI TCP Connection(64896)-172.16.52.118" daemon prio=10 tid=0x00000000405a6000 nid=0x68fe waiting for monitor entry [0x00007f2be65a3000]

   java.lang.Thread.State: BLOCKED (on object monitor)

at com.xyz.goods.service.impl.GoodsServiceImpl.**findChanellGoodsCountWithCache**(GoodsServiceImpl.java:1734)

- **waiting to lock <0x00000000fe7e3b50> (a java.lang.String)**

那么谁持有这个锁呢？

是另一个先调用了 findChanellGoodsCountWithCache 函数的线程：

"RMI TCP Connection(64878)-172.16.52.117" daemon prio=10 tid=0x0000000040822000 nid=0x6841 runnable [0x00007f2be76b3000]

   java.lang.Thread.State: RUNNABLE

at java.net.SocketInputStream.socketRead0(Native Method)

at java.net.SocketInputStream.read(SocketInputStream.java:129)

at java.io.BufferedInputStream.fill(BufferedInputStream.java:218)

at java.io.BufferedInputStream.read1(BufferedInputStream.java:258)

at java.io.BufferedInputStream.read(BufferedInputStream.java:317)

- locked <0x00000000af4ed638> (a java.io.BufferedInputStream)

at org.bson.io.Bits.readFully(Bits.java:35)

at org.bson.io.Bits.readFully(Bits.java:28)

at com.mongodb.Response.<init>(Response.java:35)

at com.mongodb.DBPort.go(DBPort.java:110)

- locked <0x00000000af442d48> (a com.mongodb.DBPort)

at com.mongodb.DBPort.go(DBPort.java:75)

- locked <0x00000000af442d48> (a com.mongodb.DBPort)

at com.mongodb.DBPort.call(DBPort.java:65)

at com.mongodb.DBTCPConnector.call(DBTCPConnector.java:202)

at com.mongodb.DBApiLayer$MyCollection.\_\_find(DBApiLayer.java:296)

at com.mongodb.DB.command(DB.java:152)

at com.mongodb.DBCollection.getCount(DBCollection.java:760)

at com.mongodb.DBCollection.getCount(DBCollection.java:731)

at com.mongodb.DBCollection.count(DBCollection.java:697)

at com.xyz.goods.manager.MongodbManager.count(MongodbManager.java:202)

at com.xyz.goods.service.impl.GoodsServiceImpl.findChanellGoodsCount(GoodsServiceImpl.java:1787)

at com.xyz.goods.service.impl.GoodsServiceImpl.**findChanellGoodsCountWithCache**(GoodsServiceImpl.java:1739)

- **locked <0x00000000fe7e3b50> (a java.lang.String)**

**示范二：**

等待另一个条件发生来将自己唤醒：

"RMI TCP Connection(idle)" daemon prio=10 tid=0x00007fd50834e800 nid=0x56b2 **waiting on condition** [0x00007fd4f1a59000]

   java.lang.Thread.State: **TIMED\_WAITING (parking)**

at sun.misc.Unsafe.park(Native Method)

- **parking to wait for  <0x00000000acd84de8>** (a java.util.concurrent.SynchronousQueue$TransferStack)

at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:198)

at java.util.concurrent.SynchronousQueue$TransferStack.awaitFulfill(SynchronousQueue.java:424)

at java.util.concurrent.SynchronousQueue$TransferStack.transfer(SynchronousQueue.java:323)

at java.util.concurrent.SynchronousQueue.poll(SynchronousQueue.java:874)

at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:945)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:907)

at java.lang.Thread.run(Thread.java:662)

1）“**TIMED\_WAITING (parking)**”中的 timed\_waiting 指等待状态，但这里指定了时间，到达指定的时间后自动退出等待状态；parking指线程处于挂起中。

2）“**waiting on condition**”需要与堆栈中的“**parking to wait for  <0x00000000acd84de8>** (a java.util.concurrent.SynchronousQueue$TransferStack)” 结合来看。首先，本线程肯定是在等待某个条件的发生，来把自己唤醒。其次，SynchronousQueue 并不是一个队列，只是线程之间移交信息的机制，当我们把一个元素放入到 SynchronousQueue 中时必须有另一个线程正在等待接受移交的任务，因此这就是本线程在等待的条件。

**示范三：**

"RMI RenewClean-[172.16.50.182:4888]" daemon prio=10 tid=0x0000000040d2c800 nid=0x97e **in Object.wait()** [0x00007f9ccafd0000]

   java.lang.Thread.State: **TIMED\_WAITING (on object monitor)**

at java.lang.Object.wait(Native Method)

- waiting on <0x0000000799b032d8> (a java.lang.ref.ReferenceQueue$Lock)

at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:118)

- locked <0x0000000799b032d8> (a java.lang.ref.ReferenceQueue$Lock)

at sun.rmi.transport.DGCClient$EndpointEntry$RenewCleanThread.run(DGCClient.java:516)

at java.lang.Thread.run(Thread.java:662)

**参考资源：**

1）CUBRID，2012，[How to Analyze Java Thread Dumps](http://www.cubrid.org/blog/dev-platform/how-to-analyze-java-thread-dumps/)；

2）郑昀，2013，[三个实例演示Java THread Dump日志分析](http://www.cnblogs.com/zhengyun_ustc/archive/2013/01/06/dumpanalysis.html)；