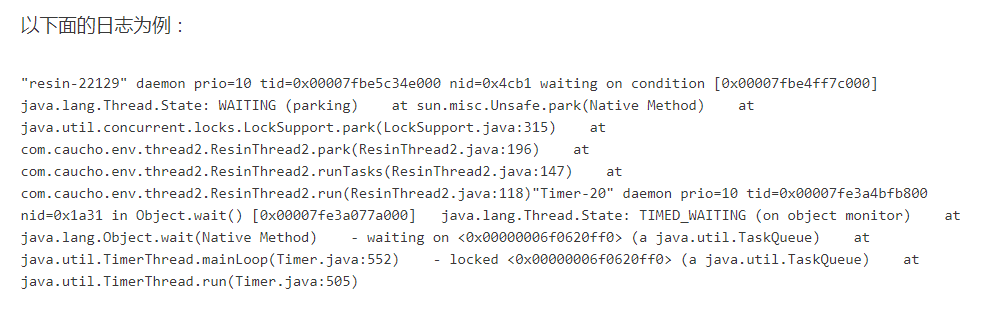
Thread Dump日志的线程信息



以上依次是：

1. "resin-22129"**线程名称：**如果使用 java.lang.Thread 类生成一个线程的时候，线程名称为 Thread-(数字) 的形式，这里是resin生成的线程；
2. daemon**线程类型：**线程分为守护线程 (daemon) 和非守护线程 (non-daemon) 两种，通常都是守护线程；
3. prio=10**线程优先级：**默认为5，数字越大优先级越高；
4. tid=0x00007fbe5c34e000**JVM线程的id：**JVM内部线程的唯一标识，通过 java.lang.Thread.getId()获取，通常用自增的方式实现；
5. nid=0x4cb1**系统线程id：**对应的系统线程id（Native Thread ID)，可以通过 top 命令进行查看，现场id是十六进制的形式；
6. waiting on condition**系统线程状态：**这里是系统的线程状态，具体的含义见下面 系统线程状态 部分；
7. [0x00007fbe4ff7c000]**起始栈地址：**线程堆栈调用的其实内存地址；
8. java.lang.Thread.State: WAITING (parking)**JVM线程状态：**这里标明了线程在代码级别的状态，详细的内容见下面的 JVM线程运行状态 部分。
9. **线程调用栈信息：**下面就是当前线程调用的详细栈信息，用于代码的分析。堆栈信息应该从下向上解读，因为程序调用的顺序是从下向上的。

系统线程状态 (Native Thread Status)

系统线程有如下状态：

deadlock

死锁线程，一般指多个线程调用期间进入了相互资源占用，导致一直等待无法释放的情况。

runnable

一般指该线程正在执行状态中，该线程占用了资源，正在处理某个操作，如通过SQL语句查询数据库、对某个文件进行写入等。

blocked

线程正处于阻塞状态，指当前线程执行过程中，所需要的资源长时间等待却一直未能获取到，被容器的线程管理器标识为阻塞状态，可以理解为等待资源超时的线程。

waiting on condition

线程正处于等待资源或等待某个条件的发生，具体的原因需要结合下面堆栈信息进行分析。

（1）如果堆栈信息明确是应用代码，则证明该线程正在等待资源，一般是大量读取某种资源且该资源采用了资源锁的情况下，线程进入等待状态，等待资源的读取，或者正在等待其他线程的执行等。

（2）如果发现有大量的线程都正处于这种状态，并且堆栈信息中得知正等待网络读写，这是因为网络阻塞导致线程无法执行，很有可能是一个网络瓶颈的征兆：

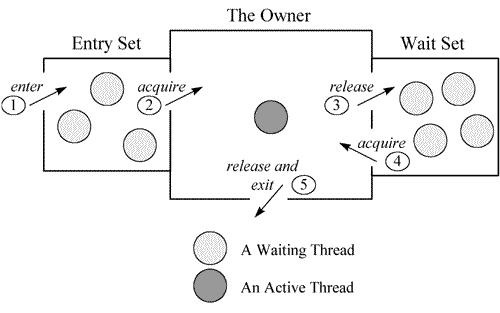
1. 网络非常繁忙，几乎消耗了所有的带宽，仍然有大量数据等待网络读写；
2. 网络可能是空闲的，但由于路由或防火墙等原因，导致包无法正常到达；

所以一定要结合系统的一些性能观察工具进行综合分析，比如netstat统计单位时间的发送包的数量，看是否很明显超过了所在网络带宽的限制；观察CPU的利用率，看系统态的CPU时间是否明显大于用户态的CPU时间。这些都指向由于网络带宽所限导致的网络瓶颈。

（3）还有一种常见的情况是该线程在 sleep，等待 sleep 的时间到了，将被唤醒。

waiting for monitor entry 或 in Object.wait()

Moniter 是Java中用以实现线程之间的互斥与协作的主要手段，它可以看成是对象或者class的锁，每个对象都有，也仅有一个 Monitor。



从上图可以看出，每个Monitor在某个时刻只能被一个线程拥有，该线程就是 "Active Thread"，而其他线程都是 "Waiting Thread"，分别在两个队列 "Entry Set"和"Waint Set"里面等待。其中在 "Entry Set" 中等待的线程状态是 waiting for monitor entry，在 "Wait Set" 中等待的线程状态是 in Object.wait()。

（1）"Entry Set"里面的线程。

我们称被 synchronized 保护起来的代码段为临界区，对应的代码如下：

synchronized(obj){}

当一个线程申请进入临界区时，它就进入了 "Entry Set" 队列中，这时候有两种可能性：

1. 该Monitor不被其他线程拥有，"Entry Set"里面也没有其他等待的线程。本线程即成为相应类或者对象的Monitor的Owner，执行临界区里面的代码；此时在Thread Dump中显示线程处于 "Runnable" 状态。
2. 该Monitor被其他线程拥有，本线程在 "Entry Set" 队列中等待。此时在Thread Dump中显示线程处于 "waiting for monity entry" 状态。

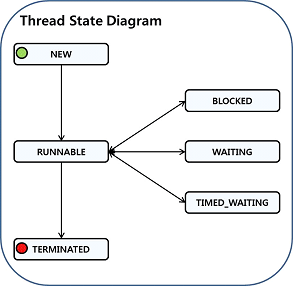
临界区的设置是为了保证其内部的代码执行的原子性和完整性，但因为临界区在任何时间只允许线程串行通过，这和我们使用多线程的初衷是相反的。如果在多线程程序中大量使用synchronized，或者不适当的使用它，会造成大量线程在临界区的入口等待，造成系统的性能大幅下降。如果在Thread Dump中发现这个情况，应该审视源码并对其进行改进。

（2）"Wait Set"里面的线程

当线程获得了Monitor，进入了临界区之后，如果发现线程继续运行的条件没有满足，它则调用对象（通常是被synchronized的对象）的wait()方法，放弃Monitor，进入 "Wait Set"队列。只有当别的线程在该对象上调用了 notify()或者notifyAll()方法，"Wait Set"队列中的线程才得到机会去竞争，但是只有一个线程获得对象的Monitor，恢复到运行态。"Wait Set"中的线程在Thread Dump中显示的状态为 in Object.wait()。通常来说，

通常来说，当CPU很忙的时候关注 Runnable 状态的线程，反之则关注 waiting for monitor entry 状态的线程。

JVM线程运行状态 (JVM Thread Status)



在 [java.lang.Thread.State](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html) 中定义了线程的状态：

[NEW](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#NEW)

至今尚未启动的线程的状态。线程刚被创建，但尚未启动。

[RUNNABLE](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#RUNNABLE)

可运行线程的线程状态。线程正在JVM中执行，有可能在等待操作系统中的其他资源，比如处理器。

[BLOCKED](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#BLOCKED)

受阻塞并且正在等待监视器的某一线程的线程状态。处于受阻塞状态的某一线程正在等待监视器锁，以便进入一个同步的块/方法，或者在调用 Object.wait 之后再次进入同步的块/方法。

在Thread Dump日志中通常显示为 *java.lang.Thread.State: BLOCKED (on object monitor)* 。

[WAITING](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#WAITING)

某一等待线程的线程状态。线程正在无期限地等待另一个线程来执行某一个特定的操作，线程因为调用下面的方法之一而处于等待状态：

1. 不带超时的 Object.wait 方法，日志中显示为 *java.lang.Thread.State: WAITING (on object monitor)*
2. 不带超时的 Thread.join 方法
3. LockSupport.park 方法，日志中显示为 *java.lang.Thread.State: WAITING (parking)*

[TIMED\_WAITING](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#TIMED_WAITING)

指定了等待时间的某一等待线程的线程状态。线程正在等待另一个线程来执行某一个特定的操作，并设定了指定等待的时间，线程因为调用下面的方法之一而处于定时等待状态：

1. Thread.sleep 方法
2. 指定超时值的 Object.wait 方法
3. 指定超时值的 Thread.join 方法
4. LockSupport.parkNanos
5. LockSupport.parkUntil

[TERMINATED](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html#TERMINATED)

线程处于终止状态。

根据Java Doc中的说明，在给定的时间上，一个只能处于上述的一种状态之中，并且这些状态都是JVM的状态，跟操作系统中的线程状态无关。

线程状态样例

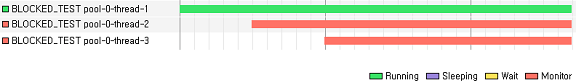
等待状态样例

CC4D5C78191041C0976DC392BAB298F5.png

"IoWaitThread" prio=6 tid=0x0000000007334800 nid=0x2b3c waiting on condition [0x000000000893f000] java.lang.Thread.State: WAITING (parking) at sun.misc.Unsafe.park(Native Method) - parking to wait for <0x00000007d5c45850> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject) at java.util.concurrent.locks.LockSupport.park(LockSupport.java:156) at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:1987) at java.util.concurrent.LinkedBlockingDeque.takeFirst(LinkedBlockingDeque.java:440) at java.util.concurrent.LinkedBlockingDeque.take(LinkedBlockingDeque.java:629) at com.nbp.theplatform.threaddump.ThreadIoWaitState$IoWaitHandler2.run(ThreadIoWaitState.java:89) at java.lang.Thread.run(Thread.java:662)

上面例子中，IoWaitThread 线程保持等待状态并从 LinkedBlockingQueue 接收消息，如果 LinkedBlockingQueue 一直没有消息，该线程的状态将不会改变。

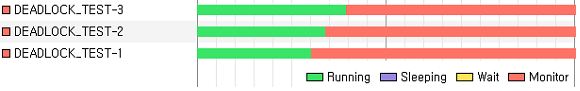
阻塞状态样例



"BLOCKED\_TEST pool-1-thread-1" prio=6 tid=0x0000000006904800 nid=0x28f4 runnable [0x000000000785f000] java.lang.Thread.State: RUNNABLE at java.io.FileOutputStream.writeBytes(Native Method) at java.io.FileOutputStream.write(FileOutputStream.java:282) at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:65) at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:123) - locked <0x0000000780a31778> (a java.io.BufferedOutputStream) at java.io.PrintStream.write(PrintStream.java:432) - locked <0x0000000780a04118> (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 <0x0000000780a040c0> (a java.io.OutputStreamWriter) at java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:168) at java.io.PrintStream.newLine(PrintStream.java:496) - locked <0x0000000780a04118> (a java.io.PrintStream) at java.io.PrintStream.println(PrintStream.java:687) - locked <0x0000000780a04118> (a java.io.PrintStream) at com.nbp.theplatform.threaddump.ThreadBlockedState.monitorLock(ThreadBlockedState.java:44) - locked <0x0000000780a000b0> (a com.nbp.theplatform.threaddump.ThreadBlockedState) at com.nbp.theplatform.threaddump.ThreadBlockedState$1.run(ThreadBlockedState.java:7) at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908) at java.lang.Thread.run(Thread.java:662) Locked ownable synchronizers: - <0x0000000780a31758> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)"BLOCKED\_TEST pool-1-thread-2" prio=6 tid=0x0000000007673800 nid=0x260c waiting for monitor entry [0x0000000008abf000] java.lang.Thread.State: BLOCKED (on object monitor) at com.nbp.theplatform.threaddump.ThreadBlockedState.monitorLock(ThreadBlockedState.java:43) - waiting to lock <0x0000000780a000b0> (a com.nbp.theplatform.threaddump.ThreadBlockedState) at com.nbp.theplatform.threaddump.ThreadBlockedState$2.run(ThreadBlockedState.java:26) at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886) at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908) at java.lang.Thread.run(Thread.java:662) Locked ownable synchronizers: - <0x0000000780b0c6a0> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)"BLOCKED\_TEST pool-1-thread-3" prio=6 tid=0x00000000074f5800 nid=0x1994 waiting for monitor entry [0x0000000008bbf000] java.lang.Thread.State: BLOCKED (on object monitor) at com.nbp.theplatform.threaddump.ThreadBlockedState.monitorLock(ThreadBlockedState.java:42) - waiting to lock <0x0000000780a000b0> (a com.nbp.theplatform.threaddump.ThreadBlockedState) at com.nbp.theplatform.threaddump.ThreadBlockedState$3.run(ThreadBlockedState.java:34) at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908) at java.lang.Thread.run(Thread.java:662) Locked ownable synchronizers: - <0x0000000780b0e1b8> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)

在上面的例子中，BLOCKED\_TEST pool-1-thread-1 线程占用了 <0x0000000780a000b0> 锁，然而 BLOCKED\_TEST pool-1-thread-2 和 BLOCKED\_TEST pool-1-thread-3 threads 正在等待获取锁。

死锁状态样例



"DEADLOCK\_TEST-1" daemon prio=6 tid=0x000000000690f800 nid=0x1820 waiting for monitor entry [0x000000000805f000] java.lang.Thread.State: BLOCKED (on object monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.goMonitorDeadlock(ThreadDeadLockState.java:197) - waiting to lock <0x00000007d58f5e60> (a com.nbp.theplatform.threaddump.ThreadDeadLockState$Monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.monitorOurLock(ThreadDeadLockState.java:182) - locked <0x00000007d58f5e48> (a com.nbp.theplatform.threaddump.ThreadDeadLockState$Monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.run(ThreadDeadLockState.java:135) Locked ownable synchronizers: - None"DEADLOCK\_TEST-2" daemon prio=6 tid=0x0000000006858800 nid=0x17b8 waiting for monitor entry [0x000000000815f000] java.lang.Thread.State: BLOCKED (on object monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.goMonitorDeadlock(ThreadDeadLockState.java:197) - waiting to lock <0x00000007d58f5e78> (a com.nbp.theplatform.threaddump.ThreadDeadLockState$Monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.monitorOurLock(ThreadDeadLockState.java:182) - locked <0x00000007d58f5e60> (a com.nbp.theplatform.threaddump.ThreadDeadLockState$Monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.run(ThreadDeadLockState.java:135) Locked ownable synchronizers: - None"DEADLOCK\_TEST-3" daemon prio=6 tid=0x0000000006859000 nid=0x25dc waiting for monitor entry [0x000000000825f000] java.lang.Thread.State: BLOCKED (on object monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.goMonitorDeadlock(ThreadDeadLockState.java:197) - waiting to lock <0x00000007d58f5e48> (a com.nbp.theplatform.threaddump.ThreadDeadLockState$Monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.monitorOurLock(ThreadDeadLockState.java:182) - locked <0x00000007d58f5e78> (a com.nbp.theplatform.threaddump.ThreadDeadLockState$Monitor) at com.nbp.theplatform.threaddump.ThreadDeadLockState$DeadlockThread.run(ThreadDeadLockState.java:135) Locked ownable synchronizers: - None

上面的例子中，当线程 A 需要获取线程 B 的锁来继续它的任务，然而线程 B 也需要获取线程 A 的锁来继续它的任务的时候发生的。在 thread dump 中，你能看到 DEADLOCK\_TEST-1 线程持有 0x00000007d58f5e48 锁，并且尝试获取 0x00000007d58f5e60 锁。你也能看到 DEADLOCK\_TEST-2 线程持有 0x00000007d58f5e60，并且尝试获取 0x00000007d58f5e78，同时 DEADLOCK\_TEST-3 线程持有 0x00000007d58f5e78，并且在尝试获取 0x00000007d58f5e48 锁，如你所见，每个线程都在等待获取另外一个线程的锁，这状态将不会被改变直到一个线程丢弃了它的锁。

无限等待的Runnable状态样例

508F30D8344F4C3BBDF66E796871D26A.png

"socketReadThread" prio=6 tid=0x0000000006a0d800 nid=0x1b40 runnable [0x00000000089ef000] java.lang.Thread.State: RUNNABLE at java.net.SocketInputStream.socketRead0(Native Method) at java.net.SocketInputStream.read(SocketInputStream.java:129) at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:264) at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:306) at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:158) - locked <0x00000007d78a2230> (a java.io.InputStreamReader) at sun.nio.cs.StreamDecoder.read0(StreamDecoder.java:107) - locked <0x00000007d78a2230> (a java.io.InputStreamReader) at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:93) at java.io.InputStreamReader.read(InputStreamReader.java:151) at com.nbp.theplatform.threaddump.ThreadSocketReadState$1.run(ThreadSocketReadState.java:27) at java.lang.Thread.run(Thread.java:662)

上例中线程的状态是RUNNABLE，但在下面的堆栈日志中发现socketReadThread 线程正在无限等待读取 socket，因此不能单纯通过线程的状态来确定线程是否处于阻塞状态，应该根据详细的堆栈信息进行分析。