Java Thread Dumps

Part 1

In this article we will try to understand Java Thread Dumps and how they can be very powerful tools to quickly understand and solve common problems. This article is divided into 2 parts. The first part focuses on following basic topics

1. Brief introduction to Java threads
2. Understand what are thread dumps
3. Format of thread dumps
4. How to take thread dumps

In the Part II we will learn to dissect and interpret thread dumps by

1. Understanding thread states
2. Best practices around analysing dumps
3. Analysing a few race conditions
4. Resources

All through this article the thread dumps shown are from real life scenarios. However, in the interest of space only snippets have been shown. We hope that the fundamental concepts can be still be easily understood and easily applied to your situation.

Part I

Introduction to Java threads

Processes and Threads are basic building blocks for concurrent programming.

A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space. Processes are often seen as synonymous with programs or applications. However, what the user sees as a single application may in fact be a set of cooperating processes. To facilitate communication between processes, most operating systems support Inter Process Communication (IPC) resources, such as pipes and sockets. IPC is used not just for communication between processes on the same system, but processes on different systems.

Threads are sometimes called *lightweight processes*. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process.

Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication. Multithreaded execution is an essential feature of the Java platform. Every application has at least one thread — or several, if you count "system" threads that do things like memory management and signal handling. But from the application programmer's point of view, you start with just one thread, called the main thread. This thread has the ability to create additional threads, as we'll demonstrate in the next section. Most implementations of the Java virtual machine run as a single process.

The rest of the article assumes you have basic understanding of programming with threads on java. If you would like to refresh your knowledge we would highly recommend Concurrency chapter from the Sun Java Tutorial. You could also look at the several other online resources.

What are thread dumps

Let us understand thread dumps by looking at a thread dump.

```
Full thread dump Java HotSpot(TM) Client VM (1.5.0_04-b05 mixed mode, sharing):

"DestroyJavaVM" prio=5 tid=0x00036218 nid=0xd68 waiting on condition
[0x00000000..0x0007fae8]

"Thread-1" prio=5 tid=0x00ab8e68 nid=0xe14 waiting on condition
[0x02d0f000..0x02d0fb68]
at java.lang.Thread.sleep(Native Method)
at org.tw.testyard.thread.Consumer.run(Consumer.java:18)
at java.lang.Thread.run(Unknown Source)

"Thread-0" prio=5 tid=0x00aa3ab8 nid=0x1530 waiting on condition
[0x02ccf000..0x02ccfbe8]
at java.lang.Thread.sleep(Native Method)
at org.tw.testyard.thread.Producer.run(Producer.java:24)
at java.lang.Thread.run(Unknown Source)

"Low Memory Detector" daemon prio=5 tid=0x00a6e950 nid=0x1698 runnable
[0x00000000..0x00000000]

"CompilerThread0" daemon prio=10 tid=0x00a6d658 nid=0x5b8 waiting on condition
[0x00000000..0x02c0fa4c]

"Signal Dispatcher" daemon prio=10 tid=0x00a6c7c0 nid=0x15e0 waiting on condition
[0x00000000..0x00000000]

"Finalizer" daemon prio=9 tid=0x0003fb00 nid=0x598 in Object.wait()
[0x02b8f000..0x02b8fa68]
at java.lang.Object.wait(Native Method)
- waiting on <0x22a80840> (a java.lang.ref.ReferenceQueue$Lock)
at java.lang.ref.ReferenceQueue.remove(Unknown Source)
- locked <0x22a80840> (a java.lang.ref.ReferenceQueue$Lock)
at java.lang.ref.ReferenceQueue.remove(Unknown Source)
at java.lang.ref.Finalizer$FinalizerThread.run(Unknown Source)

"Reference Handler" daemon prio=10 tid=0x00a47aa0 nid=0x1538 in Object.wait()
[0x02b4f000..0x02b4fae8]
at java.lang.Object.wait(Native Method)
- waiting on <0x22a80750> (a java.lang.ref.Reference$Lock)
at java.lang.Object.wait(Unknown Source)
at java.lang.ref.Reference$ReferenceHandler.run(Unknown Source)
- locked <0x22a80750> (a java.lang.ref.Reference$Lock)

"VM Thread" prio=10 tid=0x00a67ce8 nid=0xc78 runnable

"VM Periodic Task Thread" prio=10 tid=0x00a6fc90 nid=0x830 waiting on condition
```

As the name suggests thread dump is a dump of all the threads in a Java virtual machine. It contains the current execution state of both application and the JVM specific threads. The above thread dump snippet shows two application threads [Thread-0 and Thread-1] and JVM specific threads such as "Signal Dispatcher", "Finalizer" etc. For the purpose of this article we will focus only on application threads.

Thread dumps are extremely useful in the following situations

- To get a holistic view of what is happening in the application at that particular moment
- To expose glaring issues such as
  - portions of code which seem to be invoked almost always
  - portions of code where the application seems to be hung
  - locking and thread synchronization issues in an application
- Additionally they are invaluable in production environments where sophisticated profiling tools cannot be easily attached

Format of thread dumps

The format of thread dump has been changing with JSE versions. Sun or any other vendors inform users that the *format will change* between versions. However one thing which hasn't changed is the level of information contained in a thread dump. As mentioned above thread dumps is a snapshot of the JVM state listing all the application and system level threads and monitor states.

```
Full thread dump Java HotSpot(TM) Client VM (1.5.0_04-b05 mixed mode, sharing):

"Thread-1" prio=5 tid=0x00a995d0 nid=0x1300 in Object.wait()
[0x02d0f000..0x02d0fb68]
at java.lang.Object.wait(Native Method)
- waiting on <0x22aaa8d0> (a org.tw.testyard.thread.Drop)
at java.lang.Object.wait(Unknown Source)
at org.tw.testyard.thread.Drop.take(Drop.java:14)
- locked <0x22aaa8d0> (a org.tw.testyard.thread.Drop)
at org.tw.testyard.thread.Consumer.run(Consumer.java:15)
at java.lang.Thread.run(Unknown Source)

"Thread-0" prio=5 tid=0x00a88440 nid=0x6a4 waiting on condition
[0x02ccf000..0x02ccfbe8]
at java.lang.Thread.sleep(Native Method)
at org.tw.testyard.thread.Producer.run(Producer.java:24)
at java.lang.Thread.run(Unknown Source)
```

In the thread dump snippet above you can observe the following

1. The thread dump starts with "Full thread dump", followed by a list of threads currently being executed.
2. There are 2 application threads called Thread-1 and Thread-0. These are the default names which the JVM handed over to the threads.
3. "Thread-1" is waiting for a notification after it called Object.wait() on the Drop object.
4. Similarly, "Thread-0" is sleeping on a condition after it called Thread.sleep
5. At this particular instant, there are no threads in the runnable state and hence the application isn't doing any real work.

Although thread dumps also lists the state of the system threads we are not going to look deeper into system threads.

How to take thread dumps

Thread dumps can be generated by the users as well as the system in unusual situations. The users can generate thread dumps by explicitly sending a signal to the JVM or programatically by calling java.lang.Exception.printStackTrace(). Calling printStackTrace() method will only cause the stack trace of the current thread to be printed.

On windows environment thread dumps can only be generated when the process is running in the fore ground and is associated with a command line console. Thread dumps can be generated by hitting the key combination *Ctrl + \ (Back slash)*. On unix / linux enviroments thread dumps can be generated by the command *kill -QUIT <process id of jvm>* or *kill -3 <process id of jvm>*. Thread dumps are generally logged to the stderr. Depending on your start up commands you may find the thread dumps in some other log files. Please consult your system administrator for details. Taking thread dumps by either of the 2 strategies doesnt cause the application to terminate, the JVM dumps the execution state and continues.

Although very rare the JVM can also generate a thread dump when a thread causes the JVM to crash. The amount of information which gets logged is again implementation specific. Typically you will find information about the thread which caused the JVM to crash.

Summary

In this section we had a quick look at threads, thread dump formats and how to take thread dumps. The fun is just starting. In the next section we will understand thread states and how to interpret thread dumps. Click here to go to the next part