Sanket Mehta

TechDoc#8: Troubleshooting and Monitoring Java Applications

Troubleshooting and Monitoring Java Applications
Tuning the memory use of your application requires understanding both how Java uses memory and how you can gain visibility into your application's memory use.

JVM memory usage
The JVM uses memory in a number of different ways. The primary, but not singular, use of memory is in the heap. Outside of the heap, memory is also consumed by Metaspace and the stack.

- **Java Heap** The heap is where your Class instantiations or "Objects" are stored. Instance variables are stored in Objects. When discussing Java memory and optimization we most often discuss the heap because we have the most control over it and it is where Garbage Collection (and GC optimizations) take place. Heap size is controlled by the -Xms and -Xmx JVM flags.
- **Java Stack** Each thread has its own call stack. The stack stores primitive local variables and object references along with the call stack (method invocations) itself. The stack is cleaned up as stack frames move out of context so there is no GC performed here. The -Xss JVM option controls how much memory gets allocated for each thread's stack.
- **Metaspace** Metaspace stores the Class definitions of your Objects. The size of Metaspace is controlled by setting -XX:MetaspaceSize and -XX:MaxMetaspac
- **e**Size.
- **Additional JVM overhead** In addition to the above values there is some memory consumed by the JVM itself. This holds the C libraries for the JVM and some C memory allocation overhead that it takes to run the rest of the memory pools above. Visibility tools that run on the JVM won't show this overhead so while they can give an idea of how an application uses memory they can't show the total memory use of the JVM process.

Profiling memory use of a Java application
It is important to understand how an application will use memory in both a development and production environment. The majority of memory issues can be reproduced in any environment without significant effort. It is often easier to troubleshoot memory issues on your local machine because you'll have access to more tools and won't have to be as concerned with side effects that monitoring tools may cause.

Application slowdowns are revenue-impacting and can be complex to understand and solve. Many interdependent factors affect application performance and thereby the user experience (and your business): Issues at the user end, in the application code, in the database, in external services, and across the underlying infrastructure. Monitoring the Java application stack end to end is the key to ensuring peak application performance and user satisfaction.

Useful links:
JMX Tutorial: http://www.journaldev.com/1352/what-is-jmx-mbean-jconsole-tutorial
JStack: http://www.journaldev.com/1053/java-thread-dump-visualvm-jstack-kill-3-jcmd
General Java Troubleshooting: https://docs.oracle.com/javase/10/troubleshoot/general-java-troubleshooting.htm#JSTGD107