An Oracle White Paper
September 2013

Advanced Java Diagnostics and Monitoring Without Performance Overhead

ORACLE

Introduction

Oracle Java Mission Control is a set of powerful tools running on the Oracle JDK. These tools deliver advanced, unobtrusive Java monitoring and management, suitable for use both in development and production environments. This white paper gives an introduction to Java Mission Control, describing the main components in the tool chain, how its components differ from competing technologies, and how you can use Java Mission Control to monitor, manage, profile and diagnose your applications when running on the Oracle JDK.

Non-Intrusive Profiling and Diagnostics

Most technologies used today to monitor, manage, and profile the Java runtime use fairly intrusive technologies, like byte code instrumentation and JVMTI. The main focus of Java Mission Control (JMC) is to gather the data necessary with the lowest possible impact on the running system. The technology used also enables the application to run at full speed once the tool is disconnected from the Java Virtual Machine (JVM). This makes JMC suitable for use in production environments. With the minimal overhead, JMC also minimizes the observer effect and can provide more accurate data for your application than more the more common solutions.

JMX Console

The JMX Console is a tool for monitoring and managing multiple Oracle JDK instances. It captures and presents live data about garbage collection (GC) pauses, memory and CPU usage, as well as information from any custom JMX MBeans deployed in the JDK MBean server.

Java Flight Recorder

The Java Flight Recorder (JFR) is an on-demand 'flight recorder' that produces detailed recordings about the JVM and the application it is running. The recorded data can be analyzed off line, using the Flight Recorder tool in Java Mission Control. The data includes an execution profile, as well as garbage collection statistics, optimization decisions, object allocation, heap statistics, and latency events for locks and I/O.

The following sections will describe each of these tools in a little bit more detail.

The Java Mission Control Tool Chain

Java Process Browser and JMX Console

The Java Process Browser allows users to list and connect to both locally and remotely running Java applications. JMC provides full secure access with encrypted communication and configurable user access restrictions. The Java Process Browser will automatically detect locally running Java processes, as well as remote Java processes using the Java Discovery Protocol (JDP). Figure 1 shows how a single Java Mission Control instance can detect, show, and connect to one or more Java processes through different communication channels.

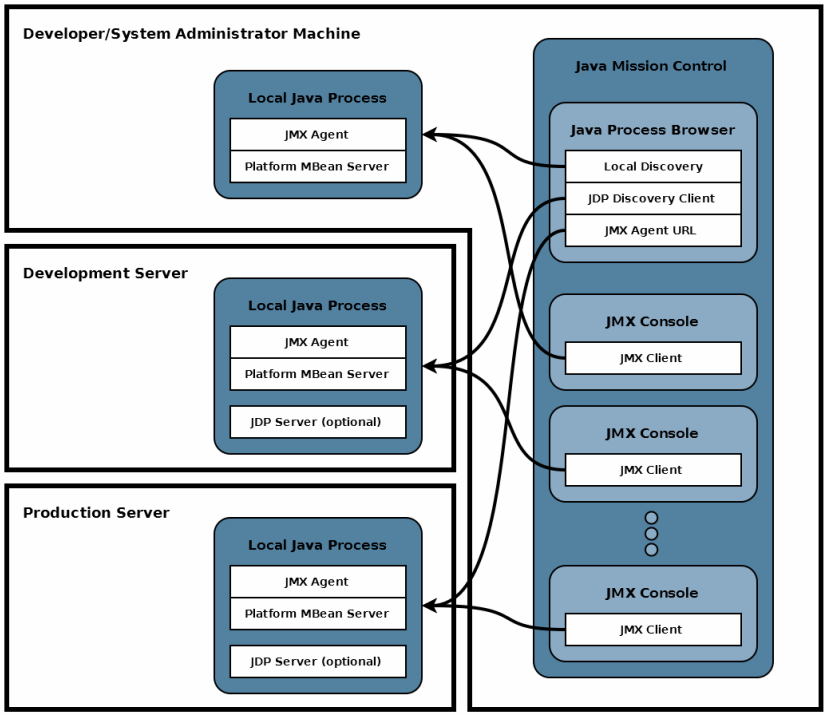

Figure 1. Diagram of Java Mission Controls communication layers.

The JMX Console, as the name implies, is a console for managing and monitoring the Oracle JDK through the JMX interface. It provides vital health data like the live set, the heap usage, CPU load, and other attributes exposed via MBeans and registered with the MBean server. Java Mission Control enables users to easily access custom application MBeans and visualize them in the JMX Console. The JMX Console also provides triggers that can monitor MBeans and trigger an action, such as showing a notification, when a condition is met.

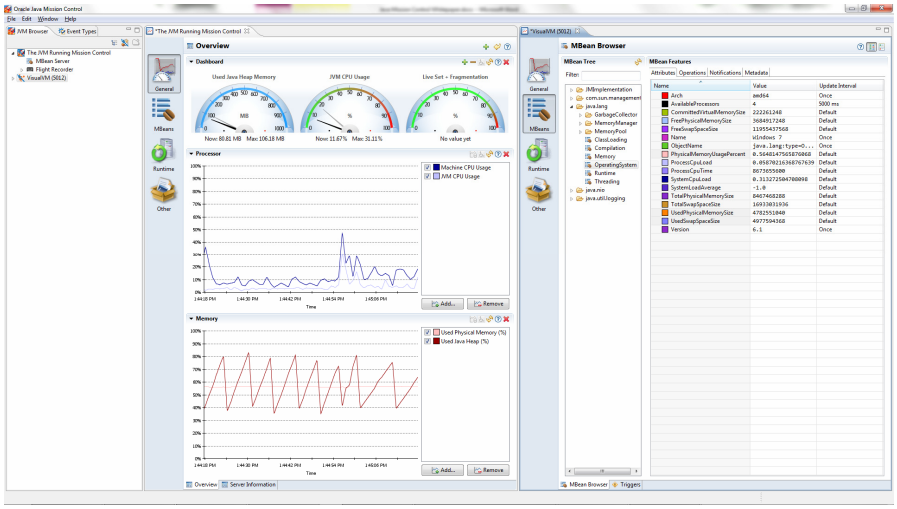

Figure 2. Screenshot showing the JMX Console in Java Mission Control connected to two different local Java processes, the first showing current CPU and heap usage, and the second listing all registered MBeans.

Java Flight Recorder

With Java Flight Recorder system administrators and developers have a new way to diagnose production issues. JFR provides a way to collect events from a Java application from the OS layer, the JVM, and all the way up to the Java application. The collected events include thread latency events such as sleep, wait, lock contention, I/O, GC, and method profiling.

With the low performance impact, less than 2% for typical Java applications, when collecting data and events, JFR can be enabled by default and continuously collect low level data from Java applications in production environments. This will allow a much faster turnaround time for system administrators and developers when a production issue occurs. Rather than turning on data gathering after the fact, the continuously collected JFR data is simply written to disk and the analysis can be done on data collected from the application leading up to the issue, rather than data collected afterwards.

Data Collection

Java Flight Recorder does not write events to disk immediately as they occur. Instead, it stores data in a hierarchy of in-memory buffers and then moves the data to the disk when the buffers are full. Initially, the JFR run time stores the event data in thread-local buffers, eliminating the need to synchronize between threads for every event, which greatly improves throughput. Once a thread-local buffer has been filled, the data is transferred to a global buffer. When this happens, synchronization is necessary between threads but, because different thread-local buffers fill up at different rates, contention is rare. Eventually, the global buffer also runs out of space and the contents in the buffer are written to the disk. Writing to the disk is expensive and JFR ensures that it is done as efficiently possible. The produced files are in a binary format that is extremely compact while also efficient for the applications to read and write.

You can configure JFR so that it does not write any data to disk. In this mode, the global buffer acts as a circular buffer and the oldest data is dropped when the buffer is full. Even with this very low-overhead operating mode all the vital data necessary for root-cause problem analysis is still collected. Because the most recent data is always available in the global buffer, it can be written to disk on demand whenever operations or surveillance systems detect a problem.

Figure 3. Possible workflow when using Java Flight Recorder in production.

Data Analysis

The Java Flight Recorder plugin in Java Mission Control provides dynamic and in-depth analysis of all collected JFR data. JMC enables developers and system administrators alike to analyze all aspects of a Java application, from the high-level behavior to improve tuning, down to lock contention latencies on specific Java objects to help find opportunities to optimize the implementation.

For the most common analysis JFR provides specialized tabs that focus on a specific area such as Code, Memory, Threads, Locks, and I/O. This makes it easy to quickly drill down in specific areas and understand how the Java application behaves.

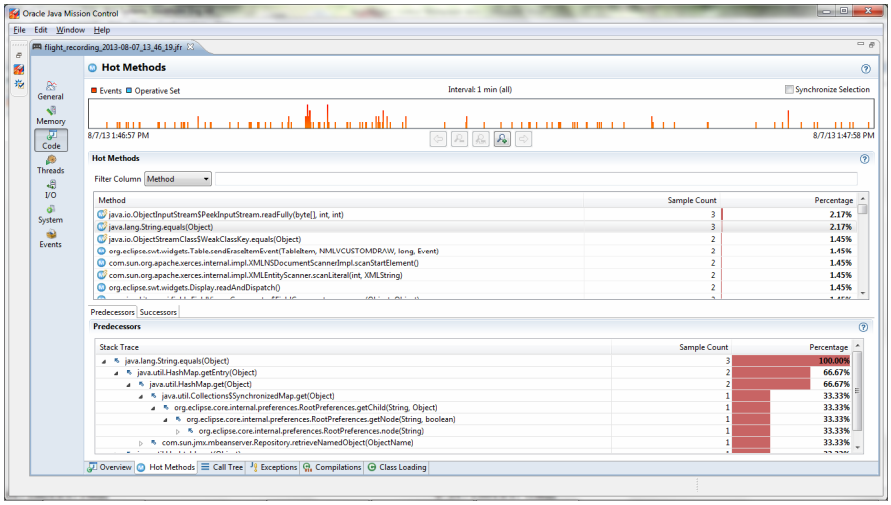

Figure 4. Method profiling information in Java Mission Control, with full stack trace for each method sample.

For advanced analysis across all events JMC provides an Event tab that allows ad hoc analysis of all the data collected. By zooming in on a specific time period and filtering events using the Operative Set users can drill down to specific events and determine the root cause of an issue.

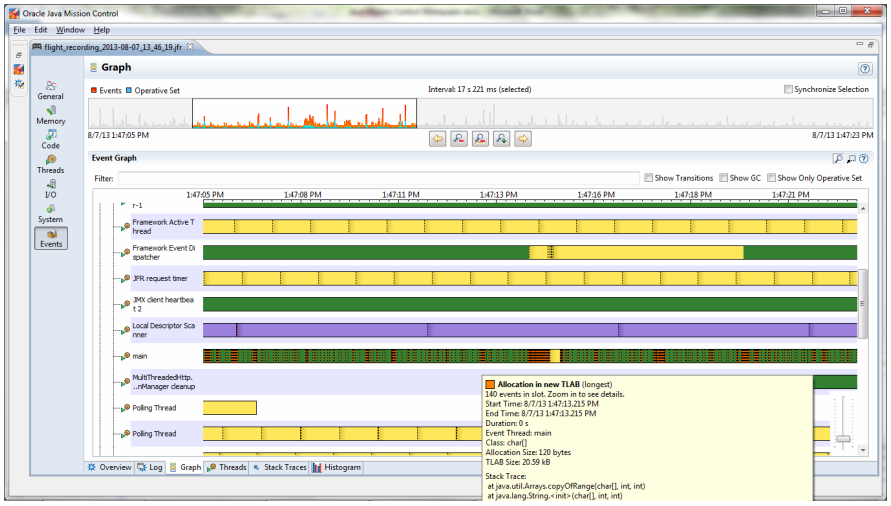

Figure 5. Advanced analysis in the Event tab showing all events in a thread graph as they occur over time.

Conclusion

Java Mission Control is a versatile tools suite for monitoring, managing, diagnosing, and profiling your Java applications. You can reliably use Java Mission Control in production environments without leaving any trace in your system after it has been used, and with a much smaller performance overhead than comparable tools when it is in use. Java Mission Control is a commercial feature and part of Java SE Advanced, but is free for development use as defined by the BCL for Java SE.

ORACLE

Advanced Java Diagnostics and Monitoring
Without Performance Overhead
September 2013

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com/java

Oracle is committed to developing practices and products that help protect the environment