# Java Monitoring and Troubleshooting Tools In Action

Bill Au

billa@apache.org

http://www.apache.org/~billa/apacheconus2008/

ApacheCon

Leading the Wave of Open Source

# Agenda

- Monitoring
- Thread and Heap Dump
- Hung or Slow App
- OutOfMemoryError
- JVM crashes

# Monitoring - java.lang.management

- RuntimeMXBean
- OperatingSystemMXBean
- ClassLoadingMXBean
- CompilationMXBean
- GarbageCollectorMXBean
- MemoryMXBean, MemoryPoolMXBean
- ThreadMXBean

Sample code in $JAVA_HOME/demo/management/

# Monitoring - java.lang.management

- Sample code:

```
import java.lang.management.*;

ThreadMXBean tb=ManagementFactory.getThreadMXBean();
if (tb.isThreadCpuTimeSupported()) {
    long[] tids=tb.getAllThreadIds();
    for (long tid : tids) {
        System.out.println("cputime:"+tb.getThreadCpuTime(tid));
        System.out.println("usertime:"+tb.getThreadUserTime(tid));
    }
}
```

# Monitoring - tools

- JDK command line utilties
  - jinfo, jmap, jstack
  - jstat
- jconsole
  - Monitor CPU usage by using JTop plug-in ($JAVA_HOME/demo/management/JTop)
- Garbage collection log: -Xloggc:<*filename*>
  - Timestamp and duration of GC
  - Heap size before and after GC

# Thread and Heap Dump

- Thread dump
  - kill -3 (or kill –SIGQUIT)
  - ThreadMXBean
  - jstack
  - jconsole
- Heap dump
  - -XX:+HeapDumpOnOutOfMemoryError
  - jmap
  - jconsole
  - hprof – beware of overhead (all object allocation is instrumented)

# Hung or Slow App

- Long GC pauses and/or high GC overhead
  - GC tuning (heap size, serial/parallel/concurrent collector, etc)
  - Baseline, tune, measure one change at a time
  - Beware of over-optimizing
- Deadlock
  - Take thread dump to run the deadlock detector
- Loop threads
  - Monitor CPU times of thread (ThreadMXBean, jconsole with JTop), inspect stack trace of suspicious threads

# Hung or Slow App

- Blocked threads
  - Analyze thread dumps for resource contention
  - Look for incautious and/or improper synchronization
- Stuck threads
  - Look for runnable threads with the same stack trace across consecutive thread dumps
  - Some common causes:
    - Blocking for a network call without specifying a connect or read timeout (default may be no timeout)
    - Long execution time for calls to backend servers

# OutOfMemoryError - heap

Java heap space

GC overhead limit exceeded

Requested array size exceeds VM limit

- Heap too small
  - Adjust maximum heap size (-Xms<*size*>)
- Excessive use of finalizers
  - Monitor objects pending finalization (MemoryMXBean, jmap, jconsole)
- look for logic error in array allocation code
- Memory leak
  - Analyze heap dump

# OutOfMemoryError – Permanent Generation

PermGen space

- Permanent generation too small
  - Adjust maximum permanent generation size ( -XX:MaxPermSize<*size*>)
- Classloader leak
  - Look for leaked threads
  - Use jhat to look for leaked classloader in a heap dump



# OutOfMemoryError – too many threads

unable to create new native thread

- Each Java thread has a native thread with its own stack
  - Lower maximum number of threads
  - Decrease maximum stack size (-Xss<*size*>)
    - watch out for StackOverflowError

# OutOfMemoryError – native memory

Request <size> bytes for <reason>. Out of swap space?
<reason> <stack trace> (Native method)

- System running out of memory
  - Check memory usage of other processes
- Process exceeds maximum process size
  - Reduce heap or stack memory usage
- Leak in JNI or native method
- Leak in VM code
  - Upgrade to the latest update release
  - Check fatal error log (hs_err_<*pid*>.log)

# JVM Crashes

- Upgrade to the latest update release
- Check fatal error log (hs_err_<*pid*>.log)
- Look for workaround
  - Sun's bug database (http://bugs.sun.com/)
  - Sun's Java forum (http://forum.java.sun.com/index.jspa)
  - Google
  - Post to Sun's Java forum and/or open a bug

# JVM Crashes

- Possible workaround:
  - Different JVM mode (server/client) on 32-bit machine
  - Different collector for GC related crashes
  - Run the JVM in interpreted mode (-Xint) for compiled Java code or compiler related crashes