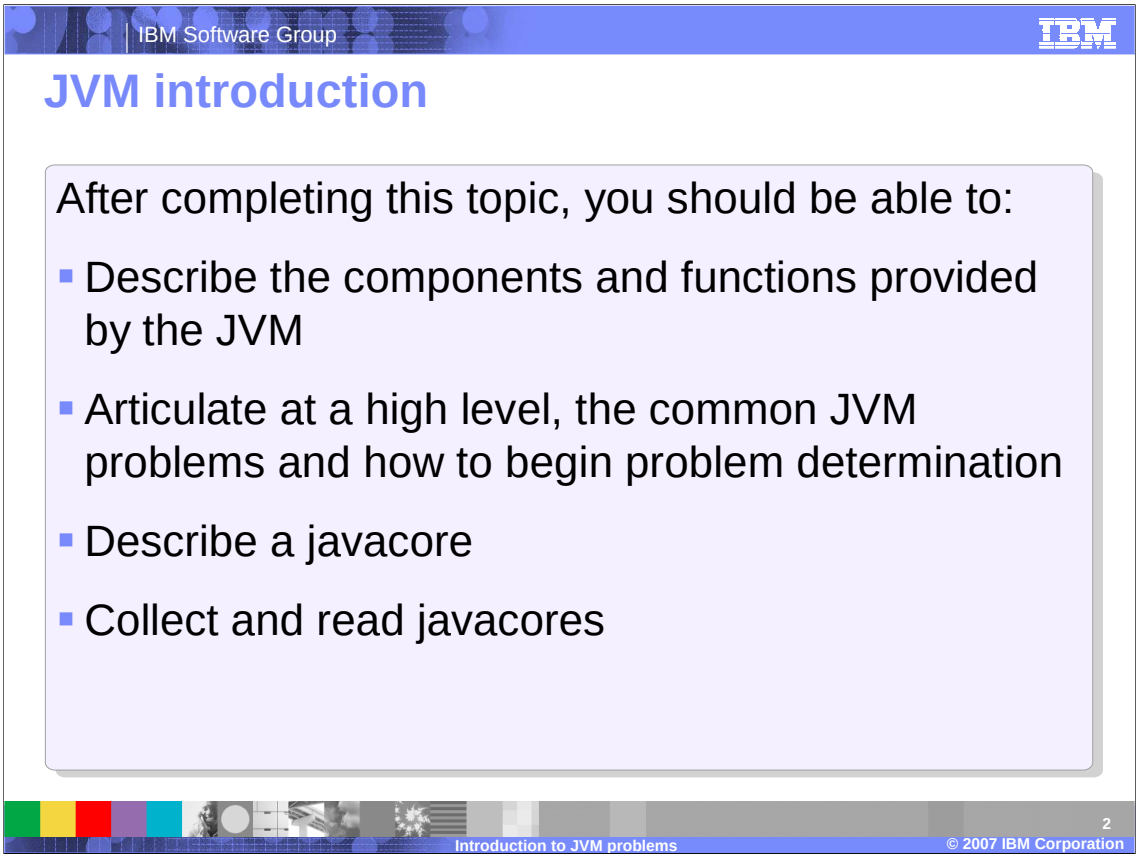
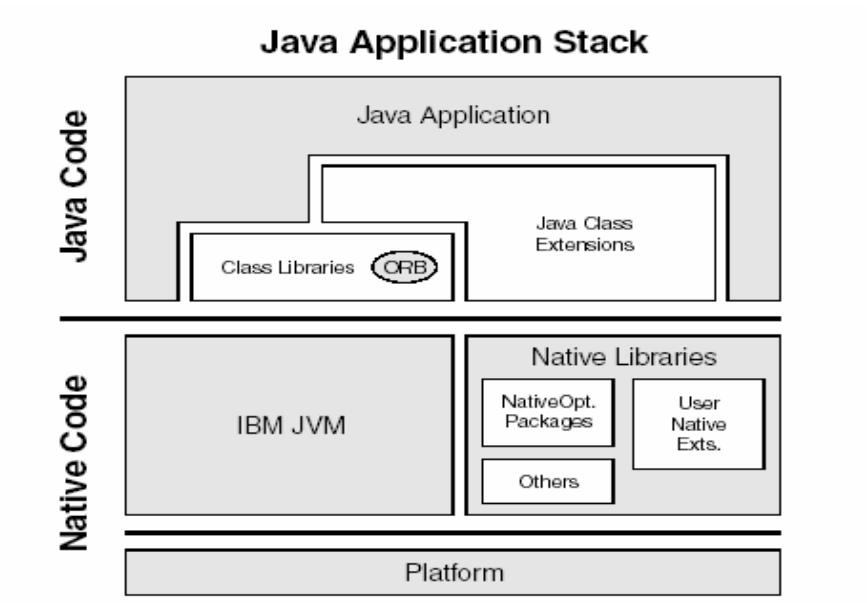
This presentation will act as an introduction to JVM-related problems when using WebSphere Application Server V6.



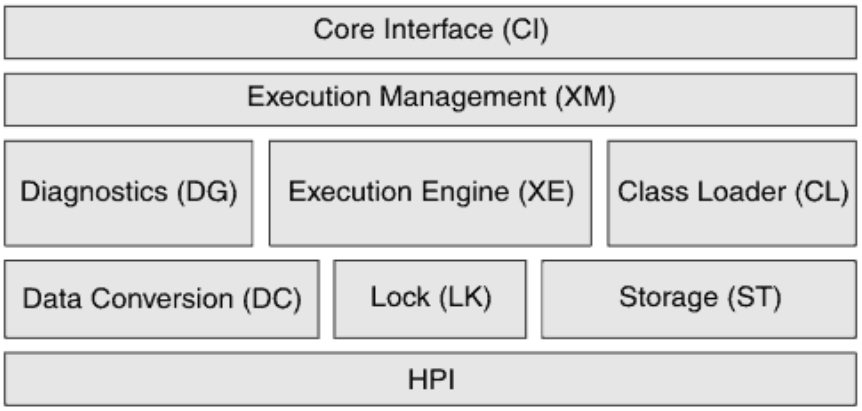
After completing this topic, you should be able to describe the key components of the JVM, identify common JVM problems, and produce and process javacore files.

JVM overview (1 of 2)



The IBM Java Virtual Machine, or JVM, is the core component of the IBM Java Runtime Environment, or JRE. The IBM JRE includes the JVM, the class libraries (including the IBM Object Request Broker), and other files that provide the runtime support that is necessary for a Java application stack. This figure shows the components of a typical Java Application Stack and the IBM JRE.

JVM overview (2 of 2)



The JVM is an interpretive computing engine that is responsible for running the bytecode in a compiled Java program. The JVM translates the Java bytecodes into the native instructions of the host machine. The application server, being a Java process, requires a JVM in order to run and to support the Java applications that are running on it. JVM settings are part of an application server configuration.

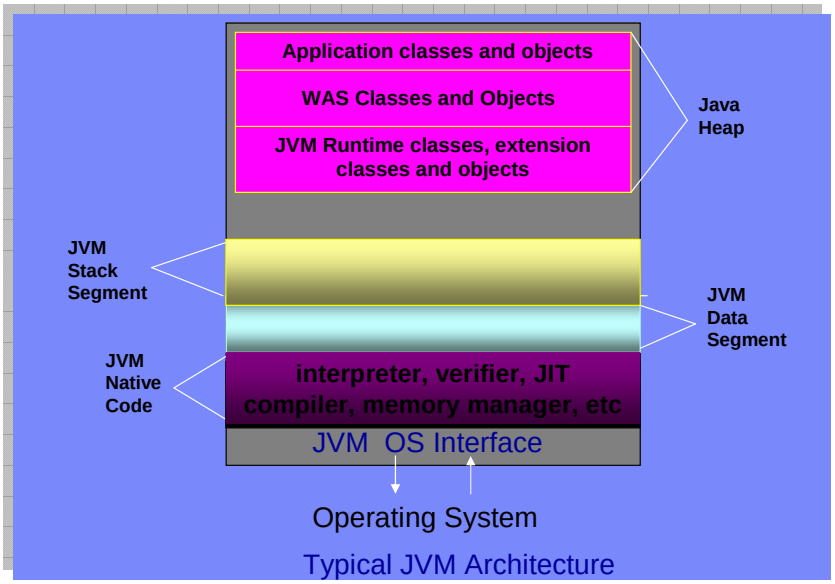
JVM version

- WebSphere supports several JVMs based on platform type
  - Windows, AIX and Linux – IBM supplied
  - Sun and HP – hybrid of IBM add-ons and vendor supplied JVM
- For a comprehensive list of the supported JVMs check
  - http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html
- To determine the JVM version in use
  - Look in the SystemOut.log file of one of the profile instances
    - <*profile_home*>/logs/server1/SystemOut.log
  - Run **java -fullversion** from the command line
    - <*was_home*>/java/bin/java –fullversion

```
C:\WebSphere\AppServer\java\bin>java –fullversion
java full version "J2RE 1.4.2 IBM Windows 32 build cn142sr1w-
20041028"
```


JVMs differ on the available tools supported and can have different behavior, even between different versions targeted for the same platform. Always look at the documentation for the specific JVM for behavioral descriptions and the options to control the JVM. To determine the JVM version in use, look in the SystemOut.log file of one of the profile instances, or run “java –fullversion” from the command line.

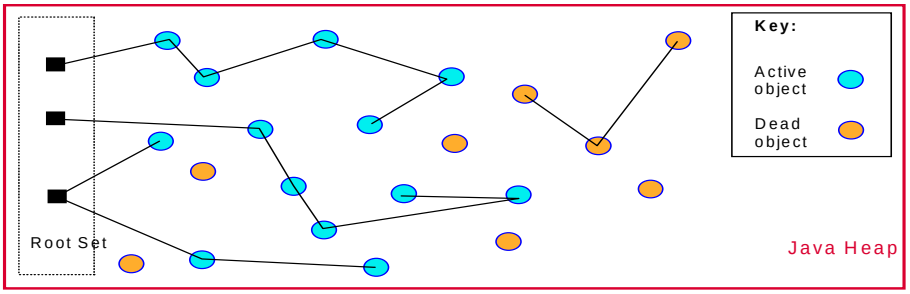
JVM memory segments



The JVM is a process whose memory layout depends on the OS platform. The diagram shown here is a typical abstraction of process memory layout. As a process, the JVM itself needs its own heap, the JVM native heap, from which it gets its own dynamic memory needs. A portion of that heap is allocated for the Java heap where classes and objects are actually stored. This is the heap that Java programmers are familiar with. Garbage collection occurs only in the Java heap. The Java heap can grow and shrink. When there are enough available memory in the JVM heap, the Java heap can be extended easily without acquiring memory from the OS. Otherwise, a request for additional memory from the OS is made and the JVM process size increases consequently.

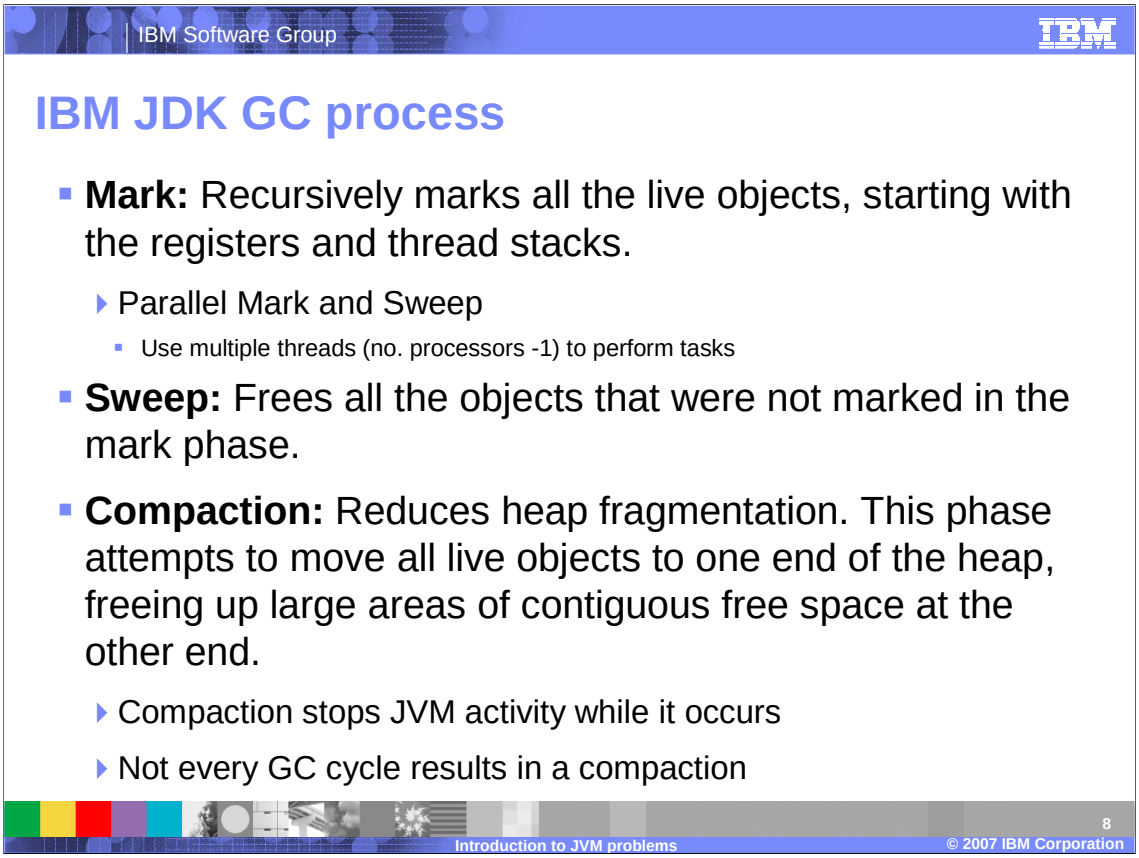
Understanding garbage collection (GC)

- JVM manages the Java heap and does garbage collection
- Object is eligible for GC when there are no more references from root to that object
- Root set - References in the stack and registers
- Reference is dropped when variable goes out of scope

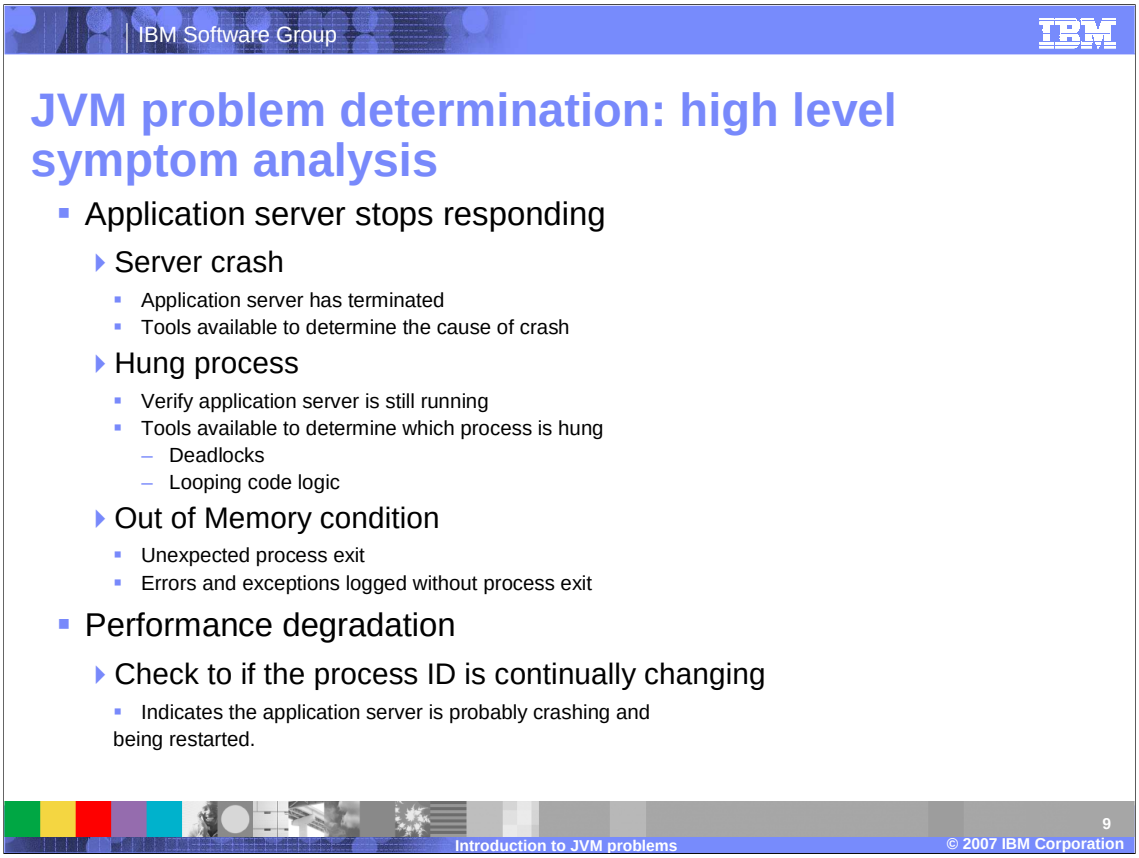


The active state of the JVM is made up of the set of stacks that represents the threads, the statics that are inside Java classes, and the set of local and global JNI references. The garbage collector, beginning at the this root set of references, scans the thread stacks and looks for pointers that look like they are referencing Java objects. It is possible that there are attributes (such as floats) being pushed onto the stack which may look like Java object references, so the garbage collector treats these entries conservatively by marking them as “dosed” to make them unmovable during compaction.

All the objects referenced by these pointers are reachable objects, so they will be listed in a mark vector, which will then be used to compare with the allocbits vector. The allocbits vector contains the information the garbage collector needs for all the objects created. The garbage collector can determine what needs to be collected by looking at the differences between these two vectors.

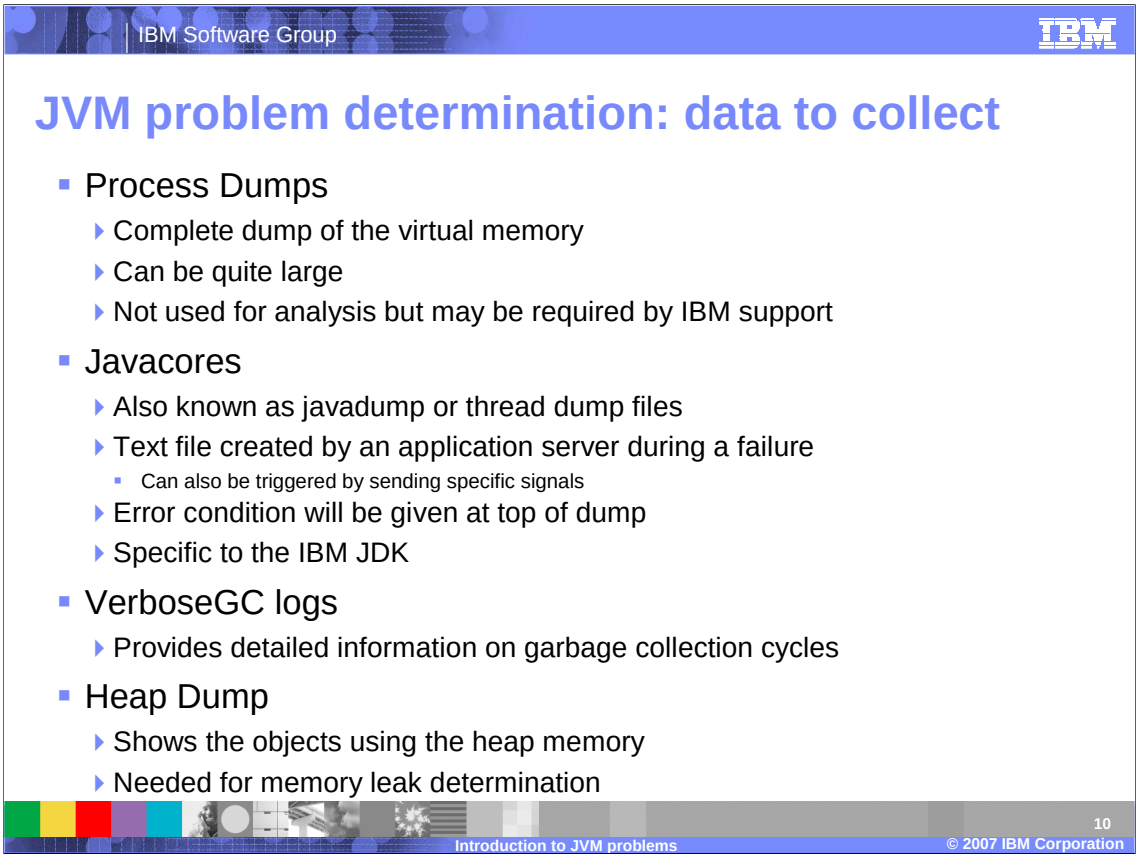


When the JVM cannot allocate an object from the current heap because of lack of space, a memory allocation fault occurs, and the Garbage Collector is invoked. The first task of the Garbage Collector is to collect all the garbage that is in the heap. This process starts when any thread calls the Garbage Collector either indirectly as a result of allocation failure, or directly by a specific call to System.gc(). The first step is to get all the locks that the garbage collection process needs. This step ensures that other threads are not suspended while they are holding critical locks. All the other threads are then suspended. Garbage collection can then begin. It occurs in three phases: mark, sweep, and compaction. During the mark phase, the garbage collector recursively marks all the live objects, starting at the root set. The sweep phase frees all the objects that were not marked in the mark phase. Finally, compaction, attempts to move all the live objects to one end of the heap. Because compaction can take a long time, the Garbage Collector tries to avoid it if possible.

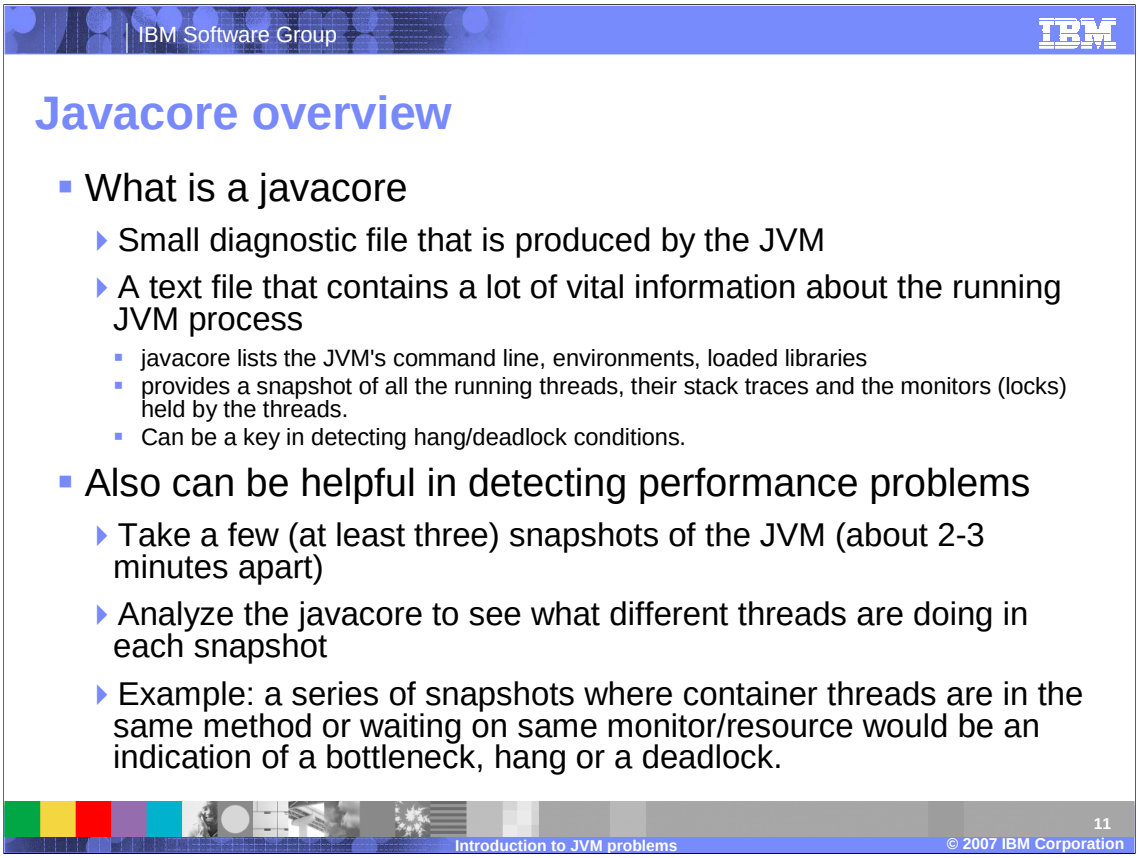


A JVM may become unavailable or just perform poorly. An application server that does not respond might be hung or the process might have ended. Users see hung applications or are not able to access new applications. If cpu activity is low but the application server has not terminated, you most likely have a hang or deadlock situation. If cpu activity is high and the application server is using the cycles, you most likely have a loop or inefficient code.

An often encountered condition is out-of-memory (OOM), which manifests itself either as an unexpected process exit with a "OutOfMemoryException“ or just a bunch of errors and exceptions but no immediate process exit. This can occur when the application server is running low on memory, perhaps due to an application issue such as a memory leak, a hardware memory failure, or higher than usual demand.



A number of diagnostic documents may be generated to assist in your problem determination efforts, including javacores, heap dumps and verbose garbage collection logs.



A javacore file is a snapshot of a running Java process. The IBM Java SDK produces a javacore in response to specific operating system signals. Javacore files show the state of every thread in the Java process and monitor information identifying Java synchronization locks.

You can use Javacore files to aid in resolving problems such as hangs or high cpu. A frequent method of analysis is to take a series of 3 javacores, separated by 3 minutes, and then compare the thread activity.

Javacore location and naming

- The javacore file is stored in the first viable location of:
  - The setting of the IBM_JAVACOREDIR environment variable
  - <WAS_install_root>/profiles/<profile>
  - TMPDIR or TEMP environment variable
  - Windows only: If the javacore cannot be stored in any of the above, it is put to STDERR.
- Javacore naming
  - Windows and Linux: javacore.YYYYMMDD.HHMMSS.PID.txtYYYY=year, MM=month, DD=day, SS=second, PID=processID
  - AIX: javacorePID.TIME.txtPID=processID, TIME=time since 1/1/1970
  - z/OS: JAVADUMP.YYYYMMDD.HHMMSS.PID.txtYYYY=year, MM=month, DD=day, SS=second, PID=processID

The location of the generated javacore file differs between platforms, but can be specified using environmental variables. For detailed information check the IBM JDK Diagnostic Guide, which explains in detail all the options for controlling javacores such as naming and location.

Javacore contents

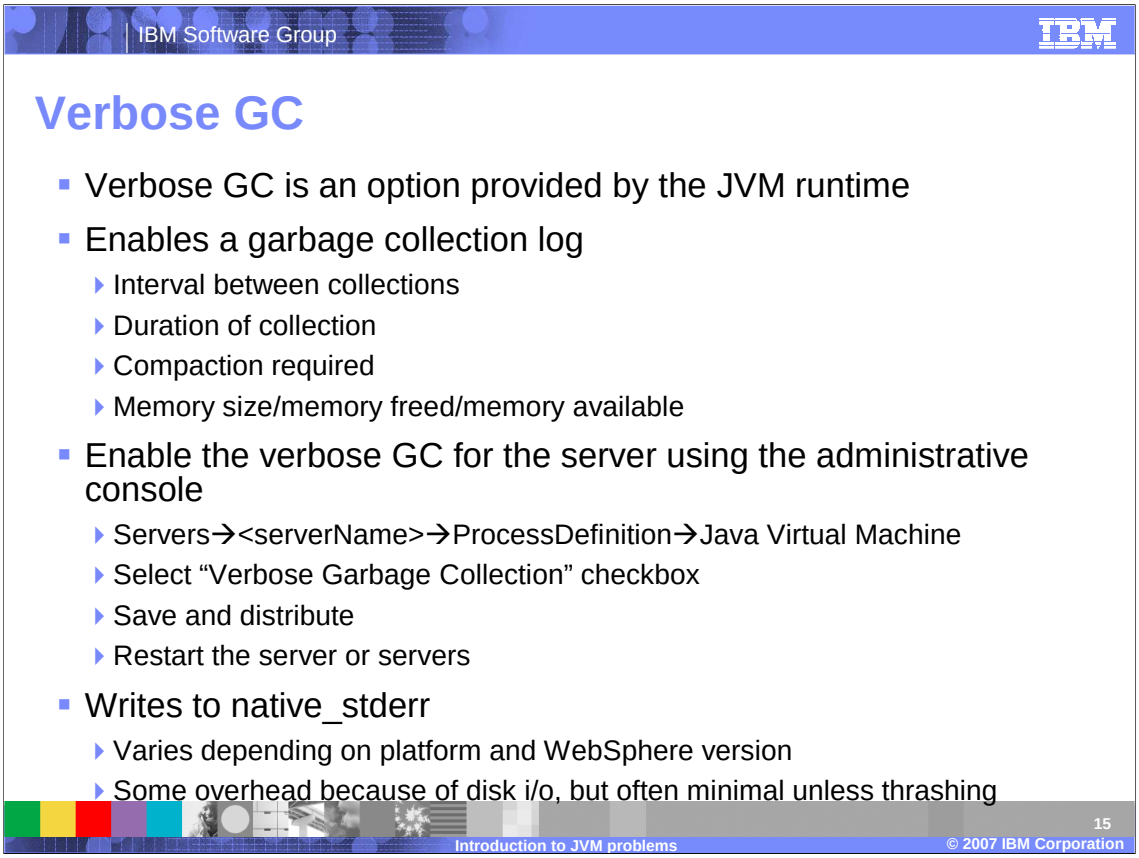
- Java version
- Java command line
  - Command line that started Java process
- Current thread details
  - Thread that is running when the signal is raised that causes the javacore to be written
- JVM thread dump
  - Current stack for every thread in the JVM
- Notable tags:
  - 1TIDATETIME
    - Indicates the time and date of when the javacore file was produced.
  - 1TISIGINFO
    - Contains the signal that caused the javacore.
  - 1XHSIGRECV
    - Indicates the library that issued the operating system signal

Every javacore file has a line that indicates the time and date of when the javacore file was produced and the fullversion of the JVM that produced the javacore.

Javacore example (excerpt)

```
NULL           ------------------------------------------------
0SECTION       TITLE subcomponent dump routine
NULL           ===============================
1TISIGINFO     signal 11 received
1TIDATETIME    Date:                 2004/04/06 at 16:04:16
1TIFILENAME    Javacore
filename:    /export/home/was4/wasapps/appvmc01/vmclaimt1/workdir/ja
vacore499836.1081281856.txt
NULL           ------------------------------------------------
0SECTION       XHPI subcomponent dump routine
NULL           ===============================
1XHTIME        Tue Apr  6 16:04:16 2004
1XHSIGRECV     SIGSEGV received at 0xd2782a88 in
/opt/WebSphere4/AppServer/java/jre/bin/libjitc.a. Processing
terminated.
1XHFULLVERSION J2RE 1.3.1 IBM AIX build ca131-20030630
........
```


This is an example of an excerpt from a javacore. In this case, the messages indicate a crash in a JVM library and should be pursued with WebSphere Application Server Support.



When the JVM cannot allocate an object from the current heap because of lack of space, a memory allocation fault occurs, and the Garbage Collector is invoked. A good way to see what is going on with garbage collection is to use verbosegc, which is enabled in WebSphere Application Server using the administrative console. It is often recommended to have verbose GC enabled permanently in production. The overhead cost on a reasonably well-tuned JVM is actually quite small. The benefits of verbosegc the first time a problem happens are considerable, while monitoring the verbose GC regularly is a simple way to monitor the health of the system.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject= Feedback about SW5706G14A_IntroToJVMProblems.ppt

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX Current IBM WebSphere z/OS

Windows, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JDK, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.