This unit looks at how to detect and troubleshoot a crash.

Unit objectives

After completing this unit, you should be able to:

- Describe what a crash is and be able to detect one.
- Analyze javacore files for crash
- Use the DTFJ-DumpReporter

After completing this unit, you should be able to describe what a crash is and be able to detect one, analyze javacore files for crash, and use the Diagnostic Tooling Framework for Java DumpReporter.

Debug a crash

After completing this topic, you should be able to:

- Describe a JVM™ crash
- Describe how to control what dump data is generated
  - List common operating system signals related to a crash
- Analyze a javacore file to detect crash symptoms

After completing this topic, you should be able to, describe a Java Virtual Machine crash, describe how to control what dump data is generated, and analyze a javacore file to detect crash symptoms.

JVM process crash defined

- Not the same as thread hang
- Symptoms
  - Process terminated with Java exception or native signal
- Usual causes
  - Out of Memory Exception
  - Call stack overflow
  - Unexpected exception (for example, Out of Disk Space)
  - Optimizer failure (for example, JiT)
  - Bad JNI call or library problem
- Analyzing—mostly Out of Memory and Call Stack issues
  - Try to generate thread dumps just before the crash
  - Thread dumps may be created during crash

A Java Virtual Machine crash is not the same as a thread hang. Symptoms are that a process terminates with a java exception or native signal. Usual causes are an out of memory exception, call stack overflow, an unexpected exception such as out of disk space, optimizer failure such as a Just In Time Compiler failure, or a bad Java Native Interface call or library problem. It is useful to try to generate thread dumps just before or during the crash.

Crash problem determination: data to collect

- Process Dumps
  - Complete dump of the virtual memory in binary format
  - Can be quite large
  - Tools available to parse files into readable formats
- Javacores
  - Also known as javadump or thread dump files
  - Text file created by an application server during a failure
    - Can also be triggered by sending specific signals
  - Error condition will be given at top of dump

A process dump is a complete dump of the virtual memory in binary format. A javacore, also known as a javadump or thread dump file, is a text file created by an application server during a failure.

JVM dump initiation: events

- Events (or conditions) that can cause dumps:
  - EXCEPTION: Unexpected synchronous terminating signal; that is, unrecoverable storage violation.
  - ERROR: Controlled abort due to an error detected internally; for example, no more memory is available.
  - INTERRUPT: Asynchronous terminating signal; for example, you pressed CTRL-C.
  - DUMP: This can be caused if you press CTRL-BREAK on Windows®, CTRL-V on z/OS®, or CTRL-\ on AIX® or Linux®.
  - OUTOFMEMORY: The JVM cannot satisfy a request for storage.

The Java Virtual Machine might produce dump files in response to specific events, depending on the setting of the environment variables JAVADUMPOPTS and JAVADUMPTOOL

JVM dump initiation: types

- The types of dump that can be produced are:
  - SYSDUMP: An unformatted dump that the operating system generated (basically a core file).
  - User specified: Whatever the JAVA_DUMP_TOOL variable specifies.
  - HEAPDUMP: An internally-generated dump of the objects that are on the Java heap.
  - JAVADUMP: An internally-generated and formatted analysis of the JVM.
- Note that platform specific variations exist

The types of dump that can be produced are SysDump, which is an unformatted dump that the operating system generated (basically a core file). A HeapDupm is an internally-generated dump of the objects that are on the Java heap, and a JavaDump is an internally-generated and formatted analysis of the Java Virtual Machine.

JAVA_DUMP_OPTS variable

- Which dumps are produced for which condition is determined by the JAVA_DUMP_OPTS variable as follows:

  JAVA_DUMP_OPTS="ON*condition*(*dumptype*[*count*],*dumptype*[*count*]),ON*condition*(*dumptype*[*count*],...)"

  - *condition* can be: ANYSIGNAL, DUMP, ERROR, INTERRUPT, EXCEPTION and OUTOFMEMORY
  - *dumptype* can be: ALL, NONE, JAVADUMP, SYSDUMP and HEAPDUMP
  - *count* is the maximum number of dumps of this type to produce.

- For example:
  - ONERROR(SYSDUMP) creates system dumps for error conditions

The JavaDumpOpts variable describes the condition under which to take the dump, the type of dump to take and the maximum number of dumps of this type to produce.

Unix operating system common signals

- **SIGQUIT**
  - Indicates a command was issued to generate a thread dump
  - Typically does not end the JVM process
- **SIGILL**
  - Means an illegal instruction was executed
    - can mean a corruption of the code segment or a branch that is not valid within the native code
  - This signal often indicates a problem caused by JIT-compiled code
- **SIGSEGV**
  - Indicates an operation that is not valid in a program
    - such as accessing an illegal memory address
  - This is typically indicative of a programming problem in one of the native libraries

Common signals on Unix® operating systems include SigQuit which indicates a command was issued to generate a thread dump and typically does not end the Java Virtual Machine process. SigIll means an illegal instruction was executed and often indicates a problem caused by Just In Time-compiled code. SigSegv indicates an invalid operation.

Windows operating system common signals

- **Memory access error**
  - Invalid memory address
  - JVM action: javacore file and abort
- **Illegal access error**
  - JVM action: javacore file and abort

Windows operating system also commonly gives invalid memory address access error, and illegal access error, both of which create a javacore file and abort.

Javacore contents

- General info
  - Timestamp and java version
  - Java command line
  - Command line that started Java process
  - Current stack for every thread in the JVM
- Current thread details
  - Thread that is running when the signal is raised that causes the javacore file to be written
- Notable tags:
  - 1TISIGINFO
    - Contains the signal that caused the javacore file
  - 1XHSIGRECV
    - Indicates the library that issued the operating system signal

A javacore contains general information, including the current stack for every thread in the Java Virtual Machine, and the current thread details for the thread that was running when the signal was raised.

Interpret the javacore file for crash information

- 1TISIGINFO contains the signal that caused the javacore file
- 1XHSIGRECV provides details on the signal. For example:
  - Crash in a JVM library

```
1XHSIGRECV  SIGSEGV received at 0xd2782a88 in
/opt/WebSphere4/AppServer/java/jre/bin/libjitc.a.
Processing terminated.
```

  - Crash in a DB2® library

```
1XHSIGRECV SIGSEGV received at 0xd44350a8 in
/usr/lpp/db2_07_01/java12/libdb2jdbc.so. Processing
terminated.
```


Examine the 1XHSIGRECV line to view the signal that caused the javacore to be written. If it is a Java Virtual Machine native library that caused the crash, a Java Standard Development Kit upgrade might resolve the problem. If the signal information does not identify the library that caused the crash, you must examine the current thread details.

Javacore fault module

- 1XHEXCPMODULE
  - Indicates the module that caused the fault
- Fault module identification:
  - The JVM module:
    - Windows: JVM.dll
    - AIX: libjvm.a
    - Linux: libjvm.so
  - The JIT module:
    - Windows: JITC.dll
    - AIX: libjitc.a
    - Linux: libjitc.so
  - Other modules may be indicated, such as DB2

Look for the full path name to that library or module as that may indicate the use of some third-party JNI library. If either the Java Virtual Machine or Just In Time compiler fault modules are indicated, first upgrade the Java Development Kit and retry to see if the problem goes away. The best way to find a workaround to a Just In Time compiler-related problem is to determine the method on which the failure is occurring and have the Just In Ttime compiler skip this method

Javacore fault module example

- Example:

```
SECTION TITLE subcomponent dump routine
NULL ===============================
1TISIGINFO signal 11 received
1TIDATETIME Date: 2005/01/24 at 18:56:08
1TIFILENAME Javacore filename:
  F:\tmp\javacore.20050124.185608.1108.txt
NULL
------------------------------------------------------------
0SECTION XHPI subcomponent dump routine
NULL ===============================
1XHEXCPCODE Exception code: C0000005 Access Violation
1XHEXCPADDRESS Fault address: 100D15B0 01:000D05B0
1XHEXCPMODULE Fault module: D:\IBM_142\jre\bin\classic\jvm.dll
```


In this example, the fault module is with the Java Virtual Machine.

Javacore current thread details

- Examine the current thread details to see which library the current thread was processing at the time of the JVM crash
- Example showing error in JIT

```
 1XHCURRENTTHD  Current Thread Details
NULL           ----------------------
2XHCURRSYSTHD  "EntigoAppsStarter" sys_thread_t:0x59AF8650
3XHNATIVESTACK Native Stack
NULL           ------------
3XHSTACKLINE   at 0xD2782A88 in dataflow_arraycheck
3XHSTACKLINE   at 0xD27226A0 in bytecode_optimization_driver
3XHSTACKLINE   at 0xD27251CC in bytecode_optimization
3XHSTACKLINE   at 0xD2685140 in JITGenNativeCode
3XHSTACKLINE   at 0xD26AB774 in jit_compile_a_method_locked
3XHSTACKLINE   at 0xD26ACD24 in jit_compiler_entry
3XHSTACKLINE   at 0xD26AD284 in _jit_fast_compile
```


Examine the current thread details to see which library the current thread was processing at the time of the Java Virtual Machine crash. The following example shows an error in the Just In Time compiler.

Steps if crash cause not identified

- Frequently, the javacore file does not clearly identify the cause of the signal. Often the Native Stack will show the following:

```
----- Native Stack -----
unable to backtrace through native code - iar 0x3062e73c not in
text area (sp is 0x2ff21748)
```

- Steps you can take
  - Disable JIT compilation.
  - Upgrading to a more recent JDK can sometimes resolve a problem.
  - Use the core file (on UNIX) or user.dmp file (on Windows) to see if this provides more information.
  - Sometimes a bad Java SDK installation can cause problems

When the javacore does not clearly identify the cause of th esignal, the native stack will follow. You can try disabling Just In Time compilation, upgrading to a more recent Java Development Kit, use the native core or dump file, or re-install the Java Standard Development Kit.

Topic summary

Having completed this topic, you should be able to:

- Describe what is meant by a JVM crash
- Understand the difference between process dump and javacore files
- Analyze a javacore file to detect crash symptoms

Having completed this topic, you should be able to describe what is meant by a Java Virtual Machine crash, understand the difference between process dump and javacore files, and analyze a javacore file to detect crash symptoms.

DTFJ-DumpReporter

After completing this topic, you should be able to:

- Describe the Diagnostic Tooling Framework for Java (DTFJ)
- Know how to convert a core dump using the DTFJ-DumpReporter

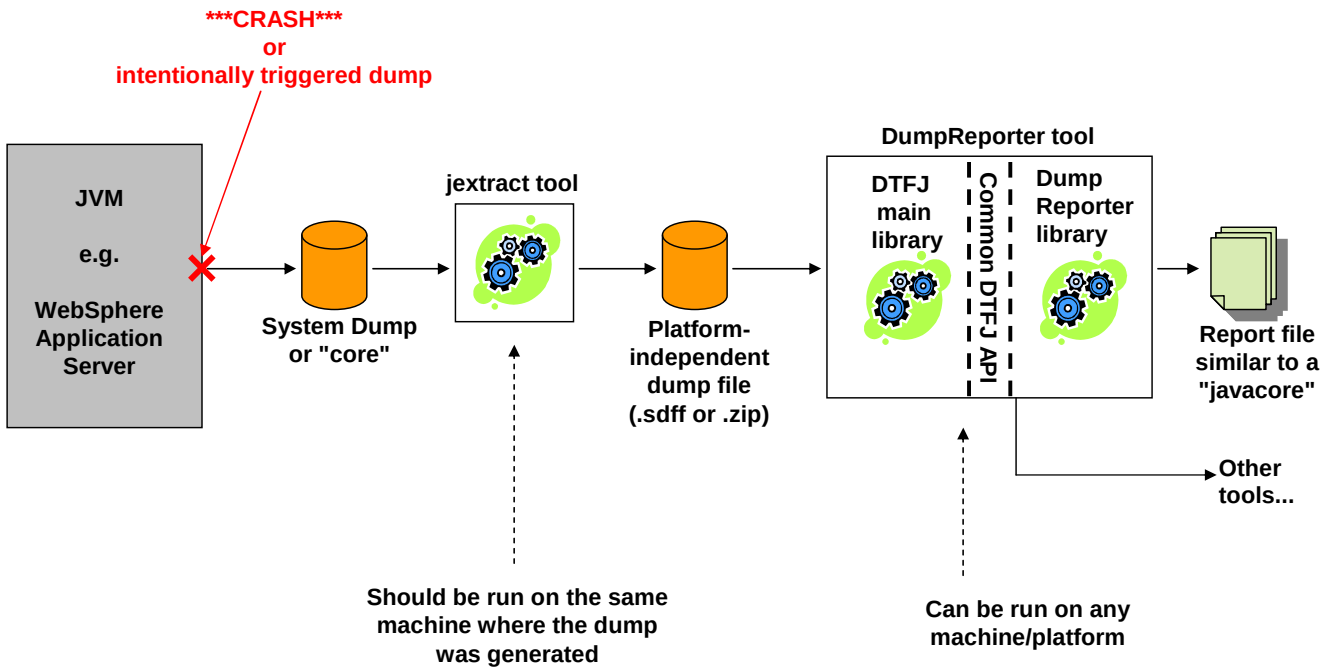
This topic describes the Diagnostic Tooling Framework for Java (DTFJ).

What is DTFJ?

- DTFJ (Diagnostic Tooling Framework for Java) is a new technology within the IBM JDK to build a variety of tools to analyze and diagnose problems in Java applications
- One of the first tools to be released within the DTFJ family is the *DTFJ-DumpReporter*
  - Reads a system dump from a JVM (for example, core file) and produces a text file similar to a javacore file
  - Useful in cases when the JVM crashes or hangs in such way that it does not produce the more familiar javacore file
- Many other DTFJ-based tools may be built in the future

The Diagnostic Tooling Framework for Java is a new technology within the IBM Java Development Kit to build a variety of tools to analyze and diagnose problems in Java applications.

How does DTFJ work?



DTFJ can examine a system core dump from a Java Virtual Machine, and produce a human-readable output file similar to a javacore.

DTJF-DumpReporter output example

```
DTFJ Extensible Analysis Library version 1.2.0.20060522-beta
DTFJ DefaultDumpReport version 1.2.0.20060522
Start report at Wed May 31 23:09:28 EDT 2006
Input dump file name: core.20060531.224507.17051.dmp.zip

Handling this dump as a J9 JVM dump

Image:
   Time of dump: [<unavailable>]
   System Type: Linux  -  SubType: 2.4.21-37.EL
   Processor Type: x86  -  SubType:
   Number of Processors: 1
   Installed Memory: 1051541504
   Host Name: [<unavailable>]  -  IP addresses: [<unavailable>]
   This Image contains 1 address spaces;  1 processes;  1 runtimes
[...]
Process: PID:17058
[...]
  Signal that triggered this dump: [<unavailable>]
   Current Thread: 17058

Java Runtime: JavaVM@08FC2D78
   Java Version: Java(TM) 2 Runtime Environment, Standard Edition(build 2.3
[...]
```


Similar to a javacore file, DTFJ lists the Java Virtual Machine's version information, command line, environments and loaded libraries.

Where is DTFJ supported?

- jextract + the main DTFJ runtime library are now shipped and supported with the standard IBM JDK
  - IBM JDK 1.4.2 SR4 and beyond -> WAS 5.1, 6.0
  - IBM JDK 1.4.2 SR4 for 64-bit platforms -> WAS 6.0.2
  - IBM JDK 1.5.0 SR1 and beyond -> WAS 6.1
  - On all IBM JDK platforms: AIX, Linux, Windows, z/OS
    - including 32-bit and 64-bit
- End-user tools must be obtained separately
- You must be on one of these supported platforms to run the DTFJ tools
  - But you may be able to process dumps generated on an older JDK version
    - within the same JDK family (i.e. use 1.4.2 DTFJ to process any dumps from 1.4.2; use 1.5.0 DTFJ to process any dumps from 1.5.0)
  - The more recent the JDK version, the more information jextract+DTFJ will be able to extract from that dump
- Not currently supported for non-IBM JDKs such as Sun and HP

End-user tools must be obtained separately, and DTFJ is not currently supported for non-IBM Java Development Kits.

How to access the DTFJ-DumpReporter tool

- Must be running on one of the supported JDK platforms/versions
- WebSphere® Application Server support is starting to use this tool
  - Available externally from WebSphere support
- Will be released through IBM Support Assistant shortly

To access the DTFJ-DumpReport tool, you must be running on a supported Java Development Kit platform & version.

How to use the DTFJ-DumpReporter tool

- Ensure that you have one of the supported JDKs on your path
- Assume that you have a file, core.1234, generated by a JVM (or java.dmp, or svcdump, and so on, for various platforms)
- **`jextract core.1234`**
  - Must be run on the same machine where the core was generated
  - Creates a file called core.1234.zip or core.1234.sdff
- **`java -jar DTFJ-DumpReporter-bin.jar core.1234.zip > Report.out`**
  - May be run on any machine
  - Creates a file Report.out
  - Examine Report.out with a text editor
- There are also various other switches to control the contents of the Report.out file; run the command without arguments to get the help text

To use DTFJ-DumpReporter, first apply jextract to the core file and then pass the results to the dump reporter tool.

Where does DTFJ come from and where is it going?

- DTFJ started as a collaboration between the Java Technology Center (JTC) and WebSphere Serviceability Team (WST)
- DTFJ is part of the strategic direction for Java diagnostics for IBM JDK:
  - Fully supported
  - Works on all IBM JDK platforms, especially the new JDK 1.5.0
  - Distributable to Customers
- For problems with jextract and the main DTFJ runtime library
  - Open a defect or PMR with the JTC
  - This is standard, fully-supported JDK code

DTFJ is part of the strategic direction for Java diagnostics for IBM Java Development Kit. It is fully supported, works on all IBM JDK platforms, especially the new JDK 1.5.0, and is distributable to customers.

Topic summary

Having completed this topic, you should be able to:

- Describe the Diagnostic Tooling Framework for Java
- Know how to convert a core dump using the DTFJ-DumpReporter

Having completed this topic, you should be able to describe the Diagnostic Tooling Framework for Java, and know how to convert a core dump using the DTFJ-DumpReporter.

Unit summary

Having completed this unit, you should be able to:

- Describe what a crash is and be able to detect one.
- Analyze javacore files for crash
- Use the DTFJ-DumpReporter

Having completed this unit, you should be able to describe what a crash is and be able to detect one, analyze javacore files for crash, and use the DTFJ-DumpReporter.

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 SW5706G16_Crashes.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 DB2 IBM WebSphere z/OS

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

Java, JDK, JNI, 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.