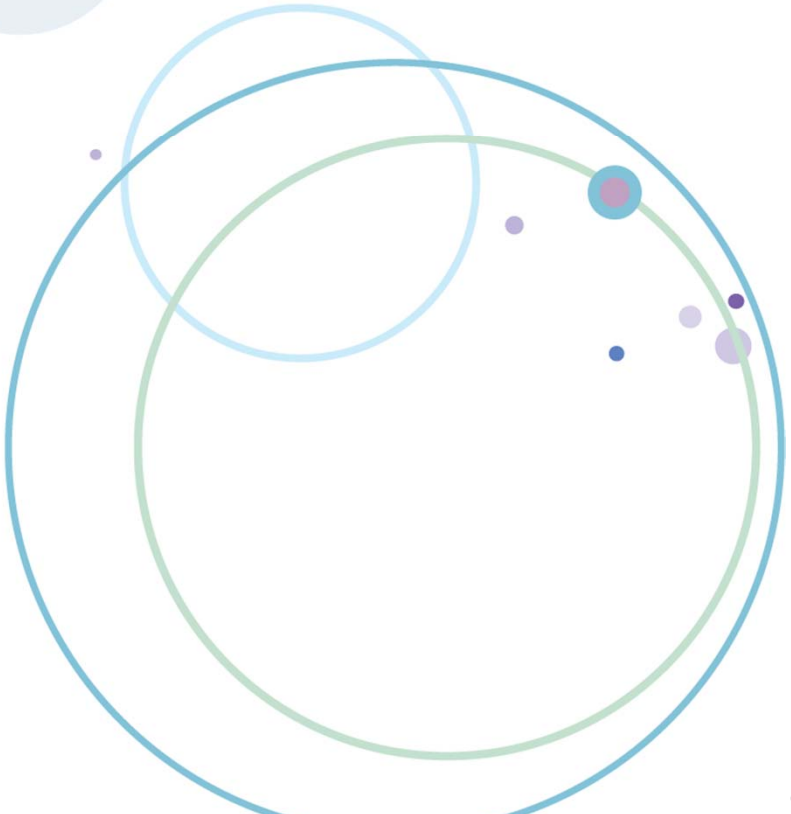
How to troubleshoot crashes



Unit objectives

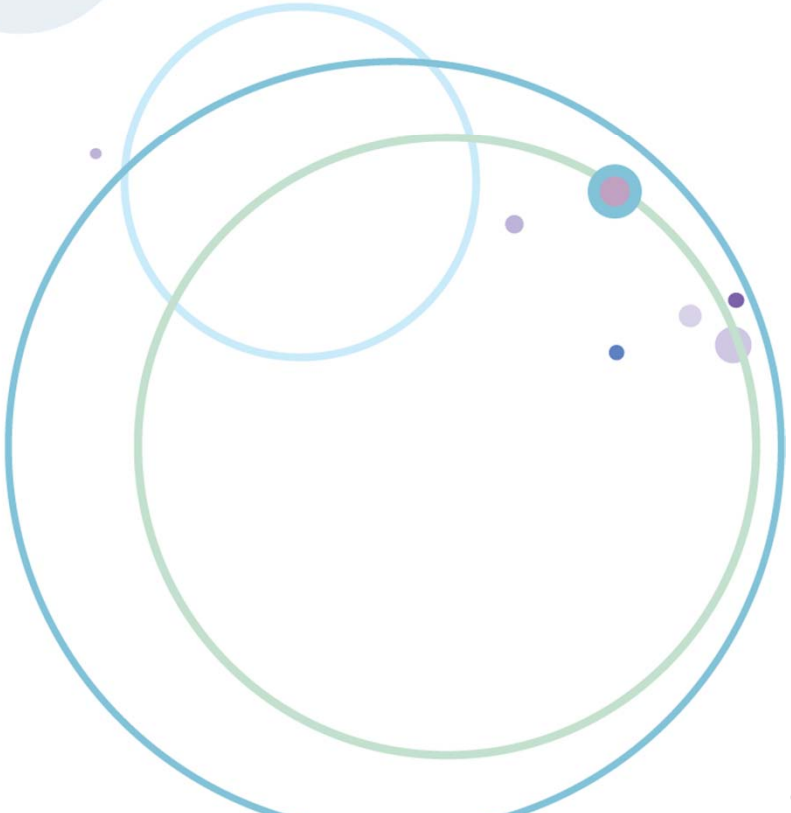
After completing this unit, you should be able to:

- Define what a crash is
- Detect a crash
- Analyze a javacore file for a crash
- Analyze system core files
- Describe the tools available for troubleshooting a crash
- Describe and use the IBM Monitoring and Diagnostic Tools for Java - Interactive Diagnostic Data Explorer

Topics

- Troubleshooting a crash
- Tools for troubleshooting crashes

Troubleshooting a crash



JVM process crash defined

- Not the same as thread hangs
- Symptoms
  - Process was stopped because of a Java exception or an OS signal
- Usual causes
  - Bad JNI call or library problem
  - Segmentation violations while running native code
  - Out-of-memory exception
  - Call stack overflow
  - Unexpected exception (for example, out of disk space)
  - Optimizer failure (for example, JIT)

Crash problem determination: Data to collect

- Javacore files
  - Also known as javadump or thread dump files
  - Text file that an application server creates during a failure
    - Can also be triggered manually
  - Error condition is given at the top of the javacore file
- Core files
  - Also known as process dumps or system core files
  - Created by underlying operating system
  - Complete dump of the virtual memory for the process
  - Can be large
  - Tools available to parse files into readable formats
- Steps to collect data
  - Look for a javacore file that is automatically created during a crash
  - If no javacore was generated, look for a system core dump

Make sure that a full core dump is produced

- Underlying operating system might have settings that prevent creation of a full core dump when a process crashes
  - For example, the **`ulimit`** on UNIX systems might specify a limit on the size of core file that is too small and the core dump is truncated
  - Some operating systems have a parameter to control the type of core files
  - On AIX, use the command "`lsattr -Elsys0|grep full`" to check whether fullcore is configured or not

- Make sure that a full core dump can be created before a problem occurs
  - Avoid recreation of problem, especially in a production environment

Diagnostics Collector

- You can configure the JVM to use the Diagnostics Collector to gather documentation and diagnostic data automatically after detecting a runtime problem such as a crash or out-of-memory condition.
  - Enable by using the generic JVM argument `-Xdiagnosticscollector`
- The Diagnostics Collector gathers system memory dumps, javacore files, heap memory dumps, verbose GC log (if present), and JVM trace files that match the time stamp for the Java problem that caused the collector to start
  - Outputs a single compressed file
  - For example, `java.gpf.<time_stamp>.<event_ID>.<pid>.zip`
- Available on IBM JDK 1.6 and 1.7
- For more details see the topic “The Diagnostics Collector” in the Java Diagnostics Guide 6

UNIX operating system common signals

- **SIGQUIT (kill -3)**
  - Indicates that a command was issued to generate a thread dump
  - Typically does not end the JVM process
- **SIGILL (kill -4)**
  - Means that an illegal instruction was issued
  - This signal 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 with JIT-compiled code
- **SIGSEGV (kill -11)**
  - Indicates an operation that is not valid in a program
  - Example: Accessing an illegal memory address
  - This signal is typically indicative of a programming problem in one of the native libraries

Windows operating system common signals

- **Memory access error**
  - Invalid memory address
  - JVM action: javacore file and stop the process
- **Illegal access error**
  - JVM action: javacore file and stop the process

Javacore subcomponents helpful for crash debugging

TITLE	Shows basic information about the event that caused the generation of the javacore file, the time it was taken, and the file name
GPINFO	• Shows some general information about the operating system • If the memory dump resulted from a general protection fault (GPF), information about the failure is provided; namely, the fault module is identified
ENVINFO	Shows information about the JRE level, details about the command line that started the JVM process, and the JVM environment
THREADS	Identifies the current thread and provides a stack trace

Javacore example that shows crash symptoms

```
0SECTION       TITLE subcomponent dump routine
NULL           ================================
1TISIGINFO     Dump Event "gpf" (00002000) received
1TIDATETIME    Date:                2007/09/25 at 15:26:44
1TIFILENAME    Javacore filename: C:\dev\javacore.20070925.152644.txt
0SECTION       GPINFO subcomponent dump routine
NULL           ================================
2XHOSLEVEL     OS Level          : Windows XP 5.1 build 2600 Service Pa
2XHCPUS        Processors -
3XHCPUARCH       Architecture    : x86
3XHNUMCPUS       How Many        : 2
1XHEXCPMODULE  Module: C:\WINDOWS\system32\msvcrt.dll
1XHEXCPMODULE  Module_base_address: 77C10000
1XHEXCPMODULE  Offset_in_DLL: 000378C0
{deleted lines}
0SECTION       THREADS subcomponent dump routine
NULL           ================================
1XMCURTHDINFO  Current Thread Details
3XMTHREADINFO      "Thread-1514" (TID:0x57BAD300,
sys_thread_t:0x429853AC, state:R, native ID:0x000037D0) prio=5
4XESTACKTRACE          at com/ibm/wa571/test/JniTest.setMessages(Native
Method)
```

C++ runtime library

Current thread is running JNI code

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 might be indicated, such as DB2

Javacore current thread details (JDK 1.4.2)

- 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
```

Crashes during JIT compilation

- To see whether the JIT is failing in the middle of a compilation, use the `verbose` option with the following extra settings:
  - `-Xjit:verbose={compileStart|compileEnd}`
- Also check `native_stderr.log` to see whether it identifies which method is causing the problem
- These verbose settings report when the JIT starts to compile a method, and when it ends
- If the JIT fails on a particular method (for example, it starts compiling, but crashes before it can end)
  - Use the `exclude={methodname}` parameter to exclude it from compilation
  - For example, `-Xjit:exclude={java/lang/Math.max(II)I}`
  - If excluding the method prevents the crash, you have an excellent workaround that you can use while the service team corrects your problem

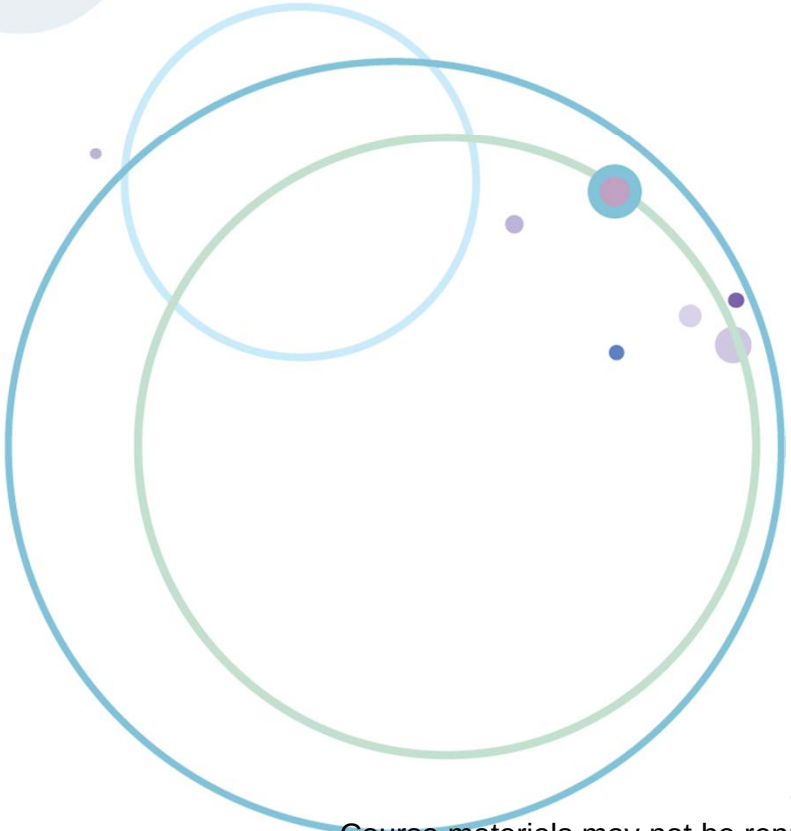
Steps if the cause of the crash is not identified

- Frequently, the javacore file does not clearly identify the cause of the signal
- Often the native stack shows the following message:

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

- Steps that you can take:
  - 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 whether this provides more information
  - Sometimes a bad Java SDK installation can cause problems

Tools for troubleshooting crashes



What is DTFJ?

- DTFJ (Diagnostic Tooling Framework for Java) is a new technology within the IBM JDK to analyze and diagnose problems in Java applications
  - Read RAS artifacts from a JVM (for example, a core file) and extract all kinds of useful information from that memory dump
- Not just one tool: an extensible framework for building many different tools
- By providing common framework, the use of specific tools for specific JVM artifacts is avoided

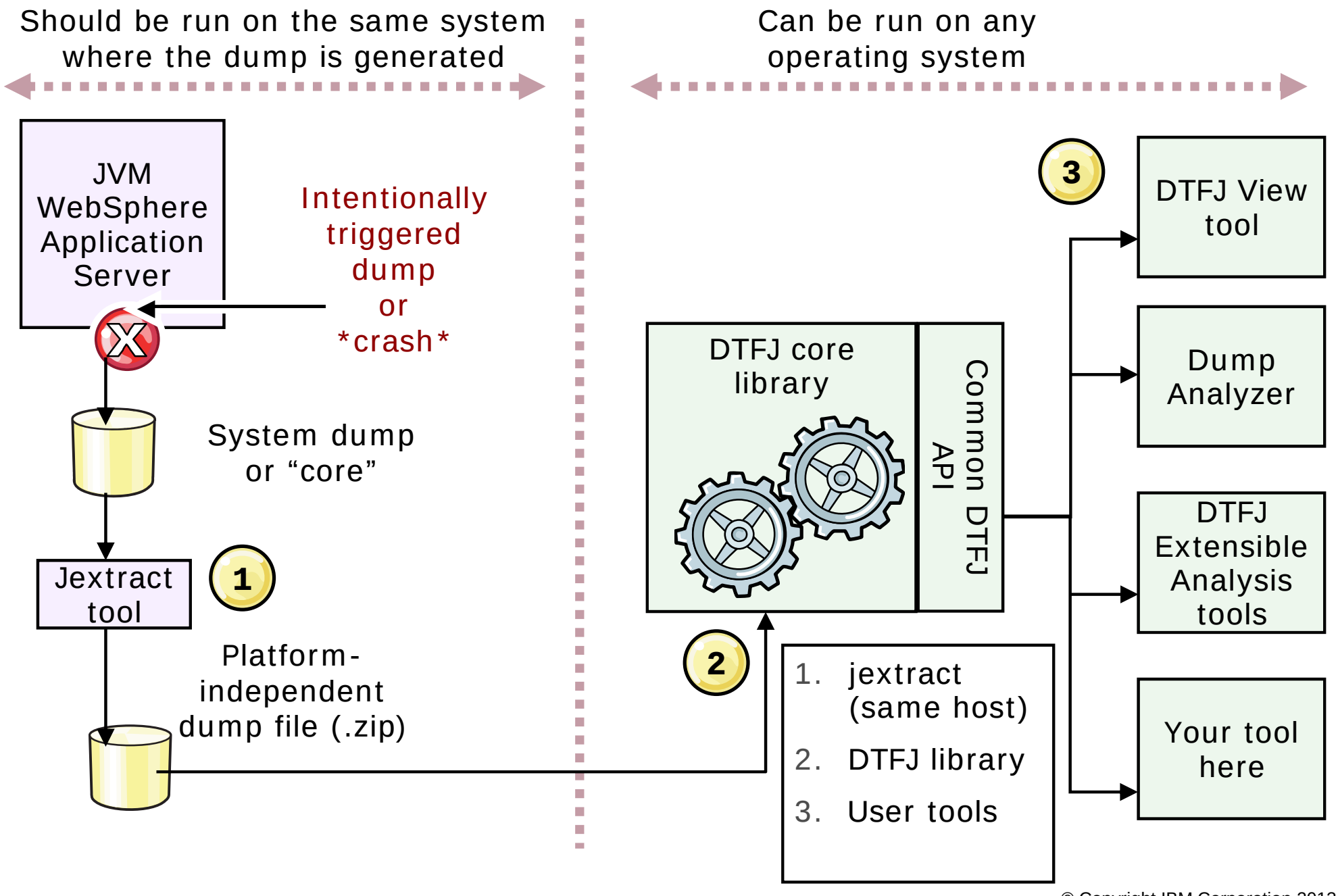
Components of the DTFJ family

- **`jextract:`** A tool to capture information from a JVM system memory dump (for example, core file) and package it into a platform-independent format

- DTFJ library or core library: A library that parses the contents of the system dump file that jextract packages, and provides access to its contents in a standardized manner, through a standard API

- DTFJ-based tools: A collection of tools that call the DTFJ library through the DTFJ API to present, and analyze information in various ways useful to the users

An example of using the DTFJ components



Where is DTFJ supported?

- jextract and the main DTFJ runtime library are now provided and the standard IBM JDK supports them
  - IBM JDK 1.4.2 SR4 and beyond: WebSphere Application Server 5.1, 6.0
  - IBM JDK 1.4.2 SR4 for 64-bit systems: WebSphere Application Server 6.0.2
  - IBM JDK 1.5.0 SR1 and beyond: WebSphere Application Server 6.1
  - IBM JDK 1.6.0
  - All IBM JDK systems: AIX, Linux, Windows, z/OS, iSeries, 32-bit, and 64-bit
- Tools must be obtained separately within IBM Support Assistant
- You might be able to process dumps generated on an older JDK version
  - Within the same JDK family (that is, 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 is able to extract from that dump
- Not currently supported for non-IBM JDKs such as Solaris and HP-UX

What is the Interactive Diagnostic Data Explorer?

- Interactive Diagnostic Data Explorer (IDDE) is a GUI-based alternative to the dump viewer (`jdmpview` command)
- IDDE provides the same function as the dump viewer, but with extra support such as the ability to save command output
- Use IDDE to more easily explore and examine dump files that the JVM produces
- Within IDDE, you enter commands in an investigation log to explore the dump file
- The support that is available in the investigation log includes the following items:
  - Command assistance
  - Auto-completion of text, and some parameters such as class names
  - The ability to save commands and output, which you can then send to other people
  - Highlighted text and flagging of issues
  - The ability to add your own comments
  - Support for using the Memory Analyzer from within IDDE

Using Interactive Diagnostic Data Explorer (IDDE)

- IDDE supersedes the Dump Analyzer tool
- Distributed and accessed through IBM Support Assistant
- Based on DTFJ, which provides cross system support
- Supports
  - Java core files
  - heap dumps (PHD file format)
  - system dumps (also known as core files)
  - Also supports multiple dump files that are contained in a compressed file, and dumps that were created on the z/OS operating system

IDDE features

Attempts to diagnose common JVM problems

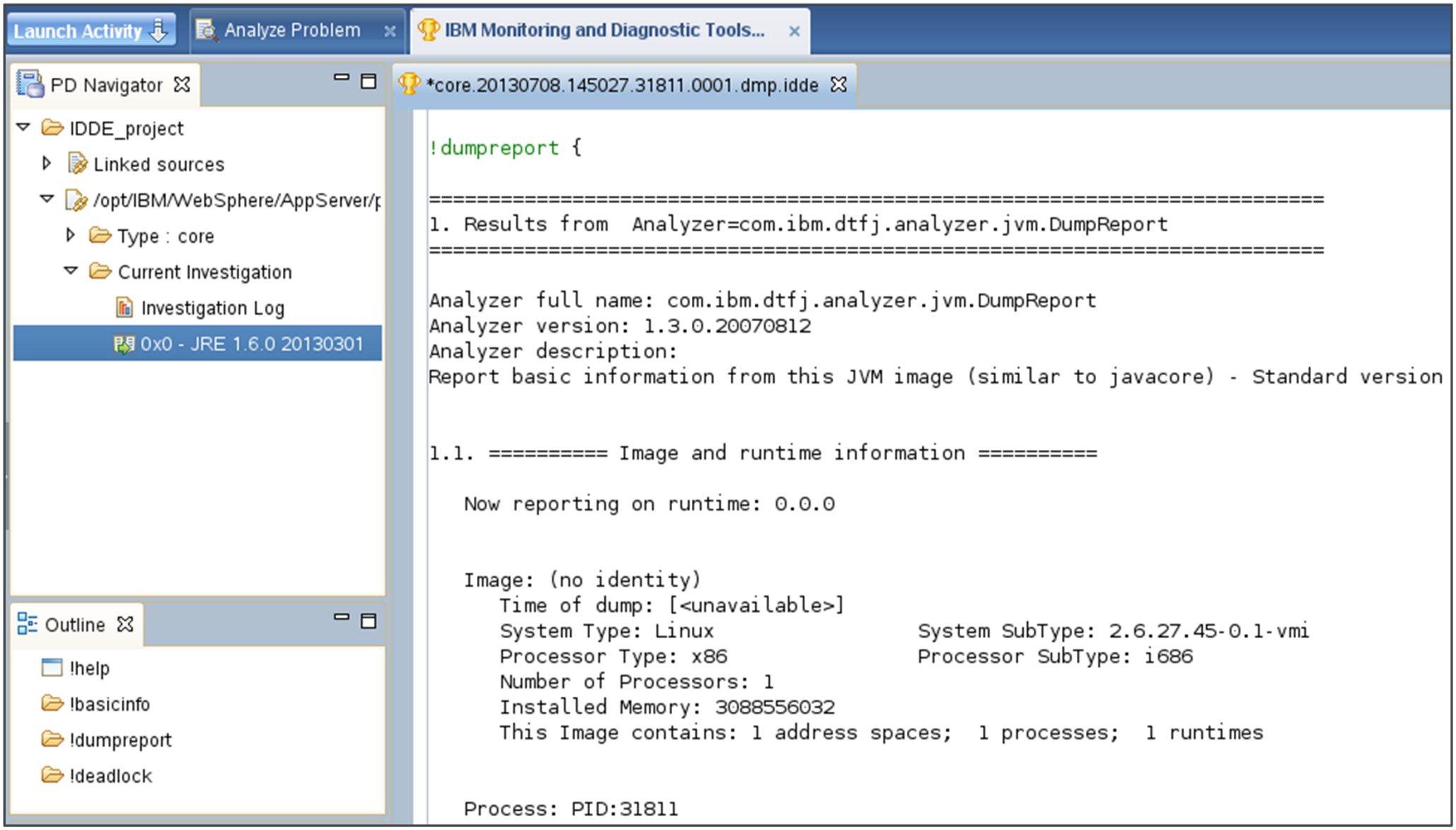
- Deadlock in Java code
  - Report thread names, locations, and other information
- Out-of-memory condition
  - Report populations and large collections, and large objects
  - Summarize the native memory usage
- Internal error (general protection fault, segmentation violation)
  - Is failure in non-IBM native code?
  - Probably use coding error, report location, and other details
  - If using JDK V5 or later, it might suggest running with **`-Xcheck:jni`** command-line option
  - Otherwise, call IBM Support

- Otherwise, it generates a default summary report
  - Suggested action is to call IBM Support and provide the output

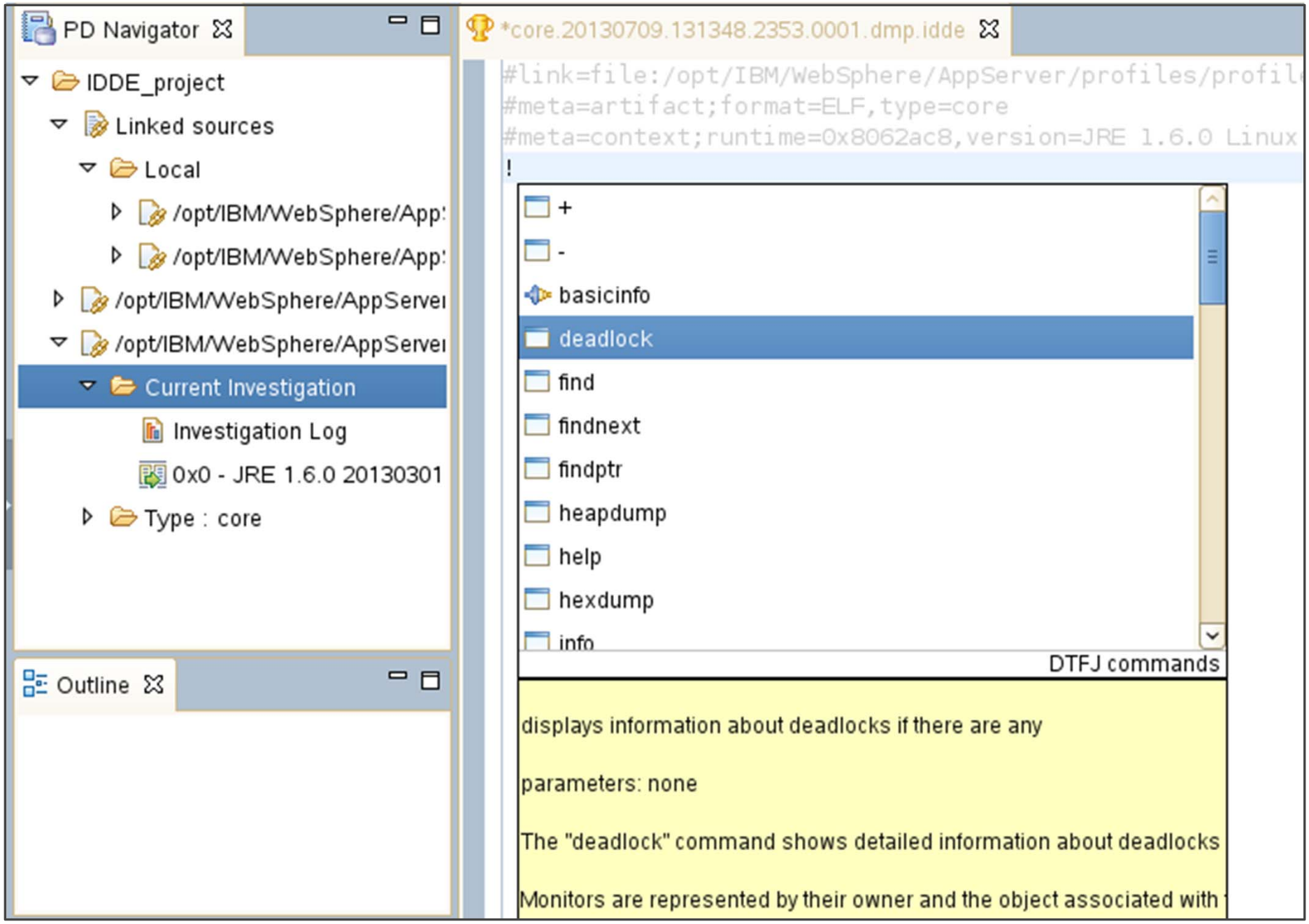
IDDE: jvm.DumpReport contents

- Basic information about the JVM process
  - Processor type, process ID, command line, JVM version, and other information
- JVM initialization arguments
  - System class path, heap tuning parameters, and other command-line options
- Environment variables
- Native libraries that are loaded in this process
- Threads (both Java threads and native threads)
  - Java thread ID, WebSphere Application Server thread ID, `java.lang.Thread object`, priority, and other thread details
  - Java stack, native stack
- Monitors and locks
- Heap memory layout

IDDE: Dump report

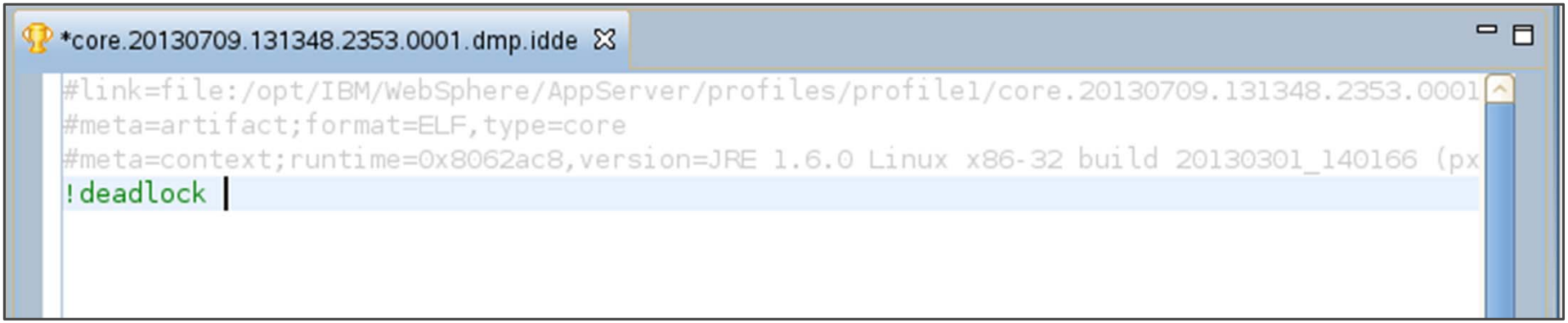


IDDE: Help menu for DTFJ commands

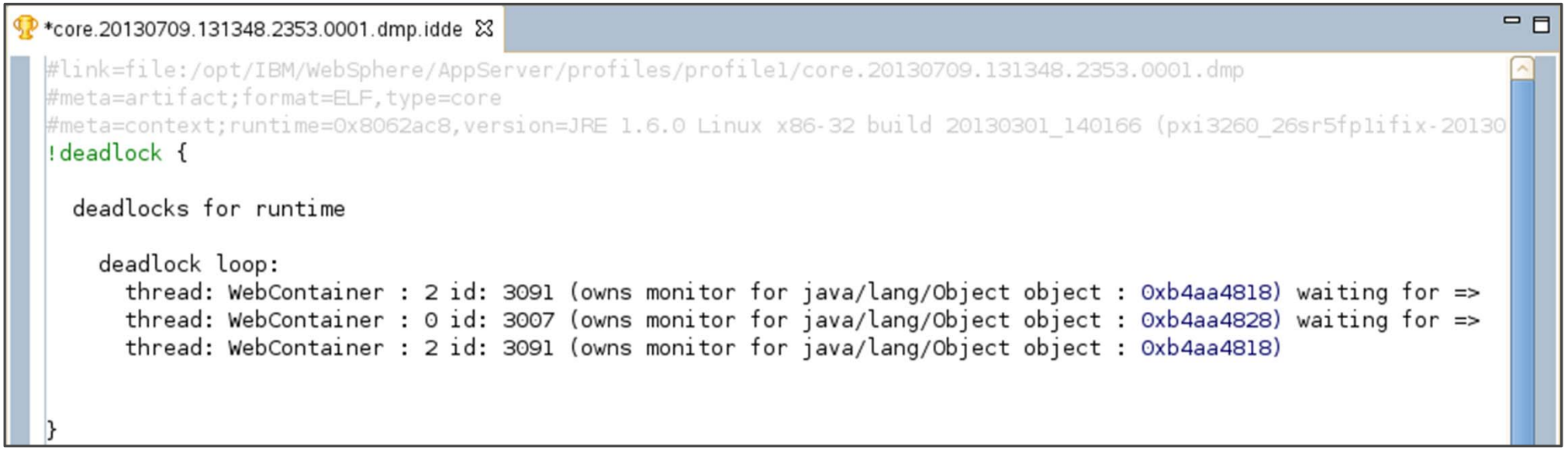


IDDE: Deadlocked threads detection

- Type `!deadlock` (or select from the command assistance menu)

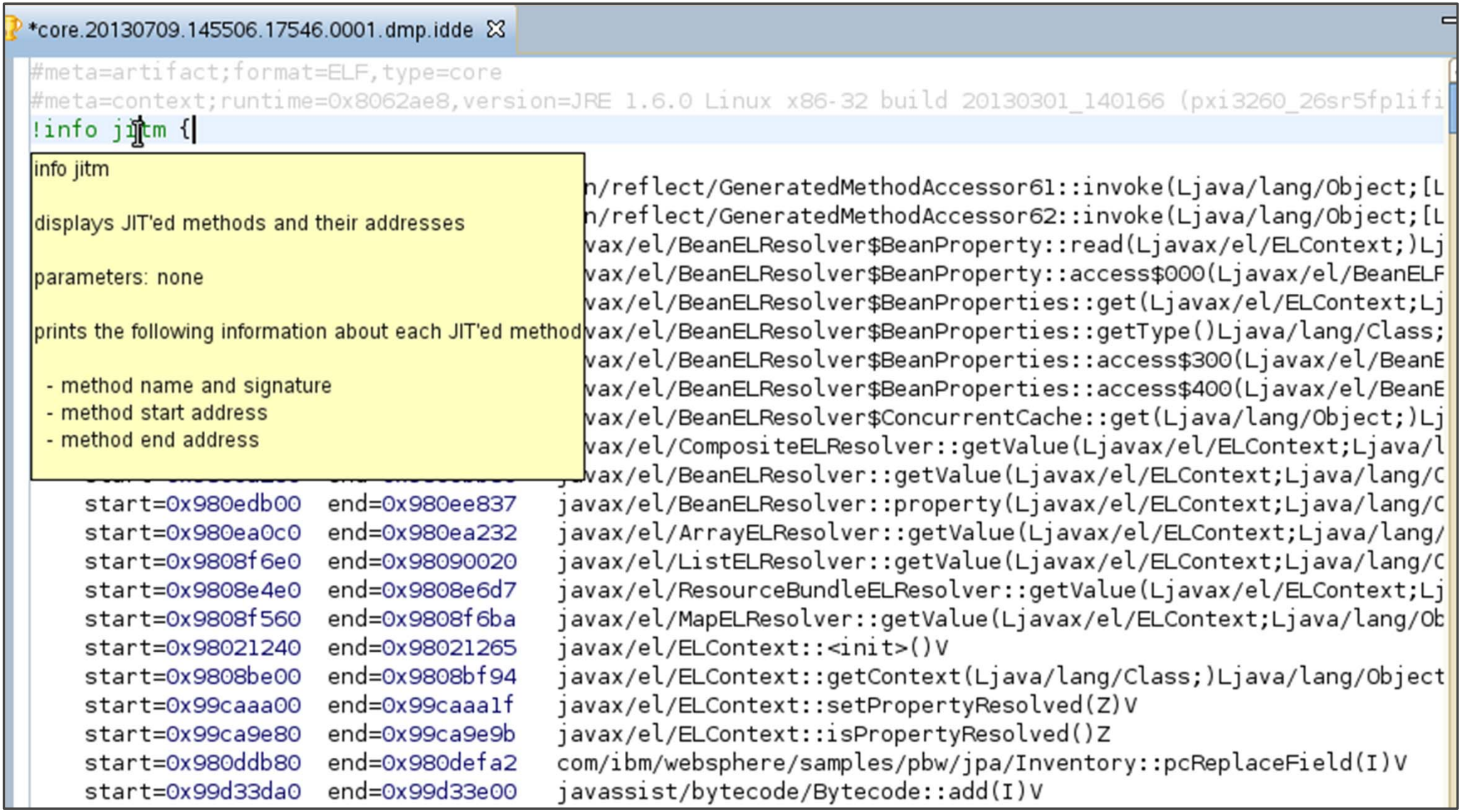


- Type Ctrl+Enter to run the command



IDDE: JIT compiled methods

- Use the `info jitm` command to list compiled threads



Unit summary

Having completed this unit, you should be able to:

- Define what a crash is
- Detect a crash
- Analyze a javacore file for a crash
- Analyze system core files
- Describe the tools available for troubleshooting a crash
- Describe and use the IBM Monitoring and Diagnostic Tools for Java - Interactive Diagnostic Data Explorer

Checkpoint questions

1. True or false: A common reason application servers crash is due to the problems with the JIT compiler.

2. True or false: A javacore file is always generated when an application server crashes.

3. True or false: System core files are in text format and can be analyzed manually by using a text editor.

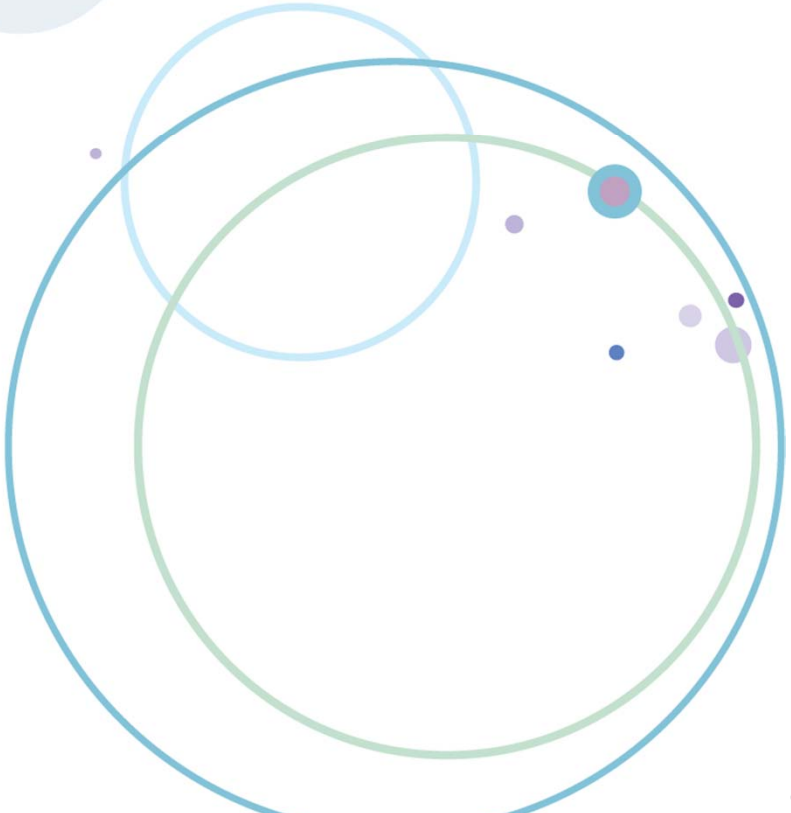
Checkpoint answers

1. True

2. False: Sometimes only a system core file is generated.

3. False: System core files are generated in binary format.

Exercise 5



Troubleshooting crashes

Exercise objectives

After completing this exercise, you should be able to:

- Analyze a javacore file for a crash
- Use various methods to trigger a system core dump
- Use the IBM Monitoring and Diagnostic Tools for Java - Interactive Diagnostic Data Explorer to analyze system core files