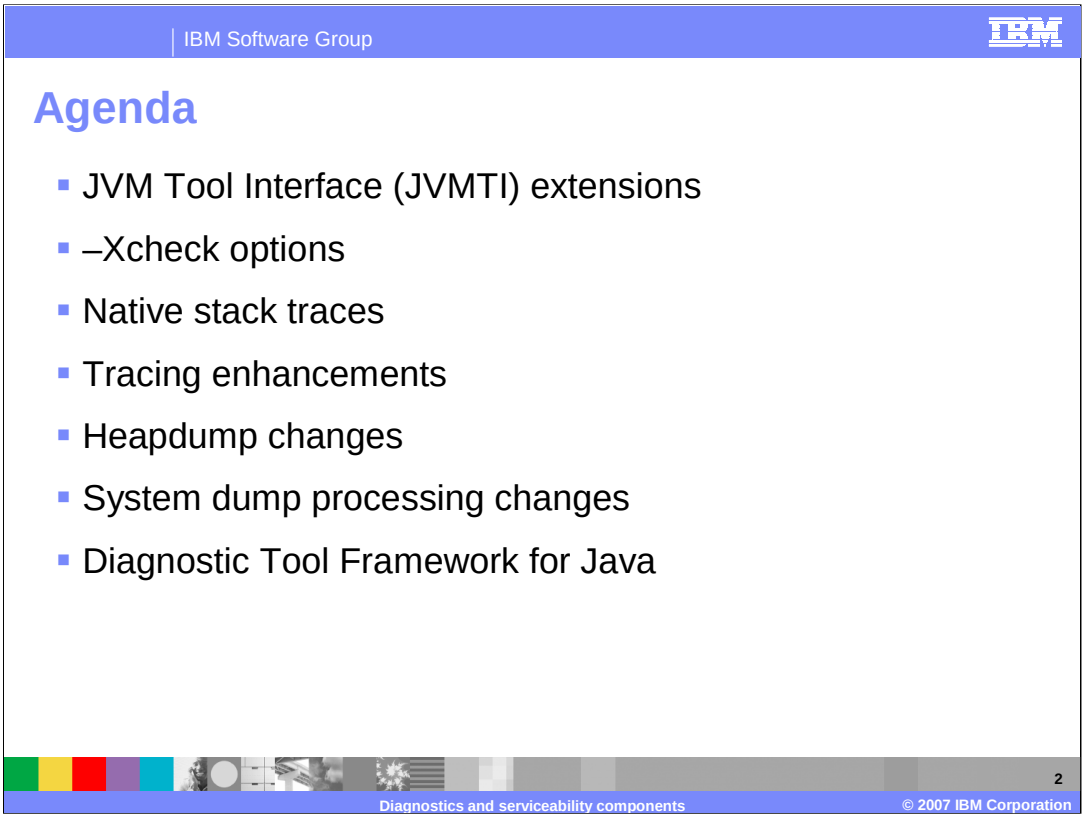
This presentation provides an overview of changes in the diagnostics and serviceability components in the IBM SDK for Java Version 6.



When you are experiencing issues with your Java applications, it is important to have solid diagnostic data available from the Java runtime environment to help you figure out what could be going wrong. The IBM SDK provides a number of built-in diagnostic components, many of which have been enhanced in Version 6 of the SDK. In addition to the industry standard JVM Tool Interface specification, IBM provides JVMTI extensions that support operations that are specific to the IBM SDK. You can include these extensions in your native code to produce helpful diagnostic data. The syntax for calling some command-line serviceability tools has changed in this release, with many commands being made available through a componentized –Xcheck parameter. New stack traces for failing threads are available in Javadumps and console dumps for certain types of failures. Additional tracepoints have been added through the SDK components to provide more detailed trace information in standard JVM traces. The default behavior for generating heap dumps has changed, and a new system dump viewer, which was developed on top of the Diagnostic Tool Framework for Java, has been packaged with the SDK.

Section

Diagnostic components

© 2007 IBM Corporation

This section will provide an overview of changes to the SDK's diagnostics and serviceability components.

JVMTI extensions

- The IBM SDK contains JVM Tool Interface extensions that support operations specific to the IBM SDK
  - The JVMTI extensions header file "ibmjvmti.h" is in sdk/include
  - See the JVMTI reference for full details
- New JVMTI extensions in Version 6
  - **com.ibm.TriggerVmDump** – method to trigger a virtual machine dump, specify the dump type, location, format
  - **com.ibm.ResetVmDump** – method to dynamically reset dump options to default
  - **com.ibm.VmDumpStart** – event provided when a dump starts
  - **com.ibm.VmDumpEnd** – event provided when a dump ends
- The extensions **com.ibm.SetVmTrace** and **com.ibm.SetVmDump** allow configuring dump and trace options at runtime
  - Were also available in Java 5

© 2007 IBM Corporation

In addition to the industry standard JVMTI specification, IBM provides extensions that support operations specific to the IBM SDK. You can use these extensions in your native code to perform dump and trace operations at runtime. To use the JVMTI extensions in your native code, you will need to use the header file ibmjvmti.h, located in the include directory of your IBM SDK package. There are two new JVMTI extension methods available in Version 6 – TriggerVmDump and SetVmDump. The TriggerVmDump method dynamically triggers a virtual machine dump. You can specify the type of dump you'd like to produce, the location for the dump file, and, in the case of Heapdumps, whether you would like a binary or text format file. The ResetVmDump method can be used to reset dump options at runtime. The virtual machine takes a snapshot of the dump options – including default virtual machine dump settings and initial user settings – before loading classes. When the reset method is called, dump settings get restored to the snapshot state. If you would like to monitor when dumps are being produced, two new events – VmDumpStart and VmDumpEnd – have been added to track dump start and end events. Two other JVMTI extensions – SetVmTrace and SetVmDump – were available in the previous release. These methods allow you to configure dump and trace options programmatically at runtime.

Understanding –Xcheck

- In previous releases, a variety of command-line tools have been available to help diagnose virtual machine or application problems
  - –Xrunjnichk, –memorycheck, and others
- In Version 6, these and other tools have been standardized as options to the –Xcheck parameter
  - Provides diagnostics for memory issues, classpath problems, JNI, and others
  - Old command-line options are no longer supported, and will be ignored
- Documentation of –Xcheck options is available in the Diagnostics Guide

© 2007 IBM Corporation

In previous releases, the IBM SDK provided multiple command-line tools to help diagnose problems in the virtual machine and in Java applications, like –Xrunjnichk, -memorycheck, and others. In Version 6, these tools are still available, but the syntax for calling them has changed. Command-line diagnostic tools that you invoke with your 'java' command have been consolidated under –Xcheck, so now you can invoke the JNI checking tool with the option –Xcheck:jni. The old command-line options, like –Xrunjnichk, are no longer supported and will be ignored. In addition to bringing existing tools together under the –Xcheck umbrella, in Version 6 there is a new classpath verification tool which you can run using –Xcheck:classpath. The -Xcheck:classpath option enables strict classpath checking, providing a warning if any classpath entry is missing or not readable. The –Xcheck tools and syntax are documented in the Diagnostics Guide.

Operating system stack backtraces

- Javadumps and console dumps now contain additional native stack trace information for failing native threads
  - Give library name, offsets, and where possible, function names
- Stack backtrace produced in Javadump for SIGSEGV, SIGILL or SIGFPE
  - In the console dump whenever it is produced
- Stack backtrace support varies by platform
- Example shows native stack trace from a console dump on AIX®

```
----------- Stack Backtrace -----------
./libj9ute24.so:0xD20BBC0C [0xD20AA000 +0x00011C0C]
./libj9ute24.so:0xD20BB634 [0xD20AA000 +0x00011634]
./libj9ute24.so:0xD20BC010 [0xD20AA000 +0x00012010]
./libj9ute24.so:0xD20BCE00 [0xD20AA000 +0x00012E00]
./libj9trc24.so:0xD1724320 [0xD171A000 +0x0000A320]
./libj9vm24.so:0xD1626078 [0xD161B000 +0x0000B078]
./libj9vm24.so:0xD161F1CC [0xD161B000 +0x000041CC]
./libj9vm24.so:0xD16294C0 [0xD161B000 +0x0000E4C0]
./libj9vm24.so:0xD16259FC [0xD161B000 +0x0000A9FC]
./libj9prt24.so:0xD20C88F0 [0xD20C5000 +0x000038F0]
./libj9vm24.so:0xD162C2A8 [0xD161B000 +0x000112A8]
./libj9vm24.so:0xD165B39C [0xD161B000 +0x0004039C]
j9:0x10002950 [0x10000000 +0x00002950]
j9:0x10001FA8 [0x10000000 +0x00001FA8]
./libj9prt24.so:0xD20C88F0 [0xD20C5000 +0x000038F0]
j9:0x10001F3C [0x10000000 +0x00001F3C]
j9:0x100001B4 [0x10000000 +0x000001B4]
---------------------------------------
```


© 2007 IBM Corporation

New in Java 6, some dumps contain stack backtraces that provide a full stack trace of the failing native thread. You can use the stack backtrace to determine if a failure is caused by an error in the JVM or the native application. A stack backtrace will be present in a console dump whenever one is generated and in Javadumps for SIGSEV, SIGILL and SIGFPE signals. The stack backtrace is platform dependent, and the information available varies by platform. Stack backtraces are not available on all platforms. The example on this slide shows a stack backtrace from a console dump on AIX. The trace provides information about the failing native thread, showing the native libraries that are being called and the sequence of function offsets in those libraries.

General tracing behavior

- Tracing behavior can be configured using –Xtrace options
  - Control where trace data will be stored and how much to capture
  - Trigger traces for specific classes, methods, and exceptions
  - Use `–Xtrace:what` to print the current tracing options
- Traces are written in binary format
  - Transform into human readable format using the trace formatter
  - **Example:** Invoke the trace formatter
    - `java com.ibm.jvm.format.TraceFormat <trace_file>`
- Additional tracepoints have been added in Java 6

© 2007 IBM Corporation

Methods for configuring JVM traces have not changed in Java 6 – you should still use –Xtrace options to configure your traces, and traces will still be produced in binary format. In the IBM SDK for Java 6, the number of internal tracepoints has increased substantially. In some cases, you might find that your trace files are over twice as large as traces for similar conditions in Java 5.

Heapdump updates

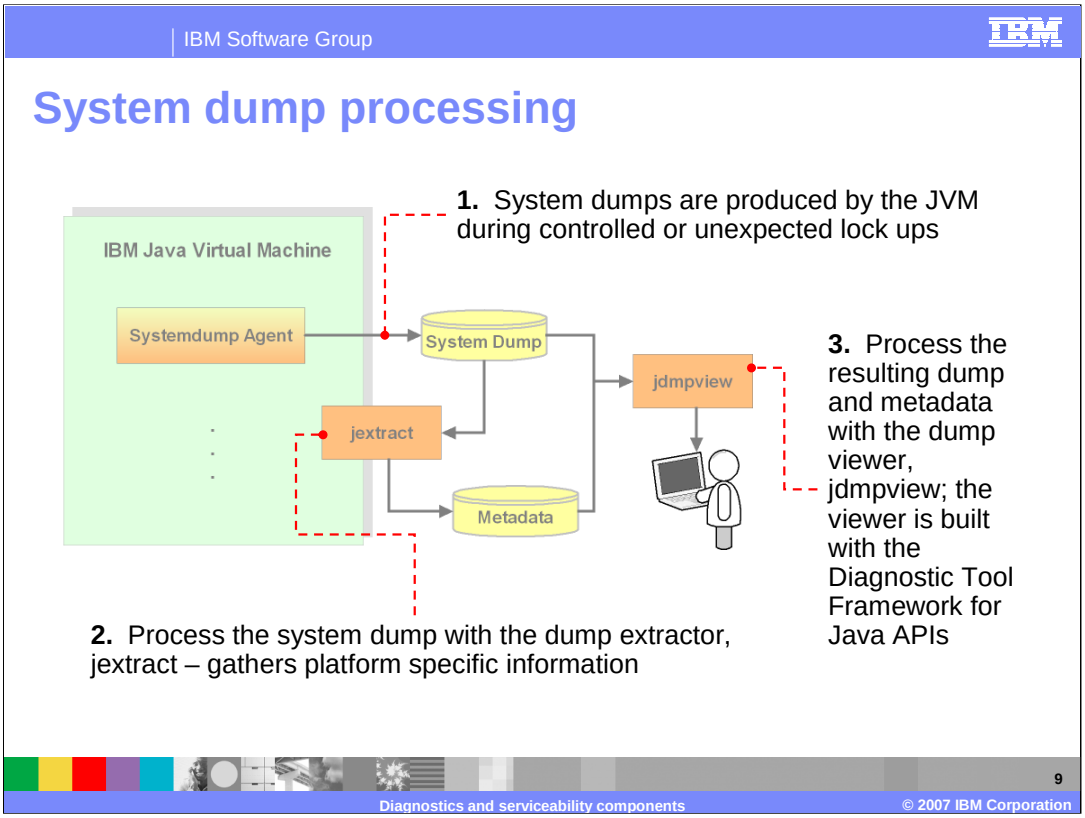
- Heapdumps are produced by default for java/lang/OutOfMemoryError events triggered by the virtual machine

```
-Xdump:heap:events=systhrow,
        filter=java/lang/OutOfMemoryError,
        label=C:\test\heapdump.%Y%m%d.%H%M%S.%pid.phd,
        ...
```

- Binary Heapdump compression
- Heapdump generator produces dump files faster

© 2007 IBM Corporation

In Version 5, Heapdumps were produced by default for any uncaught OutOfMemoryError event, including OutOfMemoryErrors produced in user code. The default behavior has changed in Version 6, so that Heapdumps are produced for OutOfMemoryErrors triggered by the virtual machine. The new systhrow event has been added in this release to support this new Heapdump behavior. The example on the slide shows a portion of the default dump agent configuration for Heapdumps. Also new in Java 6, the Heapdump generator takes advantage of compression capabilities in the binary dump file format and uses a file writing cache scheme to produce smaller Heapdump files and write them out to disk faster than in the previous release.



This diagram illustrates the system dump processing flow, using built-in components and commands in the IBM SDK. First, a system dump agent that is configured to monitor for certain events – like when a JVM has a general protection fault – will trigger a dump when those events occur. Second, before you try to analyze your system dump with any other tools, the dump file should be fed into the dump extractor, for pre-processing. The dump extractor will pull out useful metadata about your Java operating environment and create an archive file that contains the metadata and your system dump file, which you can then use in a system dump processing tool. Whenever possible, you should run the dump extractor on the system where the dump was produced. If that is not an option, you can try running the dump extractor on a machine that is running the same operating system and JVM level. Third, take the output from the dump extract and use it as input to the dump viewer, jdmpview. The dump viewer is a command-line debugger that allows you to explore the contents of your system dump. The dump viewer in Java 6 is build on the Diagnostic Tool Framework for Java APIs.

Diagnostic Tool Framework for Java

- Diagnostic Tool Framework for Java (DTFJ) is a Java API used to build diagnostic tools for Java programs
- Allows people to write tools without needing to understand the exact structure of dumps
- Can process system dumps and Javadumps
  - The system dump viewer packaged with the SDK is built on the DTFJ

© 2007 IBM Corporation

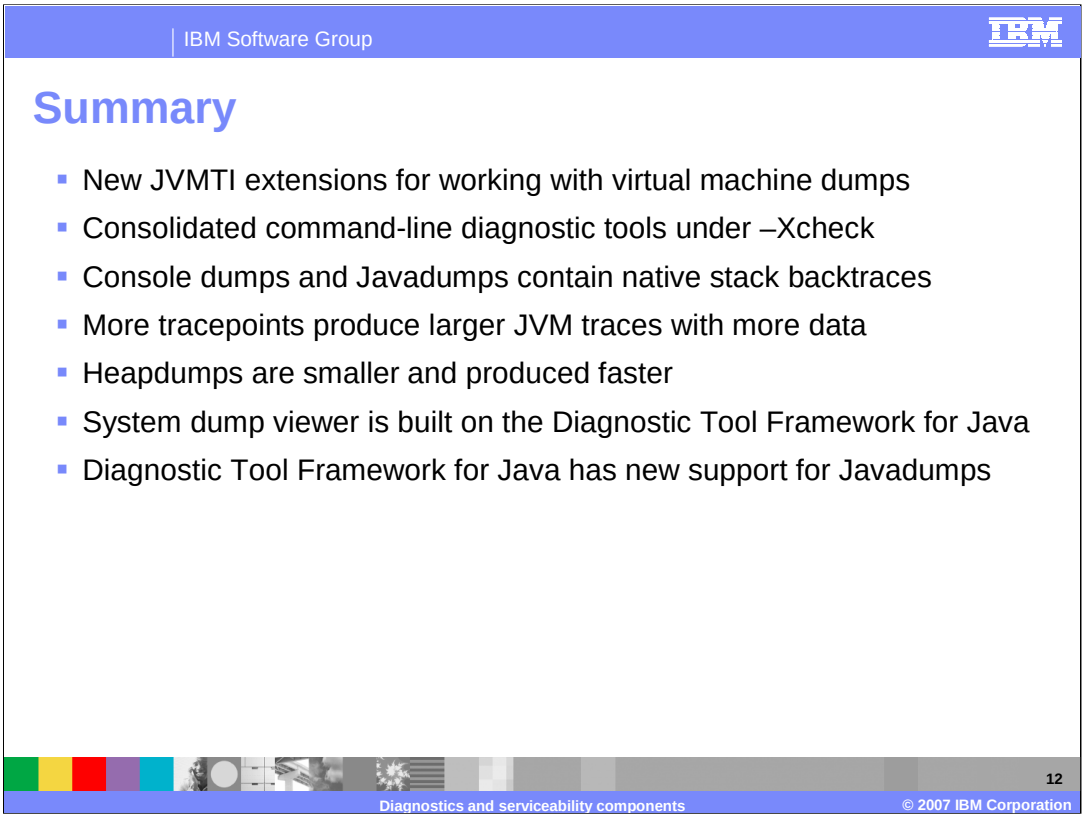
The Diagnostic Tool Framework for Java acts as a layer of abstraction between a tool developer and the underlying structure of diagnostic data in the virtual machine. The DTFJ APIs allow Java tool developers to access data in a dump, like the Java version, threads, and heap data, without needing to understand the exact structure of the dump itself. DTFJ is implemented in pure Java and tools written using DTFJ can be cross-platform. Therefore, it is possible to analyze a dump taken from one machine on another (remote and more convenient) machine. To work with a system dump, the dump must first be processed by the dump extractor, jextract. The jextract tool produces metadata from the dump, which allows the internal structure of the JVM to be analyzed. It is recommended that you run jextract on the system that produced the dump. If that is not possible, you can use a system that is running the same operating system and virtual machine level as the system that produced the dump. In Version 6, DTFJ support has been added for Javadumps. To work with a Javadump, no additional processing is required.

Section

Summary and references

© 2007 IBM Corporation

This section contains a summary and references.



The IBM SDK for Java includes a broad range of built-in diagnostic and serviceability components, many of which have been enhanced in the Version 6 release. New JVMTI extensions are available that allow you to monitor dumps, trigger dumps, and reset dump options at runtime. Existing command-line diagnostic tools, like –Xrunjnichk and –memorycheck, have been consolidated under the –Xcheck parameter, which provides standardized syntax for these tools. A new classpath validation utility is also available under –Xcheck:classpath. New native stack backtraces are available in console dumps and Javadumps. These stack backtraces provide useful information for debugging failures in native threads. Internal JVM components have been instrumented with more tracepoints in this release, so JVM traces will include more data in Version 6 than was available in Version 5. The default conditions for producing Heapdumps has been changed to trigger when memory exhaustion is reported by the virtual machine. Heapdump files are also more compressed and can be written out to file faster than in the previous release. The Diagnostic Tool Framework for Java has been enhanced to provide support for Javadumps, in addition to the support that was previously in place for processing system dumps. The system dump viewer that is packaged with the SDK is built on these DTFJ APIs.

References

- IBM Support Assistant
  - http://www.ibm.com/software/support/isa/
- Diagnostics guide
  - http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp

© 2007 IBM Corporation

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_Java6_RAS_Overview.ppt

This module is also available in PDF format at: ../Java6_RAS_Overview.pdf

© 2007 IBM Corporation

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 IBM

Java, JNI, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

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.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2007. All rights reserved.

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.