This presentation will act as an introduction to troubleshooting hangs when using WebSphere® Application Server V6.

Application server hangs

After completing this topic, you should be able to:

- Describe what a hang is
- Detect a hang condition
- Analyze javacore for hung threads

The first section of this unit will concentrate on describing and detecting hang conditions in WebSphere Application Server V6.

Application server hang defined

- Clarify the nature and extent of the hang
  - A “hang” is not the same as a “crash”
  - Is entire process is truly hung, or does it still respond to some requests?
    - Test with sample “Snoop” servlet, wsadmin commands, and so on.
  - Deadlocks:
    - Very often, one process fails to respond to a request because it has made a call to another process that is itself hung; sometimes is hard to find the true culprit.
    - Deadlocks also may occur within the same Java process, where one thread is deadlocked on another.

A hang can be defined as a process or thread which has become unresponsive while still apparently alive. Contrast this with a crash, when a process abnormally ends, hopefully with an error message. Resources normally available may be tied up by unbounded code paths, such as when the code is running in an infinite loop. Alternately, a system can become unresponsive even though all resources are idle, as in a deadlock scenario. Other applications and functions in the same JVM may still work. The problem may be contained entirely within one process, or it may involve multiple processes, as when many threads in the server are waiting for some response from the database. You may need to look at the other remote processes to fully understand what's going on in that case.

Thread hang examples

- Understand root causes
  - Customer code enters an infinite loop such as:

```
while (true) {
  i++;
}
```

  - Customer code waits infinitely such as:

```
run() {
  object1.wait();
}
```

  - Customer code creates a deadlock such as:

```
synchronized (object1) {
  synchronized (object2) {
    // Work with objects
  }
}
```

```
synchronized (object2) {
  synchronized (object1) {
    // Work with objects
  }
}
```


Thread hangs can arise in a number of scenarios, including when the application code enters an indefinite loop, waits indefinitely, or creates a deadlock or “deadly embrace”. Detecting if a thread is hung or just taking a long time to respond is a difficult problem to solve correctly. There are tools available and WebSphere has a built-in monitor to assist in identifying hung threads. These tools and the monitor will be discussed further in the unit. Using these facilities, it can be a simple process to identify when threads are hung.

WebSphere process hang detection steps

- Obtain a thread dump or javacore
  - If the process is still responsive to JMX commands, then wsadmin or the Thread Analyzer should be able to trigger the dump
  - Otherwise, may need to trigger through lower-level OS functions
    - On UNIX®, send a “kill -3” signal
    - On z/OS®, from console enter “F <servername>,JAVACORE”
    - If that fails, may need even lower-level functions (such as dbxtrace)
- For a typical hang, collect three dumps at a few minutes interval
  - To see if anything is moving within the process (but slowly)
- Examine the thread dumps with Thread Analyzer or by hand
  - Look for deadlocks
  - Look for threads that are waiting after sending a request to some other process, now awaiting a response

The basic problem determination method for hangs is to obtain one or, if possible, a series of thread dumps. If the process is still responsive to JMX™ commands, then the Thread Analyzer or a wsadmin command should be able to trigger the dump. Otherwise, depending upon your operating system, certain signals will trigger a thread dump. For a typical hang, collect three dumps at 5 minute intervals to determine if anything is moving within the process (albeit slowly). Examine the thread dumps with Thread Analyzer or by hand to look for deadlocks or to see if threads are awaiting responses from other processes. In newer JVMs, the javacore or thread dump will automatically perform deadlock detection and tell you if a deadlock has been detected. Look for the string "deadlock" in the javacore file.

Javacore hang indicators

- JVM monitor information
  - Shows synchronization locks
  - Indicates blocked threads
- Active threads
  - Look for running threads indicated by
    - state:R
    - Example:

```
"Servlet.Engine.Transports:239" (TID:0x34B94018, sys_thread_t:0x7CD4E008, state:R, native ID:0x10506) prio=5
  at
  java.net.SocketInputStream.socketRead(Native Method)
  at
  java.net.SocketInputStream.read(SocketInputStream.java(Compiled Code))
  at
  com.ibm.ws.io.Stream.read(Stream.java(Compiled Code))
  at
  com.ibm.ws.io.ReadStream.readBuffer(ReadStream.java(Compiled Code))
```


The monitor information in the javacore file shows what synchronization locks are held by which threads. It also shows which threads are blocked by monitors. This information is useful for determining the cause of a deadlocked or hung JVM. The monitor information is in a section entitled LK subcomponent dump routine. It is before the thread dump of all the threads of the JVM. A large number of threads blocked on a monitor does not mean a deadlock has occurred. It might mean that there is a monitor (synchronization lock) that is causing a backlog of work to be completed. The javacore processing dumps the current stack for every thread in the JVM. It shows the current state of the thread and produces a stack trace.

The thread state indicates if the thread is currently runnable or not. If the thread state is state:R, the thread is runnable. The thread state CW, for Conditioned Wait, indicates a thread that is in a wait state. The call stack under the thread header is the Java stack. This shows the Java calls that have been made to get the thread to its current state. The first line in the Java stack is the last Java method call that was made. It was from that location that a call into a native method might have been made. That is typically identified with the phrase **Native Method** showing the location in the Java program that was called. The native stack shows what native methods or procedures were called after the thread entered the native code. The first line in the native stack shows what the thread was doing in native code when the javacore was taken.

Javacore hang symptoms

- Check to see if threads are blocked waiting on monitors
  - May indicate deadlock
- If threads are in running state
  - Check method across multiple javacores
  - If individual threads in same method
    - may indicate looping logic
- If threads are in wait states
  - May indicate that a resource is causing hang

Threads which are blocked waiting on monitors may indicate a deadlock. Similarly, threads which are in wait states may indicate a resource causing a hand. If the threads are in a running state, observe their behavior over multiple javacores. If the individual threads are in the same method, it may indicate a loop.

Javacores contain a lot of information and may cover dozens of threads. It is recommeded that tools be used to process the javacore such as the

Thread Analyzer and the Thread Monitor. These tools will be covered in more details in the following sections.

Hung thread detection

After completing this topic, you should be able to:

- Describe the features of the thread monitor for hang detection.
- Know how to configure and view the output from the WebSphere hang thread detection facility

The first section of this unit will concentrate on the usage and features of the Thread Monitor in WebSphere Application Server V6.

WebSphere hung thread detection

- WebSphere contains a built-in hung thread detection function
- ThreadMonitor architecture was created to monitor thread pools within WebSphere
  - The ThreadMonitor monitors Web Container, ORB, and Async Bean thread pools
  - Enabled by default
- Unmanaged threads, which are threads created by applications, are not monitored.
- Upon notification of a hung thread:
  - Obtain a javacore and see what the thread is doing
  - Investigate the nature of the thread

WebSphere Application Server V6 contains a built-in hung thread detection function. It monitors the web container, ORB, and Async Bean thread pools, and is enabled by default. Note that unmanaged threads are not monitored. You can configure a hang detection policy to accommodate your applications and environment so that potential hangs can be reported, providing earlier detection of failing servers. When a hung thread is detected, WebSphere Application Server notifies you so that you can troubleshoot the problem. When notified, you should obtain a javacore to investigate the nature of the thread and to determine if the behavior is normal.

Hung thread detection internals

- When the thread pool gives work to a thread, it notifies the thread monitor
  - Thread monitor notes thread ID and timestamp
- Thread monitor compares active threads to timestamps
  - Threads active longer than the time limit are marked “potentially hung”
- Performance impact is minimal (< 1%)

When the thread pool issues work to a thread, it sends a notification to the thread monitor, which notes the thread ID and the time in a list. At user-configurable intervals, the thread monitor looks at the active threads, and compares them to the list, to determine how long each thread has been active. If a thread has been active longer than the user-specified threshold, the thread is marked as “potentially hung”, and the notifications are sent.

Hung thread detection notification

- No action taken to kill the thread--only a notification mechanism
- When a thread is suspected to be hung, notification is sent three ways:
  - JMX notification for JMX listeners
  - ThreadPool metric for PMI clients
    - Counters are updated
  - Message written to SystemOut.log:

```
[4/17/04 11:51:30:243 EST] 2d757854 ThreadMonitor W WSVR0605W: Thread
Servlet.Engine.Transports : 0 has been active for 14,198 milliseconds
and may be hung.  There are 1 threads in total in the server that may be
hung.
```


The thread monitor doesn't try to deal with the hung threads, it just issues notifications, so that the administrator or developer can deal with the issues. When a hung thread is detected, three notifications are sent: a JMX notification for JMX listeners, PMI Thread Pool data is updated for tools like the Tivoli Performance Viewer, and a message is written to the SystemOut log. The JMX notification enables vendor tools to catch the event and take appropriate action, such as triggering a JVM thread dump of the server, or issuing an electronic page or e-mail. The message written to the SystemOut log has a message ID of WSVR0605W, and shows the thread name, the approximate time that the thread has been active, and the total number of threads which may be hung.

Hung thread detection false alarms

- What about false alarms?
  - For example: a thread that takes several minutes to complete a long-running query
- If a thread previously reported to be hung completes its work, a notification is sent:

```
[2/17/04 11:51:47:210 EST] 76e0b856 ThreadMonitor W
WSVR0606W: Thread Servlet.Engine.Transports : 0 was
previously reported to be hung but has completed.  It
was active for approximately 31,166 milliseconds.
There are 0 threads in total in the server that still
may be hung.
```

- The monitor has a self-adjusting system to make a best effort to deal with false alarms.

It's possible that a thread could actually be running for longer than the specified threshold for legitimate reasons. For example, a thread could be executing a large database query that takes several minutes to return.

The thread monitor is built to recognize false alarms and adjust itself automatically. When a thread that was previously marked as "potentially hung" completes its work and exits, a notification is sent. After a certain number of false alarms, the threshold is automatically increased by 50% to account for these long-running threads. The idea is that if there are several threads that are routinely active for 20 minutes, the threshold will eventually adjust itself to be higher than 20 minutes, so as to not mark those threads as hung.

Hung thread detection configuration

- To configure access: **Servers** > **Application Servers** > *server_name* > **Administration** > **Custom Properties**
- Create custom properties on the application server:

Property	Units	Default	Description
com.ibm.websphere.threadmonitor.interval	secs.	180	interval at which the thread pools will be polled for hung threads
com.ibm.websphere.threadmonitor.threshold	secs.	600	the length of time that a thread can be active before being marked as “potentially hung”
com.ibm.websphere.threadmonitor.false.alarm.threshold	N/A	100	the number of false alarms that can occur before automatically increasing the threshold by 50%.

The hang detection policy can be configured by creating custom properties for the application server.

* com.ibm.threadmonitor.interval is the interval at which the thread pools will be polled for hung threads (in seconds). It defaults to 180 seconds, which is 3 minutes.

* com.ibm.websphere.threadmonitor.threshold is the length of time that a thread can be active before being marked as “potentially hung”. The default value is ten minutes.

* com.ibm.websphere.threadmonitor.false.alarm.threshold is the number of false alarms that can occur before automatically increasing the threshold by 50%. The default value is 100. Automatic adjustment can be disabled altogether by setting this property to zero. The application server must be restarted for these changes to take effect. To adjust the hang detection policy on the fly, use wsadmin. Refer to the Information Center for instructions.

To disable the hang detection option, set the com.ibm.websphere.threadmonitor.interval property to less than or equal to zero.

Thread Analyzer

After completing this topic, you should be able to:

- Describe Thread Analyzer's features and interface
- Describe Thread Analyzer's uses in problem determination

The final section of this unit will concentrate on the features and usage of the Thread Analyzer tool.

WebSphere Thread Analyzer

- A Stand-alone GUI tool to analyze the Javacore file produced by the JVM
- Helps non-Javacore/WebSphere experts analyze the Javacore
- Gathers and analyzes thread dumps from a WebSphere Application Server
- Thread dumps are important for several common problem areas:
  - Application Server crashes
  - Application Server hangs – for example, deadlocks
  - Performance bottlenecks – for example, many threads in contention
- Can detect some deadlock conditions and provides recommendations based on analysis

The analysis of thread dumps or javacore files is an important skill in problem determination in the event of server crashes, hangs, or performance bottlenecks.

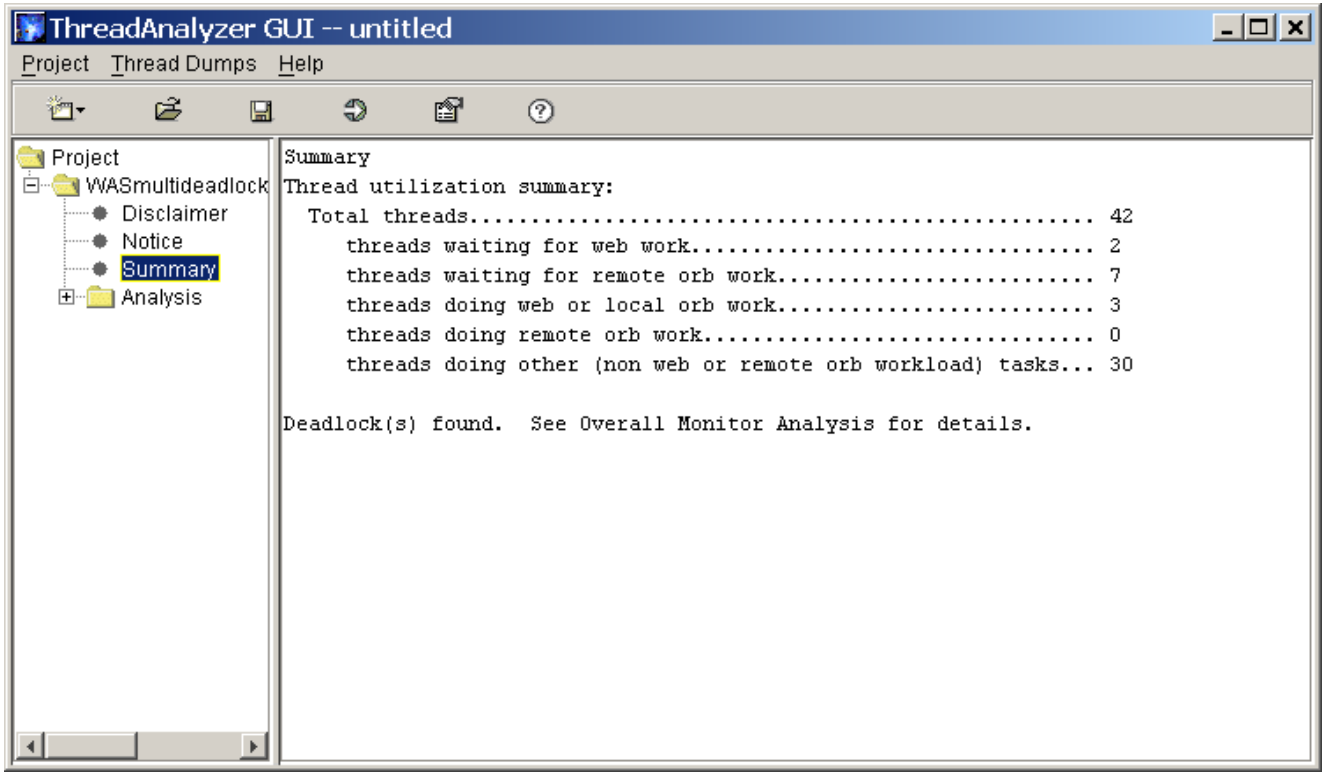
The Thread Analyzer is a stand-alone gui tool used to help gather and analyze thread dump files produced by WebSphere Application Server. Thread Analyzer can obtain a thread dump or open an existing thread dump. Thread usage can be analyzed at several different levels, starting with a high-level graphical view, and drilling down to a detailed tally of individual threads. If any deadlocks exist within the thread dump, Thread Analyzer will detect and report them. The Thread Analyzer is available as a tool plug-in for the IBM Support Assistant (ISA).

Thread Analyzer functions

- Analysis section
  - Breakdown of where current threads are executing in JVM
  - Use to determine where investigation into thread dump should start
- Overall Monitor Analysis section
  - Provides analysis of monitors within JVM
  - Use to determine if deadlocks are to blame for problem
    - Shows classic deadlocks and single-threaded waiters that exist during your application's execution
- Overall Thread Analysis section
  - Provides top of stack (TOS) analysis for threads in JVM
  - Use to determine areas of high contention in your application
    - Look for methods that are being executed by majority of threads in JVM
- Remaining sections offer similar information but break it down by container and offer advice in just those areas
  - Advice specifically for Web container and ORB service

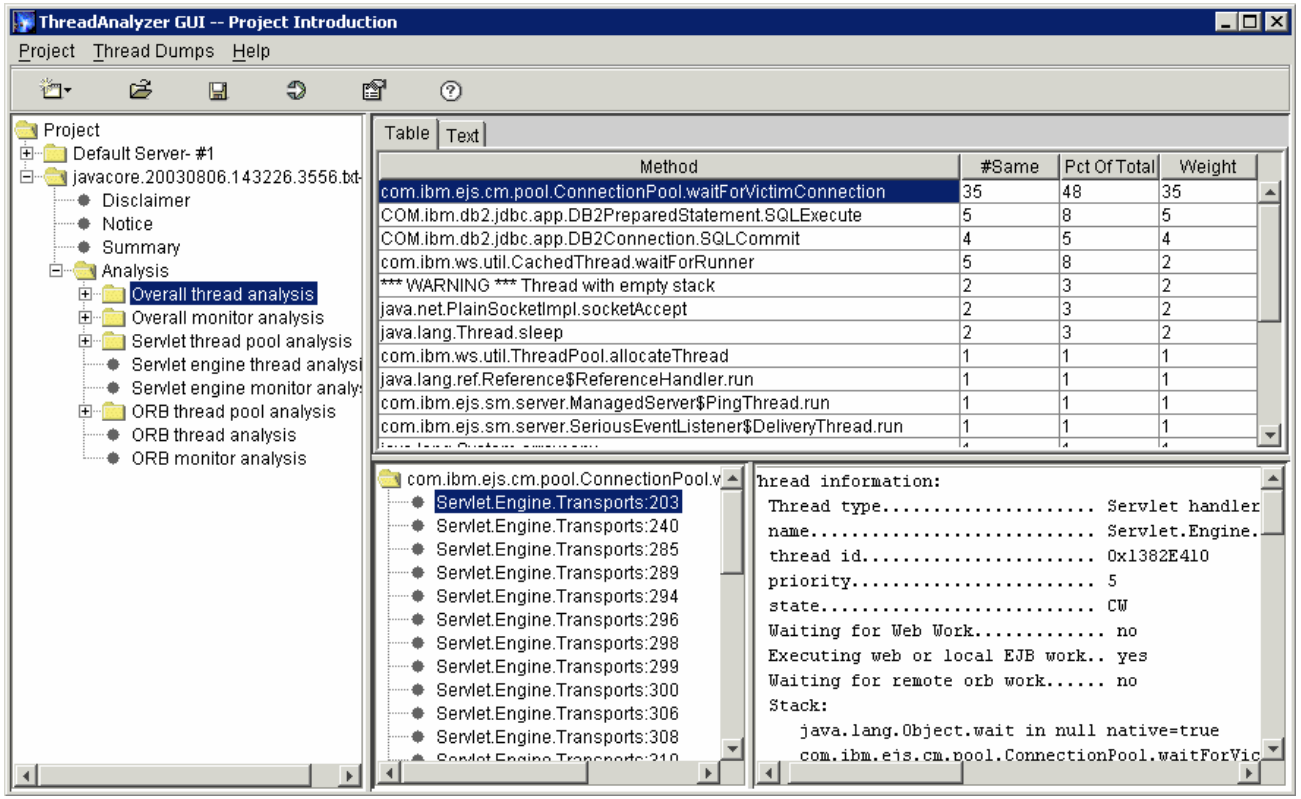
The Thread Analyzer tool contains several sections to aid you in your problem determination. The Summary and Analysis pages provides a textual and graphical breakdown of where current threads are executing in the JVM. Use this page to help guide your investigation. The Overall Monitor Analysis section provides an overview of the existing monitors in the JVM. This information will show classic deadlocks and single-threaded waiters that exist at the time of the thread dump. The Overall Thread Analysis section provides a top of stack analysis for threads in the JVM. Use this information to determine areas of high contention in your application.

Thread Analyzer: summary



The Summary view displays the breakdown of where threads are executing in the JVM. It also provides a quick point of reference to determine if there are any deadlocks present in the thread dump. The Project panel on the left displays the analysis outline for the thread dumps in the current project. This panel is common to all views. From this panel you can select any of the different analytic views for the thread dump.

Thread Analyzer: overall thread analysis

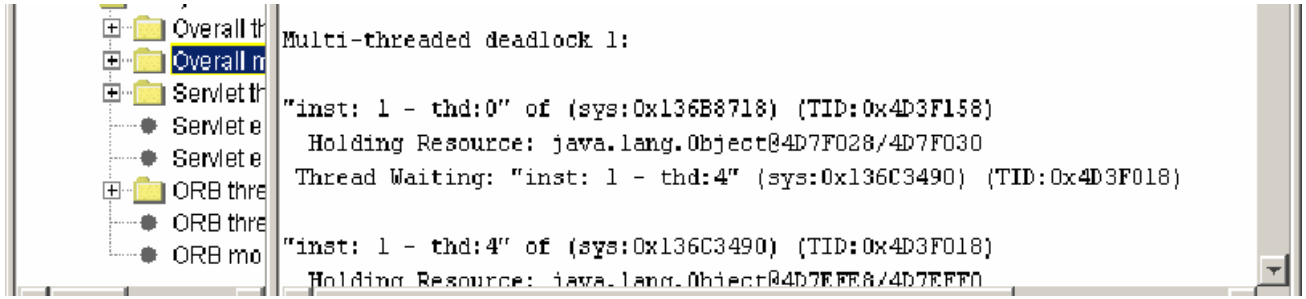


The Overall thread analysis view displays all of the methods that were being executed and the number of threads that were executing them. You can also look at the raw text data as well by flipping to the Text tab. The bottom-center panel displays the details from selecting the **com.bim.ejs.cm.pool.ConnectionPool.waitForVictimConnection** method in the panel above. The panel displays the exact threads that were executing the method at the time of the thread dump.

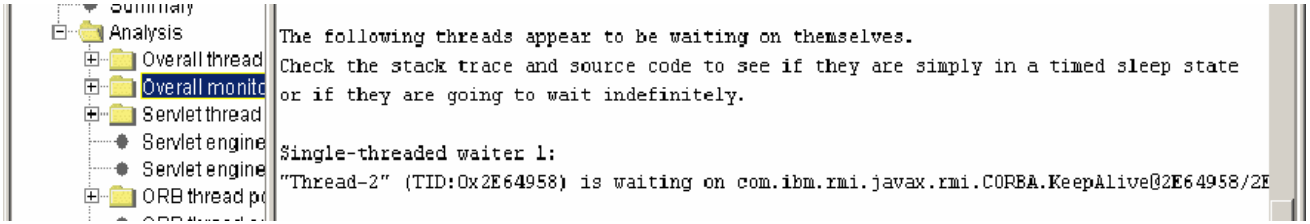
The bottom-right panel displays information for the selected thread. A thread can be in a state of Runnable or Condition Wait. Runnable, or R, means that the thread is able to run when given the chance. Condition Wait, or CW, means that the thread is waiting, perhaps because a sleep() call has been made, or the thread has been blocked for i/o.

Thread Analyzer: overall monitor analysis

- Classic deadlock analysis



- Single-threaded waiter analysis



The Overall monitor analysis section displays details on any deadlocks present in the thread dump. The first part of the analysis has information pertaining to classic deadlocks, where as the second part of the analysis has information pertaining to single-threaded waiters.

Example output: servlet thread analysis

Servlet engine thread analysis

COM.ibm.db2.jdbc.app.DB2PreparedStatement.SQLExecute
seems to be currently executing on 40 servlet threads.
Since 78% (40 out of 51) of the threads doing servlet work seem to be
executing this method, it would seem that there
is some possibility that this method and its call path
may warrant investigation.
Servlets affected:
trade_client.TradeScenarioServlet [40 occurrences]
Callers (servlet threads only):
trade.TradeBean.getBalance [1]
trade.TradeBean.buy [1]
COM.ibm.db2.jdbc.app.DB2PreparedStatement.executeQuery [11]
COM.ibm.db2.jdbc.app.DB2PreparedStatement.executeUpdate [11]
trade.EJSJDBCPersisterCMPAccountBean._create [2]
trade.EJSJDBCPersisterCMPHoldingBean.findByUserID [2]
trade.EJSJDBCPersisterCMPQuoteBean.load [1]
trade.EJSJDBCPersisterCMPProfileBean.load [2]
trade.EJSRemoteStatelessTrade.login [2]
trade.EJSJDBCPersisterCMPRegistryBean.store [2]
trade.EJSJDBCPersisterCMPAccountBean.load [1]

The circled text above is an example of the type of recommendation that you'll receive from Thread Analyzer. The message shows how the tool tries to point you in the initial direction of a problem area. In the above example, the tool has identified an SQLExecute method which is executing on a number of threads, and therefore may warrant further investigation.

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 SW5706G15_Hangs.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:

WebSphere z/OS

Java, JMX, 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.

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.