IBM Software Group

WebSphere Application Server v6 Serviceability Features

e business on demand.

IBM Confidential

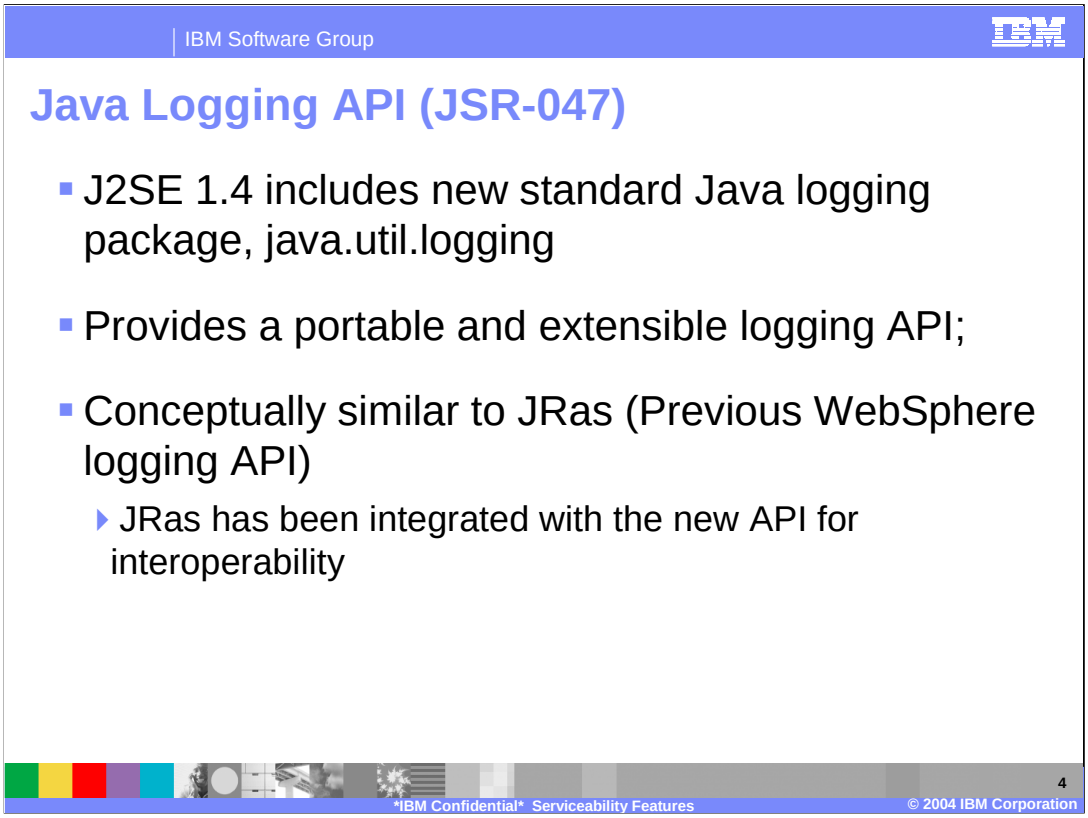
Updated October 18, 2004

Agenda

- Discuss the serviceability changes in WebSphere Application Server v6.0
  - Java Logging API (JSR-47)
  - Configuring Logging and Tracing
  - Generating an IBM Heap Dump
- Briefly cover some of the serviceability features that were added in v5.1.1
  - Hung thread detection
  - Connection leak diagnostics

Section

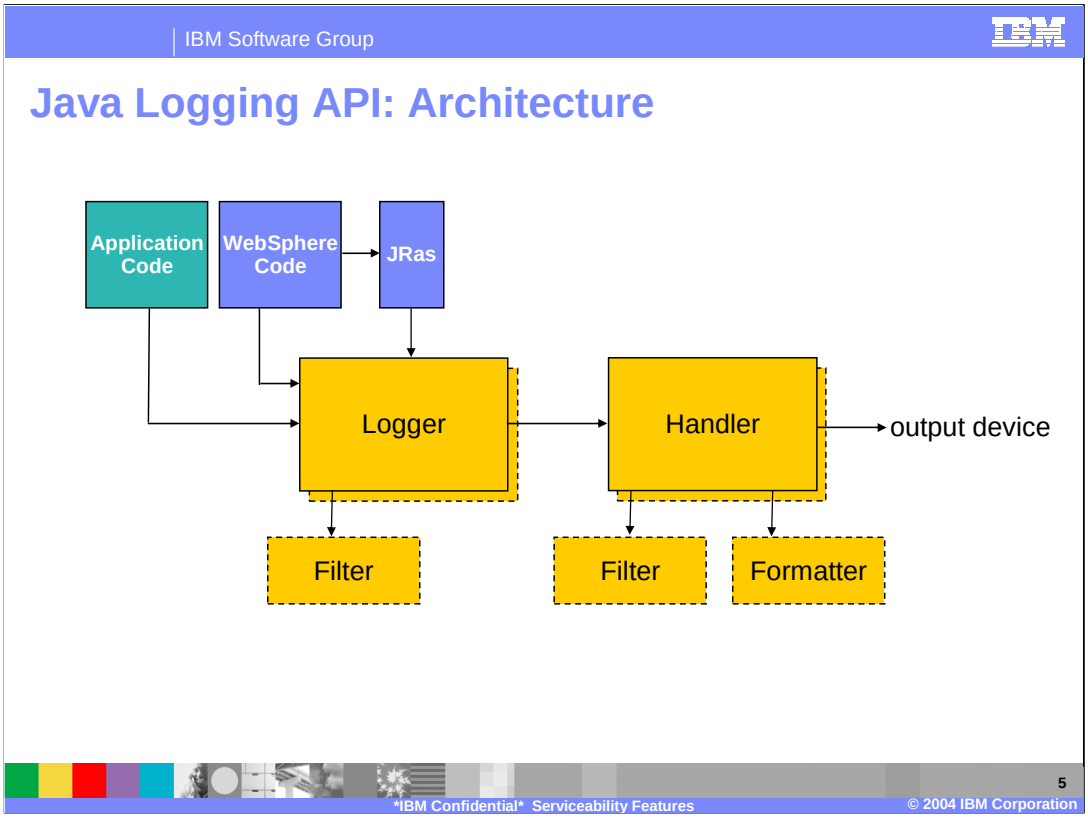
Java Logging API (JSR-047)



J2SE 1.4 includes a new package, called java.util.logging, standardized as JSR-047.

Previous versions of WebSphere exposed an API called JRas. JSR-047 and JRas have similar functionality, but JSR-047 makes application logging portable to other compliant containers. The JRas API is still exposed to applications.

Internally, WebSphere still uses JRas in conjunction with JSR-047. The messages get passed through the standard logger so the messages will be handled consistently.



This diagram shows the main elements of the Java logging architecture, and illustrates the flow of log data.

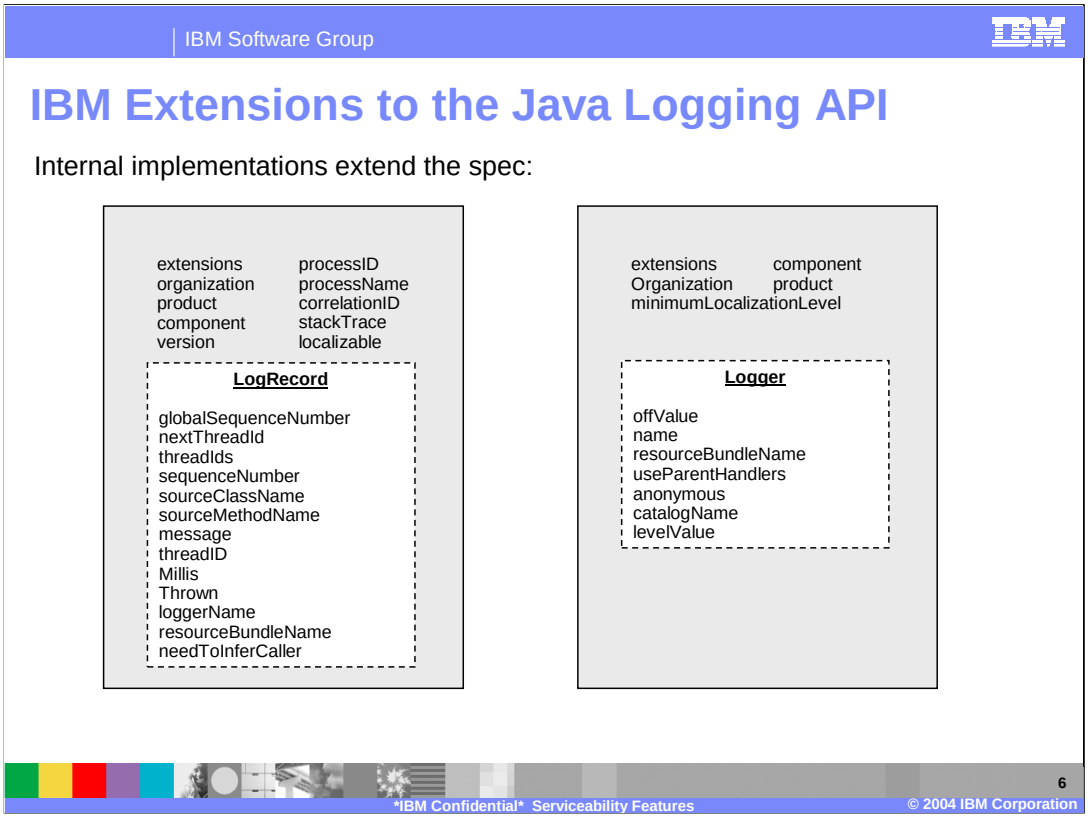
Both application code and WebSphere code make use of Logger objects to put data onto the logstream, in the form of LogRecord objects.

Logger objects can be associated with one or more Handler objects. Handlers represent output devices. For instance, one handler would represent the service log, while another might represent the StandardOut log.

Filters are used to decide which messages get forwarded through the stream and which do not. For instance, A filter would be attached to the StandardOut Handler such that only messages intended for StandardOut could pass through. You could also exclude messages that contained a particular key using a filter.

Formatters are used by Handlers to format log data for output. Localization could be implemented using a Formatter, for example.

JRas, the logging API used in previous versions of WebSphere, is integrated with the Java logging API, so that Loggers and Handlers can receive and process all messages, regardless of whether they were logged using JRas or Java logging. It is still possible for Application code to utilize JRas as well, although that functionality is deprecated in version 6.



WebSphere has embraced the standard Java logging API that it is a part of the J2SE 1.4 specification.

Internally, WebSphere components use an extended version of the API that logs more data, such as process IDs, component IDs, and version numbers. They have been extended specifically for use by the WebSphere runtime and are not available to other code as part of the SPI.

The image above shows how the two main logging objects, Logger and LogRecord, have been extended for use by internal components.

Section

Configuring Logging and Tracing

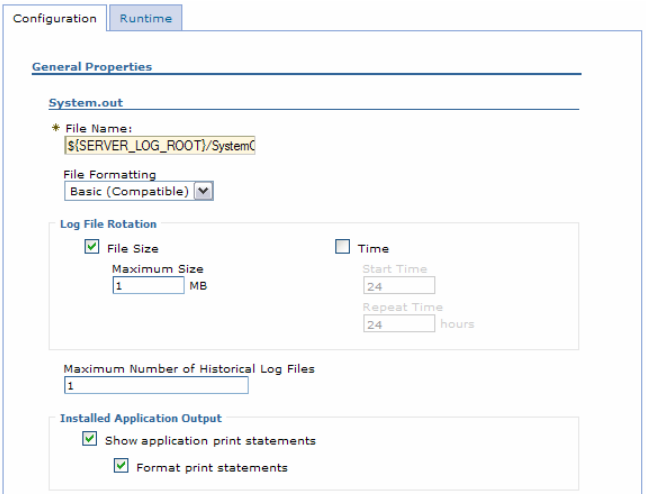
Logging and Tracing Changes

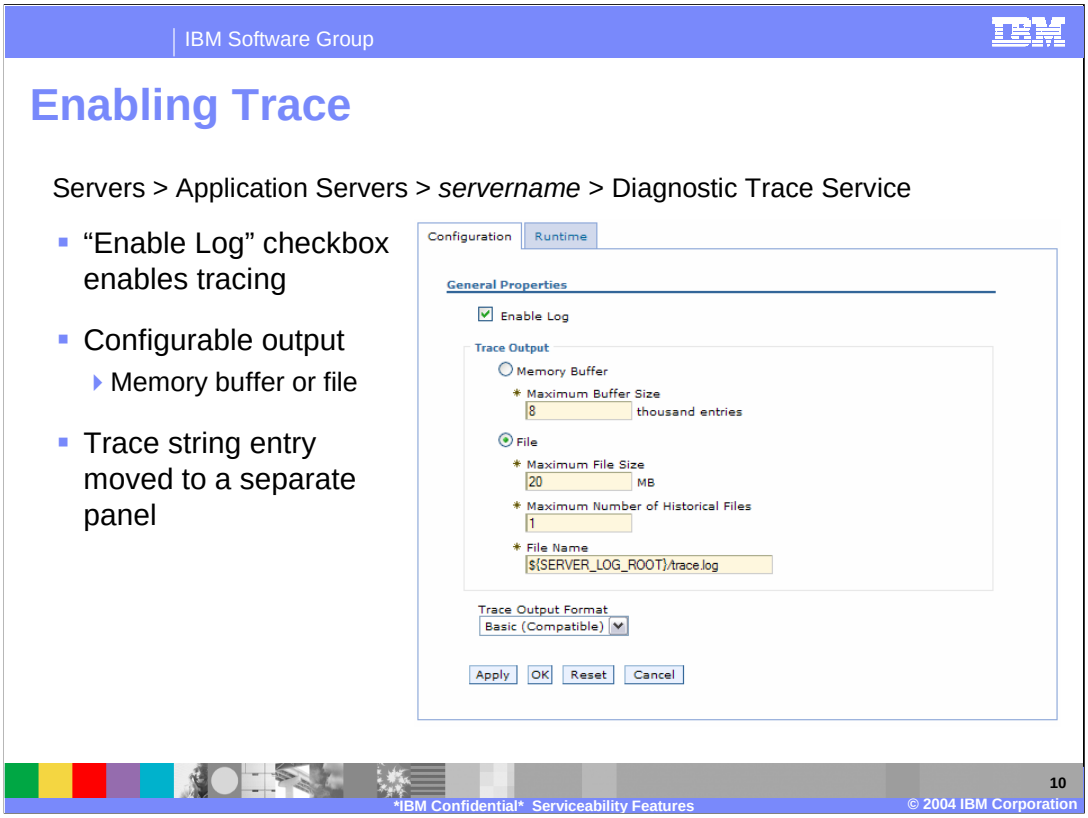
- Logging and tracing configuration panels are significantly changed
  - Related to the infrastructure changes made to implement Java logging
- Logging Detail Level (formerly trace level) is now set on a separate panel
  - Stands alone from the Diagnostic Tracing panel because it affects both logging and tracing
  - Logging and tracing are the same from an infrastructure perspective
    - Different log and trace files are just different Java logging Handlers.

Configuring JVM Logs

From Servers > Application Servers > *servername*:

- Logging and Tracing > JVM Logs
- System.out and System.err logs configured from here
- Logs are self-managing
  - Can roll over based on time or file size
  - Number of historical log files is configurable





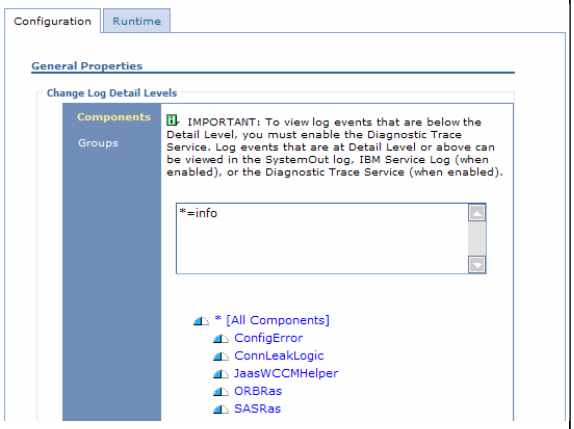
The Diagnostic Trace Service box looks mostly the same as it did in previous versions. The Configuration and Runtime tabs behave as they always have, with Configuration affecting the configuration repository and taking effect at the next startup, while Runtime takes effect immediately but is only optionally persisted to the server configuration.

You also still have the option to configure tracing to either a memory buffer or the filesystem.

The major change on this panel is the absence of a space to enter the trace string. Trace strings have been moved to a separate panel.

Setting the Log Detail Level

- *servername* > Logging and Tracing > Change Log Detail Level
- Log detail level affects tracing **and** regular logging
  - Setting levels below info reduces the amount of data in logs
  - *=off disables logging altogether
- Trace levels (fine, finer, finest) will not appear in trace file unless trace log is enabled
- Log string can be typed in or set using the graphical menu
- Default is *=info



Log Detail Levels

v6 Log Level	v5 Log Level	v5 Trace Level	Description
Off	Off		Turn off logging and tracing
Fatal	Fatal		Task cannot continue and component cannot function.
Severe	Error		Task cannot continue but component can still function.
Warning	Warning		Potential error or impending error.
Audit	Audit		Significant event affecting server state or resources
Info	Info		General information outlining overall task progress
Config			Configuration change or status
Detail			Info detailing subtask progress
Fine		Event	General trace + method entry / exit / return values
Finer		Entry / Exit	Detailed trace
Finest		Debug	Most detailed trace
All		All=enabled	Log all events

Log String Syntax

- <component / group> = <log level>
- Examples
  - com.ibm.ws.classloader.ClassGraph=finest
    - Enables finest trace level for com.ibm.ws.classloader.ClassGraph
  - EJBContainer=fine
    - Enables least verbose trace level for all components in the EJBContainer group
  - com.ibm.ws.classloader.*=finer
    - Enables detailed trace for all classes in the com.ibm.ws.classloader package
  - *=info
    - Sets the log level for all components to info

Log String Syntax (continued)

- Separate multiple trace strings using colon (:)
  - e.g., *=config:com.ibm.ejs.*=fine
- "*=<level>" sets level for all strings that are not explicitly set
  - Above string sets trace level fine for com.ibm.ejs.*, and log level config for all other components

Log String Parsing

- Log strings are parsed from left to right
  - If entries conflict, rightmost entry takes precedence
- If string does not start with “*=level”, “*=info” is prepended
  - Sets the default level for all strings not covered by the rest of the string
  - “*=info:com.ibm.ejs.*=fine” is equal to “com.ibm.ejs.*=fine”

Compatibility with v5 Trace Strings

- v5-style trace strings are supported
  - Mapped to “most similar” v6 log strings
  - Their values are not immediately obvious
  - Recommendation is to avoid using them
- “<level>=disabled” sets the log level to one step lower (less verbose) than <level>
- “<level>=enabled” sets the log level to <level>

find infocenter link on backwards compatibility.

v5 Trace Strings: Examples

v5 Trace String	Effect on v6	Why
com.ibm.ejs.ras.*=debug=enabled	com.ibm.ejs.ras.*=finest	*debug* is equivalent to *finest*, and *enabled* turns on the log level specified by the string
com.ibm.ejs.ras.*=debug=disabled	com.ibm.ejs.ras.*=finer	*debug* is equivalent to *finest*, and *disabled* sets the level one step lower. The step below *finest* is *finer*.
*=all=enabled	*=all	*enabled* turns on the specified log level.
*=all=disabled	*=info	Turns off all tracing, but leaves logging enabled **This is an exception to the previous rules.**

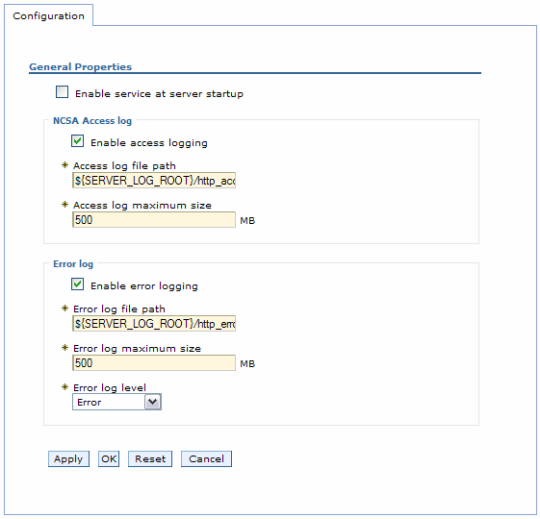
Reminder: It is recommended to use the v6-style log strings, not v5 style shown here

Mixed-version Cells

- v5 and v6 servers interpret trace strings differently
  - v5 servers recognize only v5-style trace strings
  - v6 servers recognize both styles, but the backwards-compatibility logic applies to v5-style strings
- Adapt-a-view functionality
  - Logging and tracing configuration GUI differs significantly between v5 and v6 servers in a mixed-version cell.
    - v5 servers use the v5 GUI, with trace strings specified on the “Diagnostic Trace” panel
    - v6 servers use the v6 GUI, with the log string specified on a separate panel

Embedded HTTP Server Logs

- New Admin. Console panels for configuring embedded HTTP server logs (access & error)
  - Previously could only be configured by setting custom properties
- From main application server panel, click “HTTP Error and NCSA Access Logging”
- Access and error logs can be controlled separately
- When maximum file size is reached, oldest entries are pruned



To enable the access or error log, you must check the “Enable service at server startup” checkbox, and also the checkbox for the specific log you want to enable. Notice there is no runtime tab for these logs. Logging will only begin once you have saved these changes to your configuration and restarted the application server.

Section

Generating an IBM Heap Dump

Generating an IBM Heap Dump

- IBM_HEAPDUMP=true
  - Enables heap dump
  - Heap dump is in .phd (Portable Heap Dump) format
    - .phd is a more compact format used by some of the newer tools
- IBM_JAVA_HEAPDUMP_TEXT=true
  - Formats heap dump as text (classic heap dump format)

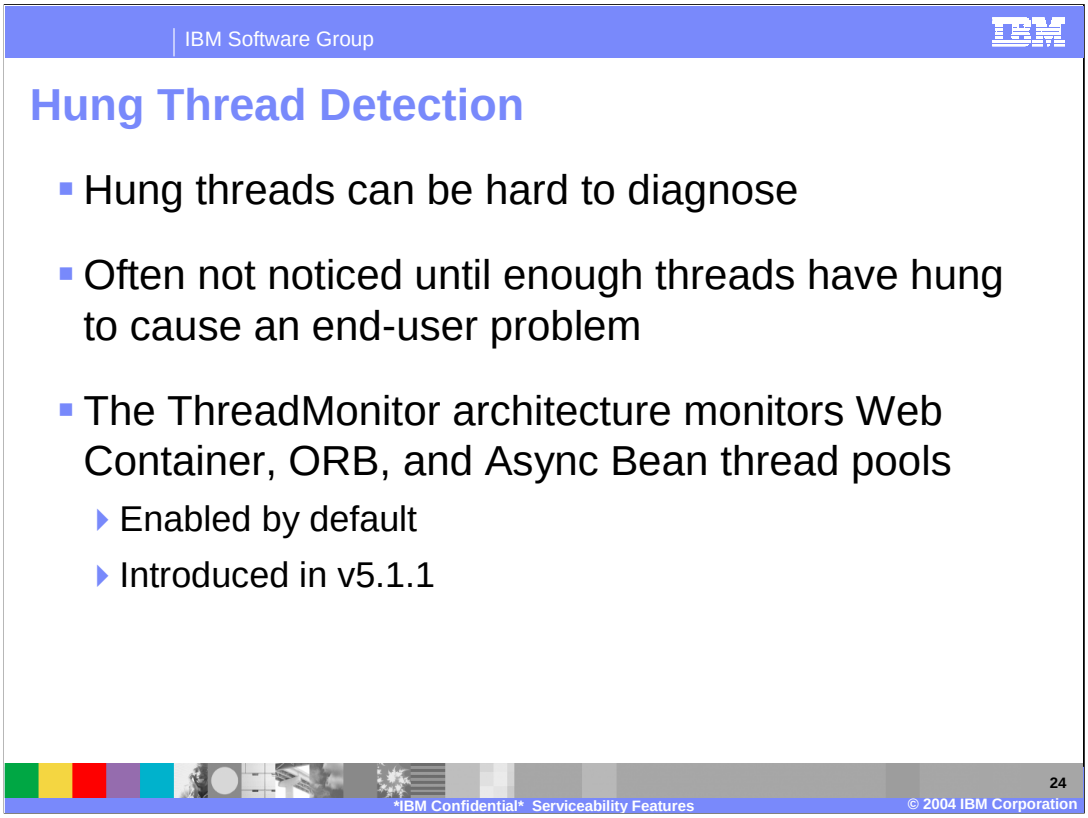
Portable Heap Dump Format

- New versions of memory tools support .phd
- LeakBot
  - Version 0.80a supports .phd
  - http://combatrock.rtp.raleigh.ibm.com/projects/leakbot/
- HeapRoots
  - Version 2.0.5 supports .phd
  - http://www.alphaworks.ibm.com/tech/heaproots
- FindRoots tool set
  - Can convert between .phd and text heap dump formats
  - http://w3.hursley.ibm.com/~dgriff/findroots.html

LeakBot and FindRoots also work with z/OS svcdumps.

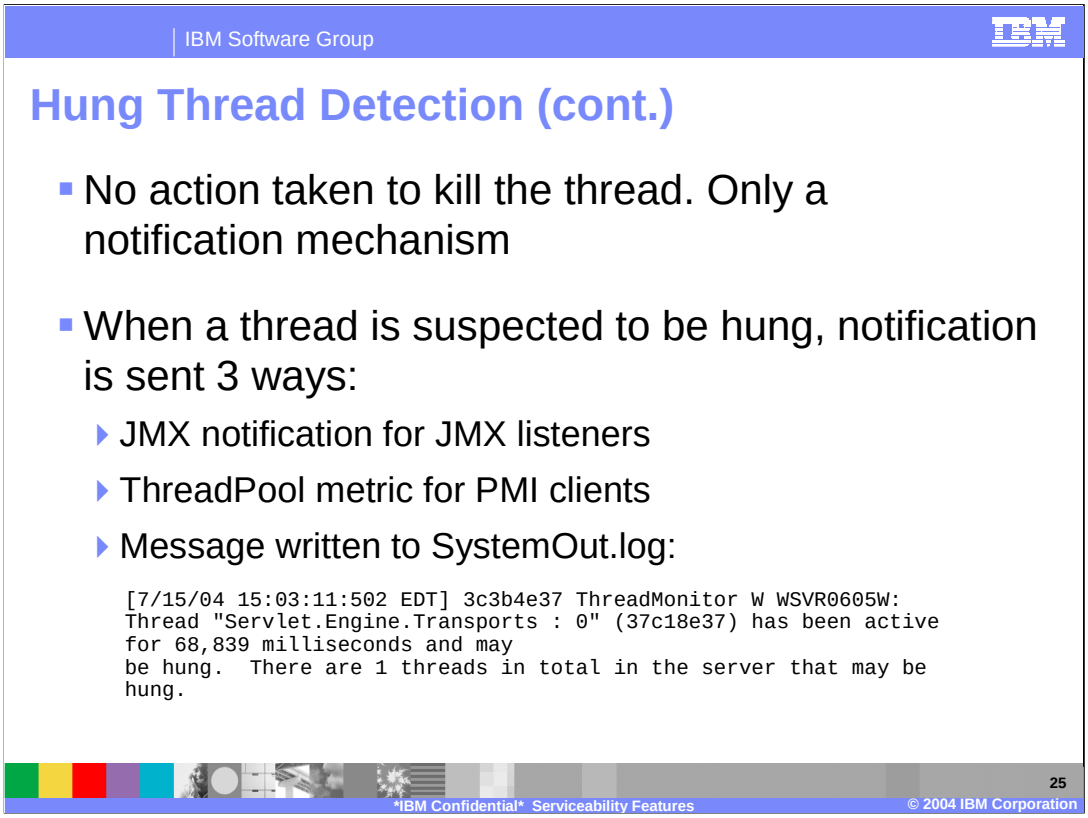
Section

Hung Thread Detection



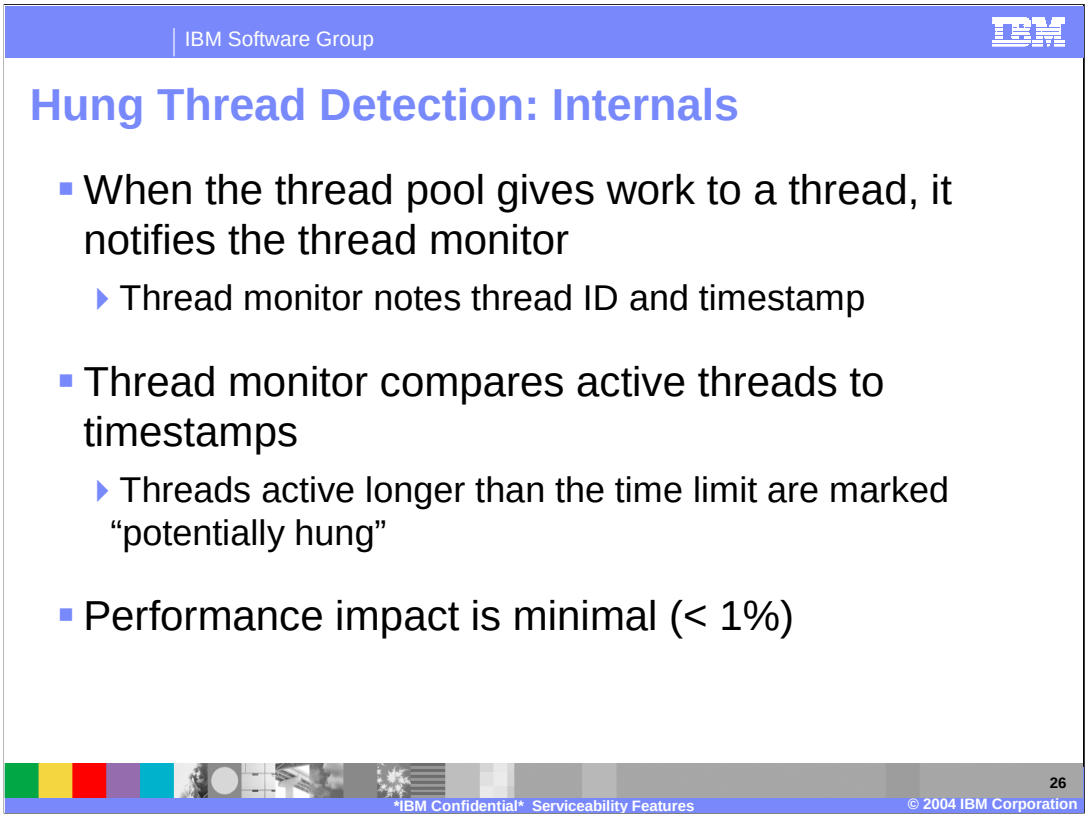
Application threads can hang for a number of reasons, including infinite loops or deadlocks.

As of version 5.1.1, there is a component known as the ThreadMonitor that monitors the Web Container, ORB, and Async Bean thread pools for hung threads.



The thread monitor doesn't try to deal with the hung threads, it just issues notifications, so that the administrator and/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.



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 as discussed on the previous slide.

The performance impact of this monitoring is minimal. Less than 1%.

Hung Thread Detection: Internals (cont.)

- What about false alarms?
  - e.g., 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:

```
[7/15/04 15:03:47:684 EDT] 37c18e37 ThreadMonitor W WSVR0606W: Thread
"Servlet.Engine.Transports : 0" (37c18e37) was previously reported to be
hung but has completed.  It was active for approximately 105,021
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

- 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.

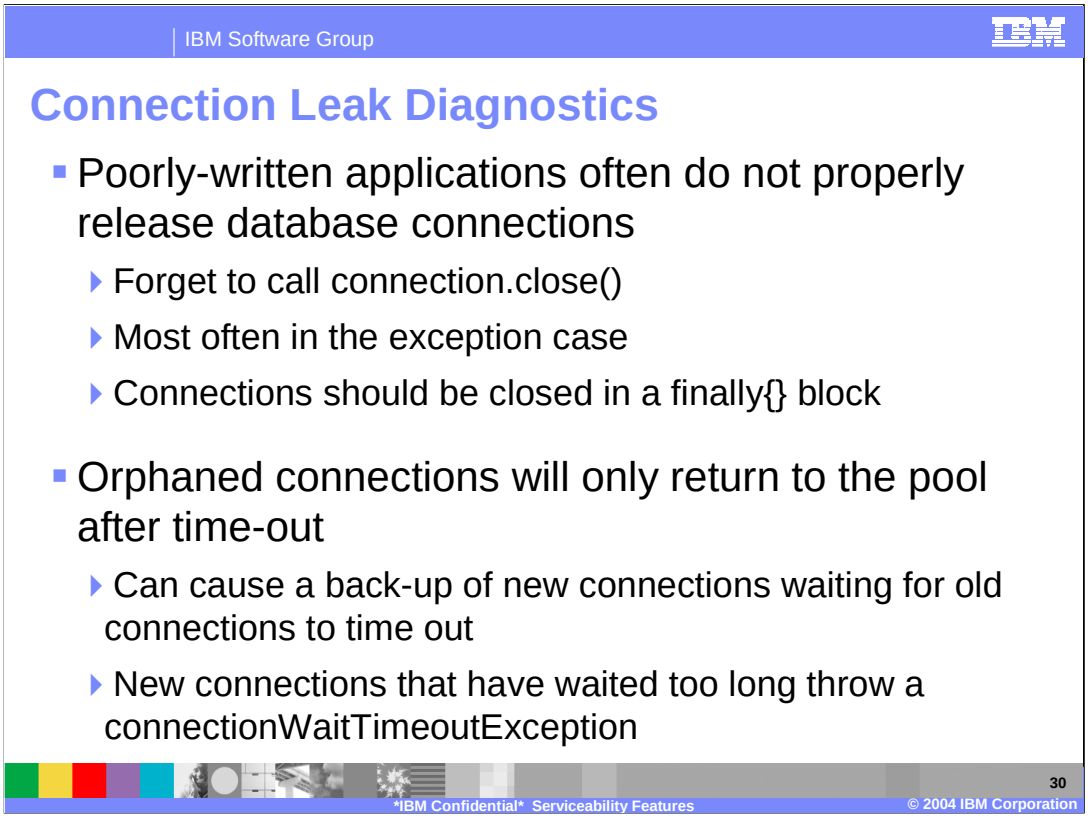
com.ibm.websphere.threadmonitor.threshold is the length of time that a thread can be active before being marked as “potentially hung”. The default value is ten minutes.

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 InfoCenter for instructions.

Section

Connection Leak Diagnostics



Applications can suffer from performance problems and even appear to “hang” if they do not close their connections properly. This is most often caused by developers not properly using the connection.close() method. To ensure that connections will be closed properly, they should be closed in a finally{} block.

WebSphere is smart enough to eventually time-out orphaned connections and return them to the pool, but for an application that makes frequent use of database connections, this might not be enough. New connections can get queued up waiting for the database while old connections are waiting to be timed out. This can bring the application grinding to a halt, and you can see connectionWaitExceptons.

Connection leaks have traditionally been hard to diagnose because the error messages do not usually provide specific enough information about the source of the problem. Usually a source code review is needed to find points in the code where connections are not properly closed.

Connection Leak Diagnostics (cont.)

- Connection manager is instrumented to print stack traces when a connectionWaitTimeoutException occurs

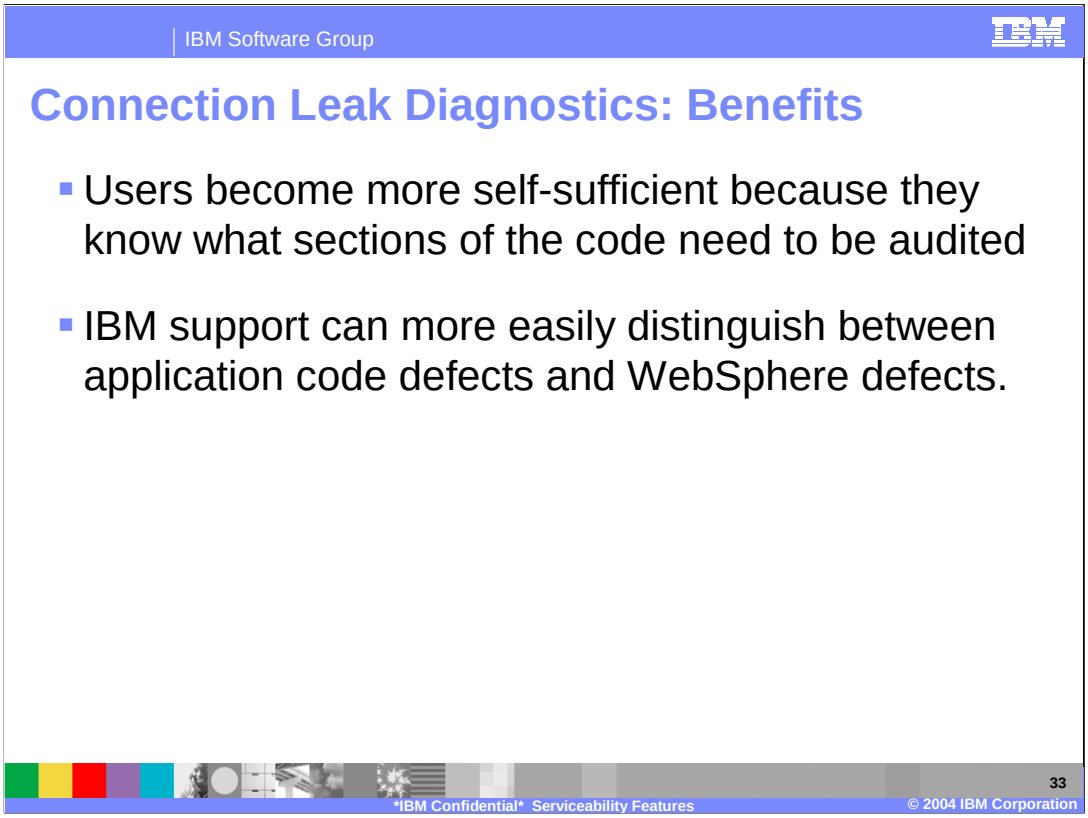
The connection manager has new instrumentation that can hold stack traces for all code that calls getConnection().

Connection Manager Diagnostics

- When activated, enables a connection manager wrapper that holds the stack trace of all getConnection() calls in a Throwable object
- When an exception is thrown due to waiting on a full connection pool, print stack traces of all open connections
- Lower performance impact than connection manager tracing (1-5% impact)

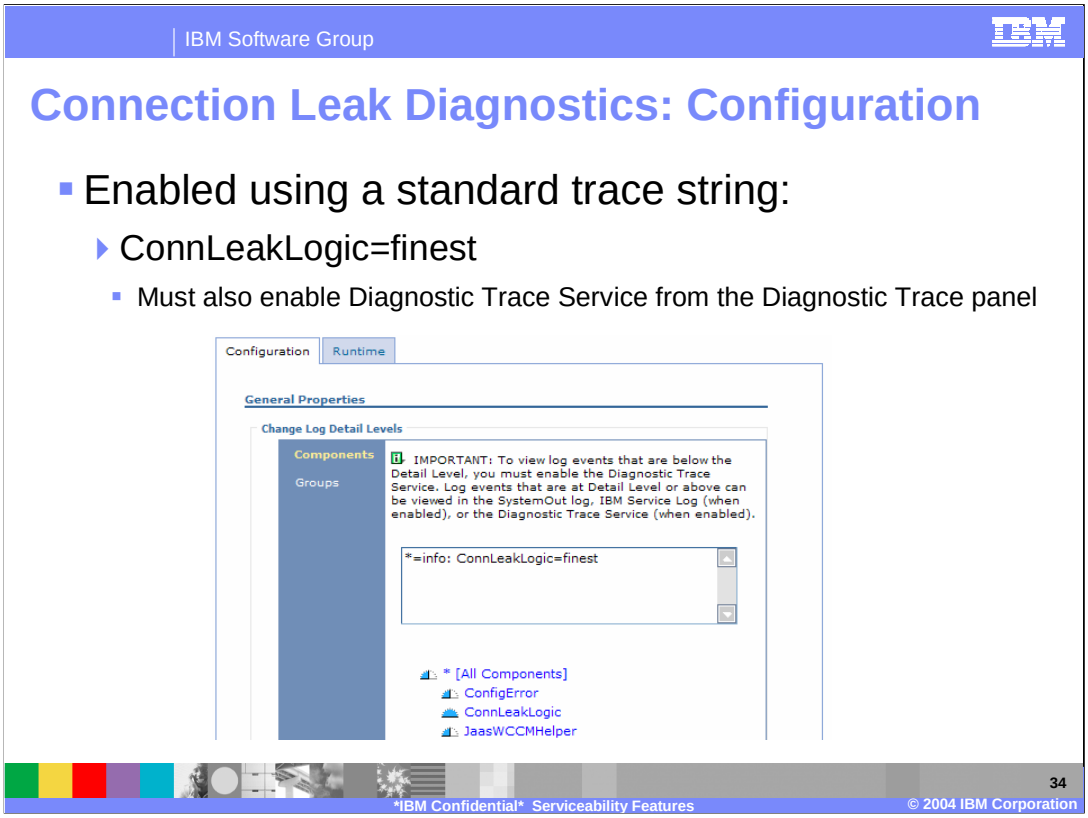
When a thread times out waiting on a connection from a full connection pool, it will throw a connectionWaitTimeoutException.

When this exception is thrown, the wrapper will print out the stack traces for every open connection.



This feature is useful because it shows you the call stacks for all open connections at the time of the exception. This enables you to significantly narrow your search area when you look at the application's source code to try and find the responsible code.

It will also be helpful for IBM support, because it will help distinguish between application defects and WebSphere defects.



To set this trace string using Jython scripting (from the Information Center):

1. Identify the server and assign it to the server variable:

```
server =
   AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1
   /') print server
```

2. Identify the trace service belonging to the server and assign it to the tc variable

```
tc = AdminConfig.list('TraceService', server) print tc
```

3. Set the trace string

```
AdminConfig.modify(tc, [['startupTraceSpecification',
   '*=info:ConnLeakLogic=finest']])
```

Summary

- Logging and tracing has been changed internally to support Java logging (JSR-047)
  - Logging and tracing configuration is different as a result
- Embedded HTTP server logs can be configured from the console
- Serviceability features were added in v5.1.1
  - Hung thread detection
  - Database connection leak diagnostics

Template Revision: 10/01/2004 4:46 PM

Trademarks and Disclaimers

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

IBM	iSeries	OS/400	Informix	WebSphere
IBM(logo)	pSeries	AIX	Cloudscape	MQSeries
e(logo)business	xSeries	CICS	DB2 Universal Database	DB2
Tivoli	zSeries	OS/390	IMS	Lotus

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are
trademarks of Intel Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Other company, product and service names may be trademarks or service marks of others.

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 and/or changes in the product(s) and/or program(s) 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 (e.g., 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.

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.