

**ciwug.org**

Troubleshooting and Debugging

WebSphere  
and

WebSphere Test Environment

Presented By: Brooke Hedrick  
brooke\_hedrick@millamilla.com

# What we will be using...

- Environments
  - WebSphere 5.1.1.8 w/ JDK 1.4.2 SR3
  - RAD 6.0.1.1 - WTE 5.1.1.8 w/ JDK 1.4.2 SR3
- Tools
  - Tivoli Performance Viewer
  - ThreadAnalyzer
  - HeapAnalyzer
  - MBeanInspector
  - QSlice
  - ProcessExplorer
  - JDB ( Java command line debugger )

# Common Debugging and Troubleshooting Problems...

- Leaking database connections
- Hung threads
- Excessive CPU usage
- Memory Leaks - Excessive memory usage
- Different behavior in WebSphere than RAD – Remote Debugging
- Garbage collection frequency

# Leaking Database Connections

This is a common problem where an application allocates a connection from a datasource, but does not return it to the datasource by closing it. We will take a look at how to look for this in your development environment before going to production.

# Leaking Database Connections

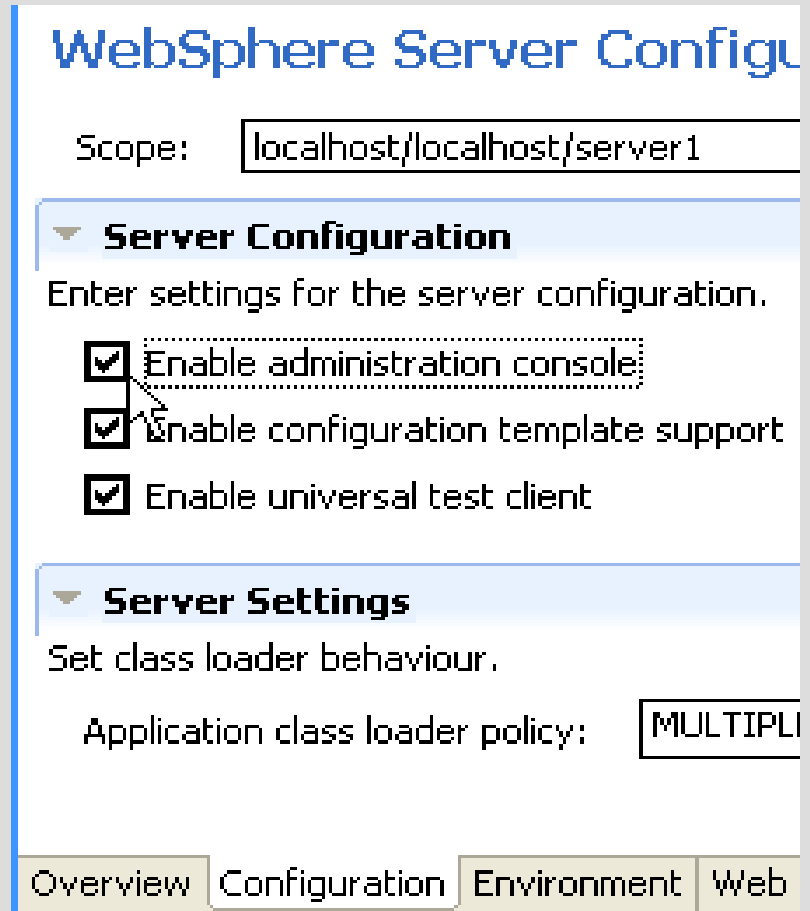
- Tools
  - RAD
  - Tivoli Performance Viewer ( included with WebSphere )

We will need to know the soap port of the application server in RAD. Once we know this, we can start TPV as follows:

```
WebSphere/AppServer/bin/tperviewer.bat <machine> <soap port>
```

# Leaking Database Connections

- Enable the administration console in the RAD WebSphere Test Environment by accessing the Configuration tab of your WTE 5.x server configuration



**WebSphere Server Configuration**

Scope: localhost/localhost/server1

**Server Configuration**

Enter settings for the server configuration.

- Enable administration console
- Enable configuration template support
- Enable universal test client

**Server Settings**

Set class loader behaviour.

Application class loader policy: MULTIPLI

Overview Configuration Environment Web

# Leaking Database Connections

- After starting your sever, access the admin console at <http://localhost:9090/admin>
- Under Server | Application Servers | server1 | Performance Monitoring Service
  - Check the startup box
  - Change connectionPoolModule from N to X
  - Click OK, Save, Save
- Restart the server

Configuration

General Properties

Startup	<input checked="" type="checkbox"/>
Initial specification level	<input type="radio"/> None - All modules below <input type="radio"/> Standard - All modules be <input checked="" type="radio"/> Custom - Modify, add or r

```
beanModule=N  
cacheModule=N  
connectionPoolModule=X  
j2cModule=N  
jvmRuntimeModule=N
```

Apply OK Reset Cancel

# Leaking Database Connections

- Connection in use for entire method...

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
try {
    InitialContext ctx = new javax.naming.InitialContext();
    DataSource ds = (javax.sql.DataSource) ctx.lookup("jdbc/mydatasource");
    Connection con = ds.getConnection();
try {
        // ...
    } finally {
        con.close();
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
} // connection is returned to the pool after here
```

# Leaking Database Connections

- Connection in use in user controlled trans...

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    try {
        InitialContext ctx = new javax.naming.InitialContext();
        DataSource ds = (javax.sql.DataSource) ctx.lookup("jdbc/mydatasource");
        javax.transaction.UserTransaction tx = (UserTransaction)
ctx.lookup("java:comp/UserTransaction");
        tx.begin();
        Connection con = ds.getConnection();
    try {
        // ...
    } finally {
        con.close();
        tx.commit(); // connection is returned to the pool after here
    }
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

# Leaking Database Connections

- Tips
  - Set your max connections to one, if possible, while developing. It will be very obvious if you need more than one connection. You will receive a timeout error when the second connection is requested.
  - If you use unshareable connections, and allocate/deallocate a connection more than once in a single method, without your own transaction block, you will use more than one connection at a time during the life of the method.

# Hung Threads

- A common error in J2EE applications is a hung thread. A hung thread can result from a simple software defect (such as an infinite loop) or a more complex cause (for example, a resource deadlock). A system can become unresponsive even though all resources are idle, as in a deadlock scenario.

# Hung Threads

- WebSphere is able to detect a hung thread based on criteria related to how long a thread can run before it is considered hung.
- It is able to autonomically adjust the time used for determining a thread is hung by settings which account for false positive hung thread notifications after a certain count.

# Hung Threads

## Configuring Hang Detection Policy

- From the administrative console, click Servers > Application Servers > server\_name > Custom Properties... Click New.
- Add the following properties:
  - **Name:** com.ibm.websphere.threadmonitor.interval
  - **Value:** The frequency (in seconds) at which managed threads in the selected application server will be interrogated.
  - **Default:** 180 seconds (three minutes).
  
  - **Name:** com.ibm.websphere.threadmonitor.threshold
  - **Value:** The length of time (in seconds) in which a thread can be active before it is considered hung. Any thread that is detected as active for longer than this length of time is reported as hung.
  - **Default:** The default value is 600 seconds (ten minutes).

# Hung Threads

## Configuring Hang Detection Policy

- Add the following properties (continued):
  - **Name:** `com.ibm.websphere.threadmonitor.false.alarm.threshold`
  - **Value:** The number of times (T) that false alarms can occur before automatically increasing the threshold. It is possible that a thread that is reported as hung eventually completes its work, resulting in a false alarm. A large number of these events indicates that the threshold value is too small. The hang detection facility can automatically respond to this situation: For every T false alarms, the threshold T is increased by a factor of 1.5. Set the value to zero (or less) to disable the automatic adjustment.
  - **Default:** 100

# Hung Threads

- These will show up in the SystemOut.log for the application server ( console in RAD )
- Format will resemble:  

```
[2/18/06 16:45:57:734 CST] 11d34bdd ThreadMonitor W  
WSVR0605W: Thread "Servlet.Engine.Transports : 2"  
(7f564bde) has been active for 643,515 milliseconds  
and may be hung. There are 1 threads in total in the  
server that may be hung.
```
- To find out what that thread is doing, you can use WSADMIN scripting, MBeanInspector, or create a simple servlet or jsp with a call to `com.ibm.jvm.Dump.JavaDump()` ;

# Hung Threads

- Take the generated javacore file and look for a reference to the offending thread as indicated in the SystemOut.log.
- For example: `"Servlet.Engine.Transports : 2"`
- This will allow you to see what code is running that appears to be hung.
- You will find the servlet thread, unless the suspected hung thread has already cleared.

```
[2/18/06 17:01:02:359 CST] 7f564bde  
ThreadMonitor W WSVR0606W: Thread  
"Servlet.Engine.Transports : 2" (7f564bde)  
was previously reported to be hung but has  
completed...
```

# High CPU Usage

- High CPU Usage is typically caused by a hung thread. System resources, such as CPU time, might be consumed by this hung transaction when threads run unbounded code paths, such as when the code is running in an infinite loop.

# High CPU Usage

- Since High CPU Usage is typically caused by hung threads, we can use nearly the same approach to determine where the issue lies.
- First, we need to identify the thread id(s), in the jvm, that are creating high CPU usage. We can use QSlice for this.
- Next, we will need to force a javacore dump in the offending JVM.
- Finally, we will use the thread id(s) we found and look in the javacore files to determine what was executing in those threads.

# Memory Leaks

- Memory leaks occur in Java™ applications when object references are unintentionally held on to when these references are no longer needed. This problem prevents the Java garbage collection process from freeing memory, even though the Java language has a built-in garbage collection mechanism which frees the programmer from any explicit object deallocation responsibilities. Memory leaks are hard to diagnose in large, complex Java applications because of the large number of objects in the Java heap and because of the complex relationships between these objects.

# Memory Leaks

- When the JVM is unable to allocate the amount of memory requested by an application, typically, you will receive a `java.lang.OutOfMemoryError` and the JVM will produce a heapdump.
- It is also possible to force the creation of a heapdump in the event you are trying to troubleshoot an existing high memory usage JVM before running into the `java.lang.OutOfMemoryError`

# Memory Leaks

- In a web application, you can easily force a heapdump by inserting:

```
com.ibm.jvm.Dump.HeapDump();
```

in a .jsp file that you would access via an application running in the JVM with the suspected memory leak.

# Memory Leaks

- We can analyze the heapdump files by using tools like Memory Dump Diagnostic For Java or Heap Analyzer.
- Both tools have the ability to help point out where there may be a memory leak, but your knowledge of the application is still necessary.
- MMD4J has a cool feature where you can compare two heapdumps

# Remote Debugging

- There are times when an application is not performing as expected on a WebSphere server, but works fine on the developer machine... Imagine that!
- One option for troubleshooting this is remote debugging. This will allow you to perform many of the same types of debugging activities within a live WebSphere Application Server that can also be performed within RAD, albeit a bit more old school.


# Remote Debugging

- First, we will need to make sure our application server is set up to support remote debugging.
- Find Servers > Application Servers > server\_name > Debugging Service
- Check the Startup box
- Click OK, Save, Save and then restart the application server

# Remote Debugging








[Application Servers](#) > [server1](#) >

## Debugging Service

A model of the attributes needed for debugging a JVM and various components, such as the BSF Manager 

### Configuration

#### General Properties

Startup	<input checked="" type="checkbox"/>	 Specifies whether the server will attempt to start the specified service when the server starts.
JVM debug port	* <input type="text" value="7777"/>	 The port that the JVM will listen on for debug connections.
JVM debug arguments	* <input type="text" value="-Djava.compiler=NONE -Xdebug -Xnc"/>	 The debug argument string used to tell the JVM to start in debug mode.
Debug class filters	<div style="border: 1px solid gray; padding: 2px;"><input type="text" value="com.ibm.servlet.*"/> <input type="text" value="com.ibm.ws.*"/> <input type="text" value="com.ibm.som.*"/> <input type="text" value="com.ibm.CORBA.*"/> <input type="text" value="com.ibm.debug.*"/></div> <input type="button" value="Add&gt;"/>	 This is an array of classes to filter out during debugging. When running in step by step mode the debugger will not stop in classes that match a filter entry.
BSF debug port	* <input type="text" value="4444"/>	 The port to start the BSF debug manager listening on.
BSF logging level	<input type="text" value="0-No logging"/> 	 The level of logging the BSF Debug Manager will provide.

Apply

OK

Reset

Cancel

# Remote Debugging

- To connect to a remote debugging session:  
`jdb -attach <host>:port`

example: `jdb -attach localhost:7777`

- Common commands you will use once attached:

`help, print/eval, dump, set, locals, list, use/sourcepath, step, next, cont, stop in, stop at, exit/quit`

# Remote Debugging

- If you don't have the source to your classes on hand and you would like to be able to debug quickly, you can use a tool like CavaJ to decompile the code from the .class file into a .java file.
- This will let you guesstimate line numbers and allows you to make a code change, recompile, and run with your new version.

# References

- Tools

- MBeanInspector:

<http://www.alphaworks.ibm.com/tech/mbeaninspector>

- Memory Dump Diagnostic for Java – MDD4J

[http://www.ibm.com/developerworks/websphere/downloads/memory\\_dump.html](http://www.ibm.com/developerworks/websphere/downloads/memory_dump.html)

- Heap Analyzer 1.3.5

<http://www.alphaworks.ibm.com/tech/heapanalyzer>

# References

- Tools (continued)
  - Qslice.exe from the windows resource kit
    - <http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/qslice-o.asp>
  - ProcessExplorer
    - <http://www.sysinternals.com/Utilities/ProcessExplorer.html>
  - Thread Analyzer (javacore files) – I had to get an update from support for the latest jdk.  
[http://www.ibm.com/developerworks/websphere/downloads/thread\\_analyzer.html](http://www.ibm.com/developerworks/websphere/downloads/thread_analyzer.html)

# References

- Tools (continued)
  - CavaJ – Java decompiler  
<http://www.devfilesland.com/developer/Sureshot/Cavaj-Java-Decompiler.html>

# References

- Information

- Hang Detection Policy:

- [http://publib.boulder.ibm.com/infocenter/wasinfo/v5r1/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb\\_confighangdet.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v5r1/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_confighangdet.html)