AppServer Hang and Crash Debugging for Documentum

- Version 3.0

Archana Arunkumar

Revision History

Docbase Version	Revised Date	Revised By	Revision Description
1.0	09/21/2006	Archana Arunkumar	Initial draft
2.0	10/05/2006	Archana Arunkumar	Incorporating Naveed's suggestions.
3.0	10/17/2006	Archana Arunkumar	Truss and other minor changes.

TABLE OF CONTENTS

1. Purpose of this Guide:

This guide provides step-by-step instructions for diagnosing an Application Server Hang or Crash for a Documentum server. This document will also outlines a list of diagnostic tools, and any other system requirements that can be used in collecting information for debugging an Application Server Hang / Crash issue on various platforms. Please note that this document needs to be reviewed before a hang or crash occurs rather than reading it for the first time while diagnosing a live problem, as there are some base data that need to be collected before a hang or crash occurs.

2. What is an Application Server (AppServer)?

Broadly speaking, an Application Server (AppServer) is a server that provides methods that client applications can call. An application server exposes business logic to client applications through various protocols, possibly including HTTP. While a Web server mainly deals with sending HTML for display in a Web browser, an application server provides access to business logic for use by client application programs. The application program can use this logic just as it would call a method on an object (or a function in the procedural world).

Such application server clients can include GUIs (graphical user interface) running on a PC, a Web server, or even other application servers. The information traveling back and forth between an application server and its client is not restricted to simple display markup. Instead, the information is program logic. Since the logic takes the form of data and method calls and not static HTML, the client can employ the exposed business logic however it wants. Moreover, the application server manages its own resources. Such gate-keeping duties include security, transaction processing, resource pooling, and messaging.

Some of the common leading Application Servers used in the industry are Tomcat Apache Application Server, IBM WebSphere Application Server, and BEA WebLogic Application Server. The debugging instructions provided in this guide will apply to all of these, unless they are specifically addressed to one or the other.

3. What is the difference between a Hang and a Crash?

3.1 To start with, we would like to describe a *hang* or *crash* scenario for AppServer as follows:

An AppServer *hang* refers to a situation where the AppServer process does not seem to respond, but still appears to be running. An active process id is still associated with the AppServer process.

An AppServer *crash* refers to a situation where in the process terminates abruptly. In the case of a crash, the process is no longer running.

3.2 You can get the AppServer Process ID (PID) by doing the following:

3.2.1 Using PID files:

Tomcat Apache Application Server:

You can get the PID from the tomcat.pid file under the directory <TOMCAT_install_root>/bin/tomcat.pid

IBM WebSphere Application Server:

You can get the PID from the server.pid file under the directory <WAS_install_root>/profiles/<profile>/logs/<server>/server.pid

BEA WebLogic Application Server:

You can get the PID from the weblogic.pid file under the directory <WL_log_dir>/logs/weblogic.pid

3.2.2 Using OS commands:

Using the appropriate operating system command and the process id (PID) of the AppServer, you can check to see if the process (AppServer) is still running or not.

In Windows:

You can go to the *Task Manager* and click on *Processes* Tab to see if the application server process is still running (you will find the process name and its associated process-id in the list of processes running). If the process is there on this list, then it is a hang. If it is not, then it is a crash.

In Unix like systems:

You can run the “ps –ef |grep java” command to the list full information on all the processes that are running. Again, you can check to see if the AppServer process is there in this list to determine if it is a hang or a crash.

On Solaris – Use “/usr/ucb/ps awxx |grep java” command to get the process as well as the full java command line that was executed to start that process. This is help identify the correct AppServer process.

3.3 Terminology:

Java Core Dump / Java Thread Dump: These two inter-exchangeable terms are used when talking about the Thread dumps that are generated by sending a QUIT signal to the process. The Java Thread Dumps give the java level info on the process.

Process Memory Dumps: The process memory dump that we collect on the AppServer process and the Content Server process on the other hand gives us the native DMCL stack info.

Call Stacks: The call stacks on the AppServer process and the Content Server process on a unix environment also gives us the native DMCL stack info as part of the process memory dumps.

4. Hang:

In order to diagnose an Application Server hang problem, we need to collect the following data when starting to experience a hang:

4.1 Tomcat Apache, BEA WebLogic and IBM WebSphere Application Servers:

	Collect the following information	On Windows	On Unix
1	Get initial system information	Please refer to section 4.1.1.1	Please refer to section 4.1.2.1
2	Java Thread dump every 2 minutes for 5 times	Depending on how you started the Application Server, you can collect this information many ways. Please refer to section 4.1.1.2	Please refer to section 4.1.2.2
3	Collect Heap dump on the AppServer	Please refer to section 4.1.1.3	Please refer to section 4.1.1.3
4	Get the current call stacks / process memory dumps of the AppServer	Use AdPlus tool to get the memory dump. Remember to attach dmcl40.dll and dmcl40.pdb files. Please refer to section 4.1.1.4	Use pstack to get the current call stacks of the AppServer if it on Unix. Please refer to section 4.1.2.3
5	Get the current call stacks / process memory dumps of each of the Content Server session processes	Use AdPlus tool to get	Use pstack to get the

		the memory dump of the Content Server (CS) process. Please refer to section 4.1.1.5	call stack of each Content Server (CS) session processes if CS is running on Unix. Please refer to section 4.1.2.4
6	After each java core dump, get the output of the following iAPI command to list the sessions: “apply,c,NULL,LIST_SESSIONS,BRIEF_INFO,B,F” “next,c,q0” “dump,c,q0”	Start iAPI on a console, connect to the docbase and issue the commands.	Start iAPI on a console, connect to the docbase and issue the commands.
7	Get final system information	Please refer to section 4.1.1.1	Please refer to section 4.1.2.1
8	If connection pooling is enabled, get the output of the following iAPI command: “dumpconnectpool,c”	Issue this command from the application itself.	Issue this command from the application itself.
9	Get the DMCL traces if possible. You will have to turn on DMCL tracing and wait for the hang to occur again.	Turn on the DMCL tracing before starting the Content Server. Please refer to section 4.1.1.6	Turn on the DMCL tracing before starting the Content Server. Please refer to section 4.1.1.6
10	Get the Content Server log.		
11	Collect all logs for the AppServer	Please refer to section 4.1.1.7	Please refer to section 4.1.1.7
12	Collect relevant machine environment information.	Please refer to section 4.1.1.8	Please refer to section 4.1.1.8
13	Collect the following performance data: CPU usage,	This can be	This can be

	memory usage, context switch, disk I/O, and paging I/O. If possible, start collecting performance data 10 minutes before the hang* until you finish collecting all requested dumps. * - When the hang is anticipated to occur.	done using the Performance Monitor tool on Windows. Please refer to section 4.1.1.9	done using the command sar or memstat on Unix/Linux. Please refer to section 4.1.2.5

4.1.1 Collecting Data on Windows:

4.1.1.1 Collecting TCP/IP information on Windows:

4.1.1.1.1 Initial output of the netstat command:

Take the initial output of netstat command to get information about the TCP/IP sockets. This needs to be done before the hang occurs and again at the beginning of the hang.

netstat –an >netstat_before.out

4.1.1.1.2 Final output of the netstat command:

Take the final output of netstat command to get information about the TCP/IP sockets. This needs to be done after all appserver information has been collected, and again after restarting the appserver.

netstat –an >netstat_after.out

4.1.1.2 Collecting Java Core Dumps on Windows:

4.1.1.2.1 Apache AppServer:

4.1.1.2.1.1 Application Server started from command line:

If the AppServer was started from the command line, then you can issue the **CTRL+BREAK** command to generate a java core dump on your console.

4.1.1.2.1.2 Application Server started as a service:

If the AppServer was started as a service and not from the command line, then issuing the CTRL+BREAK command would not work. Instead, you can get the java core dumps by using other diagnostic tools such as:

- windbg.exe
- StackTrace

Using windbg.exe:

One way to force a dump is to use the windbg debugger. The main advantage of using windbg is that it can attach to a process in a non-invasive manner (i.e. read-only). Normally Windows terminates a process after a crash dump is obtained but with non-invasive attach it is possible to obtain a crash dump and let the process continue. To attach non-invasively requires selecting the "Attach to Process" option and clicking the "Noninvasive" checkbox. Once attached, a crash dump can be obtained using the following command:

.dump /mah crash.dmp

For tomcat you can go to configure tomcat and in that when you click on the java tab there is a text area for java options you can add a line -XX:OnError="windbg.exe %%p

Windbg can be installed from the following site:
http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx

Using StackTrace utility:

Another tool that can be used is the StackTrace utility. StackTrace is a tool that allows you to attach to any java process (AppServer process in this case) and get various information (including thread dumps) from that process.

The licensing requires you to launch the program from the web site (unless you pay for it).

The web site is:
http://www.adaptj.com/root/main/download

To get a thread dump with StackTrace you can turn on the following parameters in StackTrace:

From the Process menu select Thread Dump. Enter the target Process ID or use the "..." button to select one. Use the following options:

- *Thread dump* - returns a thread dump of the target process.
- *Use Ctrl-Break* - Sends QUIT signal to the process. Do not use this option if your process was started with the -Xrs VM option (a Windows service for example) because the process will exit.
- *System properties* - returns System.getProperties() for the target process.
- *Memory usage* - returns Runtime.freeMemory(), Runtime.totalMemory(), and Runtime.maxMemory().
- *Process command line* - returns the command line used to start the target process.
- *Create console* - creates a console for a process if it does not have one.
- *Enable stack trace log* (you will need DebugView for this).

The result of the selected operations will appear inside the StackTrace editor window. When you enable the stacktrace log, you will need to have **DebugView** to view the actual log. You can get this at http://www.sysinternals.com/Utilities/DebugView.html

4.1.1.2.2 WebLogic AppServer:

4.1.1.2.1.1 Application Server started from command line:

If the AppServer was started from the command line, then you can issue the **CTRL+BREAK** command to generate a java thread dump on your console.

You can issue the following command to display a thread dump for a specified server.

threadDump ([writeToFile], [filename], [serverName])

For example, the following displays the thread dump for the server managedServer. The information is saved to a file.

wls:/mydomain/serverconfig> threadDump (writeToFile='ture', filename='C:\temp\logfile.txt', serverName='managedServer)

4.1.1.2.1.2 Application Server started as a service:

If you AppServer was started as a service, you will not be able to see the messages form the WebLogic Server that are printed to standard out or standard error. To view these messages, you must direct standard out and standard error to a file. To do this, take the following steps:

- Create a backup copy of the *WL_HOME\server\bin\installSvc.cmd* master script.
- In a text editor, open the *WL_HOME\server\bin\installSvc.cmd* master script.
- In installSvc.cmd, the last command in the script invokes the beasvc utility.
- At the end of the *beasvc* command, append the command *–log:"pathname"* where pathname is a fully qualified path and filename of the file that you want to store the server's standard out and standard error messages.
- If you started WebLogic with nohup, the log messages will show up in *nohup.out.*

4.1.1.2.3 WebSphere AppServer:

From the command prompt, enter the command **wsadmin.bat** to get a wsadmin command prompt. If security is enabled or SOAP ports have been changed, then pass these additional parameters to the batch file to get the wsadmin prompt.

wsadmin.bat [-host host_name] [-port port_number] [-user userid[-password password]

Get a handle to the problem AppServer. It must be entered to se the jvm object.

wsadmin> set jvm [$AdminControl completeObjectName type=JVM,process=server1,*]

(Note there is a space between *completeObjectName* and *type*). The *server1* above is the name of the Application Server that is not responding (hung). If wsadmin is connected to a dmgr and if the server names in cell is not unique, then you can qualify the JVM with node attribute in addition to process.

Now generate the thread dumps by issuing the following command every 2 minutes for 5 times. This will generate the javacore files in the installation root directory.

wsadmin>$AdminControl invoke $jvm dumpThreads

If you notice that javacores were not generated for Kill -3 then, please follow instructions at the following link to get a user.dmp file:
http://www-1.ibm.com/support/docview.wss?rs=180&context=SSCMPB9&q1=MustGatherDocument&uid=swg21145345&loc=en_US&cs=utf-8&lang=en

If you want to analyze the Java thread dumps, you can download the **ThreadAnalyzer** tool from:
http://www-128.ibm.com/developerworks/websphere/downloads/thread_analyzer.html

4.1.1.3 Collecting Heap Dump of AppServer:

4.1.1.3.1 Apache AppServer:

HPROF is a simple profiler agent shipped with the Java 2 SDK. It is a command line profiling tool for heap and CPU usage profiling. HPROF is capable of presenting CPU usage, heap allocation statistics, and monitor contention profiles. In addition, it can also report complete heap dumps and states of all the monitors and threads.

You can run the Tomcat application with hprof as follows:

On Windows/Solaris/Linux:
java –hprof:heap=all
–classpath $CATALINA_HOME/bin/bootsrap.jar
–Dcataline.home=$CATALINE_HOME
–Dcataline.base=$CATALINA_HOPE
org.apache.cataline.startup.Boostrap start

Then, to create the heap dumps, send a SIGQUIT signal to the AppServer process. You can do this by typing CTRL+BREAK.

For more details on the different options with which to run HPROF, please refer to:
http://java.sun.com/developer/technicalArticles/Programming/HPROF.html

4.1.1.3.2 WebLogic AppServer:

You can use the HPROF profiling tool to generate heap dumps.
To run HPROF, use the –Xrunhprof command at startup; for example:

On Windows/Solaris:
java –Xrunhprof:heap=all *ToBeProfiledClass*

where *ToBeProfiledClass* will be the WebLogic's classpath.

For more details on the different options with which to run HPROF, please refer to: http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi/jvmpi.html#hprof

4.1.1.3.3 WebSphere AppServer:

There are several ways to trigger a heap dump in the IBM JVM. The first one is sending the *SIGINT* signal to the Application Server JVM process. Typing *CTRL+BREAK* is the common way to send this signal if the process was started from a command line. The second way to trigger a Heapdump is inserting a *HeapDump()* method within the application code.

To explicity generate a Heapdump, aciviate the HeapDump utility before starting the AppServer by setting the environment variable *IBM_HEAPDUMP* to true for the AppServer process.

If *IBM_HEAPDUMP=true* is set in the WebSphere Administrative Console by selecting **Server=>Application Servers=>server_name=>Process Definition=>Environment Entries**, then Heapdump will be enabled in the typical AppServer environment.

4.1.1.3.1.1 Windows/AIX/Linux:

To generate heap dumps manually:

1. Start the wsadmin scripting client.
2. Invoke the *generateHeapDump* operation on a JVM MBean, for example,
   - Finding JVM ojbectName:
   - *<wsadmin> set objectName [$AdminControl queryNames WebSphere:type=JVM, process=<servername>, node=<nodename>,*]*
   - Invoking the *generateHeapDump* operation on JVM MBean:
   - *<wsadmin> $AdminControl invoke $objectName generateHeapDump*

To automate the heap dump generation, do the following: (Note that the below steps will not work for Sun JVMs and HP JVMs)

- Click **Servers > Application servers** in the administrative console navigation tree.
- Click *server_name* > **Runtime Performance Advisor Configuration**
- Click the **Runtime** tab
- Select the **Enable automatic heap dump collection** check box.
- Click **OK**.

4.1.1.4 Collecting Memory Dump of AppServer on Windows:

4.1.1.4.1 Using AdPlus:

ADPlus is console-based Microsoft Visual Basic script. It automates the Microsoft CDB debugger to produce memory dumps and log files that contain debug output from one or more processes. Each time that you run ADPlus, debugging information (memory dumps and text files that contain debug information) is put in a new, uniquely named folder (such as C:\Temp\Crash_Mode__Date_01-22-2001__Time_09-41-08AM) on the local file system or on a remote network share.

You can get this tool at:
http://www.microsoft.com/whdc/devtools/debugging/default.mspx

When you use ADPlus in hang mode, you must wait until the process or processes stop responding before you run the script (unlike crash mode, hang mode is not persistent). Additionally, hang mode works inside a Terminal Server session.

You can get the AppServer process PID as mentioned in section 3.2 and issue the following command to get the memory dump.

ADPlus -hang -p <PID>

More information on how to use AdPlus can be found at:
http://support.microsoft.com/kb/286350/

Along with the .dmp files that are created, please attach the server dll and pdb files (dmserver_v4.dll and if available the dmserver_v4.pdb files which are located in $DM_HOME/bin) as well. These files will contain the symbols to read the dump files.

4.1.1.5 Collecting Memory Dump of Content Server on Windows:

4.1.1.5.1 Using AdPlus:

You can get the Content Server Process PID by opening up the Task Manager / Processes tab. Use the PID associated with the Content Server process in the following command to get the memory dump of the Content Server.

ADPlus -hang -p <PID>

Note: If you application is connecting to only one server, no need to dump for all. You can connect to that docbase and using iAPI get the r_process_id attribute value from the server_config object. The r_process_id attribute has the process id of the content server process that you need to get the memory dumps for.

Along with the .dmp files that are created, please attach the client dll and pdb files (dmcl40.dll and if available the dmcl40.pdb files which are located in $DM_HOME/bin) as well. These files will contain the symbols to read the dump files.

4.1.1.6 DMCL Tracing:

You can turn on DMCL tracing by adding the following two lines in your dmcl.ini file under [DMAPI_CONFIGURATION]:

trace_file= C:\temp\dmcl_trace.txt
trace_level=10

4.1.1.7 Collecting AppServer logs on Windows:

4.1.1.7.1 Apache AppServer:

Collect the following:

- All the log files located in *install_root\logs**
- The *server.xml* located in *install_root\conf*
- The *web.xml* file located in *install_root\webapps\<webapp_name>\WEB-INF*

4.1.1.7.2 WebLogic AppServer:

The most important information is a log file with multiple thread dumps from a Managed Server. The log file is especially important for diagnosing cluster freezes and deadlocks.

- Stop the server.
- Remove or back up any log files you currently have. You should create a new log file each time you boot a server.
- Start the server with this command, which turns on verbose garbase collection and redirect both standard error and standard output to a logfile:

  % java – ms64m –mx64m ***–verbose:gc*** *–classpath $CLASSPATH –Dweblogic.domain=mydomain –Dweblogic.Name=clusterServer1 –Djava.security.policy==$WL_HOME/lib/weblogic.policy weblogic.Server >>* ***logfile.txt***

4.1.1.7.3 WebSphere AppServer:

For releases of V6.0:

- The *server.xml* file located in the *install_root\profiles\profile_name\config\cells\cell_namenodes*
- *plugin-cfg.xml* and *http_plugin.log*
- Everything in the *install_root\profiles\profile_name\logs\server_name* directory
- Everything in the *install_root\profiles\profile_name\logs\ffdc* directory

For releases of V5.0 and V5.1:

- The *server.xml* file located in the install_root*config\cells\nodes\node_name\servers\server_name* directory
- plugin-cfg.cml and http_plugin.log
- Everything in the *install_root\logs\server_name* directory
- Everything in the *install_root\logs\ffdc* directory

For all releases:

- All *javacore.date.time.id.txt* files generated. These files should be in the installation root directory (such as, *WebSphere\AppServer*).
- *netstat*.out*
- Include the Application Server *systemErr.log, systemOut.log, native_stderr.log* and *native_stdout.log,* if they are located in a different directory other than *install_root/profiles/profile_name/logs/server_name*

- Web server's *access* and *error* logs.
- If the Web server is remote, collect the appropriate file from the Web Server system including Web Server configuration files and Web Server logs.

4.1.1.8 Collecting Machine environment information:

You will want to collect the following information as well:

- dmcl.ini
- server.ini
- beginning of server log
- OS version
- environment variables
- sessionconfig
- apiconfig
- serverconfig
- docbaseconfig
- servermap
- docbrokermap
- docbasemap
- windows system information

You can get all of the above by running the **information.ebs** script referenced in Support Note 40263:

4.1.1.9 Collecting Performance Data on Windows:

4.1.1.9.1 Using Performance Monitor:

As a supplement to using the Windows Task Manager to get more detailed information on the operating system resources being consumed you can use the Windows Permformance Monitor utility. Performance Monitor (PERFMON.EXE) is a no-cost utility that is shipped with Windows.
Performance Monitor is launched as an application from the Administrative Tools group. It can also be launched from the command line by typing **PERFMON** at the command prompt.

- Start->Run->type "perform"
- Right click on the graph
- Select Add Counters

There are listed dozens of counters that you can add to the performance monitor and they will display information on the resources used by category.
For example, some of the categories that you will nedd to select are:
- Processor
- Memory
- TCP
- Paging File (Virtual Memory)

For the "Performance Object" – choose Memory

- Click on Select Counters from the list:
  --Select %Commited Bytes in Use

Another nice feature within the performance tool is a quick reference to Performance Logs and Alerts. You can select a counter and set a predefined limit for the counter. If this limit is reaches it creates an alert log.
For example, you can set an Alert if the memory usage reaches 80% of the system capacity or if the Virtual Memory reaches 99%, etc…

- Monitor your counter on the resulting graph or you can view the same information on a more readable report format by selecting the View Report icon.

More information on how to user the Performance Monitor tool can be found at:
http://www.microsoft.com/technet/prodtechnol/Windows2000Pro/reskit/part6/proch27.mspx?mfr=true

4.1.2 Collecting Data on Unix:

4.1.2.1 Collecting information on Unix:

4.1.2.1.1 Initial output of the netstat command:

Take the initial output of netstat command to get information about the TCP/IP sockets. This needs to be done before the hang occurs and again at the beginning of the hang.

netstat –an >netstat_before.out

Collect the list of all open files for the AppServer process.

lsof –p [PID_of_hung_JVM] > lsof.out

Take the initial output of vmstat command to get information about the memory usage of the system.

vmstat 5 12 > vmstat_before.out

Collect more information about the process and the files opened by the process by issuing the following commands:

On Solaris:
pfiles [PID_of_hung_JVM] > pfiles.out
pmap [PID_of_hung_JVM] > pmap.out
truss –o truss.out –p [PID_of_hung_JVM]

On AIX 5.2 and above:
procfiles [PID_of_hung_JVM] > procfiles.out
procmap [PID_of_hung_JVM] > procmap.out

truss –o truss.out –p [PID_of_hung_JVM]

On Linux:
strace –o strace.out –f –t [PID_of_hung_JVM]
pmap [PID_of_hung_JVM] > pmap.out

On HP-UX:
tusc –o tusc.out –p [PID_of_hung_JVM

4.1.2.1.2 Final output of the netstat command:

Take the final output of netstat command to get information about the TCP/IP sockets. This needs to be done after all appserver information has been collected, and again after restarting the appserver.

netstat –an >netstat_after.out

Take the final output of vmstat command to get information about the memory usage of the system.

vmstat 5 12 > vmstat_after.out

4.1.2.2 Collecting Java Core Dumps on Unix for Apache or Weblogic AppServer:

4.1.2.2.1 Apache AppServer:

On UNIX platforms you can send a signal to a program by using the kill command. This is the quit signal. For example, on Solaris you can use the following command:

kill -QUIT <PID>

where PID is the process ID of your Java AppServer. Refer to section 3.2 to get the PID of your AppServer.

4.1.2.2.2 WebLogic AppServer:

4.1.2.2.2.1 Solaris/HP/AIX:

On UNIX platforms you can send a signal to a program by using the kill command. This is the quit signal. For example, on Solaris you can use the following command:

kill -QUIT <PID>

where PID is the process ID of your Java AppServer. Refer to section 3.2 to get the PID of your AppServer.

4.1.2.2.2.2 Linux:

The Linux operating system views threads differently than other operating systems. Each thread is seen by the operating system as a process. To take a thread dump on Linux, find the process id from which all the other processes were started. Use the commands:

To obtain the root PID, use:
ps –efH1 | grep 'java **.**
The first PID reported will be the root process, assuming that the ps command has not been priped to another routine.

Use the weblogic.Admin command **THREAD_DUMP**

Another method of getting a thread dump is to user the THREAD_DUMP admin command. This method is independent of the OS on which the server instance is running.

java weblogic.Admin –url ManagedHost:8001 –username weblogic -password weblogic THREAD_DUMP

Note: This command cannot be used if unable to ping the server instance.

4.1.2.2.3 WebSphere AppServer:

Now generate the thread dumps by issuing the following command every 2 minutes for 5 times. This will generate the javacore files in the installation root directory.

/usr/proc/bin/pstack [PID_of_hung_JVM] > pstack_1.out

kill -3 [PID_of_hung_JVM]

Note: Remember to modify the pstack*.out filename while issuing the commands every 5 minutes in order not to overwrite them.

The final output of vmstat command to get information about the memory usage of the system.

vmstat 5 12 > vmstat_after.out

4.1.2.3 Collecting Call Stacks of AppServer on Unix:

On UNIX, you can pstack to get a call stack trace on the AppServer. **pstack** attaches to the active processes named by the PIDs on the command line, and prints out an execution stack trace, including a hint at what the function arguments are. If symbols exist in the binary (usually the case unless you have run strip(1)), then symbolic addresses are printed as well. If the process is part of a thread group, then pstack will print out a stack trace for each of the threads in the group.

On Solaris:
pstack <PID>

On AIX 5.2 and above:
procstack <PID>

On Linux:
lsstack <PID>

On HP-UX:
(none found)

Refer to section 3.2 to get the PID of your AppServer.

4.1.2.4 Collecting Call Stacks of Content Server on Unix:

Use the PID associated with the Content Server process in the following command to get a call stack trace of the Content Server.

On Solaris:
pstack <PID>

On AIX 5.2 and above:
procstack <PID>

On Linux:
lsstack <PID>

On HP-UX:
(none found)

Collect the pstack of all Content Server processes for the particular docbase after each java core dump of the AppServer process. Getting the pstack of all the content processes may be difficult as there will be one

Content Server process for each connection (there may be over 200 content server processes).

To make this process less painful you can make sure that you have the following in place and tested before the hang occurs, so that it can be run quickly:

Create two files called pattern and traces.sh in the same directory, and make sure that these files have execute permissions on them.

Open the pattern file using vi, write the following and save it:

```
documentum
mthdsvr
```

Open the traces.sh file using vi, write the following and save it:

```
#!/bin/sh

while read pid comm rest
do
  pstack $pid
done
```

Then, run the following command as the install owner of the docbase (hopefully all the install owners are the same) in the same directory where the traces.sh file is located.
ps -u `/usr/xpg4/bin/id -u` -o pid -o comm | /usr/xpg4/bin/grep -f pattern | traces.sh > traces.txt

This script essentially looks for the any process with the pattern 'documentum' or 'mthdsvr' – gets its PID and runs the pstack command on those PIDs.

Again, this script should be tested before the hang occurs. This should be run after each java core dump (5 times). Make sure not to overwrite the previous run's traces.txt file (Rename the file each time before running the script again). Also, note that the above script only works on Solaris. If you want to run this script on AIX 5.2 and above or Linux, modify the traces.sh file by replacing "pstack $pid" by "procstack $pid" or "lsstack $pid" respectively.

4.1.2.5 Collecting Performance Data on Unix:

Run the following command to get the screenshot from the AppServer (to get the CPU usage of each thread in AppServer):

On Solaris:
prstat –L

On AIX 5.2 and above/Linux/HP-UX:
top

Also, you can get the Reports per-processor statistics for the specified processor or processors by issuing the following command:

sar –P ALL

The sar command (CPU Utilization Report) writes to standard output the statistics for each individual processor, followed by system-wide statistics.

memstat is also another simple tool to analyze your application's dynamic memory management. In particular it can help you identify memory leaks and multiple memory frees. It lists all the processes, executables, and shared libraries that are using up virtual memory.

vmstat is also another command that generates reports on processes, memory, paging, block IO, traps and cpu activity. The first report produced gives averages since the last repboot. Additional reports give information on a sampling period of length delay. **vmstat [delay [count]]** is the command to give, where *delay* is the delay between updates in seconds and *count* is the number of updates.

vmstat 5 12 > vmstat_out.txt

5. Crash:

In order to diagnose an Application Server crash problem, we need to collect the following data when a crash occurs:

5.1 Tomcat Apache and BEA Weblogic Application Servers:

	Collect the following information	On Windows	On Unix
1.	Get the DMCL traces if possible. You will have to turn on DMCL tracing and wait for the AppServer to crash again.	Turn on the DMCL tracing before starting the Content Server. Please refer to section 4.1.1.5	Turn on the DMCL tracing before starting the Content Server. Please refer to section 4.1.1.5
2.	In the case of an AppServer crash, DMCL error pid files will be created by the DMCL in the working directory.		
3.	If possible, try to collect the process memory dumps for the AppServer before the crash* occurs. * - When the crash is anticipated to occur.	Refer to section 4.1.1.2	Refer to section 4.1.2.2

Note that unlike the hang scenario, in the case of a crash Content Server logs and Session dumps are not very useful. Also, we won't be able to get the connection pool data or the session data because of the crash. We may be able to get the last session information from the DMCL trace however.

5.2 IBM WebSphere Application Servers:

	Collect the following information	On Windows	On Unix
1.	Javacore dumps for the AppServer	Refer to section 5.2.1.1	Refer to section 5.2.2.1
2.	Process dumps for the AppServer	Refer to section 5.2.1.2	Refer to section 5.2.2.2

3.	Process (native) stdout log	Refer to section 5.2.1.3	Refer to section 5.2.1.3
4.	Collect all logs for the AppServer	Refer to section 4.1.1.7	Refer to section 4.1.1.7
5.	Collect relevant machine environment information.	Refer to section 4.1.1.8	Refer to section 4.1.1.8
6.	Get the DMCL traces if possible. You will have to turn on DMCL tracing and wait for the AppServer to crash again.	Turn on the DMCL tracing before starting the Content Server. Please refer to section 4.1.1.5	Turn on the DMCL tracing before starting the Content Server. Please refer to section 4.1.1.5
7.	In the case of an AppServer crash, DMCL error pid files will be created by the DMCL in the working directory.		
8.	If possible, try to collect the thread dumps for the AppServer before the crash* occurs. * - When the crash is anticipated to occur.	Refer to section 4.1.1.2	Refer to section 4.1.2.2

Note that unlike the hang scenario, in the case of a crash Content Server logs and Session dumps are not very useful. Also, we won't be able to get the connection pool data or the session data because of the crash. We may be able to get the last session information from the DMCL trace however.

5.2.1 Collecting Data on Windows for WebSphere AppServer:

5.2.1.1 Collecting Java Core Dumps on Windows for WebSphere AppServer:

By default, a javacore occurs when the JVM terminates unexpectedly. The JVM checks each of the following locations for existence and write-permission and stores the javacore in the first one available. Note that you must have enough free disk space (possibly up to 2.5MB) for the javacore file to be written correctly.

- The location specified by the IBM_JAVACOREDIR environment variable if set.
- *<WAS_install_root>/profiles/<profile>*.
- The location that is specified by the TMPDIR environment variable, if it is set.
- The */tmp* directory or on Windows the location that is specified by the TEMP environment variable, if it is set.
- If the javacore cannot be stored in any of the above, it is put to STDERR.

Environment variables for the server process are defined by the administrator as name/value pairs. You can manage environment variables using the following navigation path in the administrative console: **Servers → Application Server → <server> → Java and Process Management → Environment Entries.**

5.2.1.2 Collecting Process Dumps on Windows for WebSphere AppServer:

Windows has an embedded function for collecting data from processes that crash. The dumps are put into a file called user.dmp (user dumps). To ensure these dumps are enabled and to find the location where they are stored:

- Go to **Start → Run**, and type *drwtsn32*
- Look for the Crash Dump field to find the location of the dump file and make sure that the Create Crash Dump File option is selected. (If it is a Windows XP machine, then set the Crash Dump Type to *NT4 Full Compatible* in the dialog box.
- Click **OK**. Enabling these settings is a one time process.

5.2.1.3 Collecting stdout log for WebSphere AppServer:

Include the Application Server *native_stderr.log* and *native_stdout.log*, which are located in *<WAS_install_root>/profiles/<profile_name>/logs/<server_name>* directory

5.2.2 Collecting Data on Unix for WebSphere AppServer:

5.2.2.1 Collecting Java Core Dumps on Unix for WebSphere AppServer:

The javacore files will be written to:

- The location specified by the IBM_JAVACOREDIR environment variable if set.
- *<WAS_install_root>/profiles/<profile>*.
- The location that is specified by the TMPDIR environment variable, if it is set.
- The */tmp* directory or on Windows the location that is specified by the TEMP environment variable, if it is set.
- If the javacore cannot be stored in any of the above, it is put to STDERR.

Before executing any of the following instructions, make a backup copy of the core file. Perform the following for each core files.
Enter the following commands:

On Solaris:
/usr/proc/bin/pstack [core] >pstack.out
/usr/proc/bin/pmap [core] >pmap.out
/usr/proc/bin/pldd [core] >pldd.out

On AIX:
/usr/proc/bin/procstack [core] >procstack.out
/usr/proc/bin/procmap [core] >procmap.out
/usr/proc/bin/procldd [core] >procldd.out

On Linux:
/usr/proc/bin/pstack [core] >pstack.out
/usr/proc/bin/pmap [core] >pmap.out
/usr/proc/bin/pldd [core] >pldd.out

Where *[core]* is the name of the core file and **.out* are changed for each core file.

5.2.2.2 Collecting Process Dumps on AIX/Linux for WebSphere AppServer:

Process dumps are put into a file that is called core and are called core files. These core files are located in the *<WAS_install_root>/bin* directory or the *<operating_system_root>/tmp* directory.

5.2.2.3 Collecting stdout log for WebSphere AppServer:

Include the Application Server *native_stderr.log* and *native_stdout.log*, which are located in *<WAS_install_root>/profiles/<profile_name>/logs/<server_name>* directory

6. Hang – Quick Reference:

	Collect the following information
1	Get initial system information
2	Java Thread dump every 2 minutes for 5 times
3	Collect Heap dump on the AppServer
4	Get the current call stacks / process memory dumps of the AppServer
5	Get the current call stacks / process memory dumps of each of the Content Server session processes
6	After each java core dump, get the output of the following iAPI command to list the sessions: “apply,c,NULL,LIST_SESSIONS,BRIEF_INFO,B,F” “next,c,q0” “dump,c,q0”
7	Get final system information
8	If connection pooling is enabled, get the output of the following iAPI command: “dumpconnectpool,c”
9	Get the DMCL traces if possible. You will have to turn on DMCL tracing and wait for the hang to occur again.
10	Get the Content Server log.
11	Collect all logs for the AppServer
12	Collect relevant machine environment information.
13	Collect the following performance data: CPU usage, memory usage, context switch, disk I/O, and paging I/O. If possible, start collecting performance data 10 minutes before the hang* until you finish collecting all requested dumps. * - When the hang is anticipated to occur.

6. Crash – Quick Reference:

	Collect the following information
1.	Javacore dumps for the AppServer (if it exists)
2.	Process dumps for the AppServer (if it exists)
3.	Process (native) stdout log (if it exists)
4.	Collect all logs for the AppServer (if it exists)
5.	Collect relevant machine environment information.
6.	Get the DMCL traces if possible. You will have to turn on DMCL tracing and wait for the AppServer to crash again.
7.	In the case of an AppServer crash, DMCL error pid files will be created by the DMCL in the working directory.
8.	If possible, try to collect the process memory dumps for the AppServer before the crash* occurs. * - When the crash is anticipated to occur.

7. Links:

- You can install Windbg from the following location: http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx
- You can install StackTrace utility from the following location: http://www.adaptj.com/root/main/download
- You can install DebugView utility from the following location: http://www.sysinternals.com/Utilities/DebugView.html
- More information on how to use HPROF profiling tool can be found at: http://java.sun.com/developer/technicalArticles/Programming/HPROF.html and http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi/jvmpi.html#hprof
- You can install AdPlus from the following location: http://www.microsoft.com/whdc/devtools/debugging/default.mspx
- More information on how to use AdPlus can be found at: http://support.microsoft.com/kb/286350/

- More information on how to user the Performance Monitor tool can be found at: http://www.microsoft.com/technet/prodtechnol/Windows2000Pro/reskit/part6/proch27.mspx?mfr=true

- For WebSphere, if you notice that javacores were not generated for Kill -3 then, please follow instructions at the following link to get a user.dmp file: http://www-1.ibm.com/support/docview.wss?rs=180&context=SSCMPB9&q1=MustGatherDocument&uid=swg21145345&loc=en_US&cs=utf-8&lang=en

- You can download the ThreadAnalyzer tool from the following location: http://www-128.ibm.com/developerworks/websphere/downloads/thread_analyzer.html

- You can download the Samurai (Thread Analyzer Tool) from the following location: http://yusuke.homeip.net/samurai/?english