**JVM(Java Virtual Machine)**

**JVM Architecture:** JVM is the heart of any Java based Application Server. We face most of the issues due to incorrect JVM tuning. It is very important to understand the Overall architecture of the JVM in order to trouble shoot different JVM tuning related issues. Here we are going to discuss the Architecture and the Major parts of a Java Process And the Java Heap Division.

The Following Diagram is just a basic overview of a Java Process in a 2 GB process Size Machine. Usually in 32 bit Windows Operating Systems the default process size will be 2 GB (In Unix based 64 bit operating Systems it can be 4GB or more). So i draw the following Diagram of Java Process to explain the Java Process partitions in a 2Gb process size machine.

**Java Process Architecture Diagram**

In the above diagram we will find different partitions of a Java Process. Please compare the above diagram with below descriptions.

.

**1)** Just for Example we can see that Process Size is 2048 MB (2GB)

**2)**The Java Heap Size is 1024MB (means 1GB)   -Xmx1024m

**3)**Native Space = ( ProcessSize – MaxHeapSize – MaxPermSize) It means around 768 MB of Native Space.

**4)**MaxPermSpace is around -XX:MaxPermSize=256m

**5)**Young Generation Space is around    40% of Maximum Java Heap.

**Different parts of JVM:**

**1) Eden space:** Eden Space is a Part of Java Heap where the JVM initially creates any objects, where most objects die and quickly are cleaned up by the minor Garbage Collectors (Note: Full Garbage Collection is different from Minor Garbage Collection). Usually any new objects created inside a Java Method go into Eden space and the objects space is reclaimed once the method execution completes. Where as the Instance Variables of a Class usually lives longer until the Object based on that class gets destroyed. When Eden fills up it causes a minor collection, in which some surviving objects are moved to an older generation.

2) **Survivor Spaces:** Eden Space has two Survivor spaces. One survivor space is empty at any given time. These Survivor Spaces serves as the destination of the next copying collection of any living objects in Eden and the other survivor space.

The parameter SurvivorRatio can be used to tune the size of the survivor spaces.

-XX:SurvivorRatio=6 sets the ratio between each survivor space and Eden to be 1:6

If survivor spaces are too small copying collection overflows directly into the tenured generation.

3) **Young Generation: (-XX:MaxNewSize):** Till JDK1.3 and 1.4 we used to set the Young Generation Size using **-XX:MaxNewSize**. But from JDK1.4 onwards we set the YoungGeneration size using (**-Xmn**) JVM option.

Young Generation size is controlled by NewRatio.  It means setting -XX:NewRatio=3 means that the ratio between the Old Generation and the Young Generation is  1:3. Similarly -XX:NewRatio=8 means that 8:1 ratio of tenured and young generation.

**NewRatio:**NewRatio is actually the ratio between the (YoungGenaration/Old Generations) has default values of 2 on Sparc , 12 on client Intel, and 8 everywhere else.

**NOTE:**After JDK 1.4 The Young Generation Size can be set using  (**-Xmn**) as well.

**1) Virtual space-1:(MaxNewSize – NewSize):** The First Virtual Space is actually shows the difference between the -XX:NewSize and -XX:MaxNewSize.  Or we can say that it is basically a difference between the Initial Young Size and the Maximum Young Size.

**JavaHeapArea:( -Xmx and –Xms):** Java Heap is a Memory area inside the Java Process which holds the java objects.  Java Heap is a combination of Young Generation Heap and Old Generation Heap. We can set the Initial Java Heap Size using -Xms JVM parameter similarly if we want to set the Maximum Heap Size then we can use -Xmx JVM parameter to define it.

**Example:**

**-Xmx1024m** —> Means Setting the Maximum limit of Heap as 1 GB

**-Xms512m**—> Means setting Java Heap Initial Size as 512m

.

**NOTE-1):** It is always recommended to set the Initial and the Maximum Heap size values as same for better performance.

**NOTE-2):** The Theoretical limitation of Maximum Heap size for a 32 bit JVM is upto 4GB. Because of the Memory Fragmentation, Kernel Space Addressing, Swap memory usages and the Virtual Machine Overheads are some factors JVM does not allow us to allocate whole 4GB memory for Heap in a 32 bit JVM. So usually on 32-bit Windows Operating Systems the Maximum can be from 1.4 GB to 1.6 GB.

If we want a larger memory allocation according to our application requirement then we must choose the 64-bit operating systems with 64 bit JVM. 64-bit JVM provides us a larger address space. So we can have much larger Java Heap  with  the increased number of Threads allocation area. Based on the Nature of your Operating system in a 64 bit JVM you can even set the Maximum Heap size upto 32GB.

Example:        *-Xms32g -Xmx32g -Xmn4g*

**2) Virtual Space-2: (MaxHeapSize – InitialHeapSize):** The Second Virtual Space is actually the Difference between the Maximum Heap size (**-Xmx**)and the Initial Heap Size(**-Xms**). This is called as virtual space because initially the JVM will allocate the Initial Heap Size and then according to the requirement the Heap size can grow till the MaxHeapSize.

**PermGen Space: (-XX:MaxPermSize):** PermGen is a non-heap memory area where the Class Loading happens and the JVM allocates spaces for classes, class meta data,  java methods and the reference Objects here. The PermGen is independent from the Heap Area. It can be resized according to the requirement using -XX:MaxPermSize and -XX:PermSize  JVM Options. The Garbage collection happens in this area of JVM Memory as well. The Garbage collection in this area is called as “Class GC”. We can disable the Class Garbage Collection using the JVM Option -noclassgc. if  ”-noclassgc” Java Option is added while starting the Server. In that case the Classes instances which are not required will not be Garbage collected.

**Native Area:** Native Memory is an area which is usually used by the JVM for it’s internal operations and to execute the JNI codes. The JVM Uses Native Memory for Code Optimization and for loading the classes and libraries along with the intermediate code generation.

The Size of the Native Memory depends on the Architecture of the Operating System and the amount of memory which is already commited to the Java Heap. Native memory is an Process Area where the JNI codes gets loaded or JVM Libraries gets loaded or the native Performance packs and the Proxy Modules gets loaded.

There is no JVM Option available to size the Native Area. but we can calculate it approximately using the following formula:

**NativeMemory = (ProcessSize – MaxHeapSize – MaxPermSize)**

**Garbage collection:** It’s always best to enable the Garbage collection Logging in our production environment as well because it does not cause any resource overhead or any side effect on WebLogic server or another application server’s performance.  GC log helps us in investigating man issues. Apart from issues it helps us to find out if some tuning is required based on the statistics of the Garbage collection. Garbage collection logging can be enable and collected in a separate log file by using the following JAVA\_OPTIONS:

**-Xloggc:D:/gcLogs/GCLogs.log         -XX:+PrintGCDetails        -XX:+PrintGCTimeStamps**

As soon as you add these JAVA\_OPTIONS which are JVM specific (above will work for Sun and Open JDKs fine) the JVM will start generating the garbage collection logging in the GCLog.log file. Now if you will open this file then you can

**see something like following:**

|  |  |
| --- | --- |
| *01* | *4.636: [GC [PSYoungGen: 230400K->19135K(268800K)] 230400K->19135K(2058752K), 0.0635710 secs] [Times: user=0.08 sys=0.01, real=0.06 secs]* |
| *02* | *7.302: [GC [PSYoungGen: 249535K->38396K(268800K)] 249535K->51158K(2058752K), 0.0777300 secs] [Times: user=0.21 sys=0.04, real=0.07 secs]* |

|  |  |
| --- | --- |
| *03* | *7.521: [GC [PSYoungGen: 49735K->38388K(268800K)] 62496K->51933K(2058752K), 0.0741680 secs] [Times: user=0.15 sys=0.04, real=0.07 secs]* |
| *04* | *7.595: [Full GC (System) [PSYoungGen: 38388K->0K(268800K)] [PSOldGen: 13545K->51794K(1789952K)] 51933K->51794K(2058752K) [PSPermGen: 19868K->19868K(39936K)], 0.3066610 secs] [Times: user=0.28 sys=0.02, real=0.31 secs]* |

|  |  |
| --- | --- |
| *05* | *9.752: [GC [PSYoungGen: 230400K->26206K(268800K)] 282194K->78000K(2058752K), 0.0728380 secs] [Times: user=0.15 sys=0.00, real=0.08 secs]* |
| *06* | *11.906: [GC [PSYoungGen: 256606K->38393K(268800K)] 308400K->94759K(2058752K), 0.1058920 secs] [Times: user=0.19 sys=0.00, real=0.10 secs]* |

|  |  |
| --- | --- |
| *07* | *13.480: [GC [PSYoungGen: 268793K->38394K(268800K)] 325159K->109054K(2058752K), 0.0762360 secs] [Times: user=0.20 sys=0.03, real=0.08 secs]* |
| *08* | *18.115: [GC [PSYoungGen: 268794K->38384K(268800K)] 339454K->179238K(2058752K), 0.1351350 secs] [Times: user=0.42 sys=0.10, real=0.14 secs]* |

|  |  |
| --- | --- |
| *09* | *20.860: [GC [PSYoungGen: 268784K->38394K(268800K)] 409638K->200343K(2058752K), 0.1063430 secs] [Times: user=0.29 sys=0.03, real=0.11 secs]* |
| *10* | *22.148: [GC [PSYoungGen: 268794K->38399K(268800K)] 430743K->221395K(2058752K), 0.1173980 secs] [Times: user=0.24 sys=0.02, real=0.12 secs]* |

|  |  |
| --- | --- |
| *11* | *23.357: [GC [PSYoungGen: 268799K->26775K(268800K)] 451795K->231618K(2058752K), 0.0714130 secs] [Times: user=0.15 sys=0.03, real=0.08 secs]* |
| *12* | *24.449: [GC [PSYoungGen: 257175K->29170K(268800K)] 462018K->239909K(2058752K), 0.0312400 secs] [Times: user=0.06 sys=0.01, real=0.04 secs]* |

You can notice something in the above output:

**Point1:** [Full GC (System) [PSYoungGen: 38388K->0K(268800K)]    It means a Full GC is happening on the complete Heap Area including all the Areas of the Java Heap Space.

**Point2:** [GC [PSYoungGen: 230400K->19135K(268800K)]   Indicates some small GCs which keep on happening in the young generation very frequently, This garbage collection cleans the Young Generation short living Objects.

**Point3:**  Meaning of the [GC [PSYoungGen: 230400K->19135K(268800K)]   line is around 256MB (268800K) is the Young Generation Size, Before Garbage Collection in young generation the heap utilization in Young Generation area was around  255MB (230400K)  and after garbage collection it reduced up to 18MB (19135K)

**Point4:**  Same thing we can see for Full Garbage collection as well….How effective the Garbage collection was…[Full GC (System) [PSYoungGen: 38388K->0K(268800K)] [PSOldGen: 13545K->51794K(1789952K)]  Here it says that around

[(old)1789952K +  young (268800K) ]  memory space means  OldGeneration is consuming 1.75GB space and Young Generation is consuming around 255 MB space  So it means total Heap size is around 2GB.

But analyzing the Garbage collection log like above technique Line by Line is very bad…so here we have an alternative was to analyze the Garbage Collection log in few Seconds to see how much time the Full Garbage collection is taking as an average and other reports…etc.

**Step1):**Download the “garbagecat-1.0.0.jar   (881 KB) ”  tool from the follwing link:<http://garbagecat.eclipselabs.org.codespot.com/files/garbagecat-1.0.0.jar>

**Step2):** Open a command prompt and then make sure that JAVA is set in the Path so that we can use “jar” utility of JDK to run the “garbagecat-1.0.0.jar”  tool.

**Step3):** Put the “garbagecat-1.0.0.jar”  file and the “GCLog.log” file in the same directory. then run the following command:

**java      -jar      garbagecat-1.0.0.jar      GCLog.log**

**Step4):** As soon as our run the above command you will see that in your current directory following files are created:

garbagecat-1.0.0.jar

GCLog.log

gcdb.lck

gcdb.log

gcdb.properties

report.txt

**Step5):** Now open the “report.txt” file to see the Overall report of the Garbage Collection something like following:

========================================

SUMMARY:

========================================

# GC Events: 12

GC Event Types: PARALLEL\_SCAVENGE, PARALLEL\_SERIAL\_OLD

Max Heap Space: 2058752K

Max Heap Occupancy: 462018K

Max Perm Space: 39936K

Max Perm Occupancy: 19868K

Throughput: 95%

Max Pause: 306 ms

Total Pause: 1233 ms

First Timestamp: 4636 ms

Last Timestamp: 24449 ms

========================================

.

If you see that the Garbage Collection Max Pause time is very high like more than 5-7 Seconds for a 2 GB heap then you need to worry about it.

**NOTE:** Garbage collection  is a best utility to generate the Garbage Collection Report for Sun JDK and Open JDK for other JDKs you should use other tools for accurate results.

**Garbage collection algorithms:**

1) **Mark-and-sweep collector:**  This type of collector first traverses the object graph and marks reachable objects. It then scans the heap for unmarked objects and adds their memory to a list of available memory segments. This collector typically uses a single thread to do its work and is a stop-the-world collector.

**2) Mark-and-compact collector:**  A mark-and-compact collector, sometimes known as a mark-sweep-compact collector, uses the same marking phase as a mark-and-sweep collector. During the second phase, it compacts the heap by copying marked objects to a new area of the heap. These collectors are also stop-the-world collectors.

**3) Copying collector:**  This type of collector divides the heap into two areas, commonly known as semi-spaces. It uses only one semi-space at a time; the JVM creates all new objects in one semi-space. When the garbage collector runs, it copies any reachable objects it finds to the other semi-space as it finds them, thus compacting the heap as it copies live objects. All dead objects are left behind. This algorithm works well for short-lived objects, but the expense of continually copying long-lived objects makes it less efficient. Again, this is a stop-the-world collector.

**4) Incremental collector:**  Incremental collectors basically divide the heap into multiple areas and collect garbage from only one area at a time. This can create much smaller, though more frequent, pauses in your application. There are numerous approaches defining how the actual collection is handled from traditional mark-and-sweep to algorithms designed explicitly for use with multiple smaller areas like the train algorithm. See “Incremental Mature Garbage Collection Using the Train Algorithm” by Jacob Seligmann and Steffen Grarup ([http://www.daimi.aau.dk/~beta/Papers/Train/train.html](http://www.daimi.aau.dk/%7Ebeta/Papers/Train/train.html)) for more information.

**5) Generational collector**:  This type of collector divides the heap into two or more areas that it uses to store objects with different lifetimes. The JVM generally creates all new objects in one of these areas. Over time, the objects that continue to exist get tenure and move into another area for longer-lived objects. Generational collectors often use different algorithms for the different areas to optimize performance.

**6) Concurrent collectors**:  Concurrent collectors run concurrently with the application, typically as one or more background threads. These collectors typically have to stop-the-world at some point to complete certain tasks, but the amount of time they halt all processing is significantly reduced because of their other background work.

**7) Parallel collectors:**  Parallel collectors typically use one of the traditional algorithms but use multiple threads to parallelize their work on multiprocessor machines. Using multiple threads on multi-CPU machines can dramatically improve the scalability of a Java application on multiprocessor machines.

**Creating And Analyze log for Garbage Collection:**

**Creating GC log:** Set memory argument in startWebLogic.cmd.

-Xms256m –Xmx512m –XX:CompailThreshold=8000 –XX:permSize=48m –XX:MaxPermSize=128m

**analyze GC logs:**

**Srep1:**Go to admin console

**Step2:** Click the Adminserver

**Step3:** Select the monitors tab

**Step4:** Click the performance

**Step5:** Select the garbage collector

**Security Realms:** A security realm is a container for the mechanisms including users, groups, security roals, security palaces and providers. That are used to protect WebLogic resources, we can have multiple security reasons in a WebLogic servers domain. But only one can be set as the default realm.

This security realms page, lists is security realms that has been configured in this WebLogic server domain. Click the name of the realms to explore and configure that realm.

[**Setting Memory Arguments in WebLogic Application Server**](http://www.redstonecontentsolutions.com/technical-blog/setting-memory-arguments-in-weblogic-applicationserver)

If you need to change the memory allocation to your WebLogic administration server and/or managed server, the following guide will show you how. Note: This will raise the limit for both the WebLogic administration server and managed server. You can verify these settings by watching the server startup script. Optionally, if you are setting the memory for Oracle UCM, you can verify this setting on the System Audit Information page.

**Windows**

1. Open the domain environment cmd file:  
  
        <middleware home>\user\_projects\domains\base\_domain\bin\setDomainEnv.cmd  
  
2. Locate the following remark, inside the cmd file: (‘Search’ -> “@REM IF USER\_MEM\_ARGS”)  
  
        @REM IF USER\_MEM\_ARGS the environment variable is set, use it to override ALL MEM\_ARGS                   values  
  
3. Directly after this remark, add the following line:  
  
         set USER\_MEM\_ARGS=-Xms256m -Xmx1024m -XX:CompileThreshold=8000 -XX:PermSize=128m -XX:MaxPermSize=512m  
  
4. The portion of the file you have edited looks like this:  
  
        @REM IF USER\_MEM\_ARGS the environment variable is set, use it to override ALL MEM\_ARGS    values  
  
        set USER\_MEM\_ARGS=-Xms256m -Xmx1024m -XX:CompileThreshold=8000 -XX:PermSize=128m -XX:MaxPermSize=512m  
  
        if NOT “%USER\_MEM\_ARGS%”==”" (  
  
        set MEM\_ARGS=%USER\_MEM\_ARGS  
  
        )

**UNIX/Linux**

1. Open the domain environment sh file:  
        <middleware home>\user\_projects\domains\base\_domain\bin\setDomainEnv.sh  
  
2. Locate the following remark, inside the sh file: (‘Search’ -> “#REM IF USER\_MEM\_ARGS”)  
  
        #REM IF USER\_MEM\_ARGS the environment variable is set, use it to override ALL MEM\_ARGS values  
  
3. Directly after this remark, add the following line:  
  
        USER\_MEM\_ARGS=-Xms256m -Xmx1024m -XX:CompileThreshold=8000 -XX:PermSize=128m -   XX:MaxPermSize=512m  
  
4. The portion of the file you have edited looks like this:  
  
        #REM IF USER\_MEM\_ARGS the environment variable is set, use it to override ALL MEM\_ARGS values  
  
        USER\_MEM\_ARGS=-Xms256m -Xmx1024m -XX:CompileThreshold=8000 -XX:PermSize=128m -XX:MaxPermSize=512m  
  
        if NOT “%USER\_MEM\_ARGS%”==”" (  
  
        MEM\_ARGS=%USER\_MEM\_ARGS  
  
        )

**Trouble shooting issues:**

**Deployment:** We will send error log to the application team for modification.

Caused By: WebLogic.utils.ErrorCollectionException:

There are 1 nested errors:

WebLogic.j2ee.dd.xml.AnnotationProcessException: Duplicate ejb name 'BDAccountEjbBean' found: annotation 'Stateless' on bean failed due to connection pool issue: we will fix connection pool issues and then redeploy the application

Out of memory issue during the deployment:

error: **java.lang.outofmemory.permgenspace**

this error occured due to space in perm area.

setDomainEnv.sh

xx:permsize 128m

xx:maxpermsize 128m

we have set intialpermsize=maxpermsize then restarted the servers, redeployed the application

If one or two application failed when we are triggering through scipt.we will fix that issue and do a deployment using console

**JDBC:**

1) DB down (raise a ticket to db team)

2) In correct hostname or port number ( raise a ticket to network team)

3) Data base connection lost ( telnetipaddress port )

4) Data base user\_acc lock ( raise a ticket to db team for unlocking user\_acc)

5) Invalid pakage error (raise a ticket to db team)

6) TNS listener error (raise a ticket to db team)

7) Schema does not exist (raise a ticket to db team)

8) Cannot allocate resource error

Intialcapacity : 5

max : 15

increase max to 25

9) Connection leaks ( send error to application team)

10) Connection time out ( raise a tickect to db team for long running quries)

**JMS:**

stuck message issues

Check whether dest queue is available, check message format, check queue name.

rolling message issues (messages will run continuously in the loop)

delete those messages in the queue.

**Disk Space:**

If the disk space usage is 95%-100% then we will delete old log files

[root@localhost ~]# df -kh

FilesystemSize Used Avail Use% Mounted on

/dev/sda2 3.8G 1.9G 1.8G 52% /

/dev/sda1 46M 9.2M 35M 22% /boot

tmpfs 506M 0 506M 0% /dev/shm

/dev/sda3 14G 1.8G 12G 100% /home

du -kh (disk usage)

s

[root@localhost ~]# du -sh /home

1.8G /home

[root@localhost bea10.3]# du -sh \*

181M jdk160\_05

28K logs

211M jrockit\_160\_05

100M modules

24K registry.dat

8.0K registry.xml

19M user\_projects

556K utils

429M wlserver\_10.3

delete old log files

/home/bea10.3/user\_projects/domains/sherkhan/servers/AdminServer/logs

rm -rf Adminserver.log00001 Adminserver.log00002 Adminserver.log00003

rm -rf Adminserver.out00001 Adminserver.out00002 Adminserver.out00003

rm -rf access.log00001 access.log00002 access.log00003

/home/bea10.3/user\_projects/domains/sherkhan/servers/ms1/logs

rm -rf ms1.log00001

rm -rf ms1.out00001

or zip the log files

/home/bea10.3/user\_projects/domains/sherkhan/servers/AdminServer/logs

gzip -r \*

/home/bea10.3/user\_projects/domains/sherkhan/servers/AdminServer

gzip -r logs

**High CPU utilization:**

top (linux)

prstat (solaris)

top - 07:45:22 up 3:03, 3 users, load average: 0.16, 0.33, 0.17

Tasks: 113 total, 2 running, 109 sleeping, 0 stopped, 2 zombie

Cpu(s): 0.0%us, 0.7%sy, 0.0%ni, 99.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Mem: 1035400k total, 1020348k used, 15052k free, 77688k buffers

Swap: 2040212k total, 0k used, 2040212k free, 483724k cached

%cpu %Mem

9523 root 22 0 637m 239m 3660 S 98.7 23.7 0:12.79 java

If you find any zombie process count >50 raise a ticket to solaris admins

If any java processes are occupying 95-100% cpu usage then check the log files for any continuous looping messages or jdbc transaction time outs.

fix the problem and kill manged server using kill -9 pid and restart the service instance.

**404 error:**

page can't be displayed.

10.4.5 404 Not Found

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent.

1) check whether they are using correct url

2) check whether apache server is running ( ps -ef | grephttpd) ( ps -ef | grep -i apache)

3) check the diskspace of Apache server if it is full then delete the log files (df -kh)

goto Apache2.2/logs

delete old logs

4) Check whether the deployed application is in active state

5) If the deployed application is failed then fix the issue and redeploy the application

**Users are getting 404 error some times and they are able to access the application sometimes.**

1) check whether all managed servers are in running state.

if one of the managed server is in shutdown state then bring up the server.

check the http requests in access.log file for all managed server

if you are getting 404 error in one of the managed server log. then check server log for any errors

i got the in log file:

port already in use

netstat -anp | grep 8002

if the port is listened on any other instance. restat managed server.

if the issue still persists then raise a reqest to network team..

**500 error:**

Service unavailable

this error is due to server down

check apache or WebLogic service instance is the server is down then start the server.

**Slow response:**

check All WebLogic server status. bring the servers up if they are down

check network handshake requests in application logs. If you found any issues related to n/w then raise request to n/w team.

check for stuck thread issues in WebLogic. If you found any stuck thread issues then take thread dump and analyze.

checkcpu usage for java processes.

check heap size of WebLogic server gc log or in console.

If the heap size is more than 80% then take heap dump send it to l3 support.

check no of users logged in to the application.

check for long running quiries from data base side.

check for latency in data base side.

check memory leaks in gc logs.

check connection leaks in the WebLogic server side.

check space in WebLogicunix machine.

check apache server space.

**OOM(OutOfMemory):**

- Login to the Corresponding Server through Putty

- Then Check the Status of the Server instances

- Check the Server logs and Out logs for OutOfMemory Error

- Take the Access logs at the time of OOM and it will be good if we take thread dump

If Server(s) is/are in Running State.

- Analysis the Thread dump for the Cause of OutOfMemory Error (Due to App/Server)

- Then Depending on the Server Status (if not in Running State) Restart the Server.

**i) OutOfMemory during deployment:**

1.                  If the application is huge(contains more than 100 JSPs), we might encounter this problem with default JVM settings.

2.                  The reason for this is, the MaxPermSpace getting filled up.

3.                  This space is used by JVM to store its internal datastructures as well

4.                  as class definitions. JSP generated class definitions are also stored in here.

5.                  MaxPermSpace is outside java heap and cannot expand dynamically.

6.                  So fix is to increase it by passing the argument in startup script of the server: –XX:MaxPermSize=128m (default is 64m)

**ii) Memory related issues can be caused by many problems including:**

• Java Heap is undersized for the environment

• There is not enough native memory available for the java process

• Web server is overloaded

Below are some general guidelines on how to address memory-related issues. In addition, you should search the “My Oracle Support” knowledge base for the specific “OutOfMemory” error message that you see in your PIA\_WebLogic.log.

Also, you can collect further details on WebLogic memory usage by using monitoring tools referenced in section “Monitoring WebLogic Memory Usage”

**General Guidelines on Fixing OutOfMemory Issues:**

1. First, you need to determine if WebLogic is running out of “native heap” memory or “java heap” memory. Typically you are able to tell this by checking the “OutOfMemory” error message in the PIA\_WebLogic.log:

a. If the error message refers to “native” or to a “thread” related error, it is an issue with “nativememory”. Examples of errors due to running out of native memory are:

• “Unable to create new native thread”

• “Error starting thread: Not enough storage is available”

Native memory errors are more likely to occur on PeopleTools 8.50 and lower versions where you are running a 32-bit java process (which has address space limitations)

b. Any other error messages, are usually due to running out of java heap memory

2. If you are running out of java heap, you may want to start by increasing the java heap settings (if you are unable to increase java heap setting, then the other option is to add more WebLogic PIA’s to the environment).

Increase the java heap setting as follows:

a. First check the current heap setting by searching string “-Xmx” in your WebLogic log (for Unix, search PIA\_stdout.log. For Windows, search NTservice-<DOMAIN\_NAME>-PIA.log).

The “-Xmx” value shows you the current heap setting. For example, these settings show that the minimum heap (-Xms) and maximum heap (-Xmx) are set to 512mg:

Java command line=java -server –Xms512m –Xmx512m

b. For PeopleTools 8.50, try increasing the heap (at 256mg increments) up to 1.5gb. For PeopleTools 8.51 or 8.52, you can increase the heap even higher, provided the server has enough memory available.

For Unix, you can change the heap setting in file setEnv.sh. Example:

JAVA\_OPTIONS\_LINUX="-jrockit -XnoOpt -XXnoJITInline –Xms768m –Xmx768m

• For Windows, you will either need to change the setting in the Windows registry, or else change setting in setEnv.cmd and then rebuild the Windows Service

**Refer to the following document for more details on changing the java heap setting:**

Doc#638298.1: How To Increase/Decrease JVM Heap Size for WebLogic

3. If you are running out of native memory heap, then you may want to consider doing the following:

a. Lower the java heap setting (ieXmx/Xms settings) in order to allow more of the java process’ memory to be used for “Native Memory”. (see step 2b above, for instructions on changing java heap setting)

If the java heap is already being fully utilized, and you are unable to lower it, then you may want to consider adding additional PIA’s to your environment

b. Lower the thread stack size. Note that the threads use native memory, so if you lower the thread stack size, then the threads will not consume as much memory. The thread stack size is specified using parameter “-Xss”. Refer to the following document for details

651285.1: WebLogic Error: "java.lang.OutOfMemoryError: unable to create new native thread"

**Log files not rotating**:

check the diskspace if it is full then delete old logs

check whether log4j properties file set in classpath

1.  Check the Status of the Server

2. ./startWebLogic.sh

3. ./startManagedWebLogic.sh <manageservername>

4.   [0R]

5. Check through console.

6.  Check the disk Space(if full, Delete the logs and then need to restart the Server)

7.   du –kh (folder)

8.     df –kh (filesystem)

9.  Avail space is 90%

If full , mv <source path><destination path>

Delete, rm –rf<filename: adminserver.log>

**Stuck thread:**

"[STUCK]”

When a transaction is running more than 5 minutes, a message (example below) is logged to the PIA\_WebLogic.log.

<Apr 18, 2011 12:47:04 PM PDT><Error><WebLogicServer><BEA-000337><**[**STUCK] ExecuteThread: '4' for queue: 'WebLogic.kernel.Default (self-tuning)' has been busy for "675" seconds working on the request …..

Note that the message shows that the thread is ‘STUCK’. But in fact, the thread may not be stuck, but is just taking a long time to complete. These threads often successfully complete, if given enough time.

If you see a lot of long-running threads, at the time users are experiencing problem, then this indicates that the web server is having issues processing threads, which may cause the web server to hang.

Long running threads can be caused by different issues. The problem often occurs due to issues on the app server or database subsequently causing the threads to queue up and wait on the web server. So if you see a lot of stuck threads, you may want to troubleshoot further by doing the following:

1. Get a thread dump as described in section "Creating/Analyzing Thread Dumps". The thread dump may help you determine whether the threads are getting hung up on the app server or database.

2. Have your DBA check for long running SQL's and/or DB locks

3. You can also look at the 'Stuck' thread messages in the PIA\_WebLogic.log to see what user(s) are running the transactions and the specific component they are running. This may help you determine if there is a specific user and/or transaction that is causing the problems.

**Port Conflict Issue:**

While configuring a new WebLogic instance and starting it, that might be get an issue like : "Port already in use". There could be many reasons for this one.

1. on the same machine multiple standalone instances might be running one of the instance already used that port which you have given for new configuration.

2. apache might be running with the same port.

3. middleware might be running on the same machine with same port

On Solaris Operating environment we have 2 options:

1. usingpfiles command

netstat –na|grep --> identify port in use

pfiles |grep -isockname |grep port --> look for every java process is initialized by startWebLogic.sh or startManagedWebLogic.sh

2. Another way costly one (Third party package) to find the process that is using particular port is :

lsof -itcp:

3. Best way is perl script using a method it will check only standard ports which are used by the system.

getservbyport(intport\_number, const char \*protocol\_name)

#!/usr/bin/perl

($name, $aliases, $port\_number, $protocol\_name) = getservbyport(7001, "tcp");

print "Name = $name\n";

print "Aliases = $aliases\n";

print "Port Number = $port\_number\n";

print "Protocol Name = $protocol\_name\n";

**JVM memory arguments:**

-XX:-PrintGCDetails outputs detailed information at each collection

-XX:-PrintGCTimeStamps outputs a time stamp at the start of each collection

-xloggc=<filename> outputs gc information to the specified file

-XX:-DisableExplicitGC disable calls to system .gc( )

- -XX:NewSize=2m default size of new generation

-XX:MaxNewSize=size maximum size of the new generation

-XX:PermSize=64m default size of permanent generation

-XX:MaxPermSize=64m maximum size of the permanent generation

- -Xms256m Initial heap size

--Xmx512m maximum heap size

-xx:survivor Ratio=<value> Ratio of survivors spaces to young generation

-XX:-UseParallelGC Use parallel garbage collection for scavenges.

**THREAD DUMP**

**Tread dump:-** Thread dump provides a snapshot of the current active live threads. It provides the stack trace of all the java threads in the JVM. It is used when the server is hung and we want to see the threads executing and take their dump.

There are different ways to take thread dump.

**In unix**: kill -3 <pid>

**In windows:** ctrl+break

**WebLogic.Admin utility:** javaWebLogic.Admin -url t3://localhost:7001 -username WebLogic -password WebLogic THRED\_DUMP

**WLST Scripting:**

connect('WebLogic','WebLogic','t3://localhost:7001')

cd('server')

cd('AdminServer')

TreadDump()

disconnect()

exit()

**Admin console:**

**Step1:** login to the admin console

**Step2:** Click on server

**Step3:** Navigate to servers

**Step4:** Click monitor tab

**Step5:** Click on tread

**Step6:** Click on the dumpthread stack.

**Locating the Thread Dump:** The thread dump is placed in the WebLogic log file. The log file location varies depending on the OS platform:

**For UNIX:** the output is sent to:

<PS\_HOME>/webserv/<DOMAIN\_NAME>/servers/logs/PIA\_stdout.log

**For Linux:** the output is sent to:

<PS\_HOME>/webserv/<DOMAIN\_NAME>/servers/logs/PIA\_stderr.log

**For Windows:** the output is sent to:

<PS\_HOME>\webserv\<DOMAIN\_NAME>\servers\PIA\logs\NTservice-<DOMAIN\_NAME>-PIA.log

**Analyzing a Thread Dump:** The thread dump can be a bit challenging to analyze, and you may need assistance from an Oracle Support Engineer. Below are some tips on how to analyze the thread dump. This information is broken out into the following sections:

1. General Information about the thread dump

2. Overview of types of threads commonly seen in thread dump

3. Examples of different issues you may observe in the thread dump

**1) General Information about the Thread Dump:** Note that the thread dump always begins with this line:

**===== FULL THREAD DUMP ===============**

And ends with this line:

**===== END OF THREAD DUMP ===============**

The first line of the thread dump shows when the thread dump was created, followed by the exact java version you are using.

Example:

**Mon Apr 18 12:46:56 2011**

**Oracle JRockit(R) R28.0.0-679-130297-1.6.0\_17-20100312-2123-windows-ia32**

**2) Overview of Types of Threads commonly seen in Thread Dump:**

**i) Threads waiting for Requests:** You will always see some threads that are just waiting for work, as WebLogic always allocates some threads to be available and ready to process any incoming requests. These threads can easily be identified because you’ll see **“ExecuteThread.waitForRequest”** in the call stack. These threads will be in ‘ACTIVE’ or ‘STANDBY’ mode. These threads do not have much significance when troubleshooting. However, if you see a lot of these threads waiting for requests (20 or more), it most likely indicates that the environment is just recovering from a very heavy load, when the thread dump was taken (and as the load diminishes, WebLogic will remove many of these extra threads that are waiting for requests)

**Ex: at WebLogic/work/ExecuteThread.waitForRequest(ExecuteThread.java:157)**

**ii) Socket Muxer Threads:** You will also see approximately two to five socket muxer threads. These threads' main responsibility is to read the request off the socket and pass the work to the appropriate thread. WebLogic allocates a percentage of execute threads from the self-tuning thread pool to be Muxer threads. Usually you will see three or four of these threads:

**"ExecuteThread: '0' for queue: 'WebLogic.socket.Muxer'" id=25 idx=0x60 tid=2068 prio=5 alive, in native**

**iii) ListenThreads:** You will also see approximately six “listen threads”, usually three for SSL and three for non-SSL. The purpose of these threads is to wait for connections to arrive. All browser requests enter the WebLogic server through these threads.

**"DynamicListenThread[Default]" id=39 idx=0x90 tid=2812 prio=9 alive, in native**

**"DynamicSSLListenThread[DefaultSecure]" id=40 idx=0x94 tid=3148 prio=9 alive, in native**

**iv) Jolt Connection Threads:** WebLogic Server and the Tuxedo Application Server use Jolt to communicate with each other. PIA creates two threads inside the WebLogic’s JVM per Jolt connection. For each Jolt connection made between WebLogic and the Tuxedo Application Servers, you will see a LLENwReader and a LLENwWriter thread in the thread dump:

**"LLENwReader" id=52 idx=0xc4 tid=4408 prio=5 alive, in native, daemon**

**"LLENwWriter" id=53 idx=0xc8 tid=7828 prio=5 alive, waiting, native\_blocked, daemon**

**v) Threads waiting on Application Server:** If the web server is waiting on the app server to process a request, you will see the following thread (below)

**at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)**

**3) Examples of Different Issues you may Observe in Thread Dump:** Below are examples of different issues and the thread stacks you may observe.

**Many threads waiting on App Server:** If you see a lot of threads such as the one below, then this means that many of the WebLogic threads are waiting on the application server to finish processing the request:

**at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)**

**i) Many threads processing the same call stack:** If you see many threads all processing the same call stack, then you may need to review contents of the call stack in order to troubleshoot the issue. For example, in one case, the web server hung and the thread dump showed hundreds of threads like the one below. This was caused by an issue with a proxy server configuration, causing all threads to get hung up at logout:

**com.sun.net.ssl.internal.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:798)**

**psft.pt8.psp.logoutAccessedPIAs(Unknown Source)**

**ii) All threads busy and waiting on one thread:** By design, the PIA does not allow more than one request per HTTP session, to be submitted to the application server. If the PIA receives multiple requests from the same HTTP session, it will queue up all subsequent requests and process just one at a time. Typically, there should not be situations where the PIA receives multiples requests from the same HTTP session. However, this can occur in the following situations:

1. You are using a proxy server that is re-submitting requests to the web server if a response is not received within a certain time.

-OR-

2. A user submits a long-running request, and while waiting for the request to finish, the user continuously attempts to submit more requests.

When one of the above scenarios occurs, in the thread-dump you see one request waiting on Jolt to get response from the App-Server and many other threads waiting for the lock on the session to be released. Below are excerpts from a thread dump, showing this situation:

**a)** There are many threads like this that are “blocked”, and all the threads are waiting on the same lock #.

**-- Blocked trying to get lock: java/lang/String@0x27D36AC0[thin lock]**

**b)** The thread that is holding the lock on “0x27D36AC0” (that all blocked threads are waiting on), is usually processing a jolt request (ie it is waiting on the application server):

**at bea/jolt/IOBuf.waitOnBuf(IOBuf.java:119)**

**^-- Holding lock: java/lang/String@0x27D36AC0[thin lock]**

**c)** At the end of the thread dump, you may see a list of “blocked locked chains”. In this list, you’ll notice that all threads are waiting on one thread: “Thread #0” in this example. Which happens to be a jolt request (ie it is waiting on application server)

**Blocked lock chains**

**===================**

**Chain 2: "[ACTIVE] ExecuteThread: '2' for queue: 'WebLogic.kernel.Default (self-tuning)'" id=35 idx=0x80 tid=3964 waiting for java/lang/String@0x27D36AC0 held by:**

**"[ACTIVE] ExecuteThread: '0' for queue: 'WebLogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 in chain 1**

**Chain 3: "[ACTIVE] ExecuteThread: '3' for queue: 'WebLogic.kernel.Default (self-tuning)'" id=44 idx=0xa4 tid=4620 waiting for java/lang/String@0x27D36AC0 held by:**

**"[ACTIVE] ExecuteThread: '0' for queue: 'WebLogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 in chain 1**

**Chain 4: "[ACTIVE] ExecuteThread: '4' for queue: 'WebLogic.kernel.Default (self-tuning)'" id=49 idx=0xb8 tid=1120 waiting for java/lang/String@0x27D36AC0**

**held by:**

**"[ACTIVE] ExecuteThread: '0' for queue: 'WebLogic.kernel.Default (self-tuning)'" id=16 idx=0x48 tid=180 in chain 1**

**Analysing ThreadDump by using Summari tool:**

Download: The binary is available for download at <http://yusuke.homeip.net/samurai/en/samurai.jar>

**How to launch samurai:** You can simply double-click to launch Samurai on your desktop or type as following in your command prompt.

$java -jar samurai.jar

Automatic update is not available with this way. Please check and download latest version manually.

**Step1:** Drag and drop the ThreadDump into summary tool

**Step2:** When Samurai detects a thread dump in your log, a tab named "Thread Dump" will appear.

**Step3:** You can just click "Thread dumps" tab to see the analysis result. Samurai colors idle threads in **gray**, blocked threads in **red** and running threads in **green**.

There are two resultant views and Samurai shows **"Table view"** by default.  
In many cases, you are just interested in the table view and the **sequence view**. Use the table view to decide which thread needs be inspected, the sequence view to understand the thread's behavior.

**Result1:**

**Result2:**

**HeapDump:** A Heapdump is a snapshot of JVM memory – it shows the live objects on the heap along with references between objects. It is used to determine memory usage patterns and memory leak suspects. It is useful to analyse OOM(OutOfMemory) situations.

**To take Heap dump:**

Eclipse Memory Ananlyser is a very useful tool to analyze heap dumps. It has a lot of features such as Memory Leak detection where it runs an automated test to determine the suspected leaks.

Step 1) Start the WebLogic Server, with the application in active state which causes memory leak.

Step 2) Get the process id of the server using jps

Step 3) Access the application that causes memory leak

Step 4) Take heap dump at regular interval using jmap.

jmap -dump:format=b,file=dump1.bin [processId]

**Analyzer HeapDump by using Eclipse MAT:** Analyzer Open the Heap Dump in Eclipse Memory Analyzer (U can download it from<http://www.eclipse.org/mat/downloads.php>) Approximate size is 42 MB. Just u need to extract this Zip then u can directly start (no Installation needed)

**Step1:**

**Step2:** Observe the heap usage of Objects in the heap dumps. If the object instance keeps on increasing in the subequent heap dumps, force a garbage collection from the WebLogic Server console.

**Step3:** Take heap dumps again and open in the Eclipse Memory Analyzer. If the number of instances still don’t go down for those objects, you can expect to see this  
**<Jul 16, 2010 10:49:15 AM IST> <Critical> <Health> <BEA-310003> <Free memory in  
the server is 47,856 bytes. There is danger of OutOfMemoryError>  
Exception in thread “Thread-12″ java.lang.OutOfMemoryError: Java heap space  
at demo.MemoryLeakTest.runTest(MemoryLeakTest.java:14)  
at jsp\_servlet.\_\_memoryleak$1.run(\_\_memoryleak.java:86)  
at java.lang.Thread.run(Thread.java:619)**

If the leak is happening due to a WebLogic Class, it can be a known issue or an undiscovered BUG. You need to get in touch with Oracle Support. If it’s an Application Class, you need to contact the developers. Out of Memory can also happen dude to third party codes such as database drivers.

**How to install Eclips MAT? How to Analyze Heapdump using Eclips MAT and Jhat tools:**

**Step1:** download the Eclips MAT

**Step2:** Extract Memory Analyser zip file and open MemoryAnalyzer.ini

**Step3:**  Double-click on MemoryAnalyzer.exe to start Memory Analyser Tool

**Step4:** Select Search for new features to install and click Next

**Step5:** Accept the license agreements and click Next

**Step6:** Click Finish to install the extensions

**Step7:** Click Install All to ignore the warning

**Step8:** Restart Memory Analyser to reflect changes.

What is a Thread dump? How will you take in unix/linux and windows?

ANSWER:

A Java thread dump is a way of finding out what every thread in the JVM is doing at a particularpoint in time. This is especially useful if your Java application sometimes seems to hang when running under load, as an analysis of the dump will show where the threads are stuck.

1) Linux : kill -3 < ps\_id >

2) Windows (console mode) : crtl + break

3) Windows (service) : beasvc -dump -svcname:mydomain\_myserver

How should u look for in a Thread Dump?

Always have 7-10 thread dumps taken in an interval of 5 seconds each

A thread dump will give you an insight on what every thread in the JVM is currently doing.

While analyzing the thread dump you should be looking for stuck threads (threads which have not moved from a long time) – could be a resource (file / db) lock not allowing the thread to complete the task and free the monitor.

Look for thread lock (if two threads store references to different objects into the same reference value, the variable will subsequently contain a reference to one object or the other, not a reference to some other object or a corrupted reference value).

Look for recursive loops in application / server code traces, usually hampers server performance by utilizing CPU.

Analysis of a Thread Dump

The most useful tool in analyzing a server hang is a set of thread dumps. A thread dump provides information on what each of the threads is doing at a particular moment in time. A set of thread dumps (usually 3 or more taken 5 to 10 seconds apart) can help analyze the change or lack of change in each thread's state from one thread dump to another. The pattern of a problem can only be diagnosed this way. A hung server thread dump would typically show little change in thread states from the first to the last dump.

Threads can be in one of the following states:

Running or runnable thread A runnable state means that the threads could be running or are running at that instance in time.

Suspended thread Thread has been suspended by the JVM.

Waiting for Notification A thread waits for notification from another thread. before going back to runnable state

Thread stuck waiting on a condition variable Threads in a condition wait state can be thought of as waiting for an event to occur.

Thread stuck waiting on a monitor lock Monitors are used to manage access to code that should only be run by a single thread at a time

Blocked The thread waits for completion of blocking operation (waiting I/O resource, waiting the completion of another thread, waiting to acquire the lock of an object) before going back to runnable state.

How to find the heap memory of Managed Server?

/weblogic-Home/common/bin/startManagedWebLogic.sh

Here there is a parameter set in the script with memory details.

Check the XMX, XMS Parameters in startManagedWebLogic.sh

What is OutOfMemory in PermGen Space?

--------- OutOfMemory in PermGen Space---------:

Permanent Generation is a Non-Heap Memory Area inside the JVM Space. Manytimes we see OutOfMemory in this Area. PermGen Area is NOT present in JRockit JVMs.

The PermGen Area is measured independently from the other generations because this is the place where the JVM allocates Classes, Class Structures, Methods and Reflection Objects. PermGen is a Non-Heap Area.It means we DO NOT count the PermGen Area as part of Java Heap.

The OutOfMemory in PermGen Area can be seen because of the following main reasons:

Point-1) Deploying and Redeploying a very Large Application which has many Classes inside it.

Point-2) If an Application is getting deployed/Updated/redeployed repeatedly using the Auto Deployment feature of the Containers. In that case the Classes belonging to the application stays un cleaned and remains in the PermGen Area without Class Garbage Collection.

Point-3) If ”-noclassgc” Java Option is added while starting the Server. In that case the Classes instances which are not required will not be Garbage collected.

Point-4) Very Less Space for allocated the “=XX:MaxPermGen”

---------What to do in case of OutOfMemory In PermGen---------:

Point-1) Make Sure that the PermGen Area is not set to a very less value.

Point-2) Usually if an Application has Many JSP Pages in that case every JSP will be converted to a \*.class file before JSP Request Process. So a large number of JSPs causes generation of a Large number of \*.class files all these classes gets loaded in the PermGen area.

Point-3) While allocating the -XX:MaxPermSize make sure that you follow a rough Formula… which works in most of the Application Servers.

MaxPermSize = (Xmx/3) —- Very Special Cases (One Third of maximum Heap Size)

MaxPermSize = (Xmx/4) —- Recommended (One Fourth Of maximum Heap Size)

Point-4) If you are repeatedly getting the OutOfMemory in PermGen space then it could be a Classloader leak….

May be some of the classes are not being unloaded from the permgen area of JVM . So please try to increase the -XX:MaxPermSize=512M or little more and see if it goes away.

If not then add the following JAVA\_OPTIONS to trace the classloading and unloading to find out the root cause :

-XX:+TraceClassloading and -XX:+TraceClassUnloading.

What is OutOfMemory in Java Heap?

An OutOfMemory is a condition in which there is not enough space left for allocating required space for the new objects or libraries or native codes.

---------OutOfMemory in Java Heap-----------:

This happens when the JVM is not able to allocate the required memory space for a Java Object. There may be many reasons behind this…like

Point-1) Very Less Heap Size allocation. Means setting the MaxHeapSize (-Xmx) parameter to a very less value.

Point-2) The Leaking of Objects. Either the Application is not unreferencing the unused Objects or the Third part frameworks (Hibernate/Spring/Seam…etc) might not be releasing the references of the objects due to some inaccurate configurations.

Point-3) In Many cases it may be the reason that Application codes are getting the JDBC connections objects from the DataSource are not being released back to the Connection Pool.

Point-4) Garbage Collection strategy may be in correct according to the environmental/application requirements.

Point-5) In-accurate setting of Application/Frameworks Cache.

-----------What to do in case of OutOfMemory In JavaHeap-----------:

Whenever we see an OutOfMemory in the server log or in the stdout of the server. We must try to do the following things as first aid steps:

Point-1) If possible enable the following JAVA\_OPTIONS in the server start Scripts to get the informations of the Garbage Collection status.

-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails -Xloggc:/opt/app/GCLogsDirectory/gc.log

Point-2) It is always needed to see what all objects were present when the OutOfMemory error occured to identify whether those objects belongs to the Application Code/ Application Framework Codes/ The Application Server APIs. Sothat we can isolate the issue. In order to get the details of the Heap Objects collect “HeapDump” either using JHat (not a better tool) or JMap (Much Better compared to the Jhat tool).

Point-3) Once we collected the Heap Dump we can easily monitor the Heap Details using best GUI toold like “Jhat Web Browser” or using “Eclipse Memory Analyzer”.

What is Native OutOfMemory in Java Heap?

---------Native OutOfMemory---------:

Native OutOfMemory is a scenario when the JVM is not able to allocate the required Native Libraries and JNI Codes in the memory.

Native Memory is an area which is usually used by the JVM for it’s internal operations and to execute the JNI codes. The JVM Uses Native Memory for Code Optimization and for loading the classes and libraries along with the intermediate code generation.

The Size of the Native Memory depends on the Architecture of the Operating System and the amount of memory which is already commited to the Java Heap. Native memory is an Process Area where the JNI codes gets loaded or JVM Libraries gets loaded or the native Performance packs and the Proxy Modules gets loaded…

Native OutOfMemory can happen due to the following main reasons:

Point-1) Setting very small StackSize (-Xss). StackSize is a memory area which is allocated to individual threads where they can place their thread local objects/variables.

Point-2) Usually it may be seen because of Tuxedos incorrect setting. WebLogic Tuxedo Connectors allows the interoperability between the Java Applications deployed on WebLogic Server and the Native Services deployed on Tuxedo Servers. Because Tuxedos uses JNI code intensively.

Point-3) Less RAM or Swap Space.

Point-4) Usually it may occur is our Application is using a very large number of JSPs in our application. The JSPs need to be converted into the Java Code and then need to be compiled. Which reqires DTD and Custom Tag Library resolution as well. Which usually consumes more native memory.

---------What to do in case of Native OutOfMemory---------:

Point-1) Usually Native OutOfMemory causes Server/JVM Crash. So it is always recommended to apply the following JAVA\_OPTIONS flags in the Server Start Script to instruct the JVM to generate the HeapDump ”-XX:+HeapDumpOnOutOfMemoryError“

By default the heap dump is created in a file called java\_pidpid.hprof in the working directory of the VM, as in the example above. You can specify an alternative file name or directory with the “-XX:HeapDumpPath=C:/someLocation/“

Note: Above Flags are also suitable to collect HeapDump in case of JavaHeap OutOfMemory as well. But these flags never gurantees that the JVM will always generate the Heap Dump in case of any OutOfMemory Situation.

Point-2) Usually in case of Native OutOfMemory a “hs\_err\_pid.log” file is created in case of Sun JDK and “xxxx.dump” file is created in case of JRockit JDK. These log files are usually Text Files and tells about the Libraries which caused the Crash. These files need to be collected and analyzed to find out the root cause.

Point-3) Make Sure that the -XX:MaxHeapSize is not set to a Very Large Space…because it will cause a very less Native Space allocation. Because as soon as we increase the HeapSize, the Native Area decreases.

Please see the Post:http://middlewaremagic.com/weblogic/?p=4456

Point-4) Keep Monitoring the process’s memory using the Unix utility ‘ps’ like following:

ps -p -o vsz

Here you need to pass the WebLogic Server’s PID (Process ID) to get it’s Threading Details with respect to the Virtual Memory Space.

Point-5) If the Heap Usages is less Or if you see that Your Application usages less Heap Memory then it is always better to reduls the MaxHeapSize so that the Native Area will automatically gets increased.

Point-6) Sometimes the JVMs code optimization causes Native OutOfMemory or the Crash…So in this case we can disable the Code Optimization feature of JVM.

(Note: disabling the Code Optimization of JVM will decrease the Performance of JVM)

For JRockit JVM Code Optimization can be disabled using JAVA\_OPTION -Xnoopt

For Sun JDK Code Optimization can be disabled using JAVA\_OPTION –Xint

Heap Dump

jmap command is used to generate heap dump if your using Sun JDK

jrcmd command is used to generate heap dump if your using Jrocket.

Don't generate heap dump multiple times as it causes your application performance degradation as generating heap dump will take at least 15 to 30 minutes and generated file size will be minimum 2 GB. And use Eclipse mat only to analyze it. IBM heap analyzer will not help you in this case due to IBM heap analyzer supports hprof formated files only to my knowledge

Thanks to Ramesh Kumar

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Heap dump - Collection of objects that are in memory (JVM)

Thread dump - Shows what each thread in a process is doing at a given point in time along with the stack trace.

Core dump - O/S level dump file which has O/S level info in addition to the heap dump.

Heap dump - is useful to analyse OOM situations.

Thread dump - To troubleshoot slow running of your application.

Core dump - When your JVM has crashed abruptly. To find details about native calls and so on.

jmap -heap:live,format=b,file=filename pid

jrcmd pid hprofdump filename=name\_of\_dump\_file

========================================================================

A thread dump is a dump of the stacks of all live threads. Thus useful for analyzing what an app is up to at some point in time, and if done at intervals handy in diagnosing some kinds of 'execution' problems (e.g. thread deadlock).

A heap dump is a dump of the state of the Java heap memory. Thus useful for analyzing what use of memory an app is making at some point in time so handy in diagnosing some memory issues, and if done at intervals handy in diagnosing memory leaks.

===================================================================

Heapdump can be taken in number of ways:

Via Java VM parameters:

-XX:+HeapDumpOnOutOfMemoryError writes heap dump on OutOfMemoryError (recommended)

-XX:+HeapDumpOnCtrlBreak writes heap dump together with thread dump on CTRL+BREAK

Using JRockit:

-jrcmd pid hprofdump filename=name\_of\_dump\_file

Using Jmap:

-jmap -heap:format=b pid

Note: use -J-d64 jvm option if your JVM is 64 Bit Jvm “jmap -J-d64 -heap pid”

You can also manually generate a heap dump with tool VisualVM.

Using HPROF:

You can use HPROF: Heap and CPU Profiling Agent.

A complete dump of the current live objects in the heap can be obtained with:

-java -agentlib:hprof=heap=dump,format=b -jar application

This will automatically dump heap when java application is terminated. You can also force heap dump by sending QUIT signal to java process with kill -QUIT pid command.

Analysing Heapdump file using Jhat

You can use jhat (Java Heap Analysis Tool) to read the generated file:

- jhat [ options ]

The jhat command parses a java heap dump file and launches a webserver. jhat enables you to browse heap dumps using your favorite webbrowser.

Note that you should have a hprof binary format output to be able to parse it with jhat. You can useformat=b option to generate the dump in this format.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Command to get Heap Dump on UNIX Platform:

Step1: Get the process ID of the server for which you are taking heap dump using below command

ps -ef | grep "server name"

Step2: Set PATH & CLASSPATH as below

export PATH=$PATH: <JDK bin path>

export CLASSPATH=$CLASSPATH: <WL-Home\server\lib>

Step3: Run below command to take Heap Dump

jmap -heap:format=b <PID>

It will generate a file named "heap.bin" in the path where you ran the command.

EX:

./jmap -dump:format=b,file=heapJMap.bin 5549

./jmap -heap:format=b 7611

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

what is default location of thread dump,heap dump,core dump in weblogic?

Srikanth Sri serverlogs: domainhome/servers/mngserv/logs

Maheshkumar Subbaiyan 1. Thread Dump - It will be written on servername.out file.

2. HeapDump - It will be written on basically from the weblogic server started,if you have used the startWeblogic.sh from $DOMAIN\_HOME,then it will be written here.

3. Core Dump/Crash file - It will be written on basically from the weblogic server started,if you have used the startWeblogic.sh from $DOMAIN\_HOME,then it will be written here.

How to match the JVM thread id with Linux Process ID

THREAD: A process is divided into multiple lite weight process. Each lite weight process is called thread.

Thread dump provide a snap shot of current active threads. It provide stack trace of all java threads in jvm.

When do we take thread dumps:

There could be scenarios like

1.server hang

2.crash

3. Performance delays.

4. In case of stuck threads.

Different ways to take thread dumps:

Method 1 : unix command

kill -3 pid

Method 2: java command

cd /home/oracle/Middleware/Oracle\_Home/user\_projects/domains/pavan\_domain/bin

[oracle@OEL6 bin]$ . ./setDomainEnv.sh

[oracle@OEL6 bin]$/home/oracle/jdk1.7.0\_21/bin/java weblogic.Admin -url t3://localhost:9001 -username weblogic -password weblogic123 THREAD\_DUMP

Method 3: Console

Click on MS1 -> Monitoring -> Performance -> Dump Thread Stacks

Method 4: wlst

[oracle@OEL6 bin]$ . ./setDomainEnv.sh

[oracle@OEL6 pavan\_domain]$ java weblogic.WLST

wls:/offline> connect('weblogic','weblogic','t3://localhost:9001')

wls:/pavan\_domain/serverConfig> cd('Servers')

wls:/pavan\_domain/serverConfig/Servers> ls()

dr-- AdminServer

dr-- ms1

dr-- ms2

wls:/pavan\_domain/serverConfig/Servers> cd('ms1')

wls:/pavan\_domain/serverConfig/Servers/ms1> threadDump()

How to match JVM Thread ID with Linux Process id

[oracle@OEL6 pavan\_domain]$ ps -ef | grep -i Dweblogic.Name=ms1

oracle 8296 8244 0 02:20 pts/0 00:00:45 /home/oracle/jdk1.7.0\_21/bin/java -server -Xms256m -Xmx256m -XX:MaxPermSize=256m -Dweblogic.Name=ms1 -Djava.security.policy=/home/oracle/Middleware/Oracle\_Home/wlserver/server/lib/weblogic.policy -Dweblogic.ProductionModeEnabled=true -Dweblogic.security.SSL.trustedCAKeyStore=/home/oracle/Middleware/Oracle\_Home/wlserver/server/lib/cacerts -Xverify:none -Djava.endorsed.dirs=/home/oracle/jdk1.7.0\_21/jre/lib/endorsed:/home/oracle/Middleware/Oracle\_Home/wlserver/../oracle\_common/modules/endorsed -da -Dwls.home=/home/oracle/Middleware/Oracle\_Home/wlserver/server -Dweblogic.home=/home/oracle/Middleware/Oracle\_Home/wlserver/server -Dweblogic.management.server=http://localhost:9001 -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:/home/oracle/Middleware/Oracle\_Home/user\_projects/domains/pavan\_domain/gc.log weblogic.Server

[oracle@OEL6 pavan\_domain]$ top -b -H -p 8296

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

8298 oracle 20 0 1284m 224m 1112 S 2.0 22.7 0:03.61 java

8313 oracle 20 0 1284m 224m 1112 S 2.0 22.7 0:00.88 java

8296 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8297 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:04.40 java

8299 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.01 java

8300 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.03 java

8301 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8302 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:03.01 java

8303 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:03.25 java

8304 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8305 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:03.93 java

8306 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8309 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.66 java

8310 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:10.81 java

8311 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8312 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:02.77 java

8314 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:08.24 java

8315 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.12 java

8316 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.32 java

8343 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8345 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.02 java

8349 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:01.90 java

8381 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.94 java

8383 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.04 java

8386 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8387 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.04 java

8388 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8389 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.05 java

8622 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

8623 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java

Stuck Thread from thread dump:

"[STUCK] ExecuteThread: '3' for queue: 'weblogic.kernel.Default (self-tuning)'" daemon prio=10 tid=0x00007fe980d03000 nid=0x20c4 in Object.wait() [0x00007fe98d86a000]

java.lang.Thread.State: WAITING (on object monitor)

at java.lang.Object.wait(Native Method)

- waiting on <0x00000000e71abac8> (a weblogic.work.ExecuteThread)

at java.lang.Object.wait(Object.java:503)

at weblogic.work.ExecuteThread.waitForRequest(ExecuteThread.java:238)

- locked <0x00000000e71abac8> (a weblogic.work.ExecuteThread)

at weblogic.work.ExecuteThread.run(ExecuteThread.java:264)

convert nid 0x20c4 to decimal number 8388 and compare against top -b -H -p 8296 out put

8388 oracle 20 0 1284m 224m 1112 S 0.0 22.7 0:00.00 java